Categorizing the Content of GitHub README Files

BHAVYAI GUPTA, University of Calgary, Canada

KAYODE AWE, University of Calgary, Canada

MICHAEL MAN YIN LEE, University of Calgary, Canada

SUMMARY OF CONTRIBUTIONS:

1.1 Data Collection

Bhavyai developed script that downloads the README.md file from random GitHub repositories using GitHub API. The script was initially written as `.py` script, and then later modified to run on Databricks. Once few hundred README.md files are downloaded, **Michael** selected 86 files with a total of 1000 sections for manual annotation. Each of the author of this project annotated all the 1000 sections. **Kayode** is the facilitator to make sure we reach an agreement of what final manual annotations should be used as well as calculating the statistics such as the Cohen Kappa analysis to show our overall process. **Michael** is responsible for extracting the sections from README files and to feed the new annotated data back to the database for modelling and analysis.

1.2 Coding

Bhavyai is responsible for developing script to download all new README files needed for this project. **Michael** is responsible for the initial draft of modifying the original code from the author to work on databricks.

Kayode is responsible for creating the notebook for comparing the results of the old and old plus the new annotated data using the original research models.

Michael is responsible for creating the notebook for comparing the results of the old and new annotated datasets using previously unexplored models.

Bhavyai is responsible for creating the notebook for hyperparameter optimization of the models.

1.3 Writeup

The breakdown of the report writeup work can be found in the table below:

Section	Resource
Page 1 contents	Michael
Abstract/Introduction/Conclusion	Bhavyai

How was the new data labeled/collected?	Kayode
How does the newly added data compare with the original data?	Kayode
How was the data preprocessed?	Bhavyai
Preparing the model on Databricks with old + new data	Michael
How do the models perform on the original data vs the new + original data?	Michael
How does the performance of the models change based on the choice of hyper parameters?	Michael
How are the misclassifications of the best performing model distributed?	Bhavyai/Kayode
Discussion	All

1.4 Databricks notebooks

1. README downloader
   https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/279226329024279/3713457635192600/4593657877666368/latest.html

2. Original Model Comparison
   a. ENSF612 Final Project with Original Models - Old Data: https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/4199039841175524/4424314020178626/1115343936761319/latest.html
   b. ENSF612 Final Project with Original Models – Old + New Data: https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/4199039841175524/4424314020178662/1115343936761319/latest.html

3. New Model Comparison:
   a. ENSF612 Final Project with Unexplored Models - Old Data: https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/4199039841175524/4424314020178557/1115343936761319/latest.html
   b. Histogram Gradient Boost – Old Data: https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/4199039841175524/4424314020178505/1115343936761319/latest.html
   c. Histogram Gradient Boost – Old+New Data: https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/4199039841175524/4424314020178533/1115343936761319/latest.html

4. Hyperparameter Optimization
   https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/279226329024279/1055332186 68830/4593657877666368/latest.html

1 ABSTRACT

1.1 Context

README files play an essential role in shaping a developer's first impression of a software repository and in documenting the software project that the repository hosts. Yet, we lack a systematic understanding of the content of a typical README file as well as tools that can process these files automatically.

1.2 Objective

In this project, we study how the understanding of content of a typical README can be improvised by categorizing the sections of README files using a multi-label classifier.

1.3 Methods

We replicate the research paper on "Categorizing the Content of GitHub README Files" [1]. We also extend the work done in the paper by running more ML classifier models. We also tune hyper-parameters of a few models to explore better performance.

1.4 Results

We find that the LinearSVC classifier used by the authors in their research paper is still the best performing model on default hyperparameters. The weighted F1 score of the LinearSVC we achieved is 0.721.

1.5 Conclusions

Through this work, we enable the owners of software repositories on sites such as GitHub to improve the quality of their documentation, and to make it easier for the users of the software held in these repositories to find the information they need.

2 INTRODUCTION

The README.md file for a repository on GitHub is often the first project document that a developer will see when they encounter a new project. This first impression is crucial.

With more than 25 million active repositories at the end of 20171, GitHub is the most popular version control repository and Internet hosting service for software projects. When setting up a new repository, GitHub prompts its users to initialize the repository with a README.md file which by default only contains the name of the repository and is displayed prominently on the homepage of the repository.

However, up to now and apart from some anecdotal data, little is known about the content of these README files.

To close this gap, the authors of this project manually annotated 1000 sections belonging to 86 README files. This manually annotated data is used in adding to the 4,226 annotations done by the authors of the research paper. This annotation provides the extended large scale empirical data on the content of GitHub README files.

In addition to the annotation, we use the classifiers used in the research paper and a set of features to predict categories of sections in the README files. We extend the work in the research paper by exploring more classifiers and tuning hyperparameters of some of the classifiers.

Background

GitHub is a code hosting platform for version control and collaboration.4 Project artifacts on GitHub are hosted in repositories which can have many branches and are contributed to via commits. Issues and pull requests are the primary artifacts through which development work is managed and reviewed.

Due to GitHub's pricing model which regulates that public projects are always free, GitHub has become the largest open source community in the world, hosting projects from hobby developers as well as organizations such as Adobe, Twitter, and Microsoft.

Each repository on GitHub can have a README file to "tell other people why your project is useful, what they can do with your project, and how they can use it." README files on GitHub are written in GitHub Flavored Markdown, which offers special formatting for headers, emphasis, lists, images, links, and source code, among others. In 2017, 25 million active repositories were competing for developers' attention, and README files are among the first documents that a developer sees when encountering a new repository.

3 RESULTS

3.1 How was the new data labeled/collected?

The new data was collected by developing a script which downloads README.md files from GitHub using GitHub API. The databricks version of this script is linked in the summary section 1.4.1. The downloaded files were randomly chosen and unique. We ensured that the files that were downloaded belong only to the software development repository and have size that is greater than 2KB. Manual filtering was done to remove non-English readme files. There is a limit to the number of readme files can be download from GitHub. The default maximum is 60 request per hour. In order the download more GitHub files for the project, we made use of Personal Access Token (PAT). This allowed us access to download up to 5000 request per hour. With this we were able to collect enough README files for the project.

Manual annotation was carried out on 1089 sections of the new dataset. We followed the same method used by the original authors to manually label each section of the README file in eight different categories. Each person in the team manual carried out the labelling separately. The results of the manual labelling were fed into IBM's SPSS software to compute Cohen Kappa inter-rater agreement metric. The result of the computation returns an agreement of 0.941 as shown below.

Though we had a very good metric in terms agreement, we still went ahead to carefully analyzed the area of disagreement and we were able to select the best categorization for each section where we disagreed based on majority opinion.

Symmetric Measures

		Value	Asymptotic Standard Error[a]	Approximate T[b]	Approximate Significance
Measure of Agreement	Kappa	.941	.009	56.889	.000
N of Valid Cases		1089			

a. Not assuming the null hypothesis.

b. Using the asymptotic standard error assuming the null hypothesis.

Figure 1: Summary of disagreement opinion statistics

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

The new set data followed the same pattern as the original data. The table below shows the distribution when the old data and the new data were compared.

Section Type	Original files count	% Count	New file sections	% count
What	707	14.67%	216	20.21%
Why	116	2.41%	0	0.00%
How	2467	51.18%	540	50.51%
When	180	3.73%	30	2.81%
Who	322	6.68%	73	6.83%
References	858	17.80%	179	16.74%
Contributions	112	2.32%	26	2.43%
Other	58	1.20%	5	0.47%

Table 1: Distribution of old data and new data

We can clearly observe that sections on “How” have the highest percentage in both set of that. This is not surprising as developers tends to spend more time on explaining how to; run, install, update, set up download and fix errors. This explains why we have around 50% counts for the “how” section in each set of data. In contrary, the sections that fall under “other” have the least count as indicated by the table. The reason for this is that a README file section will only be categorized as “other” if there are no matching keywords used in determining the other categories.

3.3 How was the data preprocessed?

There are two pre-processing performed on the data. The headings and contents of the readme sections are abstracted to their types. Then this abstracted data is tokenized and stop words are removed.

Content abstraction abstracts contents to their types. We abstract the following types of section content: hyperlink, code block, image, and numbers. Each type is abstracted into a different string (@abstr_hyperlink, @abstr_code_section, @abstr_image and @abstr_number, respectively). Such abstraction is performed since for classification, we are more interested in existence of those types in a section than its actual content.

Type	Abstracted text
Code snippets	@abstr_code_section
Numbers	@abstr_number
Images	@abstr_image
Hyperlinks	@abstr_hyperlink
mailto links	@abstr_hyperlink

Figure 2: Summarizing the abstraction types

This abstraction is followed by tokenization, which converts a section into its constituent words, and English stop word removal. For the stop word removal, we use the stop words provided by scikit-learn.

We also encode our target readme section 1 to 8 into a matrix of 0s and 1s using MultiLabelBinarizer. The output of the MultiLabelBinarizer looks like Figure 2.

```
Section categories
['-', '1', '3', '4', '5', '6', '7', '8']

Encoded section categories in the sample readme file
[[0 0 1 0 0 0 0 0]
 [0 0 0 0 0 0 0 1]
 [0 1 1 0 0 0 0 0]
 [0 0 1 0 0 0 0 0]
 [0 0 1 0 0 0 0 0]
 [0 0 0 0 0 0 0 1]]
```

Figure 3: Sample output of a MultiLabelBinarizer

After pre-processing, we extract features from the data. We extract two kind of features – statistical features and heuristic features.

For statistical features, we count the number of times a word appears in each section. This is called the Term Frequency (TF) of a word in a section. If there are n words that appear in the set of sections used for training the classifier (after preprocessing), we would have n statistical features for each section. If a word does not appear in a section, then its TF is zero. We also compute the Inverse Document Frequency (IDF) of a word. IDF of a word is defined as the reciprocal of the number of sections in which the word appears. We use a multiplication of TF and IDF as an information retrieval feature for a particular word. A TF-IDF matrix from a sample README file is shown in Figure 3.

	abstr_code_section	abstr_hyperlink	abstr_image	abstr_number	...	use	using	vcs	version
0	0.000000	0.326924	0.000000	0.490387	...	0.000000	0.000000	0.000000	0.23611
1	0.000000	0.000000	0.000000	0.748083	...	0.000000	0.000000	0.221518	0.00000
2	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.00000
3	0.000000	0.328074	0.473882	0.328074	...	0.000000	0.000000	0.155436	0.00000
4	0.704681	0.000000	0.000000	0.000000	...	0.140936	0.281872	0.000000	0.00000
5	0.000000	0.860429	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.00000

[6 rows x 54 columns]

Figure 4: TF-IDF matrix of a sample readme file

For heuristic features, we replicate the functions used in the research paper that extract 55 binary linguistic patterns within a category of sentences to derive heuristics that can aid classification. These heuristic patterns are categorized into 4 types, namely – Linguistic patterns, Single word non English heading, repository name, and non-ascii content text. A screenshot that captures some of the heuristic features extracted from a sample readme file is below

	0	1	2	3	4	5
heur_c_k_001	0	0	0	0	0	0
heur_c_k_002	0	0	0	0	0	0
heur_c_k_003	0	0	0	0	1	0
heur_c_k_004	0	0	0	0	0	0
heur_c_k_005	0	0	0	0	0	0
heur_c_k_006	0	0	0	0	0	0
heur_c_k_007	0	0	0	0	0	0
heur_h_k_001	0	0	0	0	0	0
heur_h_k_002	0	0	0	0	0	0
heur_h_k_003	0	0	0	0	0	0
heur_h_k_004	0	0	0	0	0	0
heur_h_k_005	0	0	0	0	0	0
heur_h_k_006	0	0	0	0	0	0
heur_h_k_007	0	0	0	0	0	0
heur_h_k_008	0	0	0	0	0	0
heur_h_k_009	0	0	0	0	0	0
heur_h_k_010	0	0	0	0	0	0
heur_h_k_011	0	0	0	0	0	0
heur_h_k_012	0	0	0	0	0	0
heur_h_k_013	0	0	0	0	0	0
heur_h_k_014	0	0	0	0	0	0
heur_h_k_015	0	0	0	0	0	0

Figure 5: Part of 55 heuristic features of a sample readme file

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

With the original data, we have performed model analysis of the weighted average of precision, recall and f1 score for the following models:

	Weighted Avg		
Model Name	Precision	Recall	f1
RandomForestClassifier	0.773	0.684	0.707
LinearSVC	0.709	0.738	0.721
GaussianNB	0.402	0.662	0.492
LogisticRegression	0.593	0.607	0.566
BaggingClassifier	0.634	0.731	0.673
ExtraTreesClassifier	0.813	0.636	0.692
DecisionTreeClassifier	0.619	0.704	0.652
AdaBoostClassifier	0.558	0.801	0.655
HistGradientBoostingClassifier	0.663	0.779	0.711

Table 2: Summarizing scores on all the models on original data

The first four models, namely RandomForestClassifier, LinearSVC, GaussianNB and LogisticRegression were used in the original paper.

The last five models, BaggingClassifier, ExtraTreesClassifier, DecisionTreeClassifier, AdaBoostClassifier and HistGradientBoostingClassifier are models that have not been experimented by the original paper. Note that initially the original GradientBoost model is used but Databricks terminated the cluster before the model can finish running. HistGradientBoostingClassifier is also significantly longer to complete than the other models (took 5 hours compared to 2.35hrs for RandomForest and 11.5 minutes for LinearSVC), but it can successfully complete with the scores calculated.

As can be seen from the previous table, ExtraTrees classifier has the highest precision whereas AdaBoostClassifier has the highest Recall.

Recall that Precision = (True Positive/Total Predicted Positive) and Recall = (True Positive)/(Total Actual Positive), Extra Trees Classifier is the best model to use when the cost of any false positive is high and AdaBoost should be used if we want to filter out the most false negatives.

Now, for overall performance we should instead look at what we call the f1 score, which is a function of Precision and Recall where the formula is:

F1 = 2*(Precision*Recall)/(Precision + Recall)

From the table, the top 3 models that have the highest weighted average f1 score are: LinearSVC (f1 score = 0.721), RandomForestClassifier (f1 score = 0.707) and HistGradientBoost (f1 score = 0.711).

For the new combined data, due to additional processing time required for the addition of the new dataset, we only perform validation on the top 3 models with the highest f1 score:

	Weighted Avg					
Model Name	Precision Old	Precision New	Recall Old	Recall New	f1 Old	f1 New
LinearSVC	0.709	0.7	0.738	0.74	0.721	0.716
RandomForestClassifier	0.773	0.763	0.684	0.68	0.707	0.7
HistGradientBoost	0.663	0.653	0.779	0.781	0.711	0.706

Figure 6: Summarizing old and new scores of top three models

As can be seen, the Precision, Recall and f1 scores drop slightly when additional data are added, but they do not affect the order of the performance ranking of these models.

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

Attempts have been made to explore the hyperparameters for the project. We first explore the performance of the LinearSVC model based on the choice of hyper parameters. With LinearSVC, we explored the regularization parameter,

or the C parameter of the model. We will explore the effects on the model when parameter C is set to 0.001, 1, 100 and 10000.

For the purpose of conducting gridsearch, we developed a new method perform_grid_search as shown in Figure 7.

Grid Search to tune HyperParameters

```
def perform_grid_search(model, parameter_grid):
  '''
  Performs the grid search
  Prints the summary of of best model
  Returns the grid search object
  '''

  grid_search = GridSearchCV(model, parameter_grid, n_jobs=4, cv=10, scoring='average_precision', return_train_score=True)
  grid_search.fit(features_combined.values, labels_matrix)

  print("Best parameters   = {}".format(grid_search.best_params_))
  print("CV Training Score = {:.3f}".format(grid_search.cv_results_['mean_train_score'].max()))
  print("CV Test Score     = {:.3f}".format(grid_search.cv_results_['mean_test_score'].max()))

  return grid_search
```

Figure 7: Screenshot of grid search function

However, it seems like only the default C value (1) can be successfully completed as the other C values fails to converge:

```
  warnings.warn("Liblinear failed to converge, increase "
/databricks/python/lib/python3.8/site-packages/sklearn/svm/_base.py:985: ConvergenceWarning: Liblinear
ions.
  warnings.warn("Liblinear failed to converge, increase "
/databricks/python/lib/python3.8/site-packages/sklearn/svm/_base.py:985: ConvergenceWarning: Liblinear
ions.
  warnings.warn("Liblinear failed to converge, increase "
/databricks/python/lib/python3.8/site-packages/sklearn/svm/_base.py:985: ConvergenceWarning: Liblinear
ions.
  warnings.warn("Liblinear failed to converge, increase "
Best parameters   = {'estimator__C': 1, 'estimator__max_iter': 5000}
CV Training Score = 0.987
CV Test Score     = 0.657
```

Figure 8: Screenshot of convergence warnings on LinearSVC

We have also explored GridSearch using the Extra Trees, Ada Boost, and RandomForest and have also found that the default parameters were having the best results in most cases.

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

The LinearSVC produced a total on 558 misclassifications on the test data of size 1060. We randomly pick 200 misclassifications and try to gauge which points might have led to misclassification, for example, presence of some keywords, links, the GitHub project name, etc. An excel containing the reasons of 200 misclassifications is being shared along with this report.

Some of the key findings labelling the misclassifications are listed below –

(a) Sections often get labelled as 1 (Introduction) whenever explanation is given for some feature, process, etc.

(b) Presence of links can potentially bias towards section getting classified as 6 (References)
(c) Section 3 (How) get labelled as Section 6 (References) as their differences can sometimes be narrow

4 DISCUSSIONS

4.1 Bhavyai's discussion on the usage of models to write better README's

A real-life application of this model would be to generate badges based on the ML model output for each section. The generated badges can then be appended to every section heading. These badges help in increasing the readability of the readme file because they provide information at a glance about what each section talks about in the README file. They can also help visitors or other software developers in getting to familiarize with the repository quickly and easily.

Another application of this model would be to find what sections are missing from the README file. This could help the author of the repository to include all relevant information and write better READMEs.

4.2 Kayode's discussion

One way I can imagine is using models identify some ambiguous keywords in the README files sections. Once those ambiguities were identified and remove, it will be easy for the developers to pass across their messages in clear and concise manner.

Also, it would be interesting to develop a recommender model that will guide the developers on how to arrange the section so that they logically follow each other sequentially. This will enable the reader to quickly move to any relevant section of interest.

4.3 Michael Lee's discussion on the implications of the developed models

One useful scenario that may be useful for real-life application of this model is that it can be used to gauge the effectiveness of the Readme files. For example, in the world of internet marketing, we gauge the effectiveness of copywriting by its ability to sale their products or what they call the conversion rate. Similarly, we can adopt this method by linking our model with statistics such as the number of stars in the repository or visitor stats to see what kind of information, if included in their readmes, will increase people to visit, branch or star the repositories, therefore giving information to developers as to what information they should include in their readme files to maximize the effectiveness and increase popularity of their repository.

5 CONCLUSIONS

A README file is often the first document that a user sees when they encounter a new software repository. README files are essential in shaping the first impression of a repository and in documenting a software project. Despite their

important role, we lack a systematic understanding of the content. of README files as well as tools that can automate the discovery of relevant information contained in them.

In this project, we have reported on a qualitative study which involved the manual annotation of 1,000 sections from 86 README files for repositories hosted on GitHub, in addition to the 4,226 sections manually annotated by the research paper. We identified eight different kinds of content and found that information regarding the 'What' and 'How' of a repository is common while information on the status of a project is rare. Our best performing classifier achieved an F1 score of 0.721.

Our findings provide a point of reference for repository owners against which they can model and evaluate their README files, ultimately leading to an improvement in the quality of software documentation. Our classifier will help automate these tasks and make it easier for users and owners of repositories to discover relevant information.

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," SpringerLink, 12-Oct-2018. [Online]. Available: https://link.springer.com/article/10.1007/s10664-018-9660-3. [Accessed: 11-Dec-2021].
2. Gprana. "Gprana/READMEClassifier." GitHub, https://github.com/gprana/READMEClassifier.

APPENDIX

(a) "Appendix A Misclassifications_Explanations.xlsx"