GitHub Repository Classification

Solution for the informatiCup 2017

Willi Gierke
willi.gierke@student.hpi.de
Hasso-Plattner-Institute

Sebastian Bischoff
sebastian.bischoff@student.hpi.de
Hasso-Plattner-Institute

Potsdam, January 2017

Contents

1. Challenge Description

This years informatiCup challenge was to automatically classify GitHub repositories (repos) based on given class descriptions and sample data from the exercise description. In this work we present how we explored the given data, detected relevant features and built an application that predicts repository labels using different machine learning algorithms.

2. Data Exploration

This section explains how we extended the training data set and how we explored it using different dimension reduction algorithms and visualization tools.

2.1. Data Retrieval

The corresponding repository of the challenge includes 30 labeled repositories and 31 repositories that can be used as validation data. It wouldn't be possible to train convincing prediction models using only these provided data sets. To extend the amount of available training data (and as a first step to reduce overfitting), we used the GitHub Search API[1], GitHub Showcases[2] and automated as well as manual Google[3] searches to retrieve more data. We used manual Google searches only in the beginning as we realized very quickly that this approach is not effective enough on the long run.
One possibility to overcome this challenge was to use repos which have already been labeled by humans. As an example, GitHub Showcases are collections of repos hostet on GitHub that include content about certain topics like "Open Data" or "Web application frameworks". Another example are "awesome" GitHub repos. These projects, like "awesome-machine-learning"[4], contain curated links to further material about their topics. Thus, within the scope of this challenge, they can be classified as DOCS repos. The repository "awesome-awesomeness"[5] references these "awesome" projects. Therefor, by crawling the referenced projects in this document we were able to automatically obtain a lot of DOCS repositories.
Another possibility we used was to utilize the GitHub Search API. After searching for e.g. "course, material" and verifying the obtained results, we labeled the repos as EDU.
GitHub can automatically serve content of repositories at https://*username*.github.io if their name matches the pattern *username*.github.io. That's why a lot of users host their websites on GitHub. For this reason we used Googles "Advanced Search"[6] to search for websites whose domains included ".github.io". This approach allowed us to automatically obtain a large amount of labeled repositories in very little time. One can find the amount of retrieved, labeled repositories and their origin in table A. Overall, we were able to collect 1412 labeled repositories. One can find the distribution of the collection labeled training data in figure 2.

As one can see in table A, we tried to use key words for automated searching that are as close to the words that were used to describe the different classes as possible. Although, it's still possible that the collected training data is biased as we actively selected repositories by

[1] https://developer.github.com/v3/search/
[2] https://github.com/showcases
[3] https://google.com
[4] https://github.com/josephmisiti/awesome-machine-learning
[5] https://github.com/bayandin/awesome-awesomeness
[6] https://www.google.com/advanced_search

searching for them. As an extension, an approach that could minimize this bias would be to randomly select repositories (e.g. from the GHTorrent project) and label them manually. For the beginning, however, we neither had the time nor the manpower to label a large amount of repositories manually. Since the difficulty to collect data entries of a certain label differed, we ended up with unbalanced training data. As the class label distribution affects some classifiers heavily, we trained the models on randomly undersampled training data.

2.2. Data Analysis

To get a better idea of how the relationships between the data entries look like in a higher dimensional space, we used principal component analysis (PCA) and t-distributed stochastic neighbor embedding (t-SNE) to reduce the complexity of the data to 2D while retaining the principal components respectively the distances between the data points. The figure 3 visualizes the distribution of the labeled data entries using t-SNE. You can find the complete code to generate the figure in the t-SNE Visualization Notebook[7]. To explore the data interactively and in a three dimensional reduction you can use the Tensorflow Embedding Projector setup[8]. One can notice that the "DOCS" repositories build a cluster while it seems to be more complicated to separate the other classes.
We also used t-SNE to visualize the similarity between our retrieved training data and the given validation data as you can see in figure 4. Since the validation data does not form separate clusters or outliers, we could assume that testing the learned models on the validation data is a good way to verify how well the models generalize. On the other side, the validation data only contains roughly 30 data entries which is not enough to give reliable statements about the model performances. Furthermore, the fact that the validation data seems to be selected manually implies that it's also biased. Thus, perfect validation data would be a lot of randomly selected repositories that have been labeled manually. The additional data sets[9] from another team allowed us to validate our models better although they were also biased. As already mentioned, a perfect training and validation set would only contain repositories that have been sampled randomly and labeled manually.

3. Prediction Model

When a model fits its training data too well and doesn't learn to generalize, it's considered to "overfit". This can occur if the model is too complex so it learns the training data by heart instead of understanding how to solve the given problem in general. To prevent this, we collected more training data than we already received by the challenge, we applied regularization to hinder the model becoming too complex and we used ensemble learning.
By collecting more data than already given we created a bigger problem domain that needs to be understood by the model.
Regularization adds a measure of the models complexity to the cost function that needs to be optimized. Thus, the model does not only try to solve the given problem but it also tries to do that keeping itself as simple as possible.
Ensemble learning uses multiple trained models to calculate one final prediction. These models

[7] https://github.com/WGierke/git_better/blob/master/t-SNE%20Visualization.ipynb
[8] https://github.com/WGierke/git_better#usage
[9] https://github.com/InformatiCup/InformatiCup2017/tree/master/additional_data_sets

are trained using distinct features so they're not related to each other. The assumption is that when one model makes a mistake predicting a label, the other models don't make a mistake in this situation so the (correct predicted) label of the other models is returned. To decide which model prediction is the correct one we used the Majority Rule algorithm. One model was trained on the numerical features of a repository, one on the description, one on the content of the readme and one on the source code of each repository.
The following chapters will explain how we retrieved and cleaned the data for each model, how we selected relevant features and how we developed the prediction model.

3.1. Training and Test Data Set

To train and evaluate the classifiers, we used a train/test/validation split.
First, the collected training data was splitted in a train and a test split in a stratified manner. This ensured that the distribution of class labels was balanced in both splits.
The classifiers were then trained on the train split and their accuracy was evaluated on the test split. To calculate their final quality, we evaluated them on the validation data.

3.2. Classification Using Numeric Metadata of Repositories

To develop classifiers based on numeric metadata of repositories, we used the features explained in table 6. Most of the features were available using the GitHub API. We added the *isOwnerHomepage* and *hasHomepage* features to detect whether a repository serves its source code using GitHub pages. This could allow us to identify WEB repositories easier. We furthermore hoped that using *hasCiConfig*, so whether a repo contains a configuration file for a Continuous Integration service like Travis CI[10] or CircleCI[11], would improve the accuracy of detecting DEV repositories.

3.2.1. Data Cleaning and Preprocessing

Using the GitHub REST API and the GitHub GraphQL API, we were able to receive all features without extensive cleaning or preprocessing of the data.

3.2.2. Feature Selection

Feature Selection describes the process of removing features that yield no or very little additional information in order to decrease overfitting and accelerate model fitting. Especially the programming language features needed to be reduced using Feature Selection.
GitHub detects over 300 used programming languages[12] in repositories. The problem is that a lot of them are used only in a few repositories such that there are a lot of features that only hold very little variance and information. As an example, among the collected 1400 repositories there were 46 programming languages, like Pony or KiCad, that were only used in one repository at all.
In the beginning of the project, we dropped features with low standard deviations and a low overall sum according to chosen thresholds. While this statistical approach is very fast, it

[10] https://travis-ci.org/
[11] https://circleci.com/
[12] https://github.com/github/linguist/blob/master/lib/linguist/languages.yml

didn't improve the accuracy a lot. Instead, we used extremely randomized trees[13] to compute the feature importances. The earlier a feature is used in a decision tree to split the data by a threshold, the more important is the feature. Intuitively, the closer a feature is to the root of a decision tree, the higher is its importance score. Figure 5 visualizes that only approximately 30 of the 172 features of the training data were important for predictions. Table 7 shows the top 20 features and their importance scores. One can note that it's important whether a repository serves a website or even serves the owners homepage. This is very comprehensible as very few projects containing data, documents or homework serve their own websites. It's also remarkable that the feature *projects* feature is not important enough to occur among the top 20 features. This might be due to the fact that this project management tool was introduced late 2016 so there aren't so many project teams already using it, yet.

3.2.3. Feature Engineering

In a next step, we derived further features from the features we already collected. We used polynomial feature generation which takes the input variables and builds all possible polynomial combination of this features up to a given degree. The idea of taking input features and applying a non-linear method on it to map the original values in another space is called "kernel trick" and is used by Support Vector Machines (SVM) to learn non-linear models as well.
As an example, suppose a dataset is given with the two features *size* and *watchers* as in table 3.2.3.

Table 1: Original features

size	watchers
2	5
10	8

The transformed dataset using polynomial features with a degree up to 2 would look like table 3.2.3.

Table 2: Polynomially generated features (degree=2)

$size^1$	$watchers^1$	$size^1 \cdot watchers^1$	$size^2$	$watchers^2$
2	5	10	4	25
10	8	80	100	64

As one can see, the number of generated features increases polynomially in the number of input features. That's why the previous Feature Selection step was very important.
As an alternative we could have used deep learning techniques but one needs many training samples because of their higher learning complexity. Our roughly 1500 samples aren't enough for this. Small feed-forward neural networks are applicable to our problem while deep neural networks are not.

[13] Geurts et al., "Extremely randomized trees", Machine Learning, 63(1), 3-42, 2006

3.2.4. Numeric Metadata Prediction Model

In a next step, we applied Naives Bayes, Decision Trees, Random Forests, k-Nearest Neighbours (k-NN), Support Vector Machines (SVM) as well as Gradient Boost classifiers on the numeric features. We also ensembled the classifiers using a weighted voting approach. You can find their average accuracies scores on the official validation data and on the additional validation data provided by another team from 100 run times in table A. One can notice that all classifiers performed much better on the official validation data set than on the additional one. The ensemble classifier outperformed all other classifiers which implies that it was able to generalize better and that the ensemble approach worked.

3.2.5. Validation of Prediction Model

After the classifier was applied on a random undersampled set of our training data for 100 times, it achieved accuracies of 39,3% and 26.7% on the validation respectively additional validation data on average. Figure 6 visualizes one confusion matrix of the ensemble and table 9 shows the according boolean matrix.

3.3. Classification Using Text Data (Description and Readme)

Intuitively, one wouldn't use the numeric features like the number of branches etc. to decide what label fits the repository best. Instead, one would use the description or the content of the readme to determine it. For this reason we used term frequency–inverse document frequency (tf-idf) matrices to develop natural language processing (NLP) models that predict the label based on them. Since there's a semantic difference between the description and the readme of a repository, we discarded the idea of concatenating the text features and training one model on it. Instead, we trained two seperate models on the description respectively readme of the repositories.

3.3.1. Data Cleaning and Preprocessing

To remove words like 'the', 'a', 'and' etc. that occur very often and yield little meaning, we used the Natural Language Toolkit (NLTK) to drop English stopwords. Since it's also not important whether the singular or the plural of words are used, we also used this toolkit to stem English words.

3.3.2. Feature Generation from Existing Data

We used a count vectorizer which converts a text into a n-dimensional vector representing the vocabulary, where n is the number of unique words. After this text-to-vector conversion we transformed the vector into a tf-idf vector which is a normalized representation of the original vector. Thus, the texts are now merged into a matrix which holds as many features as there are unique words in the texts.

3.3.3. Feature Selection

To reduce overfitting and accelerate model fitting, it's possible to drop very frequent words or words that occur very rarely. In our example, we dropped words with a document frequency of under 0.01 and over 0.9.

3.3.4. Prediction Model

We chose to use the stochastic gradient descent (SGD) algorithm to apply it on the feature matrix. The reason is that SGD can work efficiently with sparse features which is why it has been successfully applied to further NLP problems before.

3.3.5. Validation of Prediction Model

As one can see in Figures 7 and 8, the description and readme classifiers outperformed the numeric ensemble classifier with 51.6% respectively 48.4% accuracy.

This is especially remarkable if one keeps in mind that both classifiers only use one feature and that the *description* feature only contains very few words in comparison to the large readme file of a repository.

3.4. Classification Using Source Code

We tested different approaches to use the source code and connected data of a repository to classify it. Data from the repositories are including source code files with comments and git workflow specific data (branches, commits...). In this chapter we mainly used the source code, file names, commit messages and the wiki pages.

3.4.1. Data Cleaning and Preprocessing

We cloned each repository and its wiki which is also a git repository locally in order to retrieve the data we need. After this step we were able to merge all non-binary source code files, all filenames, all git commit messages and all wiki pages into four different files. We didn't filter based on languages and all UTF-8 files are included. This could be an additional preprocessing step to improve, correct and simplify the stemming and classification.

3.4.2. Feature Generation from Existing Data

We were able to use the same feature generation approach based on the count vectorizer and tf-idf vector as used in the text data classification. Ugurel et al.[14] showed a similar approach successfully.

3.4.3. Prediction Model

Shortened words in the following tables are caused by the stemming of the input data.

In the following paragraph we will discuss the importance of features gained from the file names, see B.
As one can see in table 22 an obvious good indicator for the DATA category is the file ending "json". Negative Words are mainly connected with development (e.g. "package", "test" and "main") and web (e.g. "css" and "html").
Good features for the DEV category are "util", "package", "yml" and "test". This corresponds to the heavily used category or file name "util" (for utility) among developers and "test" for software tests like unit tests.

[14] "What's the Code? Automatic Classification of Source Code Archives", KDD, 2002

The strong positive word "contributing" for the DOCS category comes from the CONTRIBUTING.md file, which explains the principles of contributing to open source projects to new users. PDF files are an indication for EDU repositories, where the other features are very similar between EDU and HW.
The category WEB has no positive words because we used the complete data shown in the table 2 and didn't undersample.
So the classifier classifies each sample as WEB when no negative word occurs.

Commit messages provide an additional insightful source for our predictions which one can see in E.
The most relevant word in the category DATA is trivially "data". "json" and "csv" are common filetypes used to store the data which are probably retrieved using the "script[s]". "Merge" under the highest negative words indicates that people only linearly add files to their repository and don't cause merge conflicts.
"when" in the category DEV probably stands for conditional expressions like "Add second loop when flag is set". The other words are typical expressions in this context "test", "option", "support", "fix" and "compile". The negative words show that developers do not use the typical beginning of commit messages like "add", "update", "change" and "commit" as proposed by Chris Beams [15] and others.
Apart from "section", "article" and "book" nearly all other positive words in the DOCS category are git workflow specific phrases. The negative words separate it from the WEB category ("html", "index", "page" and "post") and from the EDU and HW category ("solution" and "slide").
Almost all of the first eleven positive words are very logical for the EDU category ("slide", "material", "note"...).
The HW category on ther other side has very special words for its domain like "solution", "assignment" and "work".
Good indicator for the WEB category are the words "post", "page", "html", "index", "site" and "blog" which are all obviously connected with this class.

We trained two different source code classifier but we will describe only the smaller and more general model (Appendix G).
The DATA category is dominated by numbers which we could read from non-binary files. The negative words separate DATA from DEV and DOCS mainly.
The DEV category is a mixture of different programming languages which are not part of the WEB category. Keywords of object oriented programming languages are dominant in DEV. In contrast HTML and CSS is recognizable in WEB.
The categories DOCS, EDU and HW are structureless with some typical words from the previous classifiers.

3.4.4. Validation of Prediction Model

Our classifier based on the cloned repositories performed well as one can see in the table 3. We trained the commit message classifier with `min_df=0.1` and `max_df=0.9`, the file name classifier with `min_df=0.1` and `max_df=0.9`, the small source code classifier with `min_df=0.3` and

[15] http://chris.beams.io/posts/git-commit/#imperative

`max_df=0.8` and the big source code classifier with `max_df=0.8`.
`max_df` specifies the maximum document frequency, `min_df` the minimum document frequency.[16] A higher `min_df` leads to fewer rare words and a better generalization, a lower `max_df` excludes words which are present in many documents and doesn't hold much information.

One can find the boolean matrix for the file name classifier on the validation data in the table 28 and on the additional validation data in the table 29.
One can find the boolean matrix for the commit message classifier on the validation data in the table 36 and on the additional validation data in the table 37.
One can find the boolean matrix for the source code classifier on the validation data in the table 49 and on the additional validation data in the table 50.

Table 3: Accuracy of different classifier

	validation	additional validation
commit message	38.7%	38.3%
file name	48.4%	29.3%
source code (1600 features)	38.7%	26.3%
source code (30 million features)	51.6%	34.3%

3.5. Overall Prediction Model

The different approaches to classify the repositories based on the source code achieved even better results than the description/readme classifiers. Though, the current state of the prediction using the source code is not ready for production use so far as it required us to download the full repository code, all commit messages and the complete wikis. Overall, this required us to download over 200 GB of data for the training data which took over 24 hours due to the slow Git cloning protocol. For this reasons we decided to ensemble the numeric ensemble classifier and the description/readme classifiers using weighted voting as we already did it with the classifiers on the numeric features.
Figure 9 visualizes our used *meta-ensemble classifier*. Due to performance reasons we're not using classifiers in production that require to clone repository content. In the beginning, we're fetching relevant features using the GitHub REST and GraphQL API. We're also adding our own calculated features such as *hasHomepage*, *isOwnerHomepage*, *hasCiConfig* etc. We then apply a set of classifiers on the numeric features and ensemble them using weighted voting. The content of the *description* and *readme* features are then transformed to a tf-idf matrix. A SGD classifier the learns to predict the labels based on them. In the end, the numeric ensemble classifier, the description classifier and the readme classifier are then ensembled using weighted voting.

[16] http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

4. Implemented Application

4.1. Tooling Choices

Python and R are the most used programming languages to solve data science tasks. While R was specifically developed for statistical computing and data visualization, Python is a general purpose programming language which was designed to be easy to understand. Libraries like NumPy, SciPy and scikit-learn are closing the gap between the machine learning capabilities of R and Python. The main reason for us to prefer Python to R was that we wanted to build a web application that makes use of our developed models and insights. Furthermore, we were already familiar with Pythons data science stack[17] as we already completed machine learning courses of our university using it. The Python libraries we used the most were pandas[18], scikit-learn[19] and Jupyter[20].
Pandas transfers the concept of a *DataFrame* from the R language to Python. A *DataFrame* is a matrix-like object whose columns and rows are identified by a name respectively an index. This allows the user to always be able to understand the content of the object in opposition to matrices as *DataFrames* always also provide semantic metadata about the content they're holding. Scikit-learn is a powerful machine learning library that provides cutting edge data science algorithms. Using scikit-learns pipelines, it's possible to chain data preprocessing, model fitting and prediction easily. Thus, it's possible for the user to completely focus on the challenge to solve and not to worry about implementation details of the used algorithms. Due to the compatible Python data science stack, scikit-learn supports the usage of *DataFrames*.
Jupyter is an interactive web application that makes it possible to cache computation results and to persist code and visualizations in separate files called *notebooks*. Therefor, it's very simple to try out and visualize new solution approaches as well as to share them with collaborators.
To build a service using our learned models we used Django[21]. Django is a high-level framework to rapidly build web applications which supports the easy integration of further plugins. We needed that to make it possible for the user to log in using their GitHub account.
To deploy our built service we used Heroku[22]. Heroku is a cloud-based Platform as a Service that allows one to deploy server applications in various supported languages. Its free tier makes it possible to host basic servers with database access without additional costs.

4.2. Functionality

Our implemented application takes a file containing GitHub repository URLs, classifies them using an ensemble model that's trained on passed training data and saves the URLs and their computed labels on the disk. If no training data is given, the input data will be classified using our pre-trained model. It's possible to pass the input data, which is supposed to have the format of the challenge example[23], using the `-i` argument. Optional training data can be passed using the `-t` argument.
As an example, to classify the example data given by the challenge using the training data

[17] https://speakerdeck.com/jakevdp/pythons-data-science-stack-jsm-2016
[18] pandas.pydata.org/
[19] http://scikit-learn.org/
[20] https://jupyter.org/
[21] https://djangoproject.com/
[22] https://heroku.com
[23] https://github.com/InformatiCup/InformatiCup2017/blob/master/example-input

given by the challenge one would run:

```
$ python app/main.py -i data/example-input.txt
                     -t data/training_data_small.csv
                     -l 10
```

The script would load all features for the training data *training_data_small.csv* and the input data *example-input.txt*. It would then train 10 *meta-ensemble classifiers* on the training data and it would use the one with the best average score on the validation sets to predict the input data. The saved output file `predictions.txt` will have the format of the challenge example[24].
For setup instructions please refer to the README.md file.

5. Validation

In this chapter we want to evaluate the performance of our chosen *meta-ensemble classifier*. After 100 times being applied on a random undersampled set of our training data, the classifier achieved accuracies of 61,3% and 47.7% on the validation respectively additional validation data. The best classifier we were able to train achieved on average an accuracy of 67.7 % on the validation data and 46.7% on the additional validation data set. One can find its confusion matrix of the official validation data in Figure 1 and the according boolean matrix with precision and recall values in table 4.//

Table 4: Boolean Matrix for Meta-Ensemble

Label	Predicted Correctly	Predicted Incorrectly	Precision	Recall
WEB	2	2	0.40	0.50
DOCS	2	2	0.50	0.50
HW	5	0	0.71	1.00
DEV	8	3	0.89	0.73
EDU	2	2	1.00	0.50
DATA	2	1	0.50	0.67
Weighted Average			0.72	0.68

[24] https://github.com/InformatiCup/InformatiCup2017/blob/master/example-output

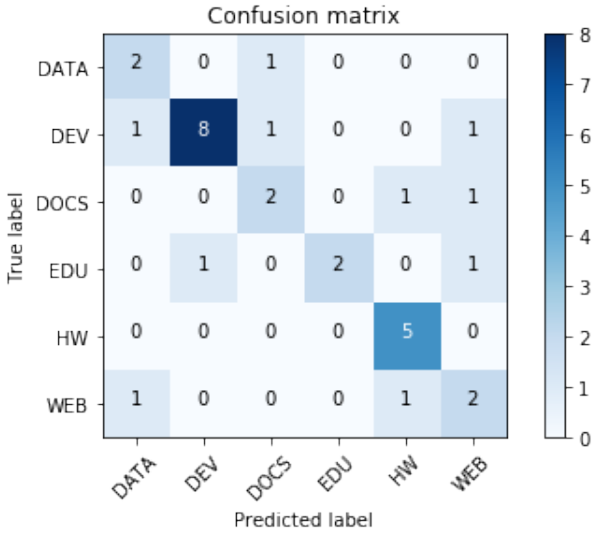

Figure 1: Confusion Matrix for Meta Ensemble (67.7% Accuracy)

In our opinion, we prefer a higher recall over a higher precision. We would like to use our prediction model to recommend repos to a user that are similar to his own repositorries. In this situation it might be better for a user to reject a repository by himself as interesting (high recall and low precision) than not seing an interesting repo at all (low recall and high precision).

The great benefit of our ensemble classifier is that the base classifiers are independent of each other. This means that even if one classifier predicts the class label incorrectly, the remaining two classifiers can outrule him using their prediction confidences.

One example would be the repository ga-chicago/wdi5-homework[25]. Since its readme file is empty, fellow approaches that might only use the content of the readme file to predict the label wouldn't be able to predict a label with high certainty. Instead, our description classifier can predict *HW* with a high confidence as the description includes positive *HW* words like "homework", "assignment" and "submission".

The readme and the numeric ensemble classifiers of our meta ensemble predict the label *DEV* for certain as there multiple positive *DEV* words included in the readme like "source code" or "bugs" and as there are a lot of issues, pull requests and commits.

Fellow solution approaches that only use numeric meta data of repos would probably classify openaddresses/openaddresses[26] as *DEV* as it contains a lot of contributions, stargazers, forks, issues and pull requests. Since there are a lot of positive *DATA* words contained in the description and the readme of the repo, like "open" and "data", the description and readme classifiers would certainly overrule the numeric ensemble classifier with the predicted *DATA* label. The description of the repo PowerDNS/pdns[27] doesn't include any words that could be assigned to any label for certain. Approaches that only use the description could simply fail here.

[25] https://github.com/ga-chicago/wdi5-homework
[26] https://github.com/openaddresses/openaddresses
[27] https://github.com/PowerDNS/pdns

6. Extensions

To bring our research work to production, we built a service[28] that classifies your public GitHub repositories using models that were trained on our training data. The server uses GitHub OAuth to authenticate GitHub users and uses their OAuth tokens to request their public repositories and their necessary features. We're planning to improve the design with visualizations of the repository distribution using D3.
Another extension would be to recommend trending GitHub projects[29] based on the public repositories of the user.
Since there is no official GitHub API for the trending repositories, we would crawl all websites that are available at
https://github.com/trending/*language*?since=*since* once a day, where *language* is a supported programming language like Python or Ruby, and *since* is one of 'daily', 'weekly' or 'monthly'. We would then recommend repositories to the user based on their classified labels, on the preferred language of the user, on the text or even code similarity between the trending projects and those of the user. To implement the latter one, we could use tf-idf matrices like we already used for the text classifiers.

[28] https://git-better.herokuapp.com/
[29] https://github.com/trending

A.
General figures and tables

Table 5: Origins of labeled training data

Amount	Label	Origin
9	DATA	Manual Google search for Open Data repositories
82	DATA	Repositories of GitHub user 'datasets'
17	EDU	GitHub Search for "course, material"
17	DOCS	GitHub Search for "documentation"
423	WEB	Google Search for "site:.github.io"
58	HW	GitHub Search for "homework, assignments, solution"
13	DEV	Showcases "Virtual Reality"
12	DEV	Showcases "Software Development Tools"
14	DEV	Showcases "Front-end JavaScript frameworks"
20	DEV	Showcases "DevOps tools"
16	DEV	Showcases "Text editors"
24	DEV	Showcases "Game Engines"
27	DEV	Showcases "Web Application Frameworks"
42	DEV	Showcases "Programming Languages"
180	DOCS	GitHub Repo Content: awesome-awesomeness
6	DATA	Showcases "Open Data"
86	HW	Github Search for "homework, solution"

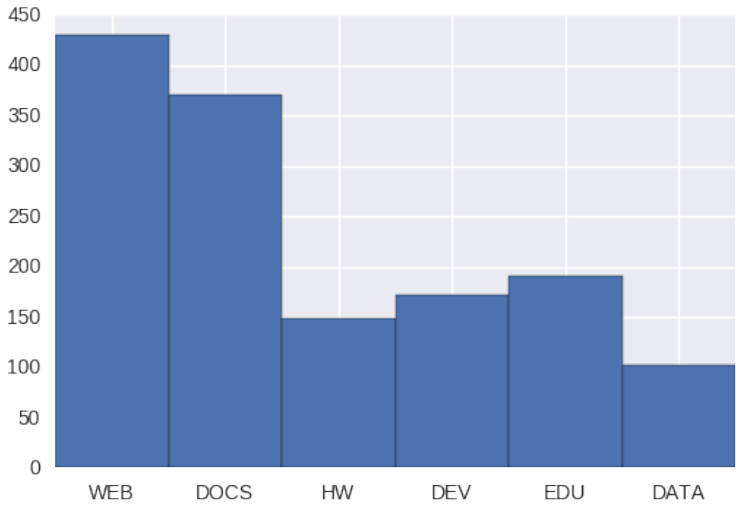

Figure 2: Training Data Distribution

Table 6: Numeric Features

Feature Name	Description
watchers	Number of users who get notifications about the repo
mentionableUsers	Number of mentionable users (collaborators, contributors, ...)
open_pull_requests	Number of open pull requests
closed_pull_requests	Number of closed pull requests
merged_pull_requests	Number of merged pull requests
open_issues	Number of open issues
closed_issues	Number of closed issues
forks	Number of forks
stargazers	Number of users who "starred" the repo
projects	Number of projects (integrated project management tool)
size	Size of the source code in kilobyte
isOwnerHomepage	Is the name of the repo REPO_OWNER.github.io or REPO_OWNER.github.com?
hasHomepage	Does the website REPO_OWNER.github.io/REPO_NAME exist?
hasLicense	Does the repo have a license file?
hasTravisConfig	Does the repo have a Travis configuration file?
hasCircleConfig	Does the repo have a CircleCI configuration file?
hasCiConfig	hasTravisConfig OR hasCircleConfig
commitsCount	Number of commits
branchesCount	Number of branches
tagsCount	Number of tags
releasesCount	Number of releases
LANGUAGE_*	How much code was written in the language in percent (e.g. LANGUAGE_Python, ...)

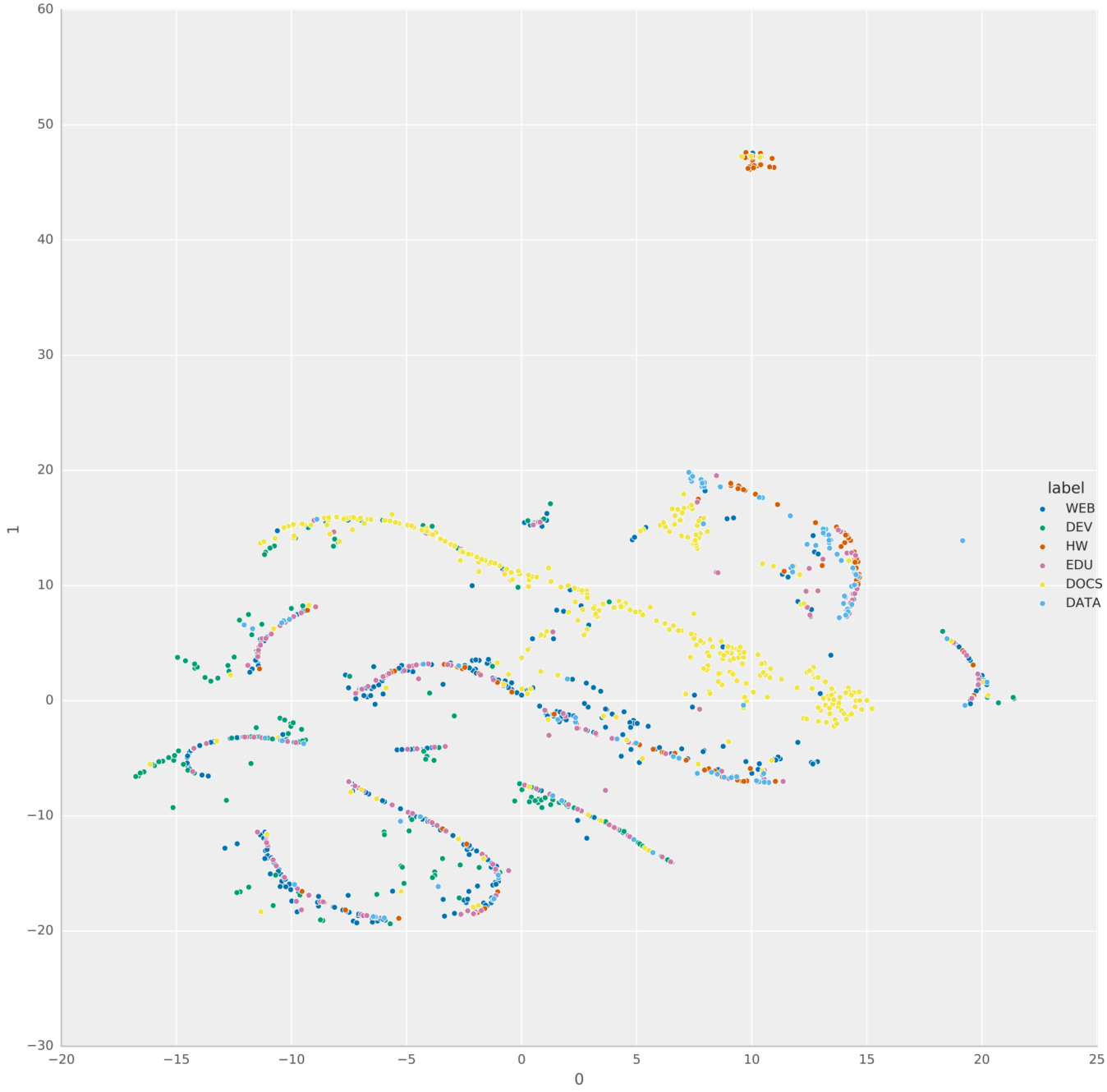

Figure 3: Distribution of the labeled data entries using t-SNE

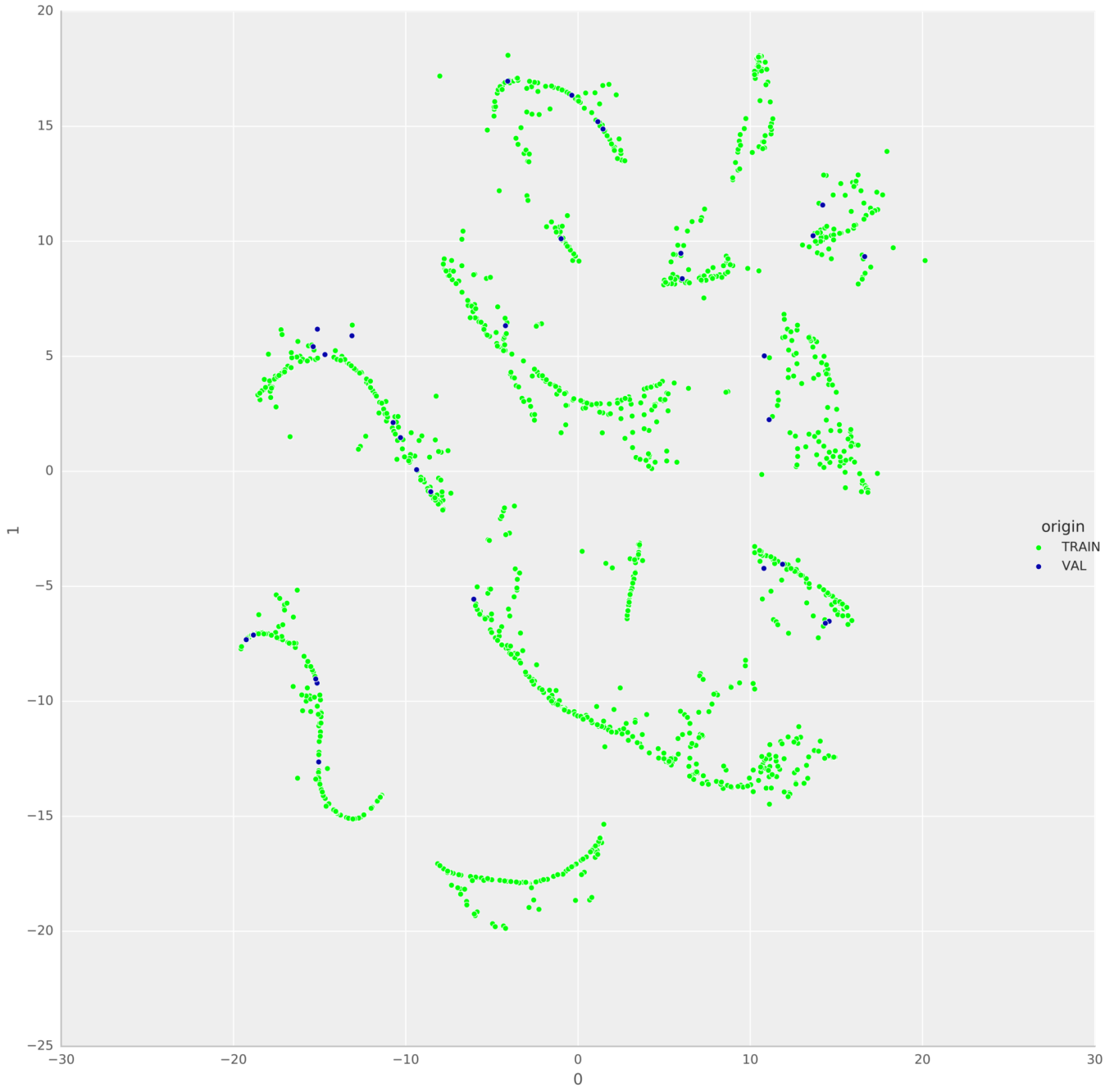

Figure 4: Distribution of the validation data entries using t-SNE

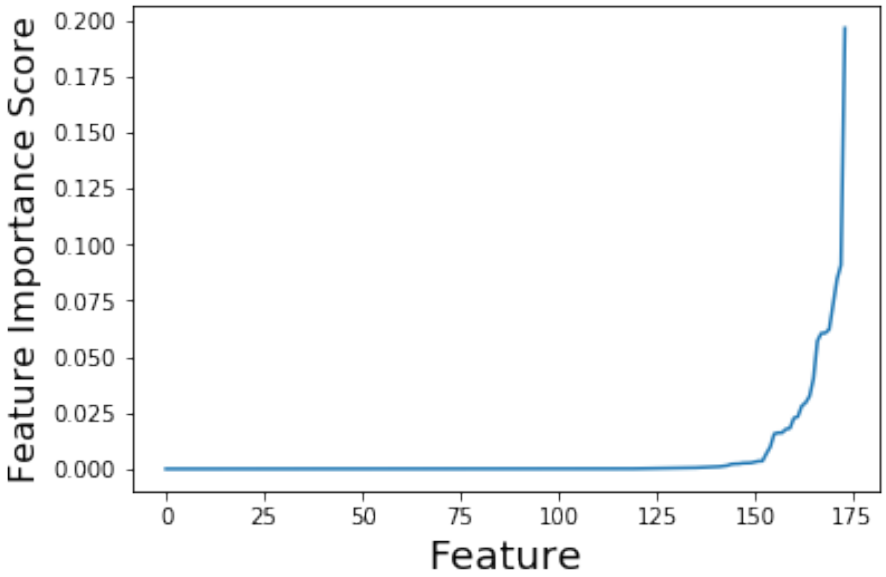

Figure 5: Feature Importance Scores for Numeric Features

Table 7: Top 20 Most Important Numeric Features

Feature Names	Scores	Feature Names	Scores
isOwnerHomepage	0.1793	hasHomepage	0.1297
stargazers	0.0726	mentionableUsers	0.0642
size	0.0581	watchers	0.0561
commitsCount	0.0492	closed_issues	0.0415
merged_pull_requests	0.0389	open_issues	0.0379
forks	0.0376	closed_pull_requests	0.0368
tagsCount	0.0245	branchesCount	0.0239
hasTravisConfig	0.0235	open_pull_requests	0.0223
releasesCount	0.0214	hasLicense	0.0148
hasCiConfig	0.0124	LANGUAGE_Python	0.0090

Table 8: Accuracies on the given validation data and additional validation data

	Naive Bayes	Tree	Forest	k-NN	SVM	Gradient Boost	Ensemble
Validation	40%	35%	42%	25%	25%	41%	45%
Add. Val.	19%	21%	23%	21%	20%	22%	25%

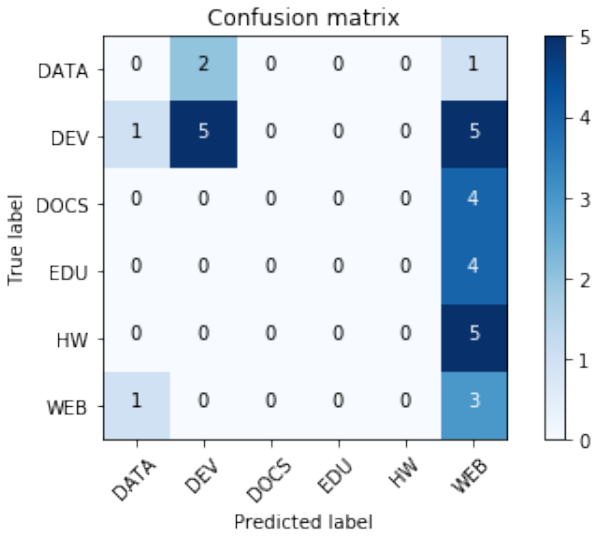

Figure 6: Confusion Matrix for Numeric Ensemble (46% Accuracy)

Table 9: Boolean Matrix for Numeric Ensemble

Label	Predicted Correctly	Predicted Incorrectly	Precision	Recall
WEB	3	1	0.14	0.75
DOCS	0	4	0.00	0.00
HW	0	5	0.00	0.00
DEV	5	6	0.71	0.45
EDU	0	4	0.00	0.00
DATA	0	3	0.00	0.00
Weighted Average			0.27	0.26

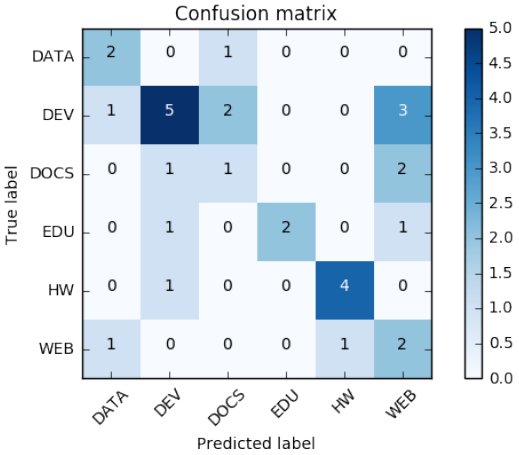

Figure 7: Readme Classifier: 51.6% Accuracy on Validation Data

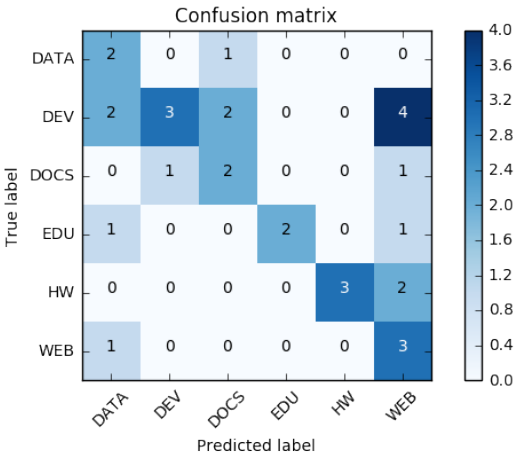

Figure 8: Description Classifier: 48.4% Accuracy on Validation Data

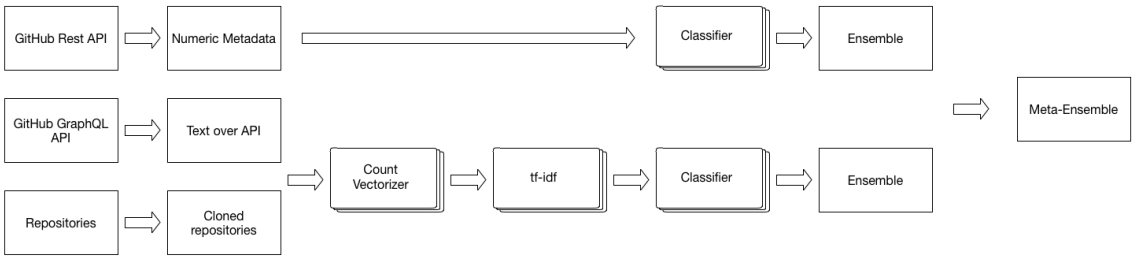

Figure 9: Explanation of our Process

B.
Evaluation of description classifier

Table 10: Classifier on description - DATA category

	Positive words		Negative words	
1	7.0361	data	-5.2058	materi
2	3.9519	dataset	-4.7668	homework
3	3.6254	us	-4.5316	websit
4	3.5437	world	-4.5211	awesom
5	3.4038	index	-4.3327	curat
6	3.3833	global	-3.8421	cours
7	3.3229	metadata	-3.6445	solut
8	3.1875	price	-3.4706	for
9	3.1035	from	-3.0764	lectur
10	3.0402	datapackag	-3.0227	blog
11	2.8330	countri	-2.9226	resourc
12	2.8262	iso	-2.8689	github
13	2.7782	gdp	-2.7790	program
14	2.7008	govern	-2.7062	javascript
15	2.6220	639	-2.6174	tool
16	2.5532	in	-2.6164	on
17	2.4712	name	-2.6149	framework
18	2.4402	refer	-2.4988	sourc
19	2.4264	as	-2.4142	page
20	2.4074	catalogu	-2.2015	document

Table 11: Classifier on description - DEV category

	Positive words		Negative words	
1	5.7393	languag	-6.1320	materi
2	5.0252	framework	-5.4495	homework
3	4.2394	build	-4.6557	cours
4	3.9844	editor	-4.5762	list
5	3.7652	compil	-3.6910	websit
6	3.7291	docker	-3.6338	resourc
7	3.6943	platform	-3.5035	awesom
8	3.6177	studio	-3.4356	blog
9	3.6162	cocos2d	-3.3799	of
10	3.5179	program	-3.3534	solut
11	3.4163	applic	-3.2057	lectur
12	3.3236	game	-3.1421	curat
13	3.3018	easi	-3.0282	data
14	3.2214	javascript	-2.6971	github
15	3.1316	is	-2.3795	collect
16	3.1202	rail	-2.3225	page
17	3.0622	app	-2.2186	document
18	3.0250	open	-2.1600	site
19	2.9714	ide	-2.0698	showcas
20	2.8593	base	-1.8867	this

Table 12: Classifier on description - DOCS category

	Positive words		Negative words	
1	8.4647	awesom	-4.0270	materi
2	6.5280	resourc	-3.9414	homework
3	6.4244	curat	-3.4181	data
4	6.1888	list	-3.0279	the
5	4.6624	document	-2.8904	websit
6	3.6095	tool	-2.7715	solut
7	3.4229	of	-2.7210	build
8	3.0736	use	-2.6546	lectur
9	2.9668	pantheon	-2.5698	cours
10	2.9316	beautif	-2.4605	revel
11	2.9316	typefac	-2.3201	open
12	2.9060	servic	-2.1158	blog
13	2.6650	doc	-1.9969	page
14	2.6100	datadog	-1.7603	amcss
15	2.5669	paa	-1.7533	python
16	2.5485	delight	-1.7496	contain
17	2.4022	know	-1.6282	compil
18	2.3836	onlin	-1.6248	in
19	2.3533	must	-1.6173	metadata
20	2.3533	watch	-1.5864	as

Table 13: Classifier on description - EDU category

	Positive words		Negative words	
1	14.4355	materi	-4.9311	homework
2	7.8989	cours	-4.3935	solut
3	6.9177	lectur	-2.9709	list
4	3.3864	univers	-2.7412	web
5	2.7723	guid	-2.6545	websit
6	2.3614	scienc	-2.2765	blog
7	2.2123	adequ	-2.1165	github
8	2.2123	fp	-2.1046	of
9	2.2058	at	-2.0799	rubi
10	2.0191	slide	-1.9963	awesom
11	1.9547	most	-1.8475	curat
12	1.9298	for	-1.7231	languag
13	1.9263	teach	-1.7161	is
14	1.8868	group	-1.7129	document
15	1.8868	pretoria	-1.6204	page
16	1.6207	chen	-1.5823	open
17	1.6207	cs35l	-1.5811	framework
18	1.6207	liu	-1.5220	homepag
19	1.6207	ta	-1.5157	librari
20	1.6207	ucla	-1.4777	price

Table 14: Classifier on description - HW category

	Positive words		Negative words	
1	12.7439	homework	-4.1812	materi
2	10.0043	solut	-2.8605	websit
3	4.7062	assign	-2.6204	web
4	2.8646	calcul	-2.5991	list
5	2.5842	cs193p	-2.3558	github
6	2.4037	winter	-2.0415	languag
7	2.0581	2015	-1.9265	awesom
8	1.8079	week	-1.9016	blog
9	1.7219	class	-1.8021	and
10	1.5998	problem	-1.7842	code
11	1.5680	wk	-1.7414	of
12	1.4612	answer	-1.6898	data
13	1.4602	html	-1.5913	curat
14	1.4382	java	-1.5866	sourc
15	1.4131	coursera	-1.5302	rail
16	1.3892	css	-1.4691	document
17	1.3027	person	-1.4291	homepag
18	1.2823	third	-1.4236	tool
19	1.2775	2016	-1.4177	framework
20	1.2339	this	-1.3875	from

Table 15: Classifier on description - WEB category

	Positive words		Negative words	
1	7.4655	websit	-6.0160	homework
2	6.1025	blog	-5.9861	materi
3	5.5628	github	-4.1386	solut
4	4.4649	page	-4.1177	for
5	4.0825	revel	-4.0522	data
6	3.3829	homepag	-3.9253	list
7	3.1017	site	-3.6970	of
8	2.8586	showcas	-3.6283	languag
9	2.7452	googl	-3.3458	awesom
10	2.7357	kizu	-3.3140	and
11	2.7357	ru	-3.2077	framework
12	2.7013	concurr	-3.1889	cours
13	2.6509	[Chinese]	-3.0793	lectur
14	2.6308	demo	-2.9587	to
15	2.6116	overthec	-2.8921	javascript
16	2.6015	[Chinese]	-2.5442	from
17	2.5841	[Chinese]	-2.4739	curat
18	2.5389	[Chinese]	-2.4250	program
19	2.2622	scientist	-2.4132	is
20	2.2314	jekyl	-2.4083	tool

C.
Evaluation of readme classifier

Table 16: Classifier on readme - DATA category

	Positive words		Negative words	
1	7.8583	data	-3.9184	https
2	4.0536	countri	-3.6546	com
3	3.3030	csv	-3.5891	cours
4	3.2380	gov	-3.4031	materi
5	3.0676	iso	-3.2363	github
6	3.0547	list	-2.9113	homework
7	3.0170	umpirski	-2.8006	solut
8	2.9225	dataset	-2.7040	io
9	2.8163	domain	-2.2393	lectur
10	2.8153	public	-1.9645	project
11	2.7540	api	-1.9335	http
12	2.6251	planet	-1.9241	scienc
13	2.6138	geojson	-1.8566	html
14	2.5873	cern	-1.8548	edu
15	2.4975	openaddress	-1.7322	my
16	2.3876	opendata	-1.6639	build
17	2.3096	from	-1.6574	jekyl
18	2.2579	price	-1.6311	you
19	2.2189	of	-1.6220	comput
20	2.1932	swagger	-1.6007	test

Table 17: Classifier on readme - DEV category

	Positive words		Negative words	
1	4.6286	build	-5.1795	data
2	3.9624	ci	-5.1652	github
3	3.6748	groovi	-3.8998	jekyl
4	3.5901	project	-3.8516	cours
5	3.5309	vr	-3.6040	awesom
6	3.4993	gearvrf	-3.4754	materi
7	3.4902	compil	-3.2811	homework
8	3.4149	travi	-3.1003	lectur
9	3.1553	leo	-2.6797	this
10	3.1352	chapel	-2.3362	solut
11	2.9426	sentri	-2.3225	for
12	2.9378	pybuild	-2.2535	of
13	2.8945	bug	-2.2084	css
14	2.8772	purescript	-1.9942	site
15	2.8738	neovim	-1.8891	page
16	2.8151	urho3d	-1.8038	locomotivecm
17	2.7934	hanami	-1.7969	www
18	2.7825	wiki	-1.7582	repositori
19	2.7728	get	-1.7416	websit
20	2.7543	revel	-1.6458	2015

Table 18: Classifier on readme - DOCS category

	Positive words		Negative words	
1	7.7354	https	-4.4972	io
2	7.1523	awesom	-3.0067	cours
3	6.7922	com	-2.8637	homework
4	4.0843	document	-2.8229	lectur
5	3.8879	galera	-2.8190	materi
6	3.6536	locomotivecm	-2.7407	instal
7	3.4437	coala	-2.1770	the
8	3.4260	book	-2.1613	jekyl
9	2.7398	cfengin	-2.0899	ci
10	2.7091	women	-1.9770	solut
11	2.6974	www	-1.8787	licens
12	2.6774	django	-1.8748	run
13	2.6365	git	-1.8032	vuej
14	2.4867	owncloud	-1.6879	project
15	2.4532	cluster	-1.6592	urho3d
16	2.4439	relay	-1.6587	my
17	2.4412	cmake	-1.6414	href
18	2.4408	tool	-1.6302	simongfxu
19	2.4261	hackag	-1.6241	gitter
20	2.4243	phalcon	-1.6030	we

Table 19: Classifier on readme - EDU category

	Positive words		Negative words	
1	10.0881	materi	-5.4069	homework
2	7.9913	lectur	-4.5302	solut
3	7.6389	cours	-3.1688	com
4	4.3368	slide	-2.8119	github
5	3.2077	notebook	-2.1720	jekyl
6	3.1980	cse3521	-2.1481	assign
7	3.0093	datasciencelab	-2.0511	data
8	2.6802	univers	-1.9381	is
9	2.5043	class	-1.8075	awesom
10	2.4515	icm	-1.7643	my
11	2.3853	lxc	-1.7424	web
12	2.2942	lesson	-1.6623	http
13	2.2457	teachapc	-1.6502	list
14	2.2403	present	-1.6117	blog
15	2.2222	kottan	-1.5987	releas
16	2.2174	test1	-1.5692	dataset
17	2.2138	student	-1.5292	cis194
18	2.1497	p5	-1.4639	org
19	2.1395	concept	-1.4086	iso
20	2.1193	will	-1.3895	the

Table 20: Classifier on readme - HW category

	Positive words		Negative words	
1	8.4521	homework	-5.8646	github
2	7.1158	solut	-5.1770	com
3	4.5826	assign	-5.1689	materi
4	3.2987	comput	-5.0642	https
5	2.4345	test	-5.0002	http
6	2.2072	hw	-3.7191	data
7	2.1898	cs193p	-3.6454	org
8	2.1821	leanpub	-3.3732	io
9	2.1021	000000	-2.9798	for
10	2.0822	2015	-2.6420	licens
11	2.0800	he	-2.1398	www
12	1.9991	tank	-2.0841	instal
13	1.9722	majesti	-2.0784	to
14	1.8371	doubl	-2.0723	and
15	1.8103	vuejs2	-1.9034	is
16	1.6699	click	-1.7960	build
17	1.6451	length	-1.6171	kizu
18	1.6429	week	-1.6145	lectur
19	1.6087	pset	-1.6121	dataset
20	1.5748	py	-1.5995	with

Table 21: Classifier on readme - WEB category

	Positive words		Negative words	
1	7.7932	jekyl	-5.9238	https
2	6.5853	io	-4.4272	and
3	5.5661	github	-3.8526	data
4	4.1321	site	-3.8485	homework
5	3.9070	websit	-3.3797	cours
6	3.1815	page	-3.1061	materi
7	3.0921	blog	-3.0131	of
8	2.8634	pidcod	-2.9675	solut
9	2.8538	post	-2.3174	www
10	2.8127	simongfxu	-2.2716	lectur
11	2.6199	mentor	-1.9422	datasciencelab
12	2.5503	sheetj	-1.8246	from
13	2.4941	kizu	-1.8075	test
14	2.4524	npm	-1.7795	vue
15	2.3540	mqtt	-1.7729	assign
16	2.2606	rancher	-1.7725	list
17	2.1954	scalaspac	-1.7685	program
18	2.1677	jser	-1.6994	awesom
19	2.0959	nswebfrog	-1.6510	org
20	2.0959	[Chinese]	-1.6095	cse3521

D.
Evaluation of file name classifier

Table 22: Classifier on file names - DATA category

	Positive words		Negative words	
1	3.2251	json	-1.5670	packag
2	1.1139	makefil	-1.5648	test
3	0.9449	sh	-1.5623	main
4	0.6635	xml	-1.4399	config
5	0.6452	py	-1.1262	png
6	0.4514	readm	-1.0827	yml
7	0.3790	licens	-0.9016	contribut
8	0.3784	index	-0.7405	jpg
9	0.3127	txt	-0.4513	css
10	0.1726	gitignor	-0.2326	travi
11	0.0919	pdf	-0.2206	util
12	0.0861	md	-0.1236	html
13			-0.0684	svg
14			-0.0304	js

Table 23: Classifier on file names - DEV category

	Positive words		Negative words	
1	3.4637	util	-1.9203	pdf
2	2.3105	packag	-1.0563	travi
3	1.8472	yml	-0.9946	css
4	1.7875	test	-0.5838	readm
5	1.4706	sh	-0.5619	md
6	1.4247	js	-0.3192	gitignor
7	1.3238	html		
8	1.2926	makefil		
9	0.9907	config		
10	0.9156	main		
11	0.7826	png		
12	0.7679	licens		
13	0.7235	index		
14	0.7082	xml		
15	0.5805	txt		
16	0.4977	py		
17	0.3078	contribut		
18	0.2526	json		
19	0.1224	jpg		
20	0.0541	svg		

Table 24: Classifier on file names - DOCS category

	Positive words		Negative words	
1	4.5622	contribut	-1.0314	config
2	2.8728	travi	-1.0044	html
3	2.0420	yml	-0.7121	xml
4	1.7639	licens	-0.6340	json
5	1.7214	md	-0.5722	makefil
6	1.2027	packag	-0.5702	main
7	1.1108	svg	-0.5697	test
8	0.6831	png	-0.5455	sh
9	0.6714	index	-0.4359	py
10	0.3497	readm	-0.4040	txt
11	0.2890	css	-0.3430	js
12	0.2780	jpg	-0.2187	pdf
13			-0.1440	util
14			-0.0722	gitignor

Table 25: Classifier on file names - EDU category

	Positive words		Negative words	
1	1.7966	pdf	-1.9717	contribut
2	1.5195	html	-1.2080	yml
3	1.3308	jpg	-1.1482	util
4	1.0053	svg	-0.8443	json
5	0.9685	png	-0.2571	packag
6	0.8496	gitignor		
7	0.6726	py		
8	0.6248	makefil		
9	0.5158	licens		
10	0.4517	css		
11	0.4501	txt		
12	0.4491	xml		
13	0.4232	sh		
14	0.3983	js		
15	0.3342	main		
16	0.3019	test		
17	0.2778	readm		
18	0.1407	md		
19	0.1085	travi		
20	0.1042	config		

Table 26: Classifier on file names - HW category

	Positive words		Negative words	
1	2.0428	css	-1.5755	contribut
2	1.7615	main	-1.3501	licens
3	1.5388	config	-1.0360	yml
4	1.2715	gitignor	-0.8501	travi
5	0.9140	txt	-0.8123	svg
6	0.7449	readm	-0.8035	util
7	0.5342	py	-0.5982	packag
8	0.5257	test	-0.4826	json
9	0.4687	xml	-0.3102	makefil
10	0.4144	index	-0.2012	sh
11	0.3984	pdf	-0.1943	html
12	0.3699	js	-0.1773	png
13	0.3128	md		
14	0.2245	jpg		

Table 27: Classifier on file names - WEB category

	Positive words		Negative words	
1			-2.4446	md
2			-2.0628	readm
3			-1.9461	js
4			-1.9392	txt
5			-1.9117	html
6			-1.9110	png
7			-1.8890	json
8			-1.8842	py
9			-1.8702	xml
10			-1.8512	pdf
11			-1.8435	gitignor
12			-1.7930	travi
13			-1.7274	test
14			-1.7246	config
15			-1.7224	jpg
16			-1.7115	main
17			-1.7010	css
18			-1.6888	makefil
19			-1.6295	licens
20			-1.6165	index

Table 28: Boolean Matrix for Validation Data

Label	Predicted Correctly	Predicted Incorrectly	Precision	Recall
WEB	1	3	0.50	0.25
DOCS	3	1	0.33	0.75
HW	2	3	0.40	0.40
DEV	7	4	0.88	0.64
EDU	1	3	0.20	0.25
DATA	1	2	0.50	0.33
Weighted Average			0.56	0.48

Table 29: Boolean Matrix for Additional Validation Data

Label	Predicted Correctly	Predicted Incorrectly	Precision	Recall
WEB	8	32	0.42	0.20
DOCS	18	18	0.28	0.50
HW	9	34	0.18	0.21
DEV	29	27	0.40	0.52
OTHER	0	57	0.00	0.00
EDU	9	26	0.13	0.26
DATA	15	18	0.65	0.45
Weighted Average			0.28	0.29

E.
Evaluation of commit message classifier

Table 30: Classifier on commit messages - DATA category

	Positive words		Negative words	
1	6.3275	data	-0.9805	add
2	4.8814	json	-0.8676	test
3	3.1334	script	-0.8634	link
4	2.9542	csv	-0.7580	merg
5	1.4706	field	-0.7288	md
6	1.2031	readi	-0.6596	ad
7	1.0819	process	-0.6568	on
8	1.0631	readm	-0.6063	solut
9	1.0128	licens	-0.5887	branch
10	0.9766	chang	-0.5734	post
11	0.9483	sourc	-0.5626	page
12	0.8778	py	-0.5615	github
13	0.8193	name	-0.5497	com
14	0.8079	to	-0.5345	slide
15	0.8021	file	-0.4961	exampl
16	0.7784	plus	-0.4944	site
17	0.7185	misc	-0.4832	use
18	0.7065	month	-0.4739	html
19	0.6837	all	-0.4679	fix
20	0.6297	and	-0.4480	assign

Table 31: Classifier on commit messages - DEV category

	Positive words		Negative words	
1	1.7923	when	-3.4524	add
2	1.4852	use	-2.8424	ad
3	1.3744	id	-2.7999	updat
4	1.3568	in	-2.2571	link
5	1.2274	test	-1.7823	data
6	1.1795	the	-1.6006	page
7	1.1672	option	-1.3643	post
8	1.1310	git	-1.3029	solut
9	1.1033	support	-1.2225	chang
10	1.0973	if	-1.2008	md
11	1.0584	is	-1.1761	github
12	0.9975	fix	-1.1358	slide
13	0.9391	v0	-1.1135	commit
14	0.9379	this	-1.0996	master
15	0.9300	that	-1.0619	readm
16	0.8955	compil	-1.0045	section
17	0.8906	type	-0.9987	typo
18	0.8627	error	-0.9902	initi
19	0.8552	into	-0.9440	from
20	0.8535	method	-0.9268	patch

Table 32: Classifier on commit messages - DOCS category

	Positive words		Negative words	
1	4.9241	add	-2.3135	html
2	4.4610	awesom	-2.0499	file
3	4.0540	section	-1.9260	index
4	4.0265	pull	-1.9059	page
5	3.9133	request	-1.7599	solut
6	3.7437	link	-1.4933	post
7	3.1669	patch	-1.4293	commit
8	2.9490	merg	-1.3702	slide
9	2.9405	list	-1.3492	for
10	2.8275	from	-1.3268	creat
11	2.4762	readm	-1.3165	data
12	2.3798	ad	-1.2928	github
13	2.0401	tool	-1.1980	chang
14	1.9745	contribut	-1.1980	fix
15	1.9428	articl	-1.1363	build
16	1.8635	book	-1.0722	json
17	1.8239	badg	-1.0615	modifi
18	1.7815	md	-1.0259	branch
19	1.6462	descript	-0.9206	class
20	1.6116	categori	-0.9126	assign

Table 33: Classifier on commit messages - EDU category

	Positive words		Negative words	
1	4.8189	slide	-1.3756	request
2	3.5687	materi	-1.3194	pull
3	2.8443	note	-1.1966	post
4	2.7333	exampl	-1.1036	from
5	2.4935	typo	-1.0821	js
6	2.1294	class	-1.0582	list
7	2.0862	cours	-1.0572	commit
8	2.0194	for	-1.0227	page
9	1.7788	ad	-0.9904	awesom
10	1.6383	week	-0.9638	new
11	1.5749	day	-0.8748	json
12	1.4916	file	-0.8607	articl
13	1.1962	more	-0.8403	link
14	1.1938	of	-0.8332	io
15	1.1884	start	-0.8131	patch
16	1.1777	pdf	-0.7909	style
17	1.1736	intro	-0.7857	index
18	1.1541	initi	-0.7219	blog
19	1.1193	outlin	-0.6889	doc
20	1.1149	code	-0.6649	cname

Table 34: Classifier on commit messages - HW category

	Positive words		Negative words	
1	3.7706	solut	-1.8244	updat
2	3.1222	assign	-1.3799	from
3	2.2168	finish	-1.1874	typo
4	1.5660	first	-1.1179	to
5	1.4488	commit	-1.0829	data
6	1.3772	initi	-1.0238	link
7	1.3315	file	-1.0042	request
8	1.2524	rm	-0.9644	pull
9	1.2426	implement	-0.9406	share
10	1.2348	final	-0.8553	materi
11	1.1015	gitignor	-0.8335	publish
12	1.0582	test	-0.7771	deploy
13	1.0376	creat	-0.7384	post
14	1.0206	readm	-0.7303	merg
15	0.9933	init	-0.7293	slide
16	0.9793	delet	-0.7153	fix
17	0.9746	complet	-0.7124	json
18	0.9540	and	-0.6619	licens
19	0.9182	work	-0.6608	replac
20	0.9062	done	-0.6249	content

Table 35: Classifier on commit messages - WEB category

	Positive words		Negative words	
1	5.2228	post	-4.2449	readm
2	4.5782	page	-3.2327	ad
3	4.0799	html	-2.3167	awesom
4	3.8468	index	-2.1704	add
5	3.1558	github	-2.0992	data
6	2.4457	site	-1.9957	solut
7	2.2197	cname	-1.8982	and
8	2.1946	updat	-1.8864	initi
9	2.0965	jekyl	-1.5155	json
10	1.9677	blog	-1.4862	section
11	1.9494	css	-1.4597	assign
12	1.7129	io	-1.2696	in
13	1.5963	style	-1.2694	first
14	1.5718	deploy	-1.2590	for
15	1.5630	logo	-1.1961	file
16	1.5305	build	-1.1522	week
17	1.5067	publish	-1.1298	creat
18	1.3971	link	-1.1067	slide
19	1.3130	com	-1.0628	is
20	1.3087	replac	-1.0259	of

Table 36: Boolean Matrix for Validation Data

Label	Predicted Correctly	Predicted Incorrectly	Precision	Recall
WEB	4	0	0.25	1.00
DOCS	0	4	0.00	0.00
HW	1	4	1.00	0.20
DEV	6	5	0.86	0.55
EDU	1	3	0.50	0.25
DATA	0	3	0.00	0.00
Weighted Average			0.56	0.39

Table 37: Boolean Matrix for Additional Validation Data

Label	Predicted Correctly	Predicted Incorrectly	Precision	Recall
WEB	34	6	0.31	0.85
DOCS	10	26	0.17	0.28
HW	15	28	0.34	0.35
DEV	28	28	0.70	0.50
OTHER	0	57	0.00	0.00
EDU	13	22	0.41	0.37
DATA	15	18	0.88	0.45
Weighted Average			0.39	0.38

F.
Evaluation of source code classifier

Table 38: Classifier on source code - DATA category

	Positive words		Negative words	
1	15.8798	csv	-5.1443	00
2	14.5357	01	-4.4907	2016
3	11.4974	countri	-4.1629	return
4	10.4433	dataset	-3.5731	utc
5	8.6672	citi	-3.4172	pm
6	8.5019	pddl	-3.3633	cu
7	8.3280	2007	-3.2122	function
8	8.2628	sic	-3.1996	color
9	7.9571	type	-3.1319	5t
10	7.6349	gov	-2.8658	awesom
11	6.9777	vnd	-2.8290	att
12	6.9381	data	-2.6345	your
13	6.9189	2011	-2.5761	ss
14	6.9100	2014	-2.5761	000000
15	6.8459	gmd	-2.5225	you
16	6.5427	counti	-2.4980	fill
17	6.1641	errorminus	-2.4679	5q
18	6.1636	errorplus	-2.2536	class
19	6.0823	iso	-2.2393	wafflej
20	5.7965	descript	-2.1996	am

Table 39: Classifier on source code - DEV category

	Positive words		Negative words	
1	12.8545	fileid	-5.1870	repositori
2	9.9441	const	-4.4365	you
3	9.6625	end	-4.4138	awesom
4	9.3110	expect	-4.0737	your
5	8.7904	foo	-3.5679	length
6	7.4921	foodicon	-3.5644	function
7	7.2273	err	-3.4935	href
8	6.6612	vertex	-3.3018	titl
9	6.1595	assert	-3.2206	descript
10	6.1401	octokit	-3.0772	10
11	5.9895	openstack	-3.0660	data
12	5.7023	self	-3.0568	number
13	5.6651	do	-3.0128	android
14	5.3153	valu	-3.0005	www
15	5.2112	return	-2.9120	color
16	5.2013	nil	-2.6552	homework
17	5.1730	0x00	-2.6512	rank
18	5.1172	env	-2.5602	cout
19	4.9893	docker	-2.5311	objectbuilder2
20	4.9574	func	-2.4871	solut

Table 40: Classifier on source code - DOCS category

	Positive words		Negative words	
19	5.2395	frame	-2.0442	b02598
20	5.0788	foral	-2.0305	request

Table 41: Classifier on source code - HW category

	Positive words		Negative words	
1	14.5114	homework	-4.4606	type
2	11.0211	int	-3.5530	null
3	10.8537	solut	-3.0308	m6
4	10.7025	number	-2.6859	0x00
5	9.5552	system	-2.6049	licens
6	8.9972	string	-2.6008	descript
7	8.0153	export	-2.5770	foo
8	7.8574	consol	-2.3945	span
9	7.1376	gladiat	-2.2420	highlight
10	7.1354	log	-2.2364	font
11	7.0993	col	-2.2311	class
12	7.0549	softuni	-2.2201	const
13	6.8284	begin_layout	-2.1499	awesom
14	6.8284	end_layout	-2.1209	org
15	6.2421	knapsackitem	-2.1174	code
16	6.1284	gene	-2.1091	www
17	6.0220	minibatch	-2.0749	fsharp
18	6.0120	guess	-2.0534	ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
19	6.0016	3296	-2.0460	expect
20	5.8436	ski	-1.9849	11

Table 42: Classifier on source code - WEB category

	Positive words		Negative words	
1	29.9622	5t	-8.6180	self
2	21.2225	href	-7.9384	end
3	21.1386	color	-7.6852	col
4	18.2118	function	-7.5040	consol
5	16.1452	glyph	-7.4804	awesom
6	15.5923	highlight	-7.4601	foodicon
7	14.7536	div	-6.9887	log
8	13.8095	li	-6.7684	int
9	13.5162	var	-6.6451	string
10	13.3720	margin	-6.5137	markdown
11	13.1328	span	-6.1330	expect
12	12.7354	font	-5.9628	section
13	12.6367	post	-5.7539	arg
14	12.4874	background	-5.5597	01
15	11.7889	left	-5.4579	assert
16	11.6455	horiz	-5.1985	should
17	11.5029	unicod	-5.1880	key
18	11.3711	class	-5.1153	let
19	11.2306	hkern	-5.0395	param
20	11.1553	adv	-5.0009	const

G.
Evaluation of small source code classifier

Table 43: Classifier on source code - DATA category

	Positive words		Negative words	
1	3.0123	01	-1.2137	return
2	2.3475	data	-0.8911	function
3	2.1259	type	-0.8050	color
4	1.4798	code	-0.7714	your
5	1.4231	unit	-0.7524	you
6	1.4073	2010	-0.6603	awesom
7	1.4021	descript	-0.6597	class
8	1.3764	2012	-0.6244	int
9	1.3739	2005	-0.5735	font
10	1.2894	2008	-0.5710	var
11	1.2882	2011	-0.5644	project
12	1.2730	2014	-0.5378	00
13	1.2515	2009	-0.5144	that
14	1.2122	40	-0.5117	div
15	1.2114	inc	-0.5086	io
16	1.1896	state	-0.5076	end
17	1.1846	07	-0.5016	unicod
18	1.1290	xml	-0.4945	href
19	1.0300	non	-0.4927	content
20	0.9655	04	-0.4904	set

Table 44: Classifier on source code - DEV category

	Positive words		Negative words	
1	3.1477	return	-1.0062	color
2	2.0089	self	-0.9998	your
3	1.7739	valu	-0.9610	awesom
4	1.6851	void	-0.9015	www
5	1.6622	end	-0.7908	href
6	1.5736	expect	-0.7666	titl
7	1.3537	param	-0.7323	you
8	1.3535	err	-0.7275	data
9	1.3212	bool	-0.7042	span
10	1.2467	int	-0.6731	margin
11	1.1633	none	-0.6638	width
12	1.1606	error	-0.6593	border
13	1.0754	path	-0.6587	10
14	1.0457	assert	-0.6542	background
15	1.0321	object	-0.6538	work
16	1.0276	static	-0.6370	css
17	1.0133	env	-0.6341	descript
18	0.9909	defin	-0.6152	right
19	0.9791	test	-0.5796	code
20	0.9308	do	-0.5735	left

Table 45: Classifier on source code - DOCS category

	Positive words		Negative words	
1	5.5009	awesom	-2.8218	return
2	4.1666	your	-2.2221	class
3	2.7342	work	-2.1735	type
4	2.7034	www	-2.0449	var
5	2.6186	ani	-2.0291	function
6	2.6072	suggest	-1.7285	color
7	2.5517	guidelin	-1.6960	string
8	2.5331	contribut	-1.5913	int
9	2.3799	make	-1.4093	id
10	2.2461	project	-1.3987	valu
11	1.9944	link	-1.3724	div
12	1.9439	org	-1.2713	self
13	1.9311	tool	-1.2642	li
14	1.8714	request	-1.2452	import
15	1.8289	softwar	-1.1437	null
16	1.7722	pull	-1.1174	data
17	1.7110	md	-1.1159	00
18	1.6576	right	-1.0825	border
19	1.6174	categori	-1.0501	end
20	1.5835	readm	-1.0185	left

Table 46: Classifier on source code - EDU category

	Positive words		Negative words	
1	2.1851	00	-1.2746	awesom
2	2.0472	metadata	-0.9354	var
3	1.8166	materi	-0.7913	descript
4	1.6193	begin	-0.7786	color
5	1.5528	import	-0.7625	return
6	1.4113	slide	-0.7439	function
7	1.3672	android	-0.7303	request
8	1.1888	class	-0.7063	border
9	1.0836	item	-0.6768	margin
10	1.0731	we	-0.6661	contribut
11	1.0380	125	-0.6621	your
12	0.8663	id	-0.6425	org
13	0.8655	set	-0.6325	unicod
14	0.8428	frame	-0.6027	suggest
15	0.8401	def	-0.5759	io
16	0.8307	fill	-0.5723	html
17	0.8305	print	-0.5664	guidelin
18	0.7886	markdown	-0.5618	make
19	0.7755	output	-0.5172	work
20	0.7733	sourc	-0.5092	md

Table 47: Classifier on source code - HW category

	Positive words		Negative words	
1	3.0063	solut	-0.9607	awesom
2	2.5122	int	-0.8005	type
3	1.8493	string	-0.7972	www
4	1.5194	number	-0.7750	org
5	1.4713	assign	-0.7390	licens
6	1.4402	end	-0.7018	descript
7	1.3307	log	-0.6876	font
8	1.3193	system	-0.6227	li
9	1.2793	javascript	-0.6089	href
10	1.2023	practic	-0.5443	titl
11	1.0704	css	-0.5419	link
12	1.0301	help	-0.5204	span
13	1.0222	repositori	-0.5163	contribut
14	1.0132	col	-0.5056	path
15	1.0104	key	-0.5001	null
16	0.9712	integ	-0.4944	code
17	0.9700	consol	-0.4750	style
18	0.9571	10	-0.4540	guidelin
19	0.8922	self	-0.4522	td
20	0.8702	problem	-0.4509	highlight

Table 48: Classifier on source code - WEB category

	Positive words		Negative words	
1	4.2494	color	-2.0881	int
2	3.6096	function	-1.8096	awesom
3	3.3109	href	-1.7540	end
4	3.1891	li	-1.7066	self
5	2.9029	div	-1.4745	string
6	2.8672	var	-1.4446	your
7	2.8131	margin	-1.3400	ani
8	2.7799	border	-1.1376	request
9	2.6863	span	-1.1149	def
10	2.5794	background	-1.1016	www
11	2.4140	font	-1.0727	public
12	2.3782	left	-1.0443	01
13	2.3198	unicod	-1.0328	licens
14	2.1168	highlight	-1.0262	softwar
15	1.8167	top	-1.0164	org
16	1.7787	width	-0.9795	make
17	1.6578	content	-0.9688	should
18	1.5868	pad	-0.9652	solut
19	1.5299	webkit	-0.9300	file
20	1.4863	height	-0.9266	sourc

Table 49: Boolean Matrix for Validation Data

Label	Predicted Correctly	Predicted Incorrectly	Precision	Recall
WEB	4	0	0.27	1.00
DOCS	4	0	0.40	1.00
HW	0	5	0.00	0.00
DEV	3	8	0.60	0.27
EDU	0	4	0.00	0.00
DATA	1	2	1.00	0.33
Weighted Average			0.40	0.39

Table 50: Boolean Matrix for Additional Validation Data

Label	Predicted Correctly	Predicted Incorrectly	Precision	Recall
WEB	35	5	0.21	0.88
DOCS	13	23	0.17	0.36
HW	2	41	0.67	0.05
DEV	17	39	0.49	0.30
OTHER	0	57	0.00	0.00
EDU	7	28	0.39	0.20
DATA	5	28	0.83	0.15
Weighted Average			0.37	0.26