How to README:
Classification of GitHub README Sections

Databricks Notebook:
https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/4985770712986055/1653466948958185/3712725817297160/latest.html

Evan K. Boerchers
Hunter S. Kimmett
Do Trong Anh

0 Individual Contributions

This is an overview of each team member's contributions to the project.

0.1 Do

I was responsible for performing a stage of data preprocessing, which is transforming non-text elements in the headers and sections contents into abstract tags. Additionally, I also used pandas to collect and filter misclassified sections, and contributed to creating evaluation functions, and also tuned the models' parameters to a reasonably optimal range.

In terms of data collection I collected about 200 sections from 20 README files, and labeled ⅔ of the new data, as well as labelled ⅓ of the misclassifications

In regards to the report I wrote section 3.4 and 3.5.

0.2 Evan

At the start of the project I was responsible for developing two crucial algorithms. The first algorithm was to parse a README and then extract the sections this algorithm is contained in heading_extractor.py. The second algorithm was used to extract each section's contents, this algorithm is contained in section_extractor.py. I started the databricks workbook and created the needed data frames from our extracted csv files. Additionally I determined statistics like avg. sections per README and avg terms per sections, as well as the percent agreement/cohen kappa of our labeling.

In terms of data collection I collected around 170 sections and labelled ⅔ of the new data. I also labeled ⅓ of the misclassifications.

In regards to the report I wrote section 2: Introduction, my potion of discussion as well as sections 3.1 and 3.3.

0.3 Hunter

I was mainly responsible for performing the feature extraction for both Statistic and Heuristic features. The segments of code that included Tokenization, Stop Words, Tags, Stemming, Vectorization, IDF, Binarization for the Statistic feature methods, as well as the 54 heuristic methods. In addition to this I also assisted with performing the first untuned ML classifications and minor help with the tuning and scoring of the next ML models. I also helped make fixes on the misclassification section.

For data collection I collected around ⅓ of the sections and labelled ⅔ of them in conjunction with my teammates, as well as ⅓ of the misclassifications.

For the report, I wrote the Abstract, Section 3.2, Section 3.6, my portion of Discussion, and the Conclusion.

1 Abstract

Context

GitHub serves as a large repository containing the programs of millions of users and their programs. README files are often included in GitHub repositories and are critical to a user's understanding of how a program functions. One of their most important tasks is to inform the user on how to run their program.

Objective

In this paper we will try to replicate and improve the results of the paper *Categorizing the Content of GitHub README Files*[1] for categorizing README files that contain how to run their program.

Method

We will try to improve the results of this paper by expanding on their dataset and tuning their best machine learning algorithm (Linear Support Vector Machine Classifier) and a new machine learning algorithm (Gradient Boosted Tree Classifier) .

Results

The Gradient Boosted Tree Classifier performed adequately but could not match the results of the paper even after adding our new dataset and tuning its hyperparameters. The Linear Support Vector Machine Classifier however was able to outperform the paper's classifier on both the original dataset and our new dataset.

Conclusion

Through this paper it is demonstrated that tuning and the addition of more data can improve the results of a machine learning algorithm's predictions of if a README file explains how a program works.

2 Introduction

Github is one of the most if not popular coding collaboration platforms. The platform itself contains millions of repositories. A critical aspect of understanding these diverse projects is the README file that comes along with most projects. In order to understand the project the user generally has to manually sift through sections and sections of material in order to find what they want. Generally the most important information needed is in regards to how the software project is to be used by the user. For this reason classification of README sections into how and not how would be extremely useful to find out the most pertinent information as fast as possible. In this project we will use big data methodology and machine learning to create a model that can predict which sections of READMES are how and not how. The methodology of this report is inspired by the 2018 paper, *Categorizing the Content of GitHub README Files* [1]. In this paper README files sections were classified into What, Why, How, When, Who, References, Contribution, and Other. In this report we will use their data as well as supplement it with 500 new records.

Data was collected using the github API and then labelled manually by at least two labellers. Ties were broken by a third labeler. To compare new and original data we obtained statistics of average sections per README and average terms per section.

Once data was obtained with section contents/headings extracted and labeled, the data was preprocessed. This preprocessing included an abstraction process that removed code, hyperlinks etc. and replaced them with tags. After this procedure the resulting contents were tokenized, stop words removed, stemmed and then vectorized using TF-IDF. Heuristic features were also generated. The resulting features were then used to train both a LinearSVC and a GradientBoostedTree Classifier. The hyperparameters C and max depth were adjusted for each model respectively eventually leading us to the best model.

With a working model achieved predictions were made on the test set. The resulting misclassifications were then given reasons for their misclassification in the form of 4 labels: 0 - model mistake, 1 - labeler mistake, 2 - section to short and 3 - section resembles opposite class.

3 Methodology and Results

The results of our README data collection, processing, and machine learning outcomes are outlined in this section.

3.1 How was the new data labeled/collected?

To start off new records needed to be collected and then labelled to supplement the original data and provide a comparison to verify our procedure with.

3.1.1 Approach

In order to ensure that our new data would match the old data all steps were taken to ensure that the collection methodology would remain the same. For this reason the github api call: https://api.github.com/repositories?since={}'.format(r_int) was used to find random repositories. Repositories were browsed manually, and each README was checked to ensure it matched the criteria, if the README fit this criteria it was added to our collection.

Criteria:

1. Is in English
2. In regards to software
3. Is a Markdown file
4. Has proper header formatting (i.e either # or ---/=== underline formatting)
5. More than 4 sections

Once enough READMES were collected each README needed to be broken up into sections. To accomplish this a python algorithm was used to detect headers in each readme file. The output of this algorithm was a CSV file with a list of all sections in every README obtained.

With the sections obtained they could then be manually labeled, 1 for how and 0 for not how. Our approach to this was to have two labelers manually parse each record and apply a label to the given section.

3.1.2 Results

From the collection phase 48 README files were harvested containing a total of 506 samples.

Given that two labellers were to label every record, the possibility of disagreement existed. Upon the completion of the initial labeling the accuracy of our labeling could be evaluated by comparing both labels with the metrics percent accuracy and the cohen kappa score. These metrics for our labeling is shown below:

Percent Agreement = 0.83

Cohen Kappa Score = 0.66

It can be seen that there are discrepancies in labeling. In order to resolve this for every record that had a labelling disagreement a third labeller would break this tie, effectively giving each record a certain label.

3.2 How does the newly added data compare with the original data?

Before adding our newly imported data to the original dataset, we measured several different metrics to compare the two sets of data.

3.2.1 Approach

In order to compare the two datasets the two following metrics were calculated:

1. Average number of sections per README file. This is calculated by counting the total number of sections in each data frame and dividing it by the count of unique values in the “url” column.
2. Average number of terms per section. This is calculated by counting the tokens in each section’s body text and dividing it by the count of rows in the data frame.

3.2.2 Results

The following table shows the metrics that were calculated to compare datasets:

	Original Data	New Data
Avg. Sections per README	10.75	10.45
Avg. Terms per Section	64.58	97.84

The average number of sections per README file was relatively similar, however the average number of terms per section both before and after abstraction were significantly different. Our new data had approximately 50% more terms in each section. The main factor that influenced this disparity is likely the small sample size of our new data. By only using 48 README files in comparison to the approximately 400 from the paper there can be a lot of variance introduced. In going through our collected READMEs it seems our results could have been biased by a couple of files that had extremely long and verbose sections, which would explain this difference.

3.3 How was the data preprocessed?

Prior to the preprocessing step each section was acquired with its appropriate label. The next step was to extract the contents of each section. To do this a python script similar to the header extractor was used. The results of this led to a dataframe with the necessary header, contents and label info needed for preprocessing.

3.3.1 Approach

There are 3 stages to the preprocessing of our data:

1. Abstraction:
   This portion of preprocessing will convert certain content segments into abstract tags, these tags include:

   - Code blocks into '@abstr_code_section'.
   - Numbers into '@abstr_number'.
   - Hyperlinks into '@abstr_hyperlink'.
   - Mail links into '@abstr_mailto'.
   - Embedded images into '@abstr_images'.

   Additionally, headings and local hyperlinks were replaced with their plain text version.

2. Statistical Feature Extraction:
   Statistical features consist of deriving statistics from the section heading+contents. The steps for obtaining these features are as follows:

   1. Raw feature extraction: Tokenizations removing stop words, removing tags, stemming.
   2. TF-IDF: Vectorization, IDF, Binarization.

3. Heuristic Feature Extraction
   Heuristic features consisted of search for specific content patterns/terms, for example the term 'you must' could indicate a how section. Each heuristic feature was checked in the contents and heading, providing a 1 if this feature existed and a 0 if it didn't.

3.3.2 Results

The results of the abstraction and preprocessing steps can be seen in our databricks code. Additionally the statistic of average terms per section can be seen to have changed after preprocessing as shown by the table below.

	Original Data	New Data
Avg. Terms per Section (post abstraction)	54.25	92.90

This is because contents such as code blocks have been replaced with the respected abstract tag.

3.4 How do the models perform on the original data vs the new+original data?

After the original and combined datasets have been pre-processed into useful features, the features will then be vectorized using VectorAssembler, transforming all features into one column of sparse vectors, which, along with the label column, can be used for model fitting.

3.4.1 Approach

- Preprocessed datasets are vectorized via VectorAssembler using all statistical features columns for both datasets, which transformed into a "features" column containing sparse vectors representative of those statistical features.
- Both datasets are then split into training and test sets randomly at a ratio of 9 : 1.
- The training sets are then fitted into the LinearSVC and GBTClassifier model respectively. Then we used those fitted models to transform the test sets into labels to be compared with the ground truth labels.

3.4.2 Results

After training our LinearSVC and GBTClassifier the validations scores shown in the below table were achieved:

		Original dataset	Combined dataset
Linear C Support Vector Machine	Accuracy	0.83	0.86
	Precision	0.89	0.91
	Recall	0.83	0.86
	F1 score	0.86	0.89
Gradient Boosted Tree	Accuracy	0.79	0.79
	Precision	0.92	0.83
	Recall	0.77	0.82
	F1 score	0.84	0.82

Note that these models contained the default hyperparameters.

3.5 How does the performance of the models change based on the choice of hyperparameters

In order to improve our model scores hyperparameters needed to be adjusted. For this purpose we made the decision to tune the regularization parameter C of the LinearSVC model and the max_depth parameter of the GBTClassifier model, which basically determine how strict/loose these models will be in accommodating new data.

3.5.1 Approach

- LinearSVC model will run on both original and combined datasets with parameter C in [0.001, 0.005, 0.01, 0.1]
- The GBTClassifier model will also run on these datasets with parameter max_depth in [5, 6, 7, 8].

3.5.2 Results

Each model was tuned with a given hyperparameter and trained on the data, giving validation scores shown in the table below:

		Original data				Combined data			
		Acc	Pre	Rec	F1	Acc	Pre	Rec	F1
LinearSVC	C=0.001	0.84	0.90	0.84	0.87	0.86	0.92	0.86	0.89
	C=0.005	0.84	0.90	0.84	0.87	0.87	0.92	0.86	0.89
	C=0.01	0.84	0.91	0.84	0.87	0.86	0.92	0.86	0.89
	C=0.1	0.84	0.91	0.84	0.87	0.86	0.92	0.85	0.88
GBT	d=5	0.79	0.92	0.77	0.84	0.79	0.83	0.82	0.82
	d=6	0.83	0.84	0.87	0.85	0.79	0.83	0.82	0.82
	d=7	0.82	0.83	0.87	0.85	0.81	0.83	0.85	0.84
	d=8	0.82	0.82	0.87	0.84	0.81	0.84	0.84	0.84

3.6 How are the misclassifications of the best performing model distributed?

In order to identify misclassifications, we exported a data frame of misclassified samples in order to manually classify each misclassification into 4 categories.

3.6.1 Approach

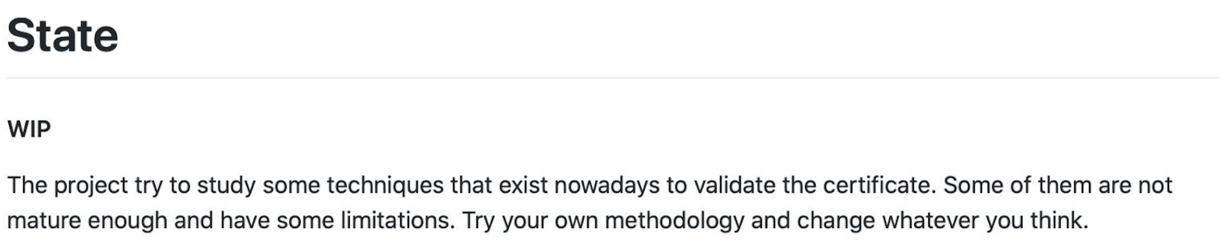
After using our highest-performing machine learning method (LinearSVC) on our combined dataset, we created a data frame with all predicted data. This data frame had columns with the header, body, actual label, and predicted label. We then dropped all rows from this data frame that had actual labels and predicted labels that matched. This data frame was then exported to a .csv file, which each group member would go through to classify misclassification into one of 4 categories. The categories are as follows:

- Label 0, Model Mistake: For when there is no perceptible explanation as to why the model failed.
- Label 1, Labeller Mistake: For when the section was misclassified by the labeller.
- Label 2, Section Too Short: For when the section lacks sufficient content to be predicted properly
- Label 3, Section Resembles Opposite Class: For when the section contains words that could be related to the opposite classification.

3.6.2 Results

The following are examples of sections that were misclassified, and what type of label we gave to this misclassification.

Model Mistake



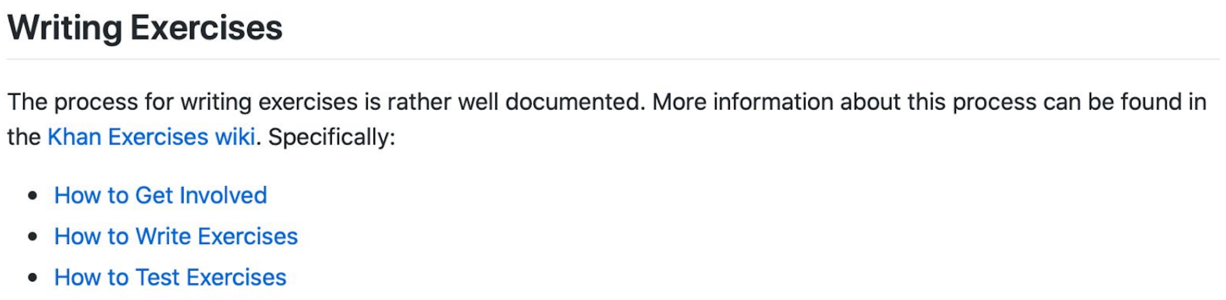
State

WIP

The project try to study some techniques that exist nowadays to validate the certificate. Some of them are not mature enough and have some limitations. Try your own methodology and change whatever you think.

The model classified this as How, where it clearly is not How and was labelled as such. It does not appear to have similar wording to a How section so it is classified as a Model Mistake.

Labeller Mistake



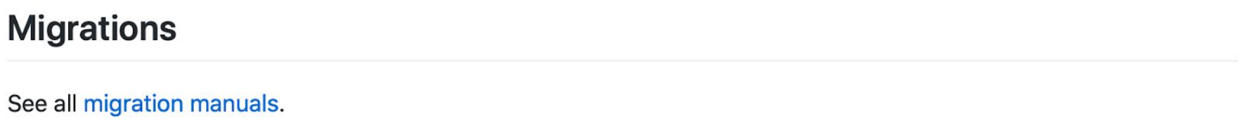
Writing Exercises

The process for writing exercises is rather well documented. More information about this process can be found in the Khan Exercises wiki. Specifically:

- How to Get Involved
- How to Write Exercises
- How to Test Exercises

This section was labelled as a How section and was classified as not How. By the criteria outlined in the paper, this section should have been labelled as not How. Thus this is a Labeller Mistake.

Section Too Short

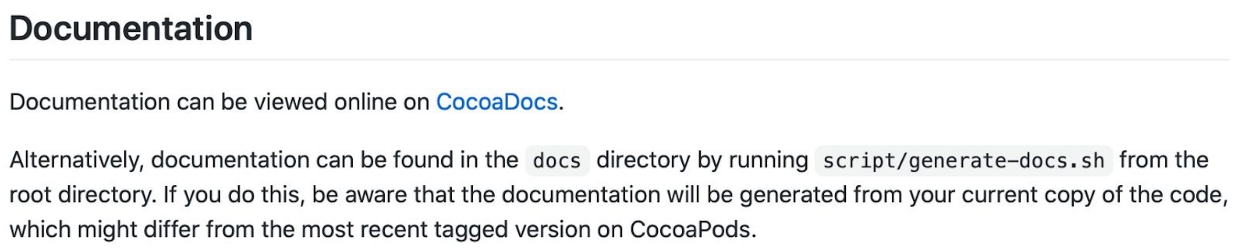


Migrations

See all migration manuals.

This section was labelled as a not How section but was classified as How. This section has such little content it is difficult for the model to make a proper prediction, thus it is classified as Section Too Short.

Section Resembles Opposite Class



Documentation

Documentation can be viewed online on CocoaDocs.

Alternatively, documentation can be found in the `docs` directory by running `script/generate-docs.sh` from the root directory. If you do this, be aware that the documentation will be generated from your current copy of the code, which might differ from the most recent tagged version on CocoaPods.

This section was classified as a How section, when it was labelled as a Not How section. This section has a lot of words and abstract code that makes it resemble a section that would be How, but it is not actually a How section. Thus, the misclassification can be classified as Section Resembles Opposite Class.

After categorizing all misclassifications the following statistics were gathered:

Label	Misclassification Reason	Percentage of Misclassifications
0	Model Mistake	22.79 %
1	Labeller Mistake	15.44 %
2	Section Too Short	38.24 %
3	Section Resembles Opposite Class	13.97 %

Through our observations it is clear that one of the main contributors to errors made by the model is sections that are too short. This would be difficult to compensate for in the model, as less data decreases its accuracy.

4 Discussion

The models work fairly well and can be useful in determining a markdown document of its type of content as well as its applied rhetorics. However the models only show their effectiveness in formal, well structured documents that provide ample contexts in their sections. In order for the models to perform well on documents of informal languages and unorthodox structures, there may be a need for better methods for preprocessing and data collection.

4.1 Real World Scenarios

Scenarios for potential application of this project's results were determined by each member.

4.1.1 Do

A potential scenario where a document classification model in general can be useful is in the machine learning on human languages. Based on the document sections and its components, a model can be trained to determine the degree of formality as well as the rhetorics used by the document's author. An advanced A.I model can use these results to mimic written languages and recognize/ replicate speech patterns. This could potentially be further developed into an official document assessment system, where A.Is can automatically categorize your documents and give them scores based on readability, coherence, comprehensiveness, etc...

4.1.2 Evan

A potential use for this project is by GitHub themselves. Once of the most important aspects of a github readme is finding out how to install and run whatever software the project contains. Github could implement our model in order to auto tag sections as 'how'. This way users of there site can filter any readme to contain only the 'how' contents, making navigation more efficient.

4.1.3 Hunter

A way for this to possibly be used would be for a text editor or IDE that is often used for writing for README files. When editing a README if the user does not have a section explaining how the code works the program could flag or notify the user that their README may be less useful if they do not include a section explaining how their project works. This could be triggered when they save or export the file, or could be a warning in the corner.

5 Conclusions

By expanding on the data from our chosen paper [1] our team was able to classify README file sections as either containing how the program works, or not containing this using a machine learning algorithm. Using a similar technique with improved hyperparameter tuning we were able to beat the F1 scores achieved by the paper by a small margin with our F1 score on the original dataset being 1% higher, and 3% higher with our new data. It may be possible for there to be an application for this machine learning method, and that by adding more data the model's scores would improve to an even greater extent.

6 References

[1] G. A. A. Prana, C. Treude, F. Thung, T. Atapattu, and D. Lo. Categorizing the Content of GitHub README Files. Empirical Software Engineering, vol. 24, no. 3, pp. 1296–1327, 2018.