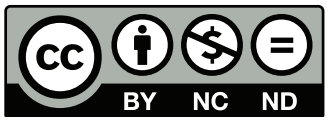
STATSREP-ML: Statistical Evaluation & Reporting Framework for Machine Learning Results

Technical Report submitted by
Christian Guckelsberger[a], Axel Schulz[b]
[a] Goldsmiths, University of London, Computational Creativity Group, UK
[b] Technische Universität Darmstadt, Telecooperation Lab, Germany

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Goldsmiths
UNIVERSITY OF LONDON



1 Introduction

Data mining aims at finding regularities and patterns in data or extracting valuable data automatically by differentiating noise from relevant information. For example, in the emergency management domain, social media data is automatically filtered using information extraction and machine learning techniques [8, 6, 7]. During the last decades, many different classifiers and approaches for feature extraction have been developed to perform this automatic classification. However, finding the best approaches is challenging due to the large number of possible combinations.

In the past 10 years, standardized qualitative and inferential statistical methods for comparing machine learning results were proposed as a replacement for rather arbitrary approaches [5, 2, 4, 3]. Nevertheless, evaluating machine-learning models manually with traditional statistics packages such as R[1] or SPSS[2] is labour-intensive and error-prone due to the complex evaluation procedure and the correct choice of methods. Furthermore, many packages do not offer dedicated procedures for reporting results from this particular domain in a comprehensive way.

STATSREP-ML[3] is an open-source solution for automating the process of evaluating machine-learning results. It calculates qualitative statistics, performs the appropriate tests and reports them in a comprehensive way. It largely, but not exclusively, relies on well-tested and robust statistics implementations in R, and uses the tests the machine-learning community largely agreed upon. Since this is an ongoing debate, the framework was designed in an extensible fashion so addition tests can be added easily.

STATSREP-ML is supported by the DKPro Text Classification (TC) Framework[4] [1], which alleviates supervised machine learning experiments with any kind of textual data. DKPro TC provides a report class to export machine learning data into the STATSREP-ML format, fostering the seamless integration of model learning, testing, evaluation and reporting.

In detail, the features of STATSREP-ML are:

- Straight-forward configuration, either programmatically or using an XML file.
- Support for sample input from k-fold cross-validation, repeated cross-validation on one or multiple datasets, and train-test splits.

[1] http://www.r-project.org
[2] http://www.ibm.com/software/de/analytics/spss/
[3] https://code.google.com/p/statistical-evaluation-for-machine-learning/
[4] http://code.google.com/p/dkpro-tc/

- Support for two or > 2 values of the independent variable, i.e. either classifiers or features. The appropriate tests for the number of groups are selected automatically.
- Support for sample sets with two independent variables, by automatically splitting the data along a predefined fixed independent variable value.
- Integration of both parametric and non-parametric omnibus- as well as post-hoc tests that are commonly used for comparing machine learning results.
- Integration of specific parametric and non-parametric tests to compare multiple models against a baseline, and support for input data annotations to indicate the baseline model.
- Automatic p-value correction and integration of several more and less conservative techniques for p-value adjustment.
- Testing of parametric test assumptions such as normality and sphericity, allowing an easy application of these algorithms.
- Generation of both a plain-text and a better structured LaTeX report, comprising sample tables, qualitative statistics, basic graphs and the evaluation results.

We describe the scope of the STATSREP-ML framework in more detail in Section 2. We then show how the framework can be set-up and configured in Sections 3 and 4, respectively. In Section 5, we describe the required sample data format, available methods for importing data and several examples highlighting typical use-cases. Section 6 describes the types of reports resulting from the evaluation and their content. To meet the future requirements of the research community, we added an additional Section 7 with detailed instructions on extending the framework by new tests. We then conclude our report in Section 8, followed by an appendix comprising setup guidelines and a short bibliography.

2 Scope

STATSREP-ML supports a large number of different types of evaluations, arising from the combination of the number of models to be evaluated, the sampling strategy of the performance values, the independent variable and the type of model comparison. In this section, we describe these parameters and their possible values in detail.

The system accepts *data sampled from k-fold cross-validation and repeated cross-validation on one or multiple datasets as well as from train-test splits*. In k-fold cross-validation, the original data is partitioned into k equally sized folds of subsamples. Subsequently, the models are trained on k-1 of these folds and tested on the remaining one. This process is repeated k times until each subsample was once

used for testing. A k-fold cross-validation generates k samples of the performance measure. In a repeated cross-validation, this process is repeated for an arbitrary number of times. Both versions can be performed over multiple datasets, too. In the latter two cases, STATSREP-ML expects the samples to be already averaged over repeated cross-validations and / or datasets. In the train-test scenario, the model is trained on one dataset and tested on another. This is usually done for multiple training- and testing-datasets. Please note that both cross-validation and train-test splits have their drawbacks: On the one hand, performance samples sourced from cross-validation are pairwise statistically dependent because they were trained on k-2 common folds. On the other hand, measures from train-test splits and sampling procedures involving multiple datasets in general might be incommensurate [2]. The framework does currently not process samples, e.g. by means of aggregation, any further. Selecting the sampling source of data is currently only required for correctly reading the samples from the input file and setting specific variables in the report.

STATSREP-ML accepts up to two independent variables, i.e. either the classification algorithms, feature sets or both can be varied. Nevertheless, the non-parametric tests only allow for one independent variable. In the case of two, one user-predefined variable is therefore fixed and the data is split into multiple sample sets and evaluated separately, resulting in multiple reports.

The framework supports both the *comparison of two and more than two models*. In this context, we understand a "model" as unique combination of a classification algorithm and a feature set. If more than two models are compared, both omnibus and post-hoc tests are performed to determine whether there is a significant difference in the performance of all models, and to identify which individual models differ significantly. It is important to note that the comparison of more than two models is subject to the multiple comparisons problem: When more than one hypothesis is examined, the individual test's p-values must be adjusted in order to draw conclusions over all hypotheses. The module supports different more or less conservative methods for adjustment[5], but they all come with the same drawback: the p-values are increased with each individual comparison, making it harder to find significant results. A larger number of samples per model can contribute to better p-values and can thus be used to compensate p-value inflation. STATSREP-ML offers two additional means to cope with this problem: The user can set a parameter to constrain the compared models only to the best k according to a chosen performance measure. Furthermore, the user can choose between *different means to compare more than two models*: In pairwise comparisons, each model is compared with each other, resulting in $\frac{N \times (N-1)}{2}$ comparisons. In a baseline comparison, $N-1$ models are compared

[5] Bonferroni, Hochberg, Holm, Benjamini-Hochberg, Benjamini-Yekutieli

against a baseline model, but not with each other, resulting in only $N-1$ comparisons. A baseline comparison provides less information, but inflates the p-values less. In the next section, we describe how STATSREP-ML has to be set-up in order to work properly.

3 Setup

The qualitative and inferential statistics as well as the diagram plotting routines are not directly implemented in Java, but executed in R. This makes it possible to rely on the wide range of validated statistical packages available on CRAN[6], the Comprehensive R Archive Network. The rJava framework[7] facilitates the communication between Java and R. STATSREP-ML does therefore not run standalone, but required the user to set-up R, the JRI framework and few path variables beforehand. The operating-system specific instructions are given in the appendix. In the next section, we describe how STATSREP-ML can be configured.

4 Configuration

STATSREP-ML can be configured either programmatically or by preparing an XML-based configuration file. Both methods have the same power. The class `ExampleInputTester.java` illustrates how the configuration is set programmatically. Alternatively, the configuration file is read into STATSREP-ML by using the following java code in the main class:

StatsConfig config = StatsConfig.getInstance("config.xml");

For illustration purposes, Figure 4 shows the standard xml configuration file that comes with the current version of the framework. In detail, the configuration file comprises the following evaluation parameters:

- `tests:` an array comprising the concrete parametric and non-parametric omnibus and post-hoc tests to be used when evaluating two or multiple models pairwise or against a baseline.
- `pCorrections:` an array comprising the methods for p-value correction to be used after multiple comparisons in a post-hoc test.
- `significanceLevels:` a set of values for low/medium/high significance levels for which test results with p-values below would be deemed low/moderate/strongly significant in the report.
- `selectBest:` two fields that allow the evaluation of a subset of models to keep p-value inflation low. The user has to specify the maximum number of models

[6] `http://cran.r-project.org`
[7] `http://rforge.net/rJava/`

```
<configuration>
<tests>
        <test>
                <class>TwoSamplesParametric</class>
                <name>DependentT</name>
        </test>
        <test>
                <class>TwoSamplesNonParametric</class>
                <name>WilcoxonSignedRank</name>
        </test>
        <test>
                <class>TwoSamplesNonParametricContingency</class>
                <name>McNemar</name>
        </test>
        <test>
                <class>MultipleSamplesParametric</class>
                <name>RepeatedMeasuresOneWayANOVA</name>
        </test>
        <test>
                <class>MultipleSamplesNonParametric</class>
                <name>Friedman</name>
        </test>
        <test>
                <class>MultipleSamplesParametricPosthoc</class>
                <name>Tukey</name>
        </test>
        <test>
                <class>MultipleSamplesNonParametricPostHoc</class>
                <name>Nemenyi</name>
        </test>
                <test>
                <class>MultipleSamplesParametricPosthocBaseline</class>
                <name>Dunett</name>
        </test>
        <test>
                <class>MultipleSamplesNonParametricPostHocBaseline</class
                    >
                <name>PairwiseWilcoxonSignedRank</name>
        </test>
</tests>
<pCorrections>
        <pCorrection>bonferroni</pCorrection>
        <pCorrection>holm</pCorrection>
        <pCorrection>hochberg</pCorrection>
        <pCorrection>hommel</pCorrection>
</pCorrections>
<significanceLevels>
        <significanceLevel>
                <level>low</level>
                <value>0.1</value>
        </significanceLevel>
                <significanceLevel>
                <level>medium</level>
                <value>0.05</value>
        </significanceLevel>
                <significanceLevel>
                <level>high</level>
                <value>0.01</value>
        </significanceLevel>
</significanceLevels>
<selectBest>
        <count>10</count>
        <measure>Weighted F-Measure</measure>
</selectBest>
<fixIndependentVariable>FeatureSet</fixIndependentVariable>
</configuration>
```

Figure 1: The standard xml configuration file

that should be evaluated, and the name of the performance measure according to which they should be selected. A baseline model will always be preserved.

- `fixIndependentVariable:` as the framework only supports one independent variable, this entry allows to either fix the Classifiers or the FeatureSets independent variable if both vary, and split the data into multiple sample sets for separate evaluation.

The allowed key-value pairs for these parameters are stored in the class `StatsConfigConstants.java`. The available tests, measures and correction methods are listed there as well. When altering the used tests, please verify that a chosen post-hoc test matches the respective a priori test. For example, a Mann-Whitney-test cannot be used post-hoc to Friedman's test, because it is not based on the Tukey statistic. In the next section, we describe how input data must be structured and imported in order to perform a statistical evaluation.

5 Importing Sample Data

The framework requires externally defined performance samples as input. In the following, we first describe the required data format. We then elaborate on how the data is imported into STATSREP-ML and illustrate them by means of several examples.

5.1 Specifying Evaluation Data

This data must adhere to the format in Table 5.1, and should be stored as `.csv` with an arbitrary separator. In the following, we describe the different columns in detail:

Train	Test	Classifier	FeatureSet	Measure	Value	IsBaseline
Dataset1	Dataset2	LibLinear	nGrams: 2	F-Measure	0,549661	0
Dataset1	Dataset3	LibLinear	nGrams: 2	F-Measure	0,553766	0
Dataset1	Dataset4	LibLinear	nGrams: 2	F-Measure	0,549661	0
Dataset2	Dataset1	LibLinear	+ALL, nGrams: 2	F-Measure	0,439875	1
Dataset2	Dataset3	LibLinear	+ALL, nGrams: 2	F-Measure	0,574944	1
Dataset2	Dataset4	LibLinear	+ALL, nGrams: 2	F-Measure	0,66053	1
...	...	...	...	...	...	...

Table 1: Format of externally defined data from a Train-Test sampling.

- `Train/Test:` These two columns specify the names of the datasets used for training and testing. If evaluations on a single dataset are performed, e.g. an ordinary cross-validation, columns one and two must be identical. It is statistically critical to group samples that were evaluated on the same dataset. In case of an ordinary cross-validation with non-aggregated performance samples, it is

therefore required to add a unique identifier for every fold, separated by a dot ".", after the dataset name. For example, both columns may contain the entries "Boston.Fold1' in one row, followed by "Boston.Fold3" and "Boston.Fold2' in the next rows.

- `Classifier:` This column is used as the naming of the respective classifier names and parameters used for configuring the classifier. This value represents an independent variables. Avoiding typos is thus critical for successful processing.
- `FeatureSet:` In this column, the unique name of a certain feature combination is listed. As in the prior case, avoiding typos is critical for successfully distinguishing values of the independent variable.
- `Measure:` Several measures might be used for evaluation, and can be specified in the same file. This column comprises the names of these measures.
- `Value:` This column contains the value of the performance measure calculated on the models' score on the datasets specified in the same row.
- `IsBaseline:` This entry is used to specify if a specific classifier/feature combination should be used as a baseline. If a baseline is given, all other models are compared against this baseline. Please note that if models should be compared against a baseline in the case of two independent variables, one baseline model must be specified for each value of the fixed independent variable. Section 5.3 contains examples for this case.

Please note that statistical tests require a `minimum amount of samples per model` to deliver meaningful results. This minimum has been set to five in STATSREP-ML, although it is highly recommended to provide at least ten or more samples per model. The evaluation will not be continued if either less samples are provided, or if the number of samples differs between models.

5.2 Reading Evaluation Data

It is critical to choose the appropriate import method for the given input data to ensure correct processing and reporting. The class `StatsProcessor.java` contains three methods for importing and evaluating data:

- `evaluateCV`(StatsConfig config, String pathToCsvFile, String outputPath, char separator)
- `evaluateRepeatedCV`(StatsConfig config, String pathToCsvFile, String outputPath, char separator, int nFolds)
- `evaluateTrainTest`(StatsConfig config, String pathToCsvFile, String outputPath, char separator)

Each method requires an object representing the current configuration as a first argument. Also, the paths to the input and output files needs to be given. Finally, the user can specify the separator that is used in the CSV file. In the case of a repeated CV, the number of folds needs to be provided. Please note that this is only used for correctly reporting the results; the framework still expects the data to be already aggregated. In case of a repeated k-fold cross validation, the samples could be averaged over all folds, resulting in as many samples per model as there are repetitions.

5.3 Examples

STATSREP-ML provides several examples for typical use cases in class `ExampleInputTester.java`:

- The `CV` methods reads an example file with performance samples from a cross-validation of three classification algorithms and a fixed feature set. Since this is only about a single independent variable, it returns one report in which models are compared based on their classification algorithm.
- The `CV2IV` method reads an example file with performance samples from a cross-validation for four classification algorithms and two feature sets. Because this implies two independent variables, STATSREP-ML will split the dataset according to the value set in `setFixIndependentVariable` and conduct separate evaluations for each subset. In this case, the feature sets are fixed and only different classification algorithms with the same feature set are compared. Furthermore, results are also generated for the variation of feature groups and keeping the classifier fixed, eventually showing the best feature combination on the datasets.
- The `CVClassifierBaseline` method is a subset of the first example. In this case, only one feature combination is present, but multiple classifiers are compared. In contrast to the first example, one feature group/classifier combination is used as a baseline strategy, eventually indicating whether other combinations are outperforming this one.
- The `CVFeaturesBaseline` method is equivalent to the second example, but in this case several feature combinations are evaluated for only one classifier. Also here, a baseline comparison is used.
- The `TrainTest2FeaturesBaseline` method reads an example file with performance samples from training and testing one classifier with 10 different feature sets on 10 datasets. One feature set represents a baseline which the others extend and which they are compared against.

The examples also illustrate the different approaches to configure STATSREP-ML: While in `CV`, the configuration is done exhaustively and programmatically,

i.e. all parameters are set manually, the method `CV2IV` uses a default constructor to load standard configuration values which can be modified later on. `CVClassifierBaseline` demonstrates a third approach by reading the configuration from an xml file.

The input files used in the examples are located in the `src/main/resources/examples` package. The first four sample sets were calculated with DKPro TC[8] [1] on the Twenty Newsgroups dataset[9]. The examples illustrate that STATSREP-ML allows to evaluate the best feature groups and best classifiers and to select the most appropriate comparison method for a given learning problem. In the next section, we describe how the evaluation results will be reported.

6 The Evaluation Reports

STATSREP-ML always produces two types of reports: a plain-text file and a little more comprehensive, better structured and readable LaTeX-file with tables and additional diagrams. The latter must be compiled separately. Both reports contain:

- An overview of the evaluated models, listing their feature sets and classifiers.
- A description of the type of evaluation, including a list of the datasets used and additional information specific to the sampling procedure, e.g. the number of folds in a k-fold cross-validation.
- A table with the individual samples for each performance measure.
- A description of both omnibus and post-hoc parametric and non-parametric tests performed, including tests of their assumptions as well as parameters.
- Tables with uncorrected and corrected p-values for post-hoc tests.
- An ordering of relationships between the models based on their statistically significant differences, or a sentence indicating that no such ordering is possible.
- A short summary of the findings.

Ordering models using their significant differences is done in accordance with [3] based on topological ordering of a directed acyclic graph. In addition to better readability, the LaTeX report offers:

- A description of the statistical significance levels and corresponding annotations in the p-value tables.

[8] `http://code.google.com/p/dkpro-tc/`
[9] `http://qwone.com/~jason/20Newsgroups/`

- Different table formats for listing samples from a single- or from multiple datasets.
- Different table formats for pairwise and baseline comparisons.
- Various diagrams:
  - A Q-Q-plot for visually validating the assumption of normality, comparing the quantiles of the samples with the quantiles of a normal distribution.
  - A Box-Whisker-plot contrasting the performances of the individual models. The means were added as red dots.
  - A directed graph representing the statistical differences between the models. If an ordering between all models cannot be inferred, this graph can be used to manually infer an ordered subset of models.

The output folder of the reports is logged. Please activate the logger output level "INFO" to see it. The next section will focus on how to extend the present methods and also represent new tests in the report.

7 Extending STATSREP-ML with New Tests

STATSREP-ML is highly extensible thanks to Java's Reflection facilities, and can thus be adapted to new developments in the machine learning evaluation community. New statistical tests can be added by following these steps:

1. A new test, either as a wrapper method including R code or a as a native implementation, must be added to the class `Statistics`. Either the JRI javadoc [10] or the present tests can be used to learn about the JRI syntax. The method has to adhere to the following rules:
   - The name of the test method must start with "test", followed by the test's name.
   - An omnibus test must return an object of type `TestResult`, while a post-hoc test must return an object of class `PairwiseTestResult`. While the former can only represent a single p-value, statistic, etc., the latter can store several p-values from multiple comparisons. Tests in R often return objects of type `htest` or `pairwise.htest`. The method `AbstractTestResult toTestResult(String resourceName)` can be used to translate these objects into their Java counterparts, by first transforming them into an object of class `AbstractTestResult` and then casting them to either `TestResult` or `PairwiseTestResult`.
   - The method must return null if the test fails.

[10] `forge.net/org/docs/`

- Post-hoc tests must indicate in their results whether p-values must still be corrected. The method `setRequiresPValueCorrection(bool requires)` on the test object must be used to set the appropriate flag.
- Omnibus tests for two samples require two double arrays as sole parameters, one for each sample set.
- Omnibus tests for > 2 samples and the corresponding post-hoc tests require a 2-dimensional double-array as sole parameter, where columns represent the different models and rows the samples.
- The test should be implemented in a robust way, allowing to deal with empty or incomplete parameters, and include proper exception handling and logging. The existing tests can be used as a template.
- The test should be properly documented, and contain references if it represents a less known approach.

2. The exact name of the new test (case sentitive!) must be added to the respective category in `StatsConfigConstants.java` in order to make it available to the configuration.
3. A key-value pair from the test's name to the test's pretty-print name must be added in `StatsConfigConstants.java`.
4. To use the test, its (non pretty-print) name must be added to the `config.xml` file in the respective category, or must be used in the programmatic configuration.

8 Conclusion

In this report, we introduced the STATSREP-ML framework for performing and reporting qualitative and inferential statistics on machine-learning results. The framework requires the performance samples from machine learning experiments to be stored in a CSV file, and can therefore be used as part of a larger evaluation pipeline. It can be configured both by means of an XML file or programmatically, fostering the direct integration in other frameworks. For instance, the DKPro TC framework supports STATSREP-ML to join automated model learning and testing with performance data evaluation and reporting. STATSREP-ML was already used extensively for evaluating results in the domain of tweet classification, illustrating the capabilities the framework offers for scientific research.

In the future, we plan to adapt the framework to developments in the research community and to add additional tests. It is also planned to integrate additional diagrams, such as Demsar's critical difference diagram for illustrating Friedman-style ranking results and pairwise significance [2]. Further improvements will be

presented on the project's website[11] and contributions of future users are highly appreciated.

9 Appendix: Setup Guidelines

STATSREP-ML requires the user to set-up dynamic libraries for R and JRI as well as path variables. This process varies slightly between different operating systems. Please make sure to use the same architecture (x64 or x86) for the different components (R, rJava, Java).

9.1 Windows

Tested on Windows 8.1

1. Download R[12] and install it.
2. Open the right binary (architecture wise), e.g. for x64: C:\Program Files\R\R-3.1.0\bin\x64\R.exe (use admin rights).
3. Use `install.packages("rJava")` to download rJava, remember the given folder path.
4. Navigate to that path and extract the zip, preferably somewhere together with the R installation.
5. Specify a PATH variable in the run configurations (of your main or testing class), where you add the paths to the native libraries (jri.dll, jvm.dll, R.dll). E.g. (be careful with the architecture folders): "C:\Program Files\R\R-3.1.0\rJava\jri\x64;C:\Program Files\Java\jdk1.7.0_51\jre\bin\server;C:\Program Files\R\R-3.1.0\bin\x64" (jvm.dll is sometimes not in \server but in \client).

9.2 Linux

Tested on Ubuntu 14.04 Trusty Tahr

1. Download R[12]. and install it. You may also use your distributions' package manager to download and install the software: In Ubuntu, type `sudo apt-get install r-base` into your console to download and install the R base package and all dependencies.
2. Please ensure that a link to your R executable is in your /bin or /usr/bin directory. This should be the case if you install the software using the native package manager. Typing the command `R` in the console will start the R command prompt.

[11] `https://code.google.com/p/statistical-evaluation-for-machine-learning/`
[12] `http://www.r-project.org/`

3. Retrieve the R installation path by typing `R.home(component = "home")` into the R command prompt. Set the R_HOME path variable to the directory where R has been installed. Example: `export R_HOME=/usr/lib/R`. To fix path variables, add them to the .bashrc file or the respective profile file in your distribution.

4. On the R command prompt, type `install.packages("rJava")` to download and install rJava. This interface will allow communication between R and the MUGC software.

5. Locate the rJava installation directory with the command `sudo find / -name rJava`. Add the subdirectory jri with the JAR files JRIEngine.jar, JRI.jar, REngine.jar to the CLASSPATH. Example: `export CLASSPATH` =.:/usr/lib/R/site-library/rJava/jri/.

6. Add the native JRI-library to the LD_LIBRARY_PATH variable. It should be located within the same directory as the JAR files. Example: `export LD_LIBRARY_PATH` = /usr/lib/R/site-library/rJava/jri/.

9.3 MacOS

Tested on OSX 10.10 Yosemite

1. Download R[12] and install it. You may also use homebrew[13] to download and install the software: `brew tap homebrew/science; brew install gcc; brew install r;`.

2. Install JRI library in R. Via command line: `r; install.packages('rJava')`.

3. Determine library folder. In R, type `.libPaths()`.

4. Add a reference to the JRI libraries as additional VM arguments. E.g., `-java.library.path`=/Library/Frameworks/R.framework/Versions/3.1/ Resources/library/rJava/jri/.

5. Specify R_HOME as path variable (in eclipse, use the `run configurations` tab of your main or testing classes) and add the path to the native libraries of R. E.g., `R_HOME` = /Library/Frameworks/R.framework/Versions/3.1/Resources/.

STATSREP-ML may ask to install additional R packages during the first run. Confirming this will install the required packages into the project folder.

[13] `http://www.brew.sh/`

References

[1] Johannes Daxenberger, Oliver Ferschke, Iryna Gurevych, and Torsten Zesch. DKPro TC: A java-based framework for supervised learning experiments on textual data. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 61–66, Baltimore, Maryland, June 2014. Association for Computational Linguistics.

[2] Janez Demsar. Statistical Comparisons of Classifiers over Multiple Data Sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[3] Manuel J. A. Eugster, Torsten Hothorn, and Friedrich Leisch. Exploratory and inferential analysis of benchmark experiments. Technical Report 030, 2008.

[4] Francisco Herrera. An Extension on "Statistical Comparisons of Classifiers over Multiple Data Sets" for all Pairwise Comparisons. *Journal of Machine Learning Research*, 9:2677–2694, 2008.

[5] Mohak Jakowicz, Nathalie; Shah. *Evaluating Learning Algorithms. A Classification Perspective*. Cambridge University Press, Cambridge, 2011.

[6] Axel Schulz, Aristotelis Hadjakos, Heiko Paulheim, Johannes Nachtwey, and Max Mühlhäuser. A multi-indicator approach for geolocalization of tweets. In *International AAAI Conference on Weblogs and Social Media*, 2013.

[7] Axel Schulz and Frederik Janssen. What is good for one city may not be good for another one: Evaluating generalization for tweet classification based on semantic abstraction. In CEUR, editor, *Proceedings of the Fifth Workshop on Semantics for Smarter Cities a Workshop at the 13th International Semantic Web Conference*, volume 1280, pages 53–67, 2014.

[8] Axel Schulz, Petar Ristoski, and Heiko Paulheim. I see a car crash: Real-time detection of small scale incidents in microblogs. In *The Semantic Web: ESWC 2013 Satellite Events*, number 7955 in Lecture Notes in Computer Science, pages 22–33. Springer Berlin Heidelberg, 2013.