ClassifyHub: An Algorithm to Classify GitHub Repositories

Marcus Soll ✉(https://orcid.org/0000-0002-6845-9825) and Malte Vosgerau

University of Hamburg, Mittelweg 177, 20148 Hamburg, Germany,
(2soll, 2vosgera)@informatik.uni-hamburg.de

Abstract. The classification of repositories found on GitHub can be considered as a hard task. However, the solution of this task could be helpful for a lot of different applications (e.g. Recommender Systems). In this paper we present ClassifyHub, an algorithm based on Ensemble Learning developed for the InformatiCup 2017 competition, which is able to tackle this classification problem with high precision and recall. In addition we provide a data set of classified repositories for further research.

1 Introduction

GitHub is the largest [3] platform to organise and collaborate on different projects (so-called repositories). The diversity of *GitHub* repositories reaches from small LaTeX templates for homework up to huge software development projects. Because of this diversity *GitHub* is ideal for many research projects (e.g. influence of programming languages to code quality [10] or social studies [14]).

To gain additional value out of the large variety of repositories on *GitHub* it would be useful to classify these repositories into different disjunctive classes. Such *clustering* could be used for many tasks like, for example, recommendation (so-called *Recommender Systems*) [1, 11]. Another application would be the improvement of search functions on *GitHub*.

In this paper we present *ClassifyHub*, an algorithm based on *Ensemble Learning* which tackles the *GitHub Classification Problem* and achieves high precision and high recall considering the hard task. This algorithm reached the final round in the InformatiCup 2017 competition, where the goal was to develop a complete software solution with a time frame of about 5 month. In addition we provide a data set with 681 classified repositories which can be used for further research.

2 Related Work

Ugurel et al. [15] classified source code archives into different application types (like *database* or *games*), however they did not focus their work on content types (like *educational* or *data set*) and furthermore only focused on application source code instead of arbitrary repositories (like repositories containing only images).

The final publication is available at Springer via
http://dx.doi.org/10.1007/978-3-319-67190-1_34

Kawaguchi et al. [6] proposed a system which classifies software in automatically generated categories. Again, the categories seem to focus on application types rather than content types. In addition, they used categories about the technology used (e.g. libraries), as well as the architecture of the software.

Maskeri et al. [8] proposed a system to automatically extract topics from the source code. These topics are more related to the implementation (like *SSL* or *Logging*) than to the content type.

3 GitHub Classification Problem

The *GitHub Classification Problem* (based on the InformatiCup 2017 task) is a problem from the area of classification. The task is the classification of repositories hosted on *GitHub* into exactly one of the following content categories:

- **DEV**: Software development projects and similar
- **HW**: Solutions for homework, exercises and similar
- **EDU**: Projects with educational purpose and similar
- **DOCS**: Documents with no educational intent and similar
- **WEB**: (Personal) websites
- **DATA**: Data sets
- **OTHER**: Repositories which do not fit in one of the above categories

Two aspects turn the *GitHub Classification Problem* into a hard tasks:

- There is a large variety of repositories on *GitHub*. One can find repositories with projects run by one person up to repositories with thousands of contributors (e.g. the Linux kernel).
- The classification of repositories is sometimes ambiguous - many projects can be classified into multiple categories.

4 Multi Classifier Solution

To tackle the *GitHub Classification Problem* we used an approach based on *Ensemble Learning* [13]: Through the combination of multiple *weak classifier* (which have to be better than random guessing) we get a single *strong classifier* which is correct on the majority of data. This works because each *weak classifier* added reduces the total error of the *strong classifier*. In our solution the probability of a class is equal to the average probability calculated by all *weak classifiers*, as shown in (1).

$$P(\text{class}) = \frac{\sum^{\text{classifier}} P_{\text{classifier}}(\text{class})}{N_{\text{classifier}}} \tag{1}$$

4.1 Weak Classifier Used

Each *weak classifier* presented in the following sections returns a value between zero and one which represents the probability with which the classifier classifies a repository into one of the classes. The total sum of all classes can be higher than one.

FileClassifier. Based on the class of the repository one will likely find different types of files in different repositories. A project of the class **DEV** is more likely to have files of the type *.cpp* (C++ source code), *.h* (C/C++ header) or *.java* (JAVA source code) while a repository of the **DOCS** class is more likely to have files of the type *.md* (Markdown) or *.pdf* (document format). Often, the file type is associated to the filename extension. The FileClassifer exploits this for classification. While learning, the FileClassifier monitors the distribution of extensions on the different classes (ignoring extensions which only occur once) as shown in (2). For classification, the probability of all known filename extensions in a repository will be averaged over all files as shown in (3).

$$P(\text{class}|\text{extension}) = \frac{N_{\text{extension in class}}}{N_{\text{extension in all classes}}} \tag{2}$$

$$P(\text{class}) = \frac{\sum^{\text{extensions}} P(\text{class}|\text{extension})}{N_{\text{files}}} \tag{3}$$

ReadmeClassifier. A lot of information about a project can be found in the self description which is usually found as a *README* file. Based on this description it is often possible to correctly classify the repository. To analyse README files we use a *Bag-of-words* representation followed by a classification using the *k-Nearest Neighbor* algorithm.

A *Bag-of-words* [12] is a special representation of a text where only the appearance of a word has a meaning but not the context in which the word appears. Although the context of the text is lost in this representation, it is still possible to get a lot of information out of it. In our case we collected words in all README files encountered during learning phase which only consist of letters and numbers (independent of capitalisation). All words that only occur once are removed. Based on the remaining words a list is created. During classification, every occurring word in the README is set to 1, all other to 0. For every README file one of these lists is created. These will then be used by the *k-Nearest Neighbor* algorithm [5] (using the Jaccard distance [2]). The probability of a class is equal to the distribution of classes with the smallest distance. To classify a new repository a Bag-of-words is created for the README file. After that the neighbourhood will be calculated. Based on the neighbourhood the probability of the classes is calculated.

MetadataClassifier. Another source for the classification of repositories is their meta data. *GitHub* provides an API to get a wide variety of meta data for repositories. For the creation of a *weak classifier* we chose the following meta data:

- Information whether the repository is a *fork*
- Information whether the repository has a website
- Size of repository
- Number of *stargazers* (equivalent to *likes*)

- Number of watchers
- Information whether the repository has a *wiki*
- Information whether the repository has *pages* (website hosted by *GitHub*)
- Number of *forks*
- Number of *bugs*
- Number of *subscribers*

These meta data is then used in a decision tree [5].

LanguageClassifier. An evidence of the class of a repository is the used programming language. *GitHub* provides a way to ask for the most used programming language in a repository. The LanguageClassifier uses the language returned by *GitHub* to calculate the probability of the different classes for a repository. This allows a broad classification of many repositories. While learning, the classifier observes the languages used for the classes and calculates the probability as shown in (4). The probability for a language not observed during training is $P(class|unknown\ language) = 0$.

$$P(\text{class}|\text{languange}) = \frac{N_{\text{class with language}}}{N_{\text{language}}} \tag{4}$$

LanguageDetailsClassifier. The LanguageDetailsClassifier is based on the same idea as the LanguageClassifier. However, it uses the percentage distribution of programming languages in a repository instead of the main language (based on file size). The *GitHub* API is used to get the distribution of programming languages. Based on this, a decision tree [5] is build which is used for classification. Because of this, the LanguageDetailsClassifier has a more detailed basis but looses some generalisation in comparison to the LanguageClassifier.

NameClassifier. Although there is a huge variety in names for repositories, there seems to be some common patterns found in the names of repositories of the different classes. For example, one finds many repositories with words like 'dataset', 'list' or 'challenge' in their name in the **DATA** class. This can be exploited for classification. For this, we use the *k-Nearest Neighbor* algorithm [5] with the Levenshtein distance (cost 1 for replacement) [4] to calculate the distance between names.

CommitMessageClassifier. Whenever someone changes something in a repository (a so called *Commit*) there must be a description of that change. These descriptions often hold information which can be used for classification. As an example, a description 'Solution exercise 2' will hint at the **HW** class.

We use the same method here as for the *ReadmeClassifier*: All messages are put into a *Bag-of-word* [12]. The probability of a class is calculated using the *k-Nearest Neighbor* algorithm [5] using the Jaccard distance [2].

RepositoryStructureClassifier. Often, the structure of a repository gives evidence for the class of the repository. For example, if two repositories contain a folder named 'lab2' and one has the class **HW**, it is very likely that the second one also belongs to the same class.

RepositoryStructureClassifier exploits this for classification. It uses a similar algorithm compared to the *ReadmeClassifier*: The structure of a repository (consisting of paths of files, folders and similar) is converted to a *Bag-of-words* [12], which is then classified using the *k-Nearest Neighbor* algorithm [5] using the Jaccard distance [2].

4.2 Implementation

We implemented *ClassifyHub* in Python using scikit-learn [9] for many machine learning algorithms. The implementation has a high degree of parallelisation because all *weak classifiers* can run independently. To show the internals of our algorithm we implemented a user interface in Qt / PyQt5.

5 Data Set

For training and evaluation purpose we created a data set containing 681 repositories of all 7 classes. We focused on an almost equal distribution over all classes to prevent an overfitting to one single class. The data set contains repositories with a wide variety to match the variety of repositories found on *GitHub*, which were picked at random and classified by hand. This includes not using the main repository all the time but also *forks* which sometimes do not get updated. The distribution of classes in the data set can be found in Table 1.

Table 1. Distribution of classes in our data set

Class	**Number repositories**
DEV	127
HW	96
EDU	74
DOCS	77
WEB	95
DATA	86
OTHER	126
Sum	681

Table 2. Average results of a 10-fold cross-validation

Target class	*Precision*	*Recall*
DEV	0.5474	0.7189
HW	0.4933	0.5311
EDU	0.5299	0.3534
DOCS	0.6314	0.4192
WEB	0.6744	0.8051
DATA	0.7681	0.7178
OTHER	0.5484	0.5430
Average	0.5990	0.5841

6 Results and Discussion

We performed a *10-fold cross-validation* on our data set, which should give us a good overview (with slightly negative tendency) over the performance of our algorithm [7]. The results are shown in Table 2. With the combination of multiple *weak classifier* we were able to achieve both high precision (59.90%) as well as high recall (58.41%). This is a good result especially because the *GitHub Classification Problem* can be considered as a hard task due to the high variety of repositories (even within a class). In addition, the distinction between the different classes is often ambiguous, even for humans (e.g. the difference between source code for homework and normal software projects). This might lead to classifications which could be considered correctly by humans, but do not correspond to the labels in the data set.

Both, high precision and recall, is achieved over all classes (with minor differences). This is useful because based on the future application both high precision and recall might be needed:

– A high *precision* might be important if, for example, the classification will be used to improve search results because a human often does not want to look through many wrong results first.
– A high *recall* might be important e.g. for automatic recommendation (like in *Recommender Systems*) to show a high variety of results. A single wrong classification has less effect here, because there is no active search which would be interrupted.

7 Conclusion

In this paper we presented *ClassifyHub*[1], an algorithm which tackles the *GitHub Classification Problem* with high precision (59.90%) and high recall (58.41%). This is achieved through the usage of *Ensemble Learning*, which combines multiple *weak classifier* to a single *strong classifier*. In addition, we provide a data set[2] with 681 classified *GitHub* repositories which can be used for further research.

Acknowledgements. We would like to thank the organisers, the jury and all participants of the InformatiCup 2017, for which *ClassifyHub* was developed.

References

1. Adomavicius, G., Tuzhilin, A.: Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. IEEE Transactions on Knowledge and Data Engineering 17(6), 734–749 (June 2005)
2. Cha, S.H.: Comprehensive survey on distance/similarity measures between probability density functions 1(4), 300–307 (2007)
3. Gousios, G., Vasilescu, B., Serebrenik, A., Zaidman, A.: Lean GHTorrent: Github data on demand. In: Proceedings of the 11th Working Conference on Mining Software Repositories. pp. 384–387. MSR 2014, ACM, New York, NY, USA (2014), `http://doi.acm.org/10.1145/2597073.2597126`

[1] `https://github.com/Top-Ranger/ClassifyHub`

[2] `https://github.com/Top-Ranger/ClassifyHub-data`

4. Jurafsky, D., Martin, J.H.: Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition. Prentice Hall, second edn. (2009)
5. Kantardzic, M.: Data Mining: Concepts, Models, Methods and Algorithms. John Wiley & Sons, Inc., 2nd edn. (2011)
6. Kawaguchi, S., Garg, P.K., Matsushita, M., Inoue, K.: Mudablue: An automatic categorization system for open source repositories. Journal of Systems and Software 79(7), 939–953 (2006), http://www.sciencedirect.com/science/article/pii/S0164121205001822, selected papers from the 11th Asia Pacific Software Engineering Conference (APSEC2004)
7. Kohavi, R.: A study of cross-validation and bootstrap for accuracy estimation and model selection. In: Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2. pp. 1137–1143. IJCAI'95, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1995), http://dl.acm.org/citation.cfm?id=1643031.1643047
8. Maskeri, G., Sarkar, S., Heafield, K.: Mining business topics in source code using latent dirichlet allocation. In: Proceedings of the 1st India Software Engineering Conference. pp. 113–120. ISEC '08, ACM, New York, NY, USA (2008), http://doi.acm.org/10.1145/1342211.1342234
9. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, É.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12, 2825–2830 (2011)
10. Ray, B., Posnett, D., Filkov, V., Devanbu, P.: A large scale study of programming languages and code quality in github. In: Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. pp. 155–165. FSE 2014, ACM, New York, NY, USA (2014), http://doi.acm.org/10.1145/2635868.2635922
11. Ricci, F., Rokach, L., Shapira, B.: Introduction to Recommender Systems Handbook, pp. 1–35. Springer US, Boston, MA (2011), http://dx.doi.org/10.1007/978-0-387-85820-3_1
12. Salton, G., McGill, M.J.: Introduction to Modern Information Retrieval. McGraw-Hill, Inc., New York, NY, USA (1986)
13. Seni, G., Elder, J.F.: Ensemble methods in data mining: improving accuracy through combining predictions. Synthesis Lectures on Data Mining and Knowledge Discovery 2(1), 1–126 (2010)
14. Tsay, J., Dabbish, L., Herbsleb, J.: Influence of social and technical factors for evaluating contribution in github. In: Proceedings of the 36th International Conference on Software Engineering. pp. 356–366. ICSE 2014, ACM, New York, NY, USA (2014), http://doi.acm.org/10.1145/2568225.2568315
15. Ugurel, S., Krovetz, R., Giles, C.L.: What's the code?: Automatic classification of source code archives. In: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 632–638. KDD '02, ACM, New York, NY, USA (2002), http://doi.acm.org/10.1145/775047.775141