|  |
| --- |
| ENSF612 Project |
| Categorizing the Content of GitHub README Files |
| Project Final Report |
|  |
| **Team Members :**  **Awe, Kayode**  **Gupta, Bhavyai**  **Lee, Michael** |
|  |

# Summary of Contributions

The contributions mentioned in this section will be split into the following subcategories:

* Data Collection
* Coding
* Report Writeup

## Data Collection

For data collection, **Bhavyai** has developed a script called "readme\_downloader.py" for collecting the readme.md files for our project. This script downloads random but unique README.md files from GitHub using the GitHub API.

Once the README.MD files are curated, we have selected 88 files with a total of 1000 sections for manual annotations. We would then each manually annotate the 1000 sections, and Kayode is the facilitator to make sure we reach an agreement of what the final annotations should be as well as calculating the statistics such as the Cohen Kappa analysis to show how our overall process.

**Michael** is responsible for extracting the sections from the readme files and to feed the new annotate data back into the database for modelling and analysis.

## Coding

**Bhavyai** is responsible for creating the script to scrape Readme files from Github for our manual annotations as described in the previous section.

**Michael** is responsible for modifying the original code from the author to work on databricks and make sure that the new data is successfully incorporated into the input for the models. Furthermore, **Michael** is responsible for the analysis of the original models as well as the new models as discussed in presentation 2 using the original data provided by the author to see which model performs the best. **Bhavyai**, **Kayode**, and **Michael** are jointly responsible for consolidating converting the code from the research paper to run on Databricks notebook for this project.

|  |  |
| --- | --- |
| **Author’s original models** | **Additional Models used** |
| RandomForestClassifier  LinearSVC  GaussianNB  LogisticRegression  KNeighborsClassifier | BaggingClassifier  ExtraTreesClassifier  DecisionTreeClassifier  AdaBoostClassifier  HistGradientBoostingClassifier |

## Report Writeup

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

|  |  |
| --- | --- |
| Section | Resource |
| Page 1 contents | Mike |
| Abstract/Intro/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 | Mike |
| How do the models perform on the original data vs the new + original data? | Mike |
| How does the performance of the models change based on the choice of hyper parameters? | Mike |
| How are the misclassifications of the best performing model distributed? | Kayode/Bhavyai |
| Discussion | All |

# Link to Databricks Notebook

<https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/279226329024279/3713457635192600/4593657877666368/latest.html>

# ABSTRACT

## 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.

## Objective

In this project, we replicate the research paper on “Categorizing the Content of GitHub README Files”. We also extend the work done in the paper by running more ML classifier models and tuning their hyper-parameters to explore the better performance.

# Preparing the Model on Databricks with Old+New Data

## Databricks Conversion

The author of GitHub README Content Classifier has provided the source code for their classifier program on Github (<https://github.com/gprana/READMEClassifier>). As part of the requirements of the ENSF612 project, one of the first task is to convert the author’s program to be able to operate on Databricks.

There are two hurdles that we need to overcome in order for us to successfully run their code on Databricks and they are:

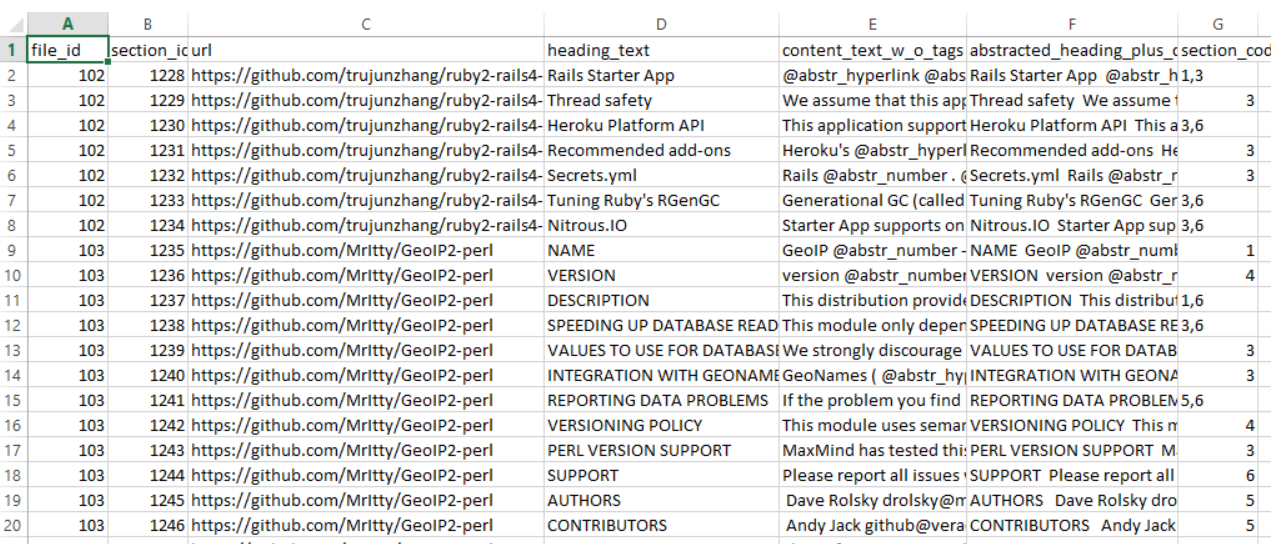
1. The current program utilizes a sql database to store and process information.
2. Databricks runs on notebooks that makes referencing python module files from their FileStore difficult.

After investigating as to how the program works, we have determined to modify their experiment\_classifier\_validation.py script to meet our needs. We copy all of the code from this file into a new file called project\_execution.py to make changes.

To address the first point, we dump their sql query data needed within the experiment\_classifier\_validation.py to do validation into a csv file called raw\_data.csv using this line of code:



This code dumps the sql query data into a csv file, which is found below:

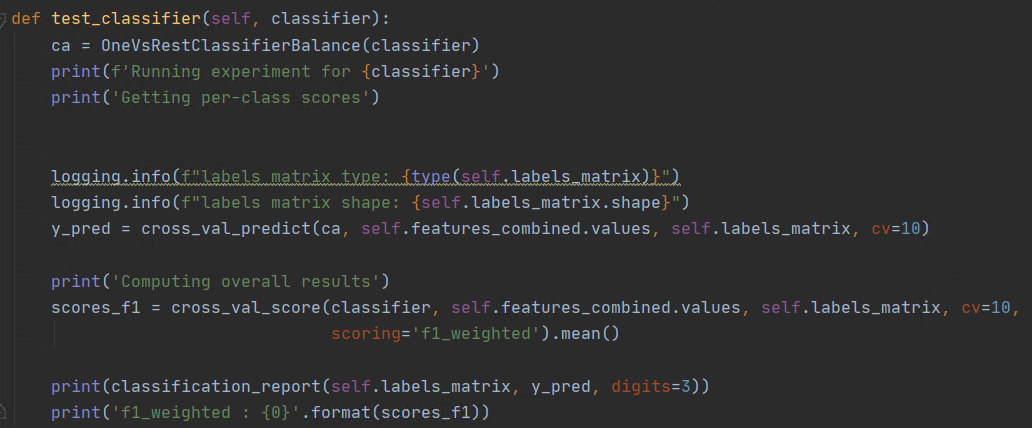


This csv file is then uploaded to the Filestore in databricks.

To address the second point, we need to look at what files did experiment\_classifier\_validation.py so that we can consolidate the code into one file. Upon looking at the file, we observed that there are two files from which experiment\_classifier\_validation.py gets reference from and they are:



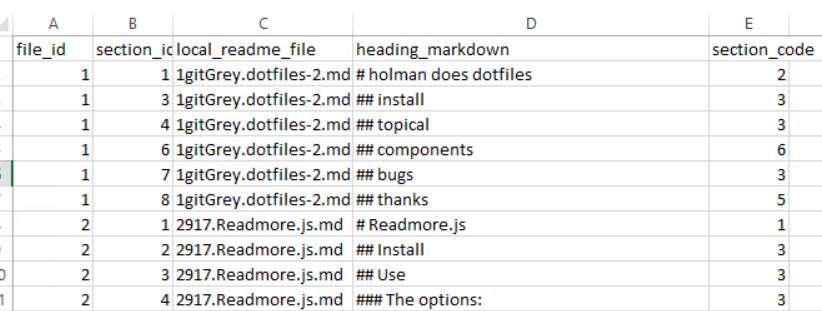
We then look at the code from those files and copy and re-organize the code in project\_execution.py so that everything can be running from a single class, called run\_algorithm. Lastly, to allow the flexibility of running multiple models easily within Databricks (originally the source code is hardcoded to run LinearSVC model only), we have created the test\_classifer function:



The resulting code is what you see in the Databrick Notebooks linked in the previous sessions.

## Adding New Data within Existing Dataset

Once we have our manual annotations finished and organized into a spreadsheet in the following format:

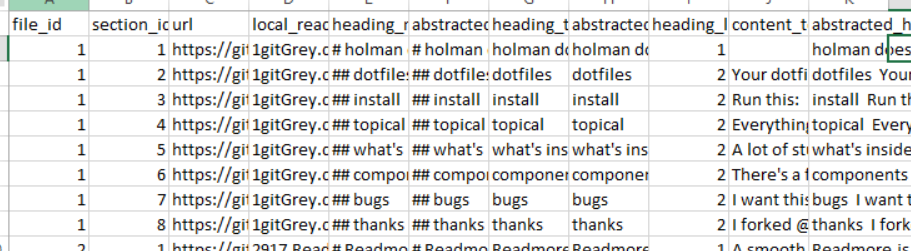


The next thing we need to do is to append these results into the existing dataset. One hurdle that we encounter is that the data required for the model to function requires more information than the spreadsheet that we populate our manual annotations on.

To produce a dataset that is compatible with the model, we follow “Use Case 3: Training Model on Existing Data and Classifying New Files” from the Research author’s Github ReadMe.md.

Similar to what we did to create raw\_data.csv previously, we added the following line to load\_and\_classify\_target.py to get the csv file for the data we need:

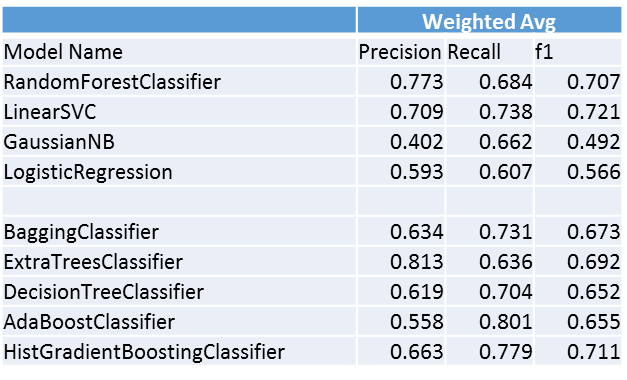




Now that we have completeclassifytarget spreadsheet along with our manual annotations, we need to combine them together. For this, we created a vba script titled datafeeder.xlsm to append the annotations to the completeclassifytarget spreadsheet. We save this into a csv file titled old\_and\_new\_data.csv, which is used in our Databricks notebooks for model analysis.

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:



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.

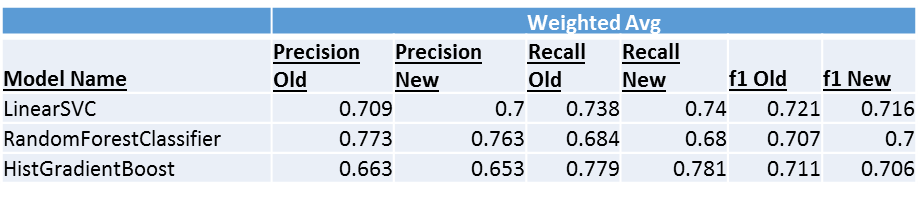
Recall that Precision = (True Positive/Total Predicted Positive) and Recall = (True Positive)/(Total Actual Positive), Extra Trees Classifier is the best model to use when the cost of any false positive is high and AdaBoost should be used if we want to filter out the most false negatives.

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:

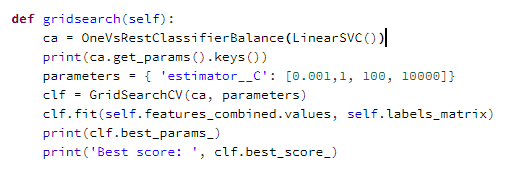


As can be seen, the Precision, Recall and f1 scores drop slightly when additional data are added, but they do not affect the order of the performance ranking of these models.

## How does the performance of the models change based on the choice of hyper parameters?

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 within the run\_algorithm class titled gridsearch:



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



# Discussion

## Michael Lee’s Discussion 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.