Friedrich-Alexander-Universität Erlangen-Nürnberg
Technische Fakultät, Department Informatik

LEONHARD HÖSCH
BACHELOR THESIS

USING MACHINE LEARNING TO CLASSIFY OPEN SOURCE PROJECTS

Submitted on 1 July 2014

Supervisor: Prof. Dr. Dirk Riehle, M.B.A.
Professur für Open-Source-Software
Department Informatik, Technische Fakultät
Friedrich-Alexander-Universität Erlangen-Nürnberg

Contents

Versicherung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, 1 July 2014

License

Erlangen, 1 July 2014

Abstract

Classifying open source software projects in the Ohloh data (Chapter 2.4: Used Data Sources) manually would take a very long time. For further research on the Ohloh data, it would be helpful to have the projects classified into groups. In order to to this, data from a snapshot of the ohloh project is analyzed and then machine learning techniques are used to classify the data. The results of the classification can then be added to the Ohloh data for further research.

1 Introduction

The Ohloh database is accessible via web under `ohloh.net` or REST API. The data was collected by crawling the web for open source projects as well as by users who added their projects to the Ohloh database. Users are also allowed to add and correct data concerning projects. The Ohloh data has already been used for several analysis by (Bruntink, 2014),(Gysin & Kuhn, 2010), (Hu & Zhao, 2008), (Deshpande & Riehle, 2008), (Arafat & Riehle, 2009) and others. In order to improve future analysis, we analyze the ohloh data and classify the projects into functional groups. As the big amount of data makes it really inefficient and time consuming to classify the existing data, we use a machine learning approach for classification.

1.1 Thesis goals

- Model of types of open source projects
- Algorithm using machine learning to classify open source projects
- Evaluation of algorithm using information retrieval measures

2 Research

2.1 Introduction

Whenever there is enough data to be analyzed, it is likely to find patterns in it. These patterns can be used on new data to gain some more information about them. If the data is big enough not to be analyzed manually, machine learning can be employed. This thesis will use machine learning to classify open source projects from the Ohloh database.

2.2 Related Work

As mentioned in the introduction, there already have been several analysis conducted on the Ohloh data by (Bruntink, 2014), (Gysin & Kuhn, 2010), (Hu & Zhao, 2008), (Deshpande & Riehle, 2008), (Arafat & Riehle, 2009) and others. The applied machine learning techniques have been discussed in multiple papers, some of which are referenced towards the end of 2.6. Despite a lot of Research, we couldn't find any research on how to classify open source software into functional categories. Although there has been a lot of work done in the field of mining software repositories, we didn't find any functional classifications of whole software projects.

2.3 Research Question

Is it possible to classify open source projects into functional categories? What are these categories? Is machine learning suited to classify the projects in the Ohloh database?

2.4 Research Approach

First, we will analyze the Ohloh data manually using the R package ggplot2. In order to do this, a snapshot of the Ohloh data is accessed via PostgreSQL. We use a snapshot, so that we don't need to query the `Ohloh.net` web service every time we run the classification and to avoid getting other results just because some new projects were added in the meantime.

Afterwards, we will define a classification model for the Ohloh data. This means we will first identify the data which might carry some information concerning the classification. This data has to be structured to run the supervised machine learning algorithms on it. Then, training and test data have to be defined.

Next, we will identify one or more machine learning algorithms, which can be run on the data.

Finally, we will evaluate the used algorithms.

2.5 Used Data Sources

Ohloh database:

- Representative sample (2013) of open source projects
- Mostly 'hard' project data like 'project name', 'numbers of code', etc., configuration management information, tags
- Easily accessible in an relational schema
- Accessible via Ohloh.net and REST API

2.6 Research Results

The snapshot of the data from Ohloh is saved locally at the Friedrich-Alexander-University at Erlangen-Nürnberg and accessed via PostgreSQL. For the layout of this database, please see Chapter 3: Layout.

As there was not much literature to be found about classifying Software at all, we defined some prioritized requirements for a classification model suited for machine learning:

1. Usefulness in further analysis (i.e. functional information on the projects)
2. Ease of creating training data for supervised machine learning

3. Preferably orthogonal categories

The reasons for these requirements are explained in Chapter 3.1: Construction of Model requirements.

We defined a model that tries to fulfill these criteria as best as possible:

- Entertainment: Anything related to games and videos.
- Workplace: This is anything related to a typical developer workplace.
- Infrastructure: Things like frameworks, web infrastructure, drivers, ...

These categories will be useful in further analysis, as they provide a differentiation in functionality. Also it will be possible to create training data based on the tags (as explained below). The categories are not orthogonal, but we had to accept this as the other two criteria were more important.
We discussed creating a 'others' category but since we couldn't find any corresponding tags, we decided to drop it.
We choose to run our further analysis on the tags because they are already structured and can be easily used to create training data. The relation between projects and tags is a N:M relation. Also, we suspected, that tags carry information about the functional classes as they were created/added by users to order their project into (partly) functional sets. We hoped that we could extract this information from the tags with our machine learning algorithms. From this point on, we will ignore all projects without tags.
To judge the importance of the tags, several analyzes were conducted on the data:

- The number of tags per project:

  There are some projects with a really big number of tags (up to 178 tags per project) while most of the projects have less than 50 tags. The graph only shows the top 2.000 projects with the highest number of tags.

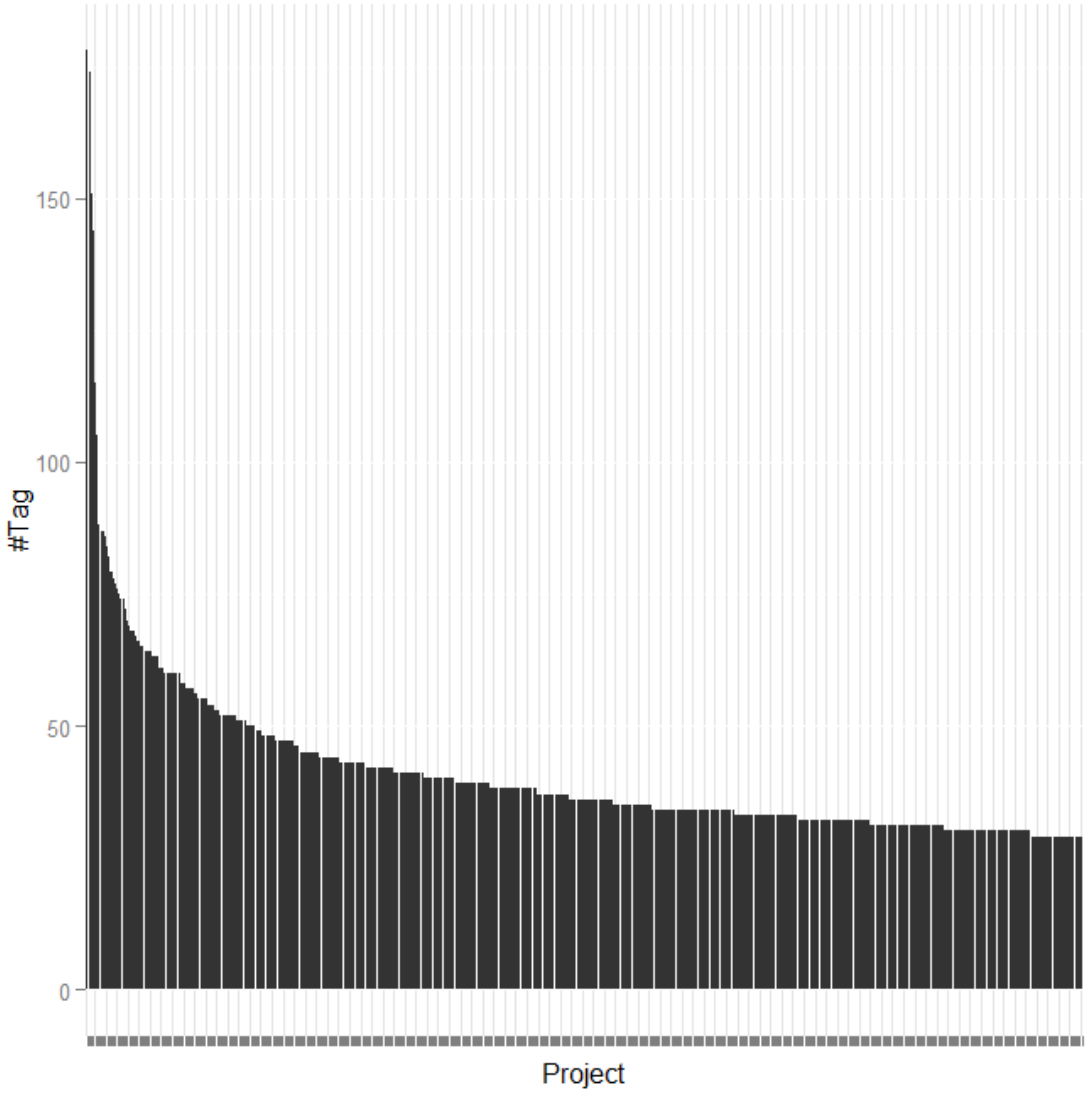

Figure 2.1: The number of tags per project

- The number of usage of the top 200 tags:

  Despite these being the 200 top used tags, it is clearly visible, that there are a couple of really often used tags (most of these tags are programming languages) while the usage of most tags is smaller by several orders of magnitude. We consider tags that are used often to (possibly) carry a big amount of information, because they aggregate a big amount of objects into a (possibly functional) group. Tags that are rarely used can't carry such a big amount of information and might even lead to over specialization for some machine learning algorithms. Because of this and performance issues,

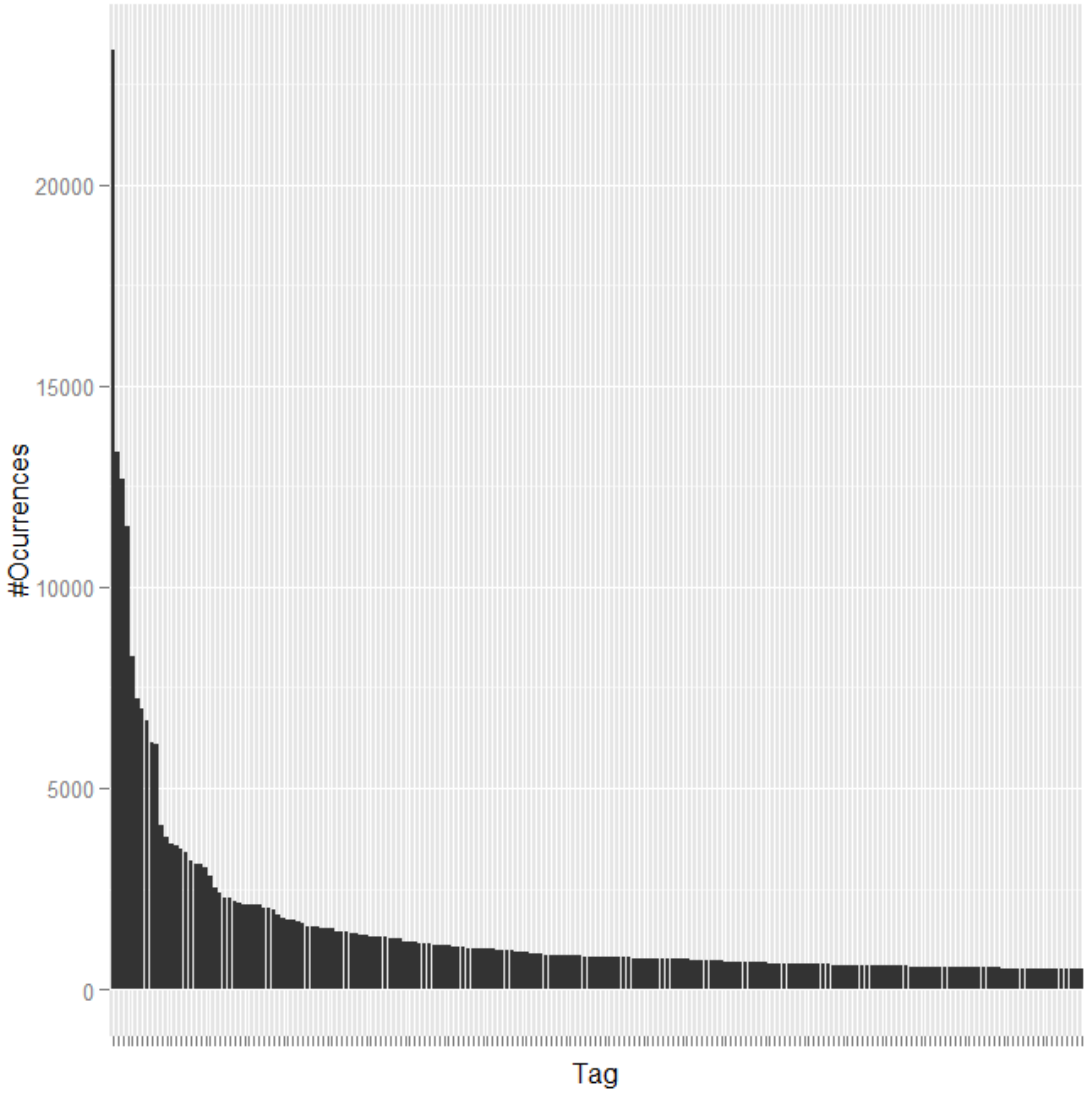

Figure 2.2: The total occurrences of the tags

we decided to only consider the 700 most used tags for further analysis. The number 700 was determined experimentally, so that the run-time of the algorithms would be at around one day.

- A histogram of the tag usage:

  This histogramm shows, how often tags are used n times ($0 \leq n \leq 250$). A big amount of tags is used under 10 times and carries therefore very little information. This supports our decision to only use the 700 most used tags.

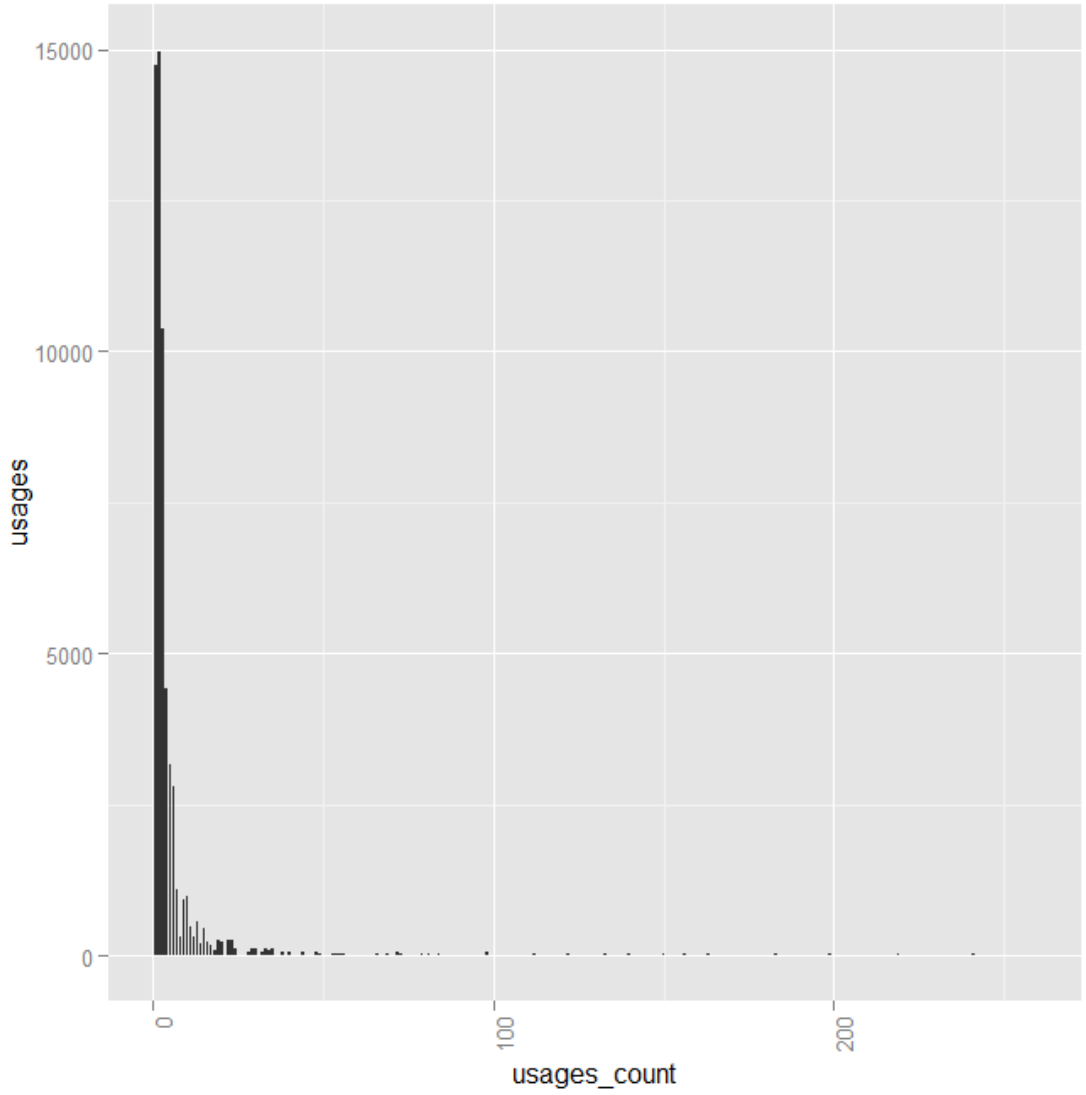

Figure 2.3: Histogram of Tag usages

In order to get the training data set, we searched for tags that seemed to fit into our categories. In particular these tags are:

- Entertainment: 'game', 'games', 'gaming', 'video'
- Infrastructure: 'infrastructure', 'framework'
- Workplace: 'ide', 'editor'

We assumed that each project, that contained at least one of these tags was to be belonging to the corresponding functional class. We accepted the fact that this is not true for every case, as the machine learning algorithms are able to deal with

uncertainty in its input data. For the same reason, the fact that some projects were added to two categories is not a problem.

The training data now consists of tuples (project_ID, tag1, tag2, ..., tag700, class). To define a test set, we selected 1000 random values from the training data set and deleted them from the training data set.

As the work was already done using R, we choose to use the package RTextTools (Timothy P. Jurka, Loren Collingwood, Amber E. Boydstun, Emiliano Grossman and Wouter van Atteveldt, 2012) to analyze the data. It employs several different machine learning algorithms and combines them to get the best results. Also, it has the capability to evaluate the algorithms used.

The employed algorithms are:

- Support Vector Machines (svm) construct one or more hyper-planes in a high-dimensional space. These hyper-planes are constructed using a maximum-margin-separator to provide the broadest categories. The number of dimensions can be higher then the number of input dimensions to keep the separators linear. In the original dimensions, the separator can thus be non-linear. For a detailed description of how svm work, please see (Norvig & Russell, 2012) S. 863-868.
- Boosting or in this case the LogitBoost algorithm assigns base classifiers to the training data. Each Iteration, the miss-classified data has their weighting decreased, whereas the correct classifiers' weights are increased. The classifiers are adjusted accordingly. This leads to a focus on the miss-classified data and thus a better learning. The iterations can be set to a given number or by stopping when the binomial log-likelihood for each boosting iteration is at its maximum. (Dettling & Buhlmann, 2003)
- Supervised latent Districhlet allocation (sLDA) is based on latent Districhlet allocation (LDA), which is a topic model, that treats the topic proportions for each document as a draw from a Ditrichlet distribution. The words in the document are obtained by repeatedly choosing a topic assignment from those proportions, then drawing a word from the corresponding topic. sLDA adds to LDA a valued variable associated with each document. The documents and responses are modeled, so that there are latent topics found, that will best predict the response variables for future (not valued) documents. (Mcauliffe & Blei, 2008)
- Bagging is a method for building ensembles that use different subsets of the training data to produce classifiers. Bagging draws a number of random instances from the training set with replacement and learns the drawn subset. This is repeated several times. Each Repetition of this process leads to a single classifier. At the End, taking a vote of these classifiers leads to

the final prediction. (Maglogiannis, Karpouzis, Wallace & Soldatos, 2007, S. 15)

- Random Forests are a generalization of recursive partitioning. The Training data is partitioned. These partitions are classified via trees. These trees can be built by recursive partitioning. Each node of these trees only contains members of a certain class. To classify new data, all of the trees are used upon it and vote for the result. The most often chosen result is the final prediction. (Clarke, Fokoue & Zhang, 2009, S. 254)
- Glmnet combines elastic-net penalties with generalized linear models. The used models are regression, two-class logistic regression and multinomial regression problems. (Friedman, Hastie & Tibshirani, 2010)
- Decision Trees are trees generated from a random subset of the test data (e.g. by divide and conquer) and are valued as they represent only a part of the full set. New data is classified by a valued vote. (Quinlan, 1986)
- Neural networks are probably the most common form of machine learning. They consist of neurons which are connected to former and following neurons. When the former neuron's (weighted) output exceeds a certain value, the current neuron is activated and 'fires' its output. Training a neural net is usually done via some regression method and requires finding the best network structure as well as the correct activation function for each neuron. (Russell & Norvig, 2009, S. 727-737)
- Maximum entropy is a stochastic model. Its main principle is: 'Model all that is known and assume nothing about that which is unknown.' In this case, 'assume nothing' means 'assume equal distribution'. Data from the training set is interpreted as statistical constraints. Each of these constraints make the model more accurate. (Berger, Pietra & Pietra, 1996, S. 40-43)

To evaluate the effectiveness of the algorithms, we run the algorithm 10 times with randomly selected test data. The mean value of correct classified data was 81%. The standard deviation is 0,50685% while the biggest deviation was 1,1%. This process took around 27 Hours per run and needed around 16GB of RAM at peak on a workstation (see Appendix 1: Workstation Configuration). As the system is not a real time system, the values for the run-time and RAM are only estimates and therefore rounded.

For the **Figures 2.4-2.7**, we measured the accuracy (the number of right results divided by the number of result class X) and recall (the number of right results divided by the number of results that should have been returned for class X) for each algorithm and each class and computed the f-score ($2 \times \frac{precision \times recall}{precision + recall}$). The f-score represents the overall quality of the algorithm:

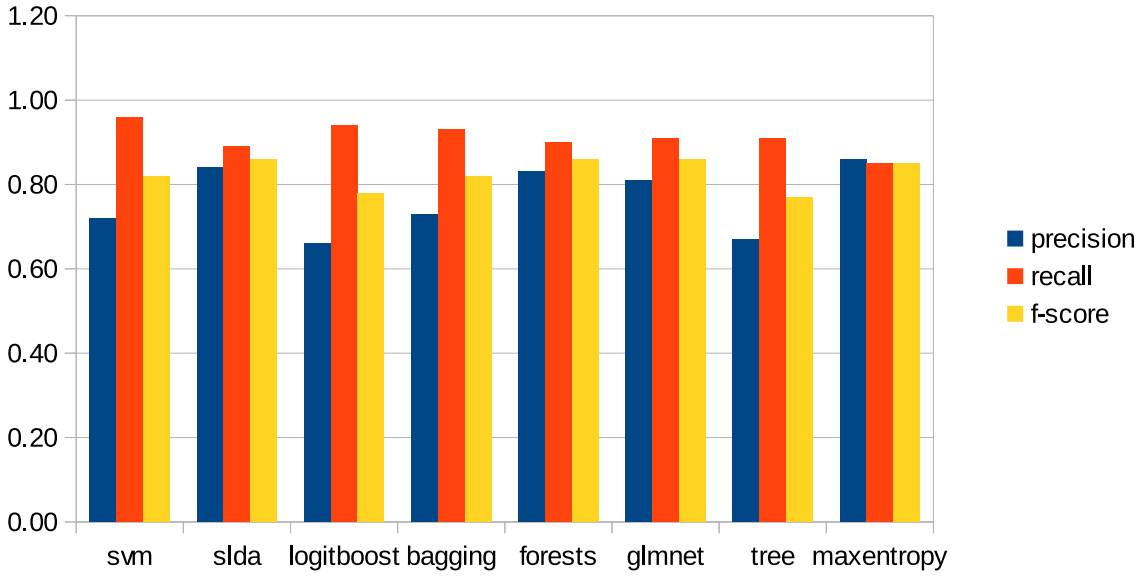

Figure 2.4: Results for Entertainment.

For Infrastructure and Entertainment, the values in **Figures 2.4-2.7** are quite acceptable. The recall for Workplace on the other side is low compared to the others' recall, while the accuracy is still high. This means, that if the algorithms return the result 'Workplace', they are mostly right, but they misplace a big amount of actual 'Workplace' results into the other classes.
This is due to the fact that only 3,8% of the actual training data is classified as Workplace, whereas 47,2% of the data is classified as Entertainment and 49,0% is classified as Infrastructure. This makes the Workplace statistically less important and the algorithms heavily favor the other classes when in doubt. SVM and Logitboost illustrate this, as they are extremely accurate when deciding for Workplace but, when in doubt, they just favor the other two classes. This phenomenon is called over-specialization. Other algorithms are more resistant to this effect, especially sLDA. To sum it up, the accuracy of the Workplace class is lower to keep overall accuracy as high as possible.

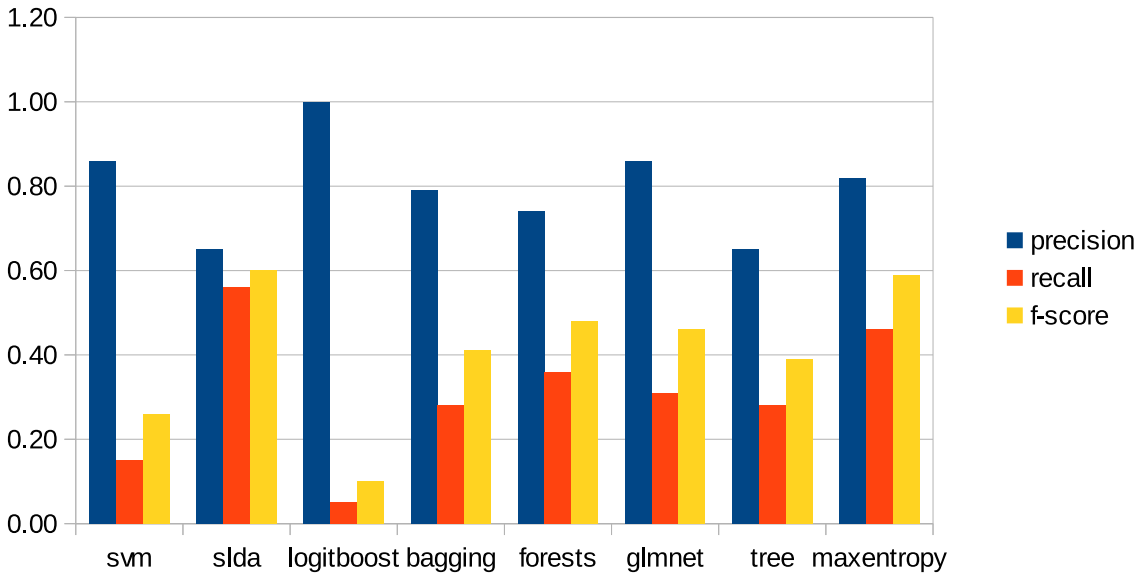

Figure 2.5: Results for Workplace.

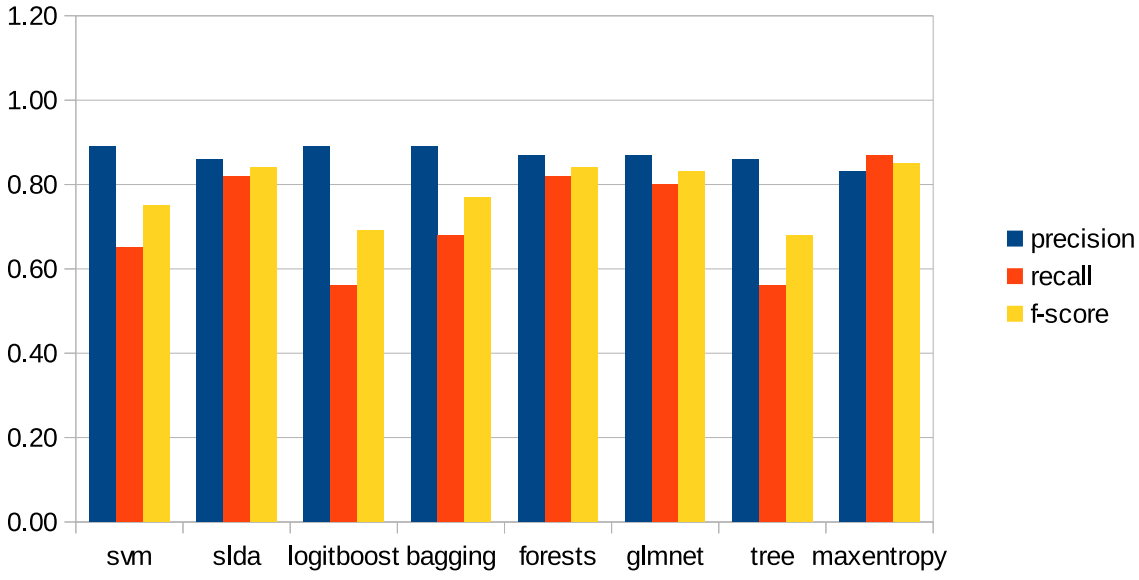

Figure 2.6: Results for Infrastructure.

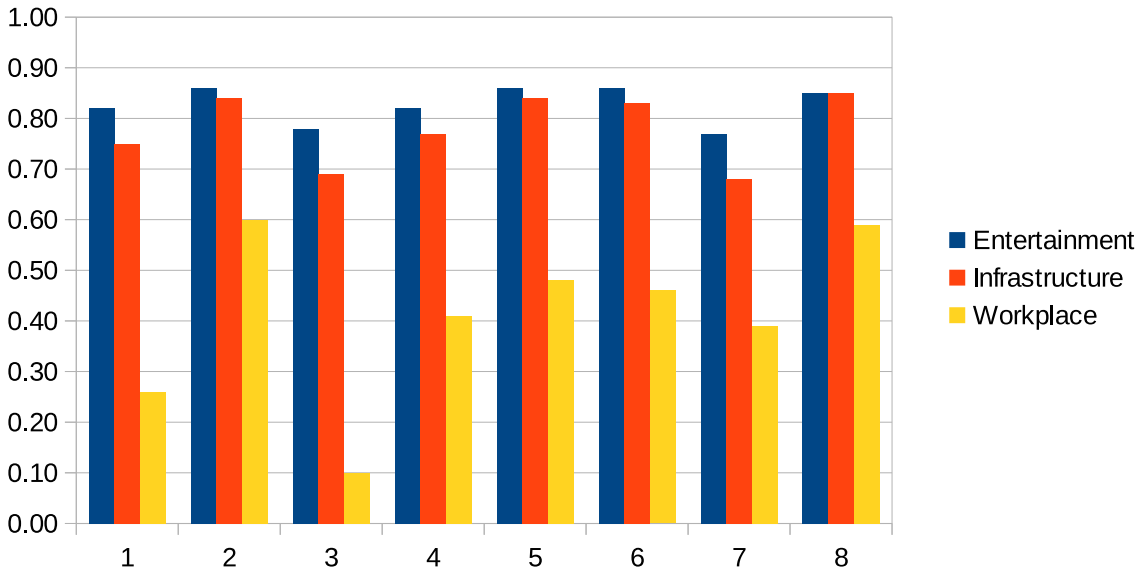

Figure 2.7: Compared f-Scores.

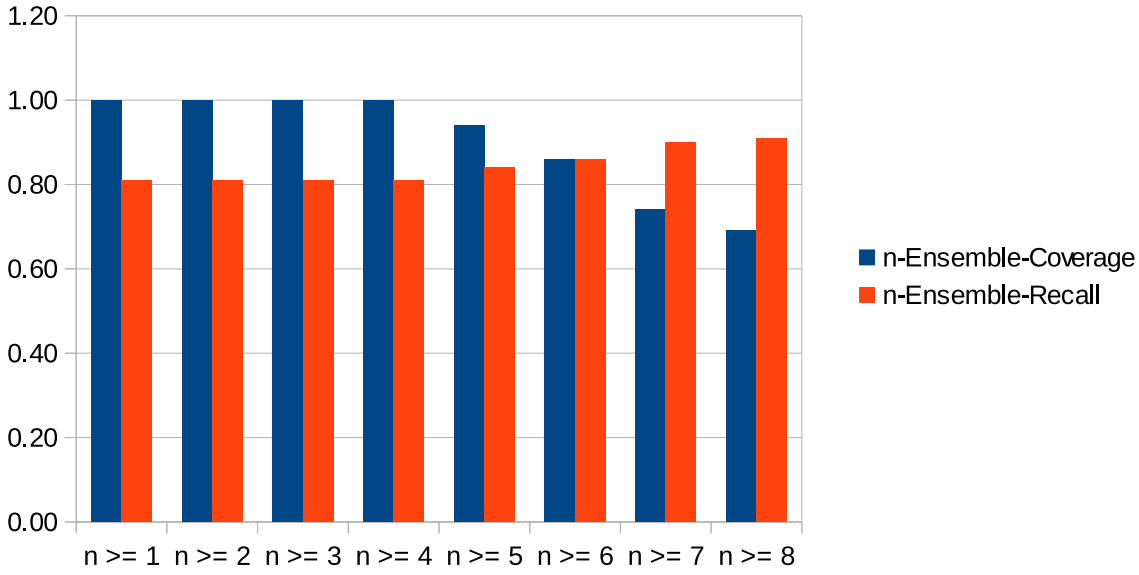

Figure 2.8: Ensemble results

Figure 2.8 shows the n-ensemble coverage, which means that n or more algorithms agree on the result and the n-ensemble recall, which is the percentage of right answers on which the majority of algorithms agrees. As an example, when all eight algorithms agree ($n \geq 8$) they got a 91% recall, but this only happens in 69% of all cases (n-coverage).
As less algorithms agree on the same class, the accuracy decreases to 81%.

2.7 Results Discussion

All in all, the algorithm classifies 81% of the projects right. Although some workplace projects are not classified correctly, the overall quality of the algorithm allows a classification. Further possible improvements are discussed in section 2.8 Conclusion and Future Work.

There are still some biases, which might have some influence on the data:

- The training data might have a bias, as the heuristic of viewing the existence of a tag as sufficient for classifying might not be right in all cases. This bias could be eliminated by getting feedback from the users of `Ohloh.net` (See the proposal in chapter 2.8).
- If a sufficiently large group starts to use the tags in a complete different way, which is not represented in the training data, the classification will not work as intended until the new use is represented in the training data and the algorithms are run once again. It can't be guaranteed, that this didn't happen at some point in the past and is reflected in the data. If this were the case, the quality of the training data would be limited.

2.8 Conclusions and Future Work

The data from Ohloh can be classified by using the tags to create training data efficiently. Afterwards, 81% of the tagged projects could be categorized right.

Improvements in the accuracy can be made by improving the training data:

- The categories can be suggested to the users of the Ohloh website. The accepted and rejected (i.e. changed) categories can then be saved again and being used as new training data. With each iteration, the training data and the algorithms should become more accurate.
- It might be possible to find some more tags that fit into the category workplace that we overlooked. Depending on the number of corresponding projects for these tags, this could have a big influence on the algorithms.

When more tags get frequently used or computation time is less relevant, it might be worth to increase the computation time to expand the number of used tags.

If the chosen classification is not the best for a specific future research, a fitting classification can be achieved by preparing a new training data set and letting the algorithms run again. Again the iterative method above can be used here to improve this training data set.

As the choice of Workplace as a functional class seems not optimal, other classes could be obtained by running a clustering algorithm on the tags.

3 Elaboration of Research

3.1 Ohloh Database Layout

The layout of the Database:

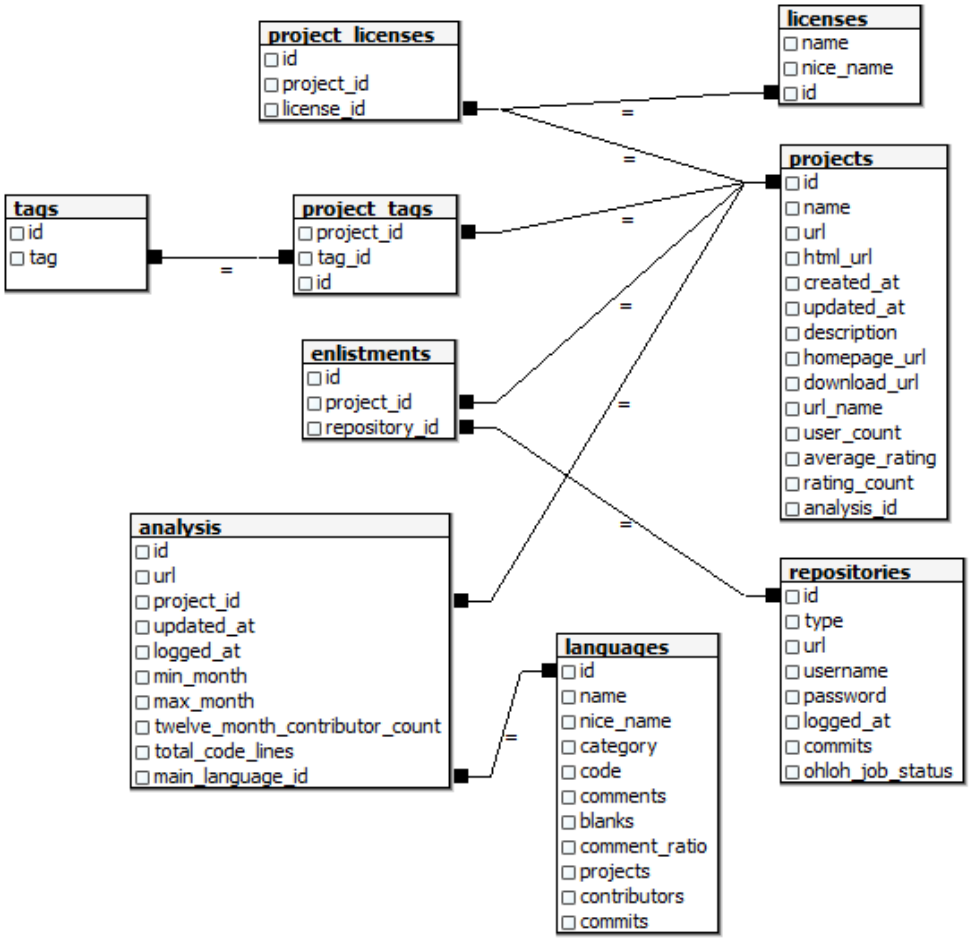

Figure 3.1: Relational Data Model of Ohloh

In our research, we used the relations tags, project_tags and projects.

3.2 Construction of Model requirements

The first point guarantees that the results of this work can be used for further analysis.
The second one is important, because it is time consuming and therefore extremely inefficient to create the training data manually with big data sets. As the Ohloh project currently has about 50.000 projects, it is not an option to manually classify a sufficient part of these projects.
Although some overlapping of the training data (i.e. training data objects that are placed in multiple categories) will not effect supervised learning too much, most supervised machine learning algorithms can only place data into one category. In this process, the information about the second category would be lost. Because of this, we want to have orthogonal categories.

4 Appendices

Appendix A Workstation Configuration

Configuration of the workstation:

- Intel(R) Core(TM) i7-950
- ASUS X56-Sabretooth
- 3x 8GB Kingston KHX1600C9D3K2/8GX
- Windows 8.1 x64
- R version 3.0.2 (Build 2013-09-25)
- Rtexttools (Timothy P. Jurka, Loren Collingwood, Amber E. Boydstun, Emiliano Grossman and Wouter van Atteveldt, 2012)

Bibliography

Arafat, O. & Riehle, D. (2009). The commit size distribution of open source software. In *System sciences, 2009. hicss'09. 42nd hawaii international conference on* (pp. 1–8).

Berger, A. L., Pietra, V. J. D. & Pietra, S. A. D. (1996, March). A maximum entropy approach to natural language processing. *Comput. Linguist.*, *22*(1), 39–71. Retrieved from `http://dl.acm.org/citation.cfm?id=234285.234289`

Bruntink, M. (2014). An initial quality analysis of the ohloh software evolution data. *Electronic Communications of the EASST*.

Clarke, B., Fokoue, E. & Zhang, H. (2009). *Principles and theory for data mining and machine learning*. Springer. Retrieved from `http://books.google.de/books?id=RQHB4_p3bJoC`

Deshpande, A. & Riehle, D. (2008). Continuous integration in open source software development. In *Open source development, communities and quality* (pp. 273–280). Springer.

Dettling, M. & Buhlmann, P. (2003). Boosting for tumor classification with gene expression data. *Bioinformatics*, *19*(9), 1061–1069.

Friedman, J., Hastie, T. & Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, *33*(1), 1–22. Retrieved from `http://www.jstatsoft.org/v33/i01/`

Gysin, F. S. & Kuhn, A. (2010). A trustability metric for code search based on developer karma. In *Proceedings of 2010 icse workshop on search-driven development: Users, infrastructure, tools and evaluation* (pp. 41–44).

Hu, D. & Zhao, J. L. (2008). A comparison of evaluation networks and collaboration networks in open source software communities. *AMCIS 2008 Proceedings*, 277.

Maglogiannis, I., Karpouzis, K., Wallace, M. & Soldatos, J. (Eds.). (2007). *Emerging artificial intelligence applications in computer engineering - real word ai systems with applications in ehealth, hci, information retrieval and pervasive technologies* (Vol. 160). IOS Press. Retrieved from `http://dblp.uni-trier.de/db/series/faia/faia160.html`

Mcauliffe, J. D. & Blei, D. M. (2008). Supervised topic models. In J. Platt, D. Koller, Y. Singer & S. Roweis (Eds.), *Advances in neural information processing systems 20* (pp. 121–128). Curran Associates, Inc. Retrieved from http://papers.nips.cc/paper/3328-supervised-topic-models.pdf

Norvig, P. & Russell, S. (2012). *Künstliche intelligenz.* Pearson Studium. Retrieved from http://books.google.de/books?id=NHV4LgEACAAJ

Quinlan, J. (1986). Induction of decision trees. *Machine Learning*, *1*(1), 81-106. Retrieved from http://dx.doi.org/10.1023/A%3A1022643204877 doi: 10.1023/A:1022643204877

Russell, S. & Norvig, P. (2009). *Artificial intelligence: A modern approach* (3rd ed.). Upper Saddle River, NJ, USA: Prentice Hall Press.

Timothy P. Jurka, Loren Collingwood, Amber E. Boydstun, Emiliano Grossman and Wouter van Atteveldt. (2012). *Rtexttools: Automatic text classification via supervised learning.* http://CRAN.R-project.org/package=RTextTools. (R package version 1.3.9; Online; accessed 28 March 2014)