1. **Instruction**
   1. **Motivation**

The applications of Computer Science in various scientific fields in the recent past has resulted in the proliferation of scientific software. Researchers and Scientists look to leverage the existing software available to carry out their experiments, perform visualizations and for other use cases. However, they have to spend a lot of time going through the documentation of these software and ultimately evaluating whether the piece of software meets their needs. To address this problem a tool called Software Metadata Extraction Framework (SOMEF) was created which automatically extracts certain metadata pertaining to scientific repositories and uses four separate classifiers to extract description, citation, installation and invocation of these repositories. The developers of this tool created a corpus from readme files of 89 GitHub repositories which was used for training purposes. However, this tool does not have the provision to classify the functionality of a given scientific repository. We aim to build on the existing line of work in SOMEF and create our own framework which we call Functionality and Metadata Extraction Framework (FAMEF).

* 1. **Objectives**

1. Based on the evaluation metrics of SOMEF, we aim to improve the performance of four existing classifiers for description, invocation, citation and installation by extending the existing corpus. The classifiers should generalize well on unseen data. The existing baseline scores of these four classifiers are summarized in Table 1.1.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Accuracy | Recall | F1 |
| Description | 0.83 | 0.79 | 0.82 |
| Invocation | 0.88 | 0.94 | 0.9 |
| Installation | 0.9 | 0.9 | 0.91 |
| Citation | 0.92 | 0.98 | 0.93 |

Table 1.1:  Existing baseline scores of classifiers used in SOMEF

1. Develop a multi class classifier to extract the functionality of repositories with an accuracy above the baseline (null accuracy: 33%)
   1. **User Stories**
2. As a researcher I would like to make use of existing scientific software in order to facilitate and expedite my work.
3. As a researcher I would like to get details about the functionality of a scientific software which would serve as a great starting point for me as it would give an indication of whether the piece of software is related to what I am looking for.
4. As a researcher I want to get description of a piece of scientific software quickly in order for me to evaluate whether it meets my needs.
5. As a researcher I want to compare similar software before using them in the industry in order to be more efficient.
   1. **Requirements**
      1. **Function**
6. The framework must automatically extract metadata of scientific repositories including **description**, **installation**, **invocation**, **citation**, and **functionality**, from the existing instructions of a piece of scientific software (README file) and improve the previous classification score.
7. The framework must additionally assign a piece of excerpt to a certain category only if the confidence of the classifier is above the specified threshold.
8. The framework must generate satisfactory results with new unseen repositories.
   * 1. **Non-Function**
9. The system should build on the existing work from SOMEF.
10. The framework must generate output within five minutes.
    * 1. **Team**
11. Skilled Data Scientists having hands on knowledge in training machine learning models and interpreting the results in order to convey them to an audience.
12. A project manager to supervise and monitor status updates in order to achieve the project deliverables.
    1. **Evaluation Metrics**
13. Precision: Extract the correct piece of information and improve the existing score
14. F1 score: Achieving a balance between precision and recall, as it is important that we extract maximum information without losing much precision.
    1. **Risks**

A picture containing graphical user interface

Description automatically generated

Table 1.2:  Risk Assessment Matrix

* 1. **Risks Mitigation Methods**

1. Decide beforehand on what labels to assign to a repository.
2. Avoid meeting in person to mitigate the risk of COVID-19.
3. Follow project management techniques like Agile in order to meet the deliverables on time and to ensure that the project team members and client are on the same page.
4. Have a project manager assess the strength and weakness of each and every individual in the team and assign tasks accordingly to avoid the possibility of any conflicts.
5. Meet with the instructors regularly to ensure that the team is headed towards the right path.
6. Thoroughly research the previous work done including the various research papers to make sure there are no hassles while building upon it.
   1. **Project Summary**

Framework Metadata Extraction Framework (FAMEF) extends the existing framework SOMEF and extracts scientific software metadata and its functionality from documentation. This streamlines the work of researchers who are looking to reuse scientific software to carry out the work. We have trained classifiers(supervised) which detects different categories of metadata. We will be evaluating the performance of classifiers against unseen repositories different from the ones used in training.

* 1. **Elevator Pitch**

Researchers used to have a hard time discovering scientific software online that fits their use case. But, thanks to FAMEF researchers can now take a backseat and bank on FAMEF to automatically generate relevant information pertaining to a scientific software and use it for their research.

1. **Project Planning**
   1. **Workflow**

This project can be distinctively understood with the help of two workflows. This arises from the fact that we had to experiment with different kinds of classifiers for description, installation, citation, invocation and functionality. After choosing the final classification model for each of the above 4 categories, we had to implement this best model into our pipeline so that it works smoothly with different input README files.

The two workflows are as follows:

1. **The modeling or training workflow**: The pre-existing 89 GitHub repositories had their description, installation, citation and invocation manually annotated. Since we increased the number of repositories by 25, we had to manually extract excerpts for these 4 categories for the extra 25 repos.

Additionally, since we were developing the functionality classifier from scratch, we had to manually label each of the 114 repositories as either a data visualization, data preparation or data analysis tool/software. These labelled datasets were stored in CSV files, namely, extended\_description.csv, extended\_invocation.csv, extended\_installation.csv, extended\_citation.csv, function.csv.

Now, we experimented with different classification models and other methods on these train datasets. To be specific, invocation classifier model was trained using extended\_invocation.csv, citation model using extended\_citation.csv, installation model using extended\_installation.csv. Extended\_description.csv was used for the description model, LDA based topic modeling and the multi-class functionality classifier. The functionality classifier additionally makes use of the function.csv.

Diagram

Description automatically generated

Table 2.1: The Training Workflow

1. **The application level or test workflow**: This explains how, given a test GitHub README, the application processes and outputs the results. The excerpts of the README are read by the pipeline. The four best binary classification models, belonging to description, installation, citation and invocation classify the corresponding excerpts into the four categories, along with a confidence level. These excerpts are returned as result into the result.json file.

The functionality classifier needs the excerpts which have been labelled as ‘description’ excerpts. For this purpose, it takes as input the output of the description classifier. It then classifies the test repository into one of the three functionalities discussed before and adds the result into the json output file.

Diagram

Description automatically generated

Table 2.2: The Test Workflow

1. **Project Process**
   1. **Data Preparation**

Since we built FAMEF based on SOMEF, we have remained 89 existing Github repositories. To extend the corpus, we decided to add another 25 new repositories.

In the SOMEF, developers extracted the excerpts of description, installation, invocation, and citation manually and saved them in 4 CSV files to train and test classifier models. We planned to use the same approach.

* + 1. **New Github Repositories**

Since FAMEF would be focused on scientific software, we needed to find software repositories which work for scientific fields, such as Math, Physics, Astronomy, and etc. The Readme files of these repositories should include description, installation, invocation, and citation as much as possible. Finally, everyone in our team found 5 new repositories.

* + 1. **Description Extraction and the CSV File**

The developers of SOMEF saved the data of description in a file called “description.csv”, which is shown below (Figure 3.1.1).

A picture containing text

Description automatically generated

Fig 3.1.1: The Original CSV File - description.csv

There are 3 columns in this file: URL, contributor, and excerpt. Sometimes, there are multiple sentences of description. The developers usually separated them into various data records.

Similarly, for every new repository found, we collected the GitHub link, the name who found and the sentence of description together as one data record and entered it into this CSV file.

We called the updated CSV file “extended\_description.csv”. The updated part is shown below (Figure 3.1.2).

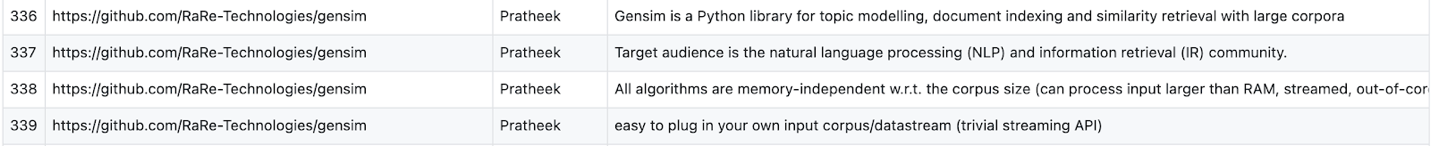


Fig 3.1.2: The Updated CSV File - extended\_description.csv

* + 1. **Installation Extraction and the CSV File**

The developers of SOMEF used the same way to extract installation and generated the CSV file. Therefore, we used the same method, updated the file related to installation and called it “extended\_installation.csv”, which is shown below (Figure 3.1.3).

Table

Description automatically generated

Fig 3.1.3: The Updated CSV File - extended\_installation.csv

* + 1. **Invocation Extraction and the CSV file**

Similarly, we updated the file related to invocation and called it “extended\_invocation.csv”, which is shown below (Figure 3.1.4).

Table

Description automatically generated

Fig 3.1.4: The Updated CSV File - extended\_invocation.csv

* + 1. **Citation Extraction and the CSV File**

We updated the citation CSV file and called it “extended\_citation.csv”, which is shown below (Figure 3.1.5).

Table

Description automatically generated

Fig 3.1.5: The Updated CSV File - extended\_citation.csv

* + 1. **Functionality Extraction and the CSV File**

Since SOMEF does not have a functionality classifier and related data, we created a new CSV file called “function.csv”. We extracted the excerpts of functionality and entered into the file with URL and the contributor’s name. In addition, we assigned a label to each instance. We show part of the file below (Figure 3.1.6).

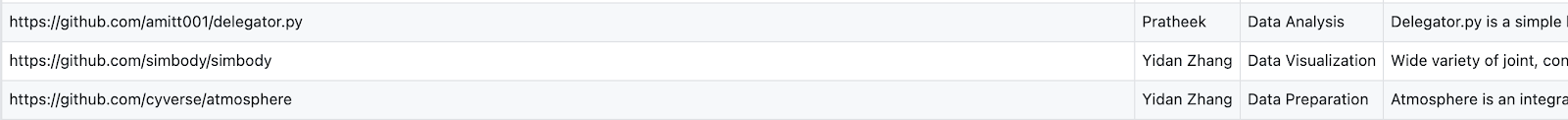


Fig 3.1.6: The Function CSV File - function.csv

There are three labels to assign: data analysis, data visualization and data preparation. We set up a standard to assign labels. We would assign the repository to data analysis if there were key words, like “machine learning” and “neural network" in its Readme file. If we found key words, like “visualization” and “UI" we would regard this repository as data visualization. As for other repositories without key words, we would give a “data preparation” label.

* + 1. **Corpus Balancing**

After extending the corpus of each classifier, we had to balance the datasets to avoid overfitting.

1. Description Dataset Balancing

To balance the description dataset, we selected the same number of instances from installation, invocation, and citation as the false class. In the end, we have 414 (around 0.57) instances as the true class and 309 (around 0.43) instances as the false class. The distribution graph is shown below (Figure3.1.7).

Chart, bar chart

Description automatically generated

Fig 3.1.7: The Distribution Graph of Description

1. Installation Dataset Balancing

To balance the installation dataset, we selected the same number of instances from description, invocation, and citation as the false class. In the end, we have 1096 (around 0.57) instances as the true class and 822 (around 0.43) instances as the false class. The distribution graph is shown above (Figure 3.1.8).

Chart, bar chart

Description automatically generated

Fig 3.1.8: The Distribution Graph of Installation

1. Invocation Dataset Balancing

To balance the invocation dataset, we selected the same number of instances from installation, description, and citation as the false class. In the end, we have 1523 (around 0.57) instances as the true class and 1140 (around 0.43) instances as the false class. The distribution graph is shown below (Figure 3.1.9).

Chart, bar chart

Description automatically generated

Fig 3.1.9: The Distribution Graph of Invocation

1. Citation Dataset Balancing

To balance the citation dataset, we selected the same number of instances from installation, invocation, and description as the false class. In the end, we have 409 (around 0.57) instances as the true class and 306 (around 0.43) instances as the false class. The distribution graph is shown above (Figure 3.1.10).

Chart, bar chart

Description automatically generated

Fig 3.1.10: The Distribution Graph of Citation

* 1. **Description Classifier**
* The description classifier aims to extract the excerpts which give us more information about what a particular tool or software can do.
* It gives the user some idea regarding when and why to use the particular piece of software.
* We followed a number of approaches to find the best model for the description classifier.
* For each of the approaches, we first split our extended corpus into a train and a test set.
* Each model was re-trained on 75% of the extended corpus, while the metrics that we report are on the remaining 25% of the data.
  + 1. **Approach One**

We **ran those top 4 models**, which had given the highest scores in terms of accuracy, recall and precision for the SOMEF tool. The difference here was that these models were being run on the extended corpus.

These models were:

1. CountVectorizer, a text feature extraction model, which converts a collection of text documents to a matrix of token counts, followed by the Logistic Regression classification algorithm.
2. TfidfVectorizer, a text feature extraction model, which converts a collection of raw documents to a matrix of TF-IDF features, followed by the Logistic Regression classification algorithm.
3. TfidfVectorizer, followed by the Multinomial Naïve Bayes classification algorithm.
4. TfidfVectorizer, followed by the Perceptron based classification algorithm.

The results of the above algorithms were quite comparable to the results of these algorithms being run on the non-extended corpus. In some cases, these performed a little better than the earlier models.

* + 1. **Approach Two**

For this approach, we choose the below 4 models again, since they were the best performing SOMEF models for the description classifier. This time they were run on the extended corpus, and we had additionally removed the stop words (a, an, is, the, etc.).

1. CountVectorizer, followed by Logistic Regression
2. TfidfVectorizer, followed by Logistic Regression
3. TfidfVectorizer, followed by Multinomial Naïve Bayes
4. TfidfVectorizer, followed by Perceptron

The results of the above algorithms were quite comparable to the results of these algorithms being run on the non-extended corpus with the stop words. In some cases, these performed a little worse than the earlier models.

From the above two approaches, we did not get a result which was genuinely better than that of SOMEF’s models. So, we decided to continue to experiment with other methods, and see if they could make the difference

* + 1. **Approach Three**

Our next approach was to try out tree-based classification algorithms on our extended corpus. A major reason for this was that we have often seen such algorithms, especially XGBoost outperform others in Data Science competitions. It implements machine learning algorithms under the Gradient Boosting framework. CountVectorizer, followed by Logistic Regression

1. TfidfVectorizer, followed by the Random Forest Classifier
2. TfidfVectorizer, followed by the XGBoost Classifier
3. CountVectorizer, followed by Perceptron

The results have been mentioned in Table 3.2. However, they were not encouraging enough. Without proper hyper parameter tuning, the tree-based methods could not do better than our baseline model, even on an average. Hence, we decided to go for proper tuning of hyperparameters to get better performing models.

* + 1. **Approach Four**

In this approach, we carried out in-depth tuning of hyperparameters for a number of approaches. The results have been shown in Table I. These results were highly encouraging, even though the tree-based methods were not the highest performing ones. The metrics for most of the below models were above our baseline.

1. CountVectorizer, followed by Logistic Regression (L1 Penalty). For this, we carried out tuning of ‘C’, which is the inverse of regularization strength, with the help of 5-fold StratifiedKFold cross validation.
2. TfidfVectorizer, followed by Logistic Regression (L2 Penalty). For this, we carried out tuning of ‘C’, which is the inverse of regularization strength, with the help of 5-fold StratifiedKFold cross validation.
3. TfidfVectorizer, followed by Multinomial Naïve Bayes. For this, we carried out tuning of ‘alpha’, which is the Additive smoothing parameter, with the help of 5-fold StratifiedKFold cross validation.
4. TfidfVectorizer, followed by Logistic Regression (L1 Penalty). For this, we carried out tuning of ‘C’, which is the inverse of regularization strength, with the help of 5-fold StratifiedKFold cross validation.
5. CountVectorizer, followed by Logistic Regression (L2 Penalty). For this, we carried out tuning of ‘C’, which is the inverse of regularization strength, with the help of 5-fold StratifiedKFold cross validation.
6. TfidfVectorizer, followed by Random Forest. For this, we carried out tuning of ‘max\_depth’ (maximum depth of the tree), ‘max\_features’ (the number of features to consider while looking for the best split), ‘n\_estimators’ (the number of trees in the forest). This hyperparameter tuning was carried out with the help of 5-fold StratifiedKFold cross validation.
7. CountfVectorizer, followed by XGBoost classifier. For this, we carried out tuning of ‘max\_depth’(the maximum depth of a tree), ‘min\_child\_weight’ (the minimum sum of weights of all observations required in a child), ‘learning\_rate’ (makes the model more robust by shrinking the weights on each step), ‘n\_estimators’ (the number of trees in the forest), ‘gamma’ (specifies the minimum loss reduction to make a split), ‘subsample’ (denotes the fraction of observations to be randomly samples for each tree), ‘colsample\_bytree’ ( the number of features to consider while looking for the best split), ‘reg\_alpha’ (L1 regularization term on weights). This hyperparameter tuning was carried out with the help of 5-fold StratifiedKFold cross validation.

We found TfidfVectorizer + Logistic Regression (L2 penalty with hyperparameter tuning) to be our best model. It gave significantly better results in terms of accuracy, precision, recall and F1 than our baseline.

In the figure below, SOMEF refers to the best performing SOMEF description classifier, while FAMEF refers to the best model we mentioned above.

Chart, bar chart

Description automatically generated

Fig 3.2.1

For our best performing model, we also display two kinds of plots: The **ROC - AUC plot** andthe **Accuracy - Recall curve**.

1. **ROC - AUC plot:** It is a plot of the false positive rate (X-axis) versus the true positive rate (Y-axis) for a number of different candidate threshold values between 0 and 1. Alternatively, it can be seen as a plot between the false alarm rate versus the hit rate. Generally, the greater the area under the curve (AUC), the better the algorithm/model.
2. **Accuracy - Recall curve:** The accuracy and recall can be calculated for thresholds using the precision\_recall\_curve function that takes the true output values and the probabilities for the positive class as input and returns the accuracy, recall and threshold values. It is an efficient way to compare accuracy and recall on a graph. While plotting accuracy and recall for each threshold as a curve, it is important that recall is plotted on the X-axis and accuracy on the Y-axis.

Below is the ROC - AUC plot for our best FAMEF model for the description classifier:

Chart, line chart

Description automatically generated

Fig 3.2.2

Below is the precision - recall curve:

A picture containing diagram

Description automatically generated

Fig 3.2.3

* + 1. **Approach Five**

We have been very comprehensive in all the approaches that we followed. For this reason, we wanted to give a couple of other approaches a go as well, even after getting our best model. We did not want to miss out on any approach that could tentatively provide us with better results.

1. Deep Learning:

* We built a neural network with 2 hidden layers, with 16 neurons each.
* These 2 layers had ‘relu’ as its activation, while the final layer had ‘sigmoid’ as its activation.
* We used a validation set, apart from a train and test set to tune our hyperparameters.
* In the end, we settled for ‘rmsprop’ optimizer, ‘binary\_crossentropy’ as loss, 35 epochs with a batch size of 512.
* The results have been discussed in Table 3.2.

1. Feature selection with another classification algorithm:

* We added another step in our existing pipeline for the best model of TfidfVectorizer + Logistic Regression (L2 penalty with hyperparameter tuning).
* We incorporated a SelectFromModel functionality after the TfidfVectorizer.
* In this functionality, we used Random Forest classifier as the feature selection algorithm.
* The result has been described in Table 3.2.

|  |  |  |  |
| --- | --- | --- | --- |
| Model | Accuracy | Recall | F1 |
| TfidfVectorizer+LogisticRegression (L2 Penalty with hyperparameter tuning) | 0.87 | 0.92 | 0.9 |
| CountVectorizer+LogisticRegression (L1 Penalty with hyperparameter tuning) | 0.86 | 0.88 | 0.88 |
| TfidfVectorizer+MultinomialNB (alpha hyperparameter tuning) | 0.86 | 0.96 | 0.89 |
| |  | | --- | | TfidfVectorizer+SelectFromModel(RF)+LogisticRegression (L2 Penalty with hyperparameter tuning) | | 0.86 | 0.89 | 0.88 |
| CountVectorizer+ DL Classifier | 0.86 | 0.81 |  |
| TfidfVectorizer+LogisticRegression (L1 Penalty with hyperparameter tuning) | 0.84 | 0.83 | 0.86 |
| CountVectorizer+LogisticRegression (L2 Penalty with hyperparameter tuning) | 0.83 | 0.81 | 0.84 |
| |  | | --- | | TfidfVectorizer+RandomForestClassifier | | 0.82 | 0.78 | 0.83 |
| TfidfVectorizer+RandomForestClassifier (tuning of hyperparameters) | 0.82 | 0.89 | 0.85 |
| TfidfVectorizer+XGBClassifier | 0.8 | 0.78 | 0.83 |
| CountVectorizer+XGBClassifier | 0.79 | 0.77 | 0.81 |
| CountVectorizer+XGBClassifier ( with stepwise tuning) | 0.76 | 0.76 | 0.77 |
| SOMEF Baseline | 0.83 | 0.79 | 0.82 |

Table 3.2: The Results of Description Classifier

* 1. **Invocation Classifier**
* The invocation classifier aims to extract the excerpts which give us more information about usability and understandability of a software/tool.
* It gives the user some idea regarding how to use the particular piece of software.
* We followed a number of approaches to find the best model for the invocation classifier.
* For each of the approaches, we first split our extended corpus into a train and a test set.
* Each model was re-trained on 75% of the extended corpus, while the metrics that we report are on the remaining 25% of the data (test set).
  + 1. **Approach One**

We **ran those top 4 models**, which had given the highest scores in terms of accuracy, recall and precision for the SOMEF tool. The difference here was that these models were being run on the extended corpus.

These models were:

1. CountVectorizer, followed by the Naïve Bayes classification algorithm.
2. TfidfVectorizer, followed by the Naïve Bayes classification algorithm.
3. TfidfVectorizer, followed by the Stochastic Gradient Descent classification algorithm.
4. CountVectorizer, followed by the Logistic Regression classification algorithm.

The results of the above algorithms were quite comparable to the results of these algorithms being run on the non-extended corpus. In some cases, these performed a little better than the earlier models.

* + 1. **Approach Two**

For this approach, we choose the below 4 models again, since they were the best performing SOMEF models for the invocation classifier. This time they were run on the extended corpus, and we had additionally removed the stop words (a, an, is, the, etc.).

1. CountVectorizer, followed by Naïve Bayes classification algorithm.
2. TfidfVectorizer, followed by Naïve Bayes classification algorithm.
3. TfidfVectorizer, followed by Stochastic Gradient Descent classification algorithm
4. CountVectorizer, followed by Logistic Regression.

The results of the above algorithms were quite comparable to the results of these algorithms being run on the non-extended corpus with the stop words. In some cases, these performed a little worse than the earlier models.

From the above two approaches, we did not get a result which was genuinely better than that of SOMEF’s models. So, we decided to continue to experiment with other methods, and see if they could make the difference.

* + 1. **Approach Three**

Our next approach was to try out tree-based classification algorithms on our extended corpus. A major reason for this was that we have often seen such algorithms, especially XGBoost outperform others in Data Science competitions. It implements machine learning algorithms under the Gradient Boosting framework.

1. TfidfVectorizer, followed by the Random Forest Classifier
2. TfidfVectorizer, followed by the XGBoost Classifier

The results have been mentioned in Table I. However, they were not encouraging enough to beat the existing baseline. Hence, we decided to go for proper tuning of hyperparameters to get better performing models.

* + 1. **Approach Four**

In this approach, we carried out in-depth tuning of hyperparameters for a number of approaches. The results have been shown in Table I. These results were highly encouraging. The metrics for most of the below models were above or equal to our baseline.

1. CountVectorizer, followed by Logistic Regression (L2 Penalty). For this, we carried out tuning of ‘C’, which is the inverse of regularization strength, with the help of 5-fold StratifiedKFold cross validation.
2. TfidfVectorizer, followed by Multinomial Naïve Bayes. For this, we carried out tuning of ‘alpha’, which is the Additive smoothing parameter, with the help of 5-fold StratifiedKFold cross validation.
3. TfidfVectorizer, followed by Stochastic Gradient Descent Classifier. For this, we carried out tuning of ‘loss’ (the loss function), ‘alpha’ (the constant that multiplies the regularization term), ‘penalty’ (the regularization term). This was carried out with the help of 5-fold StratifiedKFold cross validation.

We found TfidfVectorizer + Multinomial Naïve Bayes (alpha hyperparameter tuning) to be our best model. It gave significantly better results in terms of accuracy and precision than our baseline.

In the figure below, SOMEF refers to the best performing SOMEF invocation classifier, while FAMEF refers to the best model we mentioned above.

Chart, bar chart

Description automatically generated

Fig 3.3.1

For our best performing model, we also display two kinds of plots: The **ROC - AUC plot** and the **Accuracy - Recall curve**.

Below is the ROC - AUC plot for our best FAMEF model for the invocation classifier:

Chart

Description automatically generated

Fig 3.3.2

Below is the accuracy - recall curve:

A picture containing table

Description automatically generated

Fig 3.3.3

* + 1. **Approach Five**

We have been very comprehensive in all the approaches that we followed. We did not want to miss out on any approach that could tentatively provide us with better results.

1. Feature selection with another classification algorithm

* We added another step in our existing pipeline for the best model of TfidfVectorizer + Multinomial Naïve Bayes (alpha hyperparameter tuning).
* We incorporated a SelectFromModel functionality after the TfidfVectorizer.
* In this functionality, we used the Logistic Regression classifier as the feature selection algorithm.
* The result has been described in Table 3.3.

|  |  |  |  |
| --- | --- | --- | --- |
| Model | Accuracy | Recall | F1 |
| TfidfVectorizer+MultinomialNB (alpha hyperparameter tuning) | 0.89 | 0.91 | 0.9 |
| CountVectorizer+ MultinomialNB (alpha hyperparameter tuning) | 0.89 | 0.93 | 0.91 |
| TfidfVectorizer+StochasticGradientDescent (with hyperparameter tuning) | 0.88 | 0.9 | 0.9 |
| |  | | --- | | TfidfVectorizer+SelectFromModel(logit)+MultinomialNB (alpha hyperparameter tuning) | | 0.87 | 0.93 | 0.89 |
| CountVectorizer+ LogisticRegression (L2 Penalty with hyperparamter tuning) | 0.86 | 0.93 | 0.89 |
| TfidfVectorizer+RandomForestClassifier | 0.72 | 0.96 | 0.8 |
| SOMEF Baseline | 0.88 | 0.94 | 0.9 |

Table 3.3: The Results of Invocation Classifier

* 1. **Citation Classifier**

Citation classifier is to predict whether a sentence is about citation information or not. So each sentence can be categorized into two classes: 1 means that it is a citation sentence, 0 means it is not.

* + 1. **Retrain and improve top 4 models from SOMEF**

We found the top 4 models in SOMEF and retrained them on our extended corpus.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| sklearn Primitive - Citation | Hyperparameters | Accuracy | Recall | F-measure |
| sklearnpipeline(CountVectorizer, NaiveBayes) | - | 0.92 | 0.98 | 0.93 |
| sklearnpipeline(CountVectorizer, LogisticRegression) | - | 0.89 | 0.98 | 0.91 |
| sklearnpipeline(TFIDFVectorizer, StochasticGradientDescent) | loss = 'log' | 0.89 | 0.97 | 0.91 |
| sklearnpipeline(TFIDFVectorizer, LogisticRegression) | - | 0.88 | 0.98 | 0.91 |
| sklearnpipeline(TFIDFVectorizer, NaiveBayes) | - | 0.88 | 0.98 | 0.91 |
| sklearnpipeline(TFIDFVectorizer, Perceptron) | - | 0.87 | 0.86 | 0.89 |
| sklearnpipeline(TFIDFVectorizer, AdaBoostClassifier) | - | 0.84 | 0.78 | 0.85 |
| sklearnpipeline(TFIDFVectorizer, DecisionTreeClassifier) | - | 0.81 | 0.9 | 0.84 |
| sklearnpipeline(TFIDFVectorizer, XGBClassifier) | - | 0.8 | 0.76 | 0.81 |
| sklearnpipeline(TFIDFVectorizer, RandomForestClassifier) | - | 0.78 | 0.94 | 0.83 |
| sklearnpipeline(TFIDFVectorizer, BernoulliBayes) | - | 0.74 | 0.99 | 0.81 |

Table 3.4: citation\_classifiers.csv from SOMEF

1. Count Vectorizer and Naïve Bayes

We built the scikit-learn pipeline with Count Vectorizer and Naïve Bayes classifier and passed our training data to the pipeline and tested the model on test data. Then we did hyper-parameter tuning on alpha = 1.0, 0.5, 0.2, 0.1 and 5-fold cross validation to get the best result. Then we tried to remove stop words from our data and pass the data without stop words to the pipeline and repeat the steps mentioned above. For the best model, we plot the AOC-RUC curve and Precision-Recall curve:

1. Count Vectorizer and Logistic Regression

We built the scikit-learn pipeline with Count Vectorizer and Logistic Regression classifier with L2 penalty and pass our training data to the pipeline and test the model on test data. Then we did hyper-parameter tuning on c = 1.5, 1.0, 0.5, 0.2, 0.1 and 5-fold cross validation to get the best result. Then we tried to remove stop words from our data and pass the data without stop words to the pipeline and repeat the steps mentioned above. For the best model, we plot the AOC-RUC curve and Precision-Recall curve:

1. TFIDF Vectorizer and Stochastic Gradient Descent

We built the scikit-learn pipeline with TFIDF Vectorizer and Stochastic Gradient Descent classifier using log as loss function and L2 penalty, and pass our training data to the pipeline, and test the model on test data. Then we did hyper-parameter tuning on alpha=0.01, 0.001, 0.0001, 0.00001 and 5-fold cross validation to get the best result. Then we tried remove stop words from our data and pass the data without stop words to the pipeline and repeat the steps mentioned above. For the best model, we plot the AOC-RUC curve and Precision-Recall curve:

1. TFIDF Vectorizer and Logistic Regression

We built the scikit-learn pipeline with TFIDF Vectorizer and Logistic Regression classifier with L2 penalty and pass our training data to the pipeline and test the model on test data. Then we did hyper-parameter tuning on c = 1.5, 1.0, 0.5, 0.2, 0.1 and 5-fold cross validation to get the best result. Then we tried to remove stop words from our data and pass the data without stop words to the pipeline and repeat the steps mentioned above. For the best model, we plot the AOC-RUC curve and Precision-Recall curve:

* + 1. **Experiment with other classification models**

Besides the existing models, our second approach is to try the MLP Classifier and SVM Classifier on citation with both Count Vectorizer and TFIDF Vectorizer.

1. Count Vectorizer and MLP Classifier

We built the scikit-learn pipeline with Count Vectorizer and MLP Classifier and passed our training data to the pipeline, and tested the model on test data. Then we did hyper-parameter tuning on alpha=0.01, 0.001, 0.0001, 0.00001, and activation function = ‘relu’, ’tanh’, ’logistic’ ,and hidden layer = 100(default), 50, 20, 10. Then we perform 5-fold cross validation to get the best result. Then we tried to remove stop words from our data and pass the data without stop words to the pipeline and repeat the steps mentioned above. For the best model, we plot the AOC-RUC curve and Precision-Recall curve:

1. TFIDF Vectorizer and MLP Classifier

We built the scikit-learn pipeline with TFIDF Vectorizer and MLP classifier and passed our training data to the pipeline and tested the model on test data. Then we did hyper-parameter tuning on alpha=0.01, 0.001, 0.0001, 0.00001, and activation function = ‘relu’, ’tanh’, ’logistic’, and hidden layer = 100(default), 50, 20, 10. Then we perform 5-fold cross validation to get the best result. Then we tried to remove stop words from our data and pass the data without stop words to the pipeline and repeat the steps mentioned above. For the best model, we plot the AOC-RUC curve and Precision-Recall curve:

1. Count Vectorizer and SVM Classifier

We built the scikit-learn pipeline with Count Vectorizer and SVM Classifier with L2 penalty and pass our training data to the pipeline and test the model on test data. Then we did hyper-parameter tuning on kernel = ‘linear’, ‘poly’,’rbf’ and then performed 5-fold cross validation to get the best result. Then we tried to remove stop words from our data and pass the data without stop words to the pipeline and repeat the steps mentioned above. For the best model, we plot the AOC-RUC curve and Precision-Recall curve:

1. TFIDF Vectorizer and SVM Classifier

We built the scikit-learn pipeline with TFIDF Vectorizer and Logistic Regression classifier with L2 penalty and pass our training data to the pipeline and test the model on test data. Then we did hyper-parameter tuning on kernel = ‘linear’, ‘poly’,’rbf’ and then performed 5-fold cross validation to get the best result. Then we tried to remove stop words from our data and pass the data without stop words to the pipeline and repeat the steps mentioned above. For the best model, we plot the AOC-RUC curve and Precision-Recall curve:

* + 1. **The best model for citation classifier**

After completing the above approaches, we selected our best model based on the precision, recall, and F1 score of the models. The classifier we choose is TFIDF Vectorizer and Stochastic Gradient Descent using log as loss function and L2 penalty.

* 1. **Installation Classifier**

Citation classifier is to predict whether a sentence is about Installation instruction or not. So each sentence can be categorized into two classes: 1 means that it is an Installation sentence, 0 means it is not.

* + 1. **Retrain and improve top 4 models from SOMEF**

We found the top 4 models in SOMEF and retrained them on our extended corpus.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sklearn Primitive - Installation** | **Input Data Used** | **Accuracy** | **Recall** | **F-measure** |
| sklearnpipeline(TFIDFVectorizer, StochasticGradientDescent) | loss = 'log' | 0.9 | 0.9 | 0.91 |
| sklearnpipeline(TFIDFVectorizer, LogisticRegression) | - | 0.89 | 0.91 | 0.91 |
| sklearnpipeline(CountVectorizer, LogisticRegression) | - | 0.88 | 0.93 | 0.9 |
| sklearnpipeline(CountVectorizer, NaiveBayes) | - | 0.88 | 0.96 | 0.9 |
| sklearnpipeline(TFIDFVectorizer, Perceptron) | - | 0.86 | 0.9 | 0.88 |
| sklearnpipeline(TFIDFVectorizer, NaiveBayes) | - | 0.84 | 0.98 | 0.88 |
| sklearnpipeline(TFIDFVectorizer, RandomForestClassifier) | - | 0.84 | 0.81 | 0.85 |
| sklearnpipeline(TFIDFVectorizer, DecisionTreeClassifier) | - | 0.81 | 0.78 | 0.82 |
| sklearnpipeline(TFIDFVectorizer, AdaBoostClassifier) | - | 0.79 | 0.73 | 0.8 |
| sklearnpipeline(TFIDFVectorizer, XGBClassifier) | - | 0.78 | 0.71 | 0.79 |
| sklearnpipeline(TFIDFVectorizer, BernoulliBayes) | - | 0.75 | 0.98 | 0.82 |

Table 3.4: installation\_classifiers.csv from SOMEF

1. TFIDF Vectorizer and Stochastic Gradient Descent

We built the scikit-learn pipeline with TFIDF Vectorizer and Stochastic Gradient Descent classifier using log as loss function and L2 penalty, and pass our training data to the pipeline, and test the model on test data. Then we did hyper-parameter tuning on alpha=0.01, 0.001, 0.0001, 0.00001 and 5-fold cross validation to get the best result. Then we tried to remove stop words from our data and pass the data without stop words to the pipeline and repeat the steps mentioned above. For the best model, we plot the AOC-RUC curve and Precision-Recall curve:

1. TFIDF Vectorizer and Logistic Regression

We built the scikit-learn pipeline with TFIDF Vectorizer and Logistic Regression classifier with L2 penalty and pass our training data to the pipeline and test the model on test data. Then we did hyper-parameter tuning on c = 1.5, 1.0, 0.5, 0.2, 0.1 and 5-fold cross validation to get the best result. Then we tried to remove stop words from our data and pass the data without stop words to the pipeline and repeat the steps mentioned above. For the best model, we plot the AOC-RUC curve and Precision-Recall curve:

1. Count Vectorizer and Logistic Regression

We built the scikit-learn pipeline with Count Vectorizer and Logistic Regression classifier with L2 penalty, and pass our training data to the pipeline, and test the model on test data. Then we did hyper-parameter tuning on c = 1.5, 1.0, 0.5, 0.2, 0.1 and 5-fold cross validation to get the best result. Then we tried to remove stop words from our data and pass the data without stop words to the pipeline and repeat the steps mentioned above. For the best model, we plot the AOC-RUC curve and Precision-Recall curve:

1. Count Vectorizer and Naïve Bayes

We built the scikit-learn pipeline with Count Vectorizer and Naïve Bayes classifier with L2 penalty and pass our training data to the pipeline and test the model on test data. Then we did hyper-parameter tuning on alpha = 1.0, 0.5, 0.2, 0.1 and 5-fold cross validation to get the best result. Then we tried to remove stop words from our data and pass the data without stop words to the pipeline and repeat the steps mentioned above. For the best model, we plot the AOC-RUC curve and Precision-Recall curve:

* + 1. **Experiment with other classification models**

Besides the existing models, our second approach is to try the MLP Classifier and SVM Classifier on citation with both Count Vectorizer and TFIDF Vectorizer.

1. Count Vectorizer and MLP Classifier

We built the scikit-learn pipeline with Count Vectorizer and MLP Classifier and passed our training data to the pipeline and tested the model on test data. Then we did hyper-parameter tuning on alpha=0.01, 0.001, 0.0001, 0.00001, and activation function = ‘relu’, ’tanh’, ’logistic’, and hidden layer = 100(default), 50, 20, 10. Then we perform 5-fold cross validation to get the best result. Then we tried to remove stop words from our data and pass the data without stop words to the pipeline and repeat the steps mentioned above. For the best model, we plot the AOC-RUC curve and Precision-Recall curve:

1. TFIDF Vectorizer and MLP Classifier

We built the scikit-learn pipeline with TFIDF Vectorizer and MLP classifier and passed our training data to the pipeline and tested the model on test data. Then we did hyper-parameter tuning on alpha=0.01, 0.001, 0.0001, 0.00001, and activation function = ‘relu’, ’tanh’, ’logistic’, and hidden layer = 100(default), 50, 20, 10. Then we perform 5-fold cross validation to get the best result. Then we tried to remove stop words from our data and pass the data without stop words to the pipeline and repeat the steps mentioned above. For the best model, we plot the AOC-RUC curve and Precision-Recall curve:

1. Count Vectorizer and SVM Classifier

We built the scikit-learn pipeline with Count Vectorizer and SVM Classifier with L2 penalty and pass our training data to the pipeline and test the model on test data. Then we did hyper-parameter tuning on kernel = ‘linear’, ‘poly’,’rbf’ and then performed 5-fold cross validation to get the best result. Then we tried to remove stop words from our data and pass the data without stop words to the pipeline and repeat the steps mentioned above. For the best model, we plot the AOC-RUC curve and Precision-Recall curve:

1. TFIDF Vectorizer and SVM Classifier

We built the scikit-learn pipeline with TFIDF Vectorizer and Logistic Regression classifier with L2 penalty and pass our training data to the pipeline and test the model on test data. Then we did hyper-parameter tuning on kernel = ‘linear’, ‘poly’,’rbf’ and then performed 5-fold cross validation to get the best result. Then we tried to remove stop words from our data and pass the data without stop words to the pipeline and repeat the steps mentioned above. For the best model, we plot the AOC-RUC curve and Precision-Recall curve:

* + 1. **The best model for citation classifier**

After completing the above approaches, we selected our best model based on the precision, recall, and F1 score of the models. The classifier we choose is TFIDF Vectorizer and Stochastic Gradient Descent using log as loss function and L2 penalty.

* 1. **Functionality Classifier**
     1. **Latent Dirchlect Allocation (LDA)**

One of the challenges we faced was in labelling the functionality of repositories since their descriptions were a bit vague. We decided to use a topic modelling technique such as LDA to identify different topics in the corpus based on which we decided to annotate the functionality of repositories in our corpus. The diagram below shows the approach behind this process.

Diagram

Description automatically generated

Fig 3.6.1: LDA Workflow

We used the annotated description of repositories in our corpus to train our LDA model in order to identify distribution of words in various topics. The figure 3.6.2 summarizes the results of the LDA model.

Chart, bubble chart

Description automatically generated

Fig 3.6.2: LDA results visualization

We see that there are five different topics in the corpus out of which three topics are clustered close to each other, we observed that the words in these topics more or less talked about visualization, we decided to label functionality of repositories having these mentions as Data Visualization. The other two topics had mentions about libraries used in machine learning and deep learning such as scikit-learn, tensorflow, pytorch and keywords like processing, transformation, etc. We agreed upon labelling the two topics as Data Analysis and Data Preparation respectively. LDA helped us in resolving ambiguities with respect to labelling of functionalities and helped us proceed in the right direction.

* + 1. **Functionality Classifier**

1. Data Processing

For three categories, which are data analysis, data visualization, and data preparation, we define them as 0, 1, and 2 at first. After that, we look up the same URL on function.csv and add the label column to the instance on extented\_description.csv, so that we can generate our corpora for functionality processions, named description\_corpura. At the same time, we analyze our distribution (TABLE 6.2.1) of corpus to check out the balance, and then we extract “excerpt” column and “label” column to x variable and y variable, named x\_func and y\_func.

|  |  |
| --- | --- |
| Category | Count |
| Data Analysis | 117 |
| Data Visualization | 139 |
| Data Preparation | 75 |

Table 3.6.1: The Distribution of Functionality Corpus

1. CountVectorizer and LogisticRegression

We pass our data to the pipeline, and the steps are CountVectorizer and LogisticRegression. Because our categories are multi-class, we cannot apply the same multi\_class parameter of LogisticRegression function to other classifiers, so we shift the multi\_class to “multinomial”. Then we tried a different penalty and solver parameter combination, for instance, the penalty is “l2” and the solver is “lbfgs”, “newton-cg”, ”sag”, or “saga”. For L1 penalty, only “saga” can be worked with, and then we pick the best parameters for LogisticRegression which are “multinomial”, “lbfgs”, and “l2”. To get the accuracy score, we should set the average to “macro” because we are based on multi-class. The results of the model shown in TABLE 3.6.2.

1. TfidfVectorizer and LogisticRegression

As same as above, we make the pipeline in which TfidfVectorizer(default) is followed with LogisticRegression, and, after passing data, we tune the parameters. The best parameters for this pipeline are that the solver equals “liblinear”. The results of the model shown in TABLE 3.6.2.

After Comparing with CountVectorizer and LogisticRegression, there is no huge difference between CountVectorizer and TfidfVectorizer, and the TfidfVectorizer is more stable than CountVectorizer while we run 10 times. Therefore, we decided to use TfidfVectorizer for the following research.

1. TfidfVectorizer and NaiveBayes

By understanding the previous research, we find out that the function of Naive Bayes, called MultinomialNB, can be solved by the multi-class problem, so we change to TfidfVectorizer(default) and MultinomialNB to train our model. The results of the model shown in TABLE 3.6.2.

Even though the scores have not improved a lot, it is getting more stable than before.

1. Word-Level TfidfVectorizer and NaiveBayes

For the TfidfVectorizer, there is a parameter called analyzer which can be chosen to “word”, so we tried to change it to enhance the accuracy. However, the accuracy comes from 0.34 to 0.53, which reminds us to abandon word-level TfidfVectorizer. Therefore, we do not make pipelines with word-level TfidfVectorizer on upcoming methods.

1. TfidfVectorizer and RandomForestClassifier

Another function that we tried is RandomForestClassifier, and we combine it with TfidfVectorizer(default). The results of the model shown in TABLE 3.6.2. Because the accuracy is not over 0.60, it is impossible to get the highest scores ever by tuning any parameters.

1. TfidfVectorizer and Perceptron

According to the previous research, we try Perceptron with TfidfVectorizer(default). The results of the model shown in TABLE 3.6.2. This model is getting the best score, and It can basically stabilize above 70% accuracy, even up to 75%.

1. Removing Stop words

Because the model of TfidfVectorizer and Perceptron is the best, we decided to use new excerpts, which remove stop words by using the nltk package, and it is applied to the model. The results of the model shown in TABLE 3.6.2. However, after removing stop words, the results become unstable and lower than normal.

Therefore, we export the TfidfVectorizer and Perceptron without removing stop words as a final functionality model to apply to FAMEF, and the bar graph is shown on Fig 3.6.3.

Chart, bar chart

Description automatically generated

Fig 3.6.3: The Bar Graph of Functionality Model

|  |  |  |  |
| --- | --- | --- | --- |
| sklearn Primitive - Functionality | Accuracy | Recall | F-measure |
| CountVectorizer + LogisticRegression | 0.92 | 0.98 | 0.93 |
| TfidfVectorizer + LogisticRegression | 0.89 | 0.98 | 0.91 |
| TfidfVectorizer + NaiveBayes | 0.89 | 0.97 | 0.91 |
| TfidfVectorizer (Word Level) + NaiveBayes | 0.88 | 0.98 | 0.91 |
| TfidfVectorizer+RandomForestClassifier | 0.88 | 0.98 | 0.91 |
| TfidfVectorizer + Perceptron | 0.87 | 0.86 | 0.89 |
| TfidfVectorizer + Perceptron (After Remove Stop Words) | 0.84 | 0.78 | 0.85 |

Table 3.6.2: The Results of Functionality Classifier

* 1. **Result JSON File**

1. **Reproducibility** 
   1. **Datasets**

Our training data and test data for each classifier are randomly picked from extented\_description.csv [2], extended\_invocation.csv [3], extended\_citation.csv [4], extended\_installation.csv [5], and function.csv [6]. The developers can expand the CSV files to generate more instances to train and test.

* 1. **Repository**

For now, we upload our framework-based software to the Github repository [1] for researchers and developers who can freely use or build on FAMEF.

* 1. **Modeling Notebooks**

We also have jupyter notebooks [7] for showing every model and result of each classifier. The developers can check each model in detail on our notebooks.

* 1. **Useful Scripts**

Our scripts [8] are able to invoke classifiers’ models [10], and it will crawl the README file and basic information of the target repository. By phasing information to the dictionary data structure, the scripts [8] can predict the excerpts of each classifier and save the results as a JSON file. The developers can build on our software by understanding all scripts.

* 1. **Installation Instructions**

To mount FAMEF, you can clone the repository [1] by running git clone <https://github.com/liling10822/FAMEF.git>. Using cd FAMEF to go to the folder that you cloned, we can run pip install -e . to mount all requirements for FAMEF. For testing FAMEF installation, we can try somef --help, and If everything goes fine, following message will appear to your terminal:

Usage: somef [OPTIONS] COMMAND [ARGS]...

Options:

  -h, --help  Show this message and exit.

Commands:

  configure  Configure credentials

  describe   Running the Command Line Interface

  version    Show somef version.

* 1. **Visual Environment Setup**

For installing FAMEF in visual environment, at first, we should install virtualenv package on our computer by running the command line pip install virtualenv, after then, we need to create a blank virtual environment in FAMEF folder, called env, by running the command line below:

* For macOS and Linux: python3 -m venv env
* For Windows: py -m venv env

To activate the env environment by running the command line below:

* For macOS and Linux: source env/bin/activate
* For Windows: .\env\Script\activate

After activating, it will independently install FAMEF in the virtual environment by following the instructions. For deactivate the env environment, we can execute deactivate on terminal.

* 1. **Docker Container**

We also provide the Docker container to directly install FAMEF by using Docker file [9]. After installing Docker, we can build the image by executing docker build --tag [tagName] ., and, to run Docker container, the command line should be docker run -p 5006:5006 -it [tagName].In our Dockerfile, it will help to execute git clone the repository and install FAMEF. It could be added to other commands to the Docker file [9] to explore further.

1. **Communication with Stakeholder**

* The stakeholder involved in the project is the client who provided us an interesting problem.
* We had a number of discussions with our client which helped us to correct our approach from time to time.
* In particular we had two such meetings, apart from our final project presentation.
  1. **Meet One**
* For this meeting, our strategy was to communicate our scoping of the project to the client.
* Our Project Manager communicated with the client on our behalf.
* She discussed about the objectives of the project, the responsibilities of different team members, and a tentative deadline for the project.
* We wanted the client to know that we have a solid plan for the success of this project.
* The Project Manager also communicated the functional, non-functional requirements as well as the user stories associated with this project.
* We also felt the need to communicate the risks of the project, particularly of the covid19 situation.
* The response from the client was very constructive for us.
* This made us re-calibrate our metric goals for the project, and also polish our objectives and requirements a bit more.
  1. **Meet Two**
* In this meeting, we wanted to discuss the first iteration of our project with our client.
* This time Pratheek volunteered to present this information on our behalf.
* We talked in depth about how we were going to extend our existing repository count.
* We also talked about the possible machine learning algorithms we were going to use for each of the categories of description, installation, citation and invocation.
* We also presented a workflow of our entire project, and the plan for the remainder of the work
* However, this meeting did not go the way we would have ideally wanted it to go.
* Looking back, all the team members did agree that we were not clear in communicating our work and plans to the client in this meeting.
* We did not convey a concrete plan regarding our functionality extractor to the client.
* Additionally, this meeting made us realize that we’ll need to analyze our existing dataset more thoroughly, before we start exploring new repositories.
  1. **Final Presentation**
* Our project finally concluded with a 30 minutes presentation and Q/A session with the client.
* We kept the presentation as visually discernible as possible, without burdening the reader to go complicated texts and logic.
* We explained our approaches in brief, starting from data annotation, data preparation, building of models to the final obtained results.
* Towards the end of our presentation, we also demonstrated how our software could be run and displayed the results for a couple of GitHub repositories.
* For the Q/A session, we had decided that each question would be tackled by the person who had specifically worked on that particular part of the software.
* Overall, we were able to successfully present our project to our stakeholder and answer his/her questions effectively.

1. **Conclusion and Future Work**

In FAMEF, we improved the classifiers from SOMEF for description, invocation, installation, citation and built a multi class classifier for software functionality. We extended the training corpus from 89 readme files to 124 readme files and tried different kinds of classification model like logistic regression, Stochastic Gradient Descent, random forest, and multi-layer perceptron. We also fine tuned each model by performing hyper-parameter tuning and cross validation. From this project, we learnt that more complex model do not necessarily mean better result. We choose simpler model like logistic regression and Stochastic Gradient Descent over deep learning or ensemble methods for all the tasks.

While collecting and analyzing readme files from github.com, we found that some software is much more well-documented than others. We are able to extract description, invocation, installation, citation information and classify their functionality from those documentation to help scientist, researchers to learn about this software easily and quickly. We believe all software engineers should try to write more detailed documentation so that their work can be used by more people and it would save user many time and efforts.

Due to the time and resource limitation, we are not able to fully expand FAMEF. There is still much potential for future works. We could try NLP techniques like word2vec to further improve the classifiers. Besides, we could create a graphic User interface so our user can use FAMEF by clicking a few bottoms. And we can also show our classification result more clearly. Last but not least, we can extend our Framework to the open-source repositories from other websites such as gitlab, bitbucket and sourceforge.

**Appendix**

[1] Github repository: <https://github.com/liling10822/FAMEF>

[2] extended\_description.csv: <https://figshare.com/articles/dataset/SOMEF_Extension/13218755?file=25457465>

[3] extended\_invocation.csv:

<https://figshare.com/articles/dataset/SOMEF_Extension/13218755?file=25457474>

[4] extended\_citation.csv:

<https://figshare.com/articles/dataset/SOMEF_Extension/13218755?file=25457462>

[5] extended\_installation.csv:

<https://figshare.com/articles/dataset/SOMEF_Extension/13218755?file=25457471>

[6] function.csv: <https://figshare.com/articles/dataset/SOMEF_Extension/13218755?file=25457477>

[7] Notebooks: <https://github.com/liling10822/FAMEF/tree/main/experiments/training_allen>

[8] Scripts:<https://github.com/liling10822/FAMEF/tree/main/src/somef>

[9] Dockerfile:<https://github.com/liling10822/FAMEF/blob/main/Dockerfile>

[10] Classifier models: <https://github.com/liling10822/FAMEF/tree/main/src/somef/models>