Towards Learned Metadata Extraction for Data Lakes

Sven Langenecker

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Towards Learned Metadata Extraction for Data Lakes

Doctoral thesis by
Sven Langenecker, M.Sc.

submitted in fulfillment of the requirements for the degree of *Doktor-Ingenieur (Dr.-Ing.)*

Reviewers
Prof. Dr. rer. nat. Carsten Binnig
Prof. Paolo Papotti, Ph.D.

Department of Computer Science
Technical University of Darmstadt

Darmstadt, 2024

Sven Langenecker: *Towards Learned Metadata Extraction for Data Lakes*
Darmstadt, Technical University of Darmstadt
Year thesis published in TUprints: 2024
URN: urn:nbn:de:tuda-tuprints-274697
URL: https://tuprints.ulb.tu-darmstadt.de/27469
Date of the viva voce: 04.06.2024

Erklärung laut Promotionsordnung

§8 Abs. 1 lit. c PromO

Ich versichere hiermit, dass die elektronische Version meiner Dissertation mit der schriftlichen Version übereinstimmt.

§8 Abs. 1 lit. d PromO

Ich versichere hiermit, dass zu einem vorherigen Zeitpunkt noch keine Promotion versucht wurde. In diesem Fall sind nähere Angaben über Zeitpunkt, Hochschule, Dissertationsthema und Ergebnis dieses Versuchs mitzuteilen.

§9 Abs. 1 PromO

Ich versichere hiermit, dass die vorliegende Dissertation selbstständig und nur unter Verwendung der angegebenen Quellen verfasst wurde.

§9 Abs. 2 PromO

Die Arbeit hat bisher noch nicht zu Prüfungszwecken gedient.

Darmstadt, April 23, 2023

Sven Langenecker

Abstract

In the landscape of data-driven enterprises, the concept of data lakes serves for storing and managing massive volumes of diverse data. Unlike traditional data warehousing methods characterized by rigid structures and predefined schemas, data lakes present a paradigm shift by embracing a more fluid architecture. Here, data arrives in its raw, unaltered form, preserving its inherent complexity and richness. The lack of predefined structures or standardized schemas makes it difficult to identify, find, understand, and use the relevant data sets contained in these repositories. To address this data discovery problem and enable an easy navigation, solutions for automatic metadata extraction are essential. Hence, a variety of Machine Learning (ML) based approaches for automated extracting of semantic types from table columns have recently been proposed. While initial results of these learned approaches seem promising, it is still not clear how well these approaches can generalize to new unseen data in real-world enterprise data lakes.

This dissertation thus focuses on the challenge of making the task of semantic type extraction of table columns feasible for real-world enterprise data lakes. First, we studied existing approaches for semantic type extraction of table columns and evaluated how applicable they are in data lake environments to understand their limitations. Based on the findings that existing approaches are not usable out-of-the-box and always need to be adapted to the data lake where they are intended to be used, we advocate a weak supervision concept to adapt these learned semantic type detection models to the specific data lake with minimal effort. Thus, as a first contribution of this dissertation, we present a new data programming framework for semantic labeling based on the idea of weak supervision. Our new data programming framework comes with pre-designed Labeling Functions (LFs) to generate new training data that covers the new semantic types and data characteristics of the unseen data lake to which the learned semantic type extraction model is supposed to be applied. With the generated training data of our framework, the model can be re-trained/fine-tuned with minimal effort to achieve an adaption to the respective data lake and with this eliminate the barrier to apply recently learned semantic type detection approaches on enterprise data lakes.

Furthermore, because the semantic labeling of numerical data is more challenging than of textual data, we present as a second contribution our novel training data generation procedure called *Steered-Labeling*. *Steered-Labeling* is integrated as a core component in our data programming framework and enables to generate high quality training data for textual and numerical table columns. The basic idea of the new procedure is to separate the labeling process into two sequential steps. In the first step, the framework

labels the non-numerical columns, that are easier to label. Afterward, in the second step, the numerical columns are labeled by including the previously generated labels of the non-numeric columns, which serve as additional information. With this, the LFs achieves a much higher accuracy for numerical columns. We show by an extensive evaluation that our data programming framework with the *Steered-Labeling* procedure can adapt learned models to unseen data lakes with the automatically generated training data.

During the experiments with our framework, we observed that the re-trained/fine-tuned end models performed worse on numeric columns than on non-numeric columns, even though the generated training data of the numerical columns is quite adequate. This is mainly because the existing models were designed, trained, and tested with datasets composed mainly of non-numerical data and therefore optimized to handle these data types. Although we used two data lakes that contain numerical columns in the evaluation of our *Steered-Labeling* procedure, these datasets could not be used for the design of a new model that better supports numerical columns because they are too small for this purpose. Thus, as a third contribution, we create and provide a new corpus for the task of semantic type detection of table columns called SportsTables. By scraping tables from various web pages of different sports domains, our corpus comprises tables that contain a much higher proportion of numerical columns than those in existing corpora. Furthermore, they are much larger both in the number of columns and rows. Hence, our new corpus reflects the characteristics of real-world data lakes and poses new challenges to semantic type detection models. We show through an evaluation of several recent semantic type detection models on our corpus, that they only perform robustly on textual data.

To tackle the shortcomings of the existing models, we finally propose a new semantic type detection approach called *Pythagoras*, designed to support numerical along with non-numerical columns. To achieve this, the main idea of the new model is to use Graph Neural Networks (GNNs) together with a new graph representation of tables and their columns. This graph representation includes directed edges to aggregate necessary context information (e.g. table name, neighboring non-numerical column values) for predicting the correct semantic type of numerical columns using the GNN message passing mechanism. Thus, the model learns which contextual information is relevant for determining the semantic type. With this approach, our model can outperform all existing semantic type detection models on numerical table columns.

Zusammenfassung

In der Systemlandschaft datengesteuerter Unternehmen dient das Konzept Data Lakes der Speicherung und Verwaltung großer Mengen unterschiedlicher Daten. Im Gegensatz zu herkömmlichen Data-Warehousing-Methoden, die durch starre Strukturen und vordefinierte Schemata gekennzeichnet sind, stellen Data Lakes einen Paradigmenwechsel dar, indem sie eine dynamischere Architektur aufweisen. Die Daten gelangen hier in ihrer rohen, unveränderten Form an und bewahren so ihre inhärente Komplexität und Reichhaltigkeit. Das Fehlen von vordefinierten Strukturen und standardisierten Schemata erschwert die Identifizierung, das Auffinden, das Verständnis und die Nutzung von relevanten Datensätzen, die in diesen Repositories enthalten sind. Um dieses Problem der Datenfindung zu lösen und eine einfache Navigation zu ermöglichen, sind Lösungen zur automatischen Extraktion von Metadaten unerlässlich. Aus diesem Grund wurde inzwischen eine Vielzahl von ML-basierten Ansätzen zur automatischen Extraktion semantischer Typen aus Tabellenspalten entworfen. Während erste Ergebnisse dieser gelernten Ansätze vielversprechend erscheinen, ist allerdings noch nicht klar, inwieweit sich diese Ansätze auf neue, ungesehene Daten in realen Data Lake Umgebungen generalisieren und anwenden lassen.

Diese Dissertation konzentriert sich daher auf die Herausforderung, die Aufgabe der semantischen Extraktion von Tabellenspalten für reale Enterprise Data Lakes realisierbar zu machen. Zunächst untersuchen wir bestehende Ansätze zur semantischen Extraktion von Tabellenspalten und bewerteten ihre Anwendbarkeit in Data Lake Umgebungen, um ihre Limitierungen zu verstehen. Basierend auf der Erkenntnis, dass bestehende Ansätze nicht out-of-the-box nutzbar sind und immer an den jeweiligen Data Lake angepasst werden müssen, in dem sie eingesetzt werden sollen, schlagen wir ein *weak supervision* Konzept vor, um diese erlernten Modelle zur semantischen Typenerkennung mit geringem Aufwand an den spezifischen Data Lake anzupassen. Als ersten Beitrag dieser Dissertation stellen wir daher einen neues Datenprogrammierungs-Framework zur semantischen Kennzeichnung vor, basierend auf der *weak supervision* Idee. Unser neues Datenprogrammierungs-Framework enthält vorgefertigte LFs, mit denen neue Trainingsdaten generiert werden können, welche die neuen semantischen Typen und Datencharakteristika des ungesehenen Datensees abdecken, auf den das gelernte semantische Typenextraktionsmodell angewendet werden soll. Mit den generierten Trainingsdaten unseres Frameworks kann das Modell mit minimalem Aufwand nachtrainiert/feinabgestimmt werden, um eine Anpassung an den jeweiligen Data Lake zu erreichen und damit

die Barriere gelernter semantische Typerkennungsansätze auf Enterprise Data Lakes anzuwenden zu beseitigen.

Da die semantische Kennzeichnung numerischer Daten eine größere Herausforderung darstellt als die von textuellen Daten, stellen wir als zweiten Beitrag unser neuartiges Verfahren zur Erzeugung von Trainingsdaten vor, welches die Bezeichnung *Steered-Labeling* trägt. *Steered-Labeling* ist als Kernkomponente in unser Datenprogrammierungs-Framework integriert und ermöglicht die Generierung hochwertiger Trainingsdaten für textuelle und numerische Tabellenspalten. Die Grundidee des neuen Verfahrens besteht darin, den Labelprozess in zwei aufeinander folgende Schritte zu unterteilen. Im ersten Schritt labelt das System die nicht numerischen Spalten, bei denen die Bestimmung des semantischen Typens einfacher ist. Anschließend werden im zweiten Schritt die numerischen Spalten unter Einbeziehung der zuvor generierten Labels der nicht-numerischen Spalten, die als zusätzliche Information dienen, gelabelt. Auf diese Weise erreichen die LFs eine wesentlich höhere Genauigkeit bei numerischen Spalten. In einer umfangreichen Evaluierung zeigen wir, dass unser Dataprogrammierungs-Framework die gelernten Modelle an den ungesehenen Data Lake mittels der automatisch generierten Trainingsdaten anpassen kann.

Bei den Experimenten mit unserem Framework haben wir festgestellt, dass die nachtrainierten/feinabgestimmten Endmodelle bei numerischen Spalten schlechter abschneiden als bei nicht-numerischen Spalten, obwohl die generierten Trainingsdaten der numerischen Spalten ausreichend adäquat sind. Dies liegt vor allem daran, dass die vorhandenen Modelle mit Datensätzen entwickelt, trainiert und getestet wurden, die hauptsächlich aus nicht-numerischen Daten bestehen, und daher für die Verarbeitung dieser Datentypen optimiert sind. Zwar haben wir bei der Evaluierung unseres *Steered-Labeling*-Verfahrens zwei Data Lakes verwendet, die numerische Spalten enthalten, doch können diese Datensätze nicht für den Entwurf eines neuen Modells mit besserer Unterstützung numerischer Spalten verwendet werden, da sie für diesen Zweck zu klein sind. Aus diesem Grund ist unser dritter Beitrag die Erstellung und Bereitstellung eines neuen Korpus für die Aufgabe der semantischen Typerkennung von Tabellenspalten mit dem Namen SportsTables. Durch das Abgreifen von Tabellen von verschiedenen Webseiten aus unterschiedlichen Sportarten umfasst unser Korpus Tabellen, die einen viel höheren Anteil an numerischen Spalten enthalten als Tabellen in bestehende Korpora und sowohl in der Anzahl der Spalten als auch der Zeilen viel größer sind. Daher spiegelt unser neuer Korpus die Eigenschaften realer Data Lakes besser wider und stellt somit semantische Typenerkennungsmodelle vor neue Herausforderungen. Wir zeigen anhand einer Evaluierung von mehreren aktuellen Modellen auf unserem Korpus, dass diese nur bei textuellen Daten robust funktionieren.

Um die Unzulänglichkeiten der bestehenden Modelle zu beheben, schlagen wir schließlich einen neuen Ansatz zur semantischen Typerkennung namens *Pythagoras* vor, der sowohl numerische als auch nicht-numerische Spalten unterstützt. Um dies zu erreichen, besteht die Hauptidee des neuen Modells darin, GNNs zusammen mit einer neuen graphischen Darstellung von Tabellen und ihren Spalten zu verwenden. Diese Graphdarstellung enthält gerichtete Kanten, um die notwendigen Kontextinformationen (z. B. Tabellenname, benachbarte nichtnumerische Spaltenwerte) für die Vorhersage des korrekten semantischen Typs numerischer Spalten unter Verwendung des GNN-Nachrichtenübermittlungsmechanismus bereitzustellen. Auf diese Weise lernt das Modell, welche Kontextinformationen für die Bestimmung des semantischen Typs relevant sind und unnser Modell kann mit diesem Ansatz alle bestehenden Modelle zur Erkennung semantischer Typen bei numerischen Tabellenspalten übertreffen.

Publications

The following peer-reviewed publications are part of this cumulative dissertation. Their content is printed in Part II, Chapters 7 to 12.

[1] Sven Langenecker, Christoph Sturm, Christian Schalles, and Carsten Binnig. "Towards Learned Metadata Extraction for Data Lakes." In: *Datenbanksysteme für Business, Technologie und Web (BTW 2021), 19. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme" (DBIS), 13.-17. September 2021, Dresden, Germany, Proceedings*. Ed. by Kai-Uwe Sattler, Melanie Herschel, and Wolfgang Lehner. Vol. P-311. LNI. Gesellschaft für Informatik, Bonn, 2021, pp. 325–336. DOI: 10.18420/BTW2021-17. URL: https://doi.org/10.18420/btw2021-17.

[2] Sven Langenecker, Christoph Sturm, Christian Schalles, and Carsten Binnig. "SportsTables: A new Corpus for Semantic Type Detection." In: *Datenbanksysteme für Business, Technologie und Web (BTW 2023), 20. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme" (DBIS), 06.-10, März 2023, Dresden, Germany, Proceedings*. Ed. by Birgitta König-Ries, Stefanie Scherzinger, Wolfgang Lehner, and Gottfried Vossen. Vol. P-331. LNI. Gesellschaft für Informatik e.V., 2023, pp. 995–1008. DOI: 10.18420/BTW2023-68. URL: https://doi.org/10.18420/BTW2023-68.

[3] Sven Langenecker, Christoph Sturm, Christian Schalles, and Carsten Binnig. "Steered Training Data Generation for Learned Semantic Type Detection." In: *Proc. ACM Manag. Data* 1.2 (2023), 201:1–201:25. DOI: 10.1145/3589786. URL: https://doi.org/10.1145/3589786.

[4] Sven Langenecker, Christoph Sturm, Christian Schalles, and Carsten Binnig. "SportsTables: A New Corpus for Semantic Type Detection (Extended Version)." In: *Datenbank-Spektrum* 23.2 (2023). DOI: 10.1007/s13222-023-00457-y. URL: https://doi.org/10.1007/s13222-023-00457-y.

[5] Sven Langenecker, Christoph Sturm, Christian Schalles, and Carsten Binnig. "Pythagoras: Semantic Type Detection of Numerical Data Using Graph Neural Networks (Short Paper)." In: *Lernen, Wissen, Daten, Analysen (LWDA) Conference Proceedings, Marburg, Germany, October 9-11, 2023*. Ed. by Michael Leyer and Johannes Wichmann. Vol. 3630. CEUR Workshop Proceedings. CEUR Workshop Proceedings, 2023, pp. 146–152. URL: https://ceur-ws.org/Vol-3630/LWDA2023-paper13.pdf.

[6] Sven Langenecker, Christoph Sturm, Christian Schalles, and Carsten Binnig. "Pythagoras: Semantic Type Detection of Numerical Data in Enterprise Data Lakes." In: *Proceedings 27th International Conference on Extending Database Technology, EDBT 2024, Paestum, Italy, March 25 - March 28*. Ed. by Letizia Tanca, Qiong Luo, Giuseppe Polese, Loredana Caruccio, Xavier Oriol, and Donatella Firmani. Vol. 27. OpenProceedings.org, 2024, pp. 725–733. DOI: 10.48786/EDBT.2024.62. URL: https://doi.org/10.48786/edbt.2024.62.

Due to the nature of the synopsis and for better readability, selected paragraphs from these publications were transferred verbatim throughout the synopsis without explicit labeling as suggested in the department regulations "Kumulative Dissertation und Eigenzitate in Dissertationen" (21.09.2021) §1.

Acknowledgments

First and foremost, I would like to express my deepest gratitude to Prof. Dr. Carsten Binnig for his immense support, guidance assistance, and supervision during my doctoral journey. Even though I was an external Ph.D. student for Him, I never felt like I was treated that way. His incredible mentorship has not only shaped this dissertation but has also enriched my growth as a researcher. He was an awesome, inspiring, motivating, and supportive mentor who helped me develop personally and academically over the years. Every conversation with him was very inspiring, constructive as well as motivating and helped me a lot to achieve the goals in my research work. I'm really glad that I've had him as my advisor all these years.

The same can be said for my supervisors Prof. Dr. Christoph Sturm and Prof. Dr. Christian Schalles at DHBW Mosbach. Their support and mentorship throughout my path was incredibly valuable. In the weekly meetings, they always brought up useful ideas, answered many of my questions in different subject areas, or discussed with me various new approaches, which made a significant contribution to this dissertation. I couldn't have imagined a better team to guide me than Carsten, Christoph, and Christian.

I also sincerely thank Prof. Paolo Papotti for his valuable time and effort in reviewing this dissertation.

I would also like to thank my doctoral student colleagues at the TU Darmstadt. Their support and especially their feedback on my work were always very constructive and productive. In hindsight, I have to say that I would have loved to spend more time with them. However, as I was an external Ph.D. student and did my doctorate alongside my regular job, this was unfortunately not possible.

I am deeply indebted to my wife Lina, her support and understanding have been the cornerstone of my academic pursuit. Her patience, encouragement, and sacrifice created an ideal environment and gave me the time and space I needed to entirely dive into my research. Thanks to her ability to manage our home and all family-related aspects, I was able to do my academic work and achieve my goals. On the way to my doctorate, I couldn't imagine a more perfect wife. I am also grateful to my two sons, Maik and Emil, for their understanding and patience during this time. Their love and understanding were a constant motivation and source of joy during this journey.

I am deeply grateful to my parents, sister, and friends for their constant support and encouragement during my life. Finally, to all the people who have directly or indirectly contributed to my academic and personal development, I am very grateful.

Contents

Acronyms

CRF Conditional Random Field

DNN Deep Neural Network

GCN Graph Convolutional Network

GNN Graph Neural Network

LDA Latent Dirichlet Allocation

LF Labeling Function

LLM Large Language Model

LM Language Model

ML Machine Learning

RAG Retrieval Augmented Generation

Part I

Synopsis

1 Introduction

Data lakes are important. Nowadays, data lakes are widely being used in organizations to manage their data [26, 51]. They allow to accumulate vast amounts of raw data in its native format, pouring in from various sources with different domains, be it business operations, IoT devices, social media or customer interactions. Unlike traditional data warehouses, which require structuring data before storage, data lakes permit the retention of diverse data types without predefined schemas [24, 89] (see also Figure 1.1). This reservoir-like architecture of data lakes enables companies to collect and store extensive datasets from different sources in a very fast way without the overhead of transforming data before storage [30, 36]. As a result, their inherent scalability facilitates the storage of huge amounts of information and helps to gain valuable insights through advanced analytics and machine learning.

Data discovery in data lakes is a problem. However, effective governance and metadata management are crucial to realizing the full potential of data lakes, ensuring data quality, security, and accessibility. Considering the mass of raw data stored in data lakes, the main challenge is to have solutions that allow you to effectively navigate through the data lake to find relevant and valuable information (known as *data discovery problem*). For example, if a data scientist wants to analyze climate change, they need weather data from different locations over a long period of time. Finding such data efficiently in a data lake without a lot of manual effort is very hard. To address this data discovery problem, it is suggested to semantically annotate the stored data in order to make it easier to identify and locate. As such, in the last years various approaches for automated semantic type annotation using ML have been proposed. However, these approaches can not be used out-of-the-box on unseen data lakes and require a time-consuming adaption, which is a significant barrier to their applicability in enterprise data lakes.

Learned semantic typing to the rescue. In the following, we will first explore how metadata, particularly semantic type annotations on table columns, can help to solve the data discovery problem in data lakes. Afterward, we discuss how learned semantic type detection approaches can be leveraged to address the challenge of automatically detecting semantic types of data. Furthermore, we describe the fundamental limitations

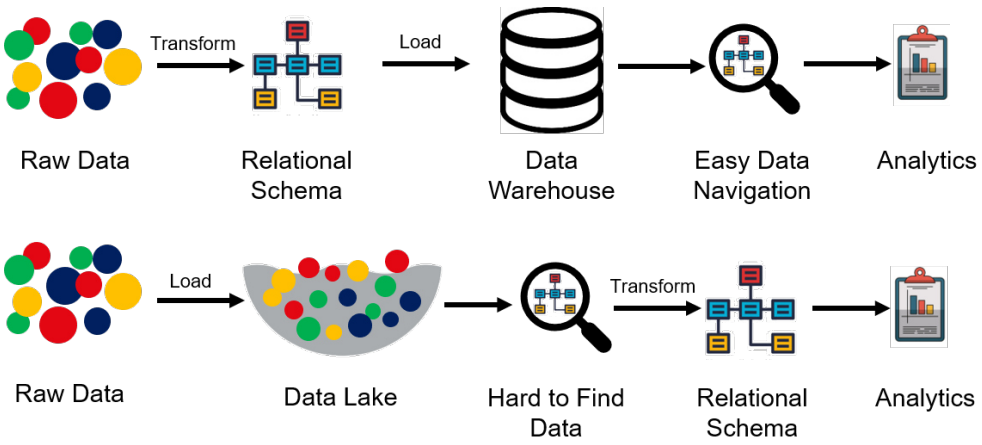


Figure 1.1: Shows the comparison of a data warehouse approach (upper part) to a data lake approach (lower part). In a data warehouse, the raw data is first transformed into a defined relational schema and then stored. The predefined relational schema and the clear structure of stored data make it easy to navigate through the data. Instead, in a data lake approach, raw data is loaded in its native format without any transformations. Due to the lack of schema and structure in the data, it is more difficult to find relevant data for a downstream analytic task.

of the recently proposed learned semantic type extraction approaches and introduce the concrete contributions of this dissertation.

1.1 The Need for Metadata in Data Lakes

Data catalogs are used for data lakes. In contrast to classical data warehouses, the idea of data lakes is that the data does not need to be organized and cleaned in advance when it is loaded into the warehouse [24]. Instead, data lakes follow a more ”lazy” approach that allows enterprises to store any available data in its raw form. This raw data is organized and cleaned once it is needed for a down stream task such as data mining or building ML models. However, due to the sheer size of data stored in data lakes, the variety of connected data sources and the absence of a comprehensive schema, data discovery in data lakes has become an important problem [72, 77, 94]. To address the data discovery problem and to improve the usability of data lakes, data catalogs are typically used. The need for such a data catalog is evident by the growing number of products available from different vendors such as Azure Purview [73], AWS

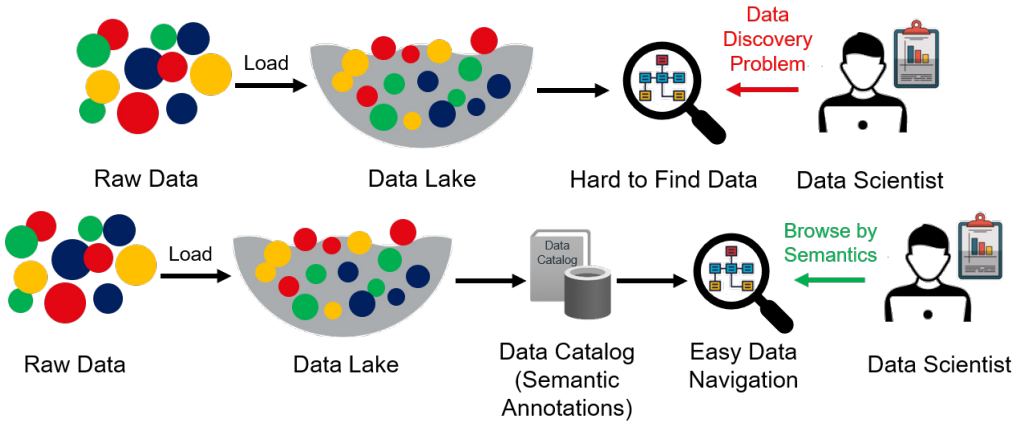


Figure 1.2: The figure shows the different data lake approaches with and without a data catalog. The upper approach does not include a data catalog, which makes it very hard for data scientists to find relevant data (data discovery problem). Instead, the lower approach utilizes a data catalog that contains semantic annotations of the stored data. These annotations enable data scientists to search and identify relevant data sets by using semantic descriptions or semantic types.

Glue Catalog [4], Google Cloud Data Catalog [33], Alation [3], Collibra [17], Atlan [5] and Dremio [25].

Data discovery by using semantic annotations. An important function of such data catalogs is to annotate semantic type information on columns of table-like data (e.g., CSV files) according to an ontology or another taxonomy used in an enterprise. As illustrated in Figure 1.2, these annotations will allow users to browse and identify relevant datasets in data lakes by using semantic descriptions or semantic classes (e.g., *country*, *city*, *temperature* etc. defined in an ontology). As an example, for a data lake of a news magazine, semantic types such as *sports.teamname* or *sport.event* are important information that allows a data journalist to identify which relevant sources she requires for preparing a news article. Furthermore, by annotating data with semantic types, users can quickly grasp the nature of the information, its intended use, and its relationships with other datasets. This understanding is essential for users trying to navigate and utilize the extensive data stored in data lakes.

1.2 Towards Learned Metadata Extraction

Managing a data catalog manually is hard. Data catalogs storing metadata like semantic annotations or descriptions help to solve the data discovery problem in data

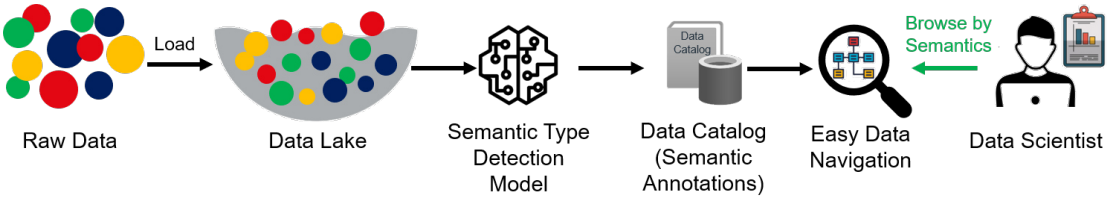


Figure 1.3: Automatic semantic type annotation using an ML model that can detect semantic types of data and thus can fill the data catalog automatically.

lakes. However, these semantic types of data must first be extracted in order to fill the data catalog and establish the assignment of data to semantic types to the data sources (i.e., per column or per table). Manually creating and upholding such a data catalog containing semantic annotations is sheer impossible due to the large amount of data and the prevalent dynamics in data lakes. In addition, making semantic annotations by hand is a daunting task and can usually only be done by domain experts who are familiar with the domain from which the data originates.

Learned approaches for automatic semantic annotation. As such, in the last years various approaches for automated semantic type extraction on table columns have been proposed. Whereas existing commercial products mainly rely on simple search based solutions such as regular expressions and dictionary look ups (e.g. [73, 74]), more recent approaches use ML or more precisely Deep Neural Networks (DNNs)[20, 50, 100, 113]. The basic idea of these approaches is to create an ML model that learns the mapping of data patterns (i.e., values in table columns) to semantic types. Afterward, the learned model is subsequently used to extract the semantics of data in the data lake and automatically fill the data catalog with semantic annotations (see Figure 1.3). This not only eliminates the effort of manually annotating data sources in data lakes but also often leads to better annotation accuracies. Furthermore, data can be semantically annotated very quickly. Thus, the catalog can be continuously updated, even if the data lake is highly dynamic and new data is added at a very high rate. We will now first describe the advantages of learned semantic type extraction approaches before we introduce recent proposed models, their architecture, and the main paradigm and methods they use.

Advantages of learned approaches. ML-based solutions for semantic type extraction in data lakes offer several advantages over traditional manual semantic annotations and search-based methods like regular expressions. Firstly, ML-based approaches enable scalability and adaptability to diverse and evolving datasets without the need for constant manual intervention. These algorithms can autonomously learn from large volumes of data, capturing complex patterns and variations that might be challenging for human

annotation or rule-based systems to grasp. Additionally, ML models can enhance accuracy by iteratively refining their understanding of semantic types through continuous learning from new data, providing more nuanced and precise annotations compared to static rule-based approaches. Furthermore, ML-driven semantic type extraction fosters a more flexible and dynamic data discovery process, enabling a more efficient search experience within data lakes. This adaptability and precision contribute significantly to enhancing the usability and effectiveness of data lakes, addressing the data discovery problem by facilitating more intuitive and comprehensive access to stored information.

Traditional vs. pre-training/fine-tuning paradigm. Existing learned semantic type detection models of table columns essentially use two different ML learning paradigms. On the one hand there are classical supervised approaches such as Sherlock or Sato [50, 113] and on the other hand there are models that rely on the pre-training/fine-tuning paradigm such as Turl or Doduo [20, 100]. Whereas classical supervised models are trained from scratch using labeled datasets specific to the task at hand, pre-training/fine-tuning models are first pre-trained with large and diverse datasets often using self-supervised or semi-supervised methods to learn a general representation of the data (i.e., tables in this use case). Subsequently, these pre-trained models undergo fine-tuning for the target task (i.e., semantic type detection) achieved by further training on a task-specific labeled dataset. This paradigm aims to minimize the required amount of labeled task-specific training data because the model has already learned how to handle the data in general during the pre-training phase. For both learning paradigms, used to build a semantic type detection model, datasets containing tables with semantically annotated columns are needed. Therefore, in recent research works, various datasets are generated for the task of semantic type detection, containing many examples of columns labeled with semantic types [1, 9, 18, 19, 40–42, 45–47, 82, 96]. These existing table corpora primarily exist of tables extracted from the web, incorporating either only or a very high percentage of textual (non-numeric) data.

Columnwise vs. tablewise models. All existing solutions for semantic type detection use these datasets to train, validate and test the models. Thereby, these approaches can be distinguished into columnwise models and tablewise models. Columnwise models exclusively leverage values from a single column, omitting the inclusion of contextual information from the table. Sherlock [50] is such a columnwise model which extracts multiple features from individual columns such as character distributions, word embeddings, text embeddings and column statistics. These features are then processed through a combination of multi-layer subnetworks and a primary network, which comprises two fully connected layers. Dosolo [100] is a columnwise model that uses the pre-trained

BERT Language Model (LM) combined with an attached output layer to implement a semantic type detection model. Given that BERT receives token sequences (i.e. text) as input, they convert a column into such a sequence. When serializing the columns, the individual column values are first converted into a string and then concatenated to a sequence.

Unlike columnwise models, tablewise models process the entire table with its columns and their values at once. The advantages of this approach lie in its ability to utilize contextual information from the table and thus enhancing the precision of semantic type prediction for individual table columns. Building upon Sherlock, Sato is a tablewise model that incorporates Latent Dirichlet Allocation (LDA) features to capture table context and integrates a Conditional Random Field (CRF) layer to learn column type dependencies. With this, Sato's prediction quality improves over Sherlock. Dosolo and Doduo are both models from [100]. In contrast to Dosolo, Doduo is a tablewise model designed to process an entire table as input to the BERT model. To achieve this, all columns and their values are concatenated one after the other to form one input sequence. The major difference between the two approaches and their serialization techniques is that with Dosolo a column type is predicted independently of other data, whereas Doduo captures the data of neighboring columns to make a semantic type prediction of an individual column.

Characteristics of all existing models. What all of these existing approaches have in common is that they are always trained on data with characteristics and semantic types out of one dataset. Finally, this leads to a specific trained model that is exclusively intended for use on a single data lake [48]. This can either be the data lake from which the training data originates or a data lake that contains very similar characteristics and semantics to the training data. Consequently, the existing models can not be used in a zero-shot manner out-of-the-box across different data lakes. Another predominant characteristic of existing approaches is that they mainly focus on the detection of semantic types of table columns containing textual data. Recent models leverage LMs (e.g., BERT) [100] or a Large Language Model (LLM) (e.g., GPT) [55] as a foundation to develop semantic type detection solutions. Since these models are specifically tailored to textual data, the resulting semantic type detection model for table columns is also specialized towards textual data. To summarize, the predominant characteristics of existing models are the following two: (1) *data lake specific* (i.e., not generalize across different data lakes), (2) *textual data focused*.

1.3 Limitations of Existing Learned Approaches

While existing learned semantic type extraction approaches have shown success and the results are promising when applied to data with characteristics and semantics that are covered by the training data used to build the model, they cause a lot of effort when customizing them to a different data lake with different data characteristics and semantics. We will next provide some more details on the fundamental limitations of state-of-the-art approaches.

First, existing semantic type detection approaches are always designed to be used on a single data lake, for which it has been trained with training data that reflects the data characteristics and semantics of the respective data lake. If the training data set does not cover the broad spectrum of data characteristics and semantic types, the performance of the learned models quickly degrades when applied to a new data lake [62]. Thus, these models do not generalize to unseen data (i.e., that was not included in the training set) and must be adapted each time for deployment on a different data lake. For instance, the DNN approach of [113] is trained and tested with the WebTables corpus from VizNet [45] comprising a total of about 80K tables (resulting in a total of about 120K pairs of columns with their associated semantic types) and 78 different semantic types of table columns. Applying this trained model on another data lake like the PublicBI Benchmark [31] results that the model is only able to extract semantic types for about 10% of the columns in the corpus. In addition, for this 10% of columns that are in principle covered by the training data, the model can not infer the types in an robust manner, only achieving a support weighted F1-Score of about 0.3, which is a major drop in comparison to the original reported performance on the test data set of the VizNet corpus (F1-Score of 0.925). For adapting the trained model to the new data lake, a re-training of the model is needed with new training data which covers the characteristics and semantic types of the new data lake. Ideally, labeled data from the new data lake would be used as training data, as this ensures that the spectrum of data diversity and semantics is covered. However, this labeled data is not inherently available and must first be generated at a high cost. This high cost and effort can easily be a barrier to the use of a learned semantic type extraction approach on the data lake. Even for pre-trained/fine-tuning approaches, where the assumption is that less new labeled data is needed to achieve an adaptation, this limitation applies [61].

Secondly, it is necessary to adapt the learned model by re-training every time new data sources are added to the data lake, which involves new data characteristics and semantics. Accordingly, the high costs of an adaptation are incurred each time the data

lake is updated or expanded with new data sources. For instance, if a model learned to extract 78 different semantic types (as it is the case with Sherlock and Sato [50, 113]) and a new data source with an additional semantic type unknown to the learned model is added to the data lake, the model will become unusable for the new data. The only way to update the underlying model is to re-train it using new training samples (i.e., pairs of columns and their semantic types) that reflect the new data from the new data source. Therefore, new training samples have to be generated, which again causes high costs.

Third, all existing state-of-the-art semantic type extraction approaches focus primarily on detecting the semantic type of non-numerical (textual) data [58, 60]. Using the models on columns containing numerical data results in a significant drop in prediction accuracy compared to using them on textual columns. Thus, as soon as the models are applied to a data lake with a larger proportion of numerical data, which is often the case in real-world enterprise data lakes [60], they will fall short and can not be utilized effectively. The reasons why existing models are not designed for numerical data is mainly because of the fact that corpora that were used to train and validate these models primarily contain non-numerical data [1, 9, 18, 19, 40–42, 45–47, 82, 96]. Therefore, the models were developed to handle mainly this data type.

Overall, existing semantic type extraction models are *data lake specific*, and must always be adapted to the respective data lake by re-training. However, the immense cost of generating new labeled data to re-train the model is unacceptable in many cases and therefore impractical for being applied in new data lake environments. This aspect makes existing models especially unusable for enterprise data lakes, as the initial high cost of adapting the model, together with the cost of updating the model when new data sources are added, pose a major barrier. Furthermore, the models are *textual data focused* since they are trained and tested with corpora that contain mainly non-numeric data and also leverage LMs like BERT, which are essentially designed for textual data. In enterprise data lakes, it is crucial to apply a model with high accuracy on numerical data, since this data type plays a dominant role, often making up a much larger proportion compared to non-numerical data and providing insights into various business domains, including finance, manufacturing, healthcare and marketing.

1.4 Contributions

This dissertation addresses the above mentioned limitations of existing semantic type extraction approaches with contributions that can be summarized as follows:

Contribution 1: Weak Supervision for Adapting Semantic Type Detection

We conduct a comprehensive analysis of the task of semantic type extraction of table columns in real-world data lakes and show that existing approaches do not generalize and have to be adapted at high effort to individual data lakes. Hence, we propose a new direction of using weak supervision to generate new labeled training data with minimal manual effort. These new training data can then be used to adapt and improve the performance of learned semantic type extraction approaches on new unseen data lakes through re-training. As an implementation of this new approach, we introduce the first data programming framework for semantic labeling called STEER in this dissertation. To generate the new training data, STEER comes with a set of LFs which are used to label unlabeled table columns with semantic types. By leveraging both a small set of already labeled columns (e.g. hand-labeled) and rule-based LFs, STEER can generate new training data with minimal overhead. This enables STEER to adapt learned semantic type detection models to new, unseen data lakes semi-automatically. As a result, barriers to the use of these models in different data lake environments are removed.

Contribution 2: Steered-Labeling Process for Numerical Data

As a second contribution of this dissertation, we introduce a novel training data generation procedure called *Steered-Labeling* that can generate high-quality training data not only for non-numerical but also for numerical data columns with minimal overhead. We integrated the *Steered-Labeling* procedure as a core component in our STEER labeling framework mentioned above. The idea of the *Steered-Labeling* procedure is that we separate the labeling process into two subsequent steps: STEER first labels the non-numerical columns that are easier to label. Afterward, STEER then uses these labels to "steer" the labeling of the numerical columns. To enable a steered labeling, we strictly separate the LFs into those for labeling non-numerical and numerical columns. As a LF that leverages the generated labels from the first step of the *Steered-Labeling* process, we propose a new LF that uses a context-aware clustering method to label numerical columns with high accuracy. Our experimental evaluation shows the benefits of the sequential execution within the *Steered-Labeling* approach compared to a non-sequential execution.

Contribution 3: Numerical Corpus for Semantic Type Detection

Driven by the observation that existing corpora which have been used for building semantic type detection models are very limited and contain almost exclusively textual data, we have created the new corpus SportsTables. To reflect the characteristics of

real-world data lakes, our corpus SportsTables has on average approx. 86% numerical columns and tables that are much larger in both number of columns and rows, posing new challenges to existing semantic type detection models which have mainly targeted non-numerical columns and small tables so far. We show this effect by demonstrating the results of an extensive study using four different state-of-the-art approaches for semantic type detection of table columns on our new corpus.

Contribution 4: A New Model Architecture for Numerical Data

As the final contribution of this dissertation, we suggest our new semantic type detection approach Pythagoras, designed to support numerical along with non-numerical data. Pythagoras uses a GNN in combination with a novel graph representation of tables to predict the semantic types for numerical data with high accuracy. Thereby, we use directed edges in our graph representation to model the information flow within tables. By using this graph structure, Pythagoras can learn selectively which context information should be taken into account to establish robust predictions on numerical data. We compare Pythagoras against five state-of-the-art approaches using two different datasets and show that our model significantly outperforms the baselines on numerical data.

1.5 Outline

The remaining chapters of this dissertation are structured as follows: Chapter 2 first show the limitations of applying a semantic type extraction model to a new unseen data lake based on experimental results. It also explains our new approach of using weak supervision to eliminate the limitations. Chapter 3 presents our data programming framework STEER, which allows to semi-automatically adapt learned semantic type detection models to new unseen data lakes. The chapter additionally shows an extensive evaluation of STEER on four different data lakes using two separate models that implement different learning paradigms. Chapter 4 introduces our new semantically labeled corpus SportsTables, which we built to tackle the shortcomings of existing corpora for the task of semantic type detection of table columns. A comparison of the statistics regarding the distribution of column data types and sizes of existing tables in the corpus explains the main characteristic differences between our corpus and the existing ones. Because existing semantic type extraction models do not provide accurate performances when used on numerical data, we present Pythagoras which is a novel model specifically designed to

robustly handle numerical data columns in Chapter 5. Finally, Chapter 6 concludes this dissertation with a summary and outlines future work.

2 Weak Supervision for Learned Semantic Type Extraction

Publication. The work on using weak supervision for learned semantic type extraction is published in the peer-reviewed publication "Towards Learned Metadata Extraction for Data Lakes" in the *Datenbanksysteme für Business, Technologie und Web (BTW 2021), 19. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme" (DBIS), 13.-17. September 2021, Dresden, Germany, Proceedings*

Contributions of the author. Sven Langenecker is the leading author of the publication [62] mentioned above. He is responsible for the analysis of using learned semantic type detection approaches on unseen data lakes and the proposed weak supervision approach. The co-authors Christoph Sturm, Christian Schalles and Carsten Binnig contributed invaluable feedback. All authors agree with the use of the publication for this dissertation.

In this chapter, we first present our initial steps in this dissertation, in which we analyze the quality of state-of-the-art learned approaches for semantic type extraction on new, previously unseen data lakes. We show the limitations of existing DNN based approaches for semantic type detection, which is that they are always data lakes specific and do not generalize to different data lakes. For deploying on a new unseen data lake, these approaches always require a costly adaption. To tackle this limitation, we suggest a new direction of using weak supervision to generate a much broader set of labeled training data for semantic type detection with low manual effort. By re-training the learned model with the automatically generated new training data, the model is adapted to the unseen data lake with minimal effort.

2.1 Study of Using Existing Learned Approaches

In the following, we present the results of our study of using learned semantic type extraction approaches on new real-world data lakes. For our initial study, we selected Sato [113] as a recent DNN-based approach. As already mentioned, while other approaches exist, our initial investigations with Sato show the inherent constraints shared by all

current models and demonstrate the need towards new solutions for adapting learned semantic type detection approaches to new data lakes. We will now first describe the dataset and the methodologies used in our experiments before we present our results.

2.1.1 Dataset and Methodology

Dataset. As a dataset, we use the Public BI Benchmark [31] data corpus in this study. The data corpus contains real-world data, extracted from the 46 biggest public workbooks in Tableau Public [101]. In this corpus, there are 206 tables each with 13 to 401 columns. The main reason for choosing this corpus for our study was that it contains labeled structured data from different real-world sources across various domains (e.g., geographic, baseball, health, railway, taxes, social media, real estate). Hence, the benchmark comes with a high diversity and heterogeneity of data sources that can typically also be found in data lakes of enterprises today.

Methodology. As mentioned before, the inherent problem of DNN-based approaches for semantic type extraction is that they rely on a representative training data set. To put it differently, if the training data set does not cover the variety of cases that are also seen in the real-world data, the performance of the learned models quickly degrades. As part of our analysis, we wanted to see to which extent this inherent limitation influences the overall quality of a learned approach such as Sato. For the study, we thus annotated the table columns in the Public BI Benchmark manually with the correct semantic types of Sato. For the annotation, we first preprocessed the data automatically and searched for string matches between the column headers of the tables in the Public BI Benchmark and the semantic types supported by Sato. However, this was only useful for a small fraction of the columns because many headers were meaningless or they did not match any of the semantic types. To guarantee the correctness of labels every column was additionally inspected and missing types where added manually.

2.1.2 Results of the Study

As a first question, we analyzed the coverage rate of the 78 semantic types supported by Sato in the Public BI Benchmark to see to which extent a pre-trained model can support real-world data if no new training data is used for re-training. For this question, we analyzed what fraction of columns in the Public BI Benchmark had a type that was covered by the training data set of Sato. The main result of this analysis was that only 10.6% of the columns are assignable to one of the semantic types. To put it differently, almost 90% of the columns in the Public BI benchmark have semantic types that cannot

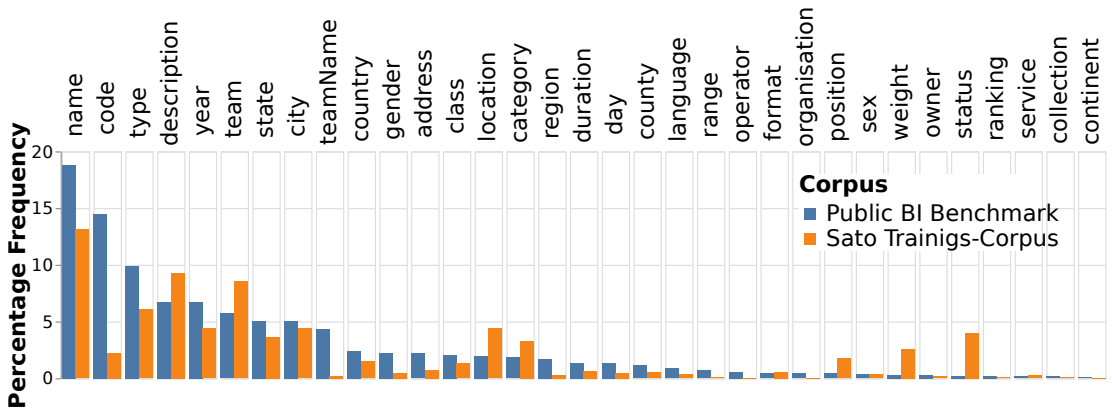


Figure 2.1: Distribution of semantic types in training data of Sato and the Public BI Benchmark

be detected by using Sato, as these are semantic types that are not covered by Sato's training data.

As a second question, for the columns of the Public BI Benchmark that have types which are supported by Sato, we then wanted to see how the distribution of the 78 semantic types in the training data used for Sato and the Public BI Benchmark looks like. The reason is that different distributions of labels in the training and testing data can have a negative impact on the overall quality of a learned approach. Based on our annotations, we thus further analyzed the percentage frequency distribution of the occurring semantic types in the Public BI Benchmark compared to the used data corpus from Sato (VizNet [45] corpus). As can be seen in Figure 2.1, the frequency for many semantic types in both data sets (i.e., original training data of Sato and the Public BI Benchmark), however, is almost identical. Therefore, Sato should in principle be able to achieve almost the same prediction quality for these columns of the Public BI Benchmark as for the test data of the VizNet corpus. In the paper of Sato [113] they reported on the test data of VizNet a macro average F1-Score of 0.735 and a support weighted F1-Score of 0.925.

As a final question, we thus aimed to analyze the 10.6% of the columns in the Public BI Benchmark that are in principle covered by the training data of Sato. To conduct this study, we used the pre-trained Sato model and applied it to only this fraction of the data of the Public BI Benchmark. For this subset, Sato achieves an F1-Score (macro average and weighted[1]) of 0.090 and 0.300 respectively, which is also shown in Table 2.1 in our

[1] **F1-Score macro average**: averaging the unweighted mean F1-Score per label
F1-Score weighted average: averaging the support-weighted mean F1-Score per label

evaluation in Section 2.3. The original paper [113] reports an F1-Score of 0.735 and 0.925 on the VizNet[2] data corpus. This indicates that the data characteristics of the supported data types of the Public BI Benchmark is different from the data characteristics of the training data of VizNet and thus Sato can not infer types in a robust manner (even if they should be supported in principle).

Main Insights. As suspected, our study has shown that a DNN-based model such as Sato trained on one data set can only cover a fraction of data types of a new dataset. Moreover, for the overlapping data types, the accuracy is still pretty low due to different data characteristics of the training data and the new dataset. While the results of our study are specific to Sato, we believe that our findings are much more general and translatable to any learned approach that relies on manually curated training data (which is inherently limited as discussed before). Hence, a new approach is required where one can easily adapt learning-based models for type extractors to new datasets that covers types and data characteristics not covered in the available manually labeled training data. As a solution for this requirement, we next present our new weak supervision approach in the next section.

2.2 Weak Supervision to Adapt Learned Approaches to New Data Lakes

The root cause of why DNN-based approaches such as Sato often fail to extract semantic types on a new dataset is that the training data lacks generality as discussed before. The main idea of using weak supervision is to generate a broad set of new labeled training data extracted with only minimal effort from the new data lake where the model should be deployed. By subsequently re-training the model with the additional generated training data, the model is adapted to the data lake and can operate robustly on it. In the following, we discuss our initial ideas for such an approach and present a concept of a LF which can generate additional training data on a new data lake.

2.2.1 Overview of Our Approach

Figure 2.2 shows an overview of our approach. The main idea is that based on a set of simple LFs, we generate new (potentially noisy) training data that is then used to re-train a model such as Sato to increase the coverage of data types and data characteristics of

[2] https://github.com/mitmedialab/viznet

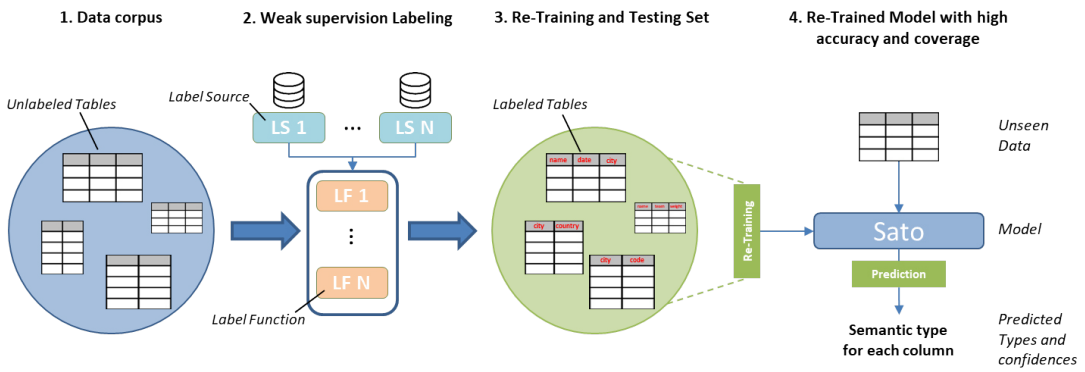


Figure 2.2: Overview and step-by-step procedure of our weak supervision approach, starting from (1) structured unlabeled data stored in a data lake to (2) semantic type labeling using weak supervision, (3) re-training/testing of Sato and (4) using re-trained Sato model for predicting semantic types of the data in the data lake.

the learned model. In other words, we apply the ideas of data programming discussed in [92] for the domain of semantic type extraction of table columns.

In our approach, we differentiate between two different classes of LFs for generating new training data: (1) The first class are LFs that can generate labels (i.e., semantic types) for completely new semantic types in a data lake that are not yet covered by a manually labeled training data set. LFs of this class can be, for example, regular expressions, dictionary look-ups, or other techniques such as using alignment with existing ontologies. This set of label functions can thus help to expand the semantic types covered in a training data set. (2) Second, as we have seen in our study, another problem of learned approaches such as Sato is that they often fail to predict semantic types even if in principle the semantic type is already covered by the training data. The main reason for this case is that the training data does not cover the wide spectrum of data characteristics that might appear in a new data set. Hence, as a second class of LFs our new approach supports functions that can generate new labeled columns that cover more data characteristics (e.g., new values) for data types that are already available in a training data set. One idea for a LF of this class is the use of word embeddings [75] to cluster new unlabeled with already labeled columns and thus generate new labeled columns for existing semantic types. A more detailed description of such a LF is given below.

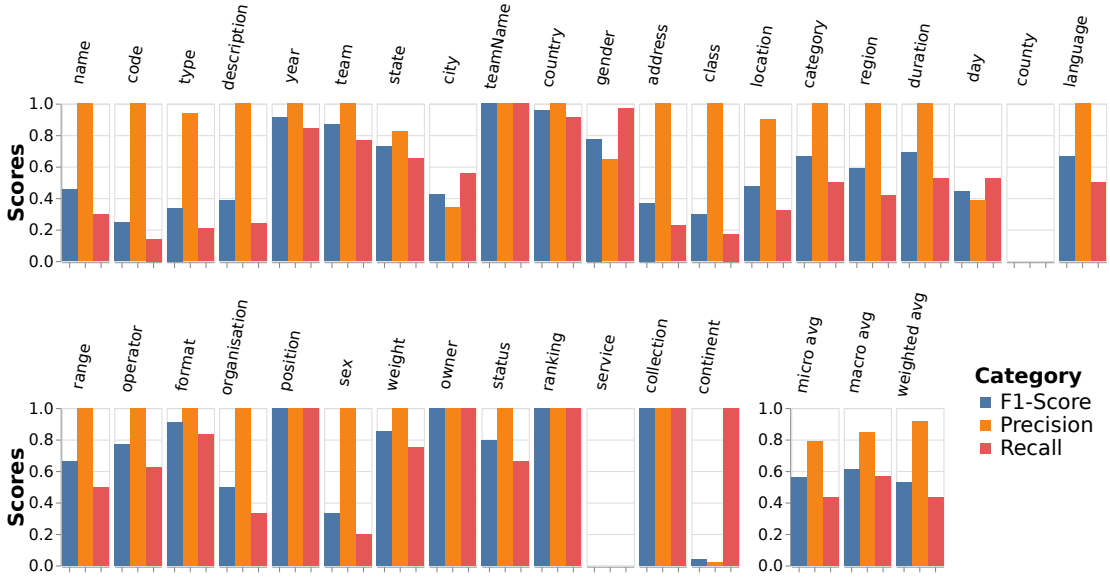


Figure 2.3: Performance of clustering semantically similar columns.

2.2.2 Label Generation Using Clustering

To present a basic idea of a LF for generating more labeled training data for an existing semantic type, we implemented a method based on clustering that we briefly introduced before. The main idea is that we start with a small training corpus of labeled columns and by clustering new non-labeled to the labeled columns, we can derive new labeled training data.

To implement this labeling approach, we first compute column embeddings for labeled and unlabeled columns based on word embeddings of individual values. We currently use *Google USE*[3] that was trained on 16 different languages and showed good results as word embeddings. But in principle, we could also use other word embeddings. Based on the embeddings of individual values, we compute an embedding for all values of a column by calculating the average across the embeddings of all values which is the dominant approach for building representations of multi-words [13, 63, 99].

Once we computed an embedding for all values of a column, we next carry out the clustering of labeled and unlabeled columns based on these embeddings. For this step, we use an agglomerative clustering algorithm[4]. In our prototype, we use this clustering method to form groups based on the cosine similarity of vectors (i.e., our embeddings)

[3] https://tfhub.dev/google/universal-sentence-encoder-multilingual-large/3
[4] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html#sklearn.cluster.AgglomerativeClustering

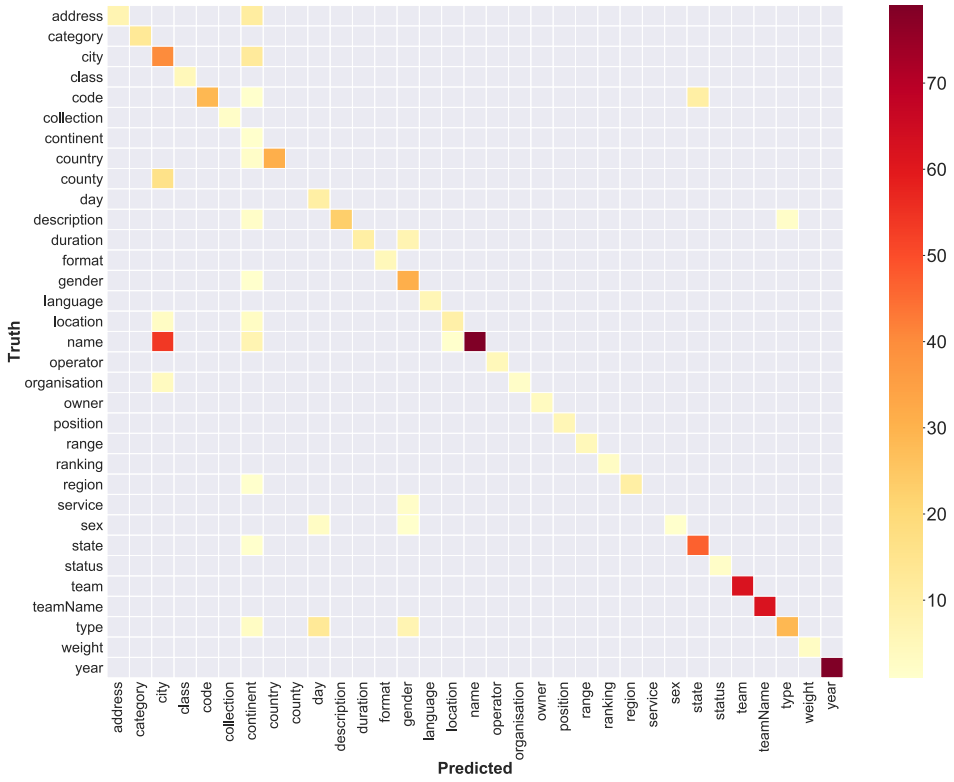


Figure 2.4: Confusion matrix of the clustering method

and a distance threshold (discussed below), rather than to generate a fixed number of clusters to not generate a fixed number of clusters. Once clustered, we then compute a semantic type per cluster based on the majority vote of columns with the same label. [69] represents a system called Raha, which relies on a similar idea for generating training data but for error detection and not for semantic type extraction.

A key parameter to be set in our clustering approach is the distance threshold which can vary between 0.0 and 1.0 (i.e., a lower value means that we produce more clusters). In our experiments, we used a threshold of 0.1 based on a hyper-parameter search on the already labeled columns. This threshold provided high accuracy on the broad spectrum of data sets in the Public BI Benchmark. In the next chapter (Chapter 3), in which we present our comprehensive data programming framework including multiple LFs, we will give a more detailed description of the implementation of this LF (see Section 3.3.1).

Initial Results. To analyze if the basic idea of clustering is working, we conducted a small experiment where we measure how well the clustering approach works on the Public BI Benchmark using our annotations of the 78 Sato types. By clustering, we wanted to

see whether columns with the same type would be assigned to the same cluster. As we see in Figure 2.3, with a few exceptions, the clustering algorithm achieves high precision. This means that there is a very high probability that all elements in one cluster belong to the semantic type representing the specific cluster. For many types, we achieve an F1-Score of 1.0 such as for the semantic types *teamName*, *position*, *owner*, *ranking* and *collection*. However, there are also a few semantic types for which only lower values are achieved for precision, recall and F1-score (e.g., *city*, *county*, *service*). By looking at the confusion matrix (see Figure 2.4), we can see why these lower values arise. For example, the semantic type *county* was always assigned to a cluster, which represents the semantic type *city*. This is explained by the fact that county and city names are very often the same, making it very difficult to distinguish between them.

Moreover, in a second experiment, we wanted to show the robustness of our clustering approach to different data characteristics. To demonstrate this we analyzed the entropy and the jaccard-coefficient for all columns with the same semantic type in the Public BI Benchmark. The intuition is that columns with a high entropy (i.e., a high degree of divergence) or pairs of columns which have a low jaccard-coefficient (i.e., where columns values are not overlapping) are harder to cluster. Overall, our approach assigns column pairs with the same semantic type to the very same cluster even if they strongly vary in the entropy or have a low overlap (i.e., a low jaccard-coefficient).

2.3 End-to-End Evaluation

In the section before, we have already shown that the basic idea of weak supervision can help to generate training data by clustering to improve the robustness w.r.t different data characteristics. In the following, we report on the initial results of using this approach in an end-to-end evaluation to show how this can boost the performance of learned semantic type extraction approaches such as Sato on unseen data lakes.

Setup and Data Preparation. We implemented our approach for automatic labeling in Python using the Google USE embeddings as mentioned before. Moreover, we used the source code provided by Sato[5] for training and evaluation. However, Sato is designed to be built and trained from scratch. Hence, we extended Sato with the appropriate functionality for incremental re-training.

End-to-End Results. To show the end-to-end performance of our approach, we restricted ourselves to the 10% of the Public BI Benchmark data that is supported by

[5] https://github.com/megagonlabs/sato/tree/master

Table 2.1: Performance comparison of the models on Public BI Benchmark

	Macro average F1	Precision	Recall	Support-weighted F1
Sherlock (not re-trained)	0.114	0.375	0.309	0.322
Sherlock (re-trained)	0.806	0.879	0.859	0.860
Sato (not re-trained)	0.090	0.322	0.304	0.300
Sato (re-trained)	**0.811**	**0.912**	**0.894**	**0.894**

Sato and its semantic types. In the experiment, we first generated additional training data and then re-trained the pre-trained Sato model with our additionally labeled data. Thereby, to generate the additional training data for the Public BI Benchmark, we used the clustering approach as discussed before as LF. We split the Public BI Benchmark into a training and testing set and used the training set as input for LF and the test set to evaluate the re-trained Sato model. As we see in Table 2.1, after re-training the Sato model with the synthesized training data of our approach, Sato achieves F1-Scores (macro average and weighted) of 0.811 and 0.89 respectively. This is a significant improvement of almost +0.60 compared to the performance of Sato without re-training. In addition to show that our approach also generalizes to other learned approaches, we furthermore used Sherlock [50] (without and with re-training). As shown in Table 2.1, this leads to a similar performance gain. In summary, these results show that our approach is in principle able to boost the performance of learning-based approaches that have been pre-trained on only a small training data set not covering all data characteristics found in a new unlabeled data lake.

2.4 Key Findings

We will now present the key findings of this chapter and discuss the motivation for the next proposed directions in this dissertation. While there are existing learned approaches that can be used for extracting semantic types in data lakes, they cannot be directly used for unseen real-world data lakes since they only support a limited set of semantic types as we have shown in our study. Furthermore, the trained models do not cover the broad spectrum of different data characteristics found across data lakes. Therefore, it is necessary to adapt the learned models to an unseen data lake before using them. However, this adaptation involves a high amount of effort, as it requires a large set of new training data that reflects the characteristics and semantics of the unseen data lake. To tackle this problem, we suggested a new direction of using weak supervision

to automatically generate additional labeled data. Subsequently, this new training data are then used to re-train the existing learned model. Our initial evaluation of using this weak supervision technique demonstrated that it can be successfully applied to adapt learned models to new data lakes semi-automatically.

In this chapter, we showed with only one LF (Labeling by Clustering) and an initial experiment that the weak supervision approach is promising and can be used in principle for adapting learned models to new unseen data lakes. In order to cover the different data characteristics and semantics of real-world data lakes, it is necessary to provide a holistic labeling framework for semantic type labeling that comes with multiple predefined LFs. Additionally, we see a requirement for additional LF that can handle specifically numerical data and their underlined semantic types, because numerical data are less descriptive than textual data. This motivates the further directions proposed in the next chapter.

3 Steered Training Data Generation for Semantic Type Detection

Publication. The work on steered training data generation for learned semantic type detection is published in the peer-reviewed publication "Steered Training Data Generation for Learned Semantic Type Detection" in the *Proc. ACM Manag. Data*

Contributions of the author. Sven Langenecker is the leading author of the publication [61] mentioned above. He is responsible for the proposed labeling framework STEER, the included pre-defined LFs for semantic labeling of table columns, the core component Steered-Labeling, experimental evaluations and the manuscript. The co-authors Christoph Sturm, Christian Schalles and Carsten Binnig contributed invaluable feedback. All authors agree with the use of the publication for this dissertation.

In the previous chapter, we observed that a key challenge for deploying semantic type detection models to new data lakes is the needed adaption to the data characteristics and semantics of the data contained in the data lake. To address this adaption challenge with minimal effort, we have shown by an initial experiment with one LF that the weak supervision approach can be used in principle. This motivates the next direction of this dissertation - a holistic data programming framework for semantic type detection of table columns.

Hence, in the following, we will present the first data programming framework for semantic type detection called *STEER*. *STEER* is based on the idea of weak supervision to generate new labeled training data for a new unseen data lake with numerical and non-numerical data types. The generated training data of *STEER* can be used to re-train an existing learned semantic type detection model to adapt it to the new environment. At the core, to generate labeled training data (i.e., pairs of columns with data and semantic types), we propose a new label generation process called *Steered-Labeling*. The intuition is that in *Steered-Labeling* we separate the process into two subsequent steps: *STEER* first labels the non-numerical columns that are easier to label. Afterwards, *STEER* then uses these labels to "steer" the labeling of the numerical columns. With this, *STEER*

is able to not only generate high quality training data of textual columns but also to semantically label numerical data with a very high precision.

To summarize in this dissertation the contributions of *STEER* and the integrated *Steered-Labeling* procedure, the upcoming sections are structured as follows. First, in Section 3.1, we give an overview of the *STEER* framework, its main components and present the details of our *Steered-Labeling* process. Afterward, Section 3.2 presents the details of our implemented LF for numerical data which is integrated into our steered-labeling approach. Section 3.3 then shows the LFs provided by *STEER* for labeling non-numerical columns. Afterward, in Section 3.4 we then show the results of our extensive experimental evaluation of *STEER* in different scenarios. In the last Section 3.5, we give a summary of this chapter and explain the relevance of the next directions in this dissertation.

3.1 Overview of *STEER*

STEER implements a novel labeling framework that can generate high quality labeled data for training semantic type detection models on a new unseen data lake with minimal overhead. We will now first introduce a general overview of *STEER* and then focus more specifically on the mentioned *Steered-Labeling* core component of the labeling framework.

3.1.1 The Labeling Framework

STEER provides a labeling framework based on the idea of weak supervision that comes with different classes of LFs for training data generation. The novel aspect of *STEER* is that it comes with a new training data generation procedure called *Steered-Labeling* that can generate high quality training data not only for non-numeric but also numerical columns that are currently not supported by any of the existing learned approaches for semantic type detection. Figure 3.1 shows the overview of all phases in *STEER* to generate training data for a semantic type detection model. Overall, the labeling framework of *STEER* can be divided into three different phases: *(1) Label Function Construction: STEER* already comes with a wide spectrum of LFs for numerical and non-numerical semantic types. These LFs can be extended by a data engineer to support specific column values or LFs for highly specific data types. *(2) Steered Training Data Generation:* The steered training data generation phase is the core of *STEER* which uses the LFs and creates training data for non-numerical and numerical data. The idea of the steered-labeling procedure is that at first *STEER* labels the non-numerical columns

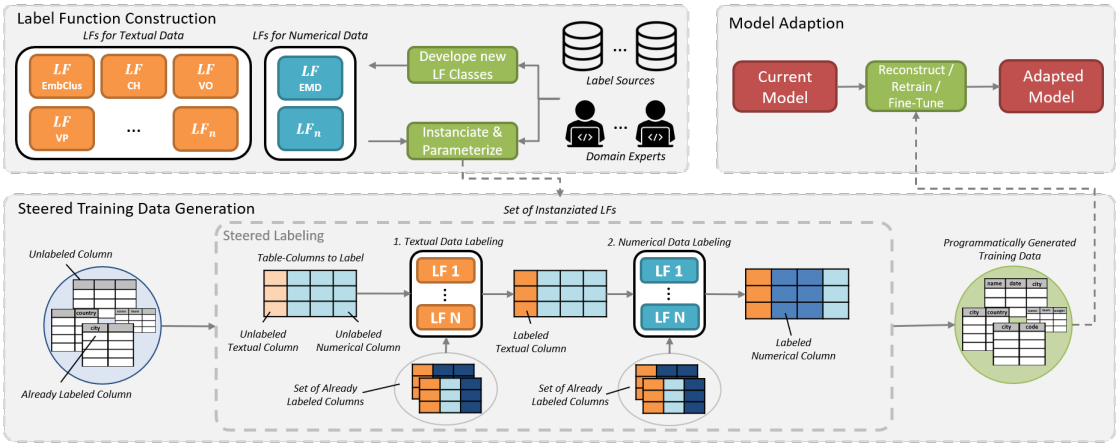


Figure 3.1: Overview of *STEER*. The main idea is that *STEER* provides a labeling framework that comes with different classes of LFs for training data generation. These LFs can be extended and need to be instantiated by domain experts with minimal overhead, for example, by providing some examples of labeled columns. Afterwards, *STEER* creates labels (i.e., pairs of columns and semantic types) that can be used as training data. *STEER* divides the LFs into groups of functions for textual data and functions for numerical data and implements at the core a novel steered-labeling process that first labels textual columns and afterwards numerical columns so that the LFs for numerical data can benefit from the labels generated before. The steered-labeling process generates high-quality training data with minimal overhead on a new unseen data lake that can then finally be used to re-train or fine-tune an existing learned metadata extraction model.

that are easier to label. Afterwards, *STEER* uses these labels to "steer" the labeling of the numerical columns as shown in Figure 3.1 (bottom). The intuition of steering is that tables with similar semantic types for textual columns also have similar semantic types for numerical columns. For example, a table about baseball teams has a column *sports team*. If a column *sports team* is present in a table, then the numerical values of the table's numerical columns are more likely to be about *height* and *weight* of players and not about *air pressure* or other numerical columns. We explain the steered-labeling procedure in more detail below. *(3) Model Adaption:* Finally, in the adaption phase an existing model such as Sato [113] or TURL [20] is adapted to the data lake by re-training or fine-tuning the model using the previously automatically generated training data.

3.1.2 Steered-Labeling Procedure

As core contribution of the training data generation in *STEER*, we introduce a new steered-labeling method to generate training data for labeling non-numerical (textual) and numerical columns. However, as we show in our evaluation, labeling numerical columns with LFs is generally more difficult than textual columns which contain semantic meaningful values such as names of cities or sports teams. In order to overcome this inherent problem and provide high precision also for LFs for numerical data, our idea is that with the *Steered-Labeling* approach numerical LFs can rely on context data from a table; i.e. the already labeled textual columns.

To enable a steered labeling, *STEER* strictly separates the LFs into those for labeling non-numerical and numerical columns. In a first step, *STEER* uses the LFs for non-numerical data to label the subset of columns in the data lake that does not contain numerical data. Based on these labeled columns, *STEER* then aims to label the numerical columns. For this, *STEER* comes with a LF that is generally applicable for all numerical types.

The idea of this LF is that (1) a small fraction of numerical columns that represent the numerical types in the data lake need to be labeled upfront. Afterwards, these labeled examples are then used to (2) generate labels for other non-labeled numerical columns by using a clustering-based LF that clusters numerical columns with similar value distributions. Steering during clustering helps the clustering-based labeling to group numerical columns from similar tables and thus increase the labeling quality.

For example, if unlabeled numerical columns of a table that also has a column *sports team* should be labeled, steering would prefer tables for the cluster-based labeling that have a *sports team* column together with other labeled numerical columns. However, it is important to note, that steering in *STEER* is optional; i.e., cluster-based labeling can also be used without steering which is needed when no other table with the same semantic non-numerical type exists or tables contain only numerical columns.

In our experiments, we show that our novel steered-labeling process leads to huge benefits compared to a non steered-labeling process where all LFs are executed in parallel. To the best of our knowledge, the labeling framework of *STEER* is the first which uses such a steered-labeling procedure to semantically label non-numerical and numerical columns.

Algorithm 1 LF EMD: The LF is based on clustering columns in tables which share the same context using the the earth mover's distance as similarity metric between numerical columns.

```
1: emd_threshold ← precalculated threshold
2: C_ln, T_ln ← set of labeled numerical cols and their table
3: C_un, T_un ← set of unlabeled numerical cols and their table
4: for All c_un in C_un do
5:     emd_results = []
6:     C_lt1 ← set of labeled textual cols of table T_un of c_un
7:     for All c_ln in C_ln do
8:         C_lt2 ← set of labeled textual cols of table T_ln of c_ln
9:         if length(C_lt1 ∩ C_lt2) > 1 then
10:            emd_results.append(earth_mover_dist(c_un, c_ln))
11:        end if
12:    end for
13:    sort(emd_results)
14:    if size(emd_results) > 0 then
15:        if emd_results[0] < emd_threshold_{c_ul} then
16:            assign semantic type of emd_results[0] to c_un
17:        end if
18:    end if
19: end for
```

3.2 Labeling Numerical Columns

Existing approaches for annotating a type to a numeric column typically compare only the distributions of the data values from labeled to unlabeled columns using earth mover's distance like in [56, 114] or the p-value of statistical hypothesis test like in [80, 91]. However, these naïve approaches are typically more inaccurate for data lakes since usually numeric columns have a lower entropy than textual columns and thus have a lower information content which makes it harder to differentiate numerical columns from one another[1]. Therefore the information provided by the value distribution of numerical columns is too limited and leads to many false annotations if it is the only semantic typing criterion. In order to overcome this inherent problem and boost the precision of the semantic type detection, our idea is instead to rely on context data from the table (e.g. information about neighboring columns) which we use to steer the labeling of numerical columns. In the following, we explain a LF that is based on this idea.

[1] Generally numeric values can be encoded with much less bits than string values resulting in lower overall entropy values [98]

3.2.1 Labeling by Context-aware Clustering

The idea of *Steered-Labeling* is integrated into a LF of *STEER* that is based on the idea of context-aware clustering. In order to use this LF, a data engineer has to provide at least one table with a labeled column per semantic type t the LF should create labels for. Afterwards, the LF uses the annotated column that has type t and the table T the column is part of to create other labeled numerical columns of the same type. Moreover, the LF assumes that columns with textual semantic data types have already been labeled as discussed in Section 3.1. LFs for textual columns will be described in detail in the next section.

The pseudocode of the LF which labels numerical columns is shown in Algorithm 1 and works as follows: Given a set of manually labeled numerical columns C_{ln} (and the tables T_{ln} they are part of), and a set of unlabeled numerical columns C_{un} (and the tables T_{un} they are part of), the LF iterates over the unlabeled columns c_{un} to label them in the respective iteration step (line 4-19). In this iteration step, the LF first retrieves context information about c_{un}; i.e., we retrieve the semantic types of all non-numerical columns of table T_{un} (line 6). In the next step, we iterate over each labeled column c_{ln} in C_{ln}. Afterwards, we then retrieve context information also for c_{ln} which is part of table T_{ln}.

In case the two tables — T_{un} and T_{ln} — share at least one column with the same semantic type, we compute the earth mover's distance between c_{un} and c_{ln} as a metric of the similarity of both columns (line 10). As such, we compute the earth mover's distance only against labeled numerical columns in tables that share the same context which improves the accuracy of the labeled training data significantly as we show in our evaluation in Section 3.4. Each earth mover's distance measurement to a labeled numerical column is then stored in a list, which we finally sort in ascending order (line 13); i.e., the most similar labeled column is first.

3.2.2 Determining the EMD Threshold

In the LF *EMD* as shown in Algorithm 1, the values of the two numerical columns are not normalized before the earth mover's distance is computed as a similarity measure. A normalization of the distribution would lead to a loss of information and to many false matches between labeled and unlabeled columns. Therefore the earth mover's distance between the values of the two numerical columns is not normalized.

As a result, the earth mover's distance values of a comparison between two columns varies in a value range between $[0, MAX\_EMD]$ where MAX_EMD can be arbitrarily large. As such, setting a fixed threshold value is not possible. Instead, we set a threshold

individually per unlabeled column c_{ul}. The intuition is that we find a threshold that considers the value distributions of the numerical values in that column.

Moreover, determining the threshold correctly is very important. When setting the threshold too low, we might not assign any numerical type while when setting the value too high, we might see a lot of false labels. Hence, to set the threshold we first compute the distribution of earth mover's distance values across a representative set of pairs of unlabeled numerical columns. That way, we can decide what a significant difference between two earth mover's distance values is and thus also when two columns are similar; i.e., the difference of the earth mover's distance value is not significant.

To calculate the threshold for the earth mover's distance we use the following equation:

$$emd\_threshold_{c_{ul}} = 0.4\, Quantile\_EMD * std\_c_{ul}$$

where $0.4\, Quantile$ comes from the the distribution of all earth mover's distance measurements between labeled columns and unlabeled columns and *std* is the standard deviation of the unlabeled column c_{ul} that is supposed to be labeled at the moment.

3.2.3 Numerical-only Tables

As described above the context-ware LF *EMD* for labeling numerical data relies on existing textual semantic types of the neighboring table columns. During computing the semantic similarity between unlabeled and labeled columns as mentioned before, two different situations can thus occur, where no context information is available. (1) Table T_{un} of the unlabeled numerical column c_{un} contains only numerical columns or no textual columns are annotated. (2) Table T_{ln} of the labeled column c_{ln} contains only numerical columns or no annotated textual columns and therefore no context information is available. We address these situations as follows. On (1) we measure the earth mover's distance against all labeled columns available without considering any context information. In case of (2) the earth mover's distance measure is made against the labeled column due to the missing context information.

3.3 Labeling Non-Numeric Columns

As mentioned in Section 3.1, *Steered-Labeling* first labels non-numerical columns and subsequently the numerical columns. For annotating semantic types to non-numerical columns, our labeling framework comes with a set of four different LFs. We separate

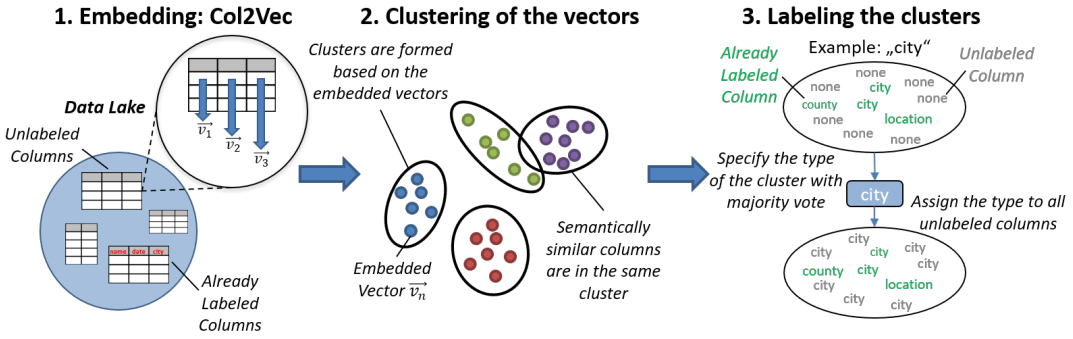


Figure 3.2: Labeling by Embedding Clustering *EmbClus*. In a first step, embeddings are computed for all columns. Afterwards, by clustering new labels are generated using existing labels from already annotated table columns.

these LFs in STEER into two categories: (1) Generic LFs that work without any manual adaption of the LF and (2) Domain-Specific LFs which require adaption of the LFs to the data types of the data lake. These LFs rely on the contribution of domain experts knowledge by providing some limited number of example values for a domain-specific data type. In the following, we will explain the different LFs for labeling non-numerical columns in detail.

3.3.1 Generic Labeling Functions

STEER provides two types of generic LFs that can be used in a domain independent manner.

Labeling by Embedding Clustering. The first generic LF is the one we already introduced in Chapter 2 initially. Since we have only presented the general idea, we will now explain the function in more detail. As it relies on embedding clustering, we call it *EmbClus*. Similar to labeling numerical types by clustering, this LF requires that a small set of columns in a data lake is already annotated with semantic types. This is often the case since data lakes are constantly growing in size and thus some columns might have a semantic type. However, one could also use this LF if a domain expert is willing to first label a small (representative) set of columns in the data lake manually.

The main idea of *EmbClus* is to use column embeddings to cluster columns with similar values and thus generate labels for previously unlabeled columns. Figure 3.2 shows the implemented algorithm and the individual detailed processes. In the first phase, we compute column embeddings for both labeled and unlabeled table columns based on

word embeddings of individual column values. As word embeddings, we currently use *Google USE*[2] [111] that was trained on 16 different languages.

In principle we could also use other word embeddings, but multilingual models can better cover the spectrum of different "custom" semantic types in different enterprise data lakes. Furthermore, the model is also designed to embed sequences of words (e.g. sentences) and thus gives us the possibility to embed column values that contain more than just one word. Based on the embeddings of individual values, we compute an embedding for all values of a column by calculating the average across the embeddings of all values which is the dominant approach for building representations of multi-words also mentioned in other papers [99].

Once we computed an embedding for all columns, we cluster labeled and unlabeled columns together based on these embeddings. The intention here is that clusters are formed with columns that have the same semantic type. For this step, we use the hierarchical agglomerative clustering algorithm[3]. In *STEER*, we use this class of clustering method not to generate a fixed number of clusters, but to form groups based on the cosine similarity of vectors (i.e. our embeddings) and a distance threshold that we discuss below.

Once clustered, we then compute a semantic type per cluster based on the majority vote of the labeled columns in that cluster. In the absence of any labeled column in a cluster, we assign no semantic type to that cluster. A key parameter to be set in our clustering algorithm is again the distance threshold, where lower values mean that we produce more clusters. In our experiments, we used a threshold of 0.01 based on a hyper-parameter search on the already labeled columns. This distance threshold provided good results on the broad spectrum of datasets in all four observed fictive data lakes.

Labeling by Column Headers (CH). Another generic LF is the *labeling by column headers* (CH). The main idea of labeling by column headers (CH) is to use the original column headers as information to derive the semantic type of the column. The original column headers often represent type information but do not directly represent semantic types (e.g., a unified ontology of the enterprise) of the data lake catalog. For generating labels with existing column headers, we again use pre-trained language embeddings to embed column headers as well as the semantic types of the given ontology that should be used for the catalog.

[2] https://tfhub.dev/google/universal-sentence-encoder-multilingual/3

[3] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html#sklearn.cluster.AgglomerativeClustering

Based on these embeddings, we match them similar to the work in [47]. As similarity measure between the column header and each semantic type, we use the cosine similarity of the vectors. Based on the similarity measure, the LF then assigns the semantic type as label to the column with the highest cosine similarity. Moreover, the cosine similarity must have a minimum similarity threshold value. In our experiments, we use a similarity threshold of 0.9 based on a hyper-parameter search on the already labeled columns. Since this LF is based on column headers, it can in general be applied to columns with textual data as well as to columns containing numerical data.

3.3.2 Domain-Specific Labeling Functions

In this section we now present our implemented LFs of the second category. *STEER* provides two types of that LF category which can be specialized by the domain expert to their own data lake.

Labeling by Value-Overlap (VO). The central concept of *Labeling by value-overlap* (VO) is to enable a domain expert to provide a list of common values for a semantic type. To give an example, a domain expert in the field of American football can provide a list of typical team names (e.g. [atlanta falcons, new england patriots,...]) for the semantic type *american_football.football_team* that occur frequently in table columns. The LF uses this list to check how many values in an unlabeled column exactly match one of the values in the provided list. Notice that this is an exact string matching, where we first convert all strings to lower case and than apply the comparison. If the number of matching values is over a threshold, the LF assigns the corresponding label to that column. In our experiments we use a threshold of 20%, which is however a hyperparameter that can be tuned per data lake.

Labeling by Value-Patterns (VP). The LF *Labeling by value-patterns* (VP) allows domain experts to specify a list of general patterns via regular expressions (regex) for a semantic type. As applied in the previous LF, we use the list of regular expressions to check how large the fraction of column values is for which a pattern matching was successful. If this fraction is over a predefined threshold, the column gets the according type. In our prototype we use a threshold of 20%, which generated high quality training data for all considered datasets.

3.3.3 Discussion

In *STEER*, each non-numeric column is labeled by each existing non-numeric LF. Therefore, after processing all LFs, discrepancies can exist, since one column can get several

Table 3.1: Characteristics of the four data sets used as data lakes.

Dataset	#Tables	Avg #Cols per Table	#Types	Ontology
Public BI	160	8.96	33	DBPedia
TURL-Corpus	401,538	1.59	105	Freebase
Public BI Num	170	13.64	52	Custom
SportsDB	78	17.83	18	Custom

different semantic types from different LFs. In this case, *STEER* combines the labels from multiple LFs by using a majority vote. We also tried out other strategies such as using a generative model that is trained on the output of the LFs (which is a strategy suggested in [92]). However, in all our experiments the majority vote provided superior performance.

3.4 Experimental Evaluation

In the following, we introduce the four datasets (*Public BI* & *TURL-Corpus* & *Public BI Num* & *SportsDB*) and describe the evaluation methodology. Moreover, we use *STEER* to re-train two different models for semantic type detection (Sato [113] and TURL [20]) on these data lakes.

3.4.1 Datasets

For evaluating *STEER*, we use a total of four different real-world data sets with a large number of different tables as data lakes (see Table 3.1). Overall, we have two data lakes with only non-numeric semantic types (*Public BI* and the *TURL-corpus*) while we have two data lakes with numeric and non-numeric semantic types (*Public BI Num* and *SportsDB*). We use the data lakes with non-numeric semantic types to generally evaluate in principle how well our labeling framework can adapt learned models to new data lakes. Instead, the other two data lakes with numeric semantic types are used to evaluate the benefits of our *Steered-Labeling* approach. We later on provide also more details on the distribution of numeric and non-numeric data types for the two data lakes for which we annotated numeric types (see Table 3.2).

Public BI [31]. As first data lake, we use the previously introduced (in Section 2.1.1) data corpus *Public BI Benchmark* [4], which we have already applied in the initial experiments

[4] https://github.com/bogdanghita/public_bi_benchmark-master_project

in Chapter 2. As mentioned before, the original data corpus contains no semantic type annotations of table columns and we therefore annotated the columns manually with the correct semantic types. For the annotations, we adopted the same semantic types used in Sato [113] which is originally trained on the VizNet data set. This allows us to evaluate a setup with a data lake that uses the same semantic types as a already trained model but might use different data distributions in the respective columns. As we showed in the study in Section 2.1, out of Sato´s 78 semantic types, only 33 were present in the *Public BI* data corpus. All annotated table columns with one of these 33 semantic types are non-numeric (i.e., the column values are all non-numeric). We restricted for this dataset the experiments to these 33 types that are present in the *Public BI* corpus and also supported by Sato, to have a dataset that represents a scenario where the semantic types are in principle supported by the pre-trained model,

TURL-Corpus [20]. As a second data lake we use the dataset from TURL [20]. *TURL-Corpus* uses the WikiTable corpus [9] as basis. To label each column they refer to the semantic types defined in the freebase ontology [34] with a total number of 255 different semantic types. What distinguishes this dataset from the *Public BI* corpus is that there are no matches with learned semantic types supported by the original Sato model. This means that we have a scenario with *TURL-Corpus* where the model has to be adapted not only to the new data characteristics but also to the new semantic types. Moreover, columns can have multiple semantic types in the original *TURL-Corpus*. To have a dataset that fits to our evaluation methodology (i.e., predict only one type per column), we manually selected the most specific semantic type out of the given semantic type set. To give an example, if one column has the labels *sports.sports_team* and *soccer.football_team*, we select the second semantic type as valid label because it is more specific.

Public BI Num [31]. To construct a data lake that comes with annotated numeric semantic types, we again used the *Public BI Benchmark* as a basis. Contrary to *Public BI*, we extended the semantic types by 19 additional numerically based types where the associated columns contain numeric values. Overall, this leads to a data lake with a significantly higher number of numerical columns and is therefore more comparable with real-world data lakes. With this data lake we are not only able to evaluate the performance of existing semantic typing models on a wide set of numerical semantic types and data, but also to evaluate our *Steered-Labeling* approach in depth and show its benefits.

SportsDB. As a fourth data lake we introduce a new corpus named *SportsDB*, which also contains a large fraction of numerical columns similar to real-world data lakes. We

constructed this corpus by extracting tables from different websites that publish statistics about football in recent years. For example, the corpus contains tables about statistics of football players in which *player name, goals, assists, games etc.* are listed. The extraction resulted in a corpus of 78 tables each containing an average of 3 textual and about 15 numerical columns. For annotating the columns with semantic types, we formed an ontology containing 18 different semantic terms (3 textual based types and 15 numerical based types) from the football domain. The assignment of the defined semantic types to the respective column was done semi-automatically using column headers and checked manually afterwards. When using this data set, we did not use the LF of *STEER*, which labels columns by headers.

3.4.2 Experimental Design

Setup. For the evaluation, the four datasets were split into three parts: labeled, unlabeled and test. The labeled split represents the set of table columns that we consider to be already labeled in the data lake. In the individual experimental setups, we apply different sizes of labeled data to make the results more comparable and show the impact of the quantity of already manually labeled data. For this, we decided to have 1 to 5 columns per semantic type already labeled as a starting point in our experiments.

To measure the performance of the different semantic typing models, we used a 20% split as test data. While creating the split, we first extracted the 20% test data and then used the remaining 80% to create the labeled and unlabeled set as described above. The unlabeled data was used as input to our *Steered-Labeling* framework to generate additional training data. To obtain statistically reliable results, we ran each experiment with five different random seeds and report the mean and standard deviation over multiple runs.
Experimental Structure. To demonstrate how well *STEER* can adapt and improve existing models to new data lakes, we have divided our evaluation in different use cases.

(1) STEER on Non-Numeric Data. In the first set of experiments, we evaluate *STEER* in combination with the existing models Sato and TURL that originally only support non-numerical semantic types on the *Public BI* and *TURL-Corpus* data set. With this experiment, we also show two scenarios: in the first scenario, we want to show the model adaption to data lakes which on the one hand contains types already seen by the model but with different data characteristics (i.e., Sato on *Public BI* and TURL on the *TURL-Corpus*). In the second scenario, we show how well *STEER* can be used to train a model on a data lake with new types the model has not seen before (Sato on *TURL-Corpus* and TURL on *Public BI*). Finally, in this set of experiments we demonstrate with this use case

the model independence of *STEER* by applying it to two different model architectures and model paradigms (Sato vs. TURL).

(2) STEER on Numeric Data. We evaluate *STEER* against data lakes which have a large proportion of numerical columns and numerically based semantic types (*Public BI Num* & *SportsDB*). Here we show the efficiency of our new *Steered-Labeling* approach by comparing a re-trained model with and without training data generated by *Steered-Labeling*.

(3) Ablation Study. In an ablation study we discuss and show the efficiency of individual LFs of our labeling framework and analyze the generated training data.

Baselines. In our experiments, we use several baselines to compare the efficiency of *STEER*.

(1) Sato baseline. We use the available learned Sato neural network called *Sato baseline* from [113] and applied it to the test data without any fine-tuning by re-training. With this, we want to see how well the existing model performs in new unseen data lakes with the same semantic types. Since only the *Public BI* dataset contains semantic types from Sato's learned model, this baseline is only used for this dataset.

(2) Sato retrain. As a second baseline we use *Sato retrain*. The idea is to use the set of existing labeled data (before applying our LFs) to re-train the Sato model. This experiment illustrates the effect of re-training an existing model with a small amount of manually labeled data. This baseline shows that the small set of existing labeled data is not sufficient to re-train a learned model and that our labeling framework *STEER*, which generates much larger training datasets, can significantly boost the performance.

(3) Turl retrain. As third baseline we use *Turl retrain*. Same as for *Sato retrain*, the manually labeled columns are used to fine-tune the pre-trained TURL model. It shows that even for the recent trend of pre-train/fine-tune models like TURL, a small existing set of labeled training data is not enough and the larger training data generated by *STEER* can significantly boost the performance.

Our Approach. To show the efficiency of *STEER* on Sato, we consider the same amount of manually labeled data to re-train *Sato retrain* and to fine-tune *Turl retrain* as a basis for generating more training data using our approach. Afterwards, we use the generated training data by our labeling framework to re-train the existing models (Sato and TURL) with this larger amount of data. The goal of this is to prove how our new approach and the additional generated training data can boost performance.

In order to report the benefits of *STEER* when using the model Sato or TURL, we fine-tune the pre-trained models with the larger amount of data generated by our labeling framework and name these model *STEER on Sato* and *STEER on Turl*. Finally, the

models are used on the test data split in order to demonstrate the benefits of using our approach in comparison to *Sato retrain* and *Turl retrain*.

3.4.3 *STEER* on Non-Numerical Data

In the following, we evaluate *STEER* on the two non-numerical data sets *Public BI* and the *TURL-corpus*.

3.4.3.1 *STEER* for Unseen Data Lakes

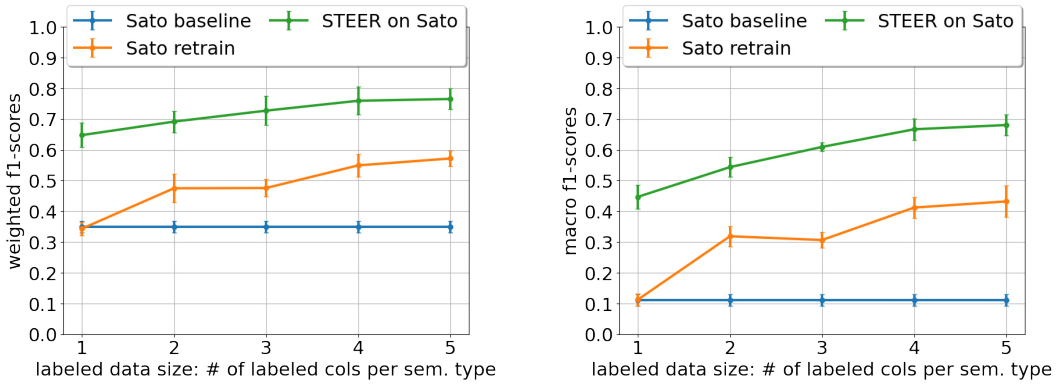
In the first experiment, we compare *STEER* against the baselines in two scenarios (known and unknown data types).

Scenario 1: Same Semantic Types. This section reports the overall results of using *STEER* in a scenario where the existing model already knows the semantic types of the new data lake from a previous training. To realize this scenario, Sato is used as model and the *Public BI* dataset as data lake.

Figure 3.3a and Figure 3.3b show the results reporting macro and support-weighted F1-Scores using the defined set-up as described before. First, we see that *Sato baseline* achieves only moderate F1-Scores, although the model supports all semantic types in the data lake. Secondly, as expected, the model *Sato retrain*, re-trained with the manually labeled data (but not with the generated training data by *STEER*), achieves better scores as *Sato baseline*. This shows the positive effect of adjusting the model to new data characteristics by re-training. This effect intensifies with the increasing amount of already labeled training data per semantic type. With the maximum size of 5 labeled columns per type, *Sato retrain* can achieve 0.43/0.57 (macro/support weighted) as average F1-Score.

Compared to *Sato retrain*, our re-trained model *STEER on Sato* outperforms the results by an average of +0.27/+0.24 (macro/weighted) F1-Score for each given size of labeled data. In total, *STEER on Sato* achieves an F1-Score of 0.681/0.765 (macro/weighted) and consequently achieves an improvement of 57.6%/34.2% over *Sato retrain* and 508%/119% to *Sato baseline*. Overall, this evaluation shows that our model *STEER on Sato* successfully optimized the adaption of the Sato model to the data lake *Public BI*.

Scenario 2: New Semantic Types. The next experiment is designed to show the results of *STEER* in a scenario where the learned model does not support the semantic types existing in the data lake. Thereby the model has to learn completely new types, which makes it more difficult to adapt the model because generally it will require a larger amount of training data. To perform the defined scenario, Sato is used as model and *TURL-Corpus* is used as data lake which comes with 105 new semantic data types. For



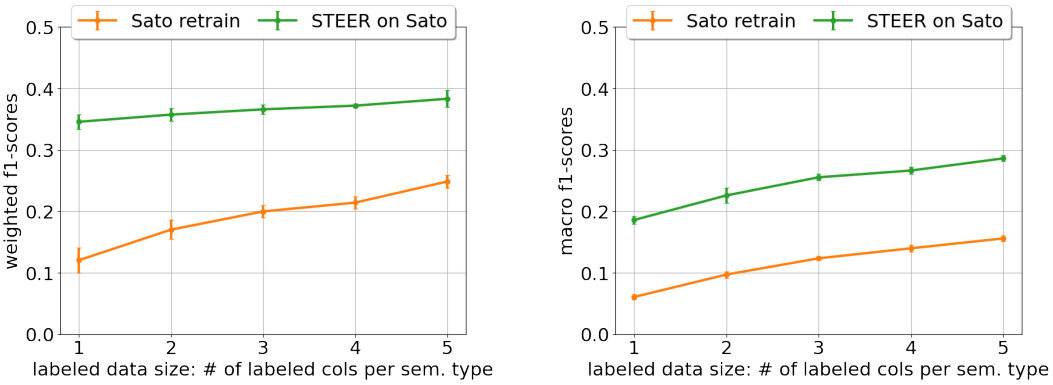
(a) Public BI (weighted F1)

(b) Public BI (macro F1)

Figure 3.3: Results on adapting Sato to the *Public BI* data lake to evaluate the scenario where the model is applied to a data lake containing already seen semantic types by the model. Our model *STEER on Sato* re-trained with the additional generated training data using *STEER* outperforms the baselines in every set-up.

this, we replaced the last layer of Sato to support 105 instead of the originally 78 types and initialized it with random weights.

Figure 3.4a and Figure 3.4b shows the results and compare our *STEER on Sato* model with the defined baselines. For *Sato retrain* we can see again that the F1-Scores of the model continuously improve. However, the gains are more moderate compared to *Public BI*. Looking at the results of *STEER on Sato*, we see a significant performance improvement in both macro and weighted F1-Scores compared to *Sato retrain*. We also see that *STEER on Sato* is able to constantly increase the model performance when having more labeled data. In other words, *STEER on Sato* is capable to efficiently use the generated additional training data for re-training. Since the dataset is overall more challenging (e.g., in diversity of values in columns with the same semantic types) and the model has to learn new semantic types, the F1 values are slightly lower overall than on *Public BI*. At labeled data size 5, *STEER on Turl* achieves best F1-Scores of 0.29/0.38. Note that the F1-Scores in our paper are lower than in the original paper [20] since they use multiple correct data types per column (which makes it easier for the model to at least predict one of them) and a large amount of manually labeled training data (i.e., 628,254 columns with manually annotated semantic labels) to fine-tune TURL, which we think is not realistic.



(a) TURL (weighted F1)

(b) TURL (macro F1)

Figure 3.4: Results on adapting Sato to the *TURL-Corpus* data lake to evaluate the scenario where the model is applied to a data lake containing completely new semantic types the model not seen before. *STEER on Sato*, which was re-trained with the generated training data of *STEER*, outperforms the baseline over all labeled data sizes.

3.4.3.2 Model Independence of STEER

The main goal of the following experiment is to show that *STEER* also can adapt and improve models with different architectures and paradigms. For this study, we additionally use TURL as model that is based on the idea of representation learning and is thus already pre-trained across a large corpus of tables. For the comparison of our approach *STEER on Turl* with the baseline *Turl retrain*, both non-numerical datasets (*Public BI* & *TURL-Corpus*) are used.

Figure 3.5a and Figure 3.5b show the results of the experiments by plotting the weighted F1-Scores separately for the two datasets. We also add the results from the previous experiment with Sato as model to see the differences of the two models on both data lakes.

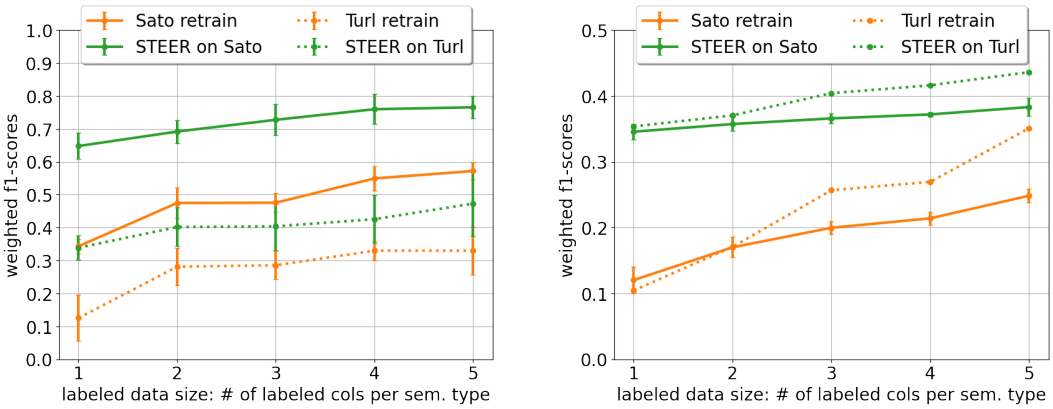
As a first aspect of the results, we can see that our model *STEER on Turl* is significantly better than the baseline *Turl retrain* in both datasets and across all labeled data sizes. Considering Figure 3.5a, *STEER* can achieve an improvement of +0.13 as average F1-Score over the labeled data sizes. In the experiments using *TURL-Corpus*, we achieve an average improvement of +0.25 F1-Score. This demonstrates that even for pre-trained/fine-tuned models, *STEER* can improve the performance with the additional generated training data during the fine-tuning. That is especially remarkable, as it is assumed that pre-trained models need fewer training data samples during fine-tuning.

Table 3.2: Average textual and numerical columns in real world tables, showing the aspect that such tables have a high proportion of columns containing numerical values.

Corpus	Domain	#Avg. Textual Cols	#Avg. Numerical Cols
Public BI Num	Sport	6.4	48.5
Public BI Num	Medicare	14.7	13.0
Public BI Num	Real Estate	14.2	22.4
Public BI Num	Government	34.0	22.0
Public BI Num	Geography	24.2	9.0
SportsDB	Football	3.0	14.83

Furthermore, despite the fact that the model was pre-trained on the *TURL-Corpus*, fine-tuning the model with the additional training data provided by *STEER* leads to performance gains. In total, *STEER on Turl* achieves the best F1-Scores at labeled data size 5 of 0.47 on *Public BI* and 0.44[5] on *TURL-Corpus*. It is important to note, that we performed another test on the *Public BI* corpus with the maximum training data size of 80%, which results in a model that achieves a performance of 0.56 weighted F1-Score. Compared to the score values of *STEER on Turl*, the gap to this theoretical maximum reachable score is remarkable.

An additional interesting detail of the experiment results is the comparison between Sato and TURL model on both data lakes. Note that when using *Public BI*, Sato already knows the semantic types (Scenario 1), while TURL has not seen the data at all. This fact is exactly vice versa when using *TURL-Corpus*. The TURL model has already seen the data during pre-training whereas Sato has not seen the data and has to learn new types (Scenario 2). When comparing *STEER on Sato* to *STEER on Turl* with *Public BI* as data lake, we can see in Figure 3.5a that *STEER on Sato* achieves a much higher F1-Score than *STEER on Turl*. Even *Sato retrain* re-trained with a much smaller amount of training data is better than *STEER on Turl*, which mainly shows the advantage of having a model pre-trained on the data. Consider the results on the *TURL-Corpus* demonstrated in Figure 3.5b, we can also see this advantage but this time for TURL; i.e., *STEER on Turl* achieves better results than *STEER on Sato*.



(a) Public BI (weighted F1)

(b) TURL (weighted F1)

Figure 3.5: Results on adapting TURL model to the *Public BI* and the *TURL-Corpus* to show that even for pre-trained/fine-tuned models, *STEER* and its generated training data lead to performance gains. We added the results of Sato from the previous experiment for comparison.

3.4.4 *STEER* on Numerical Data

Real-world data lakes often contain a significant amount of numerical columns. This fact is also shown in Table 3.2 which shows the distribution of numeric and no-numeric columns in our two corpora where we annotated numeric types. As mentioned above, extracting semantic types from numeric values is more challenging because of the very low entropy. In order to generate training data with *STEER* for numeric columns with a high quality, we have implemented our *Steered-Labeling* approach as described in Section 3.1. The experiments in this section evaluate *STEER* the two data lakes that contain numerical columns.

3.4.4.1 Efficiency of Steered-Labeling

In the following experiments we use *Public BI Num* and *SportsDB* as data lake and Sato as existing model. Like before we compare the results with the baseline *Sato retrain*, which is the model re-trained with a small amount of training data coming from the data split and the described different labeled data sizes. To show the benefits of our *Steered-Labeling* approach we introduce another baseline *STEER on Sato no steer*, which is the

[5] [20] applied TURL model on *TURL-Corpus* and reported 0.9475 as F1-Score. This was achieved in the paper by fine-tune the model with 80% training data. Notice that in our set-up we fine-tune TURL only with about 14.5% training data, leading to the large discrepancies between the two reported F1-Scores.

model re-trained with training data generated without the usage of our *Steered-Labeling* approach; i.e., we do not label textual and numeric labeling sequentially but we instead use the textual LFs also for numeric data. By contrast, the model *STEER on Sato* is re-trained with training data generated by *Steered-Labeling* and thus demonstrates the gains of our new approach.

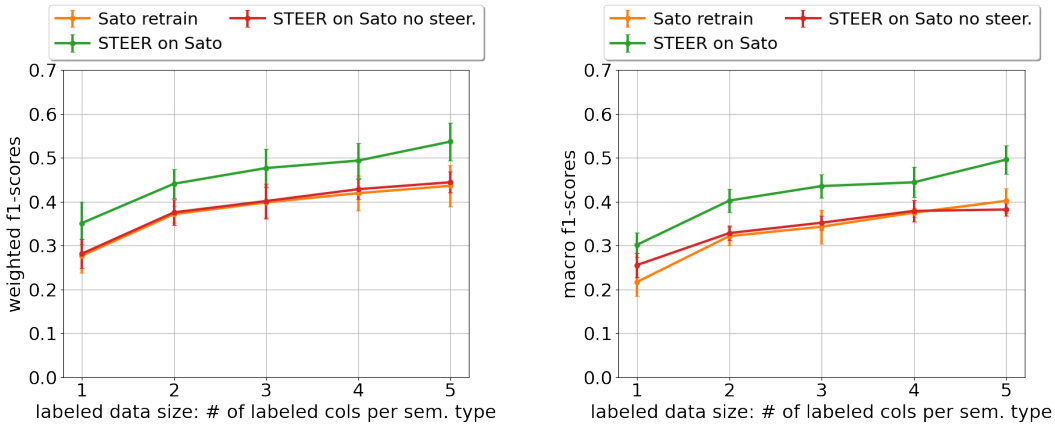
Figure 3.6 shows the results of the different models *Sato retrain*, *STEER on Sato* and *STEER on Sato no steer* on the *Public BI Num* dataset. Due to the use of *Steered-Labeling* *STEER on Sato* outperforms both baselines in macro and weighted F1-Score on all manually labeled data sizes. Compared with *STEER on Sato no steer*, the score values confirm that *Steered-Labeling* increases the quality of the generated training data and finally leads to a better end model. At labeled data size 5, *STEER on Sato* reaches best scores with 0.496/0.537 F1-Score. Considering the score values of *STEER on Sato* on *Public BI*, which contains the same textual data but not the number of numerical data, we see a drop of -0.184/-0.228 F1-Score. In addition, the results show an almost identical performance of the models *Sato retrain* and *STEER on Sato no steer*.

On Figure 3.7 we demonstrate the macro and weighted F1-Scores on the *SportsDB* corpus. The results show that our model *STEER on Sato* with *Steered-Labeling* outscores the baselines on every labeled data size. At best, *STEER on Sato* reaches 0.77/0.87 F1-Score resulting in an increase of +0.167/+0.155 in comparison to *STEER on Sato no steer*. Overall, these results also show that the *Steered-Labeling* method has significant advantages in generating the training data than without *Steered-Labeling*. One additional detail to be mentioned in the results is the comparison in the set-up with a labeled data size 1 (amount of existing labeled data). Here *STEER on Sato* outperforms *Sato retrain* by +0.632/+0.705 F1-Score, demonstrating that *STEER* can generate a large amount of good training data with a small amount of already labeled data as a basis.

Discussion. In a more detailed analysis (not included in this paper), we observed that the numerically semantic types are poorly predicted by *STEER on Sato*, even though the generated training data quality is quite adequate (see Figure 3.9c). In the future, we thus plan to work on model architectures that are more tailored towards detecting numerical data types.

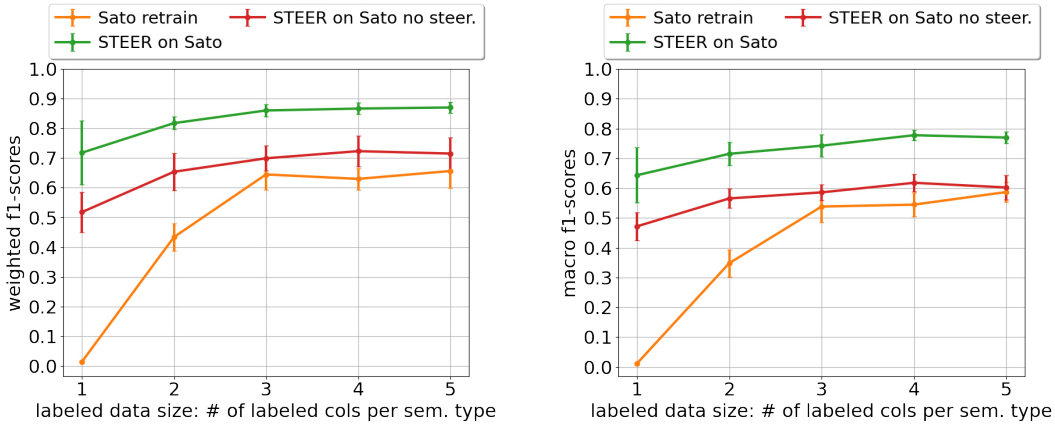
3.4.4.2 Optimization for Steering

As described in Section 3.1.2, *STEER* strictly separates the LFs into those for labeling non-numerical and numerical columns to execute first the non-numerical and then the numerical LFs. The main idea behind *Steered-Labeling* is that based on the generated



(a) Public BI Num (weighted F1) (b) Public BI Num (macro F1)

Figure 3.6: Results on adapting Sato to the *Public BI Num* data lake using our *Steered-Labeling* generated training data. In addition to the comparison of *Sato retrain* (baseline trained with initial already labeled data available) we compare also to a model which was built with generated training data without the usage of *Steered-Labeling*.



(a) SportsDB (weighted F1) (b) SportsDB (macro F1)

Figure 3.7: Results on adapting Sato to the *SportsDB* data lake using training data generated by *Steered-Labeling*. In addition to the comparison of *Sato retrain* (baseline trained with initial already labeled data available) we compare also to a model which was built with generated training data without the usage of *Steered-Labeling*.

Labeled Columns per Data Type	Reduction of Labeling Runtime
1	25.5%
2	34.5%
3	39.2%
4	44.9%
5	49.6%

Table 3.3: Runtime gains on Public BI Num using context informations from the *Steered-Labeling* process for preselecting relevant columns for the LF *EMD*

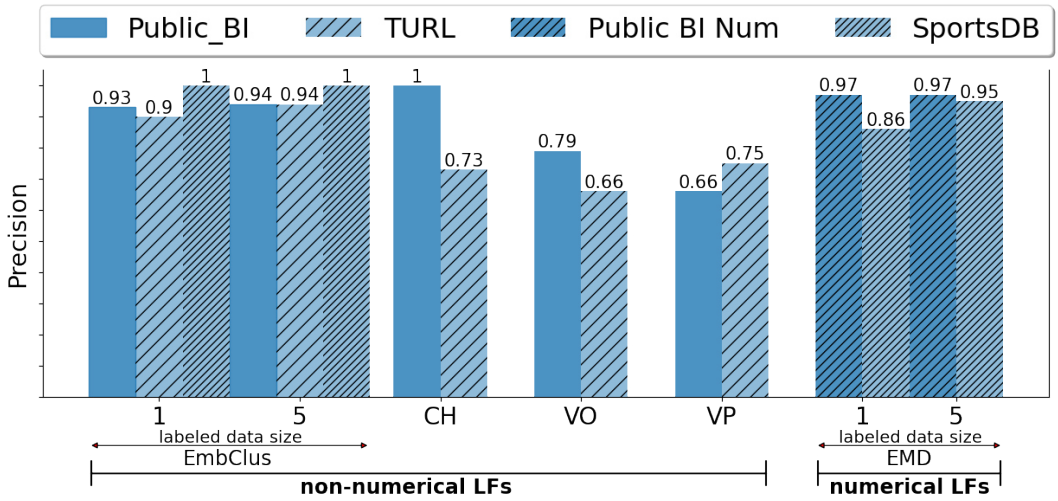
labels in step one, the numerical LFs can benefit from this additional information during the labeling. In the previous experiment, we demonstrated the accuracy gains (reported by F1-Scores) in the adapted end model by re-trained one model with (*STEER on Sato*) and another model (*STEER on Sato no steer*) without steered generated training data. However, *Steered-Labeling* does not only lead to more accurately generated training data but we can also use the context to reduce the runtime of the overall labeling process in step 2 as described in Section 3.2. To show these possible runtime gains from steering, we conducted an experiment in which our LF EMD is executed in two different modes: (1) without the preselection of contextually similar numerical labeled columns and (2) with a selection. To be more precise in (1) the LF measures the similarity against all labeled numerical columns available and in (2) only against those ones which are embedded in the same context (table with semantic equally neighbored textual columns). Table 3.3 shows the results of these experiments by listing the runtime reduction that could be achieved per labeled data size. The reduction goes from about 25% (labeled data size 1) to almost 50% (labeled data size 5), demonstrating the higher the number of labeled data the higher the percentage of runtime reduction.

3.4.5 Ablation Study

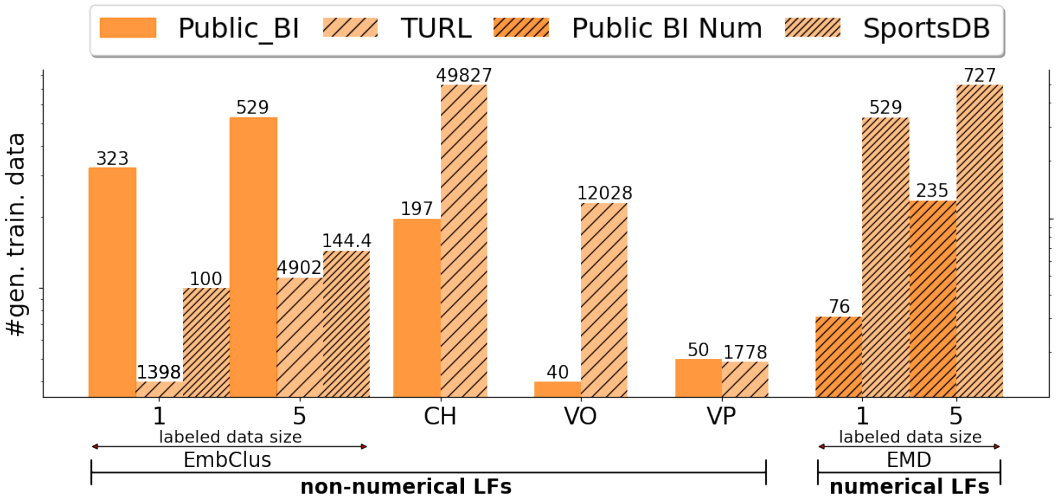
In the ablation study, we analyze the quality and quantity of generated training data for each class of LFs.

3.4.5.1 Efficiency of the LFs

To evaluate the contribution of each LF to the total amount of training data generated, we analyzed the generated training data per LF separately in an ablation study. The results per LF are shown in Figure 3.8 while Figure 3.8a shows the quality (weighted



(a) Quality of generated training data per LF class



(b) Quantity of generated training data per LF class

Figure 3.8: Quality and quantity of generated training data for each class of LFs. (a) Each LF of *STEER* (non-numerical as well as numerical) creates high-quality training data. (b) Moreover, each LF contributes to the amount of generated training data. However, the number of generated training samples (columns and their semantic type) varies per data set and LF.

precision) and Figure 3.8b the quantity (number of labeled columns) for all datasets separately.

LF *EmbClus*. For the LF *EmbClus* we plot the results for the labeled data sizes 1 & 5 showing the aspect that the larger the number of already labeled data, the more new unlabeled data can be labeled with additional small quality improvement. This is due to the fact that more labeled data is included in the clustering and thus the labeling process is extended and improved. At labeled data size of 5, the LF produces 529 (96% of the total gen. train. data) for *Public BI*, 4.902 (6.4% of the total gen. train. data) for *Turl-Corpus* and 529 (16.5% of the total gen. train. data) for *SportsDB*. If we look at the quality of the generated training data, we achieve an average precision of 0.94 for the datasets *Public BI* & *TURL-Corpus* and a perfect result of 1.0 for *SportsDB*. Notice that for this LF we do not list results for *Pubic BI Num*, because the same textual columns are included as in *Public BI* and therefore the results are the same. In summary, this demonstrates that this LF generates high quality labeled training data to adapt the model.

LF *CH* (column-headers). Focusing now on the LF *CH*, the quality of the generated training data for *Public BI* is overall high. Since the precision is at the score of 1 each label assigned by the LF to the 197 unlabeled columns is correct. For *Turl-Corpus*, the LF produces over 65% of all additional training data generated. The precision is also high at 0.73. The reason in comparison to the quality on *Public BI* is that frequently occurring types are mislabeled. For example, for the semantic types *music.album* & *music.artist* the LF generates over 5.000 new labeled columns with a precision over 0.94, but for the types *music.genre* & *music.composition* just 80 with a precision below 0.2 since column headers are too general. In conclusion, this LF works (almost) perfectly for the *Public BI* corpus and the model can benefit from the new training data when re-training. However, also for the *TURL-Corpus*, which is more complex, we can generate a good number of high-quality labels; i.e., only a small amount of training data with low quality is generated for semantic types (macro score), which may be also improved or resolved when merging the outcomes of the different LFs.[6]

LF *VO* (value-overlap). As described in Section 3.3, for the class of LF *VO*, a domain expert can provide a list of common values to a semantic type, which is then used by the LF to generate the training data. In our current prototype version, we provide such a list for two selected semantic types for the *Public BI* data corpus and 11 semantic types for the *Turl-Corpus*. As an example, for the *Public BI* semantic type *language*, we provide a list with the values *{de, en, fr, es, ...}*. In case of the *Turl-Corpus* semantic type

[6]Notice that we do not implement LFs of this class for the *Public BI Num* & *SportsDB* datasets.

film.flim_genre, the domain expert defines the common values *{crime, horror, romance, action, ...}*. For each selected type, a LF of this class is instantiated and executed on its own. In order to show the precision and the number of generated training data, we have averaged the precision values and summed up the number of generated training data over the individual outcomes per semantic type. For both *Public BI* types the LFs generate in total 40 new labeled columns with a precision of 0.79. In addition, we can see that for the semantic types belonging to the *Turl-Corpus*, we can achieve an average precision of 0.66 and generate over 12.000 new labeled columns. To give an insight into one of the best LFs here, the LF for *soccer.football_team* can label 7.238 columns with a precision of 0.98 by providing a short list of the most famous football teams.[7]

LF *VP* (value-pattern). Similar to the previously discussed LF, even in this case the user provides a list of patterns in form of regular expressions that are then used in the labeling process. In our current implementation, we defined such patterns for two *Public BI* types and eight *Turl-Corpus* types. To give an example, for the semantic type *award.award_category*, we define the list of patterns as follows {best*, worst*}. Meaning that all values starting with *best* or *worst* are counted as a pattern match. In case of the *Public BI* dataset, the type specific LFs can label 50 unlabeled columns with an overall precision of 0.66. If we analyze the LFs for *Turl-Corpus*, we create almost 1.800 new labeled columns with a slightly higher precision of 0.75. In this class of LF the pattern definitions are very important. The defined patterns should not be under-generalized (e.g. resulting in too infrequent matches and therefore the columns related to the semantic type are not found) and also not be over-generalized (e.g. resulting in too many matches for columns that actually do not belong to the semantic type) [43].

LF EMD. For the numerical LF *EMD* we plot the results for the labeled data sizes 1 & 5. The figure shows the precision and the number of generated training data for *Public BI Num* and *SportsDB*, since these are the only datasets that contains numerical columns and therefore *Steered-Labeling* with the LF *EMD* was applied. Looking at the weighted precision of the generated training data, we can see a constant value of 0.97 on *Public BI Num*, whereas on *SportsDB* there is an increase from 0.86 to 0.95. This improvement comes from the fact that the LF benefits from a higher amount of existing already labeled columns because more similarity measurements can be made from unlabeled to labeled numerical column and thus the precision of matches increases. Overall the precision values on both datasets demonstrates that the LF *EMD* extracts training data with a very high quality. Looking at the number of generated training data we see on *Public BI Num* values from 76 to 529 and on *SportsDB* values from 529 to 727 over the labeled

[7] Notice that we do not implement LFs of this class for the *Public BI Num* & *SportsDB* datasets.

data sizes 1 to 5. This increase comes for the same reason as just mentioned. The higher the amount of already labeled data the higher the probability to find a semantic match. Overall these results demonstrate that the LF *EMD* can annotate unlabeled numerical data in a good manner and therefore make a high contribution to the generated training data that can be used to adapt a model to a data lake containing high numbers of numerical data.

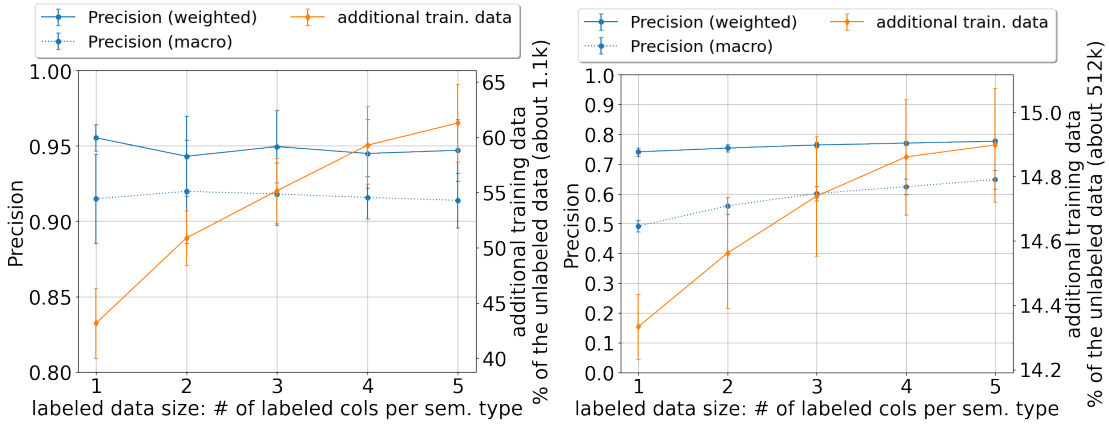
Summary. The quantity and quality of the generated training data are high for all used fictive data lakes. Consequently, as we have seen in the experiment before, the new training data can lead to a significant improvement of a learned metadata annotation model after re-training.

3.4.5.2 Analyze Generated Training Data

To better understand the gains of *STEER*, we now analyze the overall generated training data which was aggregated across the LFs by applying the majority vote. The aggregated training data represents the generated training data of *STEER* that we use for re-training the models. As in the previous section, we focus on two aspects for the analysis: quality and quantity of the generated training data to show that a significant amount of data is generated which provides high-quality (i.e. correct) labels.

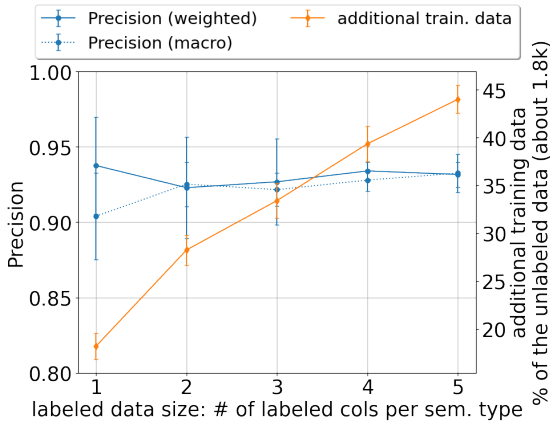
The results of this analysis for all data sets are shown in Figure 3.9. Since only *Public BI Num* and *SportsDB* are containing labeled numerical data columns, *Steered-Labeling* training data generation is only applied to these datasets. To report the quality of the generated training data, we plot the macro and support weighted precision (i.e., the fraction for which the LFs assign the correct type). For showing the quantity, we plot the number of table columns, which receive a label from the labeling framework and thereby resulting as additional training data in percentage to the total amount of unlabeled columns available.

Overall, we can see that for the datasets *Public BI*, *Public BI Num* and *SportsDB* the generated training data achieves very high weighted precision values of more than 0.9. With *SportsDB*, we even reach a quality of 0.93/0.96 (macro/weighted). Looking at the generated training data quality of *TURL-Corpus*, we see lower but still good precisions of 0.6/0.75, which also leads to benefits when it is used as additional training data as shown before (see Figure 3.4). Considering the amount of the generated training data, we achieve the highest value on *SportsDB* of up to 80% of the available unlabeled data while for *TURL-Corpus* we only label about 15%. It is important to note, however, that the *TURL-Corpus* is significantly larger than all other corpora and in absolute

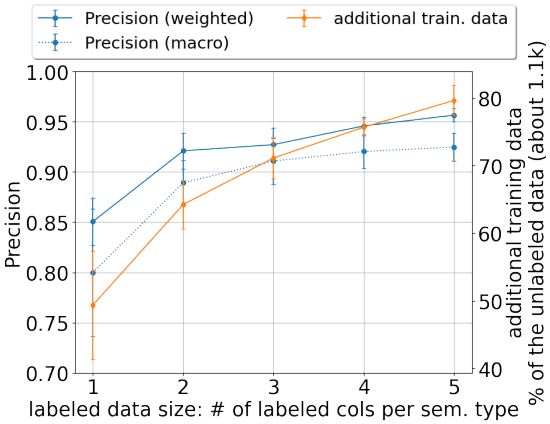


(a) Public BI

(b) Turl



(c) Public BI Num (*STEER*)



(d) SportsDB (*STEER*)

Figure 3.9: Quality (blue) and quantity (orange) of generated training data. Quality is reported with macro and weighted precision. Quantity is shown as the number of additional generated training data in percentage to the total amount of unlabeled columns available.

quantity we generate the largest corpus of additional training data with *STEER* on the *TURL-Corpus*.

3.5 Summary

Re-training a learned semantic type detection model in order to adapt it to a new, unseen data lake imposes a significant overhead. This is due to the necessary generation of new training data that covers the new semantic types and data characteristics of the environment in which the model is to be used. Because this overhead is a barrier to applying these models, we have introduced *STEER*, our data programming framework for semantic type labeling of table columns in this chapter. *STEER* uses the idea of weak supervision and comes with several integrated LFs to automatically generate new training data from a data lake to which a learned semantic type detection model is supposed to be applied. With the generated training data, *STEER* is able to re-train/fine-tune existing learned models to new data lakes with minimal overhead. To generate high quality training data not only for non-numerical but also for numerical table columns, *STEER* has integrated our novel training data generation procedure called *Steered-Labeling* as a core component. The central idea of *Steered-Labeling* is the separation of the labeling process and the execution of the LFs into two sequential steps. In the first step, the non-numerical columns are labeled with implemented LFs for non-numeric data. Afterward, *STEER* then executes the numerical LFs, which uses the previously generated labels of the non-numeric columns as additional information to label the numerical columns. With this, the LFs for numerical columns achieves a much higher accuracy. Through experiments conducted across multiple data lakes using two distinct learned models, we demonstrated in this chapter that *STEER* has the capability to adapt learned models to new, previously unseen data lakes. By comparing training data generated with and without the *Steered-Labeling* approach, we also showed the advantages of the two-step labeling process.

While searching for suitable publicly available data lakes to evaluate our *Steered-Labeling* approach, we noticed that there are no adequate data corpora for this purpose. All existing datasets containing tables with semantically labeled columns consist either purely of textual data or contain a very minimal proportion of numerical data. For this reason, we have presented the two datasets *Publi BI Num* and *SportDB*, which include a high fraction of numerical table columns. *Publi BI Num* was an existing corpus that we labeled with semantic types to make it usable for the evaluation. For *SportsDB*, we

created the corpus from scratch by extracting tables of soccer statistics from various websites as mentioned before. When using these two data lakes in our experiments, we noticed that the two adapted end models (Sato & Turl) performed worse on numerical columns than on non-numerical columns. In our opinion, this is mainly because the models were designed, trained and tested with datasets consisting only of non-numerical data. Although with *Public BI Num* and *SportsDB* we now have datasets for semantic type detection that could be used for the design of a new model that better supports numerical columns, the two datasets are too small for this purpose. This motivates the further contribution of this dissertation in creating a new corpus for the task of semantic type detection, which is characterized by a significantly higher proportion of numerical columns compared to existing corpora, as we will discuss in the next chapter.

4 SportsTables: The Missing Labeled Numerical Corpus

Publication. The work on building a new data corpus that contain a high proportion of tables with numerical columns is published in the peer-reviewed publications "SportsTables: A new Corpus for Semantic Type Detection" in the *Datenbanksysteme für Business, Technologie und Web (BTW 2023), 20. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme" (DBIS), 06.-10, März 2023, Dresden, Germany, Proceedings* & "SportsTables: A New Corpus for Semantic Type Detection (Extended Version)" in the *Datenbank-Spektrum*.

Contributions of the author. Sven Langenecker is the leading author of the publications [59] & [60] mentioned above. He is responsible for the analysis of the existing corpora for the task of semantic type detection of table columns, the built corpus SportsTables, the whole implementation of the pipeline to build the corpus, the experimental evaluation and the manuscript. The co-authors Christoph Sturm, Christian Schalles and Carsten Binnig contributed invaluable feedback. All authors agree with the use of the publications for this dissertation.

In the previous chapter, we introduced the *SportsDB* corpus to evaluate our *Steered-Labeling* procedure and show the advantages of this approach when generating new training data of numerical columns. However, the *SportsDB* corpus only consists of a limited number of table columns. To create and test new models, which especially can better extract the semantic type of numerical columns, a larger data corpus is required that comprises tables with a very high proportion of columns with numeric values. This chapter addresses this challenge and presents the creation of a new corpus that builds upon the previously mentioned *SportsDB* corpus. To this end, we first explain below why the creation of our new corpus is important and needed. Afterward, in Section 4.2 we give a detailed overview of existing corpora which was used to build and validate semantic type prediction models and discuss their main characteristics and statistics. Section 4.3 then introduce our new corpus SportsTables and describe in detail how we created the corpus and labeled the table columns with semantic types. Next, in Section 4.4, we demonstrate the main characteristics and statistics of our corpus. To show that all existing semantic type detection models fall short on numerical columns, we show in Section 4.5 results

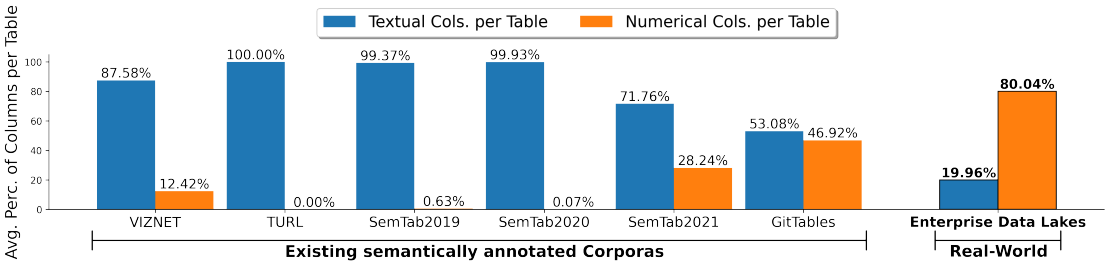


Figure 4.1: Average percentage of textual and numerical based columns per table in existing semantically annotated corpora[1] (left bars) compared to real-world data lakes (right bar). This shows the fact that there is a significant shift in the ratio of textual to numeric columns per table from existing corpora to real data lakes. Since all existing semantic type detection models were developed by using the existing corpora, shortcomings in validating the models on numerical data are present and it has not yet been studied in depth how well the models can perform on datasets containing a high proportion of numerical data.

of using our new corpus on different existing models. Section 4.6 summarizes the key insights of this chapter and explains the motivation for the final contribution in this dissertation.

4.1 The Need for a New Corpus

To build a semantic type detection model, many recent approaches rely on deep learning techniques. Consequently, corpora containing large amounts of table data with assigned semantic types are required for training and validating these models. Notably, as illustrated in Figure 4.1, almost all existing corpora providing annotated columns labeled with semantic types have a lack of table columns that contain numerical data. These datasets predominantly feature tables that incorporate either only or a very high percentage of textual data. Only GitTables [47] comprises a more balanced ratio of textual and numerical data. Nevertheless, compared to real enterprise data lakes, there is a significant discrepancy in the ratio of textual to numerical columns. An inspection of a large real-world data lake at a company[2] has shown that on average approx. 20% textual data and 80% numerical data are present (see. Figure 4.1 bars on the right).

Moreover, semantic type detection models [20, 50, 100, 113] that are trained on the available corpora also mainly target non-numerical data.

Detecting semantic types of numerical columns is generally harder than for textual columns. For example, for a textual column with the values {Germany, USA, Sweden, ...} a model can easily identify the semantic type *country*. Instead, for a numeric column with e.g. the values {20, 22, 30, 34,...} it is not that straightforward and several possibilities for a matching semantic type exist such as *age, temperature, size, money*. The fundamental reason here is that numerical values can be encoded with much fewer bits than string values [98], resulting in a lower overall entropy and thus providing less information content that can be used by a ML model to infer the underlying semantic type. The limitations of existing corpora used for building and validating semantic type detection models have revealed several crucial shortcomings. These issues remained unaddressed due to the lack of an adequate dataset specifically designed for the purpose of detecting the semantic type of numerical columns. We thus contribute a new corpus containing tables with numeric and non-numeric semantically annotated columns that reflect the distribution of real-world data lakes. Before we introduce and describe how we created our new corpus SportsTables, in the following, we first provide an overview of existing corpora which was used to build and validate semantic type detection models and describe their characteristics.

4.2 Existing Corpora: Dominated by Textual Data

In the following, we describe different existing corpora that contain annotated table columns and therefore can be used to build and validate semantic column type detection models. We summarized the main statistics for all corpora in Table 4.1.

VizNet [45]. The original VizNet corpus [45] is a collection of data tables from diverse web sources ([11, 81, 88, 105]) which initially do not contain any semantic label annotation. The corpus we consider in this paper is a subset of the original VizNet corpus, which was annotated by a set of mapping rules from column headers to semantic types and then used to build and validate the Sherlock [50] and Sato[113] prediction models. The corpus contains in total 78,733 tables and 120,609 columns annotated with 78 unique semantic

[1]Notice that for GitTables we only considered the tables and columns labeled by terms from DBpedia using the semantic annotation method as described in the GitTables paper. Therefore our reported ratios of textual and numerical data differ from those shown in the GitTables paper because they consider all data, whether annotated or not.

[2]The analyses were done at the company LÄPPLE AG

Table 4.1: Corpus statistics about the number and sizes of tables. Additionally, we see the average number of textual and numerical columns per table for each existing annotated corpora and our new SportsTables corpus. This shows the absence of numerical data columns per table in most existing corpora and the dominance of textual data columns per table in all existing corpora. Instead, our new corpus SportsTables contains on average over 6 times more numerical columns than textual columns.

Corpus	Tables	Cols	$\overline{Cols/Table}$	$\overline{Text.\ Cols/Table}$	$\overline{Num.\ Cols/Table}$	$\overline{Rows/Table}$
VIZNET	78,733	120,609	1.53	1.34	0.19	18.35
TURL	406,706	654,670	1.61	1.61	0	12.79
SemTab2019	13,765	21,682	1.58	1.57	0.01	35.61
SemTab2020	131,253	190,494	1.45	1.45	0.001	9,19
SemTab2021	795	3,072	3.86	2.77	1.09	874.6
GitTables	1.37M	9.3M	6.82	3.62	3.2	184.66
SportsTables	**1,187**	**24,838**	**20.93**	**2.83**	**18.1**	**246.72**

types. Overall, the tables in the corpus contain only 1.53 columns and 18.35 rows on average. Furthermore, the distribution of the column data types is 87.58% textual and 12.42% numerical and thus leads to the shortcomings as described before.

TURL [20]. The TURL corpus uses the WikiTable corpus [9] as basis. To label each column they refer to the semantic types defined in the Freebase ontology [34] with a total number of 255 different semantic types. What distinguishes TURL from other corpora is that columns can have multiple semantic types assigned. In total, there are 406,706 tables resulting in 654,670 columns and on average a table consists of 1.61 columns and 12.79 rows. Again, these are rather small dimensions. In addition, the Turl corpus includes no numerical data at all, which leads to the shortcomings when using the corpora as mentioned above.

SemTab. SemTab is a yearly challenge with the goal of benchmarking systems that match tabular data to knowledge graphs since 2019. The challenge includes the tasks of assigning a semantic type to a column, matching a cell to an entity and assigning a property to the relationship between columns. Every year, the challenge provides different datasets to validate the participating systems against each other. In this paper we observed the provided corpora for the years 2019 [40], 2020 [19, 41], and 2021 [1, 19, 42, 46, 82]. Statistic details of the corpora are shown in Table 4.1. In case more than one dataset was provided per year, we aggregated the statistics over all datasets included in the challenge. While SemTab2019 consists of 13,765 tables and 21,682 columns in total, there are 131,253 tables and 190,494 columns in SemTab2020. In both corpora, the dimensions of the included tables are rather small (on average 1.58 columns and 35.61 rows in 2019 and 1.45 columns and 9.19 rows in 2020). In SemTab2021, the contained

tables are the largest in terms of rows with almost 875 on average. However, the number of columns (3.86 on average) is only moderate and the corpus in general is the smallest with a total of 795 tables and 3,072 columns. Numerical data is almost nonexistent in the first two years (0.63% in 2019 / 0.07% in 2020), increasing to 28.24% numeric columns per table on average in 2021, which is still not comparable to the number of numeric data in real world data lakes.

GitTables [47]. GitTables is a large-scale corpus of relational tables created by extracting CSV files from GitHub repositories. Table columns are labeled with semantic types from Schema.org [35] and DBpedia [6] using two different automated annotation methods (syntactically/semantically similarity matching from semantic type to column header). In this paper, we have focused on the annotations origin from DBpedia and the results of the semantic annotations method as described in the GitTables paper [47]. This leads to a corpus containing over 1.37M tables and 9.3M columns in total. Although this is by far the largest collection of data tables, the dimensions of the tables are on average only moderate with 6.82 columns and 184.66 rows. Overall, GitTables incorporates the most numeric data with an almost balanced ratio of 53.08% textual and 46.92% numerical columns per table.

Discussion. The overview in Table 4.1 and the discussion before shows that most existing corpora contain no or only a minimal fraction of numerical data types which is very different from real-world data lakes. An exception is GitTables which has a much higher ratio of numerical columns. However, as we show in Section 4.4, GitTables still lacks a good coverage of different numeric semantic types which is one important aspect that we tackle with our new corpus SportsTables which covers a wide variety of different numerical semantic types. Moreover, another important (but orthogonal) aspect is that existing corpora include a large number of tables. However, on average the tables are very small in terms of the number of columns and the number of rows. Instead, our new corpus SportsTables contains fewer tables, but on average a significantly higher number of columns and rows per table to better reflect the characteristics of real-world data lakes.

4.3 The SportsTables Corpus

In the following, we will introduce our new corpus and describe in detail the implemented construction pipeline to build SportsTables.

Methodology to generate the corpus. Figure 4.2 gives an overview of our implemented pipeline to generate the new corpus. The main idea was to collect data tables

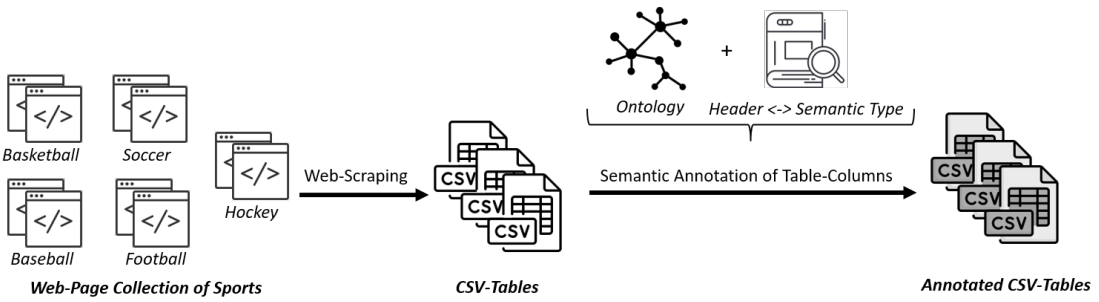


Figure 4.2: Overview of the implemented pipeline to build SportsTables. We use web-scraping techniques to extract HTML tables from a manually defined web page collection for each selected sport and convert the tables to CSV files. With the help of a defined ontology and a manually created dictionary that maps column headers to semantic types, we annotate each table column with an appropriate semantic type.

from different sports domains such as soccer, basketball, baseball, etc. since data tables coming from such kinds of sources are rich in numerical columns. For example, a soccer player statistic table of a soccer season contains typically 3 textual columns (e.g., player name, team name, field position) and 18 numerical columns (e.g., goals, games played, assists). Hence, building a collection of such tables will lead to a corpus that contains many numerical columns which are in addition semantically interpretable. As a result, the corpus will enable performance analysis of semantic type prediction models in a much more rigorous manner regarding numerical data.

Scraping data from the web [23]. A vast amount of data covering information about player statistics, team statistics, coach statistics or season rankings of different sports are available on various web pages. Therefore, for collecting the data, we built a data collection pipeline based on web scraping technology[23]. In the first step, we manually searched and defined a set of different web pages for each of the selected sports of which we want to scrape contained data tables (left side of Figure 4.2). We first converted each HTML table on the web pages to Pandas-Dataframes using Python and then saved them as CSV files (center of Figure 4.2), since this file format is most known and used to store raw structured data [76]. During the scrape process, we kept the respective column headers from the original HTML table and used them as headers in the CSV file.

Annotating columns with semantic types. Due to the low granularity of existing ontologies (e.g. DBpedia) regarding semantics of a given sport, we manually created an ontology-like set of valid semantic types for all sports. For example, in DBpedia there is the type *Person.Athlete.BasketballPlayer*, but semantic labels in the particular that

Table 4.2: Statistics about the number of unique semantic types. Showing that our new corpus has a higher proportion of numerical semantic types than textual semantic types in contrast to the existing corpora. In addition, there is a large overlap of semantic types used for textual and numeric columns in the existing corpora. In comparison, the semantic types in SportsTables are disjoint for the two column data types.

Corpus	#Textual Sem. Types	#Numerical Sem. Type	#Total Sem. Types
VIZNET	78	44	78
TURL	255	0	255
SemTab2019	360	19	360
SemTab2020	5804	32	5832
SemTab2021	177	93	251
GitTables	2646	2426	2693
SportsTables	**56**	**419**	**475**

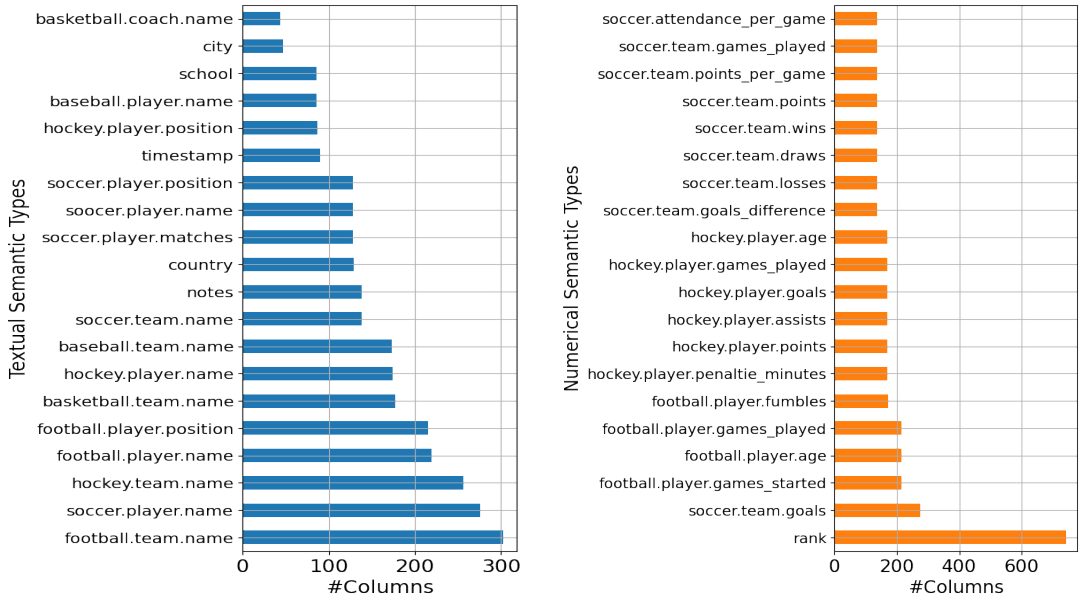
would match individual numerical columns such as *NumberOfGoals* are not defined. Next, we annotated all table columns with semantic types using a manually created dictionary that maps column headers to matching semantic types from our created set. Since the column headings were in many cases identical if the semantic content was the same, this procedure significantly reduces the manual labeling effort. In addition, to ensure that the labels are of very high quality in terms of correctness, we manually checked each assignment based on the content of the columns.

4.4 Corpus Characteristics

In the following. we discuss the statistics of the SportsTables corpus and compare them to the existing corpora.

Data statistics (Table 4.1). Using the described pipeline for creating SportsTables, a total of 1,187 tables which comprises 24,838 columns (approx. 86% numeric and 14% textual) are scraped from the web resulting in 20.93 columns (2.83 textual and 18.1 numerical) per table on average. This ratio of textual to numerical columns, as well as the total average number of columns in a table, differs significantly from existing corpora.

In Table 4.1 we can also see a comparison of the average number of textual and numerical columns per table of SportsTables versus that of the existing corpora. Here we can see that numerical columns only exist in the corpora VizNet with 0.33, SemTab2021 with 1.09, and GitTables with 3.2 columns per table. Compared to GitTables, in SportsTables there are thus on average over 6 times more numeric columns per table. Moreover, as



(a) Top 20 textual sem. types

(b) Top 20 numerical sem. types

Figure 4.3: Semantic type annotation statistics of SportsTables. (a) Shows column annotation counts of the top 20 textual semantic types. Across all kinds of sports, *player.name* and *team.name* are the most common. (b) Shows column annotation counts of the top 20 numerical semantic types. A dominant type here is *rank*, which describes a column containing the placements of e.g. a team in a season standings table.

we discuss below, our corpus uses a much richer set of numerical data types that better reflects the characteristics in real-world data lakes which is very different from GitTables. For example, when looking at the semantic types that are assigned to numerical columns in GitTables, more than half (393,925) of the columns are labeled with just a single type *Id*.

In terms of the total number of columns, the tables in SportsTables (20.93 columns per table) are on average about 3 times wider than in GitTables (6.82 columns per table), which contains the widest tables among the existing corpora. As such, the number of columns in tables of SportsTables are reflecting better the width when comparing this to the characteristics of the tables in real-world data lakes which we analyzed. Moreover, considering the average number of rows per table, it can be seen that the tables in SportsTables have on average 246.72 rows. In comparison, tables in SportsTables are larger on average than in many other corpora where tables have typically fewer rows.

Annotation statistics. Semantic type annotation follows a two step process. First, we establish a directory with manually defined mappings from column header to semantic type for each existing header. Second, we label each column with the semantic type listed in the directory for its header. As a result, 56 textual and 419 numerical semantic types are present in the corpus. Thereby textual semantic types are those which specify textual columns and numerical types are those which specify columns containing numeric values. To compare the annotation statistics, we also counted the number of textual and numerical semantic types in an analysis of the existing corpora. The results of these analyses can be seen in Table 4.2. Different from our corpus, the sets of textual and numerical types are not disjoint in all other corpora (except TURL where no numeric values are present). This indicates that individual semantic types were assigned to both textual and numerical columns which is problematic if semantic type detection models should be trained and tested on these corpora. In particular, GitTables has a very large overlap and almost all semantic types are used in both column data types. To give an example, in GitTables the semantic types *comment, name* and *description* are assigned to both column data types. Next, we take a closer look into the semantic types of our corpus.

Figure 4.3a and Figure 4.3b show the top 20 semantic types (textual and numerical) in regards to how often they were assigned to a table column. It can be seen that the most common textual types across all sports are *player.name* and *team.name*. These are types that occur in almost every table. Other types such as *country* or *city* are also common, describing the player's origin or the team's hometown, for example. Among numeric semantic types, *rank* is by far the most common and is present in almost all tables. The type describes a column containing the placement of e.g., a team in a "seasons standing" table or a player in a "top scorer" table. All other numeric semantic types show mainly an equal distribution of the frequency, which is a good precondition for training machine learning models.

SportsTables vs. GitTables. Since GitTables is the largest corpus with the most tables, one could argue that a subset of GitTables would result in a new corpus with similar characteristics as SportsTables. To analyze this, we executed a small experiment in which we filtered out only tables from GitTables where the number of textual and numerical columns (min. 3 textual and 18 numerical columns) is at least the same as it is in SportsTables. The result was a corpus containing a total of 16,909 tables and 743,432 columns. On average a table has 12.53 textual columns, 31.43 numerical columns and 17.35 rows. However, looking at the semantic types that are assigned to numerical columns, more than half (393,925) of the columns are labeled with the type

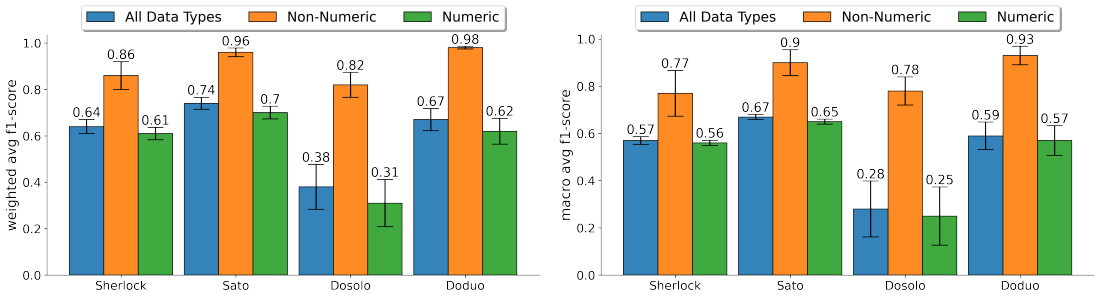


Figure 4.4: Results using different state-of-the-art semantic type detection models on our new SportsTables corpus. The overall differences in F1-Scores for predicting textual and numeric columns indicate that the models can handle textual data more effectively than numeric data.

Id. In terms of training and validating semantic type detection models, this is rather an unfavorable type representing no semantically meaning. Moreover, the next 5 most common numerically based semantic types are *parent, max, comment, created* and *story editor*, constituting a large proportion of the columns. The assignment of these types to numerical data is slightly less understandable and indicates a lack of quality in the automatically generated labels for table columns.

4.5 Study of Using SportsTables

In the following, we report on the results of using the state-of-the-art models Sherlock [50], Sato [113], Dosolo [100] and Doduo [100] on our new corpus. With this, we want to measure how well the semantic types in our corpus can be inferred by the models with a special focus on how each performs on textual and numerical columns.

Experiment Setup. For the experiments, we split the SportsTables corpus into training, validation, and test set. While creating the splits, we first extracted 20% of the data for the test set and then another 20% of the remaining 80% for the validation split. The rest of the data was used as the training set. We used the four pre-trained models as described above and re-trained them with the training data set. During the re-training, we replaced the last layer of the different models to support the number of semantic types that occur in SportsTables and then re-trained the entire neural network. In order to optimize the hyperparameters, we measured the performance of the respective re-trained models against the validation split. To report the final performance, we applied the re-trained models to the 20% test data set. For obtaining statistically reliable results, we

ran each experiment with five different random seeds and report the mean and standard deviation over multiple runs.

Results of the study. Figure 4.4 shows the results of the experiments reporting the support weighted and macro average F1-Scores in individual subplots for all four models. For each model, we plot the F1-Score across all semantic types (numerical & non-numerical) to show the total performance, but also the separate average F1-Score for only textually and numerically based semantic types, respectively. In the following we want to discuss the main aspects of the results in detail.

Non-numeric vs. numeric: As we can see in the figure, there is a significant performance difference between predicting textual and numerical semantic types for all models. While textual columns can be predicted with performances in a very promising range of 0.82-0.98, the performances for numerical columns are rather moderate ranging from 0.31 to 0.7. On average, the difference in F1-Score between textual and numeric types is 0.35 across all models. These results demonstrate that the models can better handle textual data and determine its associated semantic types more accurately than numerical data. Looking at the total performances over all types for each model, we see that they are rather moderate in the range of 0.38 to 0.74, but these insufficient results are primarily caused by poor prediction performances on the numerical based types.

Columnwise vs. tablewise: Looking and comparing the results of the columnwise models Sherlock & Dosolo and the results of the tablewise models Sato & Doduo, we observe that the tablewise models outperform the columnwise models. The results underline the known importance of considering not only individual column values for the task of semantic type detection of table columns but also to involve the table context. In particular, what we can see from the comparison of Dosolo and Doduo is how important it is, especially for numerical based columns, to include table context data for semantic type detection. As described above, numerical values provide less information content that can be used by a machine learning model to identify the type and therefore Doduo doubles the performance of Doslo by considering the complete table context. However, the resulting performance of 0.62 is rather moderate and demonstrates the shortcomings of the model on numerical semantic types. Comparing Sherlock and Sato also reflects the advantages of a tablewise semantic column type detection, whereas the performance improvement on just numerical columns is not as significant as in Dosolo vs. Doduo. We will discuss the reasons for this in the following.

Sherlock & Sato vs. Dosolo & Doduo on numeric: As described above, Sherlock & Sato (same feature set) as well as Dosolo & Doduo (same LM model) are models with an identical foundation. Focusing only on the F1-Scores on the numerical types, one

can see that Sherlock & Sato outperform Dosolo & Doduo. We think that this aspect is due to the fact that Sherlock & Sato extract features of numerical columns that better address numerical data (e.g. mean of individual digits occurring in a column), while in Dosolo & Doduo a LM model is used as the basis to encode the numerical column. LM models are optimized for text and can therefore not provide a representative encoding to infer the semantic type of numerical columns. Therefore, Dosolo & Doduo predictions on numerical columns are inferior to Sherlock & Sato.

4.6 Summary

Existing corpora used for training and validating semantic type extraction models mainly contain only tables with textual data columns. Moreover, tables in these corpora are very small regarding the total number of columns and rows. Consequently, it has not been studied precisely how well state-of-the-art models perform on a dataset that has on the one hand a very high percentage of numerical columns, as it occurs in real-world data lakes, and on the other hand contains tables that include more columns and rows. In this chapter, we thus introduced our new corpus for semantic type detection called SportsTables which contains tables that have on average approx. 3 textual columns, 18 numerical columns and 250 rows. Using our new corpus, semantic column type detection models can now be holistically validated against numerical columns. By conducting experiments with four state-of-the-art models on our new corpus, we showed that significant differences in the performance of predicting semantic types of textual data and numerical data exist across all models.

The identified limitations of the existing models on numerical columns motivated the last contribution in this dissertation – a new semantic type detection approach specifically designed to support numerical data columns, which we will introduce in the upcoming chapter.

5 Pythagoras: Semantic Type Detection of Numerical Data

Publication. The work on building a new data corpus that contain a high proportion of tables with numerical columns is published in the peer-reviewed publications "Pythagoras: Semantic Type Detection of Numerical Data Using Graph Neural Networks (Short Paper)" in the *Lernen, Wissen, Daten, Analysen (LWDA) Conference Proceedings, Marburg, Germany, October 9-11, 2023* & "Pythagoras: Semantic Type Detection of Numerical Data in Enterprise Data Lakes" in the *Proceedings 27th International Conference on Extending Database Technology, EDBT 2024, Paestum, Italy, March 25 - March 28.*

Contributions of the author. Sven Langenecker is the leading author of the publications [58] & [57] mentioned above. He is thus responsible for the proposed semantic type detection model Pythagoras, the graph representation of tables, the architecture of the model, the experimental evaluations and the manuscript. The co-authors Christoph Sturm, Christian Schalles and Carsten Binnig contributed invaluable feedback. All authors agree with the use of the publication for this dissertation.

In enterprise data lakes, numerical data plays a dominant role, making up a much larger proportion compared to non-numerical data [60] since they provide insights into various business domains, including finance, manufacturing, healthcare and marketing. Numerical data often contain critical information such as sales figures, production metrics, customer demographics and financial records. Therefore, it is essential to provide a solution that can automatically detect the correct semantic type of table columns containing numerical values, enabling data analysts and data scientists to find required data for downstream analysis and thus address the dataset discovery problem in data lakes [14, 16, 29, 53, 78]. The study in the previous chapter showed that the existing state-of-the-art semantic type detection models have shortcomings when applied to columns with numerical values. They are mainly designed to handle textual data and thereby achieve a very high grade of accuracy, but if they are used on numerical columns, the prediction accuracy drops. In this chapter, we thus introduce our new semantic type detection approach called *Pythagoras*, which can not only predict the semantic type of non-numerical table columns with high accuracy but also of numerical columns. To achieve this, the main idea of the new model architecture is to use a GNN together with a new graph representation of

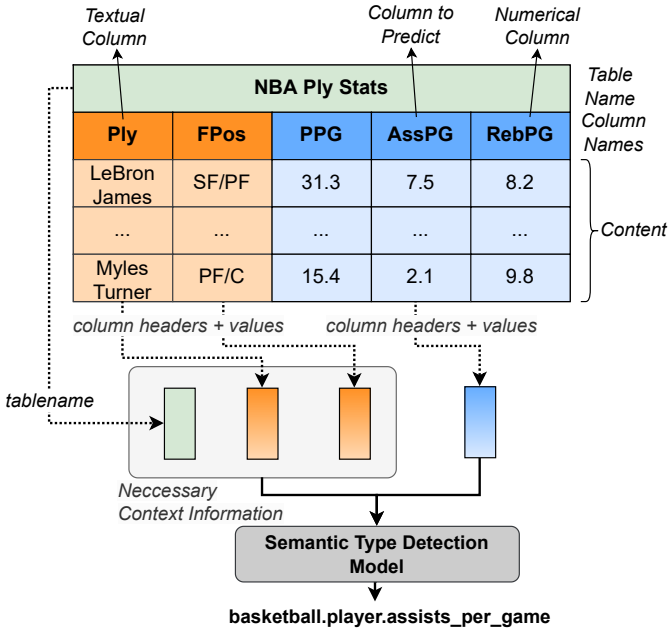


Figure 5.1: Figure shows an example of predicting the semantic type of the numerical table column 'AssPG'. To predict the correct type, it is crucial for the model to have the possibility to incorporate textual context information such as the table name and neighboring non-numerical columns.

tables and their columns. In the following, before introducing our new model *Pythagoras*, we first discuss the key aspects by detecting the semantic type of numerical table columns. Subsequently, we present our new graph representation of tables and then introduce the model architecture of *Pythagoras* in detail. Afterward, we report on the main results of our experimental evaluation which includes detailed analyses on individual semantic types as well as an ablation study.

5.1 Context is Essential for Numerical Data

To predict the semantic type of table columns containing numerical values, it is essential to have textual (non-numerical) data of the same table as context information as illustrated in Figure 5.1. Predicting, for example, the semantic type of the column 'AssPG' by using only the included values {7.5,...,2.1} is almost impossible while values of columns with textual types such as 'Ply' are more indicative for the type. The reason for this

is that numerical values have in general a limited information entropy[1] and are often similarly distributed for different semantic content [61]. To address this problem, rich non-numerical contextual information such as contents of neighboring non-numerical columns as well as column headers and table names can be leveraged to increase the accuracy to determine the correct semantic type of the numerical column. In the example of Figure 5.1 the table name 'NBA Ply Stats' and the information of the textual columns such as 'Ply' and 'FPos' can be leveraged as context information. This now allows the model to recognize that the table is from the basketball domain and allows the model to better predict the semantic type of column 'AssPG' as 'basketball.player.assists_per_game'. As such, for a semantic type detection approach that should be able to handle numerical data, it is crucial to incorporate the ability to leverage all context information to predict types for numerical data within the model architecture. Unfortunately, existing model architectures do not have such a predefined technique where non-numerical contextual information can be strategically leveraged for predicting numerical columns. Therefore, in the following, we present our new semantic type detection approach called *Pythagoras* which includes a novel architecture that can utilize non-numerical contextual information to predict types of numerical columns. The main idea of the new model architecture is to use GNNs together with a new graph representation of tables and their columns. This graph representation includes directed edges to provide necessary contextual information (e.g. table name, neighboring non-numerical column values) for predicting the correct semantic type of numerical columns using the GNN message passing mechanism. Thus, the model learns which contextual information is relevant for determining the semantic type. To the best of our knowledge, our semantic type detection model *Pythagoras* is the first approach in this direction.

5.2 Background to GNNs

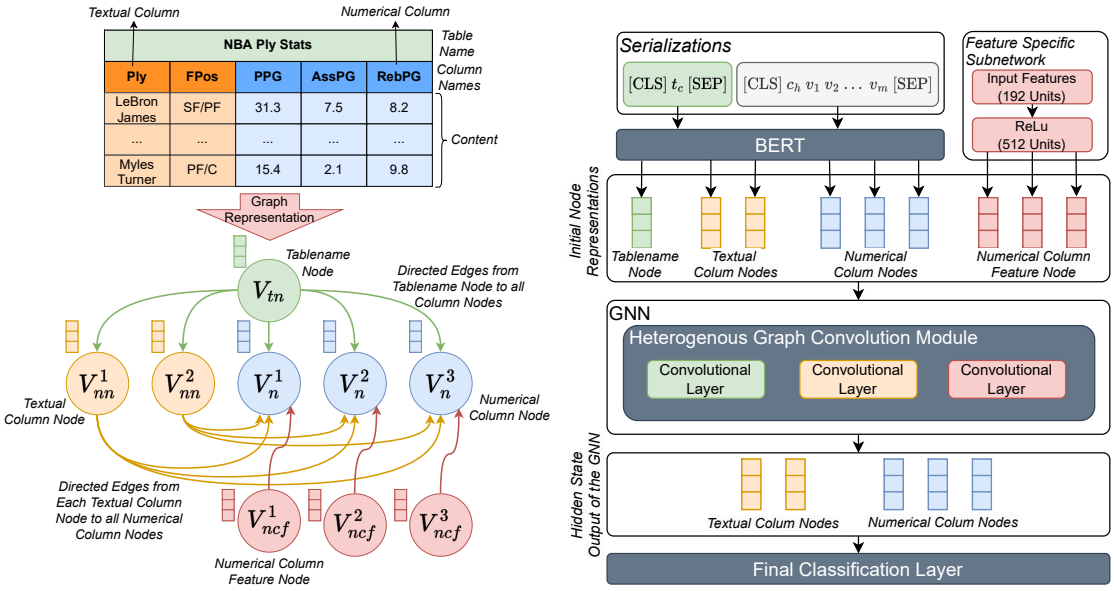
As we use GNNs and a graph representation of tables in our new semantic type detection model, in this section we give a formal definition of graph data structures and the basics of the main aspects of GNNs.

The fundamental component of graph-based machine learning with GNNs is the possibility to use a graph representation of the data as input to the model. Graph data structure gives the opportunity to represent data with intricate relationships in a

[1] Generally numerical values can be encoded with much less bits than string values resulting in lower overall entropy values [98]

simple way. In graph theory, a graph is a mathematical structure $G = \{V, E\}$ composed of two primary elements: a set of nodes V (also called vertices) and a set of edges E (also known as links or connections). Nodes represent entities or points, while edges represent relationships or connections between these entities. Graphs can be categorized into to different types: homogenous graphs and heterogeneous graphs. In a homogeneous graph, all nodes belong to the same type or category, and all edges represent the same type of relationship. These graphs are typically used in simpler scenarios where entities are of the same kind and relationships are uniform. In a heterogeneous graph, nodes can belong to different types or categories, and edges can represent different types of relationships. Heterogeneous graphs are particularly useful when modeling complex, diverse data where entities and relationships vary. For instance, in a recommendation system, a heterogeneous graph could represent users, movies, and genres, with edges representing actions like "user watches movie" or "movie belongs to genre." In our use case, we use a heterogeneous graph data structure for the table representation. With that, we are able to model the different components of a table such as the table name, non-numerical columns, and numerical columns while also modeling various relationships between these components. This allows us to pre-define the necessary information exchange in advance and provide it to the model.

GNNs are a class of deep learning models designed to process data represented as graphs. They have gained significant popularity in various fields, such as social network analysis, recommendation systems, and bioinformatics [68, 95]. GNNs typically consists of multiple layers, each of which updates node representations by aggregating information from neighboring connected nodes. One of the fundamental distinctions within GNN architectures lies in the type of aggregation layers being used. There are various implementations of such aggregation layers, which perform node updates according to different algorithms. Among these, one of the most commonly used are Graph Convolutional Networks (GCNs). GCN operates by computing weighted sums of neighboring node features, analogical to traditional convolutional layers in image processing. Because GCN layers are designed for homogeneous graphs, there exist specialized extensions and variations to work with heterogeneous graphs. Heterogeneous graph convolutional modules [103] are designed to handle heterogeneous graphs by applying sub-modules (GCN layers) on different edge types and afterward aggregate them. As we describe in more detail later in Section 5.3, we use such a module to be able to use our heterogeneous graph structure and to learn independent weights for the different edge types in the graph, which determines how strongly weighted information are sent over the edges. Overall, in our semantic type detection approach, we leverage the described node update mechanism



(a) Graph Representation (b) Model Architecture

Figure 5.2: (a) Shows the conversion of a table into a heterogeneous graph representation. The key aspect of the graph is that it provides all the necessary contextual information through its structure (nodes and directed edges), resulting in improved predictions of the semantic types of numerical columns. (b) Shows the complete model architecture of the neural network.

by aggregating information from neighboring connected nodes to update the numerical column node representation with information from a textual column. In this way, context information from textual columns are embedded into the node representation of the numerical column, thus leading to better semantically interpretable representations.

5.3 Overview of Pythagoras

In the following, we introduce our new semantic type detection approach *Pythagoras*. We begin by explaining our new graph representation of tables and then present the model architecture of our semantic type detection approach in detail.

5.3.1 Graph Representation of Tables

Figure 5.2a demonstrates how we convert a table and its columns into a graph representation using an example table. The table contains a table name (t_n), two non-numerical

columns (c_{nn}) and three numerical columns (c_n), each with column headers (c_h) and column values ($v_1, v_2, ..., v_m$). In the figure, we can see how the table is transformed into a graph $G = \{V, E\}$ composed of a set of nodes V and a set of edges E including four different node types V_{tn}, V_{nn}, V_n, and V_{ncf} for different artifacts.

The first node type V_{tn} (green node) represents the tablename. Additionally, the graph contains a node of type V_{nn} (orange nodes) for each non-numerical column. This node type represents the entire column including column values and headers. In the same manner, for each numerical column, we create a node of type V_n (blue nodes) representing numerical columns and their contents. Finally, nodes with a node type V_{ncf} (red nodes) are added for each numerical column to encode specific features of the numerical columns.

We decided to use an additional node type V_{ncf} to encode specific features for numerical columns since this allows us to first use a pre-trained LM for computing a representation based on the joint features that are shared between both non-numerical and numerical columns such as column headers. In addition, we further add the nodes V_{ncf} for the numerical-only columns, each holding a vector with additional specific features for numerical columns for which we use a separate encoding strategy with a separate simple multilayer perceptron network. To be more precise, we additionally encode 192 different statistical features for encoding a numerical column.

5.3.2 Leveraging Contextual Information

As described before, only using the numerical values for predicting the semantic type of numerical columns is in general not sufficient, and contextual information is needed. Due to this aspect, we add directed edges to our table graph representation to predefine in which way necessary additional context information should be injected through the message-passing mechanism of GNNs [54] into the numerical column representation (node V_n) and thus enrich it for better predictions.

More precisely, as shown in Figure 5.2a we construct direct edges from each non-numerical column node V^1_{nn}, V^2_{nn} to all numerical column nodes V^1_n, V^2_n, V^3_n (yellow edges) to provide the context information from the non-numerical columns to the numerical columns. Furthermore, we add directed edges in the graph from the table name node V_{tn} to all non-numerical V_{nn} as well as numerical V_n nodes (green edges). This edge handles not only the contextual information for numerical columns but also for non-numerical columns. As we will show in our experiments, using the table name as context information also leads to performance improvements for non-numerical columns. Finally, the graph

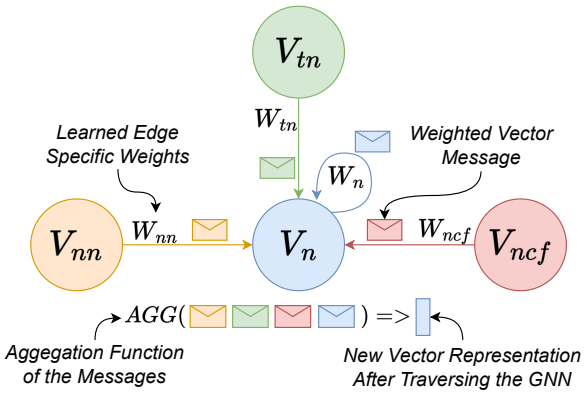


Figure 5.3: Heterogeneous graph convolutional module of *Pythagoras* for the nodes V_n. Information from the nodes V_{nn}, V_{tn} and V_{ncf} is passed to node V_n. Each edge connection $(W_{nn}, W_{tn}, W_{ncf}, W_n)$ has its own learned weights, which determine how strongly weighted the information is sent over the edges. Finally, all messages (vectors) are combined to form a new representation of the node by an aggregation function.

has directed edges for integrating the additional statistical features into the encoding of numerical columns (red edges from $V_{ncf} \rightarrow V_n$).

As a consequence when using this graph structure as a basis for our GNN-based model architecture (cf Section 5.3.3), the vector representations, computed for the numerical column nodes V_n during training, result in an enhanced information content that is more suitable for an accurate prediction of the underline semantic type. Interestingly, leveraging context by modeling edges in a GNN not only improves the prediction of numerical but also non-numerical types as we show in our evaluation (cf. Section 5.4).

5.3.3 Model Architecture

In Figure 5.2b we can see the model architecture of *Pythagoras*. In the following, we explain first the details of the model architecture and then explain how the model can be used to detect numerical semantic types.

5.3.3.1 Architecture and Training

The model comprises three essential components. These components include (1) a pre-trained LM to encode all features from non-numerical and numerical columns[2], (2) a

[2]We use BERT but *Pythagoras* is independent of how to generate the initial embeddings, and there may exist alternative language models or embedding methods that could potentially yield even better results in this context.

specific subnetwork to process the additional features of numerical columns and (3) the GNN to aggregate all information.

The upper part of the architecture in Figure 5.2b shows how we generate the input of the BERT model to get the initial representations for each column. Additionally, we use BERT to encode table names. To serialize the individual columns, we encode the input sequence for non-numerical as well as for numerical columns, using the column header and the column values as follows: $serialize(c_i)$ ::= [CLS] c_h v_1 v_2 ... v_m [SEP]. Additionally, to generate the initial representation of the node V_{tn} we thus serialize the table name as follows: $serialize(t_c)$ ::= [CLS] t_c [SEP]. For columns and table names, we use the representation computed by BERT for the CLS token as initial node representation for the GNN.

To embed the additional extracted features of the numerical column values, the model contains a feature-specific subnetwork similar to the approach in [50]. As can be seen in the architecture, the subnetwork consists of a linear layer that maps the 192 provided features to a vector that matches the shape of the other initial vector representations (BERT outputs vectors with dimensions of 768). This network is trained end to end with the GNN while the BERT parameters are frozen.

The initial vector representations generated by the BERT model and the subnetwork are used as initial internal representation for all nodes V_{tn}, V_{nn}, V_n, and V_{ncf} in our graph data structure which serves as input for our GNN model. As GNN, we use a heterogeneous graph convolutional module that combines different graph convolutional layers [54] for each occurring edge type. Since we have 3 different edge types in our graph, the heterogeneous convolutional module combines 3 independent graph convolutional layers. The heterogeneous convolution module first performs a separate graph convolution on each edge type, then sums the message aggregations on each edge type as the final result for the nodes. Figure 5.3 shows the behavior of this module for a numerical column node V_n that is connected with other nodes over the different edge types. The module works in a similar way also for non-numerical columns V_{nn} leveraging, however, only information from table name as shown in Figure 5.2a.

The module allows the model to learn separate weights for the different edge types and thus enables it to embed connected neighboring nodes and their information to different degrees. For example, the model can learn for V_n nodes that the information of the table name (provided by V_{tn}) is less important than the information of adjacent non-numerical columns (provided by V_{nn}). By learning distinct weights for each edge, we can effectively capture the nuances and dependencies in the data, ultimately enhancing the model's

ability to make contextually informed predictions that lead to the overall effectiveness of our approach, which we will show more in detail in Section 5.4.5.

After traversing the GNN network, we extract the hidden states of the nodes V_{nn} (updated representation of non-numerical columns) as well as of V_n (updated representation of numerical columns). Subsequently, these hidden states are then fed into a final classification layer to perform the semantic type classification task. In this last classification layer, the output size is determined by the number of distinct semantic types present in the corpus.

5.3.3.2 Detecting Numerical Types

To highlight the advantage of using our graph representation of tables together with a GNN for semantic type detection of numerical data types, let us take a look at the following example. Considering the node V_n^1 in Figure 5.2a which stands for the numerical column 'PPG' (point per game statistic of a basket player) of the table, the column contains values in the range of about 15-32 and the semantic type could be ambiguous in a data lake about sports event and could represent values of different types (e.g., *basketball.player.points_per_game*, *football.player.yards_per_game* or *temperature*).

However, after iterating over a GNN layer, the values of the two non-numerical columns V_{nn}^1, V_{nn}^2 are embedded because of the designed yellow edges. These provide basketball player names (Lebron James, ..., Myles Turner) as well as basketball field positions (SF/PF, ..., PF/C) as context information. According to this additional data, it is clear that the semantic type *temperature* is not very likely for this column. Because of the fact, that tables about player statistics in basketball as well as in football are structured very similarly and contain both columns with player's names and field positions, it is not yet clear whether the semantic type is *basketball.player.points_per_game* or *football.player.yards_per_game* for example.

Besides the previous context data of the non-numerical columns, information about the table name is also injected via the green edges during a GNN layer pass. This information contains the text 'NBA Ply Stats' ('NBA' is the name of the basketball league) and it is now unambiguous determinable that *basketball.player.points_per_game* must be the valid semantic type. The other passed information from the additional statistical feature nodes V_{ncf} also provides an improvement for distinguishing ambiguities, since the value range of numerical columns with different semantic types can be the same but the value distribution can be different. These different characteristics are covered by the extracted statistical features of numerical columns.

Table 5.1: Characteristics of the datasets in our experiments.

Dataset	#Tables	Non-Num. Cols./Table	Num. Cols./Table	#sem. Types
SportsTables	1,187	2.83	18.1	462
GitTables Numeric	6,577	2.08	8.95	219

5.4 Experimental Evaluation

In the following, we first introduce the two datasets SportsTables and GitTables before we describe our experimental setup and evaluation methodology. Afterward, we discuss the main results of our experiments.

5.4.1 Data Sets and Baselines

For evaluating *Pythagoras*, we use two different real-world data lakes with a large number of semantically annotated tables. When selecting the datasets, the goal was to choose a corpora that contain tables with a high proportion of numerical columns. This allows us in particular to explore and compare the existing models with *Pythagoras* on numerical data. As shown in Table 5.1, we use two corpora SportsTables [60] and GitTables Numeric which is based on [47]. Both corpora contain a high number of numerical columns per table and represents a numerical to non-numerical ratio commonly found in enterprise data lakes [61].

Dataset. *SportsTables [60].* As the first data corpus in our experiments, we use our generated SportsTables corpus, which we introduced in the previous Chapter 4. As already mentioned, the corpus contains real-world data tables collected from various sports domains such as soccer, basketball, baseball and football using web scraping techniques. Such data tables are especially rich in numerical columns as many different sport-specific statistical measurements are reported. As can be seen in Table 5.1, the tables in the corpus contain 2.83 textual and 18.1 numerical columns on average. The corpus includes a very high number of 462 unique semantic types. Thereby semantic types are very fine granular, which is a major challenge for semantic type detection models. For example, there are types such as 'basketball.player.assists_per_game' or 'soccer.player.assists_per_game'.

GitTables Numeric [47]. The original GitTables data set is a corpus of tables created by extracting CSV files from GitHub repositories. Table columns are labeled with semantic types from Schema.org [35] and DBpedia [6] using two different automated annotation

methods. In our experiments, we have focused on the annotations origin from DBpedia and the results of the semantic annotation method. For our experiments, we constructed a derived corpus called GitTables Numeric. For this corpus, we filtered out tables with a high proportion of numerical columns. To achieve this, we only included tables where at least 80% of all table columns are numerical. In order to have enough samples of each semantic type to train, validate and test the models, we also filtered out columns that have a semantic type occurring less than 10 times in total. Based on these filter criteria, we ended up with a corpus that contains 6,577 tables with 2.08 textual and 8.95 numerical columns per table on average (see Table 5.1) and a total of 219 semantic types.

Baselines. In our evaluation, we compare our model *Pythagoras* against five state-of-the-art semantic type detection models. As baselines we considered Sherlock [50], Sato [113], Dosolo [100] and Doduo [100]. Despite that Sato and Doduo also incorporate context information to predict the semantic type of a column, they do not specifically address numerical-based columns and do not offer a predefined approach for injecting contextual information into the prediction of numerical columns. All models were trained on the same data as *Pythagoras*.

Given the recent advancements in LLMs like GPT-3.5 [10, 84], which have been extensively trained on vast amounts of data, one might wonder if such models cannot predict the semantic type for non-numerical as well as for numerical columns with high accuracy through a straightforward finetuning. Finetuning an LLM to a specific task has already shown success [52, 66, 86, 102]. In light of these considerations, we additionally explore the capabilities of recent LLMs in our study by adding a fine-tuned GPT-3.5 model. We opted for fine-tuning as opposed to prompt designs due to its potential to yield higher performances and to train on a larger number of examples. To build this baseline model we fine-tuned the *gpt-3.5-turbo* model, following the instructions in [2] using the same training data we used for *Pythagoras*.

5.4.2 Experimental Design

Setup. To run the experiments, we split each dataset into three parts: training, validation and test set. We divided the datasets into 60% training, 20% validation and 20% testing splits. Since in both datasets, the gold labels were assigned in an automatic manner by using the individual column headers, we did not include the headers in the serializations of the columns, which is different from what is described in Section 5.3. When running the experiments, we trained each model using the training split and conducted hyperparameter tuning on the validation set.

Table 5.2: Experimental results on the SportsTables corpus.

Model	support weighted F1-Score			macro F1-Score		
	numerical	non-numerical	overall	numerical	non-numerical	overall
Sherlock [50]	0.609	0.856	0.641	0.555	0.767	0.57
Sato [113]	0.703	0.961	0.736	0.650	0.903	0.668
Dosolo [100]	0.313	0.822	0.379	0.245	0.782	0.285
Doduo [100]	0.623	0.98	0.67	0.567	0.933	0.594
GPT-3[3] [10]	0.446	0.872	0.501	0.404	0.760	0.423
Pythagoras	**0.829**	**0.996**	**0.851**	**0.790**	**0.97**	**0.803**

In addition, we used the performance results on the validation split during training to apply an early stopping mechanism. To measure the final performance of each model, we loaded the checkpoint of the model with the highest F1-Score on the validation set and then applied it to the test data. We ran each experiment with five different random seeds and reported the mean across multiple runs to obtain statistically reliable results. As evaluation metrics, we used support-weighted F1-Score, weighted by the number of columns per semantic type and the macro average F1-Score as used in previous studies [20, 50, 100, 113].

***Pythagoras* implementation.** We implemented our model *Pythagoras* using Python together with the modules PyTorch [85], DGL [106] and the Transformers library [107]. As described in Section 5.3, our neural network consists of three main components. A pre-trained LM to generate initial vector representations, a subnetwork for the numerical-based feature set, and a GNN that allows to exchange context information.

As pre-trained LM, we used the vanilla BERT [22] (bert-base-uncased) model to be comparable to [100] which comes with 12 encoder layers. We used tokenizer and pre-trained model of the Transformers library from Hugging Face [28]. During the training process, we froze the 12 layers of BERT, preventing their weights from being updated.

The graph data structure and the GNN were implemented with the DGL library. To update the weights of the GNN during training, we applied an Adam optimizer with an initial learning rate of 10^{-5} and a linear decay scheduler with no warm-up. Since our purpose is to realize a multi-class prediction task (one semantic type label per column), we used the cross entropy loss as a loss function.

5.4.3 Exp. 1: Overall Efficiency

[3] fine-tuned

Results on SportsTables. Table 5.2 shows the experimental results on SportsTables. For each model, we list the F1-Scores overall data types to show the total performance, but also the separate average F1-Scores for only numerical and non-numerical data types, respectively. As the first main result, we can see in the table that our model *Pythagoras* outperforms all existing state-of-the-art models in all reported aspects. Looking only at the results on the numerical columns, we can see that our model achieves an improvement of +17.92% support weighted F1-Score and +21.53% macro F1-Score. These results verify that our designed mechanism of providing context information to predict the semantic type of numerical data is more suitable than the methods in the existing models Sato and Doduo.

In Sato, contextual information is provided by a table topic vector, which is formed by an accumulation of all values in the table. Since tables in the SportsTables dataset contain a large proportion of columns with numerical values (on average 18.1 are numerical columns and 2.83 are non-numerical, see Table 5.1), this table topic vector does not have the necessary effect. In addition, Satos linear-chain CRF also does not lead to significant improvements, since the tables in SportsTables are not always structured in the same way (column orders vary between tables). This aspect can be seen by the comparison of Sato to Sherlock, which is the same model without a table topic vector and a linear-chain CRF module. The improvements from Sherlock to Sato are not significant.

Doduo also achieves only moderate performance values with 0.623/0.564 (support weighted/macro) F1-Score. On one hand, this is due to the fact that only very few individual column values can be included in the token sequence, since the BERT model is limited to 512 elements and the tables have on average 20.93 columns. On the other hand, the BERT model learns the structure of the tables which, as with Sato, has negative effects with non-identical cross-table structures. Furthermore, it is still unclear how deep the understanding of numbers is in LMs like BERT, since they are essentially pre-trained on textual data. Unlike the existing models, our model is independent of the column order of the tables due to the graph structure. If columns are arranged differently between tables, this has no negative effect.

When we examine the results on textual data, we generally observe that all models perform well. In particular, the models Sato, Doduo, as well as our model *Pythagoras* achieve high accuracy. Interestingly, also for non-numerical columns our model is slightly better than existing models with 0.996/0.970 F1-Scores. This improvement is due to the design aspect that our model uses the contextual information of the table name also for the non-numerical column representations ($V_{nn} \rightarrow V_n$ edges). Moreover, our results

Table 5.3: Experimental results on the GitTables corpus.

Model	support weighted F1-Score			macro F1-Score		
	numerical	non-numerical	overall	numerical	non-numerical	overall
Sherlock [50]	0.725	0.989	0.775	0.411	0.491	0.707
Sato [113]	0.733	0.991	0.781	0.443	0.707	0.491
Dosolo [100]	0.518	0.986	0.606	0.245	0.694	0.343
Doduo [100]	0.761	0.992	0.804	0.409	0.749	0.489
GPT-3[4] [10]	0.531	0.938	0.610	0.143	0.277	0.211
Pythagoras	**0.813**	**0.990**	**0.846**	**0.476**	**0.893**	**0.544**

demonstrate the aspect that on numerical data, the prediction of the semantic type is in general harder than the prediction of non-numerical data.

In summary, the results on the SportsTables dataset demonstrate that our model architecture, in conjunction with the graph representation of tables, leads to significantly improved performance in predicting semantic types for numerical-based columns.

Results on GitTables. Table 5.3 shows the experimental results on GitTables using the same metrics as before on SportsTables. The results show that *Pythagoras* outperforms all other models in predicting the semantic types. Considering the performance on numerical columns, it becomes evident that our model surpasses the performance of the best existing model, Doduo, by a remarkable improvement of +6.83%/16.38% F1-Score.

This gain in performance highlights the effectiveness of our model in handling numerical data, setting a new benchmark in this domain by outperforming all state-of-the-art approaches. Different from the results on the SportsTables corpus, among the baselines, Doduo and Sato perform nearly equally. This is mainly due to the aspect that the GitTables corpus contains tables with fewer columns on average, and therefore Doduo can use more column values in its token sequence and with that build a better representation using the BERT model.

Looking at the performance on non-numerical data columns, we can see that all models achieve mostly the same support weighted F1-Scores (about 0.990). However, considering the macro F1-Scores our model *Pythagoras* reaches by far the best value with 0.893. This is an improvement to the second-best model Doduo by +19.23%, showing again the benefit of providing the table name as contextual information for predicting the semantic type of non-numerical columns. In summary, the results on GitTables show that our model *Pythagoras* sets new state-of-the-art performances for predicting the semantic type of numerical table columns.

[4] fine-tuned

5.4.4 Exp. 2: Performance for Individual Types

Figure 5.4 shows a more detailed analysis of the performances between *Pythagoras* and Sato on numerical columns in SportsTables. We chose Sato as comparison model because it was the best baseline model on numerical columns in this dataset. On the left side, the pie chart shows for how many semantic types of numerical columns which model performed better regarding the F1-Score. Out of a total of 384 numerical semantic types, *Pythagoras* was able to achieve substantially better performances than Sato on 202 of them. For 80 types, the two models achieve equal F1-Scores and for 74, Sato is better than *Pythagoras*. This demonstrates that our model is not only more accurate for individual numerical semantic types but also for a very large proportion of them.

To show how large the F1-Score differences between the two models across the numerical types are, boxplots of the differences for the cases *Pythagoras*>Sato and vice versa are shown on the right of Figure 5.4. In the case where our model achieves higher F1-Scores, we can see that the median value of the distances is 0.2. The 0.75 quantile is 0.4 and there are also a few types where our model is better than Sato by more than 0.9. In addition, the distribution is shifted upwards towards the larger distance values. In the case where Sato is better, the median is about 0.1 and the 0.75 quantil is 0.2. The distribution is also shifted upwards, but not as much as in the other case. In conclusion, these results show that there are many types for which *Pythagoras* performs much better than Sato and Sato can only achieve very low F1-Scores and the differences to our model are significant. In the other case, for the majority of types in which Sato performs better, our model *Pythagoras* achieves only slightly lower scores.

Overall, this suggests that our model architecture and the method we designed for providing context information are better suited for detecting the semantic type of numerical data.

5.4.5 Exp. 3: Ablation Study

Different graph variants. To verify the different design aspects of our approach, we tested variants of *Pythagoras*. At first, we tested modifications of our graph representation of tables. In particular, we wanted to investigate which contextual information has which effect on the prediction of the semantic type. Table 5.4 shows the results of this ablation study by displaying support weighted and macro average F1-Scores on numerical columns. The first row reports the results of using our regular model and graph while the next rows presents the results when various nodes and edges are removed in the graph representation. Here w/o V_{tn} means that in the graph the node representing the table name has been

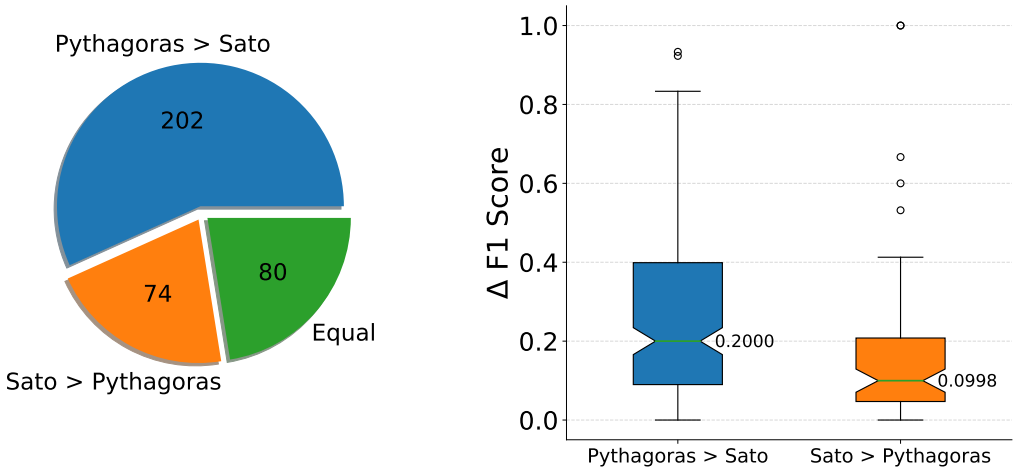


Figure 5.4: The left chart shows the number of numerical types for which *Pythagoras* performs better than Sato and vice versa. In the right chart we see box plots for the F1-Score differences between the two models on the different numerical semantic types where *Pythagoras* was better than Sato and vice versa.

Table 5.4: Ablation study results on only numerical columns of the SportsTables dataset. We tested different graph structures that provide different types of contextual information (upper part). The lower part shows results when including the column header c_h as additional information in the serialization of a column.

Variant	support weighted avg F1-Score	macro avg F1-Score
Pythagoras	0.829	0.790
w/o V_{tn}	0.812	0.759
w/o V_{nn}	0.785	0.733
w/o V_{ncf}	0.813	0.765
w/o V_{tn}, V_{nn}	0.724	0.693
w/o V_{tn}, V_{nn}, V_{ncf}	0.324	0.252
w/ original c_h	0.991	0.950
w/ synthesized c_h	0.972	0.926

removed and thus also the provision of this context information for the prediction of the semantic type of the columns. Note, that the other nodes V_{nn} and V_{ncf} are still present in the graph and still provide contextual information to the numerical columns representations. Equally, w/o V_{nn} means that the edges of non-numerical to numerical columns have been removed and thus the flow of information from the non-numerical columns no longer occurs during a GNN layer pass. However, in this variant, the other nodes are present.

The first finding that can be seen in the results is that when we remove the nodes V_{nn}, we see the highest performance drop. The F1-Score decreases in this case -0,044/-0,057 in comparison to the regular model. Thus, we can conclude that the most important contextual information for a correct prediction of the semantic type of numerical-based columns are the values of the non-numerical columns from the same table. The second most important context is the table name (V_{tn}) and the least important are the statistical features of the numerical values in the columns (V_{ncf}). Without the table name as context, the model performance decreases a bit more than without the statistical features. To see how good the performance is when making a semantic type prediction only using the numerical values of the columns (V_n and V_{ncf}), we have also considered a variant in which V_{tn} (table name) and V_{nn} (non-numerical columns) nodes are not present. With this variant, the F1-Score drops very sharply and the model only achieves values of 0.724/0.693. This result again shows the immense importance of textual context information in predicting the semantic type of numerical data. In addition, we have tested a variant in which only the V_n nodes are present (w/o V_{tn}, V_{nn}, V_{ncf}). As expected, we just get similar performances to the Dosolo model, since in this constellation both model structures are very similar.

Different column serializations. As mentioned before, in the experiments of Section 5.4.2, we did not include the original column headers c_h in the serialization of a column because they were previously used to semi automatically assign the true semantic types (gold labels) to the columns. However, to show the impact column headers can have on the performance of numerical column predictions, we created synthetic column headers and used them in an experiment. We created the synthesized column headers using GPT by giving us a list of 10 possible abbreviations for the respective column headers. For example, for the header ”Player Age” GPT provided the list [”PA”, ”PlAge”, ”PAG”, ”PLAG”, ”PlrAge”, ”PlyAg”, ”PLA”, ”PrAge”, ”PlyrA”, ”PlayA”]. Afterward, for each column, we randomly selected an abbreviation from the list and used it as the column header. The lower part of Table 5.4 shows the results of this experiment. We can see that the inclusion of column headers has an additional positive effect on predicting

the semantic types of numerical data, achieving F1-Scores of 0.972/0.926 (close to the performance when using the original highly indicative column headers).

5.5 Summary

Numerical columns often make up a large proportion of the data stored in data lakes and in many cases contain critical information. Therefore, it is even more important to have a model that can detect the underlying semantics of these data types robustly. While recent papers propose approaches for extracting semantic types, unfortunately, they have been designed primarily on non-numerical data and therefore do not provide accurate performances when used on numerical data columns. To tackle the shortcomings of the existing models, we introduced in this chapter our new semantic type detection approach called *Pythagoras*, which is specifically designed to robustly handle numerical table columns. The graph representation of tables and GNN architecture of *Pythagoras* establish an intrinsic mechanism that provides all necessary context information to determine the correct semantic type of numerical columns. By conducting experiments on two different data lakes, we compared Pythagoras against five existing models and showed that our model outperforms all other models on numerical columns. In comparison to the best existing model, we reported F1-Score increases of around +22%, which sets new benchmarks.

6 Conclusion and Future Work

This thesis contains several contributions to enable semantic type extraction of table columns stored in data lakes. Thereby, the dissertation particularly focuses on enabling an adaption of existing learned models to the respective data lake with minimal effort and also on providing new approaches for the more difficult semantic labeling of numerical data. In the following, we will first reflect on these contributions and afterward point out possible directions for future research.

6.1 Reflection

This dissertation tackles the data discovery problem in data lakes by using automatic semantic type extraction of table columns. Although approaches for the task of semantic type extraction of table columns have already been developed, they essentially have two limitations when applied to a real-world data lake: (i) They do not generalize across different data lakes, instead they are trained specifically for one data lake. As a result, costly adaptation to the individual data lake is necessary. (ii) Existing approaches are mainly designed to handle the semantic type extraction of textual data and fall short when applied to numerical data. In real-worl data lakes, however, a large proportion of the data is often numerical and contain critical information, which make an accurate extraction of the semantics of this data type essential. To tackle the mentioned limitation, this dissertation introduced four main contributions (see Section 1.4) structured as follows.

First, in Chapter 2, we showed an evaluation of the quality of state-of-the-art semantic type extraction models on new unseen real-world data lakes. The experimental results demonstrated that existing learned models do not generalize to unseen data lakes and always require a costly adaption to the individual data characteristics and semantics that occur in the data lake where the model should be applied. The results additionally pointed out that this aspect also appears to semantic types that the learned model already knows (i.e., learned by seeing the semantic types in the training data of the model), since the same semantic types of table columns can have different data patterns in different

data lakes. In order to eliminate the problem of the necessary costly adaption of the model to the respective data lake, we suggested a new direction of using weak supervision to generate new labeled training. This method of weak supervision effortlessly generated new training data directly from the corresponding data lake, enabling the use of this newly labeled data for re-training the existing model and thereby adapting it to the same data lake. An initial experiment in Chapter 2 using one LF to label table columns indicated the general effectiveness and potential of this approach.

Hence, in the following Chapter 3, we proposed the first labeling framework based on the idea of weak supervision to generate new labeled training data for a new unseen data lake. Using the generated training data by the labeling framework to re-train a learned semantic type extraction model can not only lead to a performance improvement in case we re-train the model on a data lake with types the model has already seen, but also in case we train the model on a data lake with unseen types from scratch. To label table columns and thus generate new training data, we introduced a set of pre-defined LFs, that are embedded in our labeling framework. This included a set of generic LFs, which are domain-independent and can be used on any data lake out of the box. Contrary to this, we implemented another set of more domain-specific LF. These LFs rely on the contribution of domain-experts' knowledge and be used to generate new training data for completely unknown semantic labels for which there are no samples at all. Because numerical columns are generally more difficult to label with semantic types, we have also proposed our new label generation procedure called *Steered-Labeling* in this chapter. *Steered-Labeling* is integrated as a core component in our labeling framework, which is why we subsequently named the complete labeling framework *STEER*. The intuition is that in *Steered-Labeling* we separate the process into two subsequent steps: *STEER* first labels the non-numerical columns that are easier to label. Afterward, *STEER* then uses these labels to "steer" the labeling of the numerical columns. With this, *STEER* is able to not only generate high quality training data of textual columns but also to semantically label numerical data with very high precisions. At the end of this chapter, we provided an extensive evaluation of *STEER* on four different data lakes with different characteristics. These data lakes vary in semantic types and cover a wide spectrum of numerical and non-numerical data types. Furthermore, we also showed that our approach can be used across models that implement different learning approaches. Overall, each scenario in our experiments demonstrated that *STEER* can generate training data that allows a learned model to provide high performance in the data lake where it is to be deployed. To illustrate the benefits of our *Steered-Labeling* method, we made a comparison between

training data generated with and without the steered-labeling procedure. The results showed that with *Steered-Labeling* numerical columns can be labeled more accurately.

Nevertheless, in a more detailed analysis, we observed that the semantic types of the numerical columns are poorly predicted by the existing semantic type detection models (e.g., Sherlock [50], Sato [113], Doduo [100], Turl [20]). In Chapter 4, we delved into this aspect and uncovered a key reason for the limitations in existing models when predicting the semantic type of numerical columns. This limitation arises from the fact that the corpora utilized for designing, constructing and training these models mainly consist of textual data and contain almost no numerical data. Therefore, the models are primarily not designed to process numerical data. We analyzed all existing corpora for the task of semantic type detection of table columns and indicated as a result that essentially the following shortcomings are present: (i) tables incorporate either only or a very high percentage of textual data columns, (ii) tables are very small on average and often contain only very few numbers of columns (in most cases even less than 2 columns on average) or only very few number of rows. Compared to real enterprise data lakes, these characteristics pose a substantial disparity, where many numerical columns and large tables with many columns and rows exist. Hence, we presented our new corpus for semantic type detection called SportsTables in this chapter. SportsTables is the first corpus with annotated table columns, which contains a significantly larger proportion of numerical data than textual data. In total, the tables in our corpus have on average about 3 textual and 18 numerical columns. Moreover, the tables in our new corpus are much larger in both the number of columns and the number of rows than in existing corpora which better reflects the characteristics of real-world tables.

In Chapter 5 we introduced the last contribution of this dissertation - our new semantic type detection approach *Pythagoras*, specifically designed to support numerical along with non-numerical columns. *Pythagoras* can not only predict the semantic type of non-numerical table columns with high accuracy but also of numerical table columns. To achieve this, the main idea of the new model architecture is to use a GNN together with a new graph representation of tables and their columns. In the chapter, we first introduced our new graph representation of tables, where we use directed edges to model the information flow within tables. Using this graph representation, *STEER* can learn selectively which context information should be taken into account to establish robust predictions on numerical data. Afterward, we introduced our new GNN-based neural network architecture that is able to use our new graph data structure as input and predict the semantic type of table columns. Finally, in this chapter, we showed the effectiveness of the graph representation and the model architecture of *Pythagoras* by comparing against

five existing state-of-the-art semantic type detection models on two different data lakes that mimic the data distribution of enterprise data lakes and contain tables with non-numerical and numerical columns. The results of this experiment demonstrated that our new model outperforms all baselines on numerical data significantly. To summarize, this thesis contributes important steps towards making automatic semantic type extraction of table columns stored in data lakes feasible and thus made a meaningful progress in solving the data discovery problem in data lakes.

6.2 Future Research Directions

Although the contributions of this dissertation enhanced the deployment of semantic type extraction of table columns stored in enterprise data lakes, there are many more challenges and interesting research directions to address the broader challenge of data discovery in data lakes.

6.2.1 Out-Of-Distribution Identification & Human in the Loop

In this dissertation, we presented our labeling framework *STEER*, which is used for an automatic adaptation of a semantic type extraction model to a new data lake containing new unseen data characteristics and semantics. However, thinking about a running environment of a data lake, where new data sources with new data characteristics and semantics are added ad-hoc, the possible resulting adaptation of the model must be triggered manually with the use of *STEER*. Hence, it would be beneficial if the semantic type extraction approach itself could recognize when an adaptation is required. For example, through the ability of the approach to recognize unknown data patterns or semantic types of data columns in the data lake and, in this case, not to label them semantically but to flag this occurrence as out-of-distribution. Such a procedure can also be enhanced by adding a human in the loop to achieve continuous adaptation of the model. Thereby, the semantic type extraction approach would involve the human in the loop whenever it cannot detect the semantic type of a column and needs new input to adapt what it has learned. Developing a solution based on this concept will be exciting for future work. One existing research that goes in this direction is AdaTyper [49] which can be further evaluated and improved to build such an approach.

Another interesting future research would be to investigate how well Retrieval Augmented Generation (RAG) techniques [64] can be used to facilitate the adaptation of semantic type detection models to unseen data. RAG is currently considered a state-of-

the-art approach to provide LLM models with up-to-date information to answer questions for which the answer is not available in the training data of the LLM. Thus, it is not necessary to constantly re-train the model to provide it with new information or to adapt it to different data. Through the retrieval step in RAG, all relevant information from external sources are provided to the model as context information during the prediction. In this way, the model can use the latest information to make a prediction without re-training. How the RAG technique can be used for the task of semantic type extraction of table columns would be an interesting question for future work.

6.2.2 Extract Relationships Between Table Columns

Semantic types of table columns are not the only interesting metadata of tabular data that are helpful for an easier navigation in the data lake. Relationships between columns can also be crucial meta information for data discovery (or other data management tasks like data cleaning, data quality control, etc.) in data lakes. For example, semantic column relation types such as "is_livin_in" can connect two columns with the semantic types "person.name" and "city" with each other and thus provide relevant meta information helping to better understand the semantics of the table entries. Therefore, in a holistic metadata management system for data lakes, it will be important to have components that can automatically extract both (column types and column relation types) metadata information from tables. Approaches for this task already exist in the research field. For example, TURL [20] used its pre-trained transformer-based model for tables and fine-tuned it to the task of column relation extraction. Furthermore, Doduo [100] presented a unified model for predicting column semantic types as well as column relation types since both types are interdependent and unified approaches can be therefore beneficial. Although the presented results of these existing approaches for relation extraction are promising, they have the same limitations as the approaches for columns semantic extraction as presented in Section 1.3. In particular, it is unclear how the existing approaches perform on numerical table columns. Both approaches of [20, 100] were mainly evaluated on textual data, which is, as with the column semantic type extraction, much easier than on numerical data. For this reason, it is important to evaluate in future research work how well the existing approaches for semantic relation extraction work on numerical data. To investigate this and do experiments, we first see the need to provide a suitable corpus. To the best of our knowledge, there is currently no corpus for this task that contains a sufficient number of numerical columns. One possibility to provide such a corpus would be to expand our SportsTables corpus with annotations of

semantic relations between the table columns. If the findings of this investigation reveal a necessity for a new model specifically tailored for numerical columns, one potential avenue of future work could be to utilize our *Pythagoras* model for this purpose. In our graph representation of tables, the relationships between textual and numerical columns are already represented by edges and it would only be necessary to extend this approach to include all relationships between the table columns. Subsequently, the same GNN model could be used to implement an edge classification (instead of the current node classification) to predict the semantic type of the relation between columns.

6.2.3 Metadata Extraction Beyond Tabular Data

While this dissertation has made significant contributions for automatic semantic type extraction of tabular data (table columns) within data lakes, there is still a need to extend the same capability to other data modalities. Data lakes not only contain structured data such as tables but also other data modalities like text, images, audio, video, time series and various other forms of data. To comprehensively overcome the challenge of data discovery, it will be also necessary to enable automatic metadata extraction for these data modalities in data lakes. Such solutions would significantly enhance the holistic understanding and will lead to an effective utilization of these vast reservoirs of information. Generating automatic semantic annotations of images or semantic segmentation of images are well-known tasks and there exists a lot of research work in this field, such as [39, 65, 97, 108]. However, the central research question is how to integrate these existing models for semantic annotation of images into the metadata management of data lakes. Especially in the case of having a separate model for each modality, there will be a need to standardize the semantic types or labels across models. This is not only necessary initially but permanently over the entire lifetime of the data lake where new data is constantly being added or updated. Thus, another possible avenue for future work might be to implement a unified model for metadata extraction across all modalities, resulting in a single holistic model. A promising way to build such a multimodal model for metadata extraction could be the use of LLMs. Modern LLMs such as GPT4 [83], Luminous [32] and Gemini [87] are already designed to process and understand diverse data types, exhibit capabilities to handle multimodal inputs effectively [109]. By leveraging the inherent ability of LLMs to understand text, images and possibly other modalities, a unified framework for the automatic extraction of metadata in different data formats within data lakes could be created.

Part II

Peer-Reviewed Publications

7 Towards Learned Metadata Extraction for Data Lakes

Abstract

An important task for enabling the efficient exploration of available data in a data lake is to annotate semantic type information to the available data sources. In order to reduce the manual overhead of annotation, learned approaches for automatic metadata extraction on structured data sources have been proposed recently. While initial results of these learned approaches seem promising, it is still not clear how well these approaches can generalize to new unseen data in real-world data lakes. In this paper, we aim to tackle this question and as a first contribution show the result of a study when applying Sato — a recent approach based on deep learning — to a real-world data set. In our study we show that Sato without re-training is only able to extract semantic data types for about 10% of the columns of the real-world data set. These results show the general limitation of deep learning approaches which often provide near-perfect performance on available training and testing data but fail in real settings since training data and real data often strongly vary. Hence, as a second contribution we propose a new direction of using weak supervision and present results of an initial prototype we built to generate labeled training data with low manual efforts to improve the performance of learned semantic type extraction approaches on new unseen data sets.

Bibliographic Information

The content of this chapter was previously published in the peer-reviewed work Sven Langenecker, Christoph Sturm, Christian Schalles, and Carsten Binnig. "Towards Learned Metadata Extraction for Data Lakes." In: *Datenbanksysteme für Business, Technologie und Web (BTW 2021), 19. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme" (DBIS), 13.-17. September 2021, Dresden, Germany, Proceedings.* Ed. by Kai-Uwe Sattler, Melanie Herschel, and Wolfgang Lehner. Vol. P-311. LNI. Gesellschaft für Informatik, Bonn, 2021, pp. 325–336. DOI: 10.18420/BTW2021-17. URL: https://doi.org/10.18420/btw2021-17. The contributions of the author of this dissertation are summarized in Chapter 2.

7.1 Introduction

Motivation:. Data lakes are today widely being used to manage the vast amounts of heterogeneous data sources in enterprises. Different from classical data warehouses, the idea of data lakes is that data does not need to be organized and cleaned upfront when data is loaded into the warehouse [24]. Instead, data lakes follow a more "lazy" approach that allows enterprises to store any available data in its raw form. This raw data is organized and cleaned once it is needed for a down stream task such as data mining or building machine learning models. However, due to the sheer size of data in data lakes and the absence (or incompleteness) of a comprehensive schema, data discovery in a data lake has become an important problem [72, 77, 94].

One way to address the data discovery problem, is to build data catalogs that allow users to browse the available data sources [78]. However, building such a data catalog manually would again pose high effort since metadata needs to be annotated on data sources. An important task for cataloging structured (table-like) data in a data lake (e.g.,, originating from CSV files) is to derive semantic type information for the different columns of a data set. The reason is that this information is often missing in many data sources or the column labels available in data sources are not really helpful for data discovery since they use technical names or have been annotated from users with a different background.

In order to tackle the problem of extracting semantic data types from structured data sources in data lakes, recently learned approaches for metadata extraction have been proposed [13, 50, 113]. The main idea of these learned approaches is that they use a deep learning model for semantic type detection where the models are trained on massive table corpora with already annotated columns. While initial results of these learned approaches seem promising, it is still not clear how well these approaches can deal with the variety of data in real data lakes.

Contributions:. In this paper, we aim to tackle this question and report on our initial results of analyzing the quality of the state-of-the-art learned approaches for metadata extraction on real-world data. Moreover, we also show initial results of a new direction of tackling the open problems of the learned approaches that we discovered in our analyses. In the following, we discuss the two main contributions of this paper.

As a first contribution, we show the result of a study when applying Sato [113] - a recent approach based on deep learning to extract semantic types - to a real-world data set. A inherent problem of deep learning-based approaches for semantic type extraction is that they rely on a representative training data set; i.e., a set of columns with labeled semantic types. Otherwise, if the training data set does not cover the broad spectrum of data characteristics and types, the performance of the learned models quickly degrades when applied to a new data set. In fact, we show that Sato without re-training was only able to extract semantic data types for about 10% of the columns on the data sets used in our study.

As a second contribution, we thus suggest to take a new direction for learned metadata extraction to tackle the shortcomings of the existing deep learning approaches. As mentioned before, the root cause of why existing approaches often fail to extract semantic types is that the training data of the learned approaches is too narrow and thus the performance on new data sets is often poor. Hence, in this paper we propose a new direction of using weak supervision to generate a much broader set of labeled training data for semantic type detection on the new data set. Our initial results show that our approach can significantly boost the performance of deep learning-based approaches such as Sato when re-training these approaches on the additional synthesized training data.

Outline:. In Section 7.2, we first provide an overview of approaches for metadata extraction from structured data in data lakes. Afterwards, we discuss the results of our study of using Sato as a recent learned approach on a real-world data set in Section 7.3. Moreover, we then discuss our new approach based on weak supervision in Section 7.4. Finally, we present the initial results of using our current prototype in Section 7.5 before we conclude in Section 7.6.

7.2 Overview of Existing Approaches

In the following, we give a short overview of selected existing approaches for metadata extraction. We first discuss approaches for semantic type extraction before we briefly summarize recent approaches for the extraction of relationships.

7.2.1 Extraction of Semantic Types

Approaches that automatically extract types from metadata of data sources are already well established in industry. Prominent examples are Azure Data Catalog [8], AWS Glue [7] and GOODS [38]. In addition, many other research efforts exist for developing generic metadata models and special algorithms for metadata extraction (e.g., [90]).

All these approaches rely on the fact that basic metadata information is already annotated in the data source (e.g.,, as a header row in a CSV file) such as column and table names. However, header rows exist only in few cases and even when they do, the attribute names are not always useful as a semantic type. In this case, existing systems opt for manual metadata annotation.

Considering the huge amount of heterogeneous, independent, quickly changing data sources of real-world data lakes these approaches reach their limit. Therefore, some systems aim to detect semantic types from the columns content instead of relying on already existing labels in the sources. For this purpose, there exist two main research directions for automatic semantic type detection: search-based approaches and learning-based approaches.

Search-based Approaches:. The main idea of search-based approaches is to use external information to annotate semantic information to data sets. One approach in this direction is AUTOTYPE [110] which searches for existing custom extraction code to handle more specific (domain dependent) semantic data types. By helping developers to find and extract existing type detection code the supported semantic types of AUTOTYPE can be extended semi-automatically.

Learning-based Approaches:. In contrast to search-based approaches, learning-based approaches aim to build a machine learning model that can derive semantic types of columns from example data and not from extraction code. An early approach in this class is [67] which uses machine learning techniques to annotate web tables and their columns with types. While this approach relies on graphical models for extracting semantic labels for columns, more recent approaches such as [50, 113] are based on a deep neural network.

Sato [113] which is the successor of Sherlock [50] thus requires training data with labeled semantic types. While Sherlock only uses the individual column values as features

for predicting the semantic type, Sato also uses context signals from other columns in the table to predict the semantic type of a given column.

7.2.2 Extraction of Relationships

While extraction of semantic types is one important direction for metadata extraction, there also exist other approaches that are able to derive relationships between datasets (e.g.,, an author *writes* a book) from data automatically. One prominent example for such an approach is AURUM [12]. While AURUM represents an overall system for building, maintaining and querying an enterprise knowledge graph for available data sources, SEMPROP [13] is the subsystem of AURUM, which automatically derives links (i.e., relationships) between the data sources using word embeddings.

As mentioned before, different from this work and similar to Sato in this paper we focus on the extraction of semantic types for structured data sets. Hence, in the following sections we will limit the analysis of learned approaches to this direction. However, extending our approach towards relationship extraction is an interesting avenue of future work.

7.3 Study of Using Learned Approaches

In the following, we present the results of our study of using learned semantic type extraction approaches on real-world data. For our study, we selected Sato [113] as a recent approach based on deep learning.

7.3.1 Data Sets and Methodology

Data Sets:. As a data set in this study, we use the Public BI Benchmark[1] data corpus. The data corpus contains real-world data, extracted from the 46 biggest public workbooks in Tableau Public[2]. In this corpus there are 206 tables each with 13 to 401 columns. The main reason for choosing this corpus for our study was that it contains labeled structured data from different real-world sources across various domains (e.g., geographic, baseball, health, railway, taxes, social media, real estate). Hence, the benchmark comes with a high diversity and heterogeneity of data sources that can typically also be found in data

[1] https://github.com/bogdanghita/public_bi_benchmark-master_project
[2] https://public.tableau.com

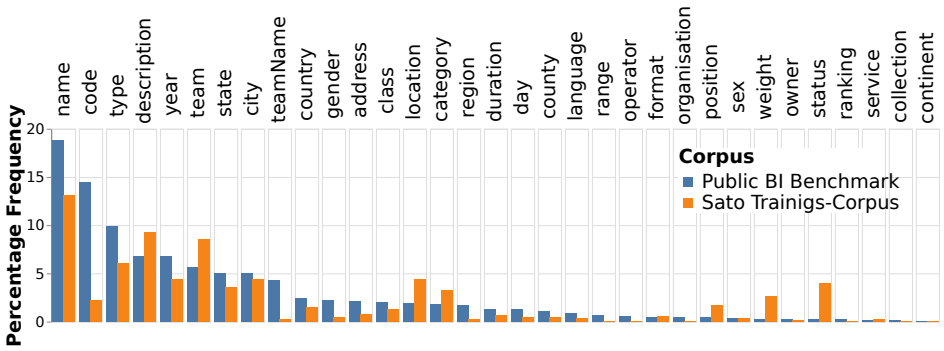


Figure 7.1: Distribution of semantic types in training data of Sato and the Public BI Benchmark

lakes of enterprises today.

Methodology:. As mentioned before, the inherent problem of deep learning-based approaches for semantic type extraction is that they rely on a representative training data set. To put it differently, if the training data set does not cover the variety of cases that are also seen in the real-world data, the performance of the learned models quickly degrades. As part of our analysis, we wanted to see to which extent this inherent limitation influences the overall quality of a learned approach such as Sato.

For the study, we thus annotated the data in the Public BI Benchmark manually with the correct semantic types of Sato. For the annotation, we first preprocessed the data automatically and searched for string matches between the column headers of the tables in the Public BI Benchmark and the semantic types supported by Sato. To guarantee the correctness of labels every column was additionally inspected and missing types where added manually.

7.3.2 Results of the Study

As a first question, we analyzed the coverage rate of the 78 semantic types supported by Sato in the Public BI Benchmark to see to which extend a pre-trained model can support real-world data if no new training data is used for re-training. For this question, we analyzed what fraction of columns in the Public BI Benchmark had a type that was covered by the training data set of Sato. The main result of this analysis was that only 10.6% of the columns are assignable to one of the semantic types.

As a second question, for the columns of the Public BI Benchmark that have types which are supported by Sato, we then wanted to see how the distribution of the 78 semantic types in the training data used for Sato and the Public BI Benchmark look like. The reason is that different distribution of labels in the training and testing data can have a negative impact on the overall quality of a learned approach. As can be seen in Figure 7.1, the frequency for many semantic types in both data sets (i.e., original training data of Sato and the Public BI Benchmark), however, is almost identical.

As a final question, we thus aimed to analyze the 10.6% of the columns in the Public BI Benchmark that are in principle covered by the training data of Sato. For this, we used the pre-trained Sato model and applied it to only this fraction of the data of the Public BI Benchmark. For this subset, Sato achieves an F_1 score (macro average and weighted[3]) of 0.090 and 0.300 respectively, which is also shown in Table 7.1 in our evaluation in Section 7.5. The original paper [113] reports an F_1 score of 0.735 and 0.925 on the VizNet[4] data corpus. This indicates that the data characteristics of the supported data types of the Public BI Benchmark is different from the data characteristics of the training data of VizNet and thus Sato can not infer types in a robust manner (even if they should be supported in principle).

Main Insights:. As suspected, our study has shown that a deep model such as Sato trained on one data set can only cover a fraction of data types of a new data set. Moreover, for the overlapping data types, the accuracy is still pretty low due to different data characteristics of the training data and the new data set. While the results of our study are specific to Sato, we believe that our findings are much more general and translatable to any learned approach that relies on manually curated training data (which is inherently limited as discussed before). Hence, a new approach is required where one can easily adapt learning-based models for type extractors to new data sets that covers types and data characteristics not covered in the available manually labeled training data. As a solution for this requirement, we next present our new weak supervision approach in the next section.

[3] **F_1 score macro average**: averaging the unweighted mean F_1 score per label
F_1 score weighted average: averaging the support-weighted mean F_1 score per label
[4] https://github.com/mitmedialab/viznet

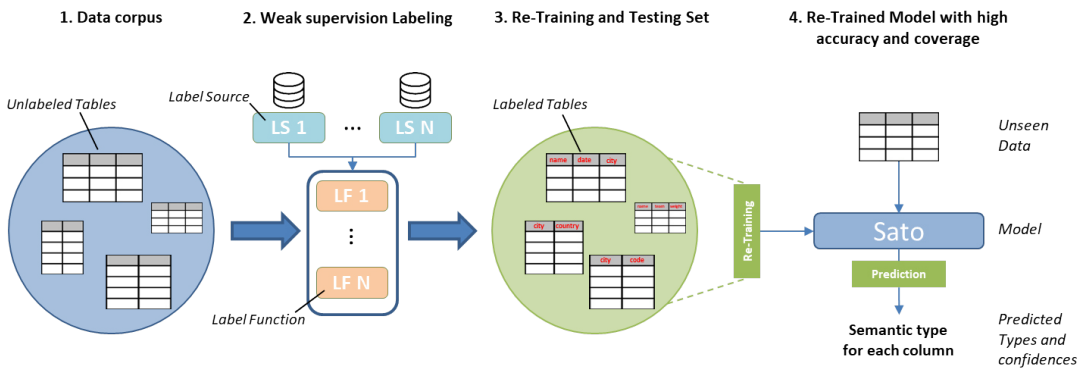


Figure 7.2: Concept and step-by-step procedure of our weak supervision approach

7.4 Weak Supervision for Semantic Type Extraction

The root cause of why deep learning-based approaches such as Sato often fail to extract semantic types on a new data set is that the training data lacks generality as discussed before. The main idea of using weak supervision is to generate a broad set of labeled training data with only minimal manual effort and thus increase the robustness when applying a learned approach such as Sato to a new data set. In the following, we discuss our initial ideas for such an approach and present the first results of our prototype to showcase its potential.

7.4.1 Overview of Our Approach

Figure 7.2 shows an overview of our approach. The main idea is that based on a set of simple labeling functions, we generate new (potentially noisy) training data that is then used to re-train a model such as Sato to increase the coverage of data types and data characteristics of the learned model. In other words, we apply the ideas of data programming discussed in [92] for the domain of semantic type extraction.

For generating new training data in our approach, we differentiate between two different classes of labeling functions: (1) The first class are labeling functions that can generate labels (i.e., semantic types) for completely new semantic types in a data lake that are not yet covered by a manually labeled training data set. Labeling functions of this class can be, for example, regular expressions, dictionary lookups, or other techniques such

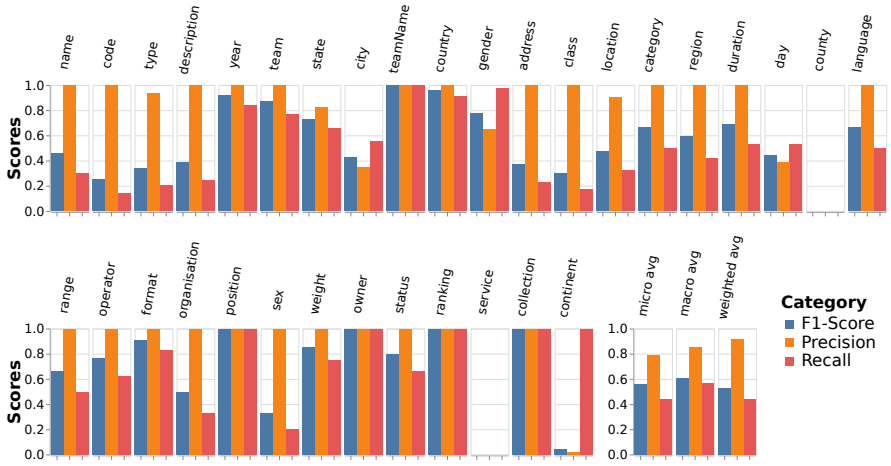


Figure 7.3: Performance of clustering semantically similar columns

as using alignment with existing ontologies. (2) Second, as we have seen in our study, another problem of learned approaches such as Sato is that they often fail to predict semantic types even if in principle the semantic type is already covered by the training data. The main reason for this case is that the training data does not cover the wide spectrum of data characteristics that might appear in a new data set. Hence, as a second class of labeling functions we support functions that can generate new labeled columns that cover more data characteristics (e.g., new values) for data types that are already available in a training data set. One idea for a labeling function of this class is the use of word embeddings [75] to cluster new unlabeled with already labeled columns and thus generate new labeled columns for existing semantic types. A more detailed description of such a labeling function is given below.

7.4.2 Label Generation using Clustering

For generating more labeled training data for an existing semantic type, we implemented a method based on clustering in our prototype system that we briefly introduced before. As mentioned, the main idea is that we can start with a small training corpus of labeled columns and by clustering new non-labeled to the labeled columns, we can derive new labeled training data.

To implement this labeling approach, we first compute column embeddings for labeled and unlabeled columns based on word embeddings of individual values. As word em-

beddings, we currently use *Google USE*[5] that was trained on 16 different languages and showed good results. But in principle we could also use other word embeddings. Based on the embeddings of individual values, we compute an embedding for all values of a column by calculating the average across the embeddings of all values which is the dominant approach for building representations of multi-words also mentioned in other papers [99]. This approach is reasonable also for us, since string-typed column values in the Public BI Benchmark are only composed of single values. In general, in the case the column values themselves consist of a sequence of words, we could also consider word embedding combining techniques as represented in [63] or [13].

Once we computed an embedding for all values of a column, we next carry out the clustering of labeled and unlabeled columns based on these embeddings. For this step, we use a agglomerative clustering algorithm[6]. In our prototype, we use this clustering method to not generate a fixed number of clusters, but to form groups based on the cosine similarity of vectors (i.e., our embeddings) and a distance threshold that we discuss below. Once clustered, we than compute a semantic type per cluster based on the majority vote of columns with the same label. [69] represents a system called Raha, which relies on a similar idea for generating training data but for error detection and not for semantic type extraction.

A key parameter to be set in our clustering approach is the distance threshold which can vary between 0.0 and 1.0 (i.e., a lower value means that we produce more clusters). In our experiments, we used a threshold of 0.1 based on a hyper-parameter search on the already labeled columns. This threshold provided high accuracy on the broad spectrum of data sets in the Public BI Benchmark.

Initial Results:. To analyze if the basic idea of clustering is working, we conducted a small experiment where we measure how well the clustering approach works on the Public BI Benchmark using our annotations of the 78 Sato types. By clustering, we wanted to see whether columns with the same type would be assigned to the same cluster. As we see in Figure 7.3, with a few exceptions, the clustering algorithm achieves high precision. This means that there is a very high probability that all elements in one cluster belong to the semantic type representing the specific cluster. For many types, we achieve an

[5] https://tfhub.dev/google/universal-sentence-encoder-multilingual-large/3
[6] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html#sklearn.cluster.AgglomerativeClustering

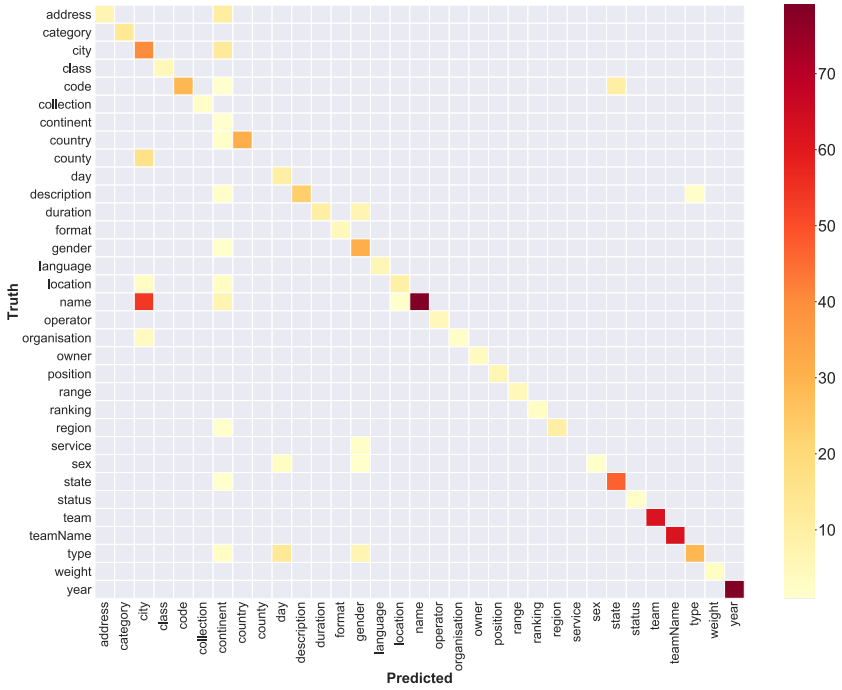


Figure 7.4: Confusion matrix of the clustering method

F_1 score of 1.0 such as for the semantic types *teamName*, *position*, *owner*, *ranking* and *collection*.

Moreover, in a second experiment, we wanted to show the robustness of our clustering approach to different data characteristics. For showing this, we analyzed the entropy and the jaccard-coefficient for all columns with the same semantic type in the Public BI Benchmark. The intuition is that columns with a high entropy (i.e., a high degree of divergence) or pairs of columns which have a low jaccard-coefficient (i.e., where columns values are not overlapping) are harder to cluster. Overall, our approach assigns column pairs with the same semantic type to the very same cluster even if they strongly vary in the entropy or have a low overlap (i.e., a low jaccard-coefficient). Unfortunately, due to space limitations we could not add further details about this experiment to the paper.

7.4.3 Future Directions

As mentioned before, in this paper we showed only a very first prototype where we apply the idea of weak supervision for synthesizing labeled training data for semantic type extraction. In our first prototype, we only covered the labeling approach based on

	Macro average F_1	Precision	Recall	Support-weighted F_1
SHERLOCK (not re-trained)	0.114	0.375	0.309	0.322
SHERLOCK (re-trained)	0.806	0.879	0.859	0.860
SATO (not re-trained)	0.090	0.322	0.304	0.300
SATO (re-trained)	**0.811**	**0.912**	**0.894**	**0.894**

Table 7.1: Performance comparison of the models on Public BI Benchmark

clustering as discussed before. Hence, the main avenue of future work is to extend this prototype and add a much broader set of labeling functions.

Furthermore, another direction is to study alternatives for the training data generation process. Currently, we directly use the potentially noisy training data generated by the labeling functions for re-training. Another possible direction as shown in [92], would be to first train a generative model that can learn how to generalize from the additional training data and thus mitigate the negative effects such as noisy data to a certain extent.

Finally, in the current state, we only consider semantic types whose data values are strings or types that provide a semantic meaning when converted to a string (such as *weights* and *dates*). The semantic type detection of numeric types such as *temperature* require additional labeling functions and therefore represent future research.

7.5 End-to-End Evaluation

In the section before, we have already shown that the basic idea of weak supervision can help to generate training data by clustering to improve the robustness w.r.t different data characteristics. In the following, we report on the initial results of using this approach in an end-to-end evaluation to show how this can boost the performance of learned type extraction approaches such as Sato.

Setup and Data Preparation:. We implemented our approach for automatic labeling in Python using the Google USE embeddings as mentioned before. Moreover, for training and evaluation, we used the source code provided by Sato[7]. However, Sato is designed to be built and trained from scratch. Hence, we extended Sato with the appropriate functionality for incremental re-training.

[7] https://github.com/megagonlabs/sato/tree/master

End-to-End Results:. For showing the end-to-end performance of our approach, we restricted ourselves to the 10% of the Public BI Benchmark data that is supported by Sato and its semantic types. For this, we first generated additional training data and then re-trained the pre-trained Sato model with our additionally labeled data. For generating additional training data, we used the clustering approach discussed before for the Public BI Benchmark. For this purpose, we split the Public BI Benchmark into a training and testing set.

As we see in Table 7.1, after re-training the Sato model with the synthesized training data of our approach, Sato achieves F_1 scores (macro average and weighted) of 0.811 and 0.89 respectively. This is a significant improvement of almost +0.60 compared to the performance of Sato without re-training. In addition to show that our approach also generalizes to other learned approaches, we furthermore used Sherlock [50] (without and with re-training). As shown in Table 7.1, this leads to a similar performance gain. In summary, these results show that our approach is in principle able to boost the performance of learning-based approaches that have been pre-trained on only a small training data set not covering all data characteristics found in a new unlabeled data set.

7.6 Conclusions

Detecting semantic types for columns of data sets stored in data lakes results in an enormous benefit building a data catalog to address the data discovery problem. While recent papers have shown initial results for learned approaches that can be used for extracting semantic types, they cannot support many real-world data sets since they only support a limited set of semantic data types as we have shown in our study. To tackle this problem, we suggested a new direction of using weak supervision for generating additional labeled training data and use this for re-training the existing learned model. An initial evaluation of our new direction using our current prototype shows that this approach can lead to huge performance gains.

8 SportsTables: A new Corpus for Semantic Type Detection

Abstract

Table corpora such as VizNet or TURL which contain annotated semantic types per column are important to build machine learning models for the task of automatic semantic type detection. However, there is a huge discrepancy between corpora that are used for training and testing since real-world data lakes contain a huge fraction of numerical data which are not present in existing corpora. Hence, in this paper, we introduce a new corpus that contains a much higher proportion of numerical columns than existing corpora. To reflect the distribution in real-world data lakes, our corpus SportsTables has on average approx. 86% numerical columns, posing new challenges to existing semantic type detection models which have mainly targeted non-numerical columns so far. To demonstrate this effect, we show the results of a first study using a state-of-the-art approach for semantic type detection on our new corpus and demonstrate significant performance differences in predicting semantic types for textual and numerical data.

Bibliographic Information

The content of this chapter was previously published in the peer-reviewed work Sven Langenecker, Christoph Sturm, Christian Schalles, and Carsten Binnig. "SportsTables: A new Corpus for Semantic Type Detection." In: *Datenbanksysteme für Business, Technologie und Web (BTW 2023), 20. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme" (DBIS), 06.-10, März 2023, Dresden, Germany, Proceedings*. Ed. by Birgitta König-Ries, Stefanie Scherzinger, Wolfgang Lehner, and Gottfried Vossen. Vol. P-331. LNI. Gesellschaft für Informatik e.V., 2023, pp. 995–1008. DOI: 10.18420/BTW2023-68. URL: https://doi.org/10.18420/BTW2023-68. The contributions of the author of this dissertation are summarized in Chapter 4.

8.1 Introduction

Semantic type detection is important for data lakes. Semantic type detection of table columns is an important task to exploit the large and constantly changing data collections residing in data lakes. However, manually annotating tables in data lakes comes at a high cost. Hence, in the past many approaches have been developed that automatically derive semantic types from table data [20, 50, 100, 113]. Many of the recent approaches use deep learning techniques to build semantic type detection models. As such, corpora containing large amounts of table data with assigned semantic types are required for training and validating. Existing annotated table corpora (e.g., VizNet, TURL) primarily contain tables extracted from the web and therefore limit the capability to represent enterprise data lakes.

Existing corpora and models fall short on real-world data lakes. However, as we can see in Figure 8.1, almost all existing corpora that provide annotated columns labeled with semantic types have a lack of table columns that contain numerical data, and tables in these datasets incorporate either only or a very high percentage of textual data. Only GitTables [47] contains a more balanced ratio of textual and numerical data. Nevertheless, compared to real enterprise data lakes, there is a significant discrepancy in the ratio of textual to numerical data. An inspection of a large real-world data lake at a company[1] has shown that on average approx. 20% textual data and 80% numerical data are present (see. Figure 8.1 bars on the right). Moreover, semantic type detection models [20, 50, 100, 113] that are trained on the available corpora also mainly target non-numerical data.

Semantic type detection for numerical data is challenging. Detecting semantic types of numerical columns is generally harder than for textual columns. For example, for a textual column with the values {Germany, USA, Sweden, ...} a model can easily identify the semantic type *country*. Instead, for a numeric column with e.g., the values {20,22,30,34,...} it is not that straightforward and several possibilities for a matching semantic type exist such as *age, temperature, size, money*. The fundamental reason here is that numerical values can be encoded with much fewer bits than string values [98], resulting in a lower overall entropy and thus providing less information content that can be used by a machine learning model to infer the underlying semantic type. Due to the existing corpora providing annotated columns that have been used to create and validate

[1]The analyses were done at the company LÄPPLE AG

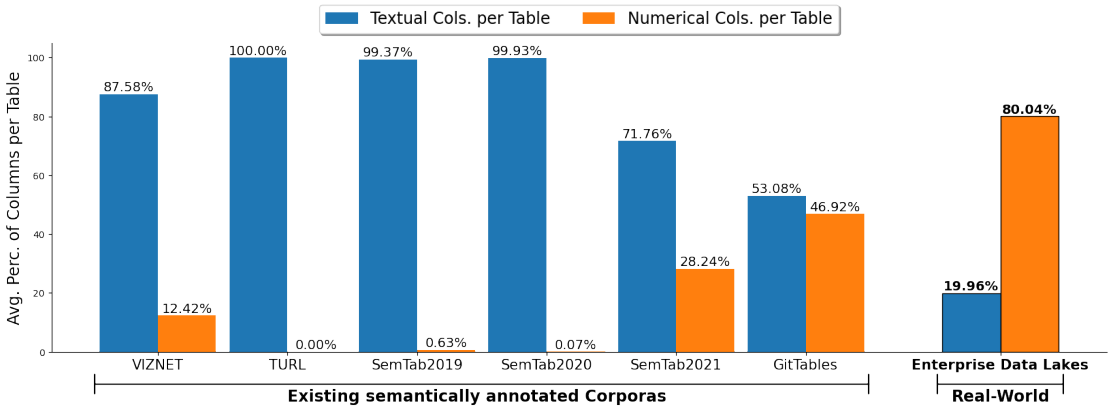


Figure 8.1: Average percentage of textual and numerical based columns per table in existing semantically annotated corpora[2] (left bars) compared to real-world data lakes (right bar). This shows the fact that there is a significant shift in the ratio of textual to numeric columns per table from existing corpora to real data lakes. Since all existing semantic type detection models were developed by using the existing corpora, shortcomings in validating the models on numerical data are present and it has not yet been studied in depth how well the models can perform on datasets containing a high proportion of numerical data.

semantic type detection models, we see several essential shortcomings that could not be addressed until now because of the absence of a sufficient dataset for this purpose.

Contributions. In this paper, we thus contribute a new corpus containing tables with semantically annotated columns with numeric and non-numeric columns that reflect the distribution of real-world data lakes. We will make the corpus available which should stimulate research directions such as working on new model architectures that can reliably annotate types to numeric and non-numerical columns. In the following, we discuss the main contribution of this paper.

As a first contribution, we present and provide our new corpus SportsTables[3]. To the best of our knowledge, SportsTables is the first corpus with annotated table columns, which contains a significantly larger proportion of numerical data than textual data. In

[2]Notice that for GitTables we only considered the tables and columns labeled by terms from DBpedia using the semantic annotation method as described in the GitTables paper. Therefore our reported ratios of textual and numerical data differ from those shown in the GitTables paper because they consider all data, whether annotated or not.

[3]Available on https://github.com/DHBWMosbachWI/SportsTables.git

total, the tables in our corpus have on average about 3 textual and 18 numerical columns. Moreover, the tables in our new corpus are much larger in both the number of columns and the number of rows than in existing corpora which better reflects the characteristics of real-world tables.

As a second contribution that comes together with the corpus, we specify an ontology with semantic types for the sports baseball, basketball, football, hockey, and soccer. This ontology provides fine granular semantic types for all kinds of sports we considered to build SportsTables and allows us to semantically describe each occurring table column, which is not possible with the current ontologies (e.g., DBpedia) at this level of detail. Using a manually created dictionary, we assign a semantic type to each existing column in SportsTables.

As a third contribution, we present our initial results of using our new corpus on Sato [113], a state-of-the-art semantic type detection model. Overall, we can see that when trained on our new corpora, Sato can improve the performance on numerical data types. However, one shortcoming that our analysis shows is that current model architectures are not targeting numerical columns. To be more precise, our analysis demonstrates that textual data columns are mostly correctly semantically interpreted with Sato (F1-Score of 1.0), but on numerical data columns, the model only achieves an F1-Score of about 0.55. This large difference indicates that new model architectures that take the characteristics of numerical columns into account are needed which is a direction that could be stimulated by the availability of our corpus.

Outline. In Section 2, we first provide an overview of existing corpora which was used to build and validate semantic type prediction models and discuss their characteristics and statistics. Afterward, in Section 3, we then introduce our new corpus SportsTables and describe in detail how we created the corpus and labeled the table columns with semantic types. Section 4 first demonstrates the main characteristics of our corpus before we then show the initial results of using our new corpus on Sato. Next, further research challenges are discussed in Section 5 before Section 6 concludes the paper.

Corpus	#Table	#Total Columns	Avg. #Columns per Table	Avg. #Rows per Table
VIZNET	78,733	120,609	1.53	18.35
TURL	406,706	654,670	1.61	12.79
SemTab2019	13,765	21,682	1.58	35.61
SemTab2020	131,253	190,494	1.45	9,19
SemTab2021	795	3,072	3.86	874.6
GitTables	1.37M	9.3M	6.82	184.66
SportsTables	**1,187**	**24,838**	**20.93**	**246.72**

Table 8.1: Corpus statistics about the number and sizes of tables.

8.2 Existing Corpora with Semantic Data Types

In the following, we describe different existing corpora that contain annotated table columns and therefore can be used to build and validate semantic column type detection models. We summarized the main statistics for all corpora in Table 8.1.

VizNet [45]. The original VizNet corpus [45] is a collection of data tables from diverse web sources ([11, 81, 88, 105]) which initially do not contain any semantic label annotation. The corpus we consider in this paper is a subset of the original VizNet corpus, which was annotated by a set of mapping rules from column headers to semantic types and then used to build and validate the Sherlock [50] and Sato[113] prediction model. The corpus contains in total 78,733 tables and 120,609 columns annotated with 78 unique semantic types. Overall, the tables in the corpus contain only 1.53 columns and 18.35 rows on average. Furthermore, the distribution of the column data types is 87.58% textual and 12.42% numerical and thus leads to the shortcomings as described before.

TURL [20]. The TURL corpus uses the WikiTable corpus [9] as basis. To label each column they refer to the semantic types defined in the Freebase ontology [34] with a total number of 255 different semantic types. What distinguishes TURL from other corpora is that columns can have multiple semantic types assigned. In total, there are 406,706 tables resulting in 654,670 columns, and on average a table consists of 1.61 columns and 12.79 rows. Again, these are rather small dimensions. In addition, the Turl corpus includes no numerical data at all, which leads to the shortcomings mentioned above when using the corpora.

SemTab. SemTab is a yearly challenge with the goal of benchmarking systems that match tabular data to knowledge graphs since 2019. The Challenge includes the tasks of assigning a semantic type to a column, matching a cell to an entity, and assigning a

property to the relationship between columns. Every year, the challenge provides different datasets to validate the participating systems against each other. In this paper we observed the provided corpora for the years 2019 [40], 2020 [19, 41], and 2021 [1, 19, 42, 46, 82]. Statistic details of the corpora are shown in Table 8.1. In case more than one dataset was provided per year, we aggregated the statistics over all datasets included in the challenge. While SemTab2019 consists of 13,765 tables and 21,682 columns in total, there are 131,253 tables and 190,494 columns in SemTab2020. In both corpora, the dimensions of the included tables are rather small (on average 1.58 columns and 35.61 rows in 2019 and 1.45 columns and 9.19 rows in 2020). In SemTab2021, the contained tables are the largest in terms of rows with almost 875 on average. However, the number of columns (3.86 on average) is only moderate and the corpus in general is the smallest with a total of 795 tables and 3,072 columns. Numerical data is almost nonexistent in the first two years (0.63% in 2019 / 0.07% in 2020), increasing to 28.24% numeric columns per table on average in 2021, which is still not comparable to the number of numeric data in real world data lakes.

GitTables [47]. GitTables is a large-scale corpus of relational tables created by extracting CSV files from GitHub repositories. Table columns are labeled with semantic types from Schema.org [35] and DBpedia [6] using two different automated annotation methods (syntactically/semantically similarity matching from semantic type to column header). In this paper, we have focused on the annotations origin from DBpedia and the results of the semantic annotations method as described in the GitTables paper [47]. This leads to a corpus containing over 1.37M tables and 9.3M columns in total. Although this is by far the largest collection of data tables, the dimensions of the tables are on average only moderate with 6.82 columns and 184,66 rows. Overall, GitTables incorporates the most numeric data with an almost balanced ratio of 53.08% textual and 46.92% numerical columns per table.

Discussion. The overview in Table 8.1 and the discussion before shows that most existing corpora contain no or only a minimal fraction of numerical data types which is very different from real-world data lakes. An exception is GitTables which has a much higher ratio of numerical columns. However, as we show in Section 8.4, GitTables still lacks a good coverage of different numeric semantic types which is one important aspect that we tackle with our new corpus SportsTables which covers a wide variety of different numerical semantic types. Moreover, another important (but orthogonal) aspect is that existing corpora include a large number of tables. However, on average the tables are very small in terms of the number of columns and the number of rows. Instead, our new

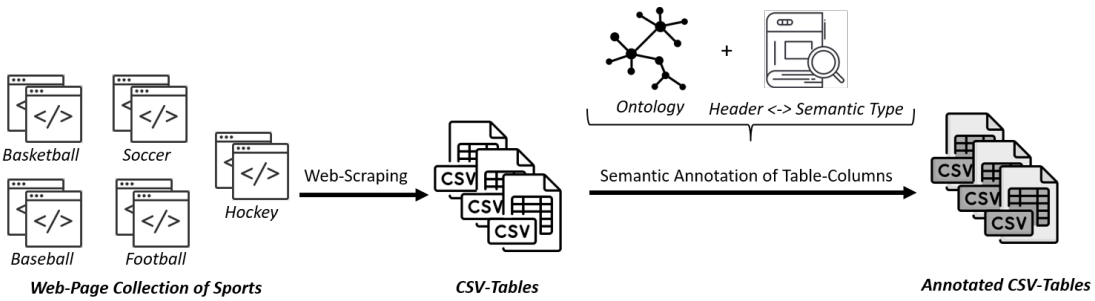


Figure 8.2: Overview of the implemented pipeline to build SportsTables. We use web-scraping techniques to extract HTML tables from a manually defined web page collection for each selected sport and convert the tables to CSV files. With the help of a defined ontology and a manually created dictionary that maps column headers to semantic types, we annotate each table column with an appropriate semantic type.

corpus SportsTables contains fewer tables, but on average a significantly higher number of columns and rows per table to better reflect the characteristics of real-world data lakes.

8.3 The SportsTables Corpus

In the following, we will introduce our new corpus and describe in detail the implemented construction pipeline to build SportsTables.

Methodology to generate corpus. Figure 8.2 gives an overview of our implemented pipeline to generate the new corpus. The main idea was to collect data tables from different sports domains such as soccer, basketball, baseball, etc. since data tables coming from such kinds of sources are rich in numerical columns. For example, a soccer player statistic table of a soccer season contains typically 3 textual columns (e.g.,, player name, team name, field position) and 18 numerical columns (e.g.,, goals, games played, assists). Hence, building a collection of such tables will lead to a corpus that contains many numerical columns which are in addition semantically interpretable. As a result, the corpus will enable to analyze the performance of semantic type prediction models in a much more rigorous manner regarding numerical data.

Scraping data from the web [23]. A vast amount of data covering information about player statistics, team statistics, coach statistics, or season rankings of different sports are available on various web pages. Therefore, for collecting the data, we built a data

Sports	#Table	#Total Cols	$\overline{\#Text.Cols/Table}$	$\overline{\#Num.Cols/Table}$	$\overline{\#Rows/Table}$
Baseball	174	3,829	3.97	18.03	76.34
Basketball	180	3,801	1.78	19.34	152.5
Football	303	6,764	2.45	19.88	354.79
Hockey	257	5,347	2.1	18.7	247.15
Soccer	273	5,097	3.9	14.77	297.11
Total	1,187	24,838	2.83	18.1	246.72

Table 8.2: Corpus statistics about the number and size of included tables. Statistics are shown broken down by individual sports taken into account and in total. Across all sports, the average number of numeric columns is much higher than textual columns.

collection pipeline based on web scraping technology[23]. In the first step, we manually searched and defined a set of different web pages for each of the selected sports of which we want to scrape contained data tables (left side of Figure 8.2). We first converted each HTML table on the web pages to Pandas-Dataframes using Python and then saved them as CSV files (center of Figure 8.2), since this file format is most known and used to store raw structured data [76]. During the scrape process, we kept the respective column headers from the original HTML table and used them as headers in the CSV file.

Annotating columns with semantic types. Due to the low granularity of existing ontologies (e.g., DBpedia) regarding semantics of a given sport, we manually created an ontology-like set of valid semantic types for all sports. For example, in DBpedia there is the type *Person.Athlete.BasketballPlayer*, but semantic labels in the particular that would match individual numerical columns such as *NumberOfGoals* are not defined. Next, we annotated all table columns with semantic types using a manually created dictionary that maps column headers to matching semantic types from our created set. Since the column headings were in many cases identical if the semantic content was the same, this procedure significantly reduces the manual labeling effort. In addition, to ensure that the labels are of very high quality in terms of correctness, we manually checked each assignment based on the content of the columns.

8.4 Analysis of the Corpus

This section describes the characteristics of SportsTables in detail and then demonstrates the significant impact of these characteristics on semantic type prediction frameworks in a small study where we apply the corpus to an existing type detection model.

8.4.1 Corpus Characteristics

In the following. we discuss the statistics of the SportsTables corpus and compare them to the existing corpora.

Data statistics (Table 8.1&Table 8.2). Using the described pipeline for creating SportsTables, a total of 1,187 tables which comprises 24,838 columns (approx. 86% numeric and 14% textual) are scraped from the web resulting in 20.93 columns (2.83 textual and 18.1 numerical) per table on average. This ratio of textual to numerical columns, as well as the total average number of columns in a table, differs significantly from existing corpora. To provide details about the contribution of different sports areas contained in SportsTables, Table 8.2 shows the main statistics by the individual areas of sports.

Figure 8.3 shows a comparison of the average number of textual and numerical columns per table of SportsTables versus that of the existing corpora. Here we can see that numerical columns only exist in the corpora VizNet with 0.33, SemTab2021 with 1.09, and GitTables with 3.2 columns per Table. Compared to GitTables, in SportsTables there are thus on average over 6 times more numeric columns per table. Moreover, as we discuss below, our corpus uses a much richer set of numerical data types that better reflects the characteristics in real-world data lakes which is very different from GitTables. For example, when looking at the semantic types that are assigned to numerical columns in GitTables, more than half (393,925) of the columns are labeled with just a single type *Id*.

In terms of the total number of columns, the tables in SportsTables (20.93 columns per table) are on average about 3 times wider than in GitTables (6.82 columns per table), which contains the widest tables among the existing corpora. As such, the number of columns in tables of SportsTables are reflecting better the width when comparing this to the characteristics of the tables in the real-world data lakes which we analyzed. Moreover, considering the average number of rows per table, it can be seen that the tables in SportsTables have on average 246.72 rows. In comparison, tables in SportsTables are larger on overage than in many other corpora where tables have typically fewer rows.

Annotation statistics. Semantic type annotation follows a two step process. First, we establish a directory with manually defined mappings from column header to semantic type for each existing header. Second, we label each column with the semantic type

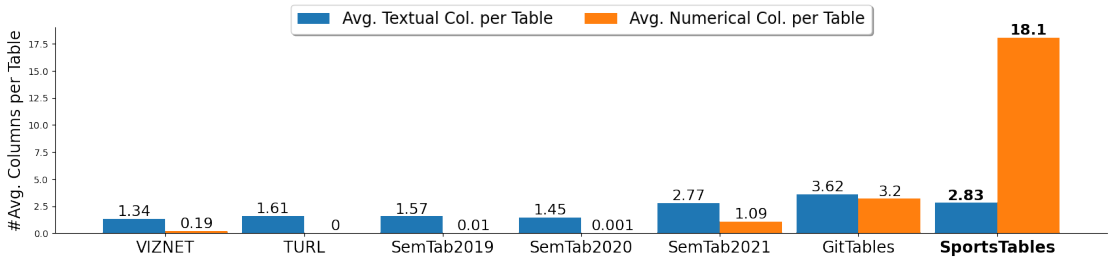


Figure 8.3: Average number of textual and numerical columns per table for each existing annotated corpora and our new SportsTables corpus. This shows the absence of numerical data columns per table in most existing corpora and the dominance of textual data columns per table in all existing corpora. Instead, our new corpus SportsTables contains on average over 6 times more numerical columns than textual columns.

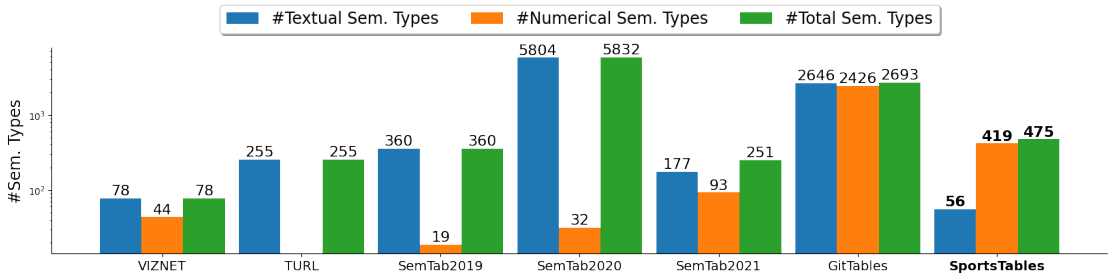
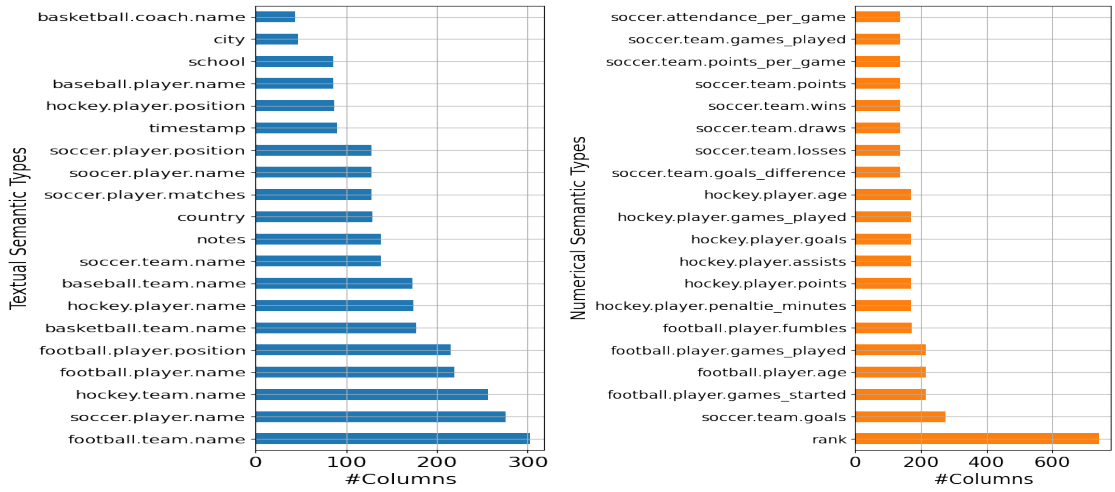


Figure 8.4: Corpus statistics about the number of unique semantic types included. Showing that our new corpus has a higher proportion of numerical semantic types than textual semantic types in contrast to the existing corpora. In addition, there is a large overlap of semantic types used for textual and numeric columns in the existing corpora. In comparison, the semantic types in SportsTables are disjoint for the two column data types.

listed in the directory for its header. As a result, 56 textual and 419 numerical semantic types are present in the corpus. Thereby textual semantic types are those which specify textual columns and numerical types are those which specify columns containing numeric values. To compare the annotation statistics, we also counted the number of textual and numerical semantic types in an analysis of the existing corpora. The results of these analyses can be seen in Figure 8.4. Different from our corpus, the sets of textual and numerical types are not disjoint in all other corpora (except TURL where no numeric values are present). This indicates that individual semantic types were assigned to both textual and numerical columns which is problematic if semantic type detection models should be trained and tested on these corpora. In particular, GitTables has a very large



(a) Top 20 textual sem. types

(b) Top 20 numerical sem. types

Figure 8.5: Semantic type annotation statistics of SportsTables. (a) Shows column annotation counts of the top 20 textual semantic types. Across all kinds of sports, *player.name* and *team.name* are the most common. (b) Shows column annotation counts of the top 20 numerical semantic types. A dominant type here is *rank*, which describes a column containing the placements of e.g., a team in a season standings table.

overlap and almost all semantic types are used in both column data types. To give an example, in GitTables the semantic types *comment, name* and *description* are assigned to both column data types. Next, we take a closer look into the semantic types of our corpus.

Figure 8.5a and Figure 8.5b shows the top 20 semantic types (textual and numerical) in regards to how often they were assigned to a table column. It can be seen that the most common textual types across all sports are *player.name* and *team.name*. These are types that occur in almost every table. Other types such as *country* or *city* are also common, describing, for example, the player's origin or the team's hometown. Among numeric semantic types, *rank* is by far the most common and is present in almost all tables. The type describes a column containing the placement of e.g.,, a team in a "seasons standing" table or a player in a "top scorer" table. All other numeric semantic types show mainly an equal distribution of the frequency, which is a good precondition for training machine learning models. In order to show not only the frequency of the top 20 semantic types, Figure 8.6 plots all semantic types (separated in textual and numerical) by the frequency of occurrences. Here we see that 19 textual and 66 numerical semantic types occur only once in the entire corpus. For the training and testing of prediction models, we would

suggest not considering these types due to the low occurrences.

SportsTables vs. GitTables. Since GitTables is the largest corpus with the most tables, one could argue that a subset of GitTables would result in a new corpus with similar characteristics as SportsTables. To analyze this, we executed a small experiment in which we filtered out only tables from GitTables where the number of textual and numerical columns (min. 3 textual and 18 numerical columns) is at least the same as it is in SportsTables. The result was a corpus containing a total of 16,909 tables and 743,432 columns. On average a table has 12.53 textual columns, 31.43 numerical columns, and 17.35 rows. However, looking at the semantic types that are assigned to numerical columns, more than half (393,925) of the columns are labeled with the type *Id*. In terms of training and validating semantic type detection models, this is rather an unfavorable type representing no semantically meaning. Moreover, the next 5 most common numerically based semantic types are *parent, max, comment, created* and *story editor*, constituting a large proportion of the columns. The assignment of these types to numerical data is slightly less understandable and indicates a lack of quality in the automatically generated labels for table columns.

8.4.2 An Initial Study of Using SportsTables

In the following, we report on the initial results of using Sato, a recent semantic type detection model, on our new corpus. With this, we want to measure how well the semantic types in our corpus can be inferred by the model with a special focus on how it performs on textual and numerical columns.

Experiment setup. For the first experiments, we only considered the soccer data from SportsTables. Thereby, we split the corpus into different sizes of train and test sets (5/95, 10/90, 15/85, 20/80), to show the results of scenarios where the model has less and more training data available. We use the pre-trained Sato model, which was trained on the VizNet corpus, and re-trained it with the different training set sizes. During re-training, we replaced the last layer of Sato to support the number of semantic types that occur in SportsTables and then re-trained the entire neural network. To measure the performance, we applied the re-trained model to the corresponding test data set.

Results of study. Figure 8.7 shows the results of the experiments reporting F1-Scores using the defined different sizes of train and test splits as described before. We plot macro

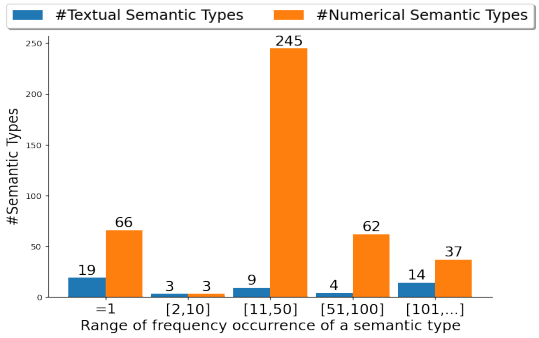


Figure 8.6: Shows how often semantic types occur in SportsTables using buckets of varying widths, which represents the frequency of occurrences. For example, 19 textual and 66 numerical types occur only once in the entire corpus.

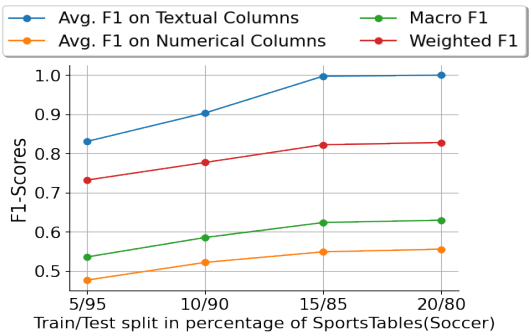


Figure 8.7: Initial results using the Sato model on our new SportsTables corpus with different train/test split sizes. The differences in F1-Scores for predicting textual and numeric columns indicate that the model can handle textual data more effectively than numeric data.

and weighted average F1-Score across all semantic types to show the total performance, but also separate average F1-Score for only textually and numerically based semantic types, respectively. As we can see in the figure, while Sato can detect numerical types, there is a significant performance difference between predicting textual and numerical semantic types for all setups. At the data split 20/80, all textual columns can be predicted correctly by the model, whereas for numerical columns only an F1-Score of 0.56 is achieved. On average, the difference in F1-Score between textual and numeric types is 0.41 across all setups. These results indicate that the model is more able to handle textual data and determine the associated semantic type more accurately than for numeric data. Across all semantic types, the weighted F1-Score increase from 0.73 to 0.82 while the macro F1-Score range from 0.53 to 0.63, which are rather moderate score values for semantic type prediction models.

8.5 Further Research Challenges

Detecting semantic types in real-world data lakes comes with many more challenges that need to be addressed. In particular, based on our findings of the analysis using Sato in Section 8.4, we think that new model architectures are needed for detecting numerical

data types which have very different characteristics from non-numerical data. In the following, we list some of the challenges we think are important to be addressed. We hope that our corpus enables research on those challenges.

Embedding numerical data: Most state-of-the-art models apply language models like BERT [22] to encode literals to infer the semantic type of a table column. Since such approaches are optimized for textual data, the performance on numerical data of such models is not entirely analyzable with the existing corpora.

Leveraging numerical context: To improve the semantic type prediction of a table column, recent approaches like Sato [113], TURL [20] and Doduo [100] incorporate also context information like the table-topic or values from neighboring columns of the same table. Given that tables in existing corpora contain almost entirely textual columns, the contexts (e.g., values from neighboring columns) used are rich in information and therefore also lead to performance improvements. However, it is unclear how effective this approach is in case the tables contain many numerical columns and only a few textual columns since the context information provided is reduced due to the lower entropy of numeric values as described before.

Supporting wide tables: Existing datasets for semantic type detection consist of tables with small numbers of columns and rows. In nearly all corpora, the existing tables contain on average less than two columns and less than 40 rows (see Table 8.1). Therefore, at the current state, it has not been analyzed how state-of-the-art models can handle such large tables. To give an example of why large tables could be a problem for recent models, we will briefly discuss Doduo[100]. Doduo uses pre-trained language models (e.g.,, BERT) and hence they have to convert the entire table into token sequences with a fixed tensor length of 512 elements so that the table and its entries can be meaningfully processed by the language model. To accomplish this, Doduo serializes the complete table and its entries as follows: for each table that has n columns $T = (c_i)_{i=1}^{n}$, where each column has N_m column values $c_i = (v_i^j)_{j=1}^{m}$, they let $serialize(T) ::= [CLS]v_1^1...[CLS]v_1^n...v_m^n[SEP]$, where the special token [CLS] marks the beginning of a new table column and [SEP] the end of a token sequence. With this methodology of serialization and the fixed given tensor length, increasing the number of table columns means that decreasing number of values of each column can be included for serialization. For example, a table with 512 columns would allow only one value per column to be considered and this would most

likely result in an insufficient semantic representation of the column based on that one value.

8.6 Conclusion

Existing corpora for training and validating semantic type detection models mainly contain tables with only or a very high proportion of textual data columns and no or just a limited number of numerical data columns. Therefore, it has not been studied precisely how well state-of-the-art models perform on a dataset with a very high percentage of numerical columns as it occurs in real-world data lakes. Moreover, tables in existing corpora are very small regarding the total number of columns and rows. To tackle these shortcomings, we built a new corpus called SportsTables which contains tables that have on average approx. 3 textual columns, 18 numerical columns, and 250 rows. With our new corpus, semantic type detection models for table columns can now be holistically validated against numerical data. We show initial results by using Sato – a state-of-the-art model – on our new corpus and report significant differences in the performance of predicting semantic types of textual data and numerical data. The corpus is available on https://github.com/DHBWMosbachWI/SportsTables.git. Finally, we think that the corpus is just a first step to stimulate more research on new model architectures that can better deal with numerical and non-numerical data types.

9 Steered Training Data Generation for Learned Semantic Type Detection

Abstract

In this paper, we introduce *STEER* to adapt learned semantic type extraction approaches to a new, unseen data lake. *STEER* provides a data programming framework for semantic labeling which is used to generate new labeled training data with minimal overhead. At its core, *STEER* comes with a novel training data generation procedure called Steered-Labeling that can generate high quality training data not only for non-numeric but also for numerical columns. With this generated training data *STEER* is able to fine-tune existing learned semantic type extraction models. We evaluate our approach on four different data lakes and show that we can significantly improve the performance of two different types of learned models across all data lakes.

Bibliographic Information

The content of this chapter was previously published in the peer-reviewed work Sven Langenecker, Christoph Sturm, Christian Schalles, and Carsten Binnig. "Steered Training Data Generation for Learned Semantic Type Detection." In: *Proc. ACM Manag. Data* 1.2 (2023), 201:1–201:25. DOI: 10.1145/3589786. URL: https://doi.org/10.1145/3589786. The contributions of the author of this dissertation are summarized in Chapter 3.

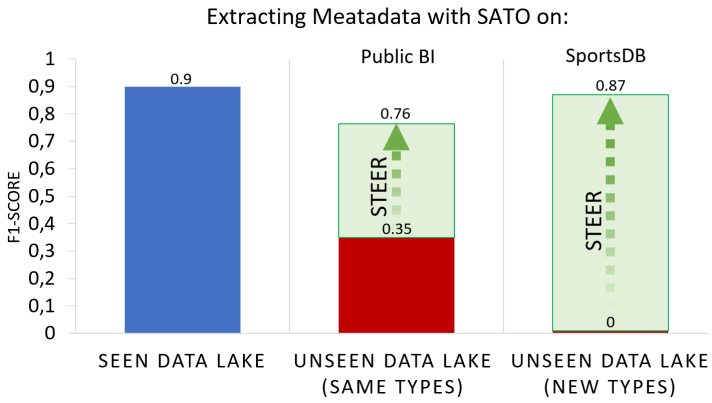


Figure 9.1: Performance of a learned metadata extraction model (SATO) on seen and unseen data lakes. The model provides high performance on the data lake (left) where it is trained for. However, using it on a new data lake with the same types but different data characteristics (middle/red) or on a data lake that includes different data types (right/red) is inferior. Hence, we introduce *STEER*, a framework to adapt learned semantic type extraction models to new, unseen data lakes with minimal effort and boost the performance (green).

9.1 Introduction

Data lakes are important. Data lakes are today widely being used in organizations to manage their data. Different from classical approaches such as data warehouses, data does not need to be organized and cleaned upfront when data is loaded into the warehouse [24]. Instead, data lakes follow a more "lazy" approach that allows enterprises to store any available data in its raw form without transforming and cleaning it in first place. This raw data is organized and cleaned once it is needed for a downstream task such as data mining or building machine learning models. However, for efficiently locating potential relevant data sources in a data lake, approaches for data discovery are needed.

Data discovery in data lakes is a problem. To address the data discovery problem and to improve the usability of data lakes, data catalogs are typically used. The need for such data catalogs is evident by the growing number of products available from different vendors such as Azure Purview [73], AWS Glue Catalog [4], Google Cloud Data Catalog [33], Alation [3], Collibra [17] and Dremio [25]. An important function of such data catalogs is to annotate semantic type information on columns of table-like data (e.g., CSV files) according to an ontology used in an enterprise. For example, for a data lake of a news magazine, semantic types such as *sports team* or *sport event* are important

information that allows a data journalist to identify which relevant sources she requires for preparing a news article. However, manually annotating data sources in a data lake with semantic types is a daunting task.

Learned semantic typing to the rescue. As such, in the last years various approaches for automated semantic type annotation have been proposed. Whereas existing commercial products mainly rely on simple search based solutions such as regular-expressions and dictionary look ups (e.g. [73, 74]), more recent approaches use machine learning [20, 50, 100, 113]. While initial results of these learned approaches are promising, unfortunately as we can see in Figure 9.1, a learned approach that was trained for data in one data lake cannot be used out-of-the-box for new unseen data sources in a different data lake [62] even if both data sources cover the same semantic types. This aspect is shown in detail in our short vision paper [62], by demonstrating the performance drop when using the learned model SATO [113] on the unseen data lake *Public BI* with the same semantic types.

The need to adopt the learned approaches. The reasons are that the data characteristics in the new (unseen) data lake might be completely different or even worse new semantic types occur that the model has not seen during training. Hence, the performance of a learned model might be completely different on the new unseen data lake. In this paper, we therefore propose *STEER*[1] which implements data programming for semantic labeling to adapt existing learned models for extracting semantics to unseen data lakes with minimal cost. As shown in Figure 9.1, *STEER* can thus not only significantly boost the performance of models on new unseen data lakes with the same types using SATO [113] on Public-BI as shown in Figure 9.1 (center) but also helps us to re-train a model to detect types on a data lake that comes with unseen semantic types as shown by the *TURL-Corpus* in Figure 9.1 (right).

Steered training data generation. In this paper we thus present the first labeling framework called *STEER* based on the idea of weak supervision to generate new labeled training data for a new unseen data lake with numerical and non-numerical data types. The generated training data of *STEER* can be used to re-train an existing learned semantic type detection model. As we show in Figure 9.1 (green bars) using the training data generated by *STEER* to re-train Sato can not only lead to a performance improvement in case we re-train Sato on a data lake with types the model has already seen (Figure 9.1, middle) but also to train the model on a data lake with unseen types

[1] https://github.com/DHBWMosbachWI/STEER.git

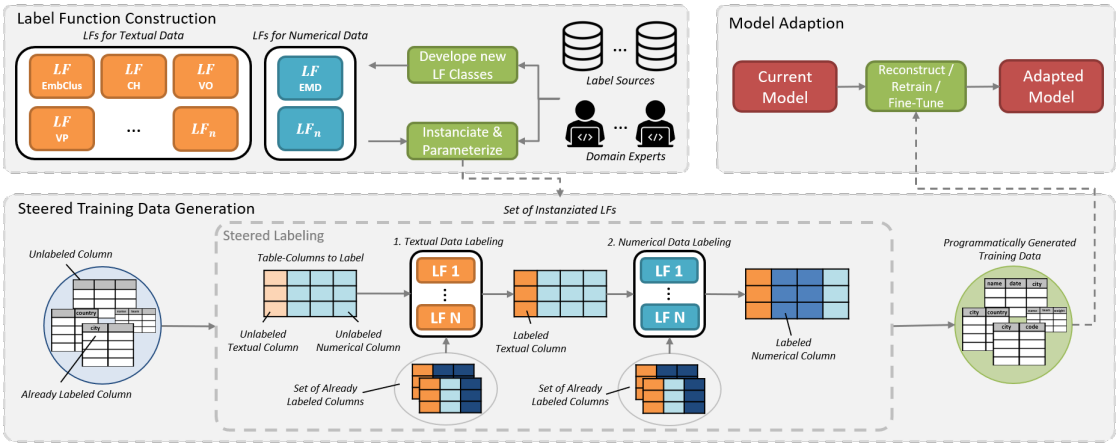


Figure 9.2: Overview of *STEER*. The main idea is that *STEER* provides a labeling framework that comes with different classes of labeling functions (LFs) for training data generation. These LFs can be extended and need to be instantiated by domain experts with minimal overhead, for example, by providing some examples of labeled columns. Afterwards, *STEER* creates labels (i.e., pairs of columns and semantic types) that can be used as training data. *STEER* divides the LFs into groups of functions for textual data and functions for numerical data and implements at the core a novel steered-labeling process that first labels textual columns and afterwards numerical columns so that the LFs for numerical data can benefit from the labels generated before. The steered-labeling process generates high-quality training data with minimal overhead on a new unseen data lake that can then finally be used to re-train or fine-tune an existing learned metadata extraction model.

from scratch (Figure 9.1, right).

Contributions of the paper. To be more precise, the main contributions of this paper are: (1) At the core, to generate labeled training data (i.e., pairs of columns with data and semantic types), we propose a new label generation process called *Steered-Labeling*. The intuition is that in *Steered-Labeling* we separate the process into two subsequent steps: *STEER* first labels the non-numerical columns that are easier to label. Afterwards, *STEER* then uses these labels to "steer" the labeling of the numerical columns. With this, *STEER* is able to not only generate high quality training data of textual columns but also to semantically label numerical data with a very high precision. (2) For labeling textual and non-textual columns, *STEER* comes with a set of pre-defined labeling functions (LFs). The base set of pre-defined labeling functions are the so-called generic labeling functions. These LFs are domain-independent and can be

used on any data set. Contrary to this, *STEER* can also be extended by some more domain-specific LFs. These LFs rely on the contribution of domain experts' knowledge to implement such functions. However, implementing such domain-specific LFs requires only minimal effort. We discuss some classes of such domain-specific LFs later in the paper. (3) Finally, as a last contribution we provide an extensive evaluation of *STEER* on four different data lakes with different characteristics that in total contain more than 643,500 columns. These data lakes vary in the semantic types and cover a wide spectrum of numerical and non-numerical data types. Moreover, in our evaluation we also show that our approach can be used across models that implement different learning approaches. In particular, we use SATO [113] which relies on a classical supervised training approach and TURL [20] that uses the pre-training/fine-tuning paradigm. The results of these experiments demonstrate that *STEER* works for both model architectures.

Overall, each scenario shows that *STEER* can generate training data that allows a learned model to provide high performance. Thereby we not only highlight the performance of the re-trained end model, but also the quality and quantity of the generated training data by *STEER*.

Outline. In Section 9.2 we first give an overview of *STEER*, its main components and present the details of our *Steered-Labeling*. Afterwards, Section 9.3 presents the details of our implemented LF for numerical data which is integrated into our steered-labeling approach. Section 9.4 then shows the LFs provided by *STEER* for labeling non-numeric columns. Afterwards, in Section 9.5 we then show the results of our extensive experimental evaluation of *STEER* in different scenarios. At the end, we discuss related work in Section 9.6 and Section 9.7 concludes the paper.

9.2 Overview of *STEER*

STEER implements a novel labeling framework that can generate high quality labeled data for training semantic type detection models on a new unseen data lake with minimal overhead.

9.2.1 The Labeling Framework

At the core, *STEER* provides a labeling framework based on the idea of weak supervision that comes with different classes of LFs for training data generation. The novel

aspect of *STEER* is that it comes with a new training data generation procedure called *Steered-Labeling* that can generate high quality training data not only for non-numeric but also numerical columns that are currently not supported by any of the existing learned approaches for semantic type detection. Figure 9.2 shows the overview of all phases in *STEER* to generate training data for a semantic type detection model.

Overall, the labeling framework of *STEER* can be divided into three different phases: *(1) Label Function Construction: STEER* already comes with a wide spectrum of labeling functions for numerical and non-numerical semantic types used by *STEER*. These labeling functions can be extended by a data engineer to support specific column values or labeling functions for highly specific data types. *(2) Steered Training Data Generation:* The steered training data generation phase is the core of *STEER* which uses the LFs and creates training data for non-numerical and numerical data. The idea of the steered-labeling procedure is that at first *STEER* labels the non-numerical columns that are easier to label. Afterwards, *STEER* uses these labels to "steer" the labeling of the numerical columns as shown in Figure 9.2 (bottom). The intuition of steering is that tables with similar semantic types for textual columns, also have similar semantic types for numerical columns. For example, a table about baseball teams has a column *sports team*. If a column *sports team* is present in a table, then the numerical values of the table's numerical columns are more likely to be about *height* and *weight* of players and not about *air pressure* or other numerical columns. We explain the steered-labeling procedure in more detail below. *(3) Model Adaption:* Finally, in the adaption phase an existing model such as SATO [113] or TURL [20] is adapted to the data lake by re-training or fine-tuning the model using the previously automatically generated training data.

9.2.2 Steered-Labeling Procedure

As core contribution of the training data generation in *STEER*, we introduce a new steered-labeling method to generate training data for labeling non-numerical (textual) and numerical columns. However, as we show in our evaluation, labeling numerical columns with LFs is generally more difficult than textual columns which contain semantic meaningful values such as names of cities or sports teams. In order to overcome this inherent problem and provide high precision also for LFs for numerical data, our idea is that with the *Steered-Labeling* approach numerical LFs can rely on context data from a

table; i.e. the already labeled textual columns.

To enable a steered labeling, *STEER* strictly separates the LFs into those for labeling non-numerical and numerical columns. In a first step, *STEER* uses the LFs for non-numerical data to label the subset of columns in the data lake that does not contain numerical data. Based on these labeled columns, *STEER* then aims to label the numerical columns. For this, *STEER* comes with a labeling function that is generally applicable for all numerical types.

The idea of this labeling function is that (1) a small fraction of numerical columns that represent the numerical types in the data lake need to be labeled upfront. Afterwards, these labeled examples are then used to (2) generate labels for other non-labeled numerical columns by using a clustering-based labeling function that clusters numerical columns with similar value distributions. Steering during clustering helps the clustering-based labeling to group numerical columns from similar tables and thus increase the labeling quality.

For example, if unlabeled numerical columns of a table that also has a column *sports team* should be labeled, steering would prefer tables for the cluster-based labeling that have a *sports team* column together with other labeled numerical columns. However, it is important to note, that steering in *STEER* is optional; i.e., cluster-based labeling can also be used without steering which is needed when no other table with the same semantic non-numerical type exists or tables contain only numerical columns.

In our experiments, we show that our novel steered-labeling process leads to huge benefits compared to a non steered-labeling process where all LFs are executed in parallel. To the best of our knowledge, the labeling framework of *STEER* is the first which uses such a steered-labeling procedure to semantically label non-numerical and numerical columns.

9.3 Labeling Numerical Columns

Existing approaches for annotating a type to a numeric column typically compare only the distributions of the data values from labeled to unlabeled columns using earth mover's distance like in [56, 114] or the p-value of statistical hypothesis test like in [80, 91].

Algorithm 2 LF EMD: The labeling function is based on clustering columns in tables which share the same context using the the earth mover's distance as similarity metric between numerical columns.

```
1: emd_threshold ← precalculated threshold
2: C_ln, T_ln ← set of labeled numerical cols and their table
3: C_un, T_un ← set of unlabeled numerical cols and their table
4: for All c_un in C_un do
5:     emd_results = []
6:     C_lt1 ← set of labeled textual cols of table T_un of c_un
7:     for All c_ln in C_ln do
8:         C_lt2 ← set of labeled textual cols of table T_ln of c_ln
9:         if length(C_lt1 ∩ C_lt2) > 1 then
10:            emd_results.append(earth_mover_dist(c_un, c_ln))
11:        end if
12:    end for
13:    sort(emd_results)
14:    if size(emd_results) > 0 then
15:        if emd_results[0] < emd_threshold_{c_ul} then
16:            assign semantic type of emd_results[0] to c_un
17:        end if
18:    end if
19: end for
```

However, these naïve approaches are typically more inaccurate for data lakes since usually numeric columns have a lower entropy than textual columns and thus have a lower information content which makes it harder to differentiate numerical columns from one another[2]. Therefore the information provided by the value distribution of numerical columns is too limited and leads to many false annotations if it is the only semantic typing criterion. In order to overcome this inherent problem and boost the precision of the semantic type detection our idea is instead to rely on context data from the table (e.g. information about neighboring columns) which we use to steer the labeling of numerical columns. In the following, we explain a labeling function that is based on this idea.

9.3.1 Labeling by Context-aware Clustering

The idea of *Steered-Labeling* is integrated into a labeling function (LF) of *STEER* that is based on the idea of context-aware clustering. In order to use this labeling function, a data engineer has to provide at least one table with a labeled column per semantic type

[2]Generally numeric values can be encoded with much less bits than string values resulting in lower overall entropy values [98]

t the LF should create labels for. Afterwards, the labeling function uses the annotated column that has type t and the table T the column is part of to create other labeled numerical columns of the same type. Moreover, the labeling function assumes that columns with textual semantic data types have already been labeled as discussed in Section 9.2. Labeling functions for textual columns will be described in detail in the next section.

The pseudocode of the labeling function which labels numerical columns is shown in Algorithm 2 and works as follows: Given a set of manually labeled numerical columns C_{ln} (and the tables T_{ln} they are part of), and a set of unlabeled numerical columns C_{un} (and the tables T_{un} they are part of), the labeling function iterates over the unlabeled columns c_{un} to label them in the respective iteration step (line 4-19). In this iteration step, the labeling function first retrieves context information about c_{un}; i.e., we retrieve the semantic types of all non-numerical columns of table T_{un} (line 6). In the next step, we iterate over each labeled column c_{ln} in C_{ln}. Afterwards, we then retrieve context information also for c_{ln} which is part of table T_{ln}.

In case the two tables — T_{un} and T_{ln} — share at least one column with the same semantic type, we compute the earth mover's distance between c_{un} and c_{ln} as a metric of the similarity of both columns (line 10). As such, we compute the earth mover's distance only against labeled numerical columns in tables that share the same context which improves the accuracy of the labeled training data significantly as we show in our evaluation in Section 9.5. Each earth mover's distance measurement to a labeled numerical column is then stored in a list, which we finally sort in ascending order (line 13); i.e., the most similar labeled column is first.

Once we iterated over all already labeled columns C_{ln}, we assign the one which is most similar to the unlabeled column c_{un} (line 16). However, we only assign a numeric data type if the earth mover's distance value is below a precalculated threshold. The threshold specifies the minimal measure of similarity that must be given to assign the same semantic type to the unlabeled column. In the following, we now describe how to set this threshold automatically for each unlabeled column c_{ul}.

9.3.2 Determining the EMD Threshold

In the LF *EMD* as shown in Algorithm 2, the values of the two numerical columns are not normalized before the earth mover's distance is computed as a similarity measure. A normalization of the distribution would lead to a loss of information and to many false matches between labeled and unlabeled columns. Therefore the earth mover's distance between the values of the two numerical columns is not normalized.

As a result, the earth mover's distance values of a comparison between two columns varies in a value range between $[0, MAX\_EMD]$ where MAX_EMD can be arbitrarily large. As such, setting a fixed threshold value is not possible. Instead, we set a threshold individually per unlabeled column c_{ul}. The intuition is that we find a threshold that considers the value distributions of the numerical values in that column.

Moreover, determining the threshold correctly is very important. When setting the threshold too low, we might not assign any numerical type while when setting the value too high, we might see a lot of false labels. Hence, to set the threshold we first compute the distribution of earth mover's distance values across a representative set of pairs of unlabeled numerical columns. That way, we can decide what a significant difference between two earth mover's distance values is and thus also when two columns are similar; i.e., the difference of the earth mover's distance value is not significant.

To calculate the threshold for the earth mover's distance we use the following equation:

$$emd\_threshold_{c_{ul}} = 0.4\, Quantile\_EMD * std\_c_{ul}$$

where $0.4\, Quantile$ comes from the the distribution of all earth mover's distance measurements between labeled columns and unlabeled columns and *std* is the standard deviation of the unlabeled column c_{ul} that is supposed to be labeled at the moment.

The standard deviation of the unlabeled column gives an indication about the normal range of values within this column and the underlying semantic type. Therefore, the earth mover's distance may be in similar dimensions when compared against a column with the same semantic type. The quantile of all earth mover's distance of labeled columns to unlabeled columns represents which distance is significant. The defined quantile of 0.4 was determined by a hyper-parameter search in our evaluation and is robust not only across all numerical data types but also across all data lakes we used in our experiments.

9.3.3 Numerical-only Tables

As described above the context-ware LF *EMD* for labeling numerical data relies on existing textual semantic types of the neighboring table columns. During computing the semantic similarity between unlabeled and labeled columns as mentioned before, two different situations can thus occur, where no context information is available. (1) Table T_{un} of the unlabeled numerical column c_{un} contains only numerical columns or no textual columns are annotated. (2) Table T_{ln} of the labeled column c_{ln} contains only numerical columns or no annotated textual columns and therefore no context information is available. We address these situations as follows. On (1) we measure the earth mover's distance against all labeled columns available without considering any context information. In case of (2) the earth mover's distance measure is made against the labeled column due to the missing context information.

9.4 Labeling Non-Numeric Columns

In our *Steered-Labeling* framework as discussed in Section 9.2, we show that we first label non-numerical columns. For this, our labeling framework provides a set of four different types of LFs for annotating semantic types to non-numerical table-columns. As discussed before, we separate the LFs in *STEER* between LFs which are (1) Generic Labeling Functions that work without any manual adaption of the LF and (2) Domain-Specific LFs which require adaption of the LFs to the data types of the data lake. These LFs rely on the contribution of domain experts knowledge by providing some limited number of example values for a domain-specific data type. Moreover, our framework is in principle extensible and a data engineer can add own LFs with algorithms that can detect semantic types of table-columns. In the following, we explain in detail the different LFs for labeling non-numerical columns.

9.4.1 Generic Labeling Functions

STEER provides two types of generic labeling functions that can be used in a domain independent manner.

Labeling by Embedding Clustering. The first generic labeling function is one that relies on embedding clustering *EmbClus*. Similar to labeling numerical types by clustering, this labeling function requires that a small set of columns in a data lake is already

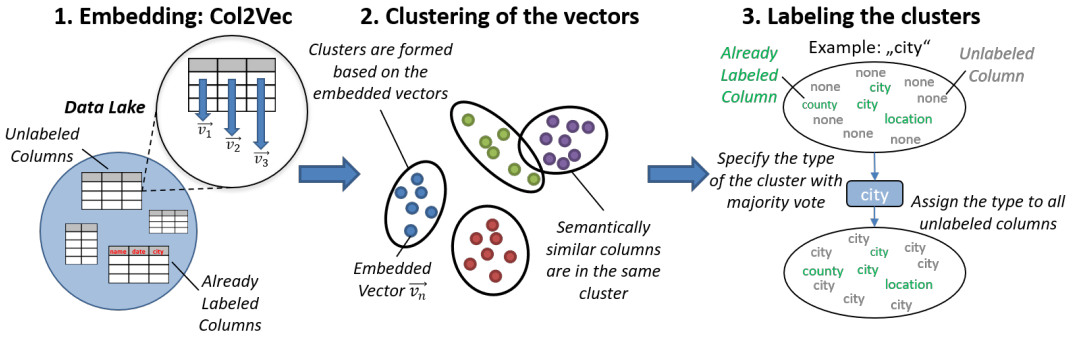


Figure 9.3: Labeling by Embedding Clustering *EmbClus*. In a first step, embeddings are computed for all columns. Afterwards, by clustering new labels are generated using existing labels from already annotated table columns.

annotated with semantic types. This is often the case since data lakes are constantly growing in size and thus some columns might have a semantic type. However, one could also use this LF if a domain expert is willing to first label a small (representative) set of columns in the data lake manually.

The main idea of *EmbClus* is to use column embeddings to cluster columns with similar values and thus generate labels for previously unlabeled columns. Figure 9.3 shows the implemented algorithm and the individual detailed processes. In the first phase, we compute column embeddings for both labeled and unlabeled table columns based on word embeddings of individual column values. As word embeddings, we currently use *Google USE*[3] [111] that was trained on 16 different languages.

In principle we could also use other word embeddings, but multilingual models can better cover the spectrum of different "custom" semantic types in different enterprise data lakes. Furthermore, the model is also designed to embed sequences of words (e.g. sentences) and thus gives us the possibility to embed column values that contain more than just one word. Based on the embeddings of individual values, we compute an embedding for all values of a column by calculating the average across the embeddings of all values which is the dominant approach for building representations of multi-words also mentioned in other papers [99].

[3]https://tfhub.dev/google/universal-sentence-encoder-multilingual/3

Once we computed an embedding for all columns, we cluster labeled and unlabeled columns together based on these embeddings. The intention here is that clusters are formed with columns that have the same semantic type. For this step, we use the hierarchical agglomerative clustering algorithm[4]. In *STEER*, we use this class of clustering method to not generate a fixed number of clusters, but to form groups based on the cosine similarity of vectors (i.e. our embeddings) and a distance threshold that we discuss below.

Once clustered, we then compute a semantic type per cluster based on the majority vote of the labeled columns in that cluster. In the absence of any labeled column in a cluster, we assign no semantic type to that cluster. A key parameter to be set in our clustering algorithm is again the distance threshold, where lower values mean that we produce more clusters. In our experiments, we used a threshold of 0.01 based on a hyper-parameter search on the already labeled columns. This distance threshold provided good results on the broad spectrum of datasets in all four observed fictive data lakes.

Labeling by Column Headers (CH). Another generic labeling function is the *labeling by column headers* (CH). The main idea of labeling by column headers (CH) is to use the original column headers as information to derive the semantic type of the column. The original column headers often represent type information but do not directly represent semantic types (e.g., a unified ontology of the enterprise) of the data lake catalog. For generating labels with existing column headers, we again use pre-trained language embeddings to embed column headers as well as the semantic types of the given ontology that should be used for the catalog.

Based on these embeddings, we match them similar to the work in [47]. As similarity measure between the column header and each semantic type, we use the cosine similarity of the vectors. Based on the similarity measure, the LF then assigns the semantic type as label to the column with the highest cosine similarity. Moreover, the cosine similarity must have a minimum similarity threshold value. In our experiments we use a similarity threshold of 0.9 based on a hyper-parameter search on the already labeled columns. Since this LF is based on column headers, it can in general be applied to columns with textual data as well as to columns containing numerical data.

[4] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html#sklearn.cluster.AgglomerativeClustering

9.4.2 Domain-Specific Labeling Functions

In this section we now present our implemented LFs of the second category. *STEER* provides two types of that LF category which can be specialized by the domain expert to their own data lake.

Labeling by Value-Overlap (VO). The central concept of *Labeling by value-overlap* (VO) is to enable a domain expert to provide a list of common values for a semantic type. To give an example, a domain expert in the field of American football can provide a list of typical team names (e.g. [atlanta falcons, new england patriots,...]) for the semantic type *american_football.football_team* that occur frequently in table columns. The LF uses this list to check how many values in an unlabeled column exactly match one of the values in the provided list. Notice that this is an exact string matching, where we first convert all strings to lower case and than apply the comparison. If the number of matching values is over a threshold, the LF assigns the corresponding label to that column. In our experiments we use a threshold of 20%, which is however a hyperparameter that can be tuned per data lake.

Labeling by Value-Patterns (VP). The LF *Labeling by value-patterns* (VP) allows domain experts to specify a list of general patterns via regular expressions (regex) for a semantic type. As applied in the previous LF, we use the list of regular expressions to check how large the fraction of column values is for which a pattern matching was successful. If this fraction is over a predefined threshold, the column gets the according type. In our prototype we use a threshold of 20%, which generated high quality training data for all considered datasets.

9.4.3 Discussion

In *STEER*, each non-numeric column is labeled by each existing non-numeric LF. Therefore, after processing all LFs, discrepancies can exist, since one column can get several different semantic types from different LFs. In this case, *STEER* combines the labels from multiple LFs by using a majority vote. We also tried out other strategies such as using a generative model that is trained on the output of the labeling functions (which is a strategy suggested in [92]). However, in all our experiments the majority vote provided superior performance.

9.5 Experimental Evaluation

In the following, we introduce the four datasets (*Public BI* & *TURL-Corpus* & *Public BI Num* & *SportsDB*) and describe the evaluation methodology. Moreover, we use *STEER* to re-train two different models for semantic type detection (SATO [113] and TURL [20]) on these data lakes.

9.5.1 Datasets

For evaluating *STEER*, we use a total of four different real-world data sets with a large number of different tables as data lakes (see Table 9.1). Overall, we have two data lakes with only non-numeric semantic types (*Public BI* and the *TURL-corpus*) while we have two data lakes with numeric and non-numeric semantic types (*Public BI Num* and *SportsDB*). We use the data lakes with non-numeric semantic types to compare against baselines (e.g., a pre-trained SATO or TURL model) that initially do not support numeric data types. We later on provide more details on the distribution of numeric and non-numeric data types for the two data lakes for which we annotated numeric types (see Table 9.2).

Public BI [31]. As first data lake we use the *Public BI Benchmark*[5] data corpus. The data corpus contains real-world data, extracted from the 47 biggest public workbooks in Tableau Public[6]. As no semantic types are available for the *Public BI*, we annotated the columns manually with the correct semantic types. For the annotation with semantic types, we adopted the same semantic types used in SATO [113] which is originally trained on the VizNet data set. This allows us to evaluate a setup with a data lake that uses the same semantic types as a already trained model but might use different data distributions in the respective columns. Overall, out of SATO´s 78 semantic types, only 33 were present in the *Public BI* data corpus. All of these data types are non-numeric. For this dataset, we thus restricted the experiments to these 33 types that are present in the *Public BI* data set and also supported by SATO to have a data corpus which represents a scenario where the semantic types are in principle supported by the pre-trained model.

TURL-Corpus [20]. As a second data lake we use the dataset from TURL [20]. *TURL-Corpus* uses the WikiTable corpus [9] as basis. To label each column they refer to the semantic types defined in the freebase ontology [34] with a total number of 255 different

[5] https://github.com/bogdanghita/public_bi_benchmark-master_project
[6] https://public.tableau.com

semantic types. What distinguishes this dataset from the *Public BI* corpus is that there are no matches with learned semantic types supported by the original SATO model. This means that we have a scenario with *TURL-Corpus* where the model has to be adapted not only to the new data characteristics but also to the new semantic types. Moreover, in the original *TURL-Corpus* columns can have multiple semantic types. To have a dataset that fits to our evaluation methodology (i.e., predict only one type per column), we manually selected the most specific semantic type out of the given semantic type set. To give an example, if one column has the labels *sports.sports_team* and *soccer.football_team*, we select the second semantic type as valid label because it is more specific.

Public BI Num [31]. To construct a data lake that comes with annotated numeric semantic types, we again used the *Public BI Benchmark* as a basis. Contrary to *Public BI*, we extended the semantic types by 19 additional numerically based types where the associated columns contain numeric values. Overall, this leads to a data lake with a significantly higher number of numerical columns and is therefore more comparable with real-world data lakes. With this data lake we are not only able to evaluate the performance of existing semantic typing models on a wide set of numerical semantic types and data, but also to evaluate our *Steered-Labeling* approach in depth and show its benefits.

SportsDB. As a fourth data lake we introduce a new corpus named *SportsDB*, which also contains a large fraction of numerical columns similar to real-world data lakes. We constructed this corpus by extracting tables from different websites that publish statistics about football in recent years. For example, the corpus contains tables about statistics of football players in which *player name, goals, assists, games, etc.* are listed. The extraction resulted in a corpus of 78 tables each containing an average of 3 textual columns and about 15 numerical columns. For annotating the columns with semantic types, we formed an ontology containing 18 different semantic terms (3 textual based types and 15 numerical based types) from the football domain. The assignment of the defined semantic types to the respective column was done semi-automatically using column headers and checked manually afterwards. When using this data set, we did not use the LF of *STEER*, which labels columns by headers.

9.5.2 Experimental Design

Setup. For the evaluation, the four datasets were split into three parts: labeled, unlabeled and test. The labeled split represents the set of table columns that we consider

Table 9.1: Characteristics of the four data sets used as data lakes.

Dataset	#Tables	Avg #Cols per Table	#Types	Ontology
Public BI	160	8.96	33	DBPedia
TURL-Corpus	401,538	1.59	105	Freebase
Public BI Num	170	13.64	52	Custom
SportsDB	78	17.83	18	Custom

to be already labeled in the data lake. In the individual experimental setups, we apply different sizes of labeled data to make the results more comparable and show the impact of the quantity of already manually labeled data. For this, we decided to have 1 to 5 columns per semantic type already labeled as a starting point in our experiments.

To measure the performance of the different semantic typing models, we used a 20% split as test data. While creating the split, we first extracted the 20% test data and then used the remaining 80% to create the labeled and unlabeled set as described above. The unlabeled data was used as input to our *Steered-Labeling* framework to generate additional training data. To obtain statistically reliable results, we ran each experiment with five different random seeds and report the mean and standard deviation over multiple runs.

Experimental Structure. To demonstrate how well *STEER* can adapt and improve existing models to new data lakes, we have divided our evaluation in different use cases.

(1) STEER on Non-Numeric Data. In the first set of experiments, we evaluate *STEER* in combination with the existing models SATO and TURL that originally only support non-numerical semantic types on the *Public BI* and *TURL-Corpus* data set. With this experiment, we also show two scenarios: in the first scenario, we want to show the model adaption to data lakes which on the one hand contains types already seen by the model but with different data characteristics (i.e., SATO on *Public BI* and TURL on the *TURL-Corpus*). In the second scenario, we show how well *STEER* can be used to train a model on a data lake with new types the model has not seen before (SATO on *TURL-Corpus* and TURL on *Public BI*). Finally, in this set of experiments we demonstrate with this use case the model independence of *STEER* by applying it to two different model architectures and model paradigms (SATO vs. TURL).

(2) STEER on Numeric Data. We evaluate *STEER* against data lakes which have a large proportion of numerical columns and numerically based semantic types (*Public BI Num* & *SportsDB*). Here we show the efficiency of our new *Steered-Labeling* approach by comparing a re-trained model with and without training data generated by *Steered-Labeling*.

(3) Ablation Study. In an ablation study we discuss and show the efficiency of individual LFs of our labeling framework and analyze the generated training data.

Baselines. In our experiments, we use several baselines to compare the efficiency of *STEER*.

(1) Sato baseline. We use the available learned SATO neural network called *Sato baseline* from [113] and applied it to the test data without any fine-tuning by re-training. With this, we want to see how well the existing model performs in new unseen data lakes with the same semantic types, similar to what we showed in Figure 9.1. Since only the *Public BI* dataset contains semantic types from SATO's learned model, this baseline is only used for this dataset.

(2) Sato retrain. As a second baseline we use *Sato retrain*. The idea is to use the set of existing labeled data (before applying our LFs) to re-train the SATO model. This experiment illustrates, the effect of re-training an existing model with a small amount of manually labeled data. This baseline shows that the small set of existing labeled data is not sufficient to re-train a learned model and that our labeling framework *STEER*, which generates much larger training datasets, can significantly boost the performance.

(3) Turl retrain. As third baseline we use *Turl retrain*. Same as for *Sato retrain*, the manually labeled columns are used to fine-tune the pre-trained TURL model. It shows that even for the recent trend of pre-train/fine-tune models like TURL, a small existing set of labeled training data is not enough and the larger training data generated by *STEER* can significantly boost the performance.

Our Approach. To show the efficiency of *STEER* on SATO, we consider the same amount of manually labeled data to re-train *Sato retrain* and to fine-tune *Turl retrain* as a basis for generating more training data using our approach. Afterwards, we use the generated training data by our labeling framework to re-train the existing models (SATO and TURL) with this larger amount of data. The goal of this is to prove how our new approach and the additional generated training data can boost performance.

In order to report the benefits of *STEER* when using the model SATO or TURL, we fine-tune the pre-trained models with the larger amount of data generated by our labeling framework and name these model *STEER on Sato* and *STEER on Turl*. Finally, the models are used on the test data split in order to demonstrate the benefits of using our approach in comparison to *Sato retrain* and *Turl retrain*.

9.5.3 STEER on Non-Numerical Data

In the following, we evaluate *STEER* on the two non-numerical data sets *Public BI* and the *TURL-corpus*.

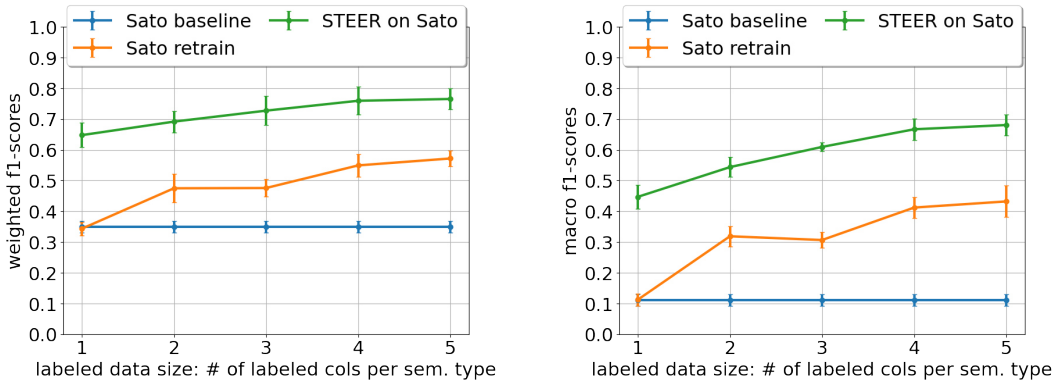
9.5.3.1 STEER for Unseen Data Lakes

In the first experiment, we compare *STEER* against the baselines in two scenarios (known and unknown data types).

Scenario 1: Same Semantic Types. This section reports the overall results of using *STEER* in a scenario where the existing model already knows the semantic types of the new data lake from a previous training. To realize this scenario, SATO is used as model and the *Public BI* dataset as data lake.

Figure 9.4a and Figure 9.4b show the results reporting macro and support-weighted F1-Scores using the defined set-up as described before. First, we see that *Sato baseline* achieves only moderate F1-Scores, although the model supports all semantic types in the data lake. Secondly, as expected, the model *Sato retrain*, re-trained with the manually labeled data (but not with the generated training data by *STEER*), achieves better scores as *Sato baseline*. This shows the positive effect of adjusting the model to new data characteristics by re-training. This effect intensifies with the increasing amount of already labeled training data per semantic type. With the maximum size of 5 labeled columns per type, *Sato retrain* can achieve 0.43/0.57 (macro/support weighted) as average F1-Score.

Compared to *Sato retrain*, our re-trained model *STEER on Sato* outperforms the results by an average of +0.27/+0.24 (macro/weighted) F1-Score for each given size of labeled data. In total, *STEER on Sato* achieves an F1-Score of 0.681/0.765 (macro/weighted) and consequently achieves an improvement of 57.6%/34.2% over *Sato retrain* and 508%/119% to *Sato baseline*. Overall, this evaluation shows that our model *STEER on Sato* success-



(a) Public BI (weighted F1)

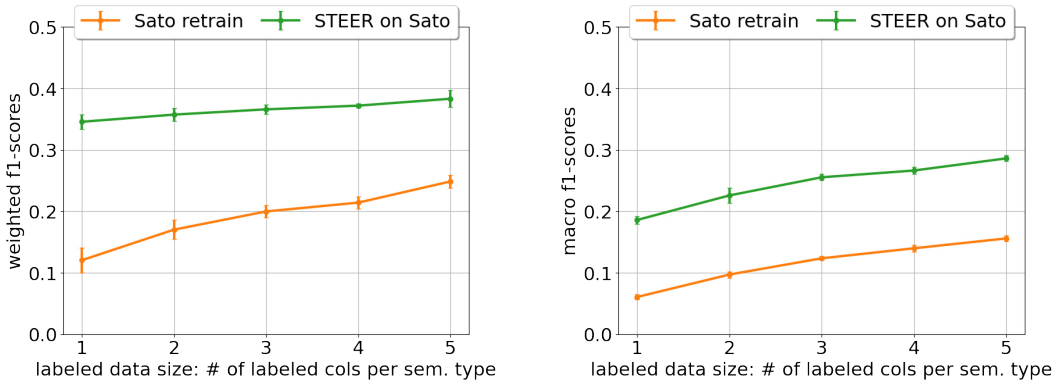
(b) Public BI (macro F1)

Figure 9.4: Results on adapting SATO to the *Public BI* data lake to evaluate the scenario where the model is applied to a data lake containing already seen semantic types by the model. Our model *STEER on Sato* re-trained with the additional generated training data using *STEER* outperforms the baselines in every set-up.

fully optimized the adaption of the SATO model to the data lake *Public BI*.

Scenario 2: New Semantic Types. The next experiment is designed to show the results of *STEER* in a scenario where the learned model does not support the semantic types existing in the data lake. Thereby the model has to learn completely new types, which makes it more difficult to adapt the model because generally it will require a larger amount of training data. To perform the defined scenario, SATO is used as model and *TURL-Corpus* is used as data lake which comes with 105 new semantic data types. For this, we replaced the last layer of SATO to support 105 instead of the originally 78 types and initialized it with random weights.

Figure 9.5a and Figure 9.5b shows the results and compare our *STEER on Sato* model with the defined baselines. For *Sato retrain* we can see again that the F1-Scores of the model continuously improve. However, the gains are more moderate compared to *Public BI*. Looking at the results of *STEER on Sato*, we see a significant performance improvement in both macro and weighted F1-Scores compared to *Sato retrain*. We also see that *STEER on Sato* is able to constantly increase the model performance when having more labeled data. In other words, *STEER on Sato* is capable to efficiently use the generated additional training data for re-training. Since the dataset is overall more challenging (e.g., in diversity of values in columns with the same semantic types) and the



(a) TURL (weighted F1)

(b) TURL (macro F1)

Figure 9.5: Results on adapting SATO to the *TURL-Corpus* data lake to evaluate the scenario where the model is applied to a data lake containing completely new semantic types the model not seen before. *STEER on Sato*, which was re-trained with the generated training data of *STEER*, outperforms the baseline over all labeled data sizes.

model has to learn new semantic types, the F1 values are slightly lower overall than on *Public BI*. At labeled data size 5, *STEER on Turl* achieves best F1-Scores of 0.29/0.38. Note that the F1-scores in our paper are lower than in the original paper [20] since they use multiple correct data types per column (which makes it easier for the model to at least predict one of them) and they use a large amount of manually labeled training data (i.e., 628,254 columns with manually annotated semantic labels) to fine-tune TURL, which we think is not realistic.

9.5.3.2 Model Independence of STEER

The main goal of the following experiment is to show that *STEER* also can adapt and improve models with different architectures and paradigms. For this study, we additionally use TURL as model that is based on the idea of representation learning and is thus already pre-trained across a large corpus of tables. For the comparison of our approach *STEER on Turl* with the baseline *Turl retrain*, both non-numerical datasets (*Public BI* & *TURL-Corpus*) are used.

Figure 9.6a and Figure 9.6b show the results of the experiments by plotting the weighted F1-Scores separately for the two datasets. We also add the results from the previous experiment with SATO as model to see the differences of the two models on

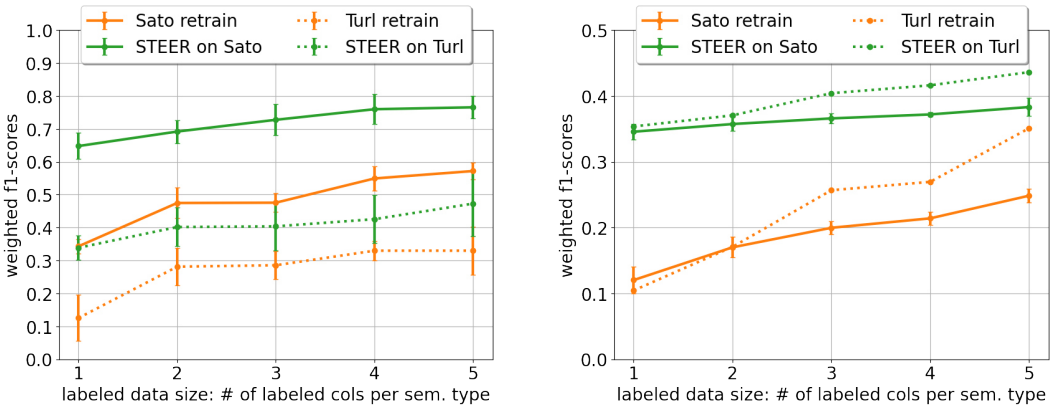
both data lakes.

As a first aspect of the results, we can see that our model *STEER on Turl* is significantly better than the baseline *Turl retrain* in both datasets and across all labeled data sizes. Considering Figure 9.6a, *STEER* can achieve an improvement of +0.13 as average F1-Score over the labeled data sizes. In the experiments using *TURL-Corpus*, we achieve an average improvement of +0.25 F1-Score. This demonstrates that even for pre-trained/fine-tuned models, *STEER* can improve the performance with the additional generated training data during the fine-tuning. That is especially remarkable, as it is assumed that pre-trained models need fewer training data samples during fine-tuning.

Furthermore, despite the fact that the model was pre-trained on the *TURL-Corpus*, fine-tuning the model with the additional training data provided by *STEER* leads to performance gains. In total, *STEER on Turl* achieves the best F1-Scores at labeled data size 5 of 0.47 on *Public BI* and 0.44[7] on *TURL-Corpus*. It is important to note, that we performed another test on the *Public BI* corpus with the maximum training data size of 80%, which results in a model that achieves a performance of 0.56 weighted F1-Score. Compared to the score values of *STEER on Turl*, the gap to this theoretical maximum reachable score is remarkable.

An additional interesting detail of the experiment results is the comparison between SATO and TURL model on both data lakes. Note that when using *Public BI*, SATO already knows the semantic types (Scenario 1), while TURL has not seen the data at all. This fact is exactly vice versa when using *TURL-Corpus*. The TURL model has already seen the data during pre-training whereas SATO has not seen the data and has to learn new types (Scenario 2). When comparing *STEER on Sato* to *STEER on Turl* with *Public BI* as data lake, we can see in Figure 9.6a that *STEER on Sato* achieves a much higher F1-Score than *STEER on Turl*. Even *Sato retrain* retrained with a much smaller amount of training data is better than *STEER on Turl*, which mainly shows the advantage of having a model pre-trained on the data. Consider the results on the *TURL-Corpus* demonstrated in Figure 9.6b, we can also see this advantage but this time for TURL; i.e., *STEER on Turl* achieves better results than *STEER on Sato*.

[7] [20] applied TURL model on *TURL-Corpus* and reported 0.9475 as F1-Score. This was achieved in the paper by fine-tune the model with 80% training data. Notice that in our set-up we fine-tune TURL only with about 14.5% training data, leading to the large discrepancies between the two reported F1-Scores.



(a) Public BI (weighted F1)

(b) TURL (weighted F1)

Figure 9.6: Results on adapting TURL model to the *Public BI* and the *TURL-Corpus* to show that even for pre-trained/fine-tuned models, *STEER* and its generated training data lead to performance gains. We added the results of SATO from the previous experiment for comparison.

9.5.4 *STEER* on Numerical Data

Real-world data lakes often contain a significant amount of numerical columns. This fact is also shown in Table 9.2 which shows the distribution of numeric and no-numeric columns in our two corpora where we annotated numeric types. As mentioned above, extracting semantic types from numeric values is more challenging because of the very low entropy. In order to generate training data with *STEER* for numeric columns with a high quality, we have implemented our *Steered-Labeling* approach as described in Section 9.2. The experiments in this section evaluate *STEER* the two data lakes that contain numerical columns.

9.5.4.1 Efficiency of Steered-Labeling

In the following experiments we use *Public BI Num* and *SportsDB* as data lake and SATO as existing model. Like before we compare the results with the baseline *Sato retrain*, which is the model re-trained with a small amount of training data coming from the data split and the described different labeled data sizes. To show the benefits of our *Steered-Labeling* approach we introduce another baseline *STEER on Sato no steer*, which is the model re-trained with training data generated without the usage of our *Steered-Labeling* approach; i.e., we do not label textual and numeric labeling sequentially but we instead use the textual LFs also for numeric data. By contrast, the model

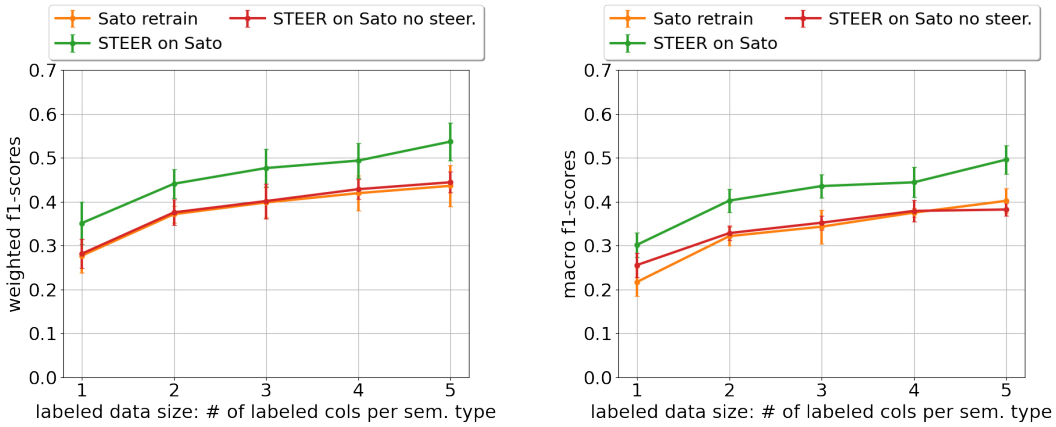
Table 9.2: Average textual and numerical columns in real world tables, showing the aspect that such tables have a high proportion of columns containing numerical values.

Corpus	Domain	#Avg. Textual Cols	#Avg. Numerical Cols
Public BI Num	Sport	6.4	48.5
Public BI Num	Medicare	14.7	13.0
Public BI Num	Real Estate	14.2	22.4
Public BI Num	Government	34.0	22.0
Public BI Num	Geography	24.2	9.0
SportsDB	Football	3.0	14.83

STEER on Sato is retrained with training data generated by *Steered-Labeling* and thus demonstrates the gains of our new approach.

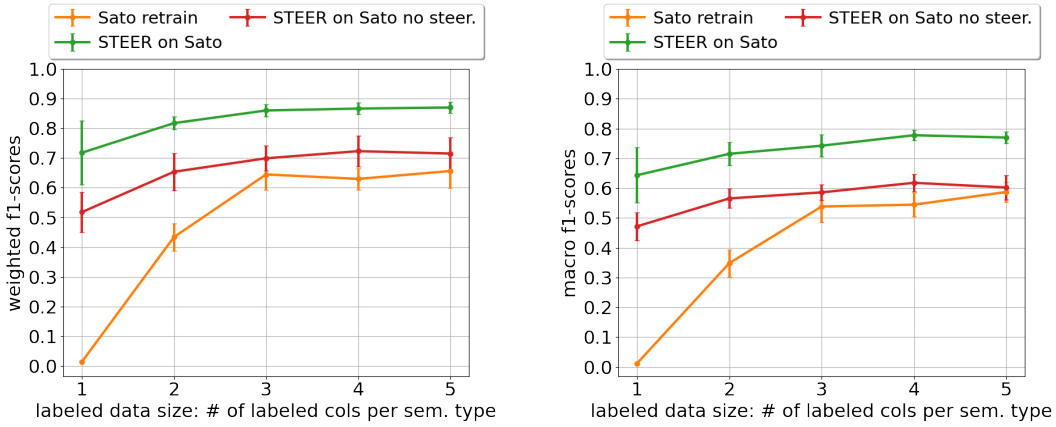
Figure 9.7 shows the results of the different models *Sato retrain*, *STEER on Sato* and *STEER on Sato no steer* on the *Public BI Num* dataset. Due to the use of *Steered-Labeling* *STEER on Sato* outperforms both baselines in macro and weighted F1-Score on all manually labeled data sizes. Compared with *STEER on Sato no steer*, the score values confirm that *Steered-Labeling* increases the quality of the generated training data and finally leads to a better end model. At labeled data size 5, *STEER on Sato* reaches best scores with 0.496/0.537 F1-Score. Considering the score values of *STEER on Sato* on *Public BI*, which contains the same textual data but not the number of numerical data, we see a drop of -0.184/-0.228 F1-Score. In addition, the results show an almost identical performance of the models *Sato retrain* and *STEER on Sato no steer*.

On Figure 9.8 we demonstrate the macro and weighted F1-Scores on the *SportsDB* corpus. The results show that our model *STEER on Sato* with *Steered-Labeling* outscores the baselines on every labeled data size. At best, *STEER on Sato* reaches 0.77/0.87 F1-Score resulting in an increase of +0.167/+0.155 in comparison to *STEER on Sato no steer*. Overall, these results also show that the *Steered-Labeling* method has significant advantages in generating the training data than without *Steered-Labeling*. One additional detail to be mentioned in the results is the comparison in the set-up with a labeled data size 1 (amount of existing labeled data). Here *STEER on Sato* outperforms *Sato retrain* by +0.632/+0.705 F1-Score, demonstrating that *STEER* can generate a large amount of good training data with a small amount of already labeled data as a basis.



(a) Public BI Num (weighted F1) (b) Public BI Num (macro F1)

Figure 9.7: Results on adapting SATO to the *Public BI Num* data lake using our *Steered-Labeling* generated training data. In addition to the comparison of *Sato retrain* (baseline trained with initial already labeled data available) we compare also to a model which was built with generated training data without the usage of *Steered-Labeling*.



(a) SportsDB (weighted F1) (b) SportsDB (macro F1)

Figure 9.8: Results on adapting SATO to the *SportsDB* data lake using training data generated by *Steered-Labeling*. In addition to the comparison of *Sato retrain* (baseline trained with initial already labeled data available) we compare also to a model which was built with generated training data without the usage of *Steered-Labeling*.

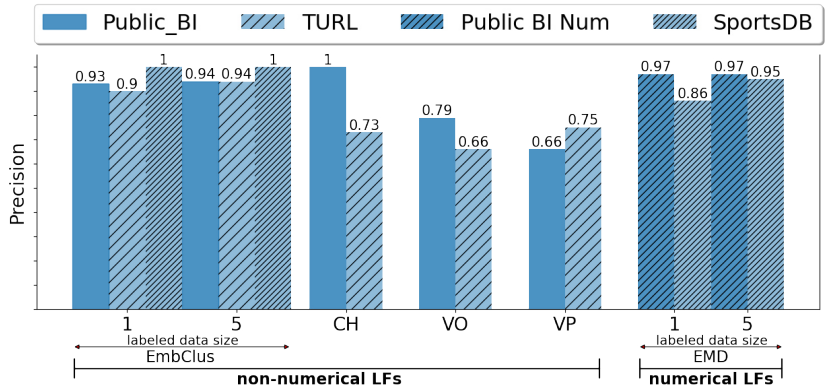
Table 9.3: Runtime gains on Public BI Num using context informations from the *Steered-Labeling* process for preselecting relevant columns for the LF *EMD*

Labeled Columns per Data Type	Reduction of Labeling Runtime
1	25.5%
2	34.5%
3	39.2%
4	44.9%
5	49.6%

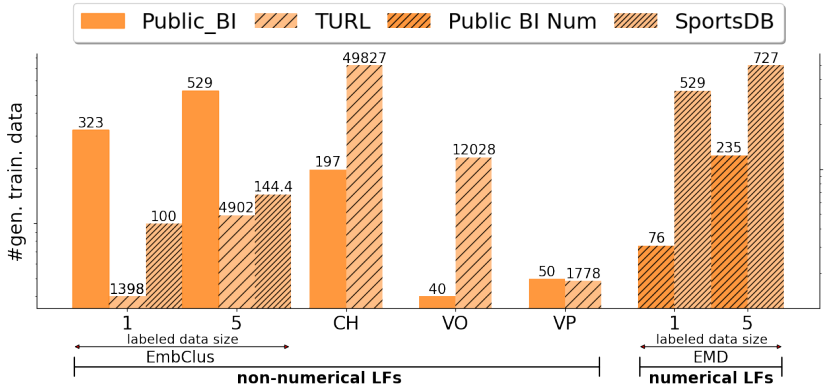
Discussion. In a more detailed analysis (not included in this paper), we observed that the numerically semantic types are poorly predicted by *STEER on Sato*, even though the generated training data quality is quite adequate (see Figure 9.10c). In the future, we thus plan to work on model architectures that are more tailored towards detecting numerical data types.

9.5.4.2 Optimization for Steering

As described in Section 9.2.2, *STEER* strictly separates the LFs into those for labeling non-numerical and numerical columns to execute first the non-numerical and then the numerical LFs. The main idea behind *Steered-Labeling* is that based on the generated labels in step one, the numerical LFs can benefit from this additional information during the labeling. In the previous experiment, we demonstrated the accuracy gains (reported by F1-Scores) in the adapted end model by re-trained one model with (*STEER on Sato*) and another model (*STEER on Sato no steer*) without steered generated training data. However, *Steered-Labeling* does not only lead to more accurately generated training data but we can also use the context to reduce the runtime of the overall labeling process in step 2 as described in Section 9.3. To show these possible runtime gains from steering, we conducted an experiment in which our LF EMD is executed in two different modes: (1) without the preselection of contextually similar numerical labeled columns and (2) with a selection. To be more precise in (1) the LF measures the similarity against all labeled numerical columns available and in (2) only against those ones which are embedded in the same context (table with semantic equally neighbored textual columns). Table 9.3 shows the results of these experiments by listing the runtime reduction that could be achieved per labeled data size. The reduction goes from about 25% (labeled data size 1) to almost 50% (labeled data size 5), demonstrating the higher the number of labeled data the higher the percentage of runtime reduction.



(a) Quality of generated training data per LF class



(b) Quantity of generated training data per LF class

Figure 9.9: Quality and quantity of generated training data for each class of LFs. (a) Each LF of *STEER* (non-numerical as well as numerical) creates high-quality training data. (b) Moreover, each LF contributes to the amount of generated training data. However, the number of generated training samples (columns and their semantic type) varies per data set and LF.

9.5.5 Ablation Study

In the ablation study, we analyze the quality and quantity of generated training data for each class of LFs.

9.5.5.1 Efficiency of the LFs

To evaluate the contribution of each LF to the total amount of training data generated, we analyzed the generated training data per LF separately in an ablation study. The results per LF are shown in Figure 9.9 while Figure 9.9a shows the quality (weighted preci-

sion) and Figure 9.9b the quantity (number of labeled columns) for all datasets separately.

LF *EmbClus*. For the LF *EmbClus* we plot the results for the labeled data sizes 1 & 5 showing the aspect that the larger the number of already labeled data, the more new unlabeled data can be labeled with additional small quality improvement. This is due to the fact that more labeled data is included in the clustering and thus the labeling process is extended and improved. At labeled data size of 5, the LF produces 529 (96% of the total gen. train. data) for *Public BI*, 4.902 (6.4% of the total gen. train. data) for *Turl-Corpus* and 529 (16.5% of the total gen. train. data) for *SportsDB*. If we look at the quality of the generated training data, we achieve an average precision of 0.94 for the datasets *Public BI* & *TURL-Corpus* and a perfect result of 1.0 for *SportsDB*. Notice that for this LF we do not list results for *Pubic BI Num*, because the same textual columns are included as in *Public BI* and therefore the results are the same. In summary, this demonstrates that this LF generates high quality labeled training data to adapt the model.

LF *CH* (column-headers). Focusing now on the LF *CH*, the quality of the generated training data for *Public BI* is overall high. Since the precision is at the score of 1 each label assigned by the LF to the 197 unlabeled columns is correct. For *Turl-Corpus*, the LF produces over 65% of all additional training data generated. The precision is also high at 0.73. The reason in comparison to the quality on *Public BI* is that frequently occurring types are mislabeled. For example, for the semantic types *music.album & music.artist* the LF generates over 5.000 new labeled columns with a precision over 0.94, but for the types *music.genre & music.composition* just 80 with a precision below 0.2 since column headers are too general. In conclusion, this LF works (almost) perfectly for the *Public BI* corpus and the model can benefit from the new training data when re-training. However, also for the *TURL-Corpus*, which is more complex, we can generate a good number of high-quality labels; i.e., only a small amount of training data with low quality is generated for semantic types (macro score), which may be also improved or resolved when merging the outcomes of the different LFs.[8]

LF *VO* (value-overlap). As described in Section 9.4, for the class of LF *VO*, a domain expert can provide a list of common values to a semantic type, which is then used by the LF to generate the training data. In our current prototype version, we provide such a list for two selected semantic types for the *Public BI* data corpus and 11 semantic types for the *Turl-Corpus*. As an example, for the *Public BI* semantic type *language*, we provide

[8]Notice that we do not implement LFs of this class for the *Public BI Num* & *SportsDB* datasets.

a list with the values *{de, en, fr, es, ...}*. In case of the *Turl-Corpus* semantic type *film.flim_genre*, the domain expert defines the common values *{crime, horror, romance, action, ...}*. For each selected type, a LF of this class is instantiated and executed on its own. In order to show the precision and the number of generated training data, we have averaged the precision values and summed up the number of generated training data over the individual outcomes per semantic type. For both *Public BI* types the LFs generate in total 40 new labeled columns with a precision of 0.79. In addition, we can see that for the semantic types belonging to the *Turl-Corpus*, we can achieve an average precision of 0.66 and generate over 12.000 new labeled columns. To give an insight into one of the best LFs here, the LF for *soccer.football_team* can label 7.238 columns with a precision of 0.98 by providing a short list of the most famous football teams.[9]

LF *VP* (value-pattern). Similar to the previously discussed LF, even in this case the user provides a list of patterns in form of regular expressions that are then used in the labeling process. In our current implementation, we defined such patterns for two *Public BI* types and eight *Turl-Corpus* types. To give an example, for the semantic type *award.award_category*, we define the list of patterns as follows {best*, worst*}. Meaning that all values starting with *best* or *worst* are counted as a pattern match. In case of the *Public BI* dataset, the type specific LFs can label 50 unlabeled columns with an overall precision of 0.66. If we analyze the LFs for *Turl-Corpus*, we create almost 1.800 new labeled columns with a slightly higher precision of 0.75. In this class of LF the pattern definitions are very important. The defined patterns should not be under-generalized (e.g. resulting in too infrequent matches and therefore the columns related to the semantic type are not found) and also not be over-generalized (e.g. resulting in too many matches for columns that actually do not belong to the semantic type) [43].

LF EMD. For the numerical LF *EMD* we plot the results for the labeled data sizes 1 & 5. The figure shows the precision and the number of generated training data for *Public BI Num* and *SportsDB*, since these are the only datasets that contains numerical columns and therefore *Steered-Labeling* with the LF *EMD* was applied. Looking at the weighted precision of the generated training data, we can see a constant value of 0.97 on *Public BI Num*, whereas on *SportsDB* there is an increase from 0.86 to 0.95. This improvement comes from the fact that the LF benefits from a higher amount of existing already labeled columns because more similarity measurements can be made from unlabeled to labeled numerical column and thus the precision of matches increases. Overall the precision

[9] Notice that we do not implement LFs of this class for the *Public BI Num* & *SportsDB* datasets.

values on both datasets demonstrates that the LF *EMD* extracts training data with a very high quality. Looking at the number of generated training data we see on *Public BI Num* values from 76 to 529 and on *SportsDB* values from 529 to 727 over the labeled data sizes 1 to 5. This increase comes for the same reason as just mentioned. The higher the amount of already labeled data the higher the probability to find a semantic match. Overall these results demonstrate that the LF *EMD* can annotate unlabeled numerical data in a good manner and therefore make a high contribution to the generated training data that can be used to adapt a model to a data lake containing high numbers of numerical data.

Summary. The quantity and quality of the generated training data are high for all used fictive data lakes. Consequently, as we have seen in the experiment before, the new training data can lead to a significant improvement of a learned metadata annotation model after re-training.

9.5.5.2 Analyze Generated Training Data

To better understand the gains of *STEER*, we now analyze the overall generated training data which was aggregated across the LFs by applying the majority vote. The aggregated training data represents the generated training data of *STEER* that we use for re-training the models. As in the previous section, we focus on two aspects for the analysis: quality and quantity of the generated training data to show that a significant amount of data is generated which provides high-quality (i.e. correct) labels.

The results of this analysis for all data sets are shown in Figure 9.10. Since only *Public BI Num* and *SportsDB* are containing labeled numerical data columns, *Steered-Labeling* training data generation is only applied to these datasets. To report the quality of the generated training data, we plot the macro and support weighted precision (i.e., the fraction for which the LFs assign the correct type). For showing the quantity, we plot the number of table columns, which receive a label from the labeling framework and thereby resulting as additional training data in percentage to the total amount of unlabeled columns available.

Overall, we can see that for the datasets *Public BI*, *Public BI Num* and *SportsDB* the generated training data achieves very high weighted precision values of more than 0.9. With *SportsDB*, we even reach a quality of 0.93/0.96 (macro/weighted). Looking at the

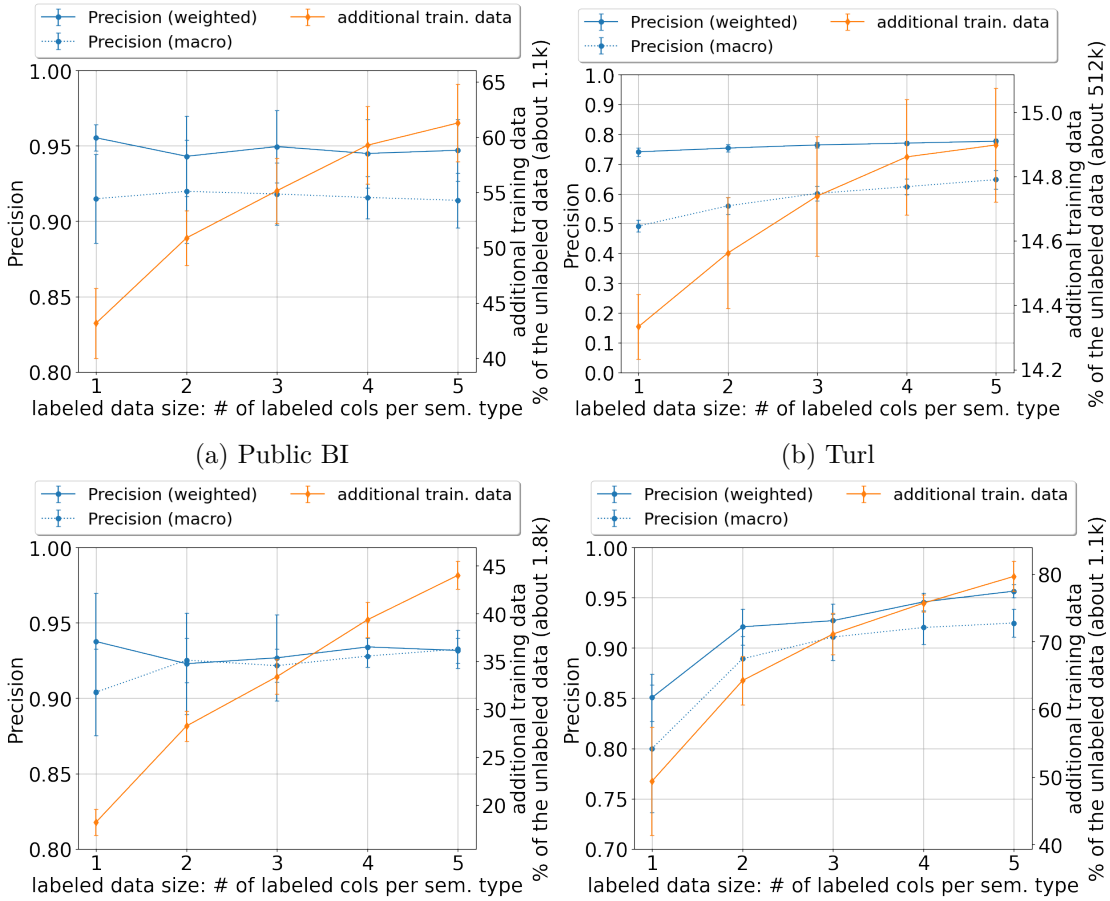
generated training data quality of *TURL-Corpus*, we see lower but still good precisions of 0.6/0.75, which also leads to benefits when it is used as additional training data as shown before (see Figure 9.5). Considering the amount of the generated training data, we achieve the highest value on *SportsDB* of up to 80% of the available unlabeled data while for *TURL-Corpus* we only label about 15%. It is important to note, however, that the *TURL-Corpus* is significantly larger than all other corpora and in absolute quantity we generate the largest corpus of additional training data with *STEER* on the *TURL-Corpus*.

9.6 Related Work

Data Programming. Our work is related to data programming which also uses labeling functions for training data generation. Existing approaches for data programming can be split into two directions. Weak Supervision Approaches (e.g. [27], [92], [93]) establish and use user-defined LFs that implement individual heuristics to assign labels to an entity. Typically several LFs are combined to produce the final result. Assisted Data Programming Approaches (e.g. [71], [79]) help the user to define labeling rules or functions by interactively guide them through the creation process (see [21] for a comparison of the different approaches). Some can even automatically define LFs as in WITAN [21] or Snuba [104]. However, all data programming approaches with a focus on labeling records in general, missing the possibility to adapt them to the special labeling problem of table columns. As such, to the best of our knowledge there exists no data programming approach that supports a multi stage labeling process, as introduced by *STEER* for high quality LFs on numerical semantic types except some early proposals [48] that are in an initial (vision) status (i.e., neither code and results are published).

Semantic Type Detection. Several approaches exist for semantic type detection. Traditional approaches rely on the metadata provided by the data sources (e.g. [37], [73], [4]). These approaches fail if column metadata information is missing or if the information is of poor quality as it is often the case in data lakes.

Search-based Approaches. Search based solutions overcome these shortcomings as they use the content of the column as information source. They often rely on regular-expressions and dictionary lookup (e.g. [73, 74]). Auto-Tag [43] provides advanced techniques to develop and adopt the search rules using a combination of automatic labeling for standard



(a) Public BI

(b) Turl

(c) Public BI Num (*STEER*)

(d) SportsDB (*STEER*)

Figure 9.10: Quality (blue) and quantity (orange) of generated training data. Quality is reported with macro and weighted precision. Quantity is shown as the number of additional generated training data in percentage to the total amount of unlabeled columns available.

types (e.g. IP-Address, Zip Code), and labeling by example approach for custom types (Customer ID, Part ID). These approaches are especially good in adapting to new datasets but often fail to provide a generic solution for the problem.

Machine Learning Approaches. Machine learning (ML) based approaches focus on the content of the columns but may take given metadata into account. Semantic Typer [91] uses TF-IDF and value distribution comparisons to capture column characteristics and map them to a semantic label. DCoM [70] is based on NLP techniques, treating the content of the columns as text to train a model for semantic type prediction. Doduo [100] extends this concept, by serializing the table and column content that is used afterwards as input for a general pre-trained language model. For the finetuning step a output layer is attached to the language model and the whole model is fine-tuned for semantic type prediction with labeled table columns. Turl [20] follows the pre-training/fine-tuning paradigm and relies on a multi-label setting. Pre-training uses unlabeled tables and both Masked Language Model and Masked Entity Recovery to learn embedding representations of these tables. For semantic type labeling, a labeled training set is needed to learn the semantic type to embedding representation mapping. Other pre-trained models such as TUTA [107], TABERT [112] and TAPAS[44] have developed alternative table encoders that perceive the structural and positional information but were not adapted to semantic type prediction. Sherlock [50] and its successor SATO [113] generate several features from the individual column values to train a deep neural network for semantic type detection. SATO also uses context information, notably the semantic type of the other columns of a table, to enhance the prediction performance. All ML approaches need labeled training data to create their model, that might not be available on a data lake. After model creation the amount of supported semantic data types is fixed, which limits their extensibility and causes problems (covariate-shift, label-shift, out-of-distribution data) when applied to new data as categorized by [48].

First approaches trying to overcome the lack of generality of ML approaches are ColNet [15] and SigmaTyper [48]. ColNet uses deep neural networks in combination with a given knowledge base (KB). The KB is used to generate labeled sample data to train the model. The semantic type prediction is done by an ensemble method that combines the prediction of the model and the vote of the KB. SigmaTyper suggests a data programming by demonstration approach, inferring labeling functions that adopt the ML model to a domain specific setting. ColNet is only evaluated on a small dataset and SigmaTyper has not published any evaluation results, preventing a final assessment of these approaches.

9.7 Conclusions

Detecting semantic types for columns of datasets stored in data lakes results in an enormous benefit building a data catalog to address the data discovery problem. While recent papers have shown initial results for learned approaches that can be used for extracting semantic types, they require high overhead for training data generation to adapt them to new data lakes which come with new semantic data types and do not cover the wide spectrum of different data characteristics. To tackle this problem, we suggested our new solution *STEER* and the underlying *Steered-Labeling* for generating new labeled training data and use this for re-training the existing model. Evaluations of *STEER* on four different datasets show that our approach leads to huge performance gains.

10 SportsTables: A new Corpus for Semantic Type Detection (Extended Version)

Abstract

Table corpora such as VizNet or TURL which contain annotated semantic types per column are important to build machine learning models for the task of automatic semantic type detection. However, there is a huge discrepancy between corpora and real-world data lakes since they contain a huge fraction of numerical data which are not present in existing corpora. Hence, in this paper, we introduce a new corpus that contains a much higher proportion of numerical columns than existing corpora. To reflect the distribution in real-world data lakes, our corpus SportsTables has on average approx. 86% numerical columns, posing new challenges to existing semantic type detection models which have mainly targeted non-numerical columns so far. To demonstrate this effect, we show in this extended version paper of [59] the results of an extensive study using four different state-of-the-art approaches for semantic type detection on our new corpus. Overall, the results demonstrate significant performance differences in predicting semantic types for textual and numerical data.

Bibliographic Information

The content of this chapter was previously published in the peer-reviewed work Sven Langenecker, Christoph Sturm, Christian Schalles, and Carsten Binnig. "SportsTables: A New Corpus for Semantic Type Detection (Extended Version)." In: *Datenbank-Spektrum* 23.2 (2023). DOI: 10.1007/s13222-023-00457-y. URL: https://doi.org/10.1007/s13222-023-00457-y. The contributions of the author of this dissertation are summarized in Chapter 4.

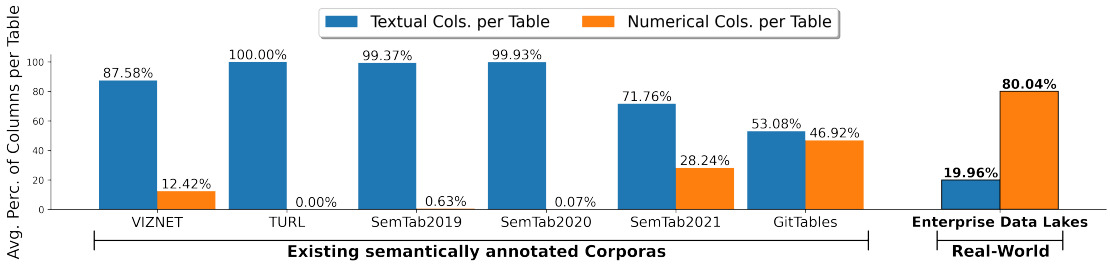


Figure 10.1: Average percentage of textual and numerical based columns per table in existing semantically annotated corpora[1] (left bars) compared to real-world data lakes (right bar). This shows the fact that there is a significant shift in the ratio of textual to numeric columns per table from existing corpora to real data lakes. Since all existing semantic type detection models were developed by using the existing corpora, shortcomings in validating the models on numerical data are present and it has not yet been studied in depth how well the models can perform on datasets containing a high proportion of numerical data.

10.1 Introduction

Semantic type detection is important for data lakes. Semantic type detection of table columns is an important task to exploit the large and constantly changing data collections residing in data lakes. However, manually annotating tables in data lakes comes at a high cost. Hence, in the past a lot of approaches have been developed that automatically derive semantic types from table data [20, 50, 100, 113]. Many of the recent approaches use deep learning techniques to build semantic type detection models. As such, corpora containing large amounts of table data with assigned semantic types are required for training and validating. Existing annotated table corpora (e.g. VizNet, TURL) primarily contain tables extracted from the web and therefore limit the capability to represent enterprise data lakes.

Existing corpora and models fall short on real-world data lakes. However, as we can see in Figure 10.1, almost all existing corpora that provide annotated columns labeled with semantic types have a lack of table columns that contain numerical data, and tables in these datasets incorporate either only for a very high percentage of textual data. Only GitTables[47] contains a more balanced ratio of textual and numerical data. Nevertheless, compared to real enterprise data lakes, there is a significant discrepancy in the ratio of textual to numerical data. An inspection of a large real-world data lake at a company[2] has shown that on average approx. 20% textual data and 80% numerical

data are present (see. Figure 10.1 bars on the right). Moreover, semantic type detection models [20, 50, 100, 113] that are trained on the available corpora also mainly target non-numerical data.

Semantic type detection for numerical data is challenging. Detecting semantic types of numerical columns is generally harder than for textual columns. For example, for a textual column with the values {Germany, USA, Sweden, ...} a model can easily identify the semantic type *country*. Instead, for a numeric column with e.g. the values {20,22,30,34,...} it is not that straightforward and several possibilities for a matching semantic type exist such as *age, temperature, size, money*. The fundamental reason here is that numerical values can be encoded with much fewer bits than string values [98], resulting in a lower overall entropy and thus providing less information content that can be used by a machine learning model to infer the underlying semantic type. Due to the existing corpora providing annotated columns that have been used to create and validate semantic type detection models, we see several essential shortcomings that could not be addressed until now because of the absence of a sufficient dataset for this purpose.

Contributions. In this paper, we thus contribute a new corpus containing tables with semantically annotated columns with numeric and non-numeric columns that reflect the distribution of real-world data lakes. We will make the corpus available which should stimulate research directions such as working on new model architectures that can reliably annotate types to numeric and non-numerical columns. In the following, we discuss the main contribution of this paper.

As a first contribution, we present and provide our new corpus SportsTables[3]. To the best of our knowledge, SportsTables is the first corpus with annotated table columns, which contains a significantly larger proportion of numerical data than textual data. In total, the tables in our corpus have on average about 3 textual and 18 numerical columns. Moreover, the tables in our new corpus are much larger in both the number of columns and the number of rows than in existing corpora which better reflects the characteristics

[1]Notice that for GitTables we only considered the tables and columns labeled by terms from DBpedia using the semantic annotation method as described in the GitTables paper. Therefore our reported ratios of textual and numerical data differ from those shown in the GitTables paper because they consider all data, whether annotated or not.
[2]The analyses were done at the company LÄPPLE AG
[3]Available on https://github.com/DHBWMosbachWI/SportsTables.git

of real-world tables.

As a second contribution that comes together with the corpus, we specify an ontology with semantic types for the sports baseball, basketball, football, hockey, and soccer. This ontology provides fine granular semantic types for all kinds of sports we considered to build SportsTables and allows us to semantically describe each occurring table column, which is not possible with the current ontologies (e.g. DBpedia) at this level of detail. Using a manually created dictionary, we assign a semantic type to each existing column in SportsTables.

As an extension of [59] and as a third contribution, we present in this paper results of extensive experimental analyses using our new corpus on four different state-of-the-art semantic type detection models. Overall, we can see that when trained on our new corpora, the models can improve the performance on numerical data types. However, one shortcoming that our analysis shows is that the current model architectures are not targeting numerical columns. To be more precise, our analysis demonstrates that textual data columns are mostly correctly semantically interpreted with the models (best F1-Score 0.98), but on numerical data columns, the models only achieve F1-Scores in the range of 0.31-0.7. This large difference indicates that new model architectures that take the characteristics of numerical columns into account are needed which is a direction that could be stimulated by the availability of our corpus.

Outline. In Section 2, we first provide an overview of existing corpora which was used to build and validate semantic type prediction models and discuss their characteristics and statistics. Afterward, in Section 3, we then introduce our new corpus SportsTables and describe in detail how we created the corpus and labeled the table columns with semantic types. Section 4 first demonstrates the main characteristics of our corpus before we then show the results of using our new corpus on the different semantic type detection models. Next, further research challenges are discussed in Section 5 before Section 6 concludes the paper.

10.2 Existing Corpora

In the following, we describe different existing corpora that contain annotated table columns and therefore can be used to build and validate semantic column type detection

Table 10.1: Corpus statistics about the number and sizes of tables. Additionally, we see the average number of textual and numerical columns per table for each existing annotated corpora and our new SportsTables corpus. This shows the absence of numerical data columns per table in most existing corpora and the dominance of textual data columns per table in all existing corpora. Instead, our new corpus SportsTables contains on average over 6 times more numerical columns than textual columns.

Corpus	Tables	Cols	$\overline{Cols/Table}$	$\overline{Text.\ Cols/Table}$	$\overline{Num.\ Cols/Table}$	$\overline{Rows/Table}$
VIZNET	78,733	120,609	1.53	1.34	0.19	18.35
TURL	406,706	654,670	1.61	1.61	0	12.79
SemTab2019	13,765	21,682	1.58	1.57	0.01	35.61
SemTab2020	131,253	190,494	1.45	1.45	0.001	9,19
SemTab2021	795	3,072	3.86	2.77	1.09	874.6
GitTables	1.37M	9.3M	6.82	3.62	3.2	184.66
SportsTables	**1,187**	**24,838**	**20.93**	**2.83**	**18.1**	**246.72**

models. We summarized the main statistics for all corpora in Table 10.1.

VizNet [45]. The original VizNet corpus [45] is a collection of data tables from diverse web sources ([11, 81, 88, 105]) which initially do not contain any semantic label annotation. The corpus we consider in this paper is a subset of the original VizNet corpus, which was annotated by a set of mapping rules from column headers to semantic types and then used to build and validate the Sherlock [50] and Sato[113] prediction models. The corpus contains in total 78,733 tables and 120,609 columns annotated with 78 unique semantic types. Overall, the tables in the corpus contain only 1.53 columns and 18.35 rows on average. Furthermore, the distribution of the column data types is 87.58% textual and 12.42% numerical and thus leads to the shortcomings as described before.

TURL [20]. The TURL corpus uses the WikiTable corpus [9] as basis. To label each column they refer to the semantic types defined in the Freebase ontology [34] with a total number of 255 different semantic types. What distinguishes TURL from other corpora is that columns can have multiple semantic types assigned. In total, there are 406,706 tables resulting in 654,670 columns, and on average a table consists of 1.61 columns and 12.79 rows. Again, these are rather small dimensions. In addition, the Turl corpus includes no numerical data at all, which leads to the shortcomings mentioned above when using the corpora.

SemTab. SemTab is a yearly challenge with the goal of benchmarking systems that match tabular data to knowledge graphs since 2019. The Challenge includes the tasks

of assigning a semantic type to a column, matching a cell to an entity, and assigning a property to the relationship between columns. Every year, the challenge provides different datasets to validate the participating systems against each other. In this paper we observed the provided corpora for the years 2019 [40], 2020 [19, 41], and 2021 [1, 19, 42, 46, 82]. Statistic details of the corpora are shown in Table 10.1. In case more than one dataset was provided per year, we aggregated the statistics over all datasets included in the challenge. While SemTab2019 consists of 13,765 tables and 21,682 columns in total, there are 131,253 tables and 190,494 columns in SemTab2020. In both corpora, the dimensions of the included tables are rather small (on average 1.58 columns and 35.61 rows in 2019 and 1.45 columns and 9.19 rows in 2020). In SemTab2021, the contained tables are the largest in terms of rows with almost 875 on average. However, the number of columns (3.86 on average) is only moderate and the corpus in general is the smallest with a total of 795 tables and 3,072 columns. Numerical data is almost nonexistent in the first two years (0.63% in 2019 / 0.07% in 2020), increasing to 28.24% numeric columns per table on average in 2021, which is still not comparable to the number of numeric data in real world data lakes.

GitTables [47]. GitTables is a large-scale corpus of relational tables created by extracting CSV files from GitHub repositories. Table columns are labeled with semantic types from Schema.org [35] and DBpedia [6] using two different automated annotation methods (syntactically/semantically similarity matching from semantic type to column header). In this paper, we have focused on the annotations origin from DBpedia and the results of the semantic annotations method as described in the GitTables paper [47]. This leads to a corpus containing over 1.37M tables and 9.3M columns in total. Although this is by far the largest collection of data tables, the dimensions of the tables are on average only moderate with 6.82 columns and 184.66 rows. Overall, GitTables incorporates the most numeric data with an almost balanced ratio of 53.08% textual and 46.92% numerical columns per table.

Discussion. The overview in Table 10.1 and the discussion before shows that most existing corpora contain no or only a minimal fraction of numerical data types which is very different from real-world data lakes. An exception is GitTables which has a much higher ratio of numerical columns. However, as we show in Section 10.4, GitTables still lacks a good coverage of different numeric semantic types which is one important aspect that we tackle with our new corpus SportsTables which covers a wide variety of different numerical semantic types. Moreover, another important (but orthogonal) aspect is that

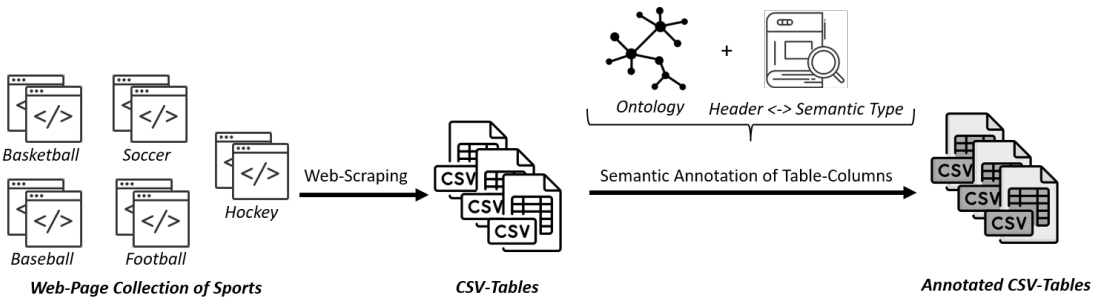


Figure 10.2: Overview of the implemented pipeline to build SportsTables. We use web-scraping techniques to extract HTML tables from a manually defined web page collection for each selected sport and convert the tables to CSV files. With the help of a defined ontology and a manually created dictionary that maps column headers to semantic types, we annotate each table column with an appropriate semantic type.

existing corpora include a large number of tables. However, on average the tables are very small in terms of the number of columns and the number of rows. Instead, our new corpus SportsTables contains fewer tables, but on average a significantly higher number of columns and rows per table to better reflect the characteristics of real-world data lakes.

10.3 The SportsTables Corpus

In the following, we will introduce our new corpus and describe in detail the implemented construction pipeline to build SportsTables.

Methodology to generate the corpus. Figure 10.2 gives an overview of our implemented pipeline to generate the new corpus. The main idea was to collect data tables from different sports domains such as soccer, basketball, baseball, etc. since data tables coming from such kinds of sources are rich in numerical columns. For example, a soccer player statistic table of a soccer season contains typically 3 textual columns (e.g., player name, team name, field position) and 18 numerical columns (e.g., goals, games played, assists). Hence, building a collection of such tables will lead to a corpus that contains many numerical columns which are in addition semantically interpretable. As a result, the corpus will enable performance analysis of semantic type prediction models in a much more rigorous manner regarding numerical data.

Scraping data from the web [23].A vast amount of data covering information about player statistics, team statistics, coach statistics, or season rankings of different sports are available on various web pages. Therefore, for collecting the data, we built a data collection pipeline based on web scraping technology[23]. In the first step, we manually searched and defined a set of different web pages for each of the selected sports of which we want to scrape contained data tables (left side of Figure 10.2). We first converted each HTML table on the web pages to Pandas-Dataframes using Python and then saved them as CSV files (center of Figure 10.2), since this file format is most known and used to store raw structured data [76]. During the scrape process, we kept the respective column headers from the original HTML table and used them as headers in the CSV file.

Annotating columns with semantic types. Due to the low granularity of existing ontologies (e.g. DBpedia) regarding semantics of a given sport, we manually created an ontology-like set of valid semantic types for all sports. For example, in DBpedia there is the type *Person.Athlete.BasketballPlayer*, but semantic labels in the particular that would match individual numerical columns such as *NumberOfGoals* are not defined. Next, we annotated all table columns with semantic types using a manually created dictionary that maps column headers to matching semantic types from our created set. Since the column headings were in many cases identical if the semantic content was the same, this procedure significantly reduces the manual labeling effort. In addition, to ensure that the labels are of very high quality in terms of correctness, we manually checked each assignment based on the content of the columns.

10.4 Analysis of the Corpus

This section describes the characteristics of SportsTables in detail and then demonstrates the significant impact of these characteristics on semantic type prediction frameworks in a study where we apply the corpus to several state-of-the-art semantic type detection models.

10.4.1 Corpus Characteristics

In the following. we discuss the statistics of the SportsTables corpus and compare them to the existing corpora.

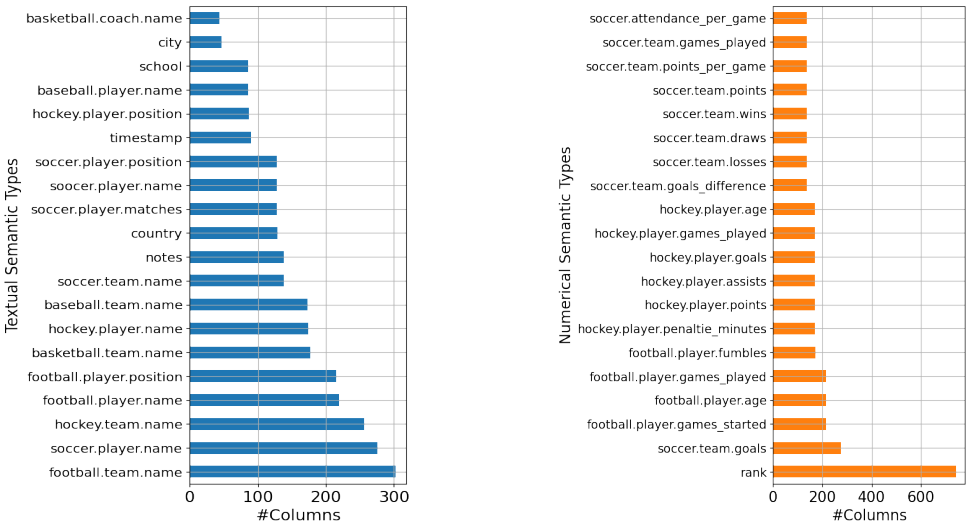
Table 10.2: Statistics about the number of unique semantic types. Showing that our new corpus has a higher proportion of numerical semantic types than textual semantic types in contrast to the existing corpora. In addition, there is a large overlap of semantic types used for textual and numeric columns in the existing corpora. In comparison, the semantic types in SportsTables are disjoint for the two column data types.

Corpus	#Textual Sem. Types	#Numerical Sem. Type	#Total Sem. Types
VIZNET	78	44	78
TURL	255	0	255
SemTab2019	360	19	360
SemTab2020	5804	32	5832
SemTab2021	177	93	251
GitTables	2646	2426	2693
SportsTables	**56**	**419**	**475**

Data statistics (Table 10.1). Using the described pipeline for creating SportsTables, a total of 1,187 tables which comprises 24,838 columns (approx. 86% numeric and 14% textual) are scraped from the web resulting in 20.93 columns (2.83 textual and 18.1 numerical) per table on average. This ratio of textual to numerical columns, as well as the total average number of columns in a table, differs significantly from existing corpora.

In Table 10.1 we can also see a comparison of the average number of textual and numerical columns per table of SportsTables versus that of the existing corpora. Here we can see that numerical columns only exist in the corpora VizNet with 0.33, SemTab2021 with 1.09, and GitTables with 3.2 columns per Table. Compared to GitTables, in SportsTables there are thus on average over 6 times more numeric columns per table. Moreover, as we discuss below, our corpus uses a much richer set of numerical data types that better reflects the characteristics in real-world data lakes which is very different from GitTables. For example, when looking at the semantic types that are assigned to numerical columns in GitTables, more than half (393,925) of the columns are labeled with just a single type *Id*.

In terms of the total number of columns, the tables in SportsTables (20.93 columns per table) are on average about 3 times wider than in GitTables (6.82 columns per table), which contains the widest tables among the existing corpora. As such, the number of columns in tables of SportsTables are reflecting better the width when comparing this to the characteristics of the tables in real-world data lakes which we analyzed. Moreover, considering the average number of rows per table, it can be seen that the tables in SportsTables have on average 246.72 rows. In comparison, tables in SportsTables are



(a) Top 20 textual sem. types

(b) Top 20 numerical sem. types

Figure 10.3: Semantic type annotation statistics of SportsTables. (a) Shows column annotation counts of the top 20 textual semantic types. Across all kinds of sports, *player.name* and *team.name* are the most common. (b) Shows column annotation counts of the top 20 numerical semantic types. A dominant type here is *rank*, which describes a column containing the placements of e.g. a team in a season standings table.

larger on average than in many other corpora where tables have typically fewer rows.

Annotation statistics. Semantic type annotation follows a two step process. First, we establish a directory with manually defined mappings from column header to semantic type for each existing header. Second, we label each column with the semantic type listed in the directory for its header. As a result, 56 textual and 419 numerical semantic types are present in the corpus. Thereby textual semantic types are those which specify textual columns and numerical types are those which specify columns containing numeric values. To compare the annotation statistics, we also counted the number of textual and numerical semantic types in an analysis of the existing corpora. The results of these analyses can be seen in Table 10.2. Different from our corpus, the sets of textual and numerical types are not disjoint in all other corpora (except TURL where no numeric values are present). This indicates that individual semantic types were assigned to both textual and numerical columns which is problematic if semantic type detection models should be trained and tested on these corpora. In particular, GitTables has a very large overlap and almost all semantic types are used in both column data types. To give an

example, in GitTables the semantic types *comment, name* and *description* are assigned to both column data types. Next, we take a closer look into the semantic types of our corpus.

Figure 10.3a and Figure 10.3b show the top 20 semantic types (textual and numerical) in regards to how often they were assigned to a table column. It can be seen that the most common textual types across all sports are *player.name* and *team.name*. These are types that occur in almost every table. Other types such as *country* or *city* are also common, describing, for example, the player's origin or the team's hometown. Among numeric semantic types, *rank* is by far the most common and is present in almost all tables. The type describes a column containing the placement of e.g., a team in a "seasons standing" table or a player in a "top scorer" table. All other numeric semantic types show mainly an equal distribution of the frequency, which is a good precondition for training machine learning models.

SportsTables vs. GitTables. Since GitTables is the largest corpus with the most tables, one could argue that a subset of GitTables would result in a new corpus with similar characteristics as SportsTables. To analyze this, we executed a small experiment in which we filtered out only tables from GitTables where the number of textual and numerical columns (min. 3 textual and 18 numerical columns) is at least the same as it is in SportsTables. The result was a corpus containing a total of 16,909 tables and 743,432 columns. On average a table has 12.53 textual columns, 31.43 numerical columns, and 17.35 rows. However, looking at the semantic types that are assigned to numerical columns, more than half (393,925) of the columns are labeled with the type *Id*. In terms of training and validating semantic type detection models, this is rather an unfavorable type representing no semantically meaning. Moreover, the next 5 most common numerically based semantic types are *parent, max, comment, created* and *story editor*, constituting a large proportion of the columns. The assignment of these types to numerical data is slightly less understandable and indicates a lack of quality in the automatically generated labels for table columns.

10.4.2 Study of Using SportsTables

In the following, we report on the results of using four different state-of-the-art semantic type detection models on our new corpus. With this, we want to measure how well the semantic types in our corpus can be inferred by the models with a special focus on how

each performs on textual and numerical columns.

Models. As state-of-the-art models, we used Sherlock [50], Sato [113], Dosolo [100] and Doduo [100] in our experiments, which all use deep learning techniques to build the model. In the following, we describe the fundamental functionalities of each model to explain the differences between them.

Sherlock: Sherlock utilizes multiple feature sets, such as character distributions, word embeddings, paragraph embeddings, and column statistics (e.g., mean of numerical values), in its single-column prediction model. Each columnwise feature set, except for the column statistics, is processed by a multi-layer feature specific subnetwork to generate compact dense vectors. The resulting outputs from the subnetworks, along with the column statistics features, are then inputted into the primary network, which comprises two fully connected layers.

Sato: Building upon Sherlock, Sato is a multi-column prediction model that incorporates LDA features to capture table context and integrates a CRF layer to account for column type dependency in its predictions. With this, Sato's prediction quality improves over Sherlock on the VizNet data corpus.

Dosolo & Doduo: Dosolo & Doduo are both models from [100] and use pre-trained language models (LM) (e.g. BERT) combined with an attached output layer to implement a model for the semantic type classification task. Given that LMs receive token sequences (i.e.text) as input, it is essential to convert a table into a token sequence so that the LM can process it. What distinguishes Dosolo and Doduo is the way in which a table is serialized into a token sequence. Dosolo implements a columnwise serialization where each column C and its values $v_1, ...v_m$ of a table is separately serialized as follows: $serialize(C) ::= [CLS]v_1, ...v_m[SEP]$. In contrast, Doduo is a tablewise model designed to process an entire table as input. To accomplish this, Doduo serializes the complete table and its entries as follows: for each table that has n columns $T = (c_i)_{i=1}^{n}$, where each column has N_m column values $c_i = (v_i^j)_{j=1}^{m}$, they let $serialize(T) ::= [CLS]v_1^1...[CLS]v_1^n...v_m^n[SEP]$. In both sequences, the special token [CLS] marks the beginning of a new table column and [SEP] the end of a token sequence. The major difference between the two approaches and their serialization techniques is that with Dosolo a column type is predicted independently of other data in the table (e.g. neighboring column values), whereas Doduo model captures the table context to make a

prediction of a column type. In summary, we can conclude that Sherlock and Dosolo are single-column (columnwise) prediction models that only take into account the individual column values for the prediction. In contrast, Sato and Doduo are multi-column (or tablewise) prediction models, which consider table contexts for predicting the semantic type of an individual column.

Experiment setup. For the experiments, we split the SportsTables corpus into training, validation, and test set. While creating the splits, we first extracted 20% of the data for the test set and then another 20% of the remaining 80% for the validation split. The rest of the data was used as the training set. We used the four pre-trained models as described above and re-trained them with the training data set. During the re-training, we replaced the last layer of the different models to support the number of semantic types that occur in SportsTables and then re-trained the entire neural network. In order to optimize the hyperparameters, we measured the performance of the respective re-trained models against the validation split. To report the final performance, we applied the re-trained models to the 20% test data set. For obtaining statistically reliable results, we ran each experiment with five different random seeds and report the mean and standard deviation over multiple runs.

Results of study. Figure 10.4 shows the results of the experiments reporting the support weighted and macro average F1-Scores in individual subplots for all four models. For each model, we plot the F1-Score across all semantic types (numerical & non-numerical) to show the total performance, but also the separate average F1-Score for only textually and numerically based semantic types, respectively. In the following we want to discuss the main aspects of the results in detail.

Non-numeric vs. numeric: As we can see in the figure, there is a significant performance difference between predicting textual and numerical semantic types for all models. While textual columns can be predicted with performances in a very promising range of 0.82-0.98, the performances for numerical columns are rather moderate ranging from 0.31 to 0.7. On average, the difference in F1-Score between textual and numeric types is 0.35 across all models. These results demonstrate that the models can better handle textual data and determine its associated semantic types more accurately than numerical data. Looking at the total performances over all types for each model, we see that they are rather moderate in the range of 0.38 to 0.74, but these insufficient

results are primarily caused by poor prediction performances on the numerical based types.

Columnwise vs. tablewise: Looking and comparing the results of the columnwise models Sherlock & Dosolo and the results of the tablewise models Sato & Doduo, we observe that the tablewise models outperform the columnwise models. The results underline the known importance of considering not only individual column values for the task of semantic type detection of table columns but also to involve the table context. In particular, what we can see from the comparison of Dosolo and Doduo is how important it is, especially for numerical based columns, to include table context data for semantic type detection. As described above, numerical values provide less information content that can be used by a machine learning model to identify the type and therefore Doduo doubles the performance of Doslo by considering the complete table context. However, the resulting performance of 0.62 is rather moderate and demonstrates the shortcomings of the model on numerical semantic types. Comparing Sherlock and Sato also reflects the advantages of a tablewise semantic column type detection, whereas the performance improvement on just numerical columns is not as significant as in Dosolo vs. Doduo. We will discuss the reasons for this in the following.

Sherlock & Sato vs. Dosolo & Doduo on numeric: As described above, Sherlock & Sato (same feature set) as well as Dosolo & Doduo (same LM model) are models with an identical foundation. Focusing only on the F1-Scores on the numerical types, one can see that Sherlock & Sato outperform Dosolo & Doduo. We think that this aspect is due to the fact that Sherlock & Sato extract features of numerical columns that better address numerical data (e.g. mean of individual digits occurring in a column), while in Dosolo & Doduo a LM model is used as the basis to encode the numerical column. LM models are optimized for text and can therefore not provide a representative encoding to infer the semantic type of numerical columns. Therefore, Dosolo & Doduo predictions on numerical columns are inferior to Sherlock & Sato.

10.5 Future Challenges

Detecting semantic types in real-world data lakes comes with many challenges that need to be addressed. In particular, based on our findings of the analysis using the different models in Section 10.4, we think that new model architectures are needed for detecting numerical data types which have different characteristics from non-numerical data. In

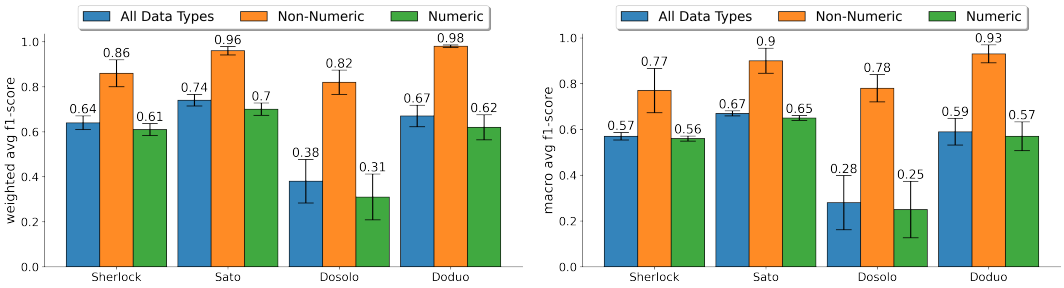


Figure 10.4: Results using different state-of-the-art semantic type detection models on our new SportsTables corpus. The overall differences in F1-Scores for predicting textual and numeric columns indicate that the models can handle textual data more effectively than numeric data.

the following, we list some of the challenges we think are important to be addressed. We hope that our corpus enables research on those challenges.

Embedding numerical data: Most state-of-the-art models like Dosolo&Doduo apply LMs like BERT [22] to encode literals to infer the semantic type of a table column. Since such approaches are optimized for textual data, using them on numerical data is not sufficient, as our experimental results show. Therefore, we need improved embeddings for numerical data, which can now be studied with SportsTables.

Leveraging numerical context: To improve the semantic type prediction of a table column, recent approaches like Sato [113], TURL [20] and Doduo [100] incorporate also context information like the table-topic or values from neighboring columns of the same table as described above. Given that tables in existing corpora contain almost entirely textual columns, the contexts (e.g. values from neighboring columns) used are rich in information and therefore also lead to performance improvements. However, it is unclear how effective this approach is in case the tables contain many numerical columns and only a few textual columns since the context information provided is reduced due to the lower entropy of numeric values as described before. Our first results using SportsTables show that in principle adding context information leads to an improvement on non-numeric and numeric types. However, we believe that specifically for the prediction of the numerical types the leverage of contextual informations needs to be researched in more depth.

Supporting wide tables: Existing datasets for semantic type detection consist of tables with small numbers of columns and rows. In nearly all corpora, the existing tables contain

on average less than two columns and less than 40 rows (see Table 10.1). Therefore, at the current state, it has not been analyzed how state-of-the-art models can handle such large tables. To give an example of why large tables could be a problem for recent models, we will briefly discuss this aspect on the Doduo model. As described above, Doduo uses pre-trained LMs (e.g., BERT) and hence serializes the entire table into token sequences with a fixed tensor length of 512 elements, which is given by the LM model. With this methodology of serialization and the fixed given tensor length, increasing the number of table columns means that decreasing number of values of each column can be included for serialization. For example, a table with 512 columns would allow only one value per column to be considered and this would most likely result in an insufficient semantic representation of the column based on that one value.

10.6 Conclusion

Existing corpora for training and validating semantic type detection models mainly contain tables with either only or a very high proportion of textual data columns and either no or just a limited number of numerical data columns. Therefore, it has not been studied precisely how well state-of-the-art models perform on a dataset with a very high percentage of numerical columns as it occurs in real-world data lakes. Moreover, tables in existing corpora are very small regarding the total number of columns and rows. To tackle these shortcomings, we built a new corpus called SportsTables which contains tables that have on average approx. 3 textual columns, 18 numerical columns, and 250 rows. With our new corpus, semantic type detection models for table columns can now be holistically validated against numerical data. We show results by using the different state-of-the-art semantic type detection approaches Sherlock, Sato, Dosolo, and Doduo on our new corpus and report significant differences in the performance of predicting semantic types of textual data and numerical data on all models. Finally, we think that the corpus is just a first step to stimulate more research on new model architectures that can better deal with numerical and non-numerical data types. The corpus is available on https://github.com/DHBWMosbachWI/SportsTables.git.

10.7 Acknowledgements

We thank the reviewers for their feedback. This research is funded by the BMBF projects AICoM and KompAKI (grant numbers 02P20A064 and 02L19C150) by the state of Hesse

as part of the NHR Program, as well as the HMWK cluster project 3AI (The Third Wave of AI). Finally, we want to thank DHBW Mosbach, hessian.AI, TU Darmstadt as well as DFKI Darmstadt for their support.

11 Pythagoras: Semantic Type Detection of Numerical Data Using Graph Neural Networks (Short Paper)

Abstract

Detecting semantic types of table columns is a crucial task to enable dataset discovery in data lakes. However, prior semantic type detection approaches have primarily focused on non-numeric data despite the fact that numeric data play an essential role in many enterprise data lakes. Therefore, typically, existing models are rather inadequate when applied to data lakes that contain a high proportion of numerical data. In this paper, we introduce *Pythagoras*, our new learned semantic type detection approach specially designed to support numerical data along with non-numerical data. *Pythagoras* uses a graph neural network based on a new graph representation of tables to predict the semantic types for numerical data with high accuracy. In our initial experiments, we thus achieve F1-Scores of 0.829 (support-weighted) and 0.790 (macro), respectively, exceeding the state-of-the-art performance significantly.

Bibliographic Information

The content of this chapter was previously published in the peer-reviewed work Sven Langenecker, Christoph Sturm, Christian Schalles, and Carsten Binnig. "Pythagoras: Semantic Type Detection of Numerical Data Using Graph Neural Networks (Short Paper)." In: *Lernen, Wissen, Daten, Analysen (LWDA) Conference Proceedings, Marburg, Germany, October 9-11, 2023*. Ed. by Michael Leyer and Johannes Wichmann. Vol. 3630. CEUR Workshop Proceedings. CEUR Workshop Proceedings, 2023, pp. 146–152. URL: https://ceur-ws.org/Vol-3630/LWDA2023-paper13.pdf. The contributions of the author of this dissertation are summarized in Chapter 5.

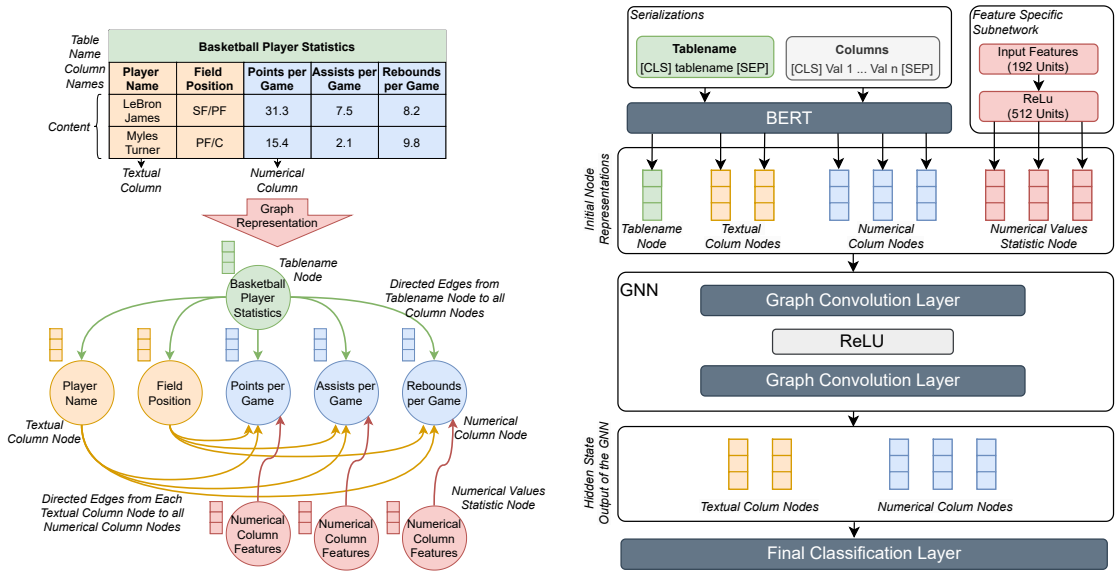
11 Pythagoras: Semantic Type Detection of Numerical Data Using Graph Neural Networks (Short Paper)

11.1 Introduction

Dataset discovery of numerical data is important in enterprise data lakes. Enterprise data lakes serve as invaluable repositories of diverse data types, enabling organizations to store and manage vast amounts of information [24]. In these data lakes, numerical data plays a dominant role, making up a much larger proportion compared to non-numerical data [59] and providing insights into various business domains, including finance, manufacturing, healthcare, and marketing. Such data often contain critical information such as sales figures, production metrics, customer demographics, and financial records. Therefore, it is essential to automatically detect the correct semantic type of table columns with numerical data enabling data scientists to find required data for downstream analysis and thus address the dataset discovery problem in data lakes [29, 53, 78].

Existing approaches are mainly designed for non-numerical data. In order to provide the task of semantic type detection, many solutions using deep learning techniques have been proposed in the past [20, 50, 61, 100, 113]. Unfortunately, all these existing approaches have primarily focused on detecting the semantic type of non-numerical data table columns, leaving a critical need for innovative approaches that effectively handle the detection of semantic types for numerical table columns [59].

Towards a new learned semantic type detection model for numerical data. In this paper, we introduce our new vision of a semantic type detection model called *Pythagoras*, which can not only predict the semantic type of non-numerical table columns with high accuracy but also of numerical table columns. To achieve this, the main idea of the new model architecture is to use graph neural networks (GNNs) together with a novel graph representation of tables and their columns. This graph representation includes directed edges to provide necessary context information (e.g. neighboring non-numerical columns) for predicting the semantic type of numerical columns using GNNs message passing mechanism. The graph representation and the new model architecture are the main contributions of this paper. Moreover, as a second contribution, we show initial highly promising results comparing Pythagoras against five existing state-of-the-art models on the *SportsTables* corpus [59]. The results of this experiment demonstrate that we outperform all existing semantic type detection models on numerical data.



(a) Graph Representation (b) Model architecture

Figure 11.1: (a) Shows the conversion of a table into a graph representation. The key aspect of the graph is that it provides all the necessary contextual information through its structure (nodes and directed edges), resulting in improved predictions of the semantic types of numerical columns. (b) Shows the complete model architecture of the neuronal network.

11.2 Overview of *Pythagoras*

In the following, we will introduce our new semantic type detection model *Pythagoras* and discuss the main design aspects that will lead to better predictions on numerical table columns.

Figure 11.1a demonstrates how we convert a table and its columns into a graph representation using an example. We can see that the table is transformed into a graph containing four different node types. The green node represents the table name. The orange and blue nodes are responsible for the representation of the textual and numerical columns. In addition, there is another node in the graph for each numerical column, which contains 192 selected statistical features (see Table 11.2 in the Appendix) of the numerical column values (red node). Because detecting semantic types of numerical columns is generally harder than for textual columns, using only the numerical column values to specify the type is too limited [61]. Hence, we designed the graph structure with directed edges to inject necessary context information into the numerical column

representation and thus enrich it for better predictions. Looking at a *Numerical Column Node* we can see that three directed edge types go towards the node. With that, the node will embed information from its connected neighbors into its own representation during a GNN layer iteration based on the message passing paradigm [54]. Specifically, the green edge provides information about the table name, the yellow edges convey information from each textual column within the table and the red edge facilitates the transmission of the additional statistical features. As a consequence, the GNN layers transforms the representation of the *Numerical Column Nodes*, leading to enhanced information content for accurate semantic type prediction. For instance, when faced with a numerical column with values in the range of 60-100, where the semantic type could be ambiguous (e.g., *basketball.player.weight* or *humidity*), the embedding of information from a neighboring textual column containing basketball player names allows for a more precise identification of the semantic type as *basketball.player.weight*.

In Figure 11.1b we can see the whole model architecture of *Pythagoras* which encodes the table structure as a graph. The upper part of the architecture illustrates how we generate the initial embedding vector representations of the nodes in the graph. For encoding table names as well as cell values (textual and numerical), we use the pre-trained transformer-based language model BERT[1] [22]. In addition, to embed the features of the *Numerical Values Statistic Nodes*, we train a feature specific subnetwork similar to the approach in [50]. This subnetwork embeds the extracted features from the numerical column to an output of fixed length using one hidden layer with a rectifier linear unit (ReLu) activation function. For each numerical column, we extract 192 features[2], including for example mean and median of all values, as well as statistical metrics regarding the occurrence of individual digits. The initial node representations, along with the discussed graph structure, serve as the input for the GNN model. As GNN, we use a graph convolutional neural network [54]. After traversing the GNN layers, we extract the hidden states of *Textual* as well as *Numerical Column Nodes* from the last convolutional layer. These hidden states are then passed as inputs to a final classification layer to perform the semantic type classification task.

[1] Note that *Pythagoras* is independent of how to generate these initial embeddings, and there may exist alternative language models or embedding methods that could potentially yield even better results in this context.

[2] The complete feature list can be found in Table 11.2 in the Appendix

Table 11.1: Experimental results of our new semantic type detection model Pythagoras in comparison to several state-of-the-art models on *SportsTables* corpus.

Model	support weighted F1-Score			macro F1-Score		
	numeric	non-numeric	overall	numeric	non-numeric	overall
Sherlock[50]	0.609	0.856	0.641	0.555	0.767	0.57
Sato[113]	0.703	0.961	0.736	0.650	0.903	0.668
Dosolo[100]	0.313	0.822	0.379	0.245	0.782	0.285
Doduo[100]	0.623	0.98	0.67	0.567	0.933	0.594
GPT-3 (fine-tuned)[10]	0.446	0.872	0.501	0.404	0.760	0.423
Pythagoras	**0.829**	**0.996**	**0.851**	**0.790**	**0.97**	**0.803**

11.3 Initial Experimental Results

In this section, we present initial experimental results applying *Pythagoras* on the *SportsTables* corpus. We compare the performance of our approach against five state-of-the-art models.

Baseline models. As state-of-the-art models we consider *Sherlock* [50], *Sato* [113], *Dosolo* [100], *Doduo* [100] and *GPT-3* [10, 84]. While *Sherlock* and *Dosolo* are models that utilize only the values of a single column for the prediction, *Sato* and *Doduo* are successors of them that adopt a context-based approach similar to our model. Despite their similarities to our model, Sato and Doduo do not specifically address the prediction of semantic types for numerical-based columns and do not offer a well-defined approach for injecting contextual information into the prediction process. Furthermore, to have another benchmark, we developed in our experiments a fine-tuned *GPT-3* model for the task of semantic type detection. We chose fine-tuning over prompt designs for higher model quality and the capacity to train on a larger number of examples[3].

Experiment setup. In our experiments, we use the *SportsTables* dataset, due to its high proportion of numerical-based columns. To perform the experiments, we split the corpus into 60/20/20 for train, validation, and test set. After training the models, the checkpoint with the best accuracy on the validation set is used for evaluation on the test set. We report end results as an average of five runs with different random seeds using the evaluation metrics support-weighted and macro F1-Score as in previous studies [20, 50, 100, 113]. To implement *Pythagoras* we used Python together with PyTorch [85],

[3]For fine-tuning the *GPT-3* model, we use OpenAIs API described in https://platform.openai.com/docs/guides/fine-tuning (visited on 09/04/2023)

DGL [106] and the Transformers library [107].

Results of study. The experimental results are shown in Table 11.1. For each model, we list the F1-Scores overall data types to show the total performance, but also the separate average F1-Scores for only numerical and non-numerical data types, respectively. The results show that our model *Pythagoras* outperforms all existing models in detecting the semantic type of numerical columns. To the best performing existing model *Sato* with F1-Scores of 0.703/0.650 (support-weighted/macro F1-Score) we can achieve an improvement of +0.126/+0.140. Furthermore, a notable observation across all existing models is the substantial performance discrepancy between the prediction on non-numerical and numerical columns. On non-numerical data, the accuracy of the models is generally high, whereas their performance on numerical data tends to be poorer. In contrast, our model exhibits a different behavior, as we are able to achieve a more balanced accuracy for both numerical and non-numerical data. In summary, the results demonstrate that our model, in conjunction with the graph representation of tables, leads to a significantly improved performance.

The road ahead. To establish the generalizability of our approach, additional experiments on diverse datasets are crucial. These experiments will validate the effectiveness of our approach across different data domains and assess its robustness. Furthermore, conducting an ablation study is essential to examine the impact of various design choices in our architecture.

11.4 Acknowledgements

This research and development project was funded by DHBW Mosbach. We also want to thank the NHR Program, the BMBF project KompAKI (grant number 02L19C150), the HMWK cluster project 3AI, hessian.AI, and DFKI Darmstadt for their support.

11.5 Appendix

11.5.1 List of Features

Table 11.2: Extracted features of a numerical column to build the representation of the *Numerical Values Statistic Node*. Except for the last feature listed, all features are also included in the model of [50].

Feature	#
Character-level distribution (any, all, mean, variance, min, max, median, sum, kurtosis, skewness of the digits 0-9 and the symbols comma, point, plus, minus, blank)	150
Number of values	1
Column entropy	1
Fraction of values with unique content	1
Fraction of values with numerical characters	1
Fraction of values with alphabetical characters	1
Mean and std. of the number of numerical characters in cell-values	2
Mean and std. of the number of alphabetical characters in cell-values	2
Mean and std. of the number special characters in cell-values	2
Mean and std. of the number of words in values	2
Percentage, count, only/has-Boolean of the None values	4
Stats, sum, min, max, median, mode, kurtosis, skewness, any/all-Boolean of length of values	10
Column values statistics (min, max, mean, median, 8*quantile, mode, skewness, kurtosis of all column values)	15
Total	**192**

12 Pythagoras: Semantic Type Detection of Numerical Data in Enterprise Data Lakes

Abstract

Detecting semantic types of table columns is a crucial task to enable dataset discovery in data lakes. However, prior semantic type detection approaches have primarily focused on non-numerical data despite the fact that numerical data play an essential role in many real-world enterprise data lakes. Therefore, existing models are typically rather inadequate when applied to data lakes that contain a high proportion of numerical data. In this paper, we introduce *Pythagoras*, our new learned semantic type detection approach specially designed to support numerical along with non-numerical data. *Pythagoras* uses a GNN in combination with a novel graph representation of tables to predict the semantic types for numerical data with high accuracy. In our experiments, we compare *Pythagoras* against five state-of-the-art approaches using two different datasets and show that our model significantly outperforms these baselines on numerical data. In comparison to the best existing approach, we achieve F1-Score increases of around +22%, which sets new benchmarks.

Bibliographic Information

The content of this chapter was previously published in the peer-reviewed work Sven Langenecker, Christoph Sturm, Christian Schalles, and Carsten Binnig. "Pythagoras: Semantic Type Detection of Numerical Data in Enterprise Data Lakes." In: *Proceedings 27th International Conference on Extending Database Technology, EDBT 2024, Paestum, Italy, March 25 - March 28*. Ed. by Letizia Tanca, Qiong Luo, Giuseppe Polese, Loredana Caruccio, Xavier Oriol, and Donatella Firmani. Vol. 27. OpenProceedings.org, 2024, pp. 725–733. DOI: 10.48786/EDBT.2024.62. URL: https://doi.org/10.48786/edbt.2024.62. The contributions of the author of this dissertation are summarized in Chapter 5.

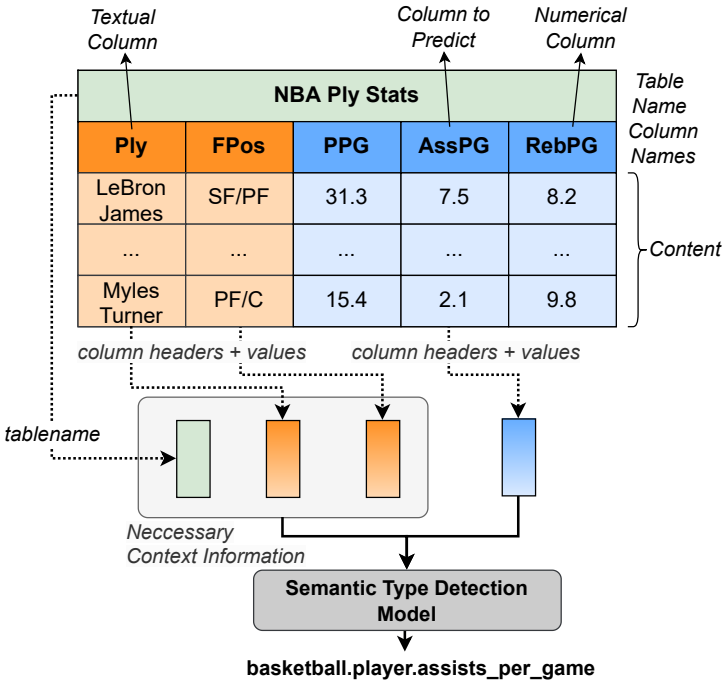


Figure 12.1: Figure shows an example of predicting the semantic type of the numerical table column 'AssPG'. To predict the correct type, it is crucial for the model to have the possibility to incorporate textual context information such as the table name and neighboring non-numerical columns.

12.1 Introduction

Enterprise data lakes contain numerical data.. Enterprise data lakes serve as invaluable repositories of diverse data types, enabling organizations to store and manage vast amounts of information [24, 72]. In enterprise data lakes, numerical data plays a dominant role, making up a much larger proportion compared to non-numerical data [59] since they provide *insights into various business domains*, including finance, manufacturing, healthcare, and marketing. Numerical data often contain critical information such as sales figures, production metrics, customer demographics, and financial records. Therefore, it is essential to provide a solution that can automatically detect the correct semantic type of table columns containing numerical values, enabling data analysts and data scientists to find required data for downstream analysis and thus address the dataset discovery problem in data lakes [14, 16, 29, 53, 78].

Semantic typing targets non-numerical data. In order to enable the discovery of data in data lakes and in particular to provide a solution for the associated task

of semantic type detection, many solutions using deep learning techniques have been proposed in the past [20, 50, 61, 100, 113]. Unfortunately, all these existing approaches have primarily focused on detecting the semantic type of non-numerical columns, and with that leaving a critical need for innovative approaches that effectively detect semantic types of numerical columns [59]. The reasons why existing models fall short on numerical data is mainly because of the fact that corpora that were used to train and validate these models primarily contain non-numerical data. Therefore, the models were developed to handle mainly non-numerical data.

Numerical data is much harder. To predict the semantic type of table columns containing numerical values, it is essential to have textual (non-numerical) data of the same table as context information as illustrated in Figure 12.1. Predicting, for example, the semantic type of the column 'AssPG' by using only the included values {7.5,...,2.1} is almost impossible while values of columns with textual types such as 'Ply' are more indicative for the type. The reason for this is that numerical values have in general a limited information entropy[1] and are often similarly distributed for different semantic content [61].

Context is essential for numerical data. To address this problem, rich non-numerical contextual information, such as the contents of neighboring non-numerical columns, column headers, and table names can be leveraged to increase accuracy in determining the correct semantic type of the numerical column. In the example of Figure 12.1 the table name 'NBA Ply Stats' and the values of the textual columns ('Ply' and 'FPos') can be leveraged as context information. This enables the model to recognize that the table belongs to the basketball domain, thereby enhancing its ability to predict the semantic type of the 'AssPG' column as 'basketball.player.assists_per_game'. As such, for a semantic type detection approach designed to handle numerical data, it is crucial to incorporate the capability to leverage all contextual information within the model architecture. Unfortunately, existing model architectures do not have such a predefined technique where non-numerical contextual information can be strategically leveraged for predicting numerical columns.

Semantic type detection for numerical data. In this paper, we thus introduce our approach based on a novel model architecture for semantic typing called *Pythagoras*, which can not only predict the semantic type of non-numerical table columns with high accuracy but also of numerical table columns. To achieve this, the main idea of the new model architecture is to use graph neural networks (GNNs) together with a new

[1] Generally numerical values can be encoded with much less bits than string values resulting in lower overall entropy values [98]

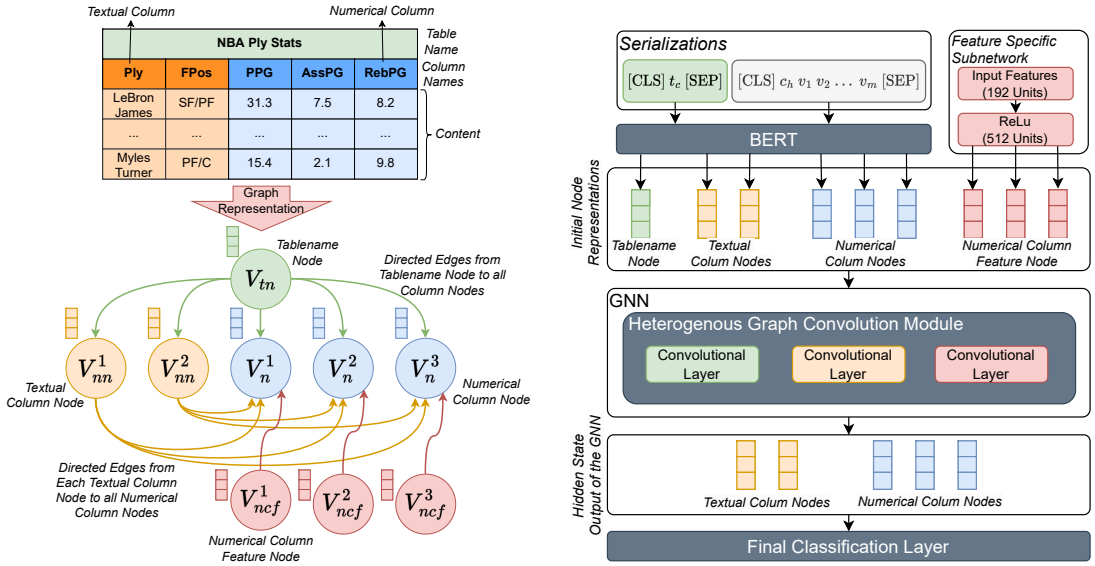
graph representation of tables and their columns. This graph representation includes directed edges to provide necessary contextual information (e.g. table name, neighboring non-numerical column values) for predicting the correct semantic type of numerical columns using the GNN message passing mechanism. Thus, the model learns which contextual information is relevant for determining the semantic type. To the best of our knowledge, our semantic type detection model *Pythagoras* is the first approach in this direction.

Contributions of the paper. The main contributions of this paper are: (1) First, we propose a new *graph representation of tables*. In this graph data structure, we use directed edges to model the information flow within tables. Using this graph representation, *Pythagoras* can learn selectively which context information should be taken into account to establish robust predictions on numerical data. (2) As a second contribution, we propose a new *GNN-based neural network architecture* that is able to use our new graph data structure as input and predict the semantic type of table columns. This architecture consists of three main components, (a) a pre-trained language model which encodes call values of tables, (b) a subnetwork to encode an additional specific feature set of the numerical values, (c) and a GNN with a heterogeneous graph convolutional module for aggregating all information and embedding context in the type prediction of columns. (3) Finally, as a third contribution, we show the effectiveness of the graph representation and the model architecture of *Pythagoras* by comparing against five existing state-of-the-art semantic type detection models on two different data lakes that mimic the data distribution of enterprise data lakes and contain tables with non-numerical and numerical columns. The results of this experiment demonstrate that our new model outperforms all baselines on numerical data significantly. To support the integration of our approach into existing applications and to enable further research, we open sourced all our code, data and trained model: `https://github.com/DHBWMosbachWI/pythagoras.git`.

Outline of the paper. In Section 12.2 we first introduce *Pythagoras* and our new graph representation of tables. Section 12.3 details the model architecture. Results and analyses of our experiments are presented in Section 12.4, followed by a discussion of related work in Section 12.5. Section 12.6 concludes the paper.

12.2 Overview of Pythagoras

In the following, we introduce our new semantic type detection approach *Pythagoras*.



(a) Graph Representation (b) Model Architecture

Figure 12.2: (a) Shows the conversion of a table into a heterogeneous graph representation. The key aspect of the graph is that it provides all the necessary contextual information through its structure (nodes and directed edges), resulting in improved predictions of the semantic types of numerical columns. (b) Shows the complete model architecture of the neural network.

12.2.1 Graph Representation of Tables

Figure 12.2a demonstrates how we convert a table and its columns into a graph representation using an example table in Figure 12.2a The table contains a table name (t_n), two non-numerical columns (c_{nn}), and three numerical columns (c_n), each with column headers (c_h) and column values ($v_1, v_2, ..., v_m$). In the figure, we can see how the table is transformed into a graph $G = \{V, E\}$ composed of a set of nodes V and a set of edges E including four different node types V_{tn}, V_{nn}, V_n, and V_{ncf} for different artifacts.

The first node type V_{tn} (green) represents the tablename. Additionally, the graph contains a node of type V_{nn} (orange) for each non-numerical column. This node type represents the entire column including column values and headers. In the same manner, for each numerical column, we create a node of type V_n (blue) representing numerical columns and their contents. Finally, nodes with a node type V_{ncf} (red) are added for each numerical column to encode specific features of the numerical columns.

We decided to use an additional node type V_{ncf} to encode specific features for numerical columns since this allows us to first use a pre-trained language model (LM) for computing

a representation based on the joint features that are shared between both non-numerical and numerical columns such as column headers . In addition, we further add the nodes V_{ncf} for the numerical-only columns, each holding a vector with additional specific features for numerical columns for which we use a separate encoding strategy with a separate simple multilayer perceptron (MLP) network. To be more precise, we additionally encode 192 different statistical features for encoding a numerical column. We publish the full list of features in an extended technical report of this paper[2].

12.2.2 Leveraging Contextual Information

As described before, only using the numerical values for predicting the semantic type of numerical columns is in general not sufficient, and contextual information is needed. Due to this aspect, we add directed edges to our table graph representation to predefine in which way necessary additional context information should be injected through the message-passing mechanism of GNNs [54] into the numerical column representation (node V_n) and thus enrich it for better predictions.

More precisely, as shown in Figure 12.2a we construct direct edges from each non-numerical column node V^1_{nn}, V^2_{nn} to all numerical column nodes V^1_n, V^2_n, V^3_n (yellow edges) to provide the context information from the non-numerical columns to the numerical columns. Furthermore, we add directed edges in the graph from the table name node V_{tn} to all non-numerical V_{nn} as well as numerical V_n nodes (green edges). This edge handles not only the contextual information for numerical columns but also for non-numerical columns. As we will show in our experiments, using the table name as context information also leads to performance improvements for non-numerical columns. Finally, the graph has directed edges for integrating the additional statistical features into the encoding of numerical columns (red edges from $V_{ncf} \rightarrow V_n$).

As a consequence when using this graph structure as basis for our GNN-based model architecture (cf Section 12.3), the vector representation computed for the numerical column nodes V_n during training result in an enhanced information content that is more suitable for an accurate prediction of the underline semantic type. Interestingly, leveraging context by modeling edges in a GNN not only improves the prediction of numerical but also non-numerical types as we show in our evaluation (cf. Section 12.4).

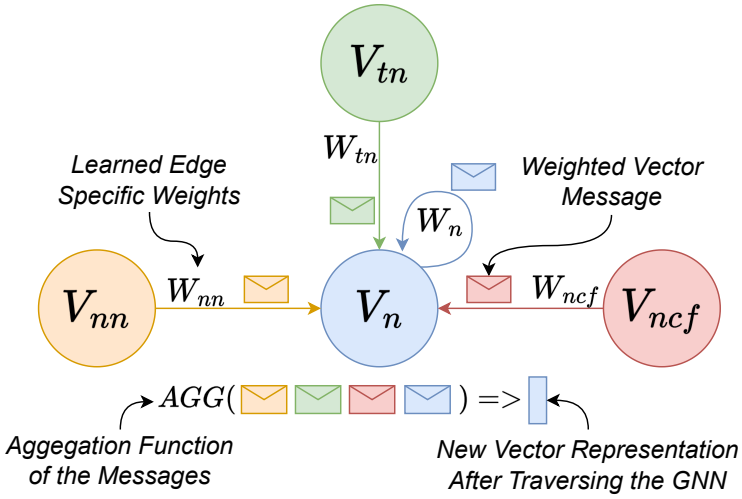


Figure 12.3: Heterogeneous graph convolutional module of *Pythagoras* for the nodes V_n. Information from the nodes V_{nn}, V_{tn} and V_{ncf} is passed to node V_n. Each edge connection $(W_{nn}, W_{tn}, W_{ncf}, W_n)$ has its own learned weights, which determine how strongly weighted the information is sent over the edges. Finally, all messages (vectors) are combined to form a new representation of the node by an aggregation function.

12.3 Model Architecture

In Figure 12.2b we can see the model architecture of *Pythagoras*. In the following, we explain first the details of the model architecture and then explain how the model can be used to detect numerical semantic types.

12.3.1 Architecture and Training

The model comprises three essential components. These components include (1) a pre-trained LM to encode all features from non-numerical and numerical columns [3], (2) a specific subnetwork to process the additional features of numerical columns and (3) the GNN to aggregate all information.

The upper part of the architecture in Figure 12.2b shows how we generate the input of the BERT model to get the initial representations for each column. Additionally, we use BERT to encode table names. To serialize the individual columns, we encode the input

[2]This technical report can be found at: https://github.com/DHBWMosbachWI/pythagoras.git
[3]We use BERT but *Pythagoras* is independent of how to generate the initial embeddings, and there may exist alternative language models or embedding methods that could potentially yield even better results in this context.

sequence for non-numerical as well as for numerical columns, using the column header and the column values as follows: $serialize(c_i)$ ::= [CLS] c_h v_1 v_2 ... v_m [SEP]. Additionally, to generate the initial representation of the node V_{tn} we thus serialize the table name as follows: $serialize(t_c)$ ::= [CLS] t_c [SEP]. For columns and table names, we use the representation computed by BERT for the CLS token as initial node representation for the GNN.

To embed the additional extracted features of the numerical column values, the model contains a feature-specific subnetwork similar to the approach in [50]. As can be seen in the architecture, the subnetwork consists of a linear layer that maps the 192 provided features to a vector that matches the shape of the other initial vector representations (BERT outputs vectors with dimensions of 768). This network is trained end to end with the GNN while the BERT parameters are frozen.

The initial vector representations generated by the BERT model and the subnetwork are used as initial internal representation for all nodes V_{tn}, V_{nn}, V_n, and V_{ncf} in our graph data structure which serves as input for our GNN model. As GNN, we use a heterogeneous graph convolutional module that combines different graph convolutional layers [54] for each occurring edge type. Since we have 3 different edge types in our graph, the heterogeneous convolutional module combines 3 independent graph convolutional layers. The heterogeneous convolution module first performs a separate graph convolution on each edge type, then sums the message aggregations on each edge type as the final result for the nodes. Figure 12.3 shows the behavior of this module for a numerical column node V_n that is connected with other nodes over the different edge types. The module works in a similar way also for non-numerical columns V_{nn} leveraging, however, only information from table name as shown in Figure 12.2a.

The module allows the model to learn separate weights for the different edge types and thus enables it to embed connected neighboring nodes and their information to different degrees. For example, the model can learn for V_n nodes that the information of the table name (provided by V_{tn}) is less important than the information of adjacent non-numerical columns (provided by V_{nn}). By learning distinct weights for each edge, we can effectively capture the nuances and dependencies in the data, ultimately enhancing the model's ability to make contextually informed predictions that lead to the overall effectiveness of our approach, which we will show more in detail in Section 12.4.5.

After traversing the GNN, we extract the hidden states of the nodes V_{nn} (representing updated non-numerical columns) as well as of V_n (representing updated numerical columns). Subsequently, these hidden states are then fed into a final classification layer to

perform the semantic type classification task. In this last classification layer, the output size is determined by the number of distinct semantic types present in the corpus.

12.3.2 Detecting Numerical Types

To highlight the advantage of using our graph representation of tables together with a GNN for semantic type detection of numerical data types, let us take a look at the following example. Considering the node V_n^1 in Figure 12.2a which stands for the numerical column 'PPG' (points per game statistic of a basket player) of the table. The column contains values in the range of about 15-32, and its semantic type could be ambiguous. The correct type might be *basketball.player.points_per_game*, *football.player.yards_per_game* or *temperature*).

However, after iterating over a GNN layer, the values of the two non-numerical columns V_{nn}^1, V_{nn}^2 are embedded because of the designed yellow edges. These provide basketball player names (Lebron James, ..., Myles Turner) as well as basketball field positions (SF/PF, ..., PF/C) as context information. According to this additional data, it is clear that the semantic type *temperature* is not very likely for this column. Because of the fact, that tables about player statistics in basketball as well as in football are structured very similarly and contain both columns with player's names and field positions, it is not yet clear whether the semantic type is *basketball.player.points_per_game* or *football.player.yards_per_game* for example.

Besides the previous context data of the non-numerical columns, information about the table name is also injected via the green edges during a GNN layer pass. This information contains the text 'NBA Ply Stats' ('NBA' is the name of the basketball league) and it is now unambiguous determinable that *basketball.player.points_per_game* must be the valid semantic type. The other passed information from the additional statistical feature nodes V_{ncf} also provides an improvement for distinguishing ambiguities, since the value range of numerical columns with different semantic types can be the same but the value distribution can be different. These different characteristics are covered by the extracted statistical features of numerical columns.

12.4 Experimental Evaluation

In the following, we first introduce the two datasets SportsTables and GitTables before we describe our experimental setup and evaluation methodology. Afterward, we discuss the main results of our experiments.

12.4.1 Data Sets and Baselines

Datasets. For evaluating *Pythagoras*, we use two different real-world data lakes with a large number of semantically annotated tables. When selecting the datasets, the goal was to choose a corpora that contain tables with a high proportion of numerical columns. This allows us in particular to explore and compare the existing models with *Pythagoras* on numerical data. As shown in Table 12.1, we use two corpora SportsTables [59] and GitTables Numeric which is based on [47]. Both corpora contain a high number of numerical columns per table and represents a numerical to non-numeric ratio commonly found in enterprise data lakes [61].

Table 12.1: Characteristics of the datasets in our experiments.

Dataset	#Tables	Non-Num. Cols./Table	Num. Cols./Table	#sem. Types
SportsTables	1,187	2.83	18.1	462
GitTables Numeric	6,577	2.08	8.95	219

SportsTables [59]. As the first data corpus in our experiments, we use SportsTables. The corpus contains real-world data tables collected from various sports domains such as soccer, basketball, baseball, and football using web scraping techniques. Such data tables are especially rich in numerical columns as many different sport-specific statistical measurements are reported. As can be seen in Table 12.1, the tables in the corpus contain 2.83 textual and 18.1 numerical columns on average. The corpus includes a very high number of 462 unique semantic types. Thereby semantic types are very fine granular, which is a major challenge for semantic type detection models. For example, there are types such as 'basketball.player.assists_per_game' or 'soccer.player.assists_per_game'.
GitTables Numeric [47]. The original GitTables data set is a corpus of tables created by extracting CSV files from GitHub repositories. Table columns are labeled with semantic types from Schema.org [35] and DBpedia [6] using two different automated annotation methods. In our experiments, we have focused on the annotations origin from DBpedia and the results of the semantic annotation method. For our experiments, we constructed a derived corpus called GitTables Numeric by specifically selecting tables that have a high proportion of numerical columns with the purpose to mimic real-world enterprise data lakes. To achieve this, we only included tables where at least 80% of all table columns are numerical. In order to have enough samples of each semantic type to train, validate, and test the models, we also filtered out columns that have a semantic type occurring less than 10 times in total. Based on these filter criteria, we ended up with a corpus that

Table 12.2: Experimental results on the SportsTables corpus.

Model	support weighted F1-Score			macro F1-Score		
	numerical	non-numerical	overall	numerical	non-numerical	overall
Sherlock [50]	0.609	0.856	0.641	0.555	0.767	0.57
Sato [113]	0.703	0.961	0.736	0.650	0.903	0.668
Dosolo [100]	0.313	0.822	0.379	0.245	0.782	0.285
Doduo [100]	0.623	0.98	0.67	0.567	0.933	0.594
GPT-3[4][10]	0.446	0.872	0.501	0.404	0.760	0.423
Pythagoras	**0.829**	**0.996**	**0.851**	**0.790**	**0.97**	**0.803**

contains 6,577 tables with 2.08 textual and 8.95 numerical columns per table on average (see Table 12.1) and a total of 219 semantic types.

Baselines. In our evaluation, we compare our model *Pythagoras* against five state-of-the-art semantic type detection models. As baselines we considered Sherlock [50], Sato [113], Dosolo [100], and Doduo [100]. Despite that Sato and Doduo also incorporate context information to predict the semantic type of a column, they do not specifically address numerical-based columns and do not offer a predefined approach for injecting contextual information into the prediction of numerical columns. All models were trained on the same data as *Pythagoras*.

Given the recent advancements in large language models (LLMs) like GPT-3.5 [10, 84], which have been extensively trained on vast amounts of data, one might wonder if such models cannot predict the semantic type for non-numerical as well as for numerical columns with high accuracy through a straightforward finetuning. Finetuning an LLM to a specific task has already shown success [52, 66, 86, 102]. In light of these considerations, we additionally explore the capabilities of recent LLMs in our study by adding a fine-tuned GPT-3.5 model. We opted for fine-tuning as opposed to prompt designs due to its potential to yield higher performances and to train on a larger number of examples. To build this baseline model we fine-tuned the *gpt-3.5-turbo* model, following the instructions in [2] using the same training data we used for *Pythagoras*.

12.4.2 Experimental Design

Setup. To run the experiments, we split each dataset into three parts: training, validation, and test set. We divided the datasets into 60% training, 20% validation, and 20% testing splits. Since in both datasets, the gold labels were assigned in an automatic manner by using the individual column headers, we did not include the headers in the

[4]fine-tuned

serializations of the columns, which is different from what is described in Section 12.2. When running the experiments, we trained each model using the training split and conducted hyperparameter tuning on the validation set.

In addition, we used the performance results on the validation split during training to apply an early stopping mechanism. To measure the final performance of each model, we loaded the checkpoint of the model with the highest F1-Score on the validation set and then applied it to the test data. We ran each experiment with five different random seeds and reported the mean across multiple runs to obtain statistically reliable results. As evaluation metrics, we used support-weighted F1-Score, weighted by the number of columns per semantic type and the macro average F1-Score as used in previous studies [20, 50, 100, 113].

***Pythagoras* implementation**. We implemented our model *Pythagoras* using Python together with the modules PyTorch [85], DGL [106] and the Transformers library [107]. As described in Section 12.2, our neural network consists of three main components. A pre-trained LM to generate initial vector representations, a subnetwork for the numerical-based feature set, and a GNN that allows to exchange context information.

As pre-trained LM, we used the vanilla BERT [22] (bert-base-uncased) model to be comparable to [100] which comes with 12 encoder layers. We used tokenizer and pre-trained model of the Transformers library from Hugging Face [28]. During the training process, we froze the 12 layers of BERT, preventing their weights from being updated.

The graph data structure and the GNN were implemented with the DGL library. To update the weights of the GNN during training, we applied an Adam optimizer with an initial learning rate of 10^{-5} and a linear decay scheduler with no warm-up. Since our purpose is to realize a multi-class prediction task (one semantic type label per column), we used the cross entropy loss as a loss function.

12.4.3 Exp. 1: Overall Efficiency

Results on SportsTables. Table 12.2 shows the experimental results on SportsTables. For each model, we list the F1-Scores overall data types to show the total performance, but also the separate average F1-Scores for only numerical and non-numerical data types, respectively. As the first main result, we can see in the table that our model *Pythagoras* outperforms all existing state-of-the-art models in all reported aspects. Looking only at the results on the numerical columns, we can see that our model achieves an improvement of +17.92% support weighted F1-Score and +21.53% macro F1-Score. These results verify

[5] fine-tuned

Table 12.3: Experimental results on the GitTables corpus.

Model	support weighted F1-Score			macro F1-Score		
	numerical	non-numerical	overall	numerical	non-numerical	overall
Sherlock [50]	0.725	0.989	0.775	0.411	0.491	0.707
Sato [113]	0.733	0.991	0.781	0.443	0.707	0.491
Dosolo [100]	0.518	0.986	0.606	0.245	0.694	0.343
Doduo [100]	0.761	0.992	0.804	0.409	0.749	0.489
GPT-3[5] [10]	0.531	0.938	0.610	0.143	0.277	0.211
Pythagoras	**0.813**	**0.990**	**0.846**	**0.476**	**0.893**	**0.544**

that our designed mechanism of providing context information to predict the semantic type of numerical data is more suitable than the methods in the existing models Sato and Doduo.

In Sato, contextual information is provided by a table topic vector, which is formed by a accumulation of all values in the table. Since tables in the SportsTables dataset contains a large proportion of columns with numerical values (on average 18.1 are numerical column and 2.83 are non-numerical, see Table 12.1), this table topic vector does not have the necessary effect. In addition, Satos linear-chain conditional random field (CRF) also does not lead to significant improvements, since the tables in SportsTables are not always structured in the same way (column orders vary between tables). This aspect can be seen by the comparison of Sato to Sherlock, which is the same model without a table topic vector and a linear-chain CRF module. The improvements from Sherlock to Sato are not significant.

Doduo also achieves only moderate performance values with 0.623/0.564 (support weighted/macro) F1-Score. On one hand, this is due to the fact that only very few individual column values can be included in the token sequence, since the BERT model is limited to 512 elements and the tables have on average 20.93 columns. On the other hand, the BERT model learns the structure of the tables which, as with Sato, has negative effects with non-identical cross-table structures. Furthermore, it is still unclear how deep the understanding of numbers is in LMs like BERT, since they are essentially pre-trained on textual data. Unlike the existing models, our model is independent of the column order of the tables due to the graph structure. If columns are arranged differently between tables, this has no negative effect.

When we examine the results on textual data, we generally observe that all models perform well. In particular, the models Sato, Doduo, as well as our model *Pythagoras* achieve high accuracy. Interestingly, also for non-numerical columns our model is slightly

better than existing models with 0.996/0.970 F1-Scores. This improvement is due to the design aspect that our model uses the contextual information of the table name also for the non-numerical column representations ($V_{nn} \rightarrow V_n$ edges). Moreover, our results demonstrate the aspect that on numerical data, the prediction of the semantic type is in general harder than the prediction of non-numerical data.

In summary, the results on the SportsTables dataset demonstrate that our model architecture, in conjunction with the graph representation of tables, leads to significantly improved performance in predicting semantic types for numerical-based columns.

Results on GitTables. Table 12.3 shows the experimental results on GitTables using the same metrics as before on SportsTables. The results show that *Pythagoras* outperforms all other models in predicting the semantic types. Considering the performance on numerical columns, it becomes evident that our model surpasses the performance of the best existing model, Doduo, by a remarkable improvement of +6.83%/16.38% F1-Score.

This gain in performance highlights the effectiveness of our model in handling numerical data, setting a new benchmark in this domain by outperforming all state-of-the-art approaches. Different from the results on the SportsTables corpus, among the baselines, Doduo and Sato perform nearly equally. This is mainly due to the aspect that the GitTables corpus contains tables with fewer columns on average, and therefore Doduo can use more column values in its token sequence and with that build a better representation using the BERT model.

Looking at the performance on non-numerical data columns, we can see that all models achieve mostly the same support weighted F1-Scores (about 0.990). However, considering the macro F1-Scores our model *Pythagoras* reaches by far the best value with 0.893. This is an improvement to the second-best model Doduo by +19.23%, showing again the benefit of providing the table name as contextual information for predicting the semantic type of non-numerical columns. In summary, the results on GitTables show that our model *Pythagoras* sets new state-of-the-art performances for predicting the semantic type of numerical table columns.

12.4.4 Exp. 2: Performance for Individual Types

Figure 12.4 shows a more detailed analysis of the performances between *Pythagoras* and Sato on numerical columns in SportsTables. We chose Sato as comparison model because it was the best baseline model on numerical columns in this dataset. On the left side, the pie chart shows for how many semantic types of numerical columns which model performed better regarding the F1-Score. Out of a total of 384 numerical semantic

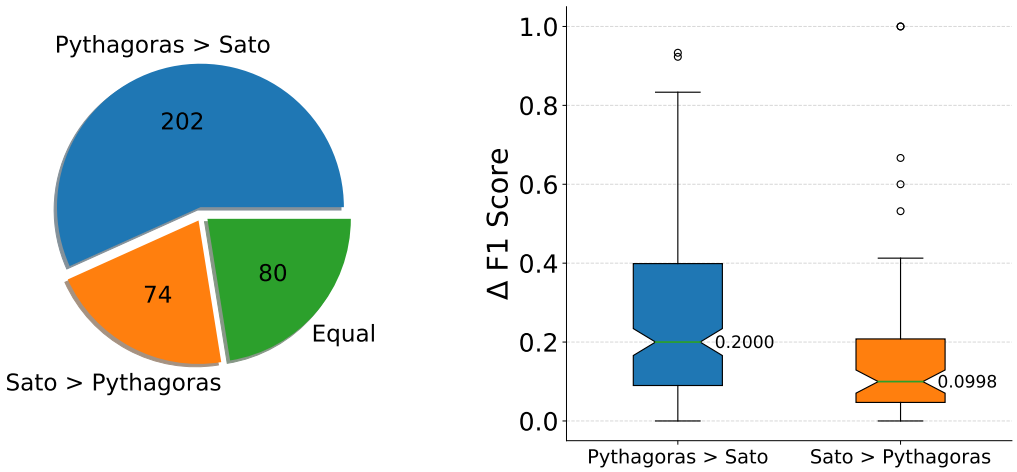


Figure 12.4: The left chart shows the number of numerical types for which *Pythagoras* performs better than Sato and vice versa. In the right chart we see box plots for the F1-Score differences between the two models on the different numerical semantic types where *Pythagoras* was better than Sato and vice versa.

types, *Pythagoras* was able to achieve substantially better performances than Sato on 202 of them. For 80 types, the two models achieve equal F1-Scores and for 74, Sato is better than *Pythagoras*. This demonstrates that our model is not only more accurate for individual numerical semantic types but also for a very large proportion of them.

To show how large the F1-Score differences between the two models across the numerical types are, boxplots of the differences for the cases *Pythagoras*>Sato and vice versa are shown on the right of Figure 12.4. In the case where our model achieves higher F1-Scores, we can see that the median value of the distances is 0.2. The 0.75 quantile is 0.4 and there are also a few types where our model is better than Sato by more than 0.9. In addition, the distribution is shifted upwards towards the larger distance values. In the case where Sato is better, the median is about 0.1 and the 0.75 quantil is 0.2. The distribution is also shifted upwards, but not as much as in the other case. In conclusion, these results show that there are many types for which *Pythagoras* performs much better than Sato and Sato can only achieve very low F1-Scores, and the differences to our model are significant. In the other case, for the majority of types in which Sato performs better, our model *Pythagoras* achieves only slightly lower scores.

Overall, this suggests that our model architecture and the method we designed for providing context information are better suited for detecting the semantic type of numerical data.

Table 12.4: Ablation study results on only numerical columns of the SportsTables dataset. We tested different graph structures that provide different types of contextual information (upper part). The lower part shows results when including the column header c_h as additional information in the serialization of a column.

Variant	support weighted avg F1-Score	macro avg F1-Score
Pythagoras	0.829	0.790
w/o V_{tn}	0.812	0.759
w/o V_{nn}	0.785	0.733
w/o V_{ncf}	0.813	0.765
w/o V_{tn}, V_{nn}	0.724	0.693
w/o V_{tn}, V_{nn}, V_{ncf}	0.324	0.252
w/ original c_h	0.991	0.950
w/ synthesized c_h	0.972	0.926

12.4.5 Exp. 3: Ablation Study

Different graph variants. To verify the different design aspects of our approach, we tested variants of *Pythagoras*. At first, we tested modifications of our graph representation of tables. In particular, we wanted to investigate which contextual information has which effect on the prediction of the semantic type. Table 12.4 shows the results of this ablation study by displaying support weighted and macro average F1-Scores on numerical columns. The first row reports the results of using our regular model and graph while the next rows presents the results when various nodes and edges are removed in the graph representation. Here w/o V_{tn} means that in the graph the node representing the table name has been removed and thus also the provision of this context information for the prediction of the semantic type of the columns. Note, that the other nodes V_{nn} and V_{ncf} are still present in the graph and still provide contextual information to the numerical columns representations. Equally, w/o V_{nn} means that the edges of non-numerical to numerical columns have been removed and thus the flow of information from the non-numerical columns no longer occurs during a GNN layer pass. However, in this variant, the other nodes are present.

The first finding that can be seen in the results is that when we remove the nodes V_{nn}, we see the highest performance drop. The F1-Score decreases in this case -0,044/-0,057 in comparison to the regular model. Thus, we can conclude that the most important contextual information for a correct prediction of the semantic type of numerical-based columns are the values of the non-numerical columns from the same table. The second

most important context is the table name (V_{tn}) and the least important are the statistical features of the numerical values in the columns (V_{ncf}). Without the table name as context, the model performance decreases a bit more than without the statistical features. To see how good the performance is when making a semantic type prediction only using the numerical values of the columns (V_n and V_{ncf}), we have also considered a variant in which V_{tn} (table name) and V_{nn} (non-numerical columns) nodes are not present. With this variant, the F1-Score drops very sharply and the model only achieves values of 0.724/0.693. This result again shows the immense importance of textual context information in predicting the semantic type of numerical data. In addition, we have tested a variant in which only the V_n nodes are present (w/o V_{tn}, V_{nn}, V_{ncf}). As expected, we just get similar performances to the Dosolo model, since in this constellation both model structures are very similar.

Different column serializations. As mentioned before, in the experiments of Section 12.4.2, we did not include the original column headers c_h in the serialization of a column because they were previously used to semi automatically assign the true semantic types (gold labels) to the columns. However, to show the impact column headers can have on the performance of numerical column predictions, we created synthetic column headers and used them in an experiment. We created the synthesized column headers using GPT by giving us a list of 10 possible abbreviations for the respective column headers. For example, for the header ”Player Age” GPT provided the list [”PA”, ”PlAge”, ”PAG”, ”PLAG”, ”PlrAge”, ”PlyAg”, ”PLA”, ”PrAge”, ”PlyrA”, ”PlayA”]. Afterward, for each column, we randomly selected an abbreviation from the list and used it as the column header. The lower part of Table 12.4 shows the results of this experiment. We can see that the inclusion of column headers has an additional positive effect on predicting the semantic types of numerical data, achieving F1-Scores of 0.972/0.926 (close to the performance when using the original highly indicative column headers).

12.5 Related Work

In the following, we will present an overview of existing approaches and discuss the main shortcomings when applied to numerical data.

Columnwise Models. Columnwise models exclusively leverage values from a single column, omitting the inclusion of contextual information from the table. Sherlock [50] is columnwise model which extracts multiple features, such as character distributions, word embeddings, text embeddings, and column statistics from individual columns.

These features are then processed through a combination of multi-layer subnetworks and a primary network, which comprises two fully connected layers. Dosolo [100] is a columnwise model that uses the pre-trained BERT model combined with an attached output layer to implement a semantic type detection model. Given that BERT receives token sequences (i.e. text) as input, they convert a column into such a sequence. When serializing the columns, the individual column values are first converted into a string and then concatenated to a sequence.

Tablewise Models. Tablewise models leverage the entire table as input. The advantages of this approach lie in its ability to utilize contextual information from the table, enhancing the precision of semantic type prediction for individual table columns. Building upon Sherlock, Sato is a tablewise model that incorporates Latent Dirichlet Allocation (LDA) features to capture table context and integrates a CRF layer to learn column type dependency. With this, Sato's prediction quality improves over Sherlock. Dosolo & Doduo are both models from [100]. In contrast to Dosolo, Doduo is a tablewise model designed to process an entire table as input to the BERT model. To do this all columns and their values are concatenated one after the other to form an input sequence. The major difference between the two approaches and their serialization techniques is that with Dosolo a column type is predicted independently of other data, whereas Doduo captures the data of neighboring columns to make a prediction of a column semantic type.

Discussion on Existing Approaches. Recent research papers [20, 100, 113] have shown that columnwise models are limited since they can not use context information when predicting the semantic types of columns. As the need for contextual information is even more important for numerical columns, columnwise models are unsuitable for its detection.

As such, Sato [113] and Doduo [100] incorporate also context information like the table-topic or values from neighboring columns as described above. However, these models are mainly designed to handle non-numerical data since used corpora contain almost entirely textual data. If tables contain many numerical and only a few textual columns the context information provided is reduced and the methodologies implemented in Sato and Doduo will not provide the necessary context information to work on numerical data. For example, the table-topic vector of Sato provides no benefit, since numerical values are dominant in the table. With Doduo, the serialization of the table also contains essentially numerical data and this leads to the same effect. However, we show that a controlled and predefined embedding of context information, as we implemented it in *Pythagoras*, leads to a significant improvement on numerical columns. Another shortcoming of Doduo is the

limited amount (512 Elements) of column values that can be included when serializing the table. With this, increasing the number of table columns means a decreasing the number of values of each column in the input.

For wide tables, this results in an insufficient column representation and also to only a few values that serve as context information. This is a problem for the prediction on numerical columns, where context information is needed. In our experiments, we demonstrated that our new model *Pythagoras* can handle wide tables containing many numerical columns in a better way. Furthermore, Sato and Doduo have the disadvantage that they essentially learn the order of the columns in the table. This creates a major dependency that tables must have the same column arrangement to ensure the model works adequately. In our opinion, table structures are not always the same, especially in data lakes. For example, Sato's pairwise potentials are learned only for adjacent columns. Whenever there is a different order of the columns, which causes direct neighbors are changed, the learned potentials are no longer useful. Doduo is also sensitive to the column order in a table because the underline BERT model is sensitive to the order in the input sequence. However, our model *Pythagoras* is completely independent of the column order due to the used graph representation of tables together with the GNN architecture.

12.6 Conclusion

The task of semantic type detection of table columns stored in data lakes is crucial to address the dataset discovery problem. Due to the fact that a large proportion of data in enterprise data lakes are numerical [61] and often contains critical information, it is even more important to provide a solution that can detect the underline semantics for these data types robustly. While recent papers propose approaches for extracting semantic types, unfortunately, they have been designed primarily on non-numerical data and therefore do not provide accurate performances when used on numerical data columns. To tackle this problem, we suggested in this paper our new semantic type detection approach *Pythagoras*, specifically designed to robustly handle numerical columns. Our graph representation of tables and GNN architecture establish an intrinsic mechanism that provides all necessary context information to determine the correct semantic type of numerical columns. Experimental results comparing *Pythagoras* against five state-of-the-art models on two different datasets containing mainly numerical table columns show that our approach sets new benchmarks for predicting the semantic type of numerical data.

12.7 Acknowledgements

This work was partially funded by the German Federal Ministry of Education and Research (BMBF) within the “The Future of Value Creation – Research on Production, Services and Work” program (funding number 02L19C150) and by the state of Hesse as part of the NHR program. We also thank hessian.AI, 3AI, DFKI, and DHBW for their support.

Bibliography

[1] Nora Abdelmageed, Sirko Schindler, and Birgitta König-Ries. *fusion-jena/BiodivTab*. Version v0.1_2021. Zenodo, Oct. 2021. DOI: 10.5281/zenodo.5584180. URL: https://doi.org/10.5281/zenodo.5584180.

[2] Open AI. *Fine-tuning*. Open AI. Mar. 2023. URL: https://platform.openai.com/docs/guides/fine-tuning.

[3] *Alation Data Catalog*. https://www.alation.com/. Accessed: 2022-10-15. 2022.

[4] *AWS Glue Data Catalog*. https://docs.aws.amazon.com/glue/latest/dg/what-is-glue.html. Accessed: 2022-10-15. 2022.

[5] *Atlan Data Discovery Catalog*. https://atlan.com/data-discovery-catalog/?ref=/data-discovery-catalog/. Accessed: 2024-01-02. 2024.

[6] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. "DBpedia: A Nucleus for a Web of Open Data." In: *The Semantic Web*. Ed. by Karl Aberer, Key-Sun Choi, Natasha Noy, Dean Allemang, Kyung-Il Lee, Lyndon Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 722–735. ISBN: 978-3-540-76298-0.

[7] Amazon AWS. *AWS Glue Concepts - AWS Glue*. https://docs.aws.amazon.com/glue/latest/dg/components-key-concepts.html. 2020. (Visited on 03/24/2020).

[8] Mircosoft Azure. *Common Data Model and Azure Data Lake Storage Gen2 - Common Data Model | Microsoft Docs*. 2020. URL: https://docs.microsoft.com/en-us/common-data-model/data-lake (visited on 03/23/2020).

[9] Chandra Sekhar Bhagavatula, Thanapon Noraset, and Doug Downey. *TabEL: Entity Linking in Web Tables*. Cham, 2015.

[10] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. "Language Models Are Few-Shot Learners." In: *Proceedings of the 34th International Conference on Neural Information Processing Systems*. NIPS'20. Vancouver, BC, Canada: Curran Associates Inc., 2020. ISBN: 9781713829546.

[11] Michael J. Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. *WebTables: Exploring the Power of Tables on the Web*. 2008. DOI: 10.14778/1453856.1453916. URL: https://doi.org/10.14778/1453856.1453916.

[12] R. Castro Fernandez, Z. Abedjan, F. Koko, G. Yuan, S. Madden, and M. Stonebraker. "Aurum: A Data Discovery System." In: *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. 2018, pp. 1001–1012. DOI: 10.1109/ICDE.2018.00094.

[13] R. Castro Fernandez, E. Mansour, A. A. Qahtan, A. Elmagarmid, I. Ilyas, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang. "Seeping Semantics: Linking Datasets Using Word Embeddings for Data Discovery." In: *ICDE '18*. 2018, pp. 989–1000.

[14] Adriane Chapman, Elena Simperl, Laura Koesten, George Konstantinidis, Luis-Daniel Ibáñez, Emilia Kacprzak, and Paul Groth. "Dataset Search: A Survey." In: *The VLDB Journal* 29.1 (Aug. 2019), pp. 251–272. ISSN: 1066-8888. DOI: 10.1007/s00778-019-00564-x. URL: https://doi.org/10.1007/s00778-019-00564-x.

[15] Jiaoyan Chen, Ernesto Jiménez-Ruiz, Ian Horrocks, and Charles Sutton. "ColNet: Embedding the Semantics of Web Tables for Column Type Prediction." In: *AAAI'19*. AAAI'19/IAAI'19/EAAI'19. Honolulu, Hawaii, USA: AAAI Press, 2019, pp. 29–36. ISBN: 978-1-57735-809-1. DOI: 10.1609/aaai.v33i01.330129. URL: https://doi.org/10.1609/aaai.v33i01.330129.

[16] Christina Christodoulakis, Eric B. Munson, Moshe Gabel, Angela Demke Brown, and Renée J. Miller. "Pytheas: Pattern-Based Table Discovery in CSV Files." In:

Proc. VLDB Endow. 13.12 (July 2020), pp. 2075–2089. ISSN: 2150-8097. DOI: 10.14778/3407790.3407810. URL: https://doi.org/10.14778/3407790.3407810.

[17] *Collibra Data Catalog*. https://www.collibra.com/us/en/products/data-catalog. Accessed: 2022-10-15. 2022.

[18] Vincenzo Cutrona, Federico Bianchi, Ernesto Jiménez-Ruiz, and Matteo Palmonari. *Tough Tables: Carefully Evaluating Entity Linking for Tabular Data*. Version 1.0. Zenodo, Nov. 2020. DOI: 10.5281/zenodo.4246370. URL: https://doi.org/10.5281/zenodo.4246370.

[19] Vincenzo Cutrona, Federico Bianchi, Ernesto Jiménez-Ruiz, and Matteo Palmonari. *Tough Tables: Carefully Evaluating Entity Linking for Tabular Data*. Version 1.0. Zenodo, Nov. 2020. DOI: 10.5281/zenodo.4246370. URL: https://doi.org/10.5281/zenodo.4246370.

[20] Xiang Deng, Huan Sun, Alyssa Lees, You Wu, and Cong Yu. "TURL: Table Understanding through Representation Learning." In: *VLDB*. Vol. 14. 3. VLDB Endowment, 2021, pp. 307–319. DOI: 10.14778/3430915.3430921. eprint: 2006.14806v2. URL: https://github.com/sunlab-osu/TURL.

[21] Benjamin Denham, Edmund M-K. Lai, Roopak Sinha, and M. Asif Naeem. "Witan: Unsupervised Labelling Function Generation for Assisted Data Programming." In: *VLDB*. Vol. 15. VLDB Endowment, July 2022, pp. 2334–2347. DOI: 10.14778/3551793.3551797. URL: https://doi.org/10.14778/3551793.3551797.

[22] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: https://aclanthology.org/N19-1423.

[23] Rabiyatou Diouf, Edouard Ngor Sarr, Ousmane Sall, Babiga Birregah, Mamadou Bousso, and Sény Ndiaye Mbaye. *Web Scraping: State-of-the-Art and Areas of Application*. 2019. DOI: 10.1109/BigData47090.2019.9005594.

[24] James Dixon. *Data Lakes Revisited*. https://jamesdixon.wordpress.com/2014/09/25/data-lakes-revisited/. 2014. (Visited on 02/17/2020).

[25] *Dremio*. https://www.dremio.com/. Accessed: 2022-10-15. 2022.

[26] Dremio. *State of the Data Lakehouse*. 2024. URL: https://hello.dremio.com/wp-state-of-the-data-lakehouse-reg.html (visited on 04/22/2024).

[27] Sara Evensen, Chang Ge, Dongjin Choi, and Çagatay Demiralp. "Data Programming by Demonstration: A Framework for Interactively Learning Labeling Functions." In: *CoRR* abs/2009.01444 (2020). arXiv: 2009.01444. URL: https://arxiv.org/abs/2009.01444.

[28] Hugging Face. *bert-base-uncased*. Hugging Face. Mar. 2023. URL: https://huggingface.co/bert-base-uncased.

[29] Grace Fan, Jin Wang, Yuliang Li, and Renée J. Miller. "Table Discovery in Data Lakes: State-of-the-Art and Future Directions." In: *Companion of the 2023 International Conference on Management of Data*. SIGMOD '23. Seattle, WA, USA: Association for Computing Machinery, 2023, pp. 69–75. ISBN: 9781450395076. DOI: 10.1145/3555041.3589409. URL: https://doi.org/10.1145/3555041.3589409.

[30] Huang Fang. "Managing data lakes in big data era: What's a data lake and why has it became popular in data management ecosystem." In: *2015 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)*. 2015, pp. 820–824. DOI: 10.1109/CYBER.2015.7288049.

[31] Bogdan Ghita. *Public BI benchmark*. https://github.com/cwida/public_bi_benchmark/tree/master. 2019.

[32] Aleph Alpha GmbH. *Luminous*. 2023. URL: https://app.aleph-alpha.com/ (visited on 12/19/2023).

[33] *Google Cloud Data Catalog*. https://cloud.google.com/data-catalog/docs/concepts/overview. Accessed: 2022-10-15. 2022.

[34] *Freebase Data Dumps*. https://developers.google.com/freebase. Accessed: 2022-10-15. 2022.

[35] R. V. Guha, Dan Brickley, and Steve Macbeth. "Schema.Org: Evolution of Structured Data on the Web." In: *Commun. ACM* 59.2 (Jan. 2016), pp. 44–51. ISSN: 0001-0782. DOI: 10.1145/2844544. URL: https://doi.org/10.1145/2844544.

[36] Rihan Hai, Christos Koutras, Christoph Quix, and Matthias Jarke. "Data Lakes: A Survey of Functions and Systems." In: *IEEE Transactions on Knowledge and Data Engineering* 35.12 (2023), pp. 12571–12590. DOI: 10.1109/TKDE.2023.3270101.

[37] A. Halevy, Flip Korn, Natasha Noy, Christopher Olston, Neoklis Polyzotis, S. Roy, and Steven Euijong Whang. "Managing Google's data lake: an overview of the Goods system." In: *IEEE Data Eng. Bull.* 39 (2016), pp. 5–14.

[38] Alon Halevy, Flip Korn, Natalya F. Noy, Christopher Olston, Neoklis Polyzotis, Sudip Roy, and Steven Euijong Whang. "Goods: Organizing Google's Datasets." In: *SIGMOD '16*. ACM, 2016, pp. 795–806. ISBN: 9781450335317. DOI: 10.1145/2882903.2903730. URL: https://doi.org/10.1145/2882903.2903730.

[39] Shijie Hao, Yuan Zhou, and Yanrong Guo. "A Brief Survey on Semantic Segmentation with Deep Learning." In: *Neurocomputing* 406 (2020), pp. 302–321. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2019.11.118. URL: https://www.sciencedirect.com/science/article/pii/S0925231220305476.

[40] Oktie Hassanzadeh, Vasilis Efthymiou, Jiaoyan Chen, Ernesto Jiménez-Ruiz, and Kavitha Srinivas. *SemTab 2019: Semantic Web Challenge on Tabular Data to Knowledge Graph Matching Data Sets*. Version 2019. Zenodo, Oct. 2019. DOI: 10.5281/zenodo.3518539. URL: https://doi.org/10.5281/zenodo.3518539.

[41] Oktie Hassanzadeh, Vasilis Efthymiou, Jiaoyan Chen, Ernesto Jiménez-Ruiz, and Kavitha Srinivas. *SemTab 2020: Semantic Web Challenge on Tabular Data to Knowledge Graph Matching Data Sets*. Version 2020. Zenodo, Nov. 2020. DOI: 10.5281/zenodo.4282879. URL: https://doi.org/10.5281/zenodo.4282879.

[42] Oktie Hassanzadeh, Vasilis Efthymiou, Jiaoyan Chen, Ernesto Jiménez-Ruiz, and Kavitha Srinivas. *SemTab 2021: Semantic Web Challenge on Tabular Data to Knowledge Graph Matching Data Sets*. Version 2021 (Hard Tables). Zenodo, Nov. 2021. DOI: 10.5281/zenodo.6154708. URL: https://doi.org/10.5281/zenodo.6154708.

[43] Yeye He, Jie Song, Yue Wang, Surajit Chaudhuri, Vishal Anil, Blake Lassiter, Yaron Goland, and Gaurav Malhotra. "Auto-Tag: Tagging-Data-By-Example in Data Lakes." In: *CoRR* abs/2112.06049 (2021). arXiv: 2112.06049. URL: https://arxiv.org/abs/2112.06049.

[44] Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Eisenschlos. "TaPas: Weakly Supervised Table Parsing via Pre-training." In: *ACL 2020*. Online: ACL, July 2020, pp. 4320–4333. DOI: 10.18653/v1/2020.acl-main.398. URL: https://aclanthology.org/2020.acl-main.398.

[45] Kevin Hu, Snehalkumar 'Neil' S. Gaikwad, Madelon Hulsebos, Michiel A. Bakker, Emanuel Zgraggen, César Hidalgo, Tim Kraska, Guoliang Li, Arvind Satyanarayan, and Çağatay Demiralp. *VizNet: Towards A Large-Scale Visualization Learning and Benchmarking Repository*. Glasgow, Scotland Uk, 2019. DOI: 10.1145/3290605.3300892. URL: https://doi.org/10.1145/3290605.3300892.

[46] Madelon Hulsebos, Cagatay Demiralp, and Paul Demiralp. *GitTables benchmark - column type detection*. Zenodo, Nov. 2021. DOI: 10.5281/zenodo.5706316. URL: https://doi.org/10.5281/zenodo.5706316.

[47] Madelon Hulsebos, Çagatay Demiralp, and Paul Groth. "GitTables: A Large-Scale Corpus of Relational Tables." In: *CoRR* abs/2106.07258 (2021). arXiv: 2106.07258. URL: https://arxiv.org/abs/2106.07258.

[48] Madelon Hulsebos, Sneha Gathani, James Gale, Isil Dillig, Paul Groth, and Çağatay Demiralp. *Making Table Understanding Work in Practice*. 2021. arXiv: 2109.05173 [cs.DB].

[49] Madelon Hulsebos, Paul Groth, and Çağatay Demiralp. "AdaTyper: Adaptive Semantic Column Type Detection." In: (2023). arXiv: 2311.13806. URL: http://arxiv.org/abs/2311.13806.

[50] Madelon Hulsebos, Kevin Hu, Michiel Bakker, Emanuel Zgraggen, Arvind Satyanarayan, Tim Kraska, Çagatay Demiralp, and César Hidalgo. "Sherlock: A Deep Learning Approach to Semantic Data Type Detection." In: *SIGKDD*. KDD '19. Anchorage, AK, USA: ACM, 2019, pp. 1500–1508. ISBN: 9781450362016. DOI: 10.1145/3292500.3330993. URL: https://doi.org/10.1145/3292500.3330993.

[51] GMI Global Market Insights. *Data Lake Market Report*. 2023. URL: https://www.gminsights.com/industry-analysis/data-lake-market (visited on 04/22/2024).

[52] Zhengbao Jiang, Frank F. Xu, Jun Araki, and Graham Neubig. "How Can We Know What Language Models Know?" In: *Transactions of the Association for Computational Linguistics* 8 (2020). Ed. by Mark Johnson, Brian Roark, and Ani Nenkova, pp. 423–438. DOI: 10.1162/tacl_a_00324. URL: https://aclanthology.org/2020.tacl-1.28.

[53] Aamod Khatiwada, Grace Fan, Roee Shraga, Zixuan Chen, Wolfgang Gatterbauer, Renée J. Miller, and Mirek Riedewald. "SANTOS: Relationship-Based Semantic Table Union Search." In: *Proc. ACM Manag. Data* 1.1 (May 2023). DOI: 10.1145/3588689. URL: https://doi.org/10.1145/3588689.

[54] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks." In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL: https://openreview.net/forum?id=SJU4ayYgl.

[55] Keti Korini and Christian Bizer. "Column type annotation using ChatGPT." Englisch. In: *Joint proceedings of workshops at the 49th International Conference on Very Large Data Bases (VLDB 2023), Vancouver, Canada, August 28 - September 1, 2023, VLDBW 2023*. Ed. by Rajesh Bordawekar, Cinzia Cappiello, and Vasilis Efthymiou. Vol. 3462. CEUR Workshop Proceedings. Aachen, Germany: RWTH Aachen, Sept. 2023, pp. 1–12. URL: https://madoc.bib.uni-mannheim.de/65132/.

[56] Christos Koutras, George Siachamis, Andra Ionescu, Kyriakos Psarakis, Jerry Brons, Marios Fragkoulis, Christoph Lofi, Angela Bonifati, and Asterios Katsifodimos. "Valentine: Evaluating Matching Techniques for Dataset Discovery." In: *ICDE*. IEEE, 2021, pp. 468–479.

[57] Sven Langenecker, Christoph Sturm, Christian Schalles, and Carsten Binnig. "Pythagoras: Semantic Type Detection of Numerical Data in Enterprise Data Lakes." In: *Proceedings 27th International Conference on Extending Database Technology, EDBT 2024, Paestum, Italy, March 25 - March 28*. Ed. by Letizia Tanca, Qiong Luo, Giuseppe Polese, Loredana Caruccio, Xavier Oriol, and Donatella Firmani. Vol. 27. OpenProceedings.org, 2024, pp. 725–733. DOI: 10.48786/EDBT.2024.62. URL: https://doi.org/10.48786/edbt.2024.62.

[58] Sven Langenecker, Christoph Sturm, Christian Schalles, and Carsten Binnig. "Pythagoras: Semantic Type Detection of Numerical Data Using Graph Neural Networks (Short Paper)." In: *Lernen, Wissen, Daten, Analysen (LWDA) Conference Proceedings, Marburg, Germany, October 9-11, 2023*. Ed. by Michael Leyer and Johannes Wichmann. Vol. 3630. CEUR Workshop Proceedings. CEUR Workshop Proceedings, 2023, pp. 146–152. URL: https://ceur-ws.org/Vol-3630/LWDA2023-paper13.pdf.

[59] Sven Langenecker, Christoph Sturm, Christian Schalles, and Carsten Binnig. "SportsTables: A new Corpus for Semantic Type Detection." In: *Datenbanksysteme für Business, Technologie und Web (BTW 2023), 20. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme" (DBIS), 06.-10, März 2023, Dresden, Germany, Proceedings*. Ed. by Birgitta König-Ries, Stefanie Scherzinger,

Wolfgang Lehner, and Gottfried Vossen. Vol. P-331. LNI. Gesellschaft für Informatik e.V., 2023, pp. 995–1008. DOI: 10.18420/BTW2023-68. URL: https://doi.org/10.18420/BTW2023-68.

[60] Sven Langenecker, Christoph Sturm, Christian Schalles, and Carsten Binnig. "SportsTables: A New Corpus for Semantic Type Detection (Extended Version)." In: *Datenbank-Spektrum* 23.2 (2023). DOI: 10.1007/s13222-023-00457-y. URL: https://doi.org/10.1007/s13222-023-00457-y.

[61] Sven Langenecker, Christoph Sturm, Christian Schalles, and Carsten Binnig. "Steered Training Data Generation for Learned Semantic Type Detection." In: *Proc. ACM Manag. Data* 1.2 (2023), 201:1–201:25. DOI: 10.1145/3589786. URL: https://doi.org/10.1145/3589786.

[62] Sven Langenecker, Christoph Sturm, Christian Schalles, and Carsten Binnig. "Towards Learned Metadata Extraction for Data Lakes." In: *Datenbanksysteme für Business, Technologie und Web (BTW 2021), 19. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme" (DBIS), 13.-17. September 2021, Dresden, Germany, Proceedings*. Ed. by Kai-Uwe Sattler, Melanie Herschel, and Wolfgang Lehner. Vol. P-311. LNI. Gesellschaft für Informatik, Bonn, 2021, pp. 325–336. DOI: 10.18420/BTW2021-17. URL: https://doi.org/10.18420/btw2021-17.

[63] Quoc Le and Tomas Mikolov. "Distributed Representations of Sentences and Documents." In: *ICML*. 2014.

[64] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks." In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 9459–9474. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf.

[65] Xirong Li, Shuai Liao, Weiyu Lan, Xiaoyong Du, and Gang Yang. "Zero-Shot Image Tagging by Hierarchical Semantic Embedding." In: *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '15. Santiago, Chile: Association for Computing Machinery, 2015, pp. 879–882. ISBN: 9781450336215. DOI: 10.1145/2766462.2767773. URL: https://doi.org/10.1145/2766462.2767773.

[66] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. "Deep Entity Matching with Pre-Trained Language Models." In: *Proc. VLDB Endow.* 14.1 (Sept. 2020), pp. 50–60. ISSN: 2150-8097. DOI: 10.14778/3421424.3421431. URL: https://doi.org/10.14778/3421424.3421431.

[67] Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. "Annotating and Searching Web Tables Using Entities, Types and Relationships." In: *Proc. VLDB Endow.* 3.1–2 (Sept. 2010), pp. 1338–1347. ISSN: 2150-8097. DOI: 10.14778/1920841.1921005. URL: https://doi.org/10.14778/1920841.1921005.

[68] Yujia Liu, Kang Zeng, Haiyang Wang, Xin Song, and Bin Zhou. "Content Matters: A GNN-Based Model Combined with Text Semantics for Social Network Cascade Prediction." In: *Advances in Knowledge Discovery and Data Mining*. Ed. by Kamal Karlapalem, Hong Cheng, Naren Ramakrishnan, R. K. Agrawal, P. Krishna Reddy, Jaideep Srivastava, and Tanmoy Chakraborty. Cham: Springer International Publishing, 2021, pp. 728–740. ISBN: 978-3-030-75762-5.

[69] Mohammad Mahdavi, Ziawasch Abedjan, Raul Castro Fernandez, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. "Raha: A Configuration-Free Error Detection System." In: *SIGMOD '19*. ACM, 2019.

[70] Subhadip Maji, Swapna Sourav Rout, and Sudeep Choudhary. "DCoM: A Deep Column Mapper for Semantic Data Type Detection." In: *CoRR* abs/2106.12871 (2021). arXiv: 2106.12871. URL: https://arxiv.org/abs/2106.12871.

[71] Neil Mallinar, Abhishek Shah, Tin Kam Ho, Rajendra Ugrani, and Ayush Gupta. "Iterative Data Programming for Expanding Text Classification Corpora." In: *AAAI'20*. AAAI Press, 2020, pp. 13332–13337. URL: https://ojs.aaai.org/index.php/AAAI/article/view/7045.

[72] Christian Mathis. "Data Lakes." In: *Datenbank-Spektrum* 17.3 (Nov. 2017), pp. 289–293. ISSN: 1618-2162. DOI: 10.1007/s13222-017-0272-7.

[73] *Azure Purview: 100+ standard data-types for auto- tagging*. https://docs.microsoft.com/en-us/azure/purview/supported-classifications. Accessed: 2022-10-15. 2022.

[74] *Microsoft Power BI, Interactive Data Visualization BI*. https://powerbi.microsoft.com. Accessed: 2022-10-15. 2022.

[75] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. "Distributed Representations of Words and Phrases and Their Compositionality." In: *NIPS'13*. 2013, pp. 3111–3119.

[76] Johann Mitlöhner, Sebastian Neumaier, Jürgen Umbrich, and Axel Polleres. "Characteristics of Open Data CSV Files." English. In: *International Conference on Open and Big Data (OBD)*. Ed. by IEEE. IEEE, 2016, pp. 72–79. DOI: 10.1109/OBD.2016.18.

[77] Fatemeh Nargesian, Ken Q. Pu, Erkang Zhu, Bahar Ghadiri Bashardoost, and Renée J. Miller. "Organizing Data Lakes for Navigation." In: *SIGMOD '20*. ACM, 2020, pp. 1939–1950. ISBN: 9781450367356. DOI: 10.1145/3318464.3380605. URL: https://doi.org/10.1145/3318464.3380605.

[78] Fatemeh Nargesian, Erkang Zhu, Renée J Miller, Ken Q Pu UOIT, and Patricia C Arocena. "Data Lake Management: Challenges and Opportunities." In: *Proc. VLDB Endow.* 12.12 (2019), pp. 1986–1989. DOI: 10.14778/3352063.3352116. URL: https://doi.org/10.14778/3352063.3352116.

[79] Mona Nashaat, Aindrila Ghosh, James Miller, and Shaikh Quader. "Asterisk: Generating Large Training Datasets with Automatic Active Supervision." In: *ACM/IMS Trans. Data Sci.* 1.2 (May 2020). ISSN: 2691-1922. DOI: 10.1145/3385188. URL: https://doi.org/10.1145/3385188.

[80] Sebastian Neumaier, Jürgen Umbrich, Josiane Xavier Parreira, and Axel Polleres. "Multi-level Semantic Labelling of Numerical Values." In: *The Semantic Web – ISWC 2016*. Cham, 2016, pp. 428–445. DOI: 10.1007/978-3-319-46523-4. URL: http://5stardata.info/.

[81] Sebastian Neumaier, Jürgen Umbrich, and Axel Polleres. "Automated Quality Assessment of Metadata across Open Data Portals." In: *J. Data and Information Quality* 8.1 (Oct. 2016). ISSN: 1936-1955. DOI: 10.1145/2964909. URL: https://doi.org/10.1145/2964909.

[82] Daniela Oliveira and Catia Pesquita. *SemTab 2021 BioTable Dataset*. Version 1.0.0. Zenodo, Oct. 2021. DOI: 10.5281/zenodo.5606585. URL: https://doi.org/10.5281/zenodo.5606585.

[83] OpenAI. *GPT-4 Technical Report*. Tech. rep. 2023, pp. 1–100. arXiv: 2303.08774. URL: http://arxiv.org/abs/2303.08774.

[84] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke E. Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Francis Christiano, Jan Leike, and Ryan J. Lowe.

"Training language models to follow instructions with human feedback." In: *ArXiv* abs/2203.02155 (2022).

[85] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library." In: *NeurIPS*. Ed. by Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché Buc, Emily B. Fox, and Roman Garnett. 2019, pp. 8024–8035. URL: http://dblp.uni-trier.de/db/conf/nips/nips2019.html#PaszkeGMLBCKLGA19.

[86] Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick S. H. Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander H. Miller. "Language Models as Knowledge Bases?" In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*. Ed. by Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan. Association for Computational Linguistics, 2019, pp. 2463–2473. DOI: 10.18653/V1/D19-1250. URL: https://doi.org/10.18653/v1/D19-1250.

[87] Sundar Pichai and Demis Hassabis. *Introducing Gemini: our largest and most capable AI model*. 2023. URL: https://blog.google/technology/ai/google-gemini-ai/#sundar-note (visited on 12/20/2023).

[88] Plotly. *Plotly*. https://chart-studio.plotly.com/feed/. 2018. (Visited on 12/14/2022).

[89] Christoph Quix and Rihan Hai. "Data Lake." In: *Encyclopedia of Big Data Technologies*. Ed. by Sherif Sakr and Albert Zomaya. Cham: Springer International Publishing, 2018, pp. 1–8. ISBN: 978-3-319-63962-8. DOI: 10.1007/978-3-319-63962-8_7-1. URL: https://doi.org/10.1007/978-3-319-63962-8_7-1.

[90] Christoph Quix, Rihan Hai, and Ivan Vatov. "Metadata Extraction and Management in Data Lakes With GEMMS." In: *Complex Syst. Informatics Model. Q.* 9 (2016), pp. 67–83. ISSN: 2255-9922. DOI: 10.7250/csimq.2016-9.04. URL: https://doi.org/10.7250/csimq.2016-9.04.

[91] S. K. Ramnandan, Amol Mittal, Craig A. Knoblock, and Pedro Szekely. "Assigning Semantic Labels to Data Sources." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in*

Bioinformatics) 9088 (2015), pp. 403–417. ISSN: 16113349. DOI: 10.1007/978-3-319-18818-8.

[92] Alexander Ratner, Stephen H. Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. "Snorkel: Rapid Training Data Creation with Weak Supervision." In: *Proc. VLDB Endow.* 11.3 (Nov. 2017), pp. 269–282. ISSN: 2150-8097. DOI: 10.14778/3157794.3157797. URL: https://doi.org/10.14778/3157794.3157797.

[93] Alexander Ratner, Christopher De Sa, Sen Wu, Daniel Selsam, and Christopher Ré. "Data Programming: Creating Large Training Sets, Quickly." In: *NIPS*. Barcelona, Spain: Curran Associates Inc., 2016, pp. 3574–3582. ISBN: 9781510838819.

[94] Franck Ravat and Yan Zhao. "Metadata Management for Data Lakes." In: *New Trends in Databases and Information Systems*. Springer, 2019, pp. 37–44. DOI: 10.1007/978-3-030-30278-8_5. URL: http://oatao.univ-toulouse.fr/25044.

[95] Manon Réau, Nicolas Renaud, Li C Xue, and Alexandre M J J Bonvin. "DeepRank-GNN: a graph neural network framework to learn patterns in protein–protein interfaces." In: *Bioinformatics* 39.1 (Nov. 2022), btac759. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btac759. eprint: https://academic.oup.com/bioinformatics/article-pdf/39/1/btac759/48448994/btac759.pdf. URL: https://doi.org/10.1093/bioinformatics/btac759.

[96] "Results of SemTab 2021." In: (2021). DOI: 10.5281/zenodo.6211551. URL: https://doi.org/10.5281/zenodo.6211551.

[97] Shivam Sawarn and Gerard Deepak. "MASSTagger: Metadata Aware Semantic Strategy for Automatic Image Tagging." In: *Digital Technologies and Applications*. Ed. by Saad Motahhir and Badre Bossoufi. Cham: Springer International Publishing, 2022, pp. 429–438. ISBN: 978-3-031-01942-5.

[98] Claude Elwood Shannon. "A Mathematical Theory of Communication." In: *The Bell System Technical Journal* 27 (1948), pp. 379–423. URL: http://plan9.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf (visited on 04/22/2003).

[99] Richard Socher, Brody Huval, Christopher D Manning, and Andrew Y Ng. "Semantic compositionality through recursive matrix-vector spaces." In: *Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning*. Association for Computational Linguistics. 2012, pp. 1201–1211.

[100] Yoshihiko Suhara, Jinfeng Li, Yuliang Li, Dan Zhang, Çağatay Demiralp, Chen Chen, and Wang-Chiew Tan. "Annotating Columns with Pre-Trained Language Models." In: *SIGMOD*. New York, NY, USA: ACM, 2022, pp. 1493–1503. ISBN: 9781450392495.

[101] A SALESFORCE COMPANY TABLEAU SOFTWARE LLC. *Tableau Public*. https://public.tableau.com. 2023.

[102] Nan Tang, Ju Fan, Fangyi Li, Jianhong Tu, Xiaoyong Du, Guoliang Li, Sam Madden, and Mourad Ouzzani. "RPT: Relational Pre-Trained Transformer is Almost All You Need towards Democratizing Data Preparation." In: *Proc. VLDB Endow.* 14.8 (Apr. 2021), pp. 1254–1261. ISSN: 2150-8097. DOI: 10.14778/3457390.3457391. URL: https://doi.org/10.14778/3457390.3457391.

[103] DGL Team. *HeteroGraphConv*. DGL Team. Oct. 2023. URL: https://docs.dgl.ai/generated/dgl.nn.pytorch.HeteroGraphConv.html.

[104] Paroma Varma and Christopher Ré. "Snuba: Automating Weak Supervision to Label Training Data." In: *VLDB*. Vol. 12. 3. VLDB Endowment, Nov. 2018, pp. 223–236. DOI: 10.14778/3291264.3291268. URL: https://doi.org/10.14778/3291264.3291268.

[105] Fernanda B. Viegas, Martin Wattenberg, Frank van Ham, Jesse Kriss, and Matt McKeon. "ManyEyes: A Site for Visualization at Internet Scale." In: *IEEE Transactions on Visualization and Computer Graphics* 13.6 (Nov. 2007), pp. 1121–1128. ISSN: 1077-2626. DOI: 10.1109/TVCG.2007.70577. URL: https://doi.org/10.1109/TVCG.2007.70577.

[106] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. "Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks." In: *arXiv preprint arXiv:1909.01315* (2019).

[107] Zhiruo Wang, Haoyu Dong, Ran Jia, Jia Li, Zhiyi Fu, Shi Han, and Dongmei Zhang. "TUTA: Tree-based Transformers for Generally Structured Table Pre-training." In: *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining* (2020).

[108] Roger C.F. Wong and Clement H.C. Leung. "Automatic Semantic Annotation of Real-World Web Images." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30.11 (2008), pp. 1933–1944. DOI: 10.1109/TPAMI.2008.125.

[109] Jiayang Wu, Wensheng Gan, Zefeng Chen, Shicheng Wan, and Philip S. Yu. "Multimodal Large Language Models: A Survey." In: (2023). arXiv: 2306.13549. URL: https://arxiv.org/abs/2311.13165.

[110] Cong Yan and Yeye He. "Synthesizing Type-Detection Logic for Rich Semantic Data Types Using Open-Source Code." In: *SIGMOD '18*. ACM, 2018, pp. 35–50. ISBN: 9781450347037. DOI: 10.1145/3183713.3196888. URL: https://doi.org/10.1145/3183713.3196888.

[111] Yinfei Yang, Daniel Cer, Amin Ahmad, Mandy Guo, Jax Law, Noah Constant, Gustavo Hernandez Abrego, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. "Multilingual Universal Sentence Encoder for Semantic Retrieval." In: *CoRR* abs/1907.04307 (2019). arXiv: 1907.04307.

[112] Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. "TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data." In: *ACL2020*. Online: ACL, July 2020, pp. 8413–8426. DOI: 10.18653/v1/2020.acl-main.745. URL: https://aclanthology.org/2020.acl-main.745.

[113] Dan Zhang, Madelon Hulsebos, Yoshihiko Suhara, Çağatay Demiralp, Jinfeng Li, and Wang-Chiew Tan. "Sato: Contextual Semantic Type Detection in Tables." In: *VLDB*. Vol. 13. 12. VLDB Endowment, July 2020, pp. 1835–1848. DOI: 10.14778/3407790.3407793. URL: https://doi.org/10.14778/3407790.3407793.

[114] Meihui Zhang, Marios Hadjieleftheriou, Beng Chin Ooi, Cecilia M. Procopiuc, and Divesh Srivastava. "Automatic Discovery of Attributes in Relational Databases." In: *SIGMOD*. New York, NY, USA: ACM, 2011, pp. 109–120. ISBN: 9781450306614.