A Report for Functionality and Metadata Extraction Framework (FAMEF) — An Extension Project of SOMEF

Authors: Pratheek Athreya, Ling Li,
Sharad Sharma, Yi Xie, Yidan Zhang (PM)

1. Introduction

1.1. Motivation

The applications of Computer Science in various scientific fields in the recent past have 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 [1]. 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).

1.2. 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 1.2.1.

	Accuracy	**Precision**	**Recall**	**F-measure**
Description	0.83	0.85	0.79	0.82
Invocation	0.88	0.86	0.94	0.9
Installation	0.9	0.92	0.9	0.91
Citation	0.92	0.89	0.98	0.93

Table 1.2.1: Existing baseline scores of classifiers used in SOMEF

2) Develop a multi class classifier to extract the functionality of repositories with an accuracy above the baseline (null accuracy: 33%)

1.3. User Stories

1) As a researcher I would like to make use of existing scientific software in order to facilitate and expedite my work.
2) 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.
3) 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.
4) As a researcher I want to compare similar software before using them in the industry in order to be more efficient.

1.4. Requirements

1.4.1. Functional

1) 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.
2) 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.
3) The framework must generate satisfactory results with new unseen repositories.

1.4.2. Non-Functional

1) The system should build on the existing work from SOMEF.
2) The framework must generate output within five minutes.

1.4.3. Team

1) Skilled Data Scientists having hands on knowledge in training machine learning models and interpreting the results in order to convey them to an audience.
2) A project manager to supervise and monitor status updates in order to achieve the project deliverables.

1.5. Evaluation Metrics

- **Precision:** Extract the correct piece of information and improve the existing score. For instance, if 15 excerpts are classified as description and 12 of them belong to description, then the precision for description classifier would be 0.8 (TP/Predicted Yes)
- **Accuracy:** It is a measure of how often the classifier is correct. For example, if there are 20 excerpts, 15 of which belong to description and 5 belong to other categories, and if our classifier classifies 12 of them as description, the accuracy would be 0.85 (TP+TN/Total).
- **Recall:** It is a fraction of relevant excerpts that have been correctly classified. For example, if there are 13 excerpts that belong to description and our classifier identifies 12 excerpts as description, the recall would be 12/13 (TP/TP+FN).
- **F-measure**: Achieving a balance between precision and recall, as it is important that we extract maximum information without losing much precision.

1.6. Risks

Risk Assessment Matrix

SCALE OF LIKELIHOOD		SCALE OF SEVERITY		
		ACCEPTABLE	TOLERABLE	GENERALLY UNACCEPTABLE
	NOT LIKELY	LOW : Annotation Disagreement	MEDIUM : Not finding a lot of new repositories	MEDIUM : Not Completing project on time
	POSSIBLE	LOW: Argument with team members	MEDIUM: Extending the corpus and not observing a lot of improvement in performance of classifiers	HIGH: Health hazards due to COVID 19
	PROBABLE	MEDIUM: Trying new classifiers and not being able to improve the existing baseline	HIGH: Misinterpreting client's requiremets	HIGH: Not delivering what is promised to the client

Table 1.6.1: Risk Assessment Matrix

1.7. Risk 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.

1.8. 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. 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.

1.9. 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.

2. Project Management and Planning

2.1. Project Management Methodology We Chose and Reasons

A suitable methodology for managing a project can help a team achieve goals and improve efficiency, which is very important for both team and project. Before we started the project, we needed to decide on our methodology of project management. Based on the background and experience of team members, we decided to choose from Waterfall, Agile Scrum, and Kanban.

2.1.1. Waterfall - Not Used

We ruled out Waterfall for two reasons.

- **Small Project**: As we know, Waterfall is usually structured as one big project in a sequential process, while FAMEF is only a small extension project of SOMEF. As mentioned earlier, the goals of FAMEF were to improve performance and add one feature without involving the system design of the entire software.
- **Technology Not Clear**: Waterfall works well when the team understands technology very clearly. Obviously, it did not fit our case very well. To improve the performance of the software, we probably would explore new methods and technologies. We could not guarantee that we understood these techniques and methods at that time.

Hence, Waterfall was not our option.

2.1.2. Agile Scrum - Not Used

We decided not to use Agile Scrum either. It was a hard decision for us since Scrum is very popular and frequently used by software development teams. Although it can be applied to all kinds of teamwork, it is not the most suitable for our team and project for two reasons.

- **Not Flexible**: Scrum requires regular fixed-length sprints and the team cannot make changes during the sprint, which is not flexible enough for our team. Since all team members were temporary and part-time workers, we had to finish the project in a limited time, which required us to be able to add or reduce tasks accordingly.
- **Complex Role Requirement**: Scrum requires three kinds of roles in a team: product owner, scrum master, and development team, which is complex for a small project

Therefore, Agile Scrum did not suit us and FAMEF.

2.1.3. Kanban — What We Used

We chose Kanban as our methodology for three reasons.

- **Limit Work**: Kanban can maximize customer value and efficiency and reduce the time it takes to take a task or user story from start to finish. We could limit the number of “work in process” activities, which would make people only focus on specific tasks and ensure the accomplishment of all open tasks.
- **Flexible**: It is flexible that we could manage tasks based on our schedule and continuously improve our flow of work.

- **Easy Communication**: Kanban visualizes work to make our communication easier and makes it easy for us to collaborate. We would turn to Github's Basic Kanban, which is a good tool to allow all team members to know clearly about what everyone should be doing and what tasks they have in their pipeline. Overall, Kanban is the most suitable methodology. We will demonstrate more details about how we used Github's Basic Kanban to manage our project later

2.2. Project Planning

We should assemble a project plan to bring our project to fruition. The plan would determine the wastes, the deliverables, and the milestones of the project, which could help the team understand the client's requirements and the goals. Besides, it includes a Gantt Diagram, which helped us in the managing our time.

2.2.1. Wastes

- **Transport**: Since the meetings with the client were only required for the project manager, the client had to talk to the project manager about the requirements and concerns, and then the project manager would communicate with the person in charge. This process involved a significant overhead. To avoid this waste, our project manager encouraged everyone to attend as much as possible.
- **Over-Production**: In the beginning, we forked the SOMEF and uploaded our code and CSV files on it. Later, we decided to create a new repository and called our project "FAMEF" so that we had to upload our existing code and CSV files to the new repository again.
- **Over-Processing**: Since we worked remotely, which made communication difficult, we sometimes had to explain the same questions multiple times.

2.2.2. Deliverables

- Create a Google Slides presentation including all information of the project scope
- Create 5 CSV files for extended corpus after adding new repositories
- Improve the performances of 4 existing classifiers
- Create a functionality classifier and add an extraction result to the output JSON file
- Create a Google Slides presentation including project summary, motivation, objectives, results and resources, and conclusions for final presentation
- Create a demo video for final representation
- Create result visualizations to show the performance improvements for presentation and report
- Publish a project report

2.2.3. Milestones

FAMEF is an extension of SOMEF Project. There are four milestones.

- **Scoping**: The project scope is very important to help the team understand the project and enables the project manager to allocate the proper labor to complete the project. To scope the project properly, the project manager should organize the team, learn about the background of team members, discuss the project, and communicate with the client timely. This milestone was focused on addressing these concerns and completing related documents, like slides and reports. Also, the project manager should check with the client, discuss within the team, and prepare for the development phase.
- **Four Existing Classifiers Improvement**: This milestone was focused on one of the main objectives of FAMEF. The team would work on improving the performance of the existing four classifiers, which includes description, installation, invocation, citation, by exploring different classification models.
- **Functionality Classifier Creation**: This milestone was focused on one of the main objectives of FAMEF. The team would create a new classifier — functionality — for the software and find the model with the best performance. Additionally, the developers should add the extraction result of functionality to the output JSON file.
- **Presentation (Demo) and Report**: This milestone would be focused on preparing for the presentation, presenting results to clients, and submitting the project report. To prepare for the presentation, the team would create result visualizations, record a demo video, publish slides, and attend a dress rehearsal.

2.2.4. Gantt Diagram

We created a Gantt Diagram to manage our time. It is shown below (Table 2.2.1 and Table 2.2.2)

Title	W1 (9/14 - 9/20)	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12 (11/30 - 12/2)
Scoping and Create a Google Slides Presentation for Project Scope		D1 | M1 (End on 9/25)										
Create 5 CSV Files after Extending Corpus		Start on 9/26			D2							
Improve the Performances of 4 Existing Classifiers (Sharad and Yi)								D3 | M2				
Create A Functionality Classifier and Add an Extraction Result To the Output JSON File (Pratheek, Ling and Yidan)								D4 | M3				
Create Google Slides for Presentation										D5 (End on 11/18)		
Create Demo Video For Presentation (Yidan and Ling)										D6 (End on 11/18)		
Create Result Visualizations For Presentation and Report										D7 (End on 11/18)		
Presentation (Demo) and Report										D8.1 | M4.1		D8.2 | M4.2

Table 2.2.1: Gantt Diagram

Deliverable	
Milestone	
W	Week

Table 2.2.2: Gantt Diagram Legend

2.3. Github Basic Kanban

We used Github Basic Kanban to manage our project. We created issues, which included our tasks, questions, and deliverables, and assigned them to someone to finish. After the weekly meeting, we would put tasks that had to be finished to the column called “In progress. New issues, would be assigned to the column called “To do”. We also checked with everyone about what they had completed the last week and put those in the column called “Done”. A screenshot of our Kanban is shown below (Figure 2.3.1).

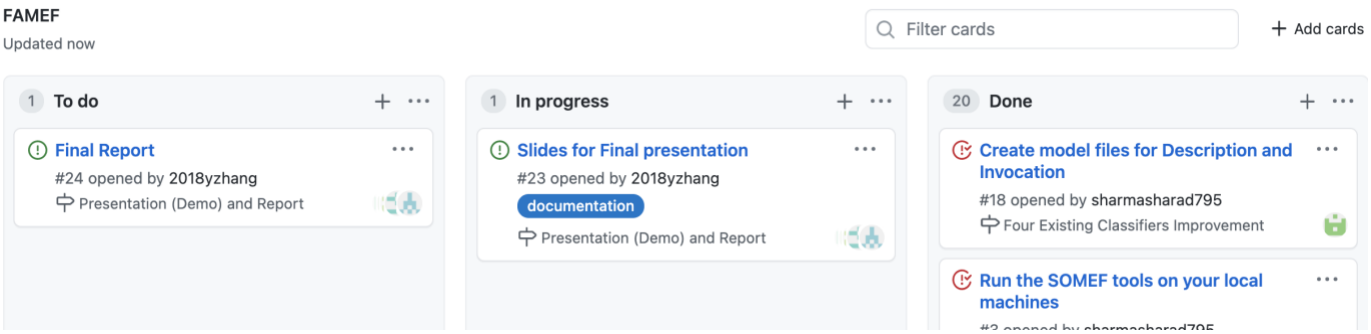

Figure 2.3.1: Github Basic Kanban Screenshot

2.4. Team Communication and Collaboration

Communication and collaboration are important for a project's success. Due to Covid-19, we had to work remotely, which made it more difficult for us to communicate and collaborate. But we had reduced this barrier by using different tools. To ensure the timeliness of communication within the team, we mainly used two applications: WhatsApp and Zoom. We used Google shared folder to collect and share all documents related to the project, such as slides, and meeting notes and we managed our code on Github.

2.4.1. Weekly Meetings

In the beginning, we had agreed to use Zoom to hold a weekly meeting every Tuesday afternoon from 4:30 to 6. After November 1, as the most important and final stage of the project, we had added a weekly meeting on Friday or Sunday depending on the schedule of all team members. At the Zoom meeting, we generally discussed four or five questions:

- Q1: What did everyone accomplish last week?
- Q2: What difficulties did everyone have and whether they required any help?
- Q3: What would we need to do this week?
- Q4: How to divide the work
- Q5: What should we improve according to the instructor's feedback — we would discuss this question if we had a meeting with the client that week)

We took turns writing the weekly meeting minutes, which should include the answers to Q3, Q4, and Q5. Sometimes we encountered some problems (Q2) that could not be solved. We would also record it so that one of us could ask the instructor on our half.

2.4.2. Daily Communication and Collaboration

We have a WhatsApp chat group to allow us to update individual processes and discuss issues at any time. We would timely provide help, suggestions, and ideas to someone in

need, so as to avoid stacking up problems in the weekly meeting and thus improve efficiency

2.4.3. Code Management

Github is a popular and easy tool to manage codes. We created a new Github repository called FAMEF under Ling Li's account to manage our codes (Appendix [1]). We required everyone to update the code before they were going to work on the project. Also, everyone needed to submit their updated code every night.

2.4.4. Meeting with the Client

We had four Zoom meetings with our client, Daniel Garijo. The table (Figure 2.4.1) below shows the topics we discussed and when. We appreciate all suggestions and feedback Mr. Garijo has provided. These meetings were only required for the project manager. Indeed, every team member attended these meetings, which helped them better understand the client's requirements and feedback.

Time	Agenda
2020/9/25	Scoping
2020/11/3	Data Pipeline and Results
2020/11/20	Presentation and Q&A

Figure 2.4.1: Client Meetings Overview

2.5. Workflow

This project can be distinctly 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, the 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.

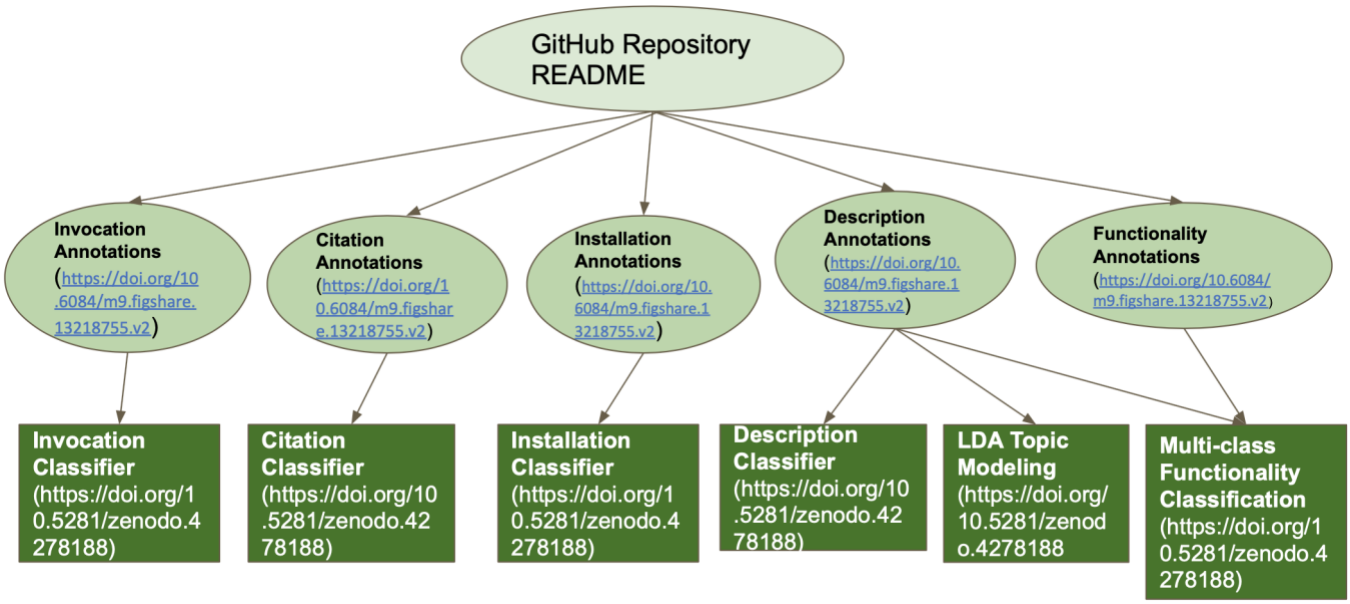

Figure 2.5.1: The Training Workflow

2) **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.

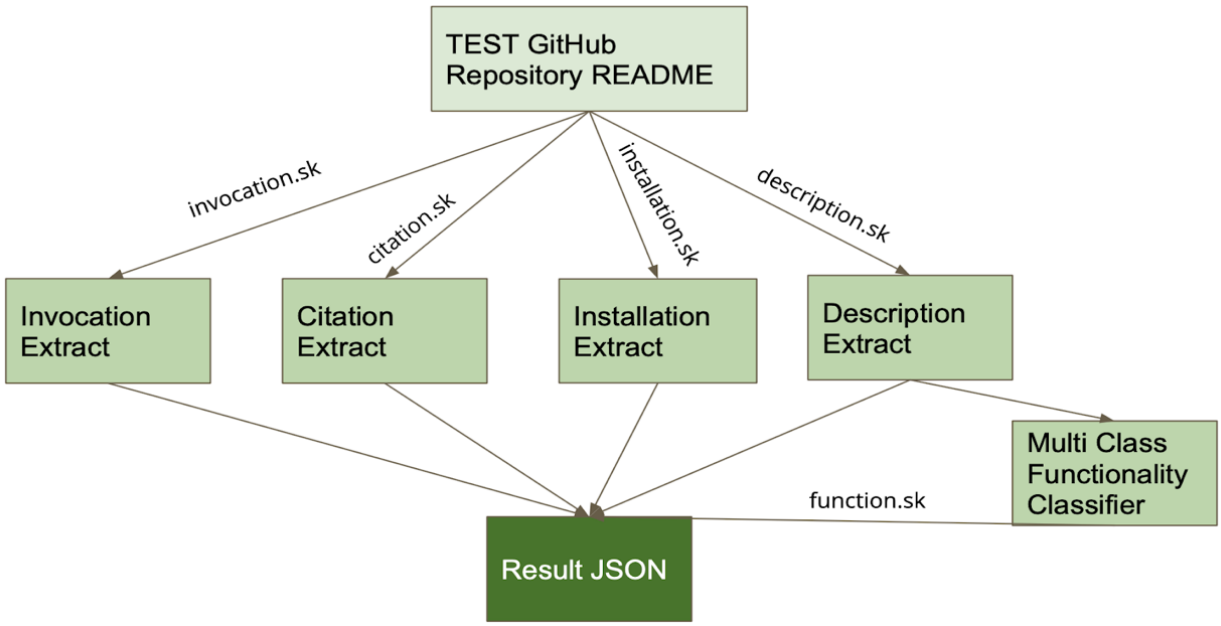

Figure 2.5.2: The Test Workflow

3. Project Process

3.1. Data Preparation

Since we built FAMEF based on SOMEF, we had 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.

3.1.1. 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.

3.1.2. 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).

URL	contributor	excerpt
https://github.com/GoogleChrome/puppeteer	Allen Mao	Puppeteer is a Node library which provides a high-level API to control Chrome or Chromium over the DevTools Protocol. Puppe
https://github.com/JimmySuen/integral-human-pose	Allen Mao	The major contributors of this repository include Xiao Sun, Chuankang Li, Bin Xiao, Fangyin Wei, Yichen Wei.
https://github.com/JimmySuen/integral-human-pose	Allen Mao	Integral Regression is initially described in an ECCV 2018 paper. (Slides).
https://github.com/JimmySuen/integral-human-pose	Allen Mao	We build a 3D pose estimation system based mainly on the Integral Regression, placing second in the ECCV2018 3D Human P

Figure 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” (Appendix [2]). The updated part is shown below (Figure 3.1.2).

336	https://github.com/RaRe-Technologies/gensim	Pratheek	Gensim is a Python library for topic modelling, document indexing and similarity retrieval with large corpora
337	https://github.com/RaRe-Technologies/gensim	Pratheek	Target audience is the natural language processing (NLP) and information retrieval (IR) community.
338	https://github.com/RaRe-Technologies/gensim	Pratheek	All algorithms are memory-independent w.r.t. the corpus size (can process input larger than RAM, streamed, out-of-cor
339	https://github.com/RaRe-Technologies/gensim	Pratheek	easy to plug in your own input corpus/datastream (trivial streaming API)

Figure 3.1.2: The Updated CSV File - extended_description.csv

3.1.3. 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” (Appendix [5]), which is shown below (Figure 3.1.3).

951	https://github.com/unagiootoro/ruby-dnn	Sharad	Add this line to your application's Gemfile:
952	https://github.com/unagiootoro/ruby-dnn	Sharad	gem 'ruby-dnn'
953	https://github.com/unagiootoro/ruby-dnn	Sharad	And then execute:
954	https://github.com/unagiootoro/ruby-dnn	Sharad	$ bundle

Figure 3.1.3: The Updated CSV File - extended_installation.csv

3.1.4. Invocation Extraction and the CSV file

Similarly, we updated the file related to invocation and called it "extended_invocation.csv" (Appendix [3]), which is shown below (Figure 3.1.4).

1515	https://github.com/Codecademy/EventHub	Yi Xie	Compile and run
1516	https://github.com/Codecademy/EventHub	Yi Xie	# set up proper JAVA_HOME for mac
1517	https://github.com/Codecademy/EventHub	Yi Xie	export JAVA_HOME=$(/usr/libexec/java_home)
1518	https://github.com/Codecademy/EventHub	Yi Xie	git clone https://github.com/Codecademy/EventHub.git
1519	https://github.com/Codecademy/EventHub	Yi Xie	cd EventHub

Figure 3.1.4: The Updated CSV File - extended_invocation.csv

3.1.5. Citation Extraction and the CSV File

We updated the citation CSV file and called it "extended_citation.csv" (Appendix [4]), which is shown below (Figure 3.1.5).

391	https://github.com/CMU-Perceptual-Computing-Lab/openpose	Ling Li	(the face detector was trained using the same procedure than the hand detector).
392	https://github.com/CMU-Perceptual-Computing-Lab/openpose	Ling Li	@article{8765346,
393	https://github.com/CMU-Perceptual-Computing-Lab/openpose	Ling Li	author = {Z. {Cao} and G. {Hidalgo Martinez} and T. {Simon} and S. {Wei} and Y. A. {Sheikh}},
394	https://github.com/CMU-Perceptual-Computing-Lab/openpose	Ling Li	journal = {IEEE Transactions on Pattern Analysis and Machine Intelligence},
395	https://github.com/CMU-Perceptual-Computing-Lab/openpose	Ling Li	title = {OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields},
396	https://github.com/CMU-Perceptual-Computing-Lab/openpose	Ling Li	year = {2019}

Figure 3.1.5: The Updated CSV File - extended_citation.csv

3.1.6. 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" (Appendix [6]). 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).

https://github.com/amitt001/delegator.py	Pratheek	Data Analysis	Delegator.py is a simple l
https://github.com/simbody/simbody	Yidan Zhang	Data Visualization	Wide variety of joint, con
https://github.com/cyverse/atmosphere	Yidan Zhang	Data Preparation	Atmosphere is an integra

Figure 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.

3.1.7. 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, citation and TreeBank 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).

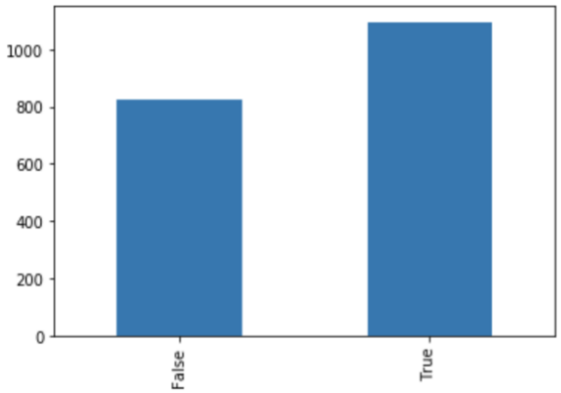

Figure 3.1.7: The Distribution Graph of Description

2) Installation Dataset Balancing

   To balance the installation dataset, we selected the same number of instances from description, invocation, citation and TreeBank 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).

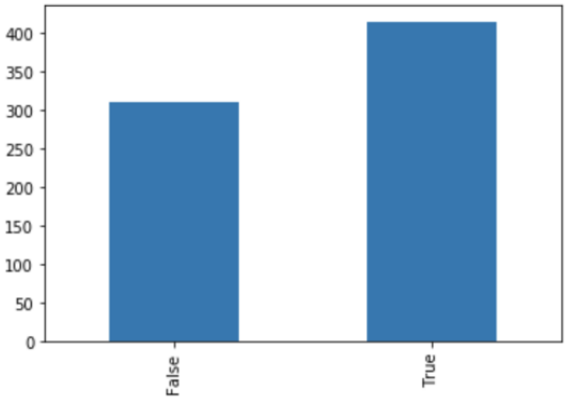

Figure 3.1.8: The Distribution Graph of Installation

3) Invocation Dataset Balancing

To balance the invocation dataset, we selected the same number of instances from installation, description, citation and TreeBank 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).

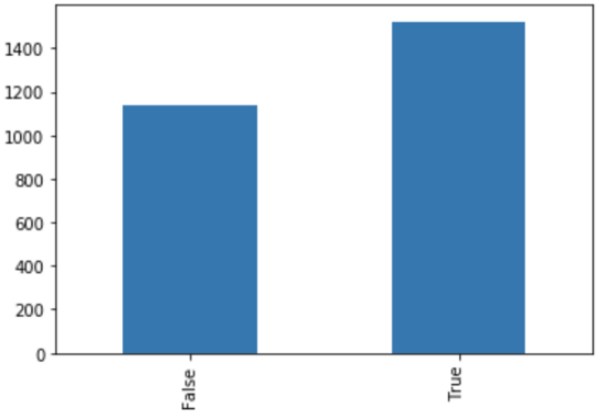

Figure 3.1.9: The Distribution Graph of Invocation

4) Citation Dataset Balancing

To balance the citation dataset, we selected the same number of instances from installation, invocation, description and TreeBank 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).

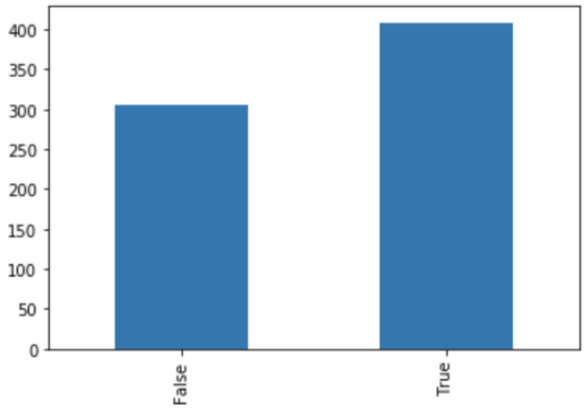

Figure 3.1.10: The Distribution Graph of Citation

3.2. 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).

3.2.1. 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.

3.2.2. 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

3.2.3. 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) CountVectorizer, followed by Logistic Regression
2) TfidfVectorizer, followed by the Random Forest Classifier
3) TfidfVectorizer, followed by the XGBoost Classifier
4) CountVectorizer, followed by Perceptron

The results have been mentioned in Table 3.2.1. 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.

3.2.4. 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 3.2.1. 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.

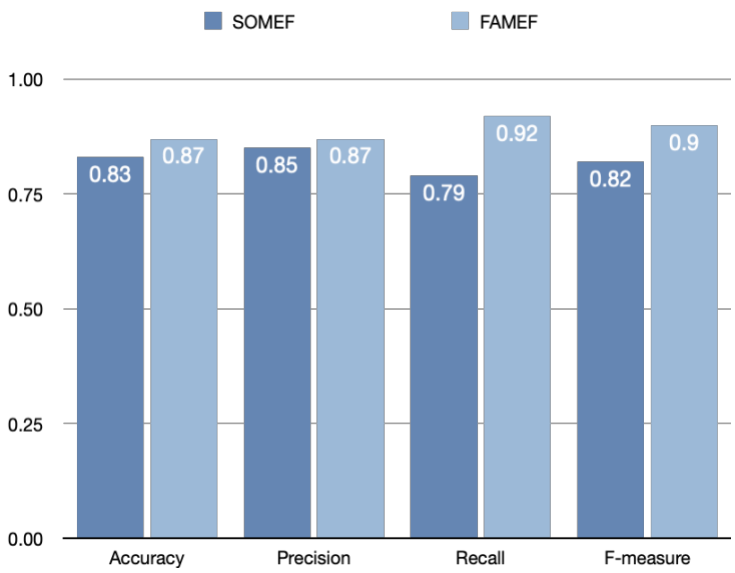

Figure 3.2.1

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

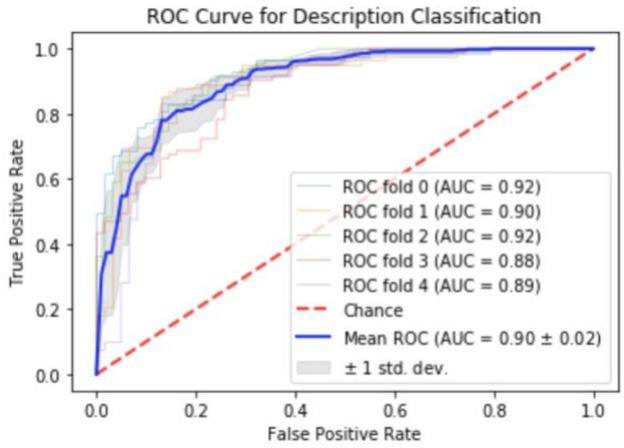

Figure 3.2.2

Below is the precision - recall curve:

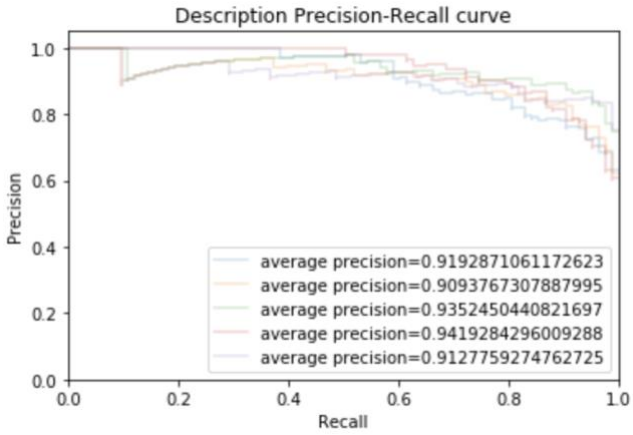

Figure 3.2.3

3.2.5. 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.

2) 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 3.2.1.

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

Table 3.2.1: The Results of Description Classifier

3.3. 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).

3.3.1. 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.

3.3.2. 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.

3.3.3. 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 3.3.1. 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.

3.3.4. 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 3.3.1. 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.

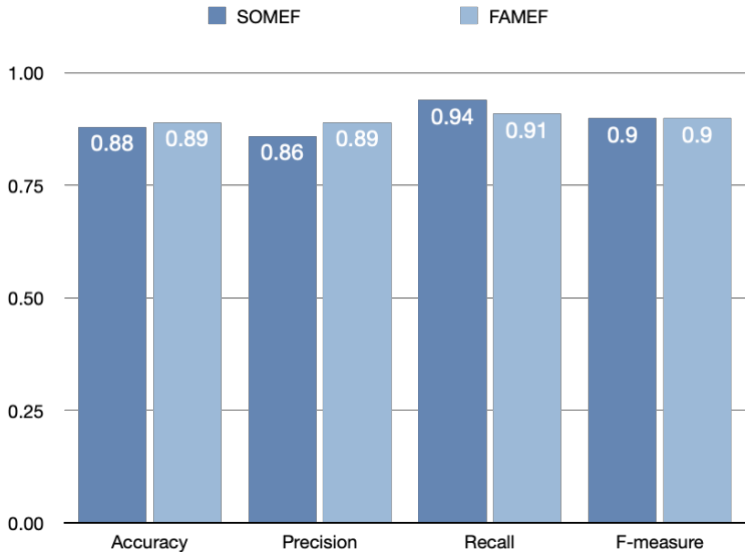

Figure 3.3.1

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:

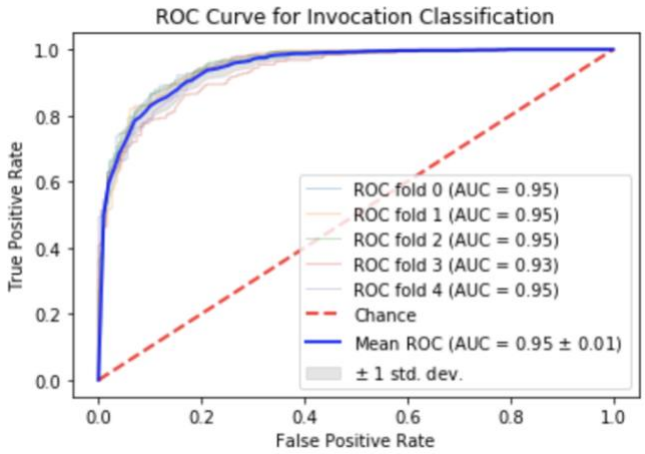

Figure 3.3.2

Below is the accuracy - recall curve:

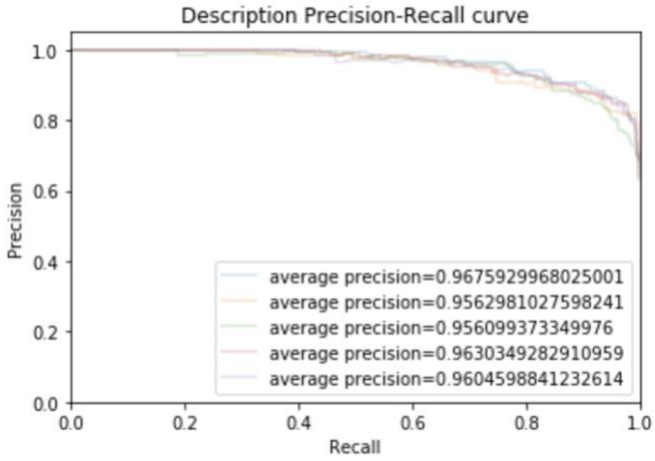

Figure 3.3.3

3.3.5. 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 3.3.1.

Invocation Models	**Accuracy**	**Precision**	**Recall**	**F-measure**
TfidfVectorizer+MultinomialNB (alpha hyperparameter tuning)	0.89	0.89	0.91	0.9
CountVectorizer+ MultinomialNB (alpha hyperparameter tuning)	0.89	0.89	0.93	0.91
TfidfVectorizer+StochasticGradientDescent (with hyperparameter tuning)	0.88	0.9	0.9	0.9
TfidfVectorizer+SelectFromModel(logit)+ MultinomialNB (alpha hyperparameter tuning)	0.87	0.86	0.93	0.89

CountVectorizer+ LogisticRegression (L2 Penalty with hyperparameter tuning)	0.86	0.85	0.93	0.89
TfidfVectorizer+RandomForestClassifier	0.72	0.69	0.96	0.8
SOMEF Baseline	0.88	0.86	0.94	0.9

Table 3.3.1: The Results of Invocation Classifier

3.4. Citation Classifier

- A citation classifier is to predict whether a sentence is about citation information or not.
- A citation excerpt usually contains information such as the author's name, license, article title, book or journal title, and organization name.
- Each sentence can be categorized into two classes: 1 means that it is a citation sentence, 0 means it is not.
- We split our extended corpus into training and testing sets, the percentages are 75% and 25% respectively.

3.4.1. Retrain and improve top 4 models from SOMEF

We found the top 4 models from SOMEF and retrained them on our extended corpus. Then we tried to improve them using multiple approaches. We will discuss each of the models in detail.

1) Count Vectorizer followed by Naïve Bayes

We built the scikit-learn pipeline with Count Vectorizer and Naïve Bayes classifier 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:

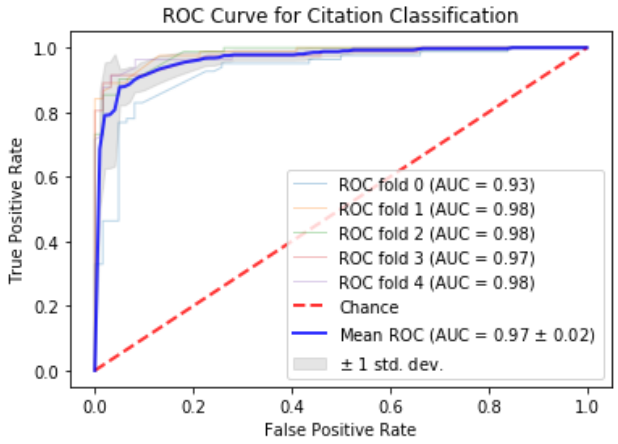

Fig 3.4.1

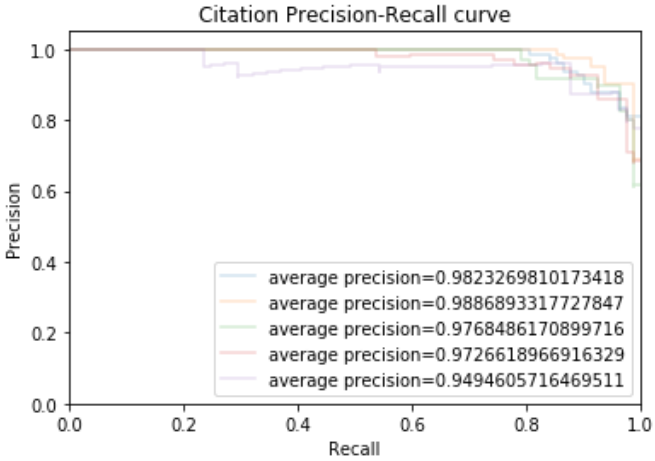

Fig 3.4.2

2) Count Vectorizer followed by 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:

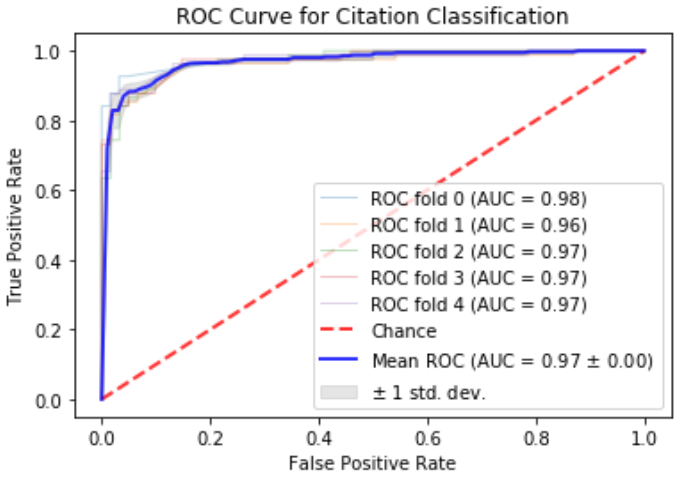

Fig 3.4.3

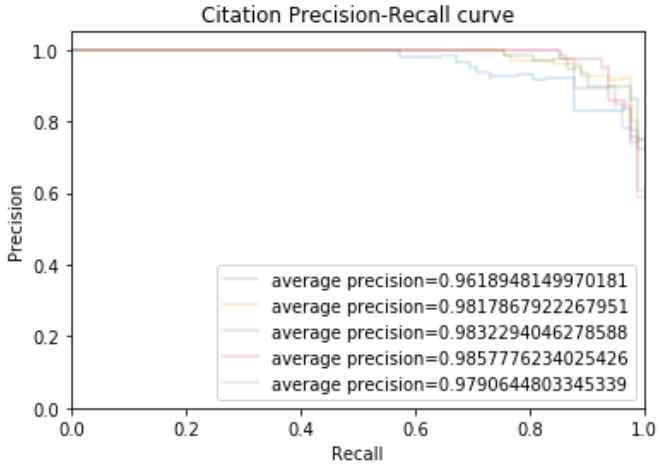

Fig 3.4.4

3) TFIDF Vectorizer followed by Stochastic Gradient Descent

We built the scikit-learn pipeline with TFIDF Vectorizer and Stochastic Gradient Descent classifier using the 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:

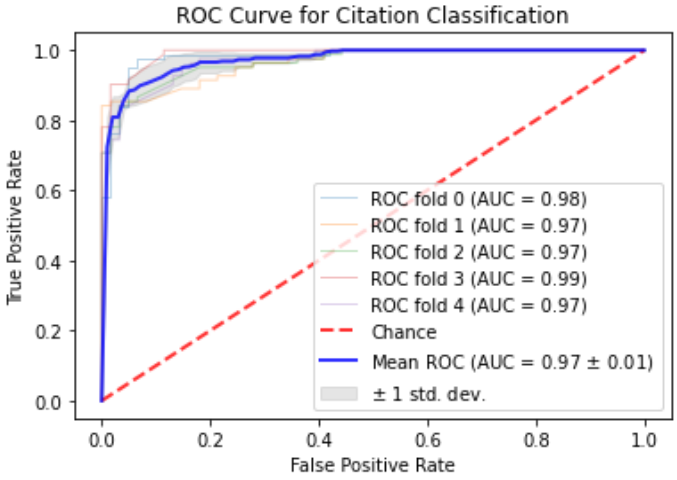

Fig 3.4.5

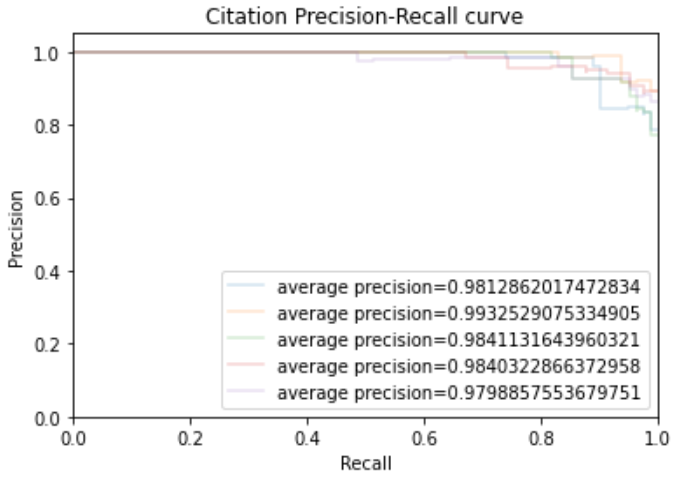

Fig 3.4.6

4) TFIDF Vectorizer followed by 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:

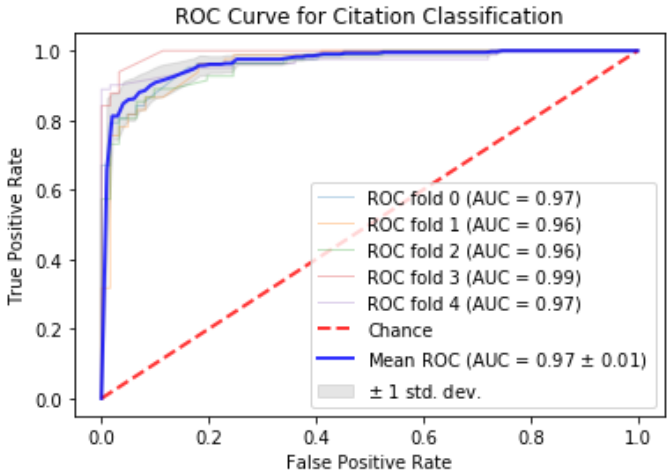

Fig 3.4.7

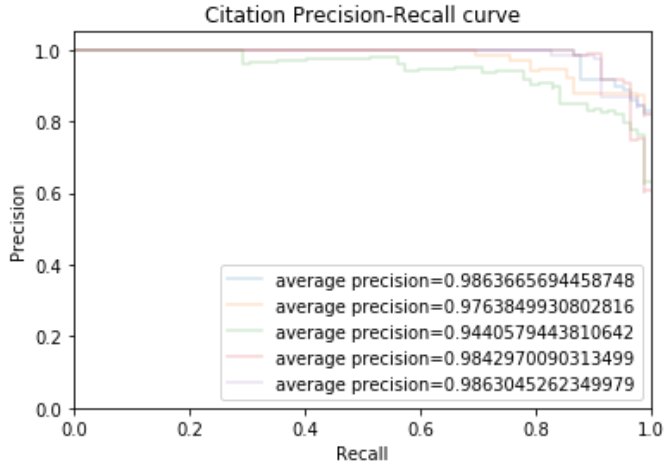

Fig 3.4.8

3.4.2. 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 followed by MLP Classifier

We built the scikit-learn pipeline with Count Vectorizer and MLP Classifier 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 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:

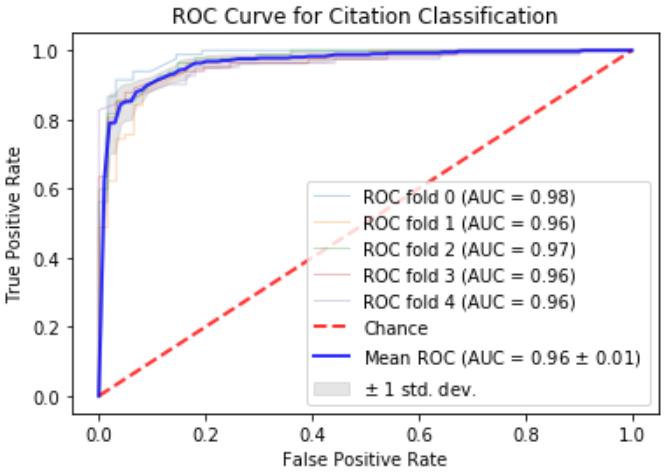

Fig 3.4.9

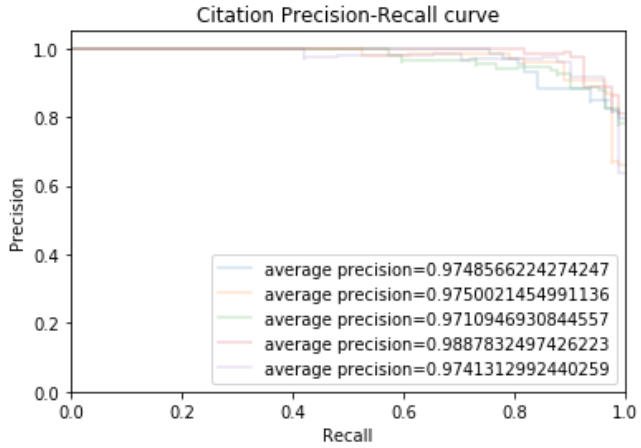

Fig 3.4.10

2) TFIDF Vectorizer followed by MLP Classifier

We built the scikit-learn pipeline with TFIDF Vectorizer and MLP classifier 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 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:

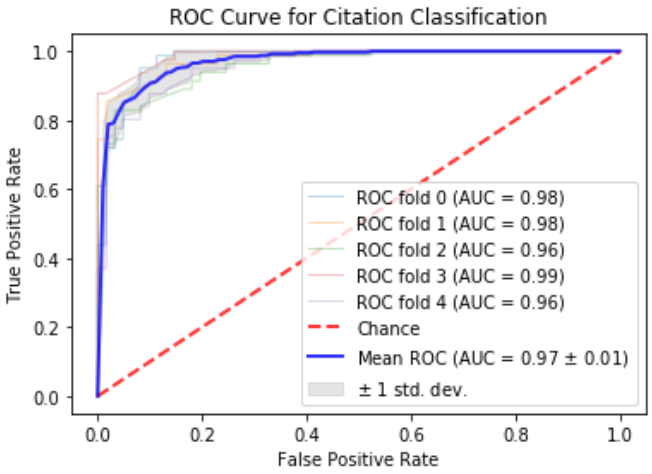

Fig 3.4.11

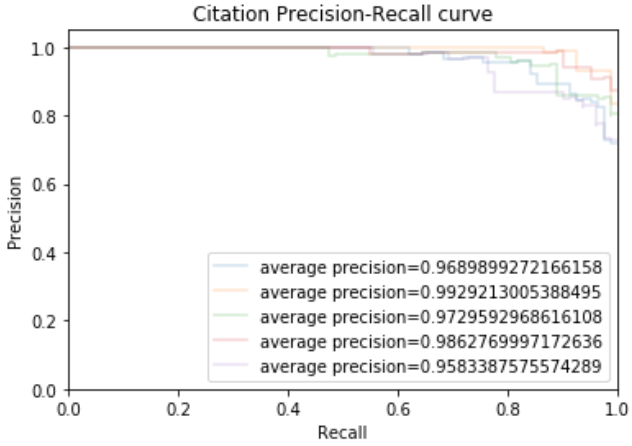

Fig 3.4.12

3) Count Vectorizer followed by 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:

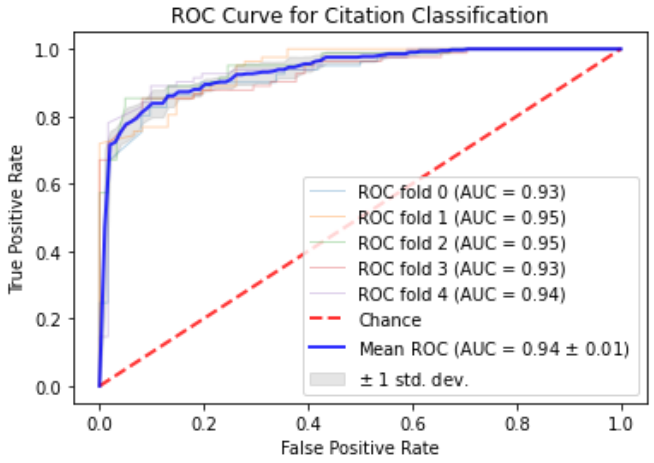

Fig 3.4.13

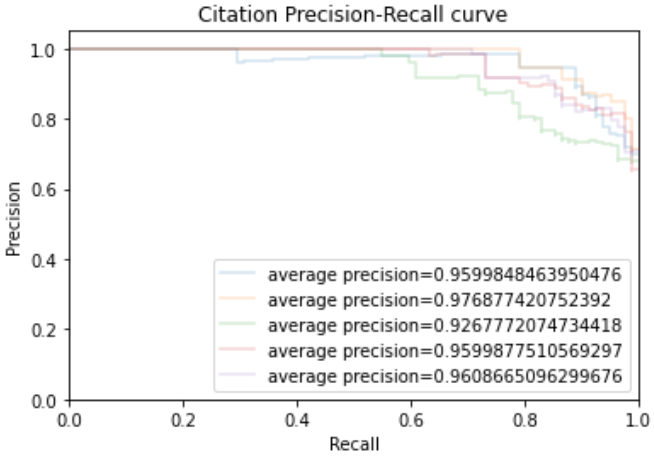

Fig 3.4.14

4) TFIDF Vectorizer followed by 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:

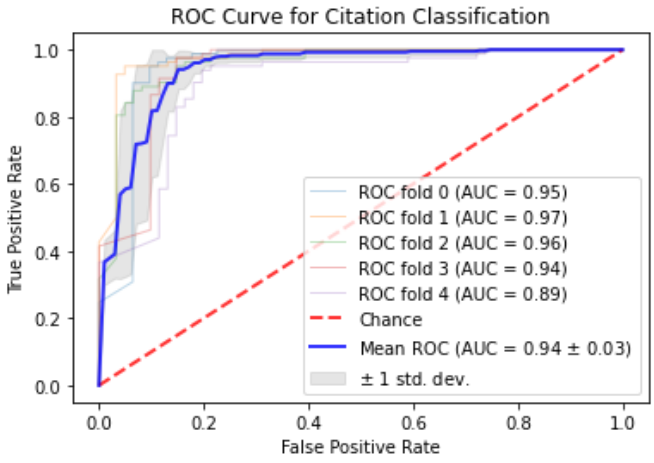

Fig 3.4.15

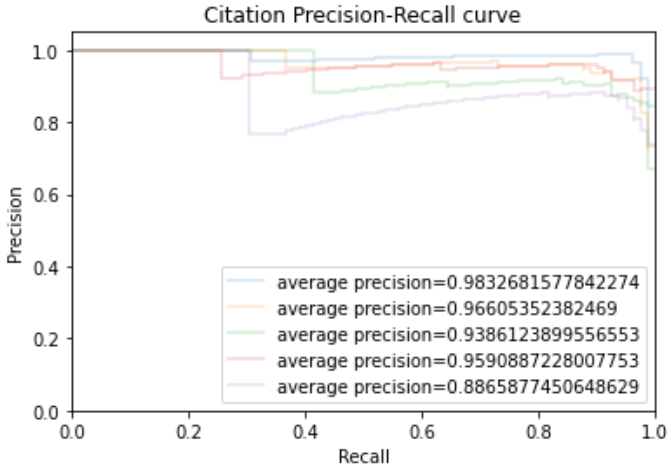

Fig 3.4.16

3.4.3. 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 the log as loss function and L2 penalty. The ranking of all models is shown in the following table 3.4.1.

Sklearn Primitive-Citation	**Accuracy**	**Precision**	**Recall**	**F-measure**
sklearnpipeline(TFIDFVectorizer, StochasticGradientDescent)	0.95	0.96	0.94	0.95
Sklearnpipeline(CountVectorizer, NaiveBayes)	0.93	0.94	0.92	0.93
Sklearnpipeline(CountVectorizer, MLPClassifier)	0.93	0.93	0.93	0.93

Sklearnpipeline(TFIDFVectorizer, LogisticRegression)	0.92	0.93	0.9	0.91
Sklearnpipeline(TFIDFVectorizer, MLPClassifier)	0.91	0.92	0.89	0.9
Sklearnpipeline(TFIDFVectorizer, SVMClassifier)	0.91	0.91	0.9	0.9
Sklearnpipeline(CountVectorizer, LogisticRegression)	0.88	0.89	0.87	0.88
Sklearnpipeline(CountVectorizer, SVMClassifier)	0.76	0.78	0.73	0.74

Table 3.4.1 Rankings of citation classifiers

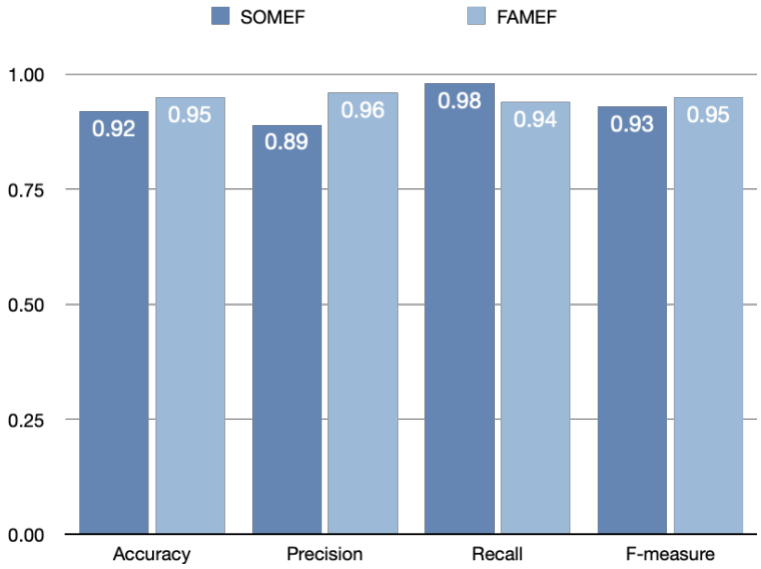

Fig 3.4.17: Comparison of FAMEF and SOMEF for citation

3.5. Installation Classifier

- The installation classifier predicts whether a sentence is about Installation instruction or not
- An installation excerpt usually contains installation steps and command line codes.
- Each sentence can be categorized into two classes: 1 means that it is an Installation sentence, 0 means it is not.
- We split our extended corpus into training and testing sets, the percentages are 75% and 25% respectively.

3.5.1. Retrain and improve top 4 models from SOMEF

We found the top 4 models in SOMEF and retrained them on our extended corpus. Then we tried to improve them using multiple approaches. We will discuss each of the models in detail.

1) TFIDF Vectorizer followed by Stochastic Gradient Descent

We built the scikit-learn pipeline with TFIDF Vectorizer and Stochastic Gradient Descent classifier using the log as loss function and L2 penalty, 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 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:

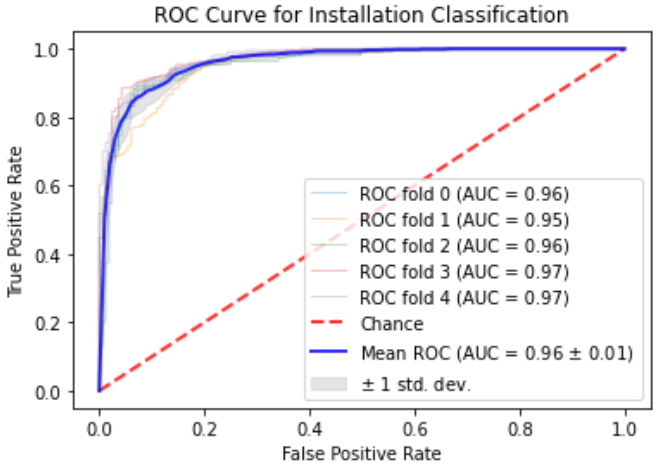

Fig 3.5.1

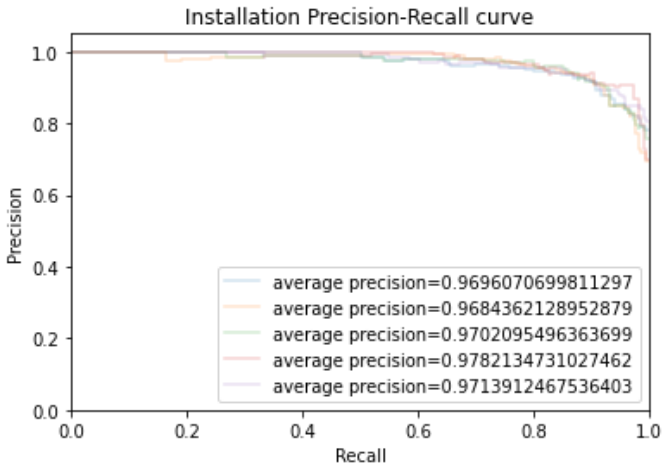

Fig 3.5.2

2) TFIDF Vectorizer followed by Logistic Regression

We built the scikit-learn pipeline with TFIDF Vectorizer and Logistic Regression classifier with L2 penalty and passed our training data to the pipeline and tested 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:

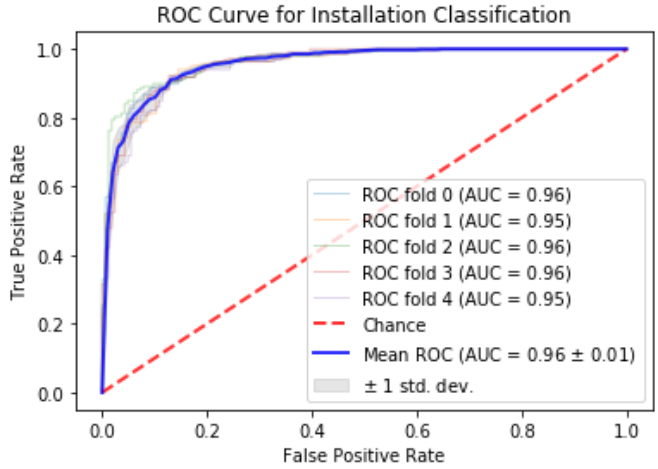

Fig 3.5.3

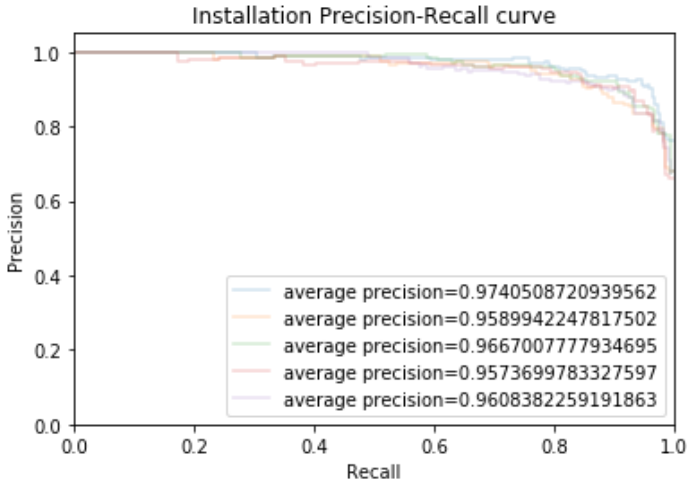

Fig 3.5.4

3) Count Vectorizer followed by Logistic Regression

We built the scikit-learn pipeline with Count Vectorizer and Logistic Regression classifier with L2 penalty, and passed our training data to the pipeline, and tested 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:

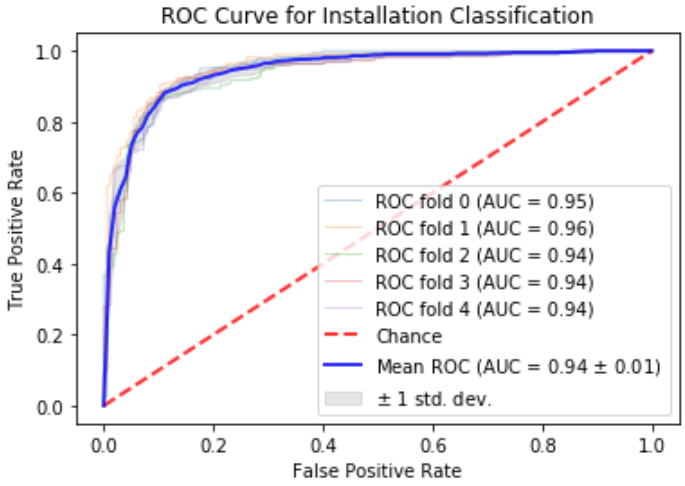

Fig 3.5.5

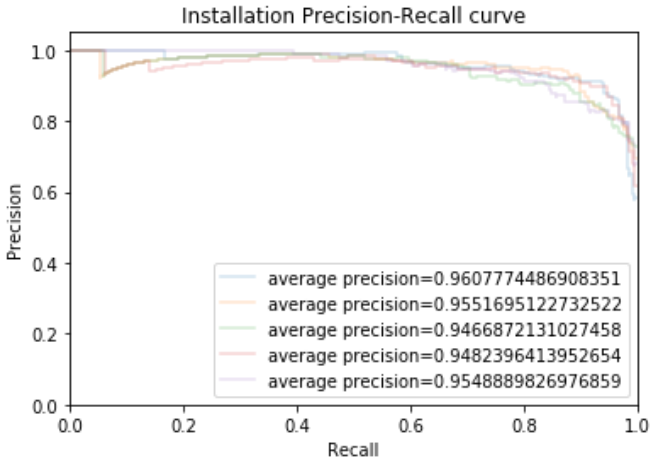

Fig 3.5.6

4) Count Vectorizer followed by Naïve Bayes

We built the scikit-learn pipeline with Count Vectorizer and Naïve Bayes classifier with L2 penalty 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:

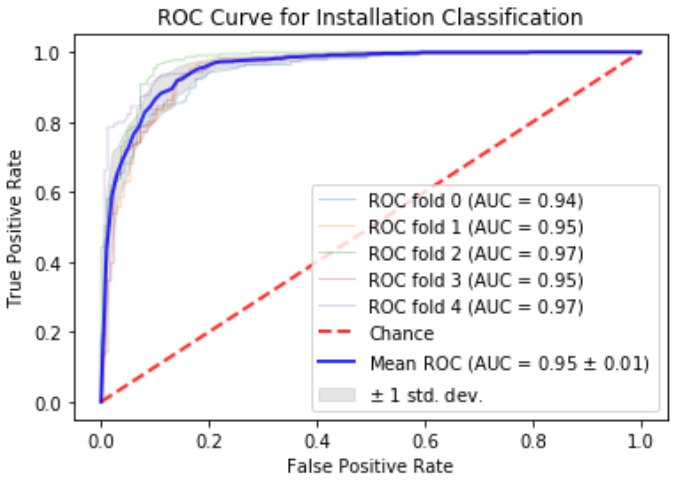

Fig 3.5.7

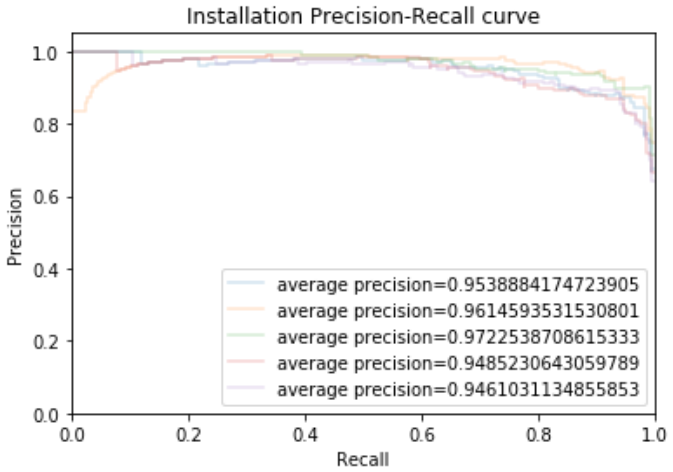

Fig 3.5.8

3.5.2. Experiment with other classification models

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

1) Count Vectorizer followed by 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:

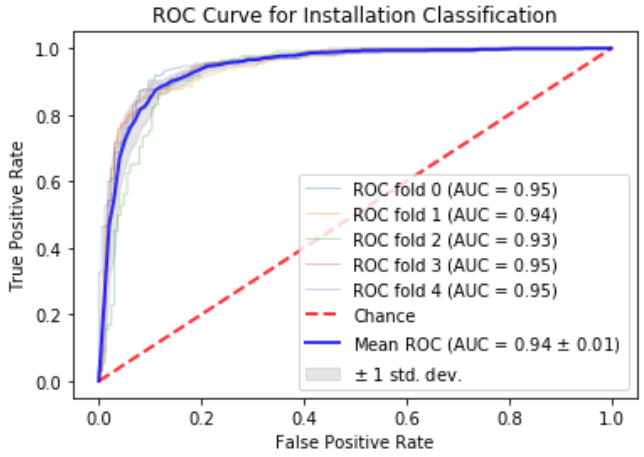

Fig 3.5.9

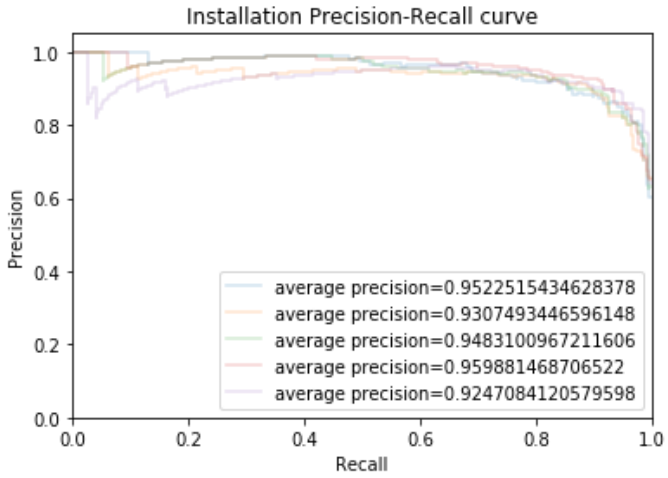

Fig 3.5.10

2) TFIDF Vectorizer followed by 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 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:

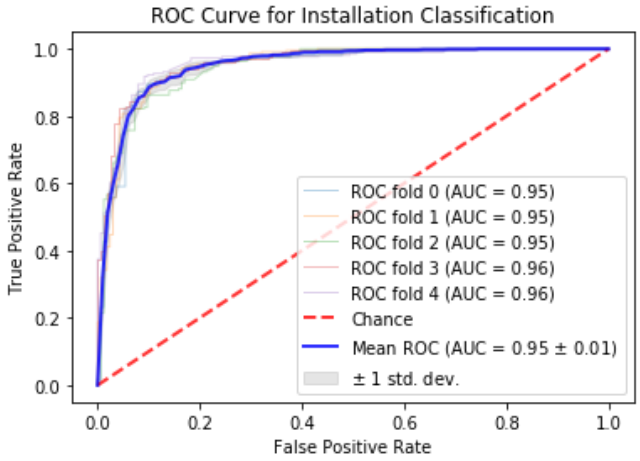

Fig 3.5.11

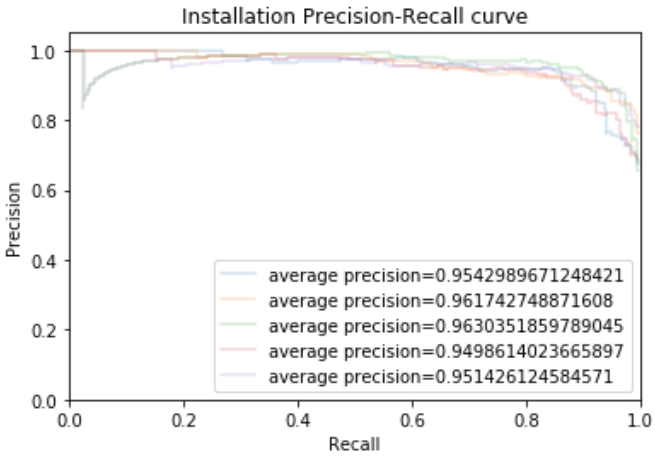

Fig 3.5.12

3) Count Vectorizer followed by SVM Classifier

We built the scikit-learn pipeline with Count Vectorizer and SVM Classifier with L2 penalty and passed our training data to the pipeline and tested 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:

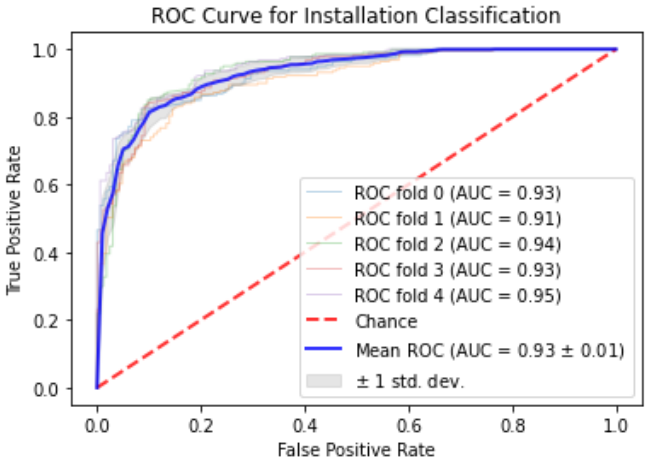

Fig 3.5.13

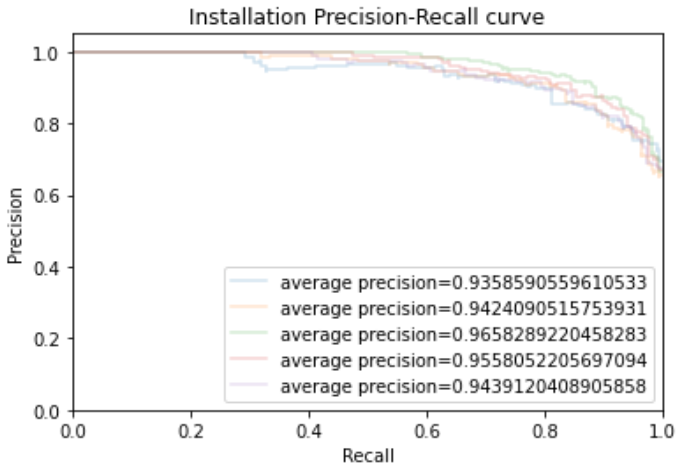

Fig 3.5.14

4) TFIDF Vectorizer followed by SVM Classifier

We built the scikit-learn pipeline with TFIDF Vectorizer and Logistic Regression classifier with L2 penalty and passed our training data to the pipeline and tested 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:

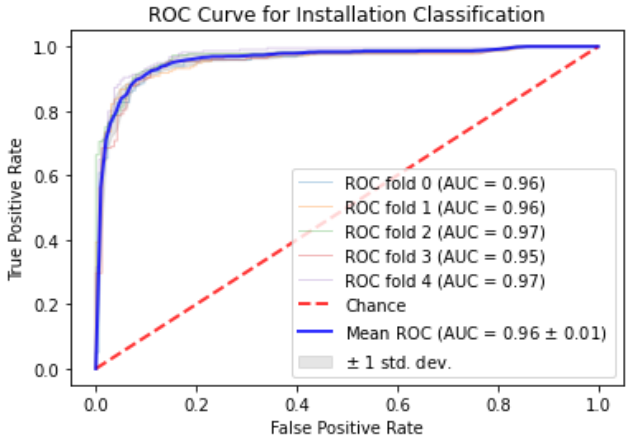

Fig 3.5.15

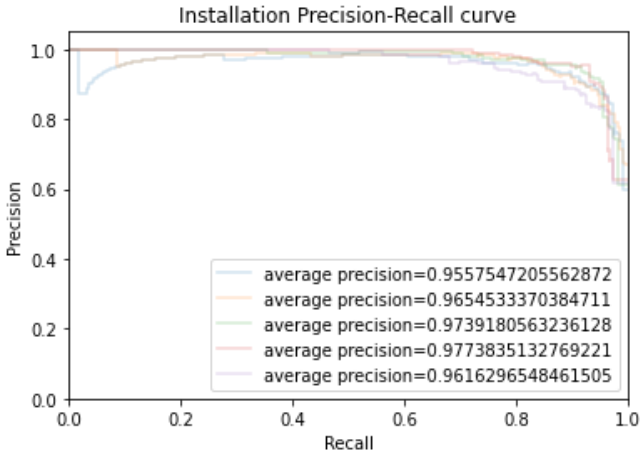

Fig 3.5.16

3.5.3. The best model for installation 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 chose is TFIDF Vectorizer and Stochastic Gradient Descent using the log as loss function and L2 penalty. The ranking of all models is shown in the following table 3.5.1.

Sklearn Primitive-Installation	**Accuracy**	**Precision**	**Recall**	**F-measure**
Sklearnpipeline(TFIDFVectorizer, StochasticGradientDescent)	0.92	0.91	0.92	0.92
Sklearnpipeline(TFIDFVectorizer, SVMClassifier)	0.91	0.91	0.91	0.91
Sklearnpipeline(TFIDFVectorizer, LogisticRegression)	0.9	0.89	0.89	0.89

Sklearnpipeline(CountVectorizer, LogisticRegression)	0.89	0.89	0.89	0.89
Sklearnpipeline(CountVectorizer, NaiveBayes)	0.9	0.91	0.88	0.89
Sklearnpipeline(TFIDFVectorizer, MLPClassifier)	0.9	0.9	0.89	0.89
Sklearnpipeline(CountVectorizer, MLPClassifier)	0.89	0.89	0.89	0.89
Sklearnpipeline(CountVectorizer, SVMClassifier)	0.86	0.85	0.86	0.86

Table 3.5.1 Rankings of all installation classifiers

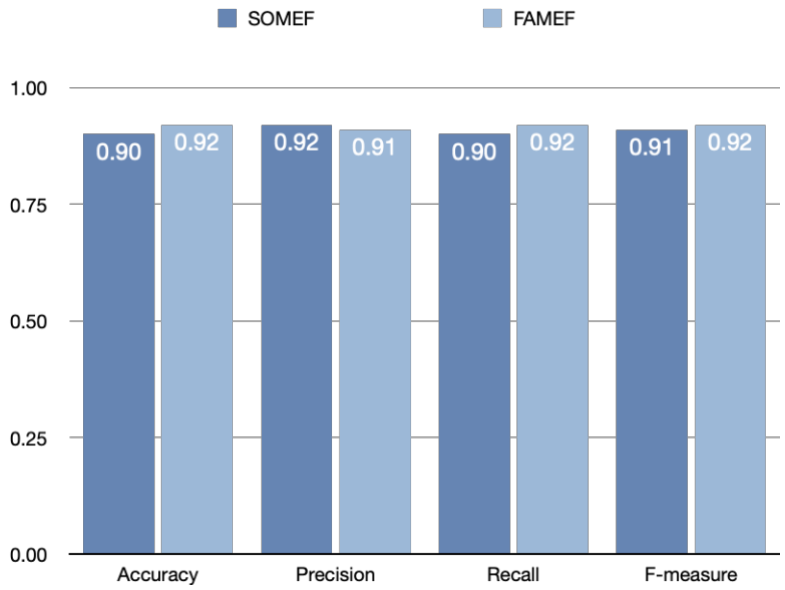

Fig 3.5.17 Comparison of FAMEF and SOMEF for installation

3.6. Functionality Classifier

3.6.1. Latent Dirichlet 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.

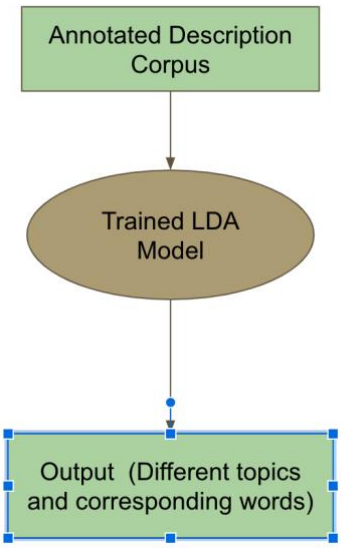

Figure 3.6.1: LDA Workflow

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.

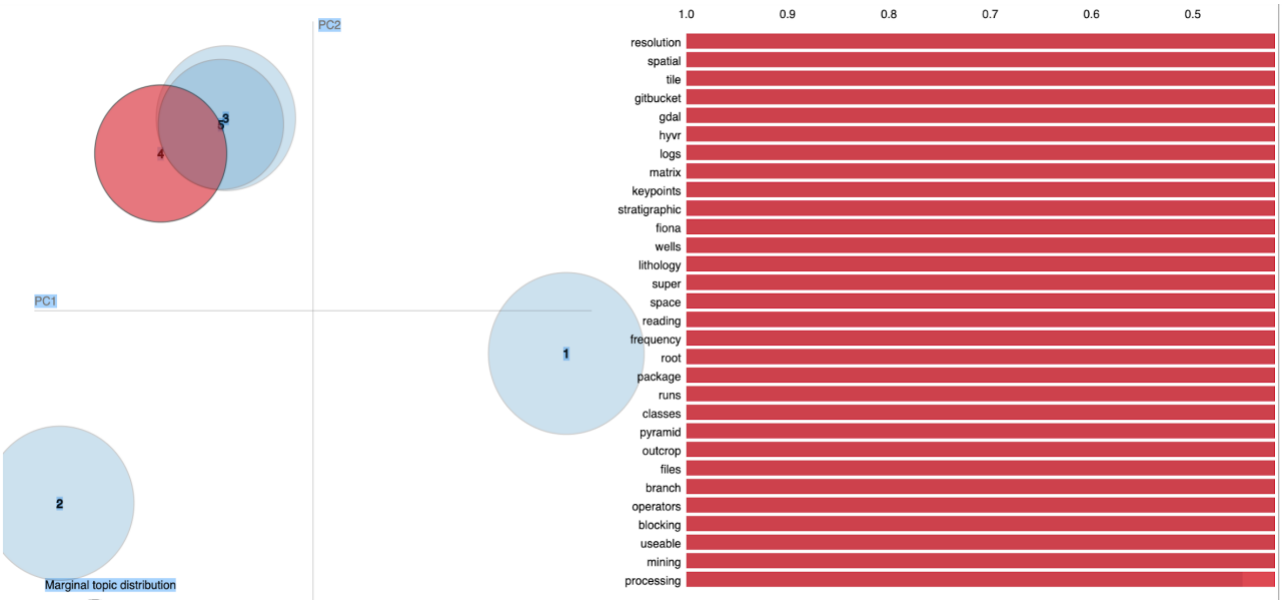

Figure 3.6.2: LDA results visualization

We got a distribution of words in our corpus and we broadly identified five different topics/clusters of which three of them are clustered close to each other. The three clusters/topics (3,4,5) in the figure had words like fastai, neural,models, pytorch, etc. We decided to combine these three topics and reached a consensus on labelling repositories having any of these keywords as data analysis. For Topic 1 we saw mentions

about words like html, charts, visualization, etc., we decided we would label repositories containing these words as Data Visualization. For the remaining topic, which is topic 2, we observed it had words like transformation, processing, preparation, etc. We decided on labelling repositories having these keywords Data Preparation as well as repositories that did not have any matching keywords. LDA helped us in resolving ambiguities with respect to identifying the functionality of repositories which was vital for us in going to the next step which is building a multi-class classifier to classify functionality.

3.6.2. Functionality Classifier

Since functionality classifier is multi-class, we needed to do further processing.

1) Data Preprocessing

   - The file extented_description.csv (Appendix [2]) gives an overview of description for each repository, we made use of the same in order to identify relevant keywords to label the functionality of repositories. As mentioned in Section 3.1.6, we divided labels into 3 classes including data preparation, data analysis, and data visualization. We stored the results in a file called function.csv (Appendix [6]). This formed the corpora for functionality classifier.

   - Table 3.6.1 shows the distribution of categories.

Category	Count
Data Analysis	117
Data Visualization	139
Data Preparation	75

Table 3.6.1: The Distribution of Functionality Corpus

2) CountVectorizer and Logistic Regression

   We passed our prepared data to the scikit-learn pipeline, with CountVectorizer and Logistic Regression. Since our categories are multi-class, we cannot apply the same multi_class parameter of LogisticRegression function to other classifiers, so we shifted the multi_class to “multinomial”. Then we tried adifferent 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 picked 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 dealing with multi-class classifier. The results of the model shown in TABLE 3.6.2.

3) TfidfVectorizer and LogisticRegression

   As discussed above, we made the pipeline in which TfidfVectorizer(default) is followed by LogisticRegression, and, after passing data, we tuned 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, we discerned that TfidfVectorizer is more stable than CountVectorizer which we ran 10 times. Therefore, we decided to use TfidfVectorizer to vectorize our corpora(sentence-level).

4) TfidfVectorizer and NaiveBayes

   By delving into the previous research, we found out that MultinomialNB, can be used for multi-class classification, so we used TfidfVectorizer(default) and MultinomialNB to train our model. The results of the model are shown in Table 3.6.2.
   Even though the scores have not improved a lot, it was getting more stable than before.

5) Word-Level TfidfVectorizer and NaiveBayes

   For the TfidfVectorizer, there is a parameter called analyzer which can be changed to “word”, so we tried to change it to enhance the accuracy. However, the accuracy drops to 0.34 from 0.53, hence we abandoned word-level TfidfVectorizer. Therefore, we did not make pipelines with word-level TfidfVectorizer for the remaining approaches.

6) TfidfVectorizer and RandomForestClassifier

   Another classifier that we tried is RandomForestClassifier, and we combined it with TfidfVectorizer(default). The results of the model are 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.

7) TfidfVectorizer and Perceptron

   Based on the previous research, another classifier that was employed was Perceptron, so we decided to create a pipeline with TfidfVectorizer(default)

with Perceptron.The results of the model are shown in Table 3.6.2. We got the best score for this model with accuracy equal to 75%.

8) Removing Stop words

Since the model of TfidfVectorizer and Perceptron is the best, we decided to use new excerpts, which removed stop words by using the nltk package, and it was applied to the model. The results of the model are shown in Table 3.6.2. However, after removing stop words, the results became unstable and lower than normal.

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

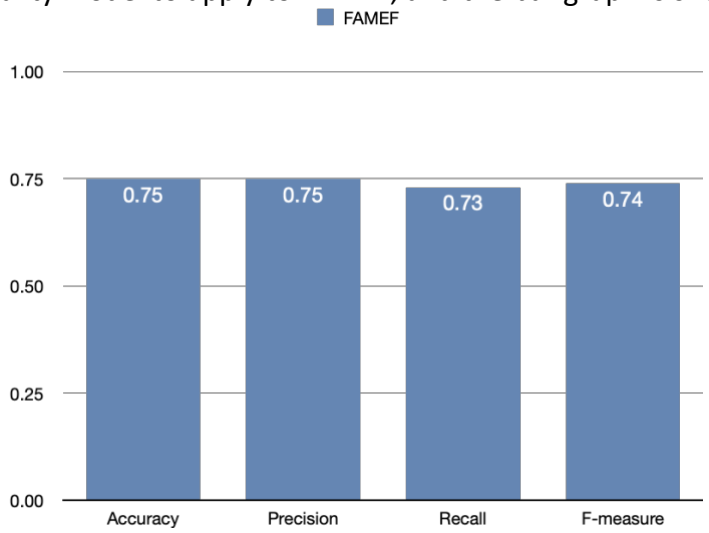

Figure 3.6.3: The Bar Graph of Functionality Model

sklearn Primitive - Functionality	**Accuracy**	**Precision**	**Recall**	**F-measure**
CountVectorizer + LogisticRegression	0.66	0.68	0.61	0.64
TfidfVectorizer + LogisticRegression	0.65	0.77	0.57	0.66
TfidfVectorizer + NaiveBayes	0.67	0.78	0.56	0.66
TfidfVectorizer (Word Level) + NaiveBayes	0.53	-	-	-
TfidfVectorizer+RandomForestClassifier	0.58	0.61	0.50	0.55
TfidfVectorizer + Perceptron	0.75	0.75	0.73	0.74
TfidfVectorizer + Perceptron (After Remove Stop Words)	0.70	0.70	0.69	0.70

Table 3.6.2: The Results of Functionality Classifier

3.7. Result JSON File

The results of FAMEF is a JSON file which is named by users on the command line. We will be showing a sample JSON file which is extracted from a repository called Prep Buddy[2].

3.7.1. The Results of "description"

"description" has "excerpt", "confidence", and "technique" parts to explain the description of the prep-buddy repository. "excerpt" is the prediction after passing the README file to the description classifier, and "confidence" shows the confidence of the excerpt, which may change while applying different techniques. Meanwhile, "technique" is the technique that we use to extract "excerpt", for example, the "technique" below is "metadata" which means it comes from the metadata of web crawling. If there is no prediction, it will export [] to that category. The real JSON result is shown in Fig 3.7.1.


```
"description": [
 {
  "excerpt": "A Scala / Java / Python library for cleansing, transforming
  "confidence": [
   1.0
  ],
  "technique": "metadata"
 }
],
```


Fig 3.7.1: "description" Result in JSON File

3.7.2. The Results of "citation"

"citation" has "excerpt", "confidence", and "technique" parts to explain the citation of the prep-buddy repository. "excerpt" is the prediction after passing the README file to the citation classifier, and "confidence" shows the confidence of the excerpt, which may change while applying different techniques. Meanwhile, "technique" is the technique that we use to extract "excerpt", but if there is no prediction, it will export [] to that category. In prep-buddy, there is no citation information, so it's just output "citation": [].

3.7.3. The Results of "installation"

"installation" has "excerpt", "confidence", and "technique" parts to explain the installation of the prep-buddy repository as before. The meaning of the contents is similar. However, in our example, we have 2 instances of installation, and the one

“technique” of them is “classifier” which means the result comes from our model. The real JSON result is shown on Fig 3.7.2.


```
"installation": [
 {
  "excerpt": "A Scala / Java / Python library for cleaning, transforming
  "confidence": [
   1
  ],
  "technique": "wordnet"
 },
 {
  "excerpt": "If you don't have pip. Intsall pip. \npip install prep-budd
  "confidence": [
   0.8663125468092844,
   0.9897244792800712
  ],
  "technique": "classifier"
 }
],
```


Fig 3.7.2: “installation” Result in JSON File

3.7.4. The Results of “invocation”

“invocation” has “excerpt”, “confidence”, and “technique” parts to explain the invocation of the prep-buddy repository. “excerpt” is the prediction after passing the README file to the invocation classifier, and “confidence” shows the confidence of the excerpt, which may change while applying different techniques. Meanwhile, “technique” is the technique that we use to extract “excerpt”, but if there is no prediction, it will return an empty array[] to that category. In prep-buddy, there is no invocation information, so it just outputs “invocation”: [].

3.7.5. The Results of listing Metadata

We provide other metadatas of the extracted repository as well, for example, “documentation”, “download”, “requirement”, “codeRepository”, “owner”, “ownerType”, “DataCreated”, “dateModified”, “license”, “name”, “fullName”, “issueTracker”, “forks_url”, “stargazers_count”, “forks_count”, “downloadUrl”, “topics”, “languages”, “readme_url”, and “releases”. The information comes from web crawling without processing, and the “releases” example and the “forks_count” example are shown in Fig 3.7.3 and Fig 3.7.4

```
"releases": {
 "excerpt": [
  {
   "tag_name": "v-0.5.1",
   "name": "Few Bug Fixes ",
   "author_name": "blpabhishek",
   "authorType": "User",
   "body": "Removed the take sample method from the AbstractRDD constructor.
   "tarball_url": "https://api.github.com/repos/data-commons/prep-buddy/tarb
   "zipball_url": "https://api.github.com/repos/data-commons/prep-buddy/zipb
   "html_url": "https://github.com/data-commons/prep-buddy/releases/tag/v-0.
   "url": "https://api.github.com/repos/data-commons/prep-buddy/releases/521
   "dateCreated": "2017-01-20T06:28:02Z",
   "datePublished": "2017-01-20T06:36:36Z"
  },
  {
   "tag_name": "v0.5.0",
   "name": "Beta release",
   "author_name": "blpabhishek",
   "authorType": "User",
   "body": "This is the first release which is compatible  with Apache-spark
   "tarball_url": "https://api.github.com/repos/data-commons/prep-buddy/tarb
   "zipball_url": "https://api.github.com/repos/data-commons/prep-buddy/zipb
   "html_url": "https://github.com/data-commons/prep-buddy/releases/tag/v0.5
   "url": "https://api.github.com/repos/data-commons/prep-buddy/releases/382
   "dateCreated": "2016-08-08T07:50:37Z",
   "datePublished": "2016-08-08T09:18:21Z"
  }
 ],
 "confidence": [
  1.0
 ],
 "technique": "metadata"
},
```


Fig 3.7.3: “releases” Result in JSON File


```
"forks_count": {
 "excerpt": {
  "count": 8,
  "date": "Thu, 19 Nov 2020 20:11:49 GMT"
 },
 "confidence": [
  1.0
 ],
 "technique": "metadata"
},
```


Fig 3.7.4: “forks_count” Result in JSON File

3.7.6. The Results of “functionality”

“functionality” has “excerpt”, “confidence”, and “technique” parts to explain the functionality of the prep-buddy repository as before. The meaning of the contents is similar, and for our prediction of functionality, it will have 3 types, which are “Data Analysis”, “Data Visualization” and “Data Preparation”. “Technique” is “Perceptron” because we use the Perceptron model to predict the functionality. The real JSON result is shown on Fig 3.7.5.

```
"functionality": {
 "excerpt": "Data Preparation",
 "confidence": [
  1.0
 ],
 "technique": "Perceptron"
}
}
```


Fig 3.7.5: “functionality” Result in JSON File

4. Reproducibility

4.1. Datasets

Our training data and test data for each classifier are picked from extented_description.csv (Appendix [2]), extended_invocation.csv [3], extended_citation.csv (Appendix [4]), extended_installation.csv [5], and function.csv (Appendix [6]). The developers can expand the CSV files to generate more instances to train and test. The whole folder is available on Figshare (Appendix [12]) as well.

4.2. Repository

For now, we upload our framework-based software to the Github repository (Appendix [1]) and Zenodo (Appendix [13]) for researchers and developers who can freely use or build on FAMEF.

4.3. Modeling Notebooks

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

4.4. Useful Scripts

Our scripts [8] are able to invoke classifiers’ models (Appendix [10]), and it will crawl the README file and basic information of the target repository. By passing information to the dictionary data structure, the scripts (Appendix [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.

4.5. Installation Instructions

To mount FAMEF, you can clone the repository (Appendix [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.
```

4.6. Virtual Environment Setup

For installing FAMEF in virtual 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.

4.7. Docker Container

We also provide the Docker container to directly install FAMEF by using Docker file (Appendix [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 (Appendix [9]) to explore further.

5. 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 in ensuring we were on the right path.
- In particular we had two such meetings, apart from our final project presentation.

5.1. Meeting 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 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.

5.2. Meeting 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.

5.3. 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 through 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.

6. 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 114 readme files and tried different kinds of classification models like logistic regression, Stochastic Gradient Descent, random forest, and multi-layer perceptron. We also fine-tuned each model by performing hyper-parameter tuning, feature selection, and cross-validation. From this project, we learned that more complex models do not necessarily mean better results. We chose simpler models like logistic regression and Stochastic Gradient Descent over deep learning or ensemble methods for all the tasks.

Classifier	Accuracy	Precision	Recall	F-measure
Description	0.83 -> 0.87	0.85 -> 0.87	0.79 -> 0.92	0.82 -> 0.90
Invocation	0.88 -> 0.89	0.86 -> 0.89	0.94 -> 0.91	0.90 -> 0.90
Installation	0.90 -> 0.92	0.92 -> 0.91	0.90 -> 0.92	0.91 -> 0.92

Citatsion	0.92 -> 0.95	0.89 -> 0.96	0.98 -> 0.94	0.93 -> 0.95
Functionality	0.75	0.75	0.73	0.74

Table 6.1 Metrics for all classifiers compared with baseline

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

Due to time and resource limitations, we are not able to fully expand FAMEF. There is still much potential for future works. Firstly, we could try NLP techniques like word2vec to further improve the classifiers. Besides, we could create a graphic user interface so our users can use FAMEF by clicking a few buttons. The UI can also allow us to show our classification results in a more pleasant way. Last but not least, we can extend our Framework to open-source repositories from other websites such as Gitlab, BitBucket, and SourceForge. In this way, we will be able to serve users from more platforms and make more contributions to the open-source community.

Reference

[1] A. Mao, D. Garijo and S. Fakhraei, "SoMEF: A Framework for Capturing Scientific Software Metadata from its Documentation," 2019 IEEE International Conference on Big Data (Big Data), Los Angeles, CA, USA, 2019, pp. 3032-3037, doi: 10.1109/BigData47090.2019.9006447.

[2] Prep Buddy, (2017), Github repository, https://github.com/data-commons/prep-buddy

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

[11] README file: https://github.com/liling10822/FAMEF/blob/main/README.md

[12] Figshare DOI: https://doi.org/10.6084/m9.figshare.13218755.v2

[13] Zenodo DOI: https://doi.org/10.5281/zenodo.4278188

[14] Documentation: https://liling10822.github.io/FAMEF/

FAMEF (Functionality and Metadata Extraction Framework)

DOI 10.5281/zenodo.4278188 doi 10.6084/13218755

FAMEF (Functionality and Metadata Extraction Framework) is an extension version based on somef, which is a command line interface for automatically extracting relevant information from readme files.

On the basis of somef, the accuracies of FAMEF in the description classifiers, invocation classifiers, installation classifiers, and citation classifiers is increased, and FAMEF contains the function classifier to analyze the functionality of readme files.

Contributors: Ling Li, Pratheek Athreya, Sharad Narayan Sharma, Yi Xie, Yidan Zhang

Documentation

To see the documentation of FAMEF

Installion

Mounting FAMEF in virtual environment

Install virtualenv package on your computer by running the command line below:

`pip install virtualenv`

To run FAMEF, please follow the next steps:

Clone this GitHub repository

```
git clone https://github.com/liling10822/FAMEF.git
```

Run `cd FAMEF` to go to the folder that you cloned

Create a blank virtual environment called env by running the command line below:

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

Activate the env environment by running the command line below:

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

Install FAMEF in the virtual environment(note that you should be in the folder that you just cloned)

```
pip install -e .
```

Test FAMEF installation

```
somef --help
```

If everything goes fine, you should see:

```
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.
```

Usage

Configure

Before running FAMEF, you must configure it appropriately. Run

```
somef configure
```

And you will be asked to provide the following:

- A GitHub authentication token [**optional, leave blank if not used**], which FAMEF uses to retrieve metadata from GitHub. If you don't include an authentication token, you can still use FAMEF. However, you may be limited to a series of requests per hour. For more information, see https://help.github.com/en/github/authenticating-to-github/creating-a-personal-access-token-for-the-command-line
- The path to the trained classifiers (pickle files). If you have your own classifiers, you can provide them here. Otherwise, you can leave it blank

Run FAMEF

```
$ somef describe --help
  SOMEF Command Line Interface
Usage: somef describe [OPTIONS]

  Running the Command Line Interface

Options:
  -t, --threshold FLOAT           Threshold to classify the text  [required]
  Input: [mutually_exclusive, required]
    -r, --repo_url URL            Github Repository URL
    -d, --doc_src PATH            Path to the README file source
    -i, --in_file PATH            A file of newline separated links to GitHub
                                  repositories

  Output: [required_any]
    -o, --output PATH             Path to the output file. If supplied, the
                                  output will be in JSON

    -g, --graph_out PATH          Path to the output Knowledge Graph file. If
                                  supplied, the output will be a Knowledge
                                  Graph, in the format given in the --format
                                  option

  -f, --graph_format [turtle|json-ld]
                                  If the --graph_out option is given, this is
                                  the format that the graph will be stored in

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

Usage example:

The following command extracts all metadata available from https://github.com/dgarijo/Widoco/.

```
somef describe -r https://github.com/dgarijo/Widoco/ -o test.json -t 0.8
```

The results will generate the test.json in the FAMEF folder.

To get distribution of topics present in the corpus and classify which topic a repo belongs to run the following:

```
python lda.py name of output file generated in the previous step (test.json) name of output file
```

Run on docker

FAMEF also can be run on the Docker. To set up and run it, please follow the below steps:

1. After you git clone this respository, you need to install Docker if you don't have one. Play with Docker (https://labs.play-with-docker.com/) is a good choice for someone who does not have a Linux/MacOS system.
2. Before you build the project, please run `cd FAMEF` in your terminal.
3. Build the project:

   ```
   docker build --tag [tagName] .
   ```

4. Run Docker:

```
docker run -p 5006:5006 -it [tagName]
```

In our Dockerfile, we help you excute `git clone` the repository and install it. You could add other commands to the Dockfile to explore furthur.

Reference

```
@INPROCEEDINGS{9006447,
author={A. {Mao} and D. {Garijo} and S. {Fakhraei}},
booktitle={2019 IEEE International Conference on Big Data (Big Data)},
title={SoMEF: A Framework for Capturing Scientific Software Metadata from its Documentation},
year={2019},
url={http://dgarijo.com/papers/SoMEF.pdf},
pages={3032-3037}
}
```