Github Repository Classification

Marvin Bornstein
marvin.bornstein@student.hpi.de
(mail@mbornstein.de)

Carsten Walther
carsten.walther@student.hpi.de

March 30, 2017

1 Problem and model selection

The problem was described as a classification problem and we were given a few repositories with their respective labels. With these few data points, we thought of making it an unsupervised learning problem. The idea was to use a clustering algorithm, like *kMeans*, the try to label each cluster using the 5 given data points for each class. This approach did not keep up with the performance of the supervised learning algorithms when we received additional data, though. In the end, it achieved an accuracy of around 57% with a standard deviation of 2.5% in comparison to 70% accuracy of our voting classifier (see section 4) with a standard deviation of 0.5%.

2 metrics

In this section we present the metrics we use for repository classification. We extract them using the Github API and the cloned repository. Because the API calls are limited and cloning a repository is causing a lot of network traffic, we build an architecture that saves the repository to the file system and caches metrics that have been calculated already.

2.1 defining metrics

In order to define the features easily and access the cache, we used python *decorators*, *annotations* and some meta programming magic. Figure 1 shows how we can add metrics as features to our data. All features are stored per repository in a *json* file, which is only updated for new metrics. We consider this a valuable architecture for an iterative and explorative data science work flow. It enabled us to test our model quickly, analyze the confusion matrix and add crucial metrics easily at one location.

```python
from metrics.caching import CachedMetric

@CachedMetric
def forks_count(repo: 'repo_overview'):
    return repo.forks_count

@CachedMetric
def intro_or_course_in_description_or_title(repo: 'repo_overview'):
    terms = ['intro', 'course']
    return sum(repo.description.lower().count(term) for term in terms if repo.description) + \
           sum(repo.name.lower().count(term) for term in terms)

@CachedMetric
def hw_terminology_commits(repo_path: 'cloned_repo_path'):
    common_terms = ['exercise', 'assignment', 'question', 'task', 'homework', 'student', 'solution']

    def git_list_commit_messages():
        return subprocess.check_output('git log --format="\%s" | tee', shell=True)
                .decode('utf-8', errors='ignore')

    commit_messages = execute_in_dir(git_list_commit_messages, repo_path)
    commit_messages = commit_messages.lower()
    return sum(commit_messages.count(term) for term in common_terms)
```

Figure 1: Example on how we define metrics for a repository.

2.2 metrics details

First, we thought about what characteristics repositories in a specific class might have. Initially, we came up with lots of predictors for *DEV* repositories, since there is lots of stuff to look for, such as the scripting or programing language, the availability of tests a makefile etc. However, it turned out, that the majority of repositories are *DEV* repositories, because that is what Github is usually used for. That is why, we needed to distinguish *DEV* repos from the rest.

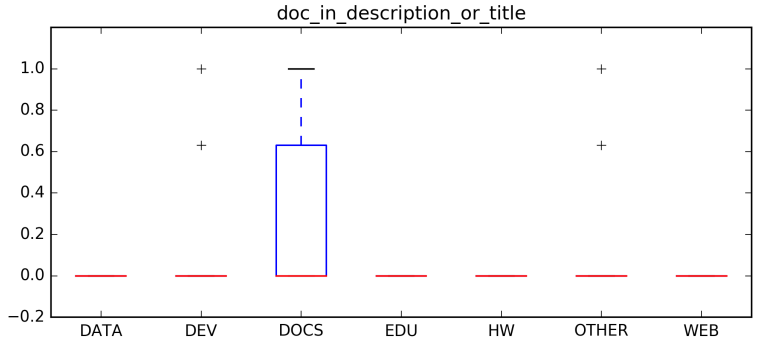
For all, but *avg entropy*, *up-to-dateness* and *education e-mail ratio*, we applied the natural logarithm to weaken the *exponential distribution*. That made sense, because for most of the metrics, the difference between 1 and 5 is more interesting than a difference between 1000 and 1100.

Also all metrics are scaled to be between 0 and 1 (that does not effect the random forest, but improves the results for the neural networks a lot).

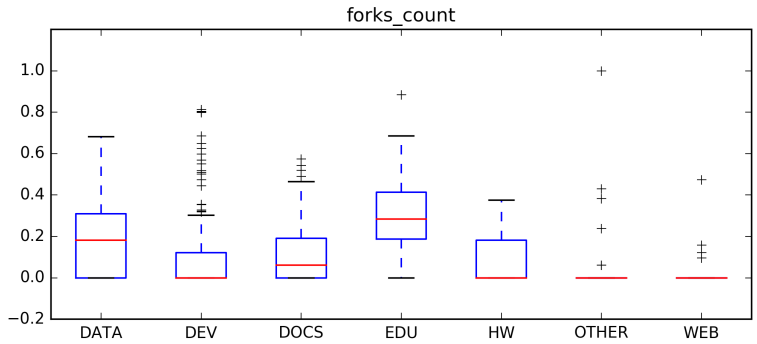
In oder to estimate how good the features can separate the classes, we plotted box plots containing each class.

2.2.1 API repo overview

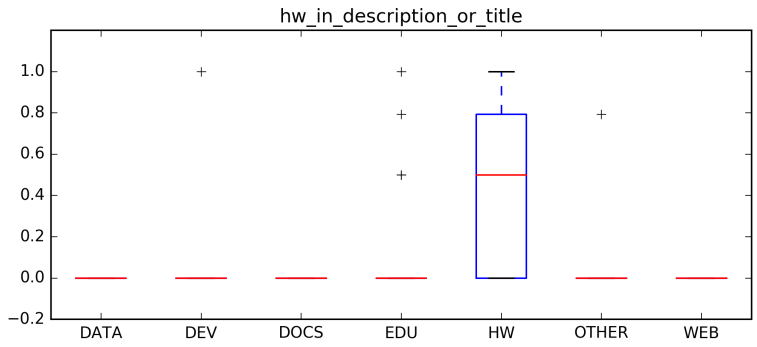
DOCS terms in repository title or description We search for the term 'docs', 'documents' or 'documentation' in the repository title or description. This is a strong indicator if it is a *DOCS* repository.



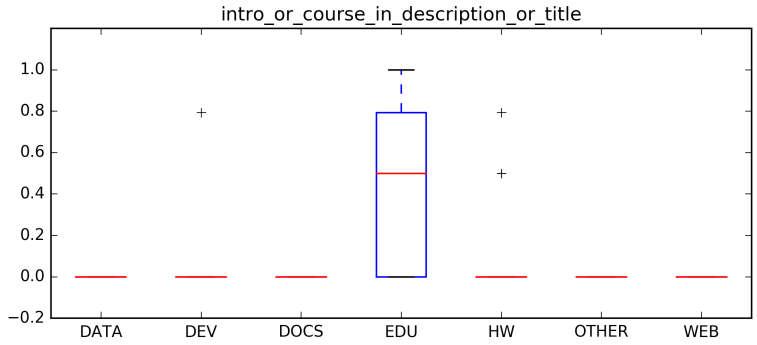
Forks count This metric contains the number of forks for a repository.



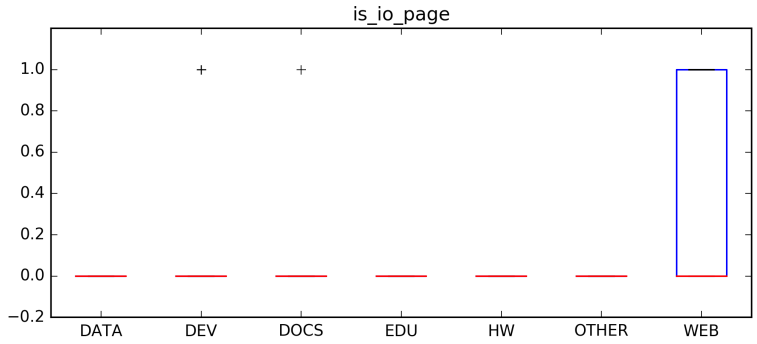
Homework terms in description or title The terms 'homework' and 'assignment' in the repo description or title are strong indicators for a *HW* repo.



Intro or course in title or description This metric searches for the terms 'intro' and 'course' in the repository title and description. If that is the case that is a strong indicator for a *EDU* repository.



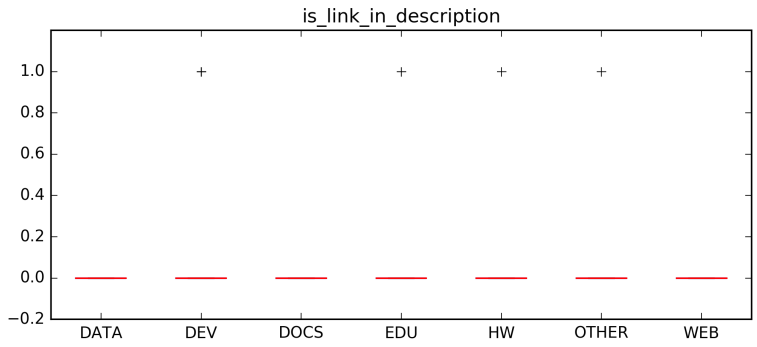
Is github.io page Github host websites via Github pages. There are dedicated repositories with the name scheme <username>.github.io. This metrics helps us to identify such repositories as *WEB*.

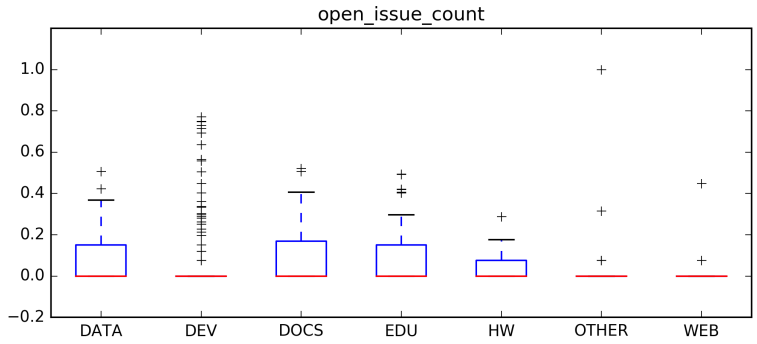


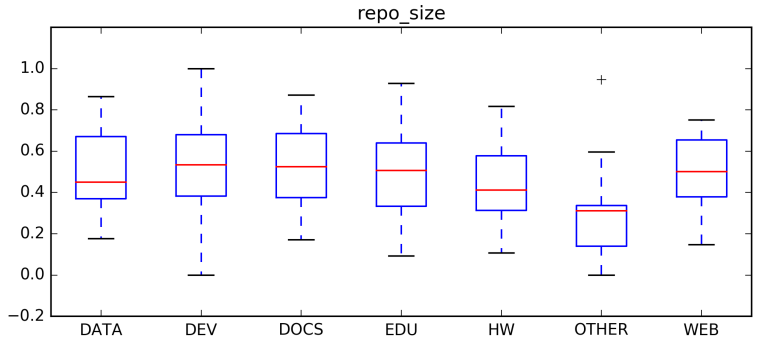
Link in description When looking at false predicted *DOCS* repositories, we discovered, that they had links in the description, so we added this as a feature.

Open issue count The number of currently open issues. Many open issues indicate a popular repository. We expect mostly DEV to have a high open issue count.

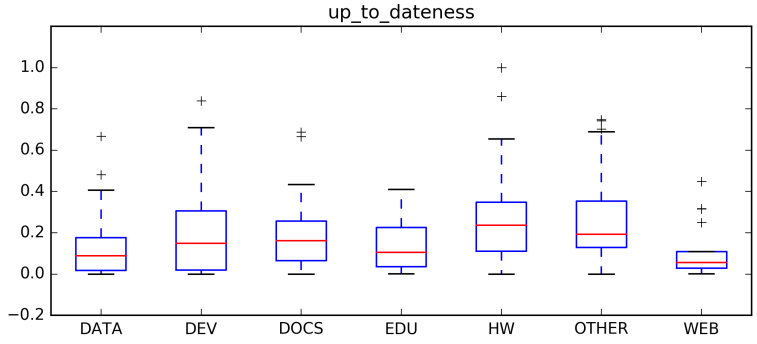
Repository size The repository size in kilobytes. Repositories of each category can have a small size but we only expect *DEV*, *DOCS*, *DATA* and *WEB* repository to have a big size.





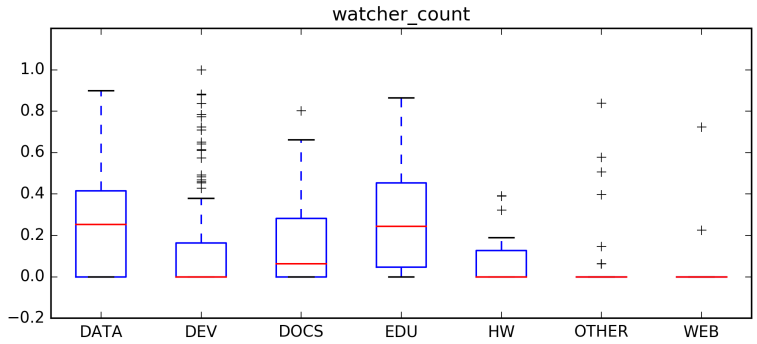


Up-to-dateness Measures the time since the last commit. We expect HW repositories to have a low up-to-dateness because they are not worked on after deadline.



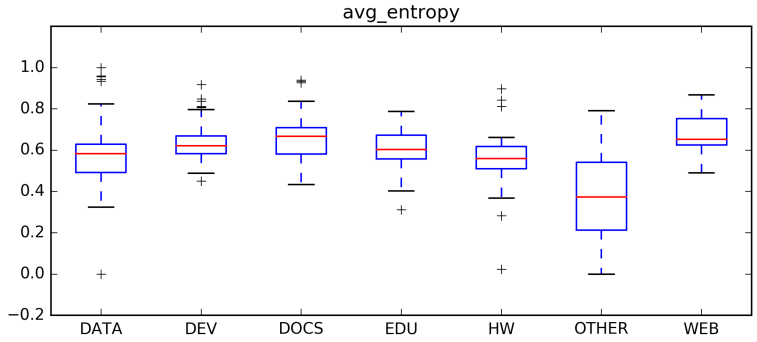
Watcher count The watcher count indicates the popularity of a repository. We expect only *DEV* repositories to have a high watcher count. *EDU* could also have an higher than

average watcher count.

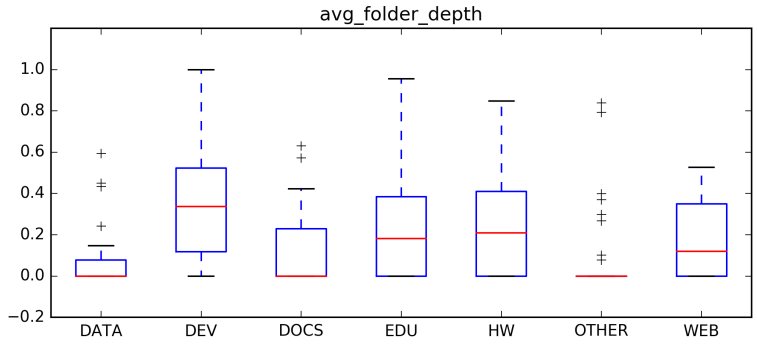


2.2.2 cloned repository

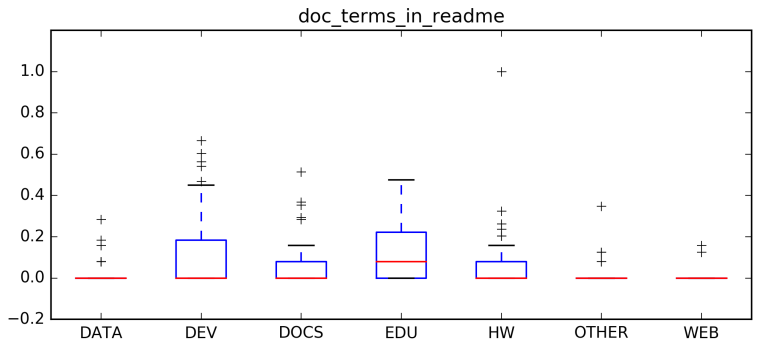
Average entropy The entropy of each repository file is calculated and the average of all these files is returned. We expect *DATA* and *EDU* to have a high entropy because of many pictures and videos. *DOCS* and *WEB* have a medium entropy because of some pictures. *DEV* and *HW* have a low entropy because they contain mostly source code. Is actually helps to predict *OTHER* really well, because of an entropy of 0.3 or lower.



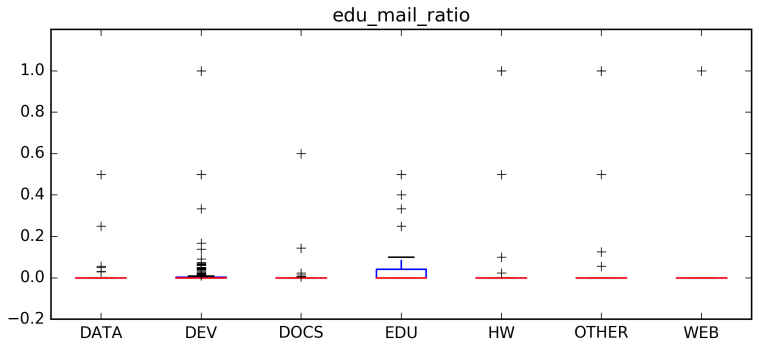
Average folder depth This averages the depth of all folders. We expected Java repos to have a deep folder structure, but *HW* to a flat one.



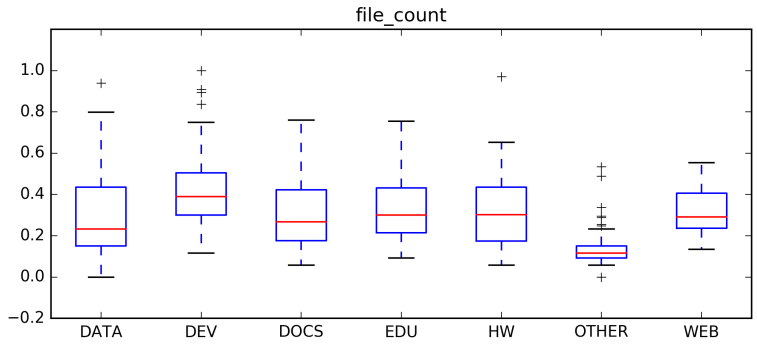
DOC terms in read me Returns the number of common DOC terms in the Readme file. Under common DOC terms we understand: 'documentation', 'usage', 'guide', 'installation', 'getting started', 'quickstart', 'tutorial' and 'setup'.



Education e-mail ratio This is the ration between university mail addresses to all unique addresses that appear in a repository. We extract them from the commit messages. We expect *EDU* to have a high *EDU* mail ratio.



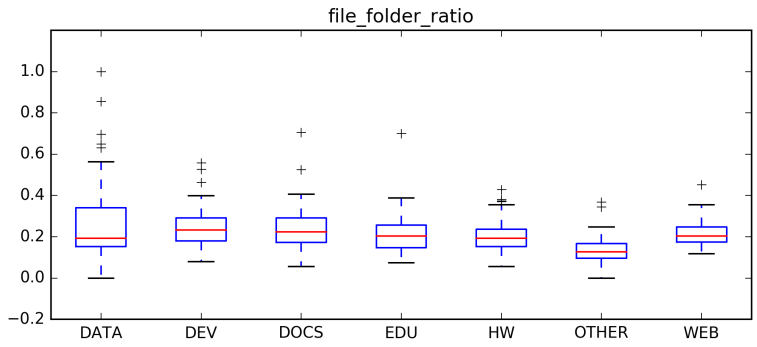
File count This is the number files in the repository excluding the .git directory. Additionally we calculate the ratio of specific file types to the total file count. There are metrics for each of the following file types: pdf, html, png, latex, source code and markdown. We identify these file types by the file extension.

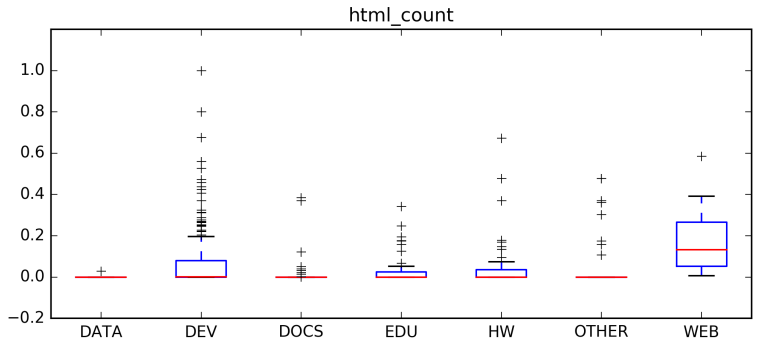


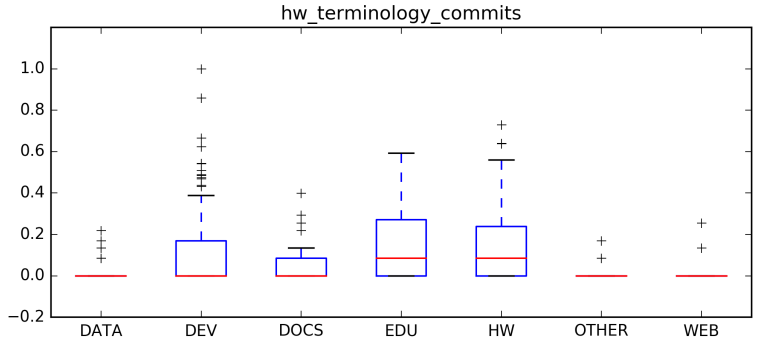
File to folder ratio This calculates the ratio between number of files and directories. We expect the DOCS and some DEV (e.g. java programs) to have a low file-folder-ratio.

HTML count This metrics should identify *WEB* repos

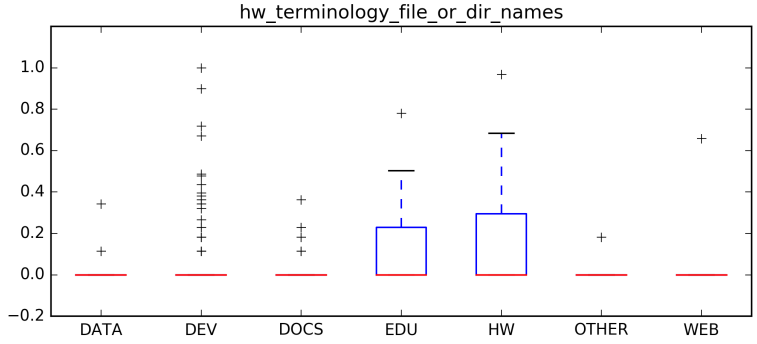
HW terminology in commits We search all commit messages for common HW terms and return the number of occurrences. Under common HW terms we understand: 'exercise', 'assignment', 'question', 'task', 'homework', 'student' and 'solution'



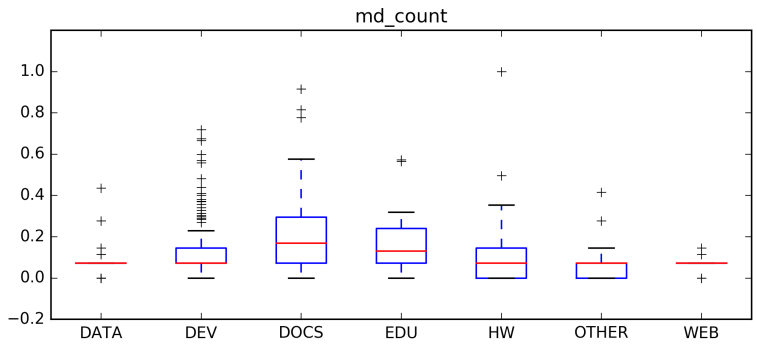




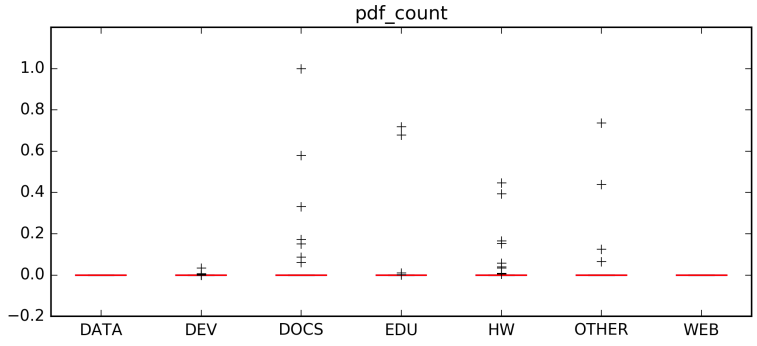
HW terminology in filenames or directory names Like HW terminology in commits but we look for the terms in file names.



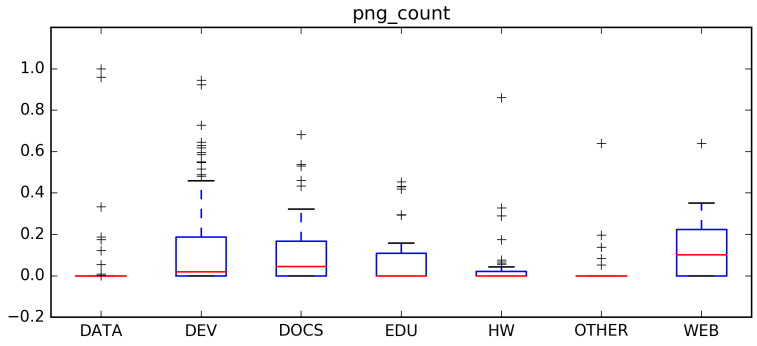
md file count Existence of md files indicates *DOCS* and if more than one exists *WEB*



pdf count Was supposed to identify *DOCS*, but does not seem to be a strong indicator



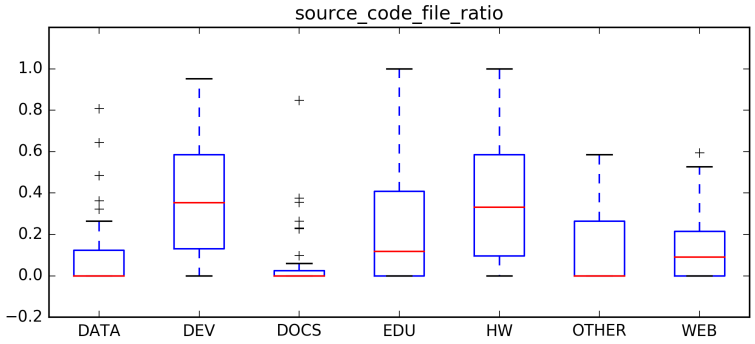
png count We supposed png to be used for screenshots and info graphics rather than for pictures. This may identify *DOCS* or *EDU*



source code file ratio Initially thought to identify *DEV* repos

2.3 Confusion Matrix

In order to engineer features that increase the quality of the results, we looked at the confusion matrix. The final features result in a quite balanced matrix, i.e. *DEV* is still overly predicted, but has more correct predictions in a class than wrong ones (see figure 2).



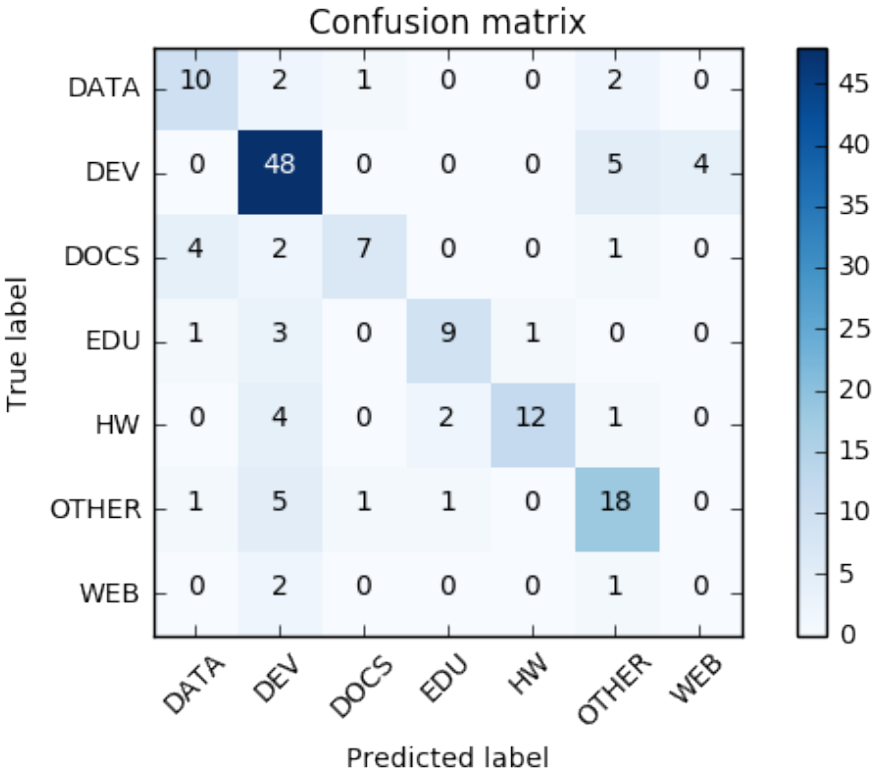


Figure 2: Confusion matrix of our final voting classifier model. 433 repositories with a train/test ration of $\frac{2}{3}/\frac{1}{3}$

3 Model

We came up with a voting classifier that combines:

1. LogisticRegression (one vs. rest)
2. Support Vector Classifier
3. RandomForestClassifier
4. MLPClassifier (100 fully connected hidden units)
5. MLPClassifier (50 x 20 fully connected hidden units)
6. GradientBoostingClassifier

For more details and the parameters we used, please refer to the source code.

When the accuracy of the individual models were about 65%, the accuracy of the voting classifier got to 70% and its confusion matrix was nicely balanced.

4 Validation

The table shows our manual prediction versus the predictions generated by our model. The accuracy of the automatic prediction is 45%. Our accuracy on our full dataset is 70%. The dataset contains of the given repositories from InformatiCup and some manual classified repositories. The complete list can be found in *data/testset.csv*. We used a 3-fold cross-validation to calculate the score in order to use the limited dataset size most efficiently.

Repository	manual pred	automatic pred
https://github.com/ga-chicago/wdi5-homework	HW	HW
https://github.com/Aggregates/MI_HW2	HW	HW
https://github.com/datasciencelabs/2016/	EDU	EDU
https://github.com/githubteacher/intro-november-2015	EDU	EDU
https://github.com/atom/atom	DEV	DEV
https://github.com/jmcglone/jmcglone.github.io	WEB	DEV
https://github.com/hpi-swt2-exercise/java-tdd-challenge	EDU	DEV
https://github.com/alphagov/performanceplatform-documentation	DOCS	DOCS
https://github.com/harvesthq/how-to-walkabout	DOCS	DOCS
https://github.com/vhf/free-programming-books	DATA	DATA
https://github.com/d3/d3	DEV	DATA
https://github.com/carlosmn/CoMa-II	HW	DEV
https://github.com/git/git-scm.com	WEB	DEV
https://github.com/PowerDNS/pdns	DEV	DEV
https://github.com/cmrberry/cs6300-git-practice	HW	OTHER
https://github.com/Sefaria/Sefaria-Project	DEV	DEV
https://github.com/mongodb/docs	DOCS	DEV
https://github.com/sindresorhus/eslint-config-xo	DEV	DATA
https://github.com/e-books/backbone.en.douceur	DATA	DATA
https://github.com/erikflowers/weather-icons	DEV	DATA
https://github.com/tensorflow/tensorflow	DEV	DEV
https://github.com/cs231n/cs231n.github.io	WEB	DEV
https://github.com/m2mtech/smashtag-2015	EDU	DEV
https://github.com/openaddresses/openaddresses	DATA	DEV
https://github.com/benbalter/congressional-districts	DATA	DATA
https://github.com/Chicago/food-inspections-evaluation	DEV	DATA
https://github.com/OpenInstitute/OpenDuka	WEB	DEV
https://github.com/torvalds/linux	DEV	DEV
https://github.com/bhuga/bhuga.net	WEB	DOCS
https://github.com/macloo/just_enough_code	EDU	DEV
https://github.com/hughperkins/howto-jenkins-ssl	EDU	DATA

4.1 Precision and Recall

Category	Precision	Recall
DEV	5/15	5/9
HW	2/2	2/4
EDU	2/2	2/6
DOCS	2/3	2/3
WEB	0/0	0/5
DATA	3/7	3/4
OTHER	0/1	0/0

The category DEV is over-predicted. The reason for this is the high percentage of DEV repositories. The model is with a high probability correct to guess DEV. WEB is under-predicted. This result is unexpected since we have some metrics that mainly focus on WEB like *is Github io page* or *html file ratio*.

It depends on the user if precision or recall is more important. A quick search for Github

repositories requires a high precision. The user want few but most relevant results. On the other hand a researcher might want to have a high recall e.g. if he studies WEB repositories. He wants the most complete result and accepts to re-validate the repositories himself to make sure that they are WEB repositories.

5 Program usage

The entry point of the program is main.py. Running *python3 main.py* starts the test mode. This mode trains and validates different models. When given a file as parameter (e.g. *python3 main.py data/valset_unclassified.txt*), the program classifies all repositories in that file and print the results to stdout.

Saving the trained model with pickle or joblib resulted a strange loss in accuracy when loading it again. That is why we fall to the solution of training the model at the start of the prediction phase.

5.1 3 repos we can classify well

1. *DATA* repositories with lots of pictures, because of a high entropy, e.g. `https://github.com/ColinHite/DGM-2210-Hite-C.-2015/tree/master/images`
2. repositories with relatively high amount of same file types, such as md files, e.g. `https://github.com/e-books/backbone.en.douceur`
3. `https://github.com/ppau/documents`, because we have a metric that checks the title for doc specific terms (this also let us distinguish *DOCS*, *HW* and *EDU* repos)