FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Classification of low-level tasks to high-level tasks using JIRA data

Shivathanu Gopirajan Chitra

Mestrado em Engenharia de Software

Supervisor: Prof. João Pedro Mendes Moreira

Co-Supervisor: Prof. Filipe Figueiredo Correia

July 28, 2021

Classification of low-level tasks to high-level tasks using JIRA data

Shivathanu Gopirajan Chitra

Mestrado em Engenharia de Software

Approved in oral examination by the committee:

Chair: Prof. Nuno Honório Rodrigues Flores (PhD), FEUP
External Examiner: Prof. Jácome Miguel Costa Cunha (PhD), Universidade do Minho
Supervisor: Prof. João Pedro Mendes Moreira (PhD), FEUP
Co-Supervisor: Prof. Filipe Figueiredo Correia (PhD), FEUP

July 28, 2021

Abstract

In large enterprises, there is often a large number of tasks being created within projects. Hence to manage a project, it becomes necessary for project managers to efficiently organise the tasks based on the level of detail. It is possible to split the tasks and create an association between them. Project management tools allow the user to perform this functionality, but this association needs to be done manually for creation of each task. It would be efficient to have a tool that can perform the association based on knowledge of already associated tasks. In the field of machine learning, an approach of inductive learning is followed, where the rules are discovered by models and an output is produced by observing examples termed as training datasets.

The objective of this dissertation is to perform experiments using machine learning models on classifying text documents using the Jira dataset. An optimal performing model decided from the experimentation process will be used to build a tool for task association. The dataset used for this project deals with high imbalance between classes, hence we have selected classification models that are more suitable for dealing with class imbalance. In this dissertation project, we implemented supervised machine learning models such as Support Vector Machine, Naive Bayes, Random forest, SGDClassifier (Stochastic Gradient Descent), XGBoost.

As part of the experimentation process, before the implementation of the model we have explored optimizing the settings for TF-IDF measure used in the text transformation step. The models are evaluated using 6-fold cross validation in order to compare the performance of different models. Experimental results are obtained from the models trained on a set of 1924 JIRA records.

In order to improve the results obtained, we explored different hyperparameters by manually changing a few parameters that were input to different models during the training phase. For better optimization of the models, we use an automatic hyperparameter optimization software framework named Optuna that implements state-of-the-art algorithms for removing unpromising trials and choosing the right hyperparameters.

Evaluation metrics such as accuracy, f1-score, precision and recall are obtained for each model. Based on the different experiments performed and evaluation results compared, this dissertation concludes that the type of classification model named as "LinearSVC" had the highest performance in terms of f1-score, accuracy and also in the model execution time.

Keywords: Supervised machine learning, JIRA, Classification, Stochastic Gradient Descent, Support Vector Machines, Naive Bayes, Random forest, XGBoost.

Resumo

Nas grandes empresas, há muitas vezes um grande número de tarefas que estão a ser criadas no âmbito dos projectos. Assim, para gerir um projecto, torna-se necessário que os gestores de projecto organizem eficazmente as tarefas com base no nível de pormenor. É possível dividir as tarefas e criar uma associação entre elas. As ferramentas de gerenciamento de projetos permitem ao usuário realizar esta funcionalidade, mas esta associação precisa ser feita manualmente para a criação de cada tarefa. Seria eficiente ter uma ferramenta que possa executar a associação com base no conhecimento de tarefas já associadas. No campo da aprendizagem de máquinas, segue-se uma abordagem da aprendizagem indutiva, onde as regras são descobertas por modelos e uma saída é produzida pela observação de exemplos denominados como conjuntos de dados de formação.

O objectivo desta dissertação é realizar experiências utilizando modelos de aprendizagem por máquina na classificação de documentos de texto do conjunto de dados Jira. Um modelo de desempenho ideal decidido a partir do processo de experimentação será usado para construir uma ferramenta para a associação de Tarefas. O conjunto de dados utilizado para este projecto lida com um elevado desequilíbrio entre classes, pelo que seleccionámos modelos de classificação que são mais adequados para lidar com o desequilíbrio de classes. Neste projeto de dissertação, implementamos modelos supervisionados de aprendizagem de máquinas, tais como suporte a máquina vetorial, ingénuo Bayes, Floresta aleatória, SGDClassifier (descendente de gradiente estocástico), XGBoost.

Como parte do processo de experimentação, antes da implementação do modelo, explorámos a optimização das definições para a medida TF-IDF utilizada na etapa de transformação de texto. Os modelos são avaliados utilizando a validação cruzada de 6 vezes, a fim de comparar o desempenho de diferentes modelos. Os resultados experimentais são obtidos a partir dos modelos formados num conjunto de registos JIRA de 1924.

A fim de melhorar os resultados obtidos, explorámos diferentes hiperparâmetros alterando manualmente alguns parâmetros que foram introduzidos em diferentes modelos durante a fase de treino. Para melhor optimização dos modelos, utilizamos uma estrutura de software de optimização automática de hiperparâmetros que implementa algoritmos de última geração para remover ensaios não prometedores e escolher os hiperparâmetros certos.

Métricas de avaliação tais como precisão, f1-score, precisão e recall são obtidas para cada modelo. Com base nas diferentes experiências realizadas e resultados de avaliação comparados, esta dissertação conclui que o SGDClassifier teve o melhor desempenho em termos de f1-score, precisão e também no tempo de execução do modelo.

Keywords: Aprendizagem supervisionada de máquinas, Classificação de tarefas Jira, Descida Estocástica Gradiente, Máquinas Vectoriais de Apoio, Naive Bayes, Floresta Aleatória, XGBoost.

Acknowledgements

I would like to express my sincere thanks to Professor João Pedro Mendes Moreira and Professor Filipe Figueiredo Correia for guiding me in the right direction during the thesis. I would say performing this work would have been impossible for me if it was not under the supervision of both the professors. I thank both the professors for the time they spent from their busy schedules to clarify any doubts that I have. Every week seemed to move fast with interesting challenges to solve that the professors have provided me. And now when I look back in time where I started I feel proud about the learnings I gathered along the way.

I am grateful to Ricardo Graça from Fraunhofer, for providing me with structured data which removed many roadblocks that could have possibly complicated this work. During the initial meeting I had with Ricardo, he clearly explained to me the details required for this work.

Thanks to god for protecting me, especially during these pandemic times. I am grateful for the good health, and his answered prayers that allowed me to perform this work with my fullest capabilities.

I would like to take this opportunity to thank my friend Arjun Sah for suggesting this masters programme and motivating me at all times. Special thanks to Professor Ana Paiva and Professor Nuno Flores for giving me a chance to study at FEUP. Thanks to my employer Mudey and colleagues for the industrial learnings.

Most importantly, thanks to my family members who helped me fulfil my goal to take this course. Without their support it would be impossible for me to reach this stage. Special mention to my father Gopirajan, my mother Chitra and my uncle Suriyakumar.

Shivathanu Gopirajan Chitra

"You can have data without information,
but you cannot have information without data"

Daniel Keys Moran

Contents

List of Figures

List of Tables

Abbreviations

CRISP-DM	CRoss Industry Standard Process for Data Mining
ML	Machine learning
NLP	Natural Language Processing
SVM	Support Vector Machines
SGD	Stochastic Gradient Descent
NB	Naive Bayes
RF	Random forest
CV	Cross Validation
TF-IDF	Term frequency-Inverse Document frequency
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative
API	Application Programming Interface

Chapter 1

Introduction

Defining the project in terms of the tasks required to achieve a goal is critical for the success of a project. Project managers usually spend the majority of their time in the requirements phase, defining the topic and decomposing that topic to the lowest granular elements possible commonly referred to as tasks. The user might introduce a new task into the system and wants to associate it with an already existing topic.

Under the context of extreme programming [10] these tasks are referred to as "User stories". User stories were created by customers in an easy to understand text format to represent the tasks that need to be done by the system. The notion of epic later appeared in a book by Mike Cohn [9]. These concepts evolved and have been adapted in many project management tools. Structuring the project by relating user stories to epics helps managers to plan and efficiently track the project status at various stages such as development, testing and delivery of the project.

We have discussed the conventional use case of a project definition. During the creation of tasks, for associating it to an epic users perform a manual search from the list of available epics. It would be productive if it was possible to have a mechanism that can recommend the most suitable epic to the task from the list of possible epics already created. To facilitate this solution, we propose to build a tool that implements machine learning models for classifying the tasks and associating them to an epic based on a matching text description. In the field of machine learning, this problem of associating the low-level tasks to a parent task using text can be solved through a supervised method named text classification.

1.1 Motivation

Project managers deal with numerous actions as part of daily project management activities. It would always be efficient to replace labour-intensive tasks with intelligent solutions [17] inside an organization. There is a possibility that the approaches used in project management can also be improved to help managers reduce the time taken to complete a set of activities. Although

there is a large set of project management tools currently being used inside the organization by both managers and developers to ease their work, there is a prospect for improvement. This leads the focus of our work towards features implemented in a tool named JIRA [21] regarding the association of task and epics. While associating a task to an epic in Jira, there does not exist a way for the user to know the suitable epic. Users have to manually search the epic to be associated from the list of already available epics. It can be a time-consuming and repetitive activity for the user to read each epic description to understand and associate it to a task. The potential to introduce a machine learning process to solve this problem is a huge motivation for our work. The complexity of solving this task association problem is based on the quality of JIRA data and the number of projects involved in the system. We propose a tool that can understand the word association from text description present in both task and epic and relate them together. Based on this relationship found, the tool will be able to suggest a suitable epic to a task. Hence this dissertation focuses mainly on the classification topics used in machine learning that enables us to build this tool.

1.2 Scope of this work

This project is developed in coordination with FEUP for project PROMESSA, to solve the existing problem faced by the project management team within the organization Fraunhofer. The main goal of this project is to use a machine learning model to predict a suitable epic for a given user story. The goal is subdivided as below, with key steps needed to be performed to achieve the end goal:

- Perform an initial understanding of the dataset and identify problems that might affect the model selection.
- Research the classification technique applicable for the text dataset and choose the best suitable models for this problem.
- Experiment with the dataset with chosen classification models to find an optimal solution.
- Develop a script with the optimal model achieved to be applied for a real-world project.

1.3 Research Questions

This dissertation is constructed in phases approaching the below research questions during the development and validation.

- **RQ1:** What is the best baseline model to start for classification problems according to literature. ?
- **RQ2:** How good are the different models used for classifying tasks. ?
- **RQ3:** What evaluation metric would be most suitable for distinguishing the optimal classifier. ?

1.4 Usage of CRISP-DM Methodology

Throughout this work, we have followed the CRISP-DM methodology considering its influence towards standardizing the process in data science projects [29]. We initiated our work with "business understanding, data understanding and data preparation" followed by "modeling, evaluation and deployment". During the process, we addressed the research questions mentioned in section 1.3. We used a tool named *"Jupyter Notebook"* for performing our analysis on the data and our development activities. State of the art review work on related fields [15] and exploratory data analysis performed in the data preparation phase provided potential direction to perform the right experiments and in turn enabled us to find answers for our first research question 1.3. We based our choice of selection on the implementation of text classifiers using a study on the literature review referenced in this section 3.2. We performed various experiments that are detailed in sections 3.2 and 3.3. Performing different experiments provided results that helped us answer our second and last research questions.

1.5 Document Structure

This document consists of 4 chapters. In Chapter 2, research is made on a few project management tools and the features are compared with Jira to understand how different tools implement the association between different tasks. A summary of the discussion to compare features that relate to associating tasks is provided at the end of this section. Based on this study we could answer one of the research questions raised in Section 1.2. A second section to Chapter 2 finds a reference work that was performed in a research paper [15]. The work done by the authors of this paper closely resembles the approaches taken in this dissertation work. Since the dataset and the problem statement is also based on Jira which is to classify the user stories based on descriptions and associated to a category, it is important to mention that initial work performed in this dissertation is driven and hugely motivated by the reference work [15].

In Chapter 3, the implementation details performed based on the CRISP-DM methodology to achieve the end goal for this work is described in sequential phases such as (a) data preparation that involves text preprocessing techniques used to reduce the words in a sentence to their root words that imply meaning for the model, then the (b) selection and application of classification models is explained with the source code associated with each model. In the next section, (c) Model evaluation is done by a cross-validation method where the data is split and the model is trained for each set to generalize the model. The strategy used for performing this cross-validation approach is detailed in this section. In the end, (d) how hyperparameter tuning is made to improve the performance of each model is explained with the relevant source code.

In the next Chapter 4, a table of results obtained for each model is provided. Evaluation metrics used to measure the results are summarized with a discussion at the end comparing the results. In the end, the optimal performing model is selected based on the comparison of using the different evaluation metrics.

The conclusion is the last Chapter 5 of this dissertation, where we discuss the limitations and research contributions are provided. Future work that this dissertation can lead, is given at the end of this document.

Chapter 2

State of the art review

This section provides a baseline for the methodology covering the state of the art research done on project management tool features, use of supervised machine learning in textual data and at the end of the section a reference work done in the classification of user stories using Jira data. Since this dissertation is based on the data from the Jira tool, the tool features specific to the focus of this dissertation are explained. A short research on two other tools that share similar features to Jira named Rally, Trello is summarised and a comparison done at the end to educate the reader about terms relevant to the tools. After a brief study of the tools, we will be looking into the machine learning details related to the classification of texts.

2.1 Study of project management tools

In recent times, there are many project management tools available in the software field, in this section three tools are chosen and their features on grouping tasks and how they provide the ability to map an epic to a task is studied. One research question along the way for discussion could be, *"Currently, how the association of tasks and epics is performed in agile project tools. Is it automatic or manually done?"*. This ability to decompose the single large task into multiple smaller tasks based on the scope is required for development teams to build that feature and keep agile releases cadence. For example, considering a scenario of a user planning an international business meeting to understand the task decomposition. This task can be decomposed into smaller tasks such as finding a suitable place, preparing invitations for participants and arranging the transportation of guests. Once the required tasks are defined, these can be associated with an epic labelled as “International business meeting”. So it is important to know the tools allow us to associate the tasks to epics, as the quote says,

“It is essential to have good tools,
but it is also essential that the tools should be used in the right way.”
- Wallace D. Wattles

2.1.1 Jira

As per the report from [1]Atlassian, more than 500 thousand agile projects were created. Teams functioning in agile practises are found to be more release oriented. Research done in that report mentions that teams that release often and early typically have 30 tasks in a 2 weeks sprint. So teams should find an average number of tasks to be present in a sprint. To do that, the task scope needs to be recognized and modified to fit in the sprint. This leads to possible questions such as *"Does the original task need to be an epic?"* or *"Should be further decomposed to smaller tasks?"*. Jira provides the ability to perform these modifications by the user along with the sprint. Jira provides two ways of doing this task association and decomposition, a simple way to do task association to an epic is to select an already created task from the Backlog and associate it to an epic from the list of available epics. And after, the task becomes a part of a larger scope. Conversely, the decomposition can be done by selecting an epic and creating tasks inside it. Another feature Jira introduced in 2018 named [2]Roadmap allows users to visualize how the planned product will be built over a while. This roadmap view also allows the user to perform the task association, decomposition in a simple but intuitive way.

2.1.2 Rally

Rally is an agile software project management tool more suited for enterprise needs. This tool differs from Jira in the agile process adoption. Teams cannot customize the workflow with the flavour of agile they might need such as Scrum, Kanban. From the many features this tool has to offer, we are more interested in the task association feature. Rally has a wide integration with external applications and one such application that can be compared with the Roadmap feature in Jira is *"Story Hierarchy App"*. This application allows users to manage relationships between tasks. Rally also offers features such as view story hierarchies, ability to reparent story inside the roadmap view. A noteworthy feature is the bulk association of user stories to parent stories using the *"Edit Multiple Work Items"* as mentioned in [3]Rally official docs webpage.

2.1.3 Trello

Trello is a collaboration tool that works based on the kanban method and uses board views to organize the workflow. This tool differs from Jira in a way that it can be used for managing any type of tasks related to software project management, for example: holiday trip, marketing plans, etc. whereas Jira targets users dealing with software projects. An analogy to explain Trello can be, thinking of a whiteboard filled with sticky notes with each note containing a task and its related artefacts such as photos, attachments. Users can also comment on the cards. One feature that Trello uses for associating parent and child tasks is provided by a plugin named *"Hello Epics*

[1]https://www.atlassian.com/blog/jira-software/break-decomposing-user-stories-jira
[2]https://techcrunch.com/2020/01/23/jira-software-gets-better-roadmaps/
[3]https://techdocs.broadcom.com/us/en/ca-enterprise-software/agile-development-and-management/rally-platform-ca-agile-central/rally/using-top/work-effeciently-rally/bulk-delete.html

Power-Up". This is an add-on feature that users need to install to perform the association between cards. Tasks are referred to as cards in Trello terms. This "Hello Epics Power-Up" plugin works by complementing the epics and stories workflow found in the Jira tool. This add-on feature uses dependency to describe a relationship between a larger picture card or parent and a more detailed lower level card or child card. An important solution this plugin provides is the ability to attach child cards to parent cards, and also allows tracking the status of the child cards. Using this feature it is possible to get a quick insight into the status of the task and how much work is left to be done so that the tasks can be prioritized.

2.1.4 Discussion

The study of the techniques used by project management tools for associating user stories and epics helps in answering one of the research question mentioned in Section 2.1. From the above tools section, it is understood that the association of user stories and epics is being done manually by users based on the feature relevance in terms of a parent-child relationship. Based on the research done on the features provided by these tools, it could be interpreted that Jira manages to handle the complexity involved in associating user stories at different hierarchies across multiple projects while providing a streamlined user experience.

A comparison table summarising the features that support the objective of this work can be found in the table below.

Features	Jira	Rally	Trello
Parent-Child relationship support	Default	Default	Add-on
Hierarchy of tasks	Multiple	Multiple	Multiple
Association of task to epic	Manual	Manual	Manual

Table 2.1: Comparison of project management tools

2.2 Machine learning approaches

Nowadays many business problems are solved using natural language processing tasks which are referred to as *"Text Classification"*. With advances in natural language processing and machine learning, classifying text data has become easier. Machine learning is a great resource for prediction, but it needs data for learning. It is said that around 80% of information in the world is not structured, of which textual data is the most common type. It becomes difficult to understand, process and analyze text data. Text classification solves this problem by automating processes analyzing data, making data-oriented decisions. Although it should be first understood that text classification is a supervised learning process. It can be done either manually or automatically. Manually classifying data into categories requires a human coordinator constantly interpreting the textual information and categorizing it. This can achieve good outputs but it consumes more time

and is highly inefficient. Automatically classifying involves machine learning, natural language processing to categorize the text data in a much more efficient way. There can be more approaches to this, but usually, text classification falls under three categories such as,

- Rule based systems
- Machine learning based systems
- Hybrid systems

2.2.1 Rule based systems

In rule-based systems, there usually exists a predefined set of rules. These rules guide the system to have categories based on relevant elements found in the text. Categories are predefined in this case. It would be better to consider a use case, let us assume we want to classify articles from a news website into two categories as Politics and Sports. To be able to perform this, it is required to define a set of words that will fall into two categories such as *"Sports"* (Basketball, Football, Ronaldo, etc.) and *"Politics"* (Biden, Trump, etc.). Now when a new text is fed into a classification system, the count of the words that relate to sports and politics are gathered. And if the number of words relating to sports is greater than the politics related words, then the text is considered to be more sports specific and classified as Sports category and vice-versa. These types of rule-based systems have their advantages and disadvantages. Since the rules are developed by humans, it is easily perceived by humans. But it requires high training for someone new to the domain to understand the terms and usually time-consuming to create rules. It is complicated to maintain the rules working for changing systems, new rules can be created but it often affects the way pre-existing rules work.

2.2.2 Machine Learning systems

The limitations found in rule-based systems leads us to the usage of Machine Learning systems. In this type of system, the classification is based on already learned observations. Figure 2.1 shows a diagrammatic depiction of the machine learning process used for classification of texts. By feeding the labelled examples to the machine learning algorithm, it is possible to learn the relationship between the text and output (i.e category).

This process requires sequential steps of data pre-processing, a transformation step which will be explained in detail in the following chapters. For this dissertation project, we will be using machine learning-based text classification to classify the tasks into epics. It only requires enough learning samples for making accurate predictions. Hybrid systems are an ensemble of rule-based and machine learning-based systems.

Before we research the machine learning aspects, it is important to have an understanding of the key terminologies [23] that will be used in the following chapters, We will explain each machine learning terminology with an analogy of interactions in a web page for understanding.

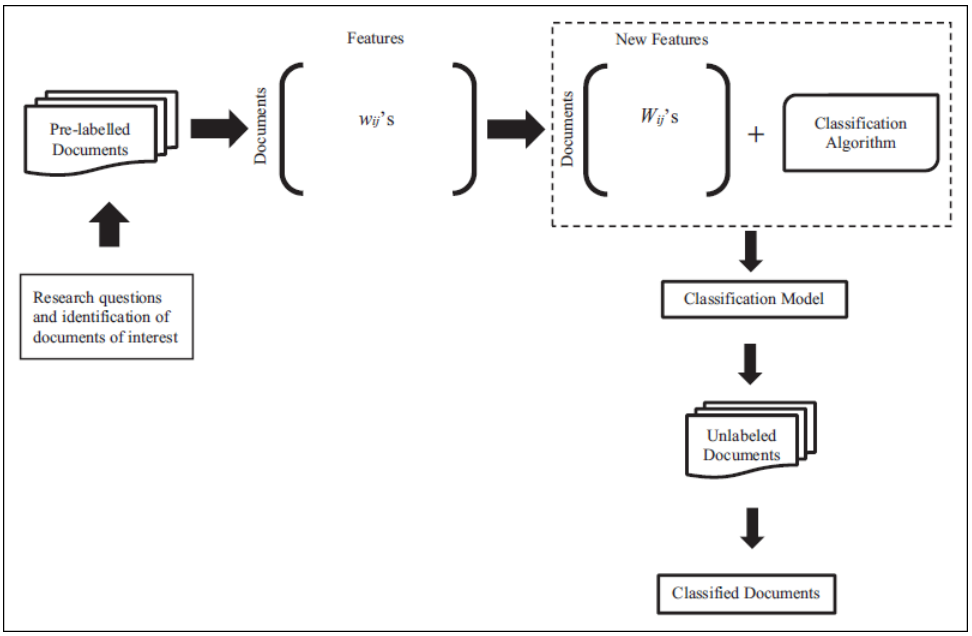

Figure 2.1: Diagrammatic depiction of text classification process represented by Kobayashi et al. [17]

- **Instance:** The instance is the object/element about which prediction will be made.
  For example, a news article in a web page containing *"Masters in software engineering"* can be considered an instance that needs to be classified as *"about medicine"* or *"about education"*.
- **Label / Class:** It is the output that will be produced by the prediction algorithm, or also it can be an answer supplied in the training data.
  For example: the label or class for a web page could be *"about education"*.
- **Feature:** A property in the instance used for prediction.
  For example, the web page might have the feature *"Glossary section about education"*.
- **Metric:** A number that is observed to measure the performance of the model.

2.3 Related work on text classification using JIRA data

2.3.1 Process overview

We found only one related work in the research area specific to the classification of tasks using machine learning in the context of project management tool. Hence in this section, details of a related work done on machine learning implementing text classification models titled *"Applying Machine Learning for Automatic User Story Categorization in Mobile Enterprises Application Development"* [15] is provided. An overview of the process the authors Matthias Jurisch et al. [15] follow using the classification methods to achieve the objective is summarized. This research

paper provides a baseline reference model to initiate the implementation part of our work. A discussion is made at the end of this section to highlight key takeaways from the conclusions of the research paper.

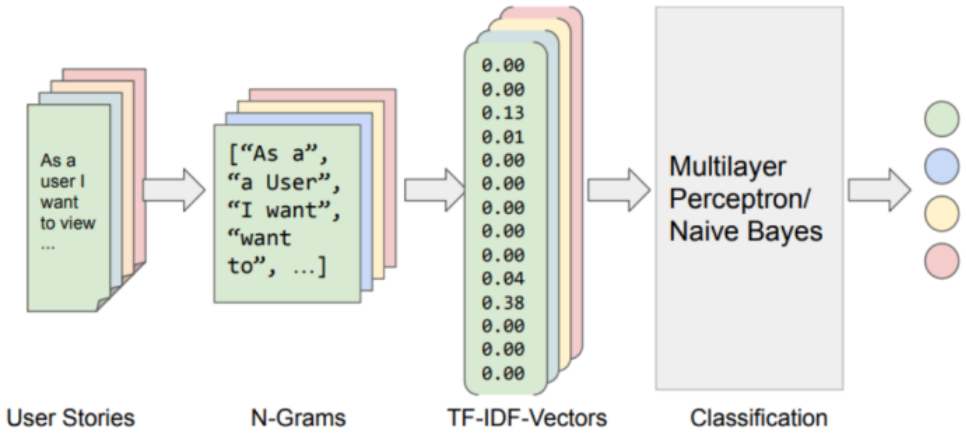

Figure 2.2: Overview of the text classification model represented by Matthias Jurisch et al. [17]

The text classification model used in the process is represented by figure 2.2. It is reported that recommendation of user stories can be done through information retrieval techniques in this paper [22]. A proposal to recommend the documents by choosing user stories from the same project and relating it to each other using measures of similarity in textual descriptions is provided. It is stated in this research paper that this technique is only able to recommend user stories from the same project and it is difficult to relate user stories from other projects. The research paper we reference for this work has addressed these issues in the information retrieval techniques [19] and proposed an approach by making modifications in the labelling process. An overview of the process is as follows.

- Categories are defined initially at the start of the process. The categories are based on the type of the user, the user story functionality, and the instance interacting with this function. Then a subset of user stories are selected and the defined categories are assigned manually to each one of the user stories. After the manually labelling process, it will result in a set of user stories that are assigned to three categories.

- These sets of labelled user stories are used as a training dataset for the text classification model. Once the model is able to learn from the provided examples, it is deployed to classify the user stories and assign a category.

As shown in the figure 2.2, at first the user stories are split as tokens and n-grams are formed. From the Jira dataset available, 403 user stories are selected in total and split into 330 stories for training and the remaining for the test set. An algorithm to find the document occurrences for each tuple of n-grams named "Term Frequency-Inverse Document Frequency" [2] which is

usually found in the information retrieval process is applied to the training data. This algorithm evaluates how important a word is to a document in a collection by providing a TF-IDF score. The significance of n-grams is derived by $tf_{d,t}$, which is the result of multiplying the frequency of n-gram (t) in a document (d), and the frequency of n-gram (t) in all the documents.

$$tf-idf: \quad W_{d,t} = tf_{d,t} * log(\frac{N}{df_t}) \tag{2.1}$$

In order to derive how common the term (t) occurs, the inverse document frequency $log(\frac{N}{df_t})$ is used, where N is the number of documents provided and df_t is the frequency of occurrence of all terms (t) in documents. The combination of both these frequencies explains how frequent or rare the word is in the entire document set. The most common words will have a score of 0. This calculation is done by dividing the total number of documents and the number of documents containing the word and taking the logarithm of document frequency. Hence the closer the number approaches 0 denotes the word relevance to the document. Otherwise, the number will approach 1. The two frequency numbers are multiplied to obtain a "Term Frequency-Inverse Document Frequency" score.

The vector obtained from the above equation 2.1 is referred to as a feature vector and is input to a text classification model for training. For classification models, a type of neural network named multilayer perceptron is used since the amount of training data is minimal. Also, a naive bayes classifier is used on the same training dataset and a comparison between the results is made. In the next discussion section, we will discuss the important findings reported in the research paper and also conclude the best performing model selected with the results.

2.3.2 Discussion

In this research paper, it is concluded that neural networks can complement classification approaches involving text data. The results are obtained without hyperparameter tuning but a standard configuration is used. The models were developed using sklearn [12]. Based on the comparison between multilayer perceptron and naive bayes classifier, it is found that multilayer perceptron performs better than naive bayes. Also considering the less complexity in building both the models, the neural network model is more suitable. The results show that with more training data, the multilayer perceptron performs much better than naive bayes.

Chapter 3

Classification of JIRA tasks

In this chapter we will describe the implementation details of this dissertation project. We will be covering the text classification process in sequential steps starting from the data preparation that involves data understanding, preprocessing, transformation stages and the application of text classification models, evaluting models using 6-fold cross validation and finally hyperparameter tuning of the model to improve the performance. To provide an overview of the experimental setup, we use *python* as source code language, and *Jupyter* notebook for experimentation purposes. This chapter also describes in detail the technology specifics used for the implementation which includes different classification models. For the classification process, the dataset used for this dissertation already has labels defined for training data, hence manually labelling the class is not required.

3.1 Dataset preparation

For classification models, data preparation is an essential step. The idea of data preparation for a text classification model is to remove irrelevant parts of text that can degrade the classification model performance. A table 3.2 aggregating the count of epics, count of tasks per project is presented in later section to have an overview of the data. Also plots such as scatterplot 3.2 are formed on the aggregated table to see the number of instances per project. This analysis is important to decide the best fit classification model that can be applied for this work. Data preparation in this work is done in multiple steps such as data preprocessing that applies text cleaning techniques. The usage of these modules will be explained in detail under relevant sections and the next section is followed by data transformation which is commonly termed as feature extraction.

3.1.1 Data understanding

The data obtained is from the Jira tool used by the project management team within an organization named Fraunhofer. The data contains tasks from different projects. A detailed representation of important features the dataset contains is provided in table 3.1.

Field	Type	Description	Example / Possible Values
project_id	INT	Unique project identifier	459
project_key	TEXT	Short textual representation of a project as key	QUAEWATSICA
project_name	TEXT	Name of the project	Esse incidunt est blanditiis amet eum qui.
summary	TEXT	Description of the task	Review all flows that require notifications
type	TEXT	Type of task	New Feature;Bug;Improvement;Task;
issue	TEXT	Key for the task, used to uniquely identify the task	QUAEWATSICA-11
epic	TEXT	Key for the epic, used to uniquely identify the epic	QUAEWATSICA-10
labels	TEXT	Label for the task	iOS
created	TIMESTAMP	Date and time of the records when inserted into database	2018-09-28T16:07:13.000
updated	TEXT	Date and time of the records when updated in the database	2019-09-28T16:07:13.000
issue_status	TEXT	Current status of the task	Open;Resolved;Closed;New;
user_name	TEXT	Name of the user in the system	carlos robertson
user_key	TEXT	Key of the user, mostly formed from the user name	carlos.robertson
assignee	TEXT	Name of the user to whom task is assigned	carl white
creator	TEXT	Name of the user who created the task	kathy owens
reporter	TEXT	Name of the user who reported the task	kathy owens
priority	TEXT	Priority of the task	Trivial;Minor;Major;Critical;Blocker
resolution	ENUM	Resolution for the task	Fixed;Duplicate;Done;Incomplete

Table 3.1: Useful dataset features

For this work we will be using the labelled data, so we filter the data having tasks and epic mapping and after, the records count to *1924* tasks which belongs to *132* epics. The data distribution is not even with the largest category of epic containing *621* tasks and the lowest category containing *6* tasks. The dataset used for both training and validation spans across different projects

between the year *2015* to *2020*. The development team assigned to each project works based on the agile methodology in Jira, so the tasks are worked in agile development phases named as sprints. As can be seen in the class diagram 3.1 the *jira_issues* table is normalized and has a relationship with the project and sprints table. The project details such as *id*, *key*, *name* are present in the projects table. Details regarding the sprints such as *id*, *state*, *start_date*, *end_date*, *complete_date*, *activate_date*, *goal* are present in the *sprints* table. It is also found in the *users* table, the details of all the users inside the project teams using the Jira tool. The tasks created under all the projects and sprints are managed in the *jira_issues* table which keeps track of all the tasks information such as *story_summary*, *issue_type*, *issue_key*, *epic_key*, *sprints_info*, *issue_status*, *priority*, *assignee*, *creator* and *reporter*. In order to keep track of all the history of the transactions done related to the tasks such as adding attachments, modifying the rank of the tasks, reassigning the task to a different person, etc, a *changelog* table is used. Since all the changelogs trace back to the original story created, there is a *changelog* column in the *jira_issues* table containing all the changelog records in json format. For this work we are most interested in the textual information related to the task highlighted in Figure 3.1 such as *summary*, *issue_key*, *epic_key*, since these columns denote the task and epic relationship. *summary* is the textual description about the task, *issue_key* uniquely identifies the task and *epic_key* uniquely identifies the epic.

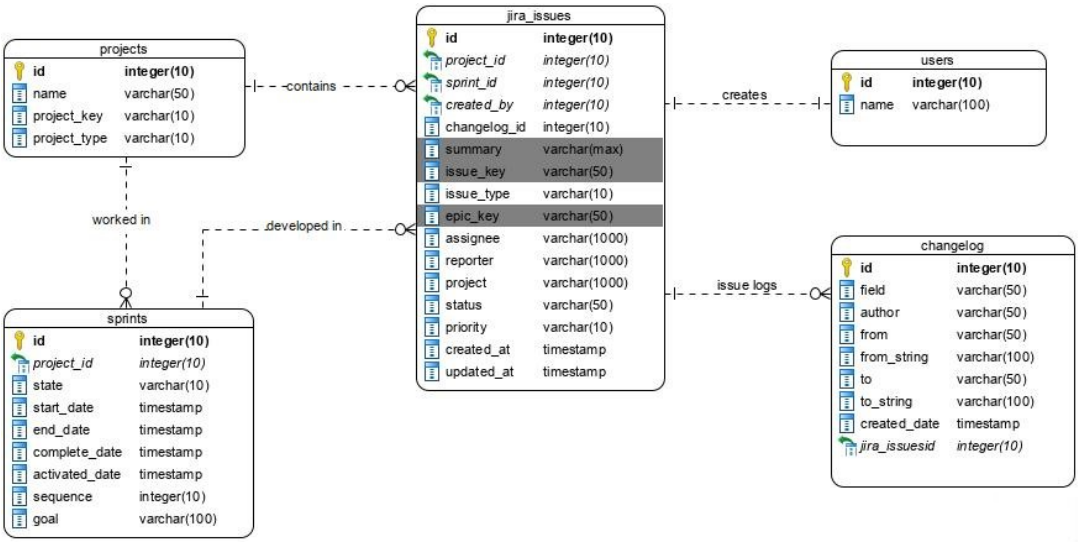

Figure 3.1: Class diagram of the jira dataset

The tasks are aggregated and managed in the table *jira_issues*. This table is a denormalized view of all the tables present in figure 3.1 and contains detailed information of the tasks with project, user, sprint details. Since the text classification model will work based on textual information, only a few columns related to the task that are most relevant to the model are considered in the output of the data preparation stage. In this case, the task occurring for one project does not

relate to other projects so the model we will be able to only predict tasks related to each project. Hence the training will also involve the project data. For understanding the density of the task occurrences for project it is presented by aggregating the counts in the table 3.2.

project_id	**project_key**	**task_count**	**epic_count**
109	EXLUEILWITZ	47	6
286	REICIENDISMACGYVER	81	2
292	VELITONDRICKA	27	3
455	ETLANG	165	28
459	QUAEWATSICA	621	12
581	MOLLITIAWEHNER	78	7
616	OPTIOHERMANN-RUTHERFORD	179	15
652	FACILISJAKUBOWSKI	72	6
688	ODIODURGAN-BOEHM	156	15
798	ODIORUECKER-WATSICA	22	5
818	CORPORISVEUM-HEATHCOTE	63	10
840	NISIRUTHERFORD-TROMP	271	14
938	FACILISLUBOWITZ	6	1
950	CONSEQUATURCASSIN-GLOVER	136	8

Table 3.2: Aggregate of tasks and epics per project

It is found from figure 3.2 that there is a class imbalance in the number of instances found per project.

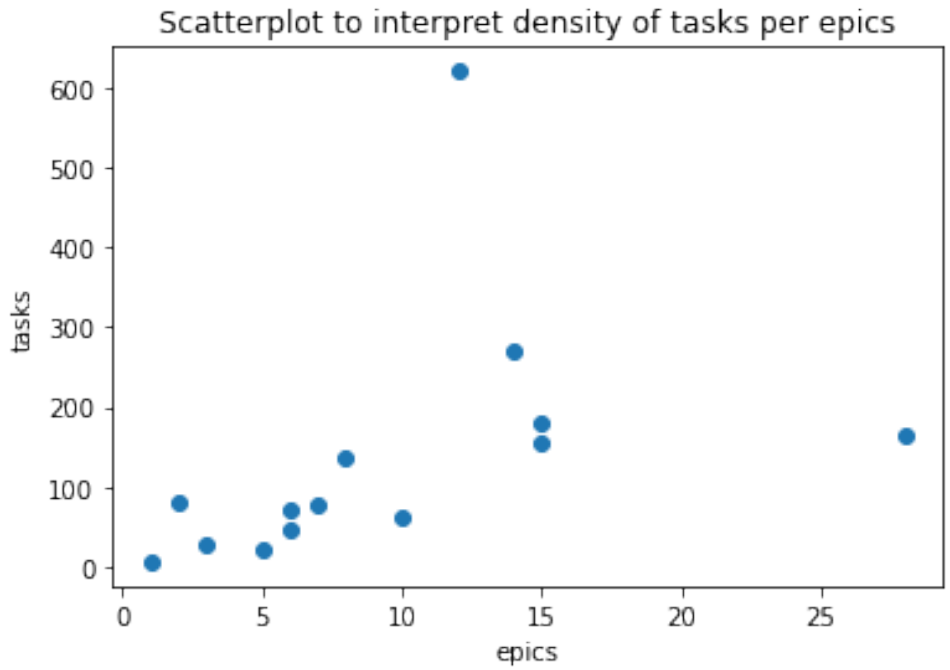

Figure 3.2: Plot to interpret the density of tasks vs epics per project using aggregated project data from table 3.2

3.1.2 Text preprocessing

Text preprocessing is the first step commonly found in the text classification process. This step improves the performance of the model by reducing noise found in the text data and also prepares the data suitable for feature extraction. Text documents available in the industry usually contain irrelevant words namely stop words, spellings, etc. For most of the classification algorithms these irrelevant features can reduce the performance. In this section we will be briefing about different text cleaning techniques available provided as reference by [18]. We also provide the source code repository here [1]. We perform the preprocessing steps, training and code required for prediction is available in the repository. Code snippet 3.1 used for text cleaning is provided in this section. We use a python module named "spaCy" [25] that applies industry-standard and bundled with natural language processing techniques to help text classification problems. This module is used for text cleaning purposes for our problem.

```
def text_preprocessing(text, accented_chars=True, contractions=True,
                       convert_num=True, extra_whitespace=True,
                       lemmatization=True, lowercase=True, punctuations=True,
                       remove_html=True, remove_num=True, special_chars=True,
                       stop_words=True):
    """preprocess text with default option set to true for all steps"""
    if remove_html == True: #remove html tags
        text = strip_html_tags(text)
    if extra_whitespace == True: #remove extra whitespaces
        text = remove_whitespace(text)
    if accented_chars == True: #remove accented characters
        text = remove_accented_chars(text)
    if contractions == True: #expand contractions
        text = expand_contractions(text)
    if lowercase == True: #convert all characters to lowercase
        text = text.lower()

    doc = nlp(text) #tokenise text

    clean_text = []

    for token in doc:
        flag = True
        edit = token.text
        # remove stop words
        if stop_words == True and token.is_stop and token.pos_ != 'NUM':
            flag = False
        # remove punctuations
        if punctuations == True and token.pos_ == 'PUNCT' and flag == True:
            flag = False
        # remove special characters
```

[1] https://github.com/promessa-project/jira-task-classification.git

```
        if special_chars == True and token.pos_ == 'SYM' and flag == True:
            flag = False
        # remove numbers
        if remove_num == True and (token.pos_ == 'NUM' or token.text.isnumeric()) \
        and flag == True:
            flag = False
        # convert number words to numeric numbers
        if convert_num == True and token.pos_ == 'NUM' and flag == True:
            edit = w2n.word_to_num(token.text)
        # convert tokens to base form
        elif lemmatization == True and token.lemma_ != "-PRON-" and flag == True:
            edit = token.lemma_
        # append tokens edited and not removed to list
        if edit != "" and flag == True:
            clean_text.append(edit)
    clean_text = "|".join(clean_text)
    return clean_text
```

Listing 3.1: Source code used for cleaning text to be used for text classification model

3.1.2.1 Tokenization

Tokenization is a type of pre-processing technique that involves breaking the complete text into words, symbols. These individual terms are referred to as tokens. Breaking the text into tokens allows text classification models to classify the tokens based on the relevance, for example:

> *He used to play football for six hours daily, till he was at the age of six.*
>
> In this scenario, the tokens are as below,
>
> *["He" "used" "to" "play" "football" "for" "six" "hours" "daily" "till" "he" "was" "at" "the" "age" "of" "six"]*

The source code we used to achieve the tokenization in our dissertation project is as follows,

```
import spacy
...
nlp = spacy.load('en_core_web_sm')
...
doc = nlp(text) #tokenise text
```

Listing 3.2: Source code used for tokenization in text preprocessing

3.1.2.2 Stop words

Most text documents may include words that do not add significance to the meaning of the text. A few examples are [“a”, “at”, “it”, “they”, “itself”]. In text mining and "Natural Language Processing", these stop words are used to remove insignificant words and this enables the classification model to work with important words. The source code implemented to remove stop words is as follows,

```
import spacy
...
nlp = spacy.load('en_core_web_sm')
...
doc = nlp(text) #tokenise text
...
for token in doc:
    flag = True
    edit = token.text
    # remove stop words
    if stop_words == True and token.is_stop and token.pos_ != 'NUM':
        #remove stopwords
```

Listing 3.3: Source code used for removing stopwords in text preprocessing

3.1.2.3 Lemmatization

Lemmatization is an NLP process to convert the word from the text to its root form (lemma). This allows consistency across the vocabulary of text. The source code used is given below,

```
import spacy
...
nlp = spacy.load('en_core_web_sm')
...
doc = nlp(text) #tokenise text
...
for token in doc:
    flag = True
    edit = token.text
    ...
    # convert tokens to base form
    elif lemmatization == True and token.lemma_ != "-PRON-" and flag == True:
        edit = token.lemma_
```

Listing 3.4: Source code used for lemmatizing text in text preprocessing

3.1.2.4 Abbreviation and Slang

"Abbreviation and Slang" is another type of anomaly found in the text document. This type of anomaly can be handled in the text preprocessing step. An abbreviation is a contracted version of a word, for example: “Support Vector Machines” can be abbreviated as SVM. Slang is an informal language that is derived from the actual text to convey the same meaning. In order to normalize these texts, it is preferred to convert the informal language to formal language. The source code used to achieve this is as follows.

```
1 import contractions
2
3 def expand_contractions(text):
4     """expand shortened words, e.g. don't to do not"""
5     text = contractions.fix(text)
6     return text
```

Listing 3.5: Source code used for handling abbreviations and slang in text preprocessing

Other text cleaning techniques such as remove html tags, remove accented characters, trim whitespaces are also applied to the given text to maintain uniformity in the text features. The source code can be referred in detail from section 3.1.

3.1.3 Text transformation

Transforming the text to a suitable structure that can be understood by classification model is critical, which is provided by the text transformation step. The majority of the classification models accept vectors as input, thus it is essential to represent the document as a vector and the vocabulary of text as a matrix. In this case the features are individual terms of the vocabulary. Assuming that the words are the lowest level of unit in a text, the vector size is equal to the unique elements in a vocabulary. Hence the total document can be represented as the weight of individual terms in a vocabulary. The weight can be the count of individual terms or binary weight of the terms. With binary weighting, the presence of word is denoted by 1 or 0 for absence of word. When transformation is applied to the entire vocabulary, the rows will be the documents and columns become individual terms in the vocabulary, each entry will be the weight of that term in each document. *“TF (Term frequency)"* are calculated by taking the count of term occurrences in a document. It is a common practice to normalize the term occurrence by dividing the term count with the maximum term count in the document to normalize the frequency scoring for both high frequently occuring and less frequently occuring words. Other weighting options include weighting terms with respect to the vocabulary. Common weights that are based on vocabulary are given by *“IDF (Inverse document frequency)”*. The equation on how this frequency is calculated is provided in equation 2.1 in this dissertation work. In this dissertation work, both the weights are combined *"TF-IDF (Term frequency-Inverse document frequency)"* so that the resulting weights represent the importance of a term in a document with respect to the vocabulary. It is important to note

that the performance of a text classification model is highly dependent on the text transformation step. Although the order of the words in a text is not considered in this process, it is simple and effective. It can be possible that a few aspects of the word are ignored during the transformation step. So during the evaluation step, when the classification performance is noted to be poor it is recommended to analyse the transformation step.

We will be looking at a detailed flow of the text transformation done after text preprocessing with an abstract flow representation from a reference paper [16]. The preprocessed data set in our case is the text cleaned from the field *"summary"* present in table 3.1. This text data will be input to the transformation algorithm. An illustration of the process for calculating "Term frequency-Inverse document frequency" score is given in figures 3.3 and 3.4

Figure 3.3: Example for illustrating TF calculation
reference from [16]

In figure 3.4 the document frequency score is represented, the equation is provided below as referenced in [16],

$$Document\,frequency: \quad d_{f,t} = \frac{|d_j \in D : t_j \in d_j|}{|D|} \tag{3.1}$$

In the above equation, $|D|$ represents the total documents count and $|d_j \in D : t_j \in d_j|$ denotes the term occurrence in the number of documents. This document frequency is then provided to the calculation referenced in equation 2.1 to calculate the "Term frequency-Inverse document frequency" score for each terms in the document.

	Doc1	Doc2	Doc3	Doc4	$\|d_j \in D: t_i \in d_j\|$
Cloud Computing	1	3	2	2	4
Internet of Things	1	0	4	0	2
BigData	1	1	0	1	3
Security	1	2	3	3	4
Scientific Workflow	0	2	0	9	2
Scheduling	0	1	0	5	2
Resource Management	0	4	2	0	2
Cloud Storage	0	0	0	0	0
Privacy	0	0	0	0	0
Cloud	2	0	0	0	1

Keyword	DF
Cloud Computing	1
Internet of Things	0.5
BigData	0.75
Security	1
Scientific Workflow	0.5
Scheduling	0.5
Resource Management	0.5
Cloud Storage	0
Privacy	0
Cloud	0.25

Figure 3.4: Example for illustrating DF calculation
reference from [16]

We use the scikit-learn [12] package that offers modules for transforming the matrix of counts to represent "Term frequency-Inverse document frequency" scores. The source code is available here in Listing 3.6.

```
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer

# Get the counts of words
count_vect = CountVectorizer(binary=True)
word_counts = count_vect.fit_transform(data['summary'])


# Get the frequency of the words
tfidf_transformer = TfidfTransformer()
tfidf = tfidf_transformer.fit_transform(word_counts)
```

Listing 3.6: Source code used for performing text transformation

3.2 Text classification techniques

After performing the data understanding and text transformation steps, types of classification techniques are selected in this section. From Figure 3.2 we could perceive the presence of unbalanced classes. There are many approaches of classification to categorize texts for this type of problem. In general classification problems can be divided into supervised type, unsupervised and semi-supervised. We have chosen a supervised classification technique since the labels are already available along with the text to be classified, so it is possible for the model to learn from the examples. Although it might be challenging to train a classification model in a highly imbalanced class, we have taken suitable measures to handle the imbalance problem which will be covered in the sections below. From the numerous models available, we have selected a few as referenced [27] such as random forest, LinearSVC (SVM with linear kernel), Naive Bayes, XGBoost, SGDClassifier. We benchmark the five models in the discussion section at the end. It is important to note that choosing the right parameter while training the model is critical for model performance, which will be covered while discussing the details of each model.

Machine learning problem

In ML terms, the classification process comes under the principle of supervised learning, where before the actual classification is initiated the model is provided with knowledge about the classes. Supervised learning could be sometimes difficult if the labels to classes need to be assigned manually which is highly inefficient in large datasets. When supervised learning needs to be applied to text datasets, it becomes complex to handle non-linear data distributions or highly imbalanced classes which could be seen in Figure 3.2. Machine learning model work in order to learn a function [**?**]. It can be better represented using the formula,

$$Y = f(X) + e \tag{3.2}$$

This formula 3.2 is a basic representation of model trying to learn a function in order to predict the variable (Y) for new inputs (X). There is usually a error (e) attached with the new input (X) that can deteriorate the prediction ability for the model. So measures are taken to maintain the error to as minimal as possible.

Implementation of text classifiers

In this section we discuss the ML models used for train and test phases during the implementation. A brief overview about the models and justification for choosing the models for text classification is provided. We also cover the advantages of each model and steps taken to optimise the performance of the models.

3.2.1 Multinomial Naive Bayes

Multinomial Naive Bayes is a type of Naive Bayes classifier and is often used as a baseline model for classifying text. The term “naive” [1] in the "Naive Bayes" naming is due to its assumption that a feature appearing in the data is not relevant to other features. These models are referred to as *“probabilistic classifiers”* due to their ability to predict a probability distribution over a set of classes and after output a highest probability class. This probability is calculated based on the Bayes theorem. The Bayes theorem can be defined using the formula [8],

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)} \tag{3.3}$$

In the above equation, A denotes the class and B represent the features. From the equation 3.3, $P(A|B)$ denotes the probability of class A provided all the features B.

For a text dataset, it is recommended to use a multinomial NB model due to the presence of multidimensional data. Multinomial NB functions by assuming the input document as a “bag-of-words” and considers frequency of the words into account. Since in the text transformation step, we output a word frequency based on tf-idf it can complement the functionality of this type of model. We use the *"MultinomialNB"* module from scikit-learn python package in this work. The source code in extension to text transformation step 3.6 is as follows,

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.naive_bayes import MultinomialNB

# Get the counts of words
count_vect = CountVectorizer(binary=True)
word_counts = count_vect.fit_transform(data['summary'])

# Get the frequency of the words
tfidf_transformer = TfidfTransformer()
tfidf = tfidf_transformer.fit_transform(word_counts)

model = MultinomialNB()
clf = model.fit(tfidf, train_labels)
```

Listing 3.7: Source code used for training MultinomialNB model

3.2.2 Random forest

Random forest models uses ensemble learning techniques to classify documents. It was developed in 1995 by Tin Kam Ho [13]. Later in the year 2000 Breiman experimented with this models to increase the accuracy of classification algorithms. Since random forest uses an ensemble of

decision trees it was proved that it is efficient in classification approaches [5]. This type of model works based on creating random decision trees. As can be seen in the figure 3.5, an illustration of how a random forest works with trees in parallel is provided. The properties that will be supplied during the training stage of the model such as number of selection of trees *(n_ estimators)*, depth of trees *(max_ depth)* hugely influences the performance. Although random forests are efficient in training text data, it can be slower in creating predictions after training steps [3].

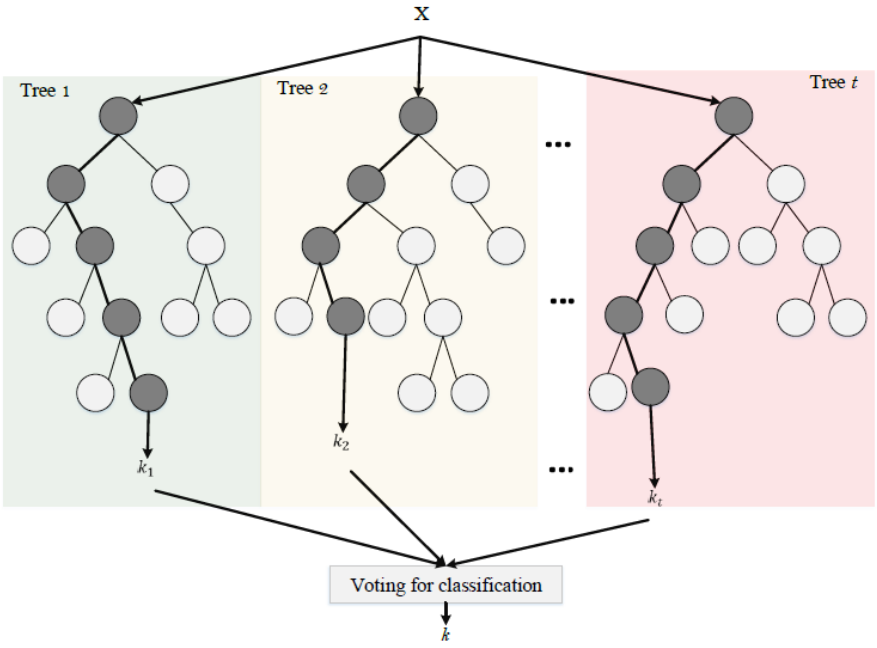

Figure 3.5: Illustration of Random forest tree
referenced from [18]

The source code simplified to show the implemented Random forest model is as follows,

```
...
from sklearn.ensemble import RandomForestClassifier

...
model = RandomForestClassifier(n_estimators=1000, max_depth=3, random_state=0)
clf = model.fit(tfidf, train_labels)
```

Listing 3.8: Source code used for training Random forest model

3.2.3 Linear SVC

Support Vector Machine is one of the supervised machine learning models most suited for classification tasks ranging from text categorization, information extraction, etc. SVM model was proposed by Vapnik [28] to solve problems with only two-classes. As shown in figure 3.6, these

models are based on finding a separation between the hyperplanes defined by the classes [6]. It could be stated from [27] that the complexity of the classifier is dependent on the number of support vectors instead of the dimensions in data hence it is most suited for highly distributed data. They are seen to produce consistent hyperplanes for training data sets even if repeated, hence even when the data is sparse it produces the same accuracy.

Standard kernels used can be customized to increase the performance of the model. In our case we used a linear kernel which is also a variant of SVM named LinearSVC. There are reasons for linearSVC to be a best fit for text classification problems, since the text usually has a lot of *features (words)*. Training a SVM with a kernel as linear is found to be faster as it is only required to optimize the C (regularization parameter). Generally having a higher C parameter is not desirable [8].

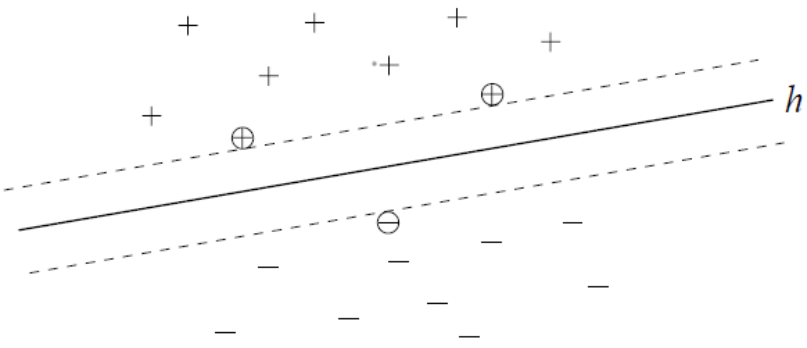

Figure 3.6: Illustration of SVM hyperplane pattern
referenced from [14] denotes the positive and negative separation between training examples with highest margin. Support vectors are the examples closer to the hyperplane (examples marked in circles)

The source code simplified to show the implemented LinearSVC model is as follows,

```
1 ...
2 from sklearn.svm import LinearSVC
3
4 ...
5 model = LinearSVC()
6 clf = model.fit(tfidf, train_labels)
```

Listing 3.9: Source code used for training LinearSVC model

3.2.4 SGDClassifier

"Stochastic Gradient Descent (SGD) Classifier" is a type of linear classifier implemented with an optimization method named SGD. This classifier shares common characteristics with LinearSVC

and is more applicable to problems with sparse data arrays having floating point values as features. The optimization used for this classifier reduces the loss function (rate of incorrect prediction for samples) by applying an algorithm known as gradient descent. Gradient descent also supports in finding the right parameters. This type of classifiers has drawn more attention towards text classification and natural language problems that involve sparse distributed classes within data [11]. The advantages of using SGD for text classification far outweighs the limitations. A few of the advantages are ease of implementation and availability of optimized hyperparameters such as regularization, number of iterations, learning rate etc. This technique takes care of the loss function and classification penalties by implementing a stochastic gradient descent learning path.

The source code simplified to show the implemented SGDClassifier is as follows,

```
from sklearn.linear_model import SGDClassifier

...
model = SGDClassifier()
clf = model.fit(tfidf, train_labels)
```

Listing 3.10: Source code used for training using SGDClassifier

3.2.5 XGBoost

XGBoost falls under a family of tree algorithms that implements boosting techniques. This type implements a regularized model to prune overfitting [7]. Hence this type of model is capable of handling sparsity in data. The machine learning components used in this model implement *"Gradient Boosting"* methodology. It is able to solve problems faster and in a much efficient way because of a technique named *"parallel tree boosting"*. This model is not part of *scikit-learn* package, instead a separate module named *xgboost*. The source code used for implementing XGBoost is as follows,

```
from xgboost import XGBClassifier

...
model = XGBClassifier()
clf = model.fit(tfidf, train_labels)
```

Listing 3.11: Source code used for training using XGBoost

3.3 Model Evaluation

We evaluate our model using a model evaluation technique known as cross validation.

	fold1		fold2		fold3		fold4		fold5		fold6	
	train	test	train	test	train	test	train	test	train	test	train	test
QUAEWATSICA	517	104	517	104	517	104	518	103	518	103	518	103
ETLANG	137	28	137	28	137	28	138	27	138	27	138	27
VELITONDRICKA	22	5	22	5	22	5	23	4	23	4	23	4
MOLLITIAWEHNER	65	13	65	13	65	13	65	13	65	13	65	13
EXLUEILWITZ	39	8	39	8	39	8	39	8	39	8	40	7
ODIODURGAN-BOEHM	130	26	130	26	130	26	130	26	130	26	130	26
REICIENDISMACGYVER	67	14	67	14	67	14	68	13	68	13	68	13
OPTIOHERMANN-RUTHERFORD	149	30	149	30	149	30	149	30	149	30	150	29
ODIORUECKER-WATSICA	18	4	18	4	18	4	18	4	18	4	19	3
NISIRUTHERFORD-TROMP	225	46	226	45	226	45	226	45	226	45	226	45
CORPORISVEUM-HEATHCOTE	52	11	52	11	52	11	53	10	53	10	53	10
CONSEQUATURCASSIN-GLOVER	113	23	113	23	113	23	113	23	114	22	114	22

Table 3.3: Table showing statistics about number of records for train, test split for each fold in a 6-fold validation

For this work, a 6-fold cross validation is applied on the project dataset. We create 6 different models per project. In table 3.3 statistics obtained after performing the 6-fold split are provided. The train and test dataset contains the tasks and epics associated for each projects. Since we already have obtained the prediction results from the different classifiers from Section 3.2, it is recommended to evaluate the model once it is trained to check the generalization ability of the model in order to predict new data [4]. A model that has learnt from the complete dataset can only predict well for existing dataset, but will be unable to generalize for new data. This problem is referred to as *overfitting*. Another problem known as *underfitting* where the model has not learnt well from the dataset, so it would not be possible for the model to accurately predict new data. Both problems pose a major concern for model accuracy, and lead to poor classification performance. Now a possible question to approach this problem can be, *"How can we measure the ability of the model to generalize for new data ?"*. Generally for evaluating the model, a subset of data (also known *hold-out* dataset) which is taken from the existing data is used for validation purposes and the other major chunk of data will be used for training. This process is repeated for specific iterations, using "data resampling procedures". One such type of sampling method is referred to as cross validation.

In this section we will cover the evaluation of the model using cross validation procedure. Cross validating the datasets and repeatedly training the model on each dataset allows us to tune the model parameters and inturn improve the performance of the model. A type of cross validation known as k-fold cross validation is used in this dissertation for evaluating the model. In the k-fold

validation process the single hold out method is repeated “k” number of times in a way that the test set does not overlap. The available dataset is split into k non-overlapping sets of almost equal sizes. The term “fold” denotes the subsets of data after splitting. In this case, the training set comprises “k-1” subsets. The repetition of this process is done until all the “k” subsets have served as validation data. The cross validation result will be the average of the results obtained from each iteration. An illustration for this process where k=10 (10-fold cross validation) is given in figure 3.7.

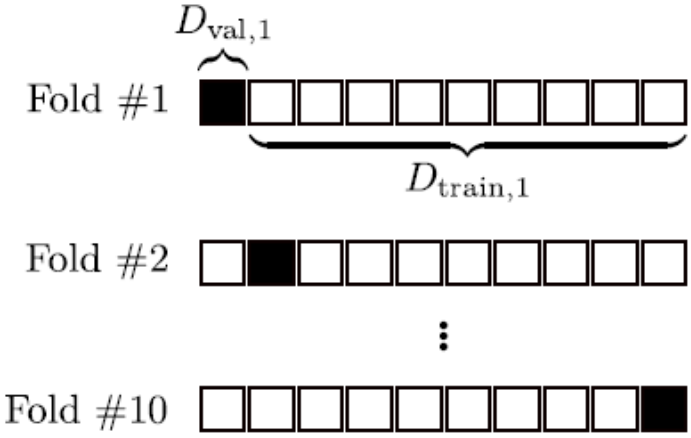

Figure 3.7: 10-fold cross validation process
The term D denotes the dataset. Each validation set will contain 10 percentage of the total set. Model is trained on training set and applied to validation set. [4]

As the data present in each project are non-overlapping, it is decided to validate using the split by project approach. For this work data is split into 6-folds, since the lowest number of instances that can be split equally is 6 as can be seen in table 3.2. The evaluation process used is as follows, the project data is taken from the complete dataset and input to a k-fold method provided by *“scikit-learn”* package which takes care of the equal disjoint splits required. The repeated validation is then performed on each split and evaluation metrics are obtained. And finally the average of the metrics obtained from all the projects measures the performance of the model. We give importance to f1-score which will be covered in details in the results chapter 4. Source used to perform k-fold cross validation is as follows.

```
1 import numpy as np
2 from sklearn.model_selection import KFold
3 from sklearn.metrics import f1_score
4
5 proj_f1_scores = []
6 model_f1_scores = []
```

```
7
8  def evaluate_model():
9      # apply cross validation with 6-fold on project dataset
10     for proj in project_list:
11         kf = KFold(n_splits=6, shuffle=True, random_state=1)
12         for train_index, test_index in kf.split(x, y):
13             x_train, x_test = x[train_index], x[test_index]
14             y_train, y_test = y[train_index], y[test_index]
15
16             # text transformation steps
17             ...
18
19             predicted = predict.predict_model_cv(clf, count_vect, x_test)
20             proj_f1_scores.append(accuracy_score(y_test, predicted))
21
22     model_f1_scores.append(np.array(proj_f1_scores).mean())
```

Listing 3.12: Source code used for K-fold cross validation

3.4 Hyperparameter Tuning for ML models

The supervised models used for this work such as Random forest, LinearSVC, XGBoost use hyperparameters that need to be set before we can train them. These chosen settings can be default values that are mentioned in the software library, configured manually by the user based on trial-error to get maximum predictive capacity, or can be from a tuning procedure. We start our training process with the default setting mentioned in the *"scikit-learn"* package, and after the model evaluation results we make use of the automatic tuning procedure. Although we use the default settings that may be optimal according to the literature and industry standards, we tend to improve the performance suitable for this model by modifying the settings. We observed that manually setting the hyperparameter is time-consuming and highly inefficient, hence it is decided to perform an automatic hyperparameter search that is provided by an optimization framework named *"Optuna"*. This framework employs a *"define-by-run" API* strategy which dynamically constructs the parameter search. Since we already have the model evaluation function as referred to in section 3.12, it is possible to configure optuna runs on this function. We try to optimize the results of four models such as Random forest, LinearSVC, SGDClassifier and XGBoost. Few parameters that can be optimized in the models are "C" parameter in LinearSVC and "max_depth" in RandomForestClassifier. The default parameters are initially passed to this framework, and 100 run trials are specified. Optuna performs 100 trials by automatically changing the parameter settings for each run and obtains the result. The most optimal run with the improved result is provided at the final iteration. Source code used for performing the optuna run trial is as follows.

```
1  import optuna
2
```

```
3  import models.evaluate_model as model
4
5  def optimize_model():
6      study = optuna.create_study(direction='maximize')
7      study.optimize(model.evaluate_model, n_trials=100)
8
9  trial = study.best_trial
10
11 print("Optimal F1 Score: {}".format(trial.value))
```

Listing 3.13: Source code used for model optimization

3.5 Evaluation Metrics

Different characteristics of the model can be evaluated by the different metrics available. For this work, we use the evaluation metrics present in table 4.1 and present the results. An overview of the evaluation metrics will be discussed in this section. Relevant scenarios for the usage of the different metrics are provided below each metric description. Metrics beyond accuracy such as precision, recall, f1-score and weighted accuracy are discussed.

- **Accuracy:** Accuracy is easier to understand and easily applicable to problems related to binary data and multiclass. It is calculated by the proportion of true predictions among the total predictions.

$$Accuracy: \frac{TruePositive + TrueNegative}{TruePositive + FalsePositive + TrueNegative + FalseNegative} \tag{3.4}$$

  Scenario for usage: Accuracy is more suitable for evaluating problems that deal with class balance or in other words no class imbalance.

  Caveats: We would not want accuracy for evaluating model performance where the target class contains sparse data. Assuming we would want to predict if there will be a major earthquake in the upcoming 10 years, which is a case of imbalanced problem where 99 % of instances belong to one class and the rest 1 % instance to the other class. In this case, the model would be able to predict every instance that belongs to the majority class and the result for accuracy would be 99 %. As mentioned in the article [2], the accuracy might not be proper evaluation measure for imbalanced classification problem.

- **Precision:** Precision denotes the proportion of true positives (predicted as true) that are actually true. The formula for calculating precision is as follows,

[2] https://www.fharrell.com/post/class-damage/

$$Precision : \frac{TP}{TP+FP} \tag{3.5}$$

Scenario for usage: Precision is more suitable for problems where predicting the positives are of much importance. We would not want precision as a metric for the earthquake prediction as well. The precision will be 0 since there was no positive prediction. Precision used in banking problems to predict the interest rate is important, otherwise, the system may lose the trust of customers.

Caveats: Predicting only positives will not cover the negative scenarios, so chances of missing the negative scenario may not evaluate the complete performance of the model.

- **Recall:** Recall is more similar to precision. It calculates what proportion of class that is positive is predicted as positive.

$$Recall : \frac{TP}{TP+FN} \tag{3.6}$$

Scenario for usage: Recall for prediction of earthquake problem will be 0 also since it does not predict positives. This metric is applicable for scenarios where classifying the maximum number of positives is important. Let us look at an example of a system to test a person for covid positive. The system may want to capture the maximum positive samples even if it is not sure at first.

Caveats: Recall is 1 if all the predictions are true positives. This leads to our next metric which is the tradeoff between precision and recall.

- **F1-Score:** F1 Score combines both precision and recall. This metric as described in [20] is the harmonic mean of recall and precision.

$$F1Score : 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3.7}$$

Scenario for usage: When it is required for the model to have positive recall and precision.

- **Weighted accuracy:** Weighted accuracy could be related to the accuracy, but this metric takes into account the mean of each data point concerning the whole data set. The formula used in this problem is as follows,

$$Weightedaccuracy : \frac{count(task)}{count(totaltask)} * accuracy \tag{3.8}$$

Source code used in our problem for evaluating the classification of JIRA tasks using the "scikit-learn" package is as follows,

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
...
accuracy_score(y_test, predicted)
f1_score(y_test, predicted, average='weighted')
precision_score(y_test, predicted, average='weighted')
recall_score(y_test, predicted, average='weighted')
weighted_score = (len(x_test) / len(x)) * accuracy_score(y_test, predicted)
```

Listing 3.14: Scikit-learn function to calculate evaluation metrics

3.6 Discussion

We have built the classifiers with evaluation performed on each model, and finally an optimization strategy done. It is observed that the usage of optuna framework reduced the difficulties in the manual parameter settings. In this section, we also discuss the optimal hyperparameter settings that helped on the performance improvements for the model, as can be seen in table 3.4.

Classifier	Hyperparameter	Type	Value
Random Forest	n_estimators	integer	1000
	max_depth	integer	32
LinearSVC	kernel	string	linear
	gamma	string	auto
	C	numeric	1
XGBoost	n_estimators	integer	900
	max_depth	integer	20
	reg_alpha	integer	3
	reg_lambda	integer	3
	min_child_weight	integer	2
	gamma	integer	3
	learning_rate	numeric	0.4
	colsample_bytree	numeric	0.2
	nthread	integer	-1

Table 3.4: Optimal Hyperparameter setting for the classifiers

A quick summary of the advantages and pitfalls of the techniques used in this chapter for text transformation and text classification as referenced from [18] also applicable for this work is provided in table 3.5.

Model	Usage	Advantages	Pitfalls
TF-IDF	Text-transformation	1. More accurate calculation for frequency of words due to IDF. 2. Common words like ("are", "the") does not affect result.	1. Importance is not given to the position of words in text. 2. Does not understand the semantics of text.
Multinomial Naive Bayes	Text Classification	1. More suitable for classification problems, since text contains large data. 2. Faster computation times. 3. Easy implementation.	1. Works based on the assumption of distribution in data.
Random forest	Text Classification	1. Faster to train these decision tree ensembles. 2. Does not require extensive data preprocessing.	1. Slower to predict. 2. Need to choose the number of tree carefully, since it affects model performance. 3. Time complexity for prediction increases with more number of trees.
LinearSVC	Text Classification	1. Works well against overffitting problems in text data due to presence of high dimensional data points.	1. Model performance depends on selecting the right kernel 2. Hyperparameter "C" used for regularization is of much importance for accuracy. 3.2.3
SGDClassifier	Text Classification	1. Supports multi-class problems. 2. Applies well to large data and sparse classes.	1. Requires hyperparameters such as multiple iterations. 2. Data need to be shuffled before training.
XGBClassifier	Text Classification	Employs boosting technique that increases the accuracy (based on the strategy that multiple weak learner performs better than a individual strong learner).	Additional care required while tuning the different hyperparameters.

Table 3.5: Comparison of models used for classification of JIRA tasks

Chapter 4

Results and Discussions

In this section, we discuss the final results obtained after optimizing the model. Evaluation metrics used to analyse the performance of the model are described in detail. To analyse and obtain the best performing classifier it is important to select the right evaluation metrics during training. Generally, most of the classifiers use accuracy as an evaluation metric, but it has its limitations such as the ability to distinguish less, gather less information and more bias towards majority class [20]. These limitations found using accuracy have driven us to gather other evaluation metrics that will be discussed in section 3.5. Since our problem deals with multiple classes, additional focus is done on evaluation metrics that help in the evaluation of the classifier effectiveness.

4.1 Table of results

Table 4.1 describes the accuracy, f1-score, precision, recall and weighted accuracy for the five different models.

Model	Accuracy	F1-Score	Precision	Recall	Weighted accuracy
Multinomial NB	0.558057	0.48738	0.488987	0.558057	0.558291
Random Forest	0.505743	0.409342	0.394819	0.505743	0.506297
LinearSVC	0.635616	0.616557	0.639121	0.635616	0.634877
SGDClassifier	0.600936	0.593595	0.628427	0.600936	0.601662
XGBoost	0.560157	0.516469	0.526218	0.560157	0.561481

Table 4.1: Overall results for classifiers used for classifying tasks

As can be seen from the table above, the two best classifiers are LinearSVC 3.2.3 and SGDClassifier 3.2.4. LinearSVC model outperformed other models in terms of all the evaluation metrics. Answers to research questions are provided in this chapter. In one of the answers to the research questions, we interpret the results obtained from the different models.

	MultinomialNB	RF	LinearSVC	SGDClassifier	XGBoost
QUAEWATSICA	0.39	0.14	0.53	0.49	0.44
ETLANG	0.17	0.15	0.44	0.42	0.3
VELITONDRICKA	0.67	0.52	0.68	0.58	0.7
MOLLITIAWEHNER	0.77	0.77	0.84	0.81	0.77
EXLUEILWITZ	0.61	0.66	0.75	0.77	0.67
ODIODURGAN-BOEHM	0.5	0.42	0.6	0.57	0.41
REICIENDISMACGYVER	0.54	0.47	0.54	0.6	0.48
OPTIOHERMANN-RUTHE..	0.33	0.12	0.57	0.53	0.37
ODIORUECKER-WATSICA	0.46	0.3	0.33	0.25	0.29
NISIRUTHERFORD-TROMP	0.47	0.45	0.76	0.75	0.64
CORPORISVEUM-HEAT...	0.23	0.21	0.48	0.5	0.32
CONSEQUATURCASSIN-...	0.65	0.65	0.81	0.82	0.74

Table 4.2: Table showing f1-score results for each project based on the record folds in table 3.3

From the table 4.2, it can be seen that the LinearSVC model is able to achieve the highest f1-scores for each project, and the results for project *MOLLITIAWEHNER* is the best when compared to other projects. SGDClassifier is the second best performing model in terms of all the projects and marginally outperforms XGBoost model. It could also be interpreted that the project *MOLLITIAWEHNER* had almost equal f1-score results for all the different models. Results obtained for project *ETLANG* had average differences. Another finding here is that Random forest model trained with 1000 trees, is our least performing model according to the results with 12 % f1-score for project *OPTIOHERMANN-RUTHERFORD*. The variance in f1-results can be due to the insufficient data available for few projects 3.3. The results presented in the above table is provided from the optimization step generated by Optuna with 100 trials. We stop with 100 runs since the results were not seen to improve after 100 runs. It is noted that optuna is able to help with optimizing the required parameter setting with each runs. Table 4.1 is the average results of the project f1-scores found in table 4.2.

4.2 Answers to Research Questions

4.2.1 RQ1: What is the best baseline model to start for classification problems according to literature. ?

For this classification problem, we started building a Multinomial Naive Bayes Classifier since it is computationally less expensive and easier to implement. According to the research paper [24], Naive bayes works well with small datasets since we have 1924 records that are annotated and hence it is decided to train using the naive bayes model. As the adjective “Naive” states, it means the model assumes that the features are considered to be mutually independent i.e, the probability of one feature is not dependent on the presence of another feature as described in Section 3.2.1. Due to these considerations of independence features and the ability to make random guesses from small datasets, NB is trained to get a base accuracy from the dataset. The accuracy is the main

evaluation metric obtained from the NB classifier, this accuracy can then be compared with other models for interpreting the results. In this problem, a type of Naive Bayes known as multinomial NB is used due to the presence of text data having count and frequency. Multinomial NB is more suited for text classification problems.

4.2.2 RQ2: How good are the different models used for classifying tasks. ?

As can be seen from the results (Table 4.1) the SVM (Support Vector Machines) model types such as LinearSVC (kernel=linear) and SGDClassifier performed well in terms of all the evaluation metrics. The results are interpretable and support the research finding [24]. Experiments performed on research [24] also proves that types of SVM models perform well for text data since they deal with large dimensional space in a relatively small training set. In our case, LinearSVC which is a type of SVM using linear kernel performed best overall. From the results obtained we found that all the five different models performed on an average scale between 50 % - 65 %. Generally, accuracy will be used for evaluating performance measures, but research findings [26] critiques traditional ML measures and argues that f1-score is more concrete metric for evaluating models. Because f1-score gives equal importance to false positives and false negatives classes, this metric is well applicable to imbalanced class distribution problems. This research paper [26] also concludes LinearSVC is more preferred than other algorithms which proves more concrete evidence and support the obtained results for this work. Based on the comparison of f1-score measure for the different models obtained, XGboost falls third in terms of performance. Naive bayes performed marginally behind XGBoost in terms of all the metrics. And Random forest is our least performing model even with 1000 trees used as a parameter. The results obtained complement the findings of different research papers seen in this section. Another interesting characteristic in addition to the different metrics to be considered is the performance time as an evaluation measure. We performed 100 continuous runs to measure the training time taken by each model and captured the results. This is also an additional metric for our work to prove the superiority of one model over the other. Table 4.3 shows the run times in seconds for each model.

Trials	Random forest	Multinomial Naive Bayes	SGDClassifier	LinearSVC	XGBoost
10 runs	342	221	362	378	3002
20 runs	876	360	695	492	7192
50 runs	1120	452	823	632	15200
100 runs	2239	679	1350	933	28938

Table 4.3: Training times taken by models captured in seconds to measure performance

It can be seen that XGBoost is the slowest in terms of training and Naive bayes being the fastest. SVM is comparatively faster than other models. LinearSVC measures are marginally better than the SGDClassifier and Random forest model for the first 10 runs. And Random forest increases the time complexity for every 20 runs. The time complexity mentioned in table 3.5 in the Random forest is found to be true with this experiment being made. Time taken for training is also important since we plan to deploy this model as a daily script that will train the model

according to the setup planned as mentioned in chapter 5. Hence the model needs to run within a considerably less amount of time.

4.2.3 RQ3: What evaluation metric would be most suitable for distinguishing the optimal classifier ?

It is important to choose the right evaluation metric to find the optimal classifier. A summary of use cases for each evaluation metric can be found in section 3.5. We provide some context before we justify the usage of the f1-score as the best evaluation metric to be considered for classification problems. Precision and recall evaluates either positive or negative classes, but f1 score is the harmonic mean and considers both precision and recall together to get a mean score. The limitations of accuracy captured in Section 3.5 lead us to choose a metric that is not biased towards the majority class. We also strongly based our selection criteria using experimental results on research papers [24] [26]. The results obtained for our work are comparable to the results obtained in the research papers which also supports f1 scores for SVM proved to be higher than other classifiers.

Chapter 5

Conclusions

This work presents, implements and analyses the state-of-the-art techniques used in the field of machine learning to solve the problem of classifying JIRA low-level tasks to high-level tasks. It uses the text classification approach to categorize the tasks into categories named epics. We perform this work based on the CRISP-DM methodology starting from business understanding, data preparation, modeling, evaluation and deployment. A majority of the time for this work was spent on the data understanding that also helped to discover the knowledge of the business. It was important to understand the functional and technical details followed by the teams within the organization under the context of JIRA tool usage since this work is planned to help team members with an intelligent way to approach a mundane activity, which is associating the task to an epic in JIRA using machine learning.

Using the knowledge gained from the initial study, we have implemented five different machine learning models that are most suitable for this problem. The advantages and pitfalls of each technique used during modeling are discussed. At the end of the modeling phase, we evaluate the model to check the generalization capability. Results for the different models are interpreted and conclusions were made. Based on the output from different classification evaluation metrics and performance aspects, it is decided that the LinearSVC model performed the best with the existing dataset.

This tool named “Jira task classification” developed as part of the PROMESSA project can be found in the repository [1]. The source code consists of two separate modules such as machine learning module and software integration module. The machine learning module contains the text preprocessing, feature building and training scripts required for the modeling. The LinearSVC model is planned to be trained with a daily job scheduled to run every night to retrain the model with new incoming data from the JIRA tool. A software engineering module built using the python framework named “Flask” will expose a prediction API over HTTP to serve the prediction results with the trained model. This tool is bundled as a python “pip” package that will be hosted

[1] https://github.com/promessa-project/jira-task-classification.git

directly from a Github private repository to be integrated with the “PromessaAPI” source module developed by the core development team. This setup facilitates the business to access the “Jira task classification” module through “PromessaAPI”.

5.1 Research Contributions

- **Data preparation findings:** Most learning from this work is in the data preprocessing, text transformation and modeling steps. During the data processing step, we transformed the raw data into processed information that will be useful for the model using the different preprocessing techniques widely used in text classification problems. The text cleaning process takes into account the stop words, text lemmatization, etc. We understood that this is an important step for text classification problems, since text datasets deal with large noise.
- **Text transformation findings:** Experimenting different inputs to obtain TF-IDF vectors from the available corpus of words helped us find the suitable features for the model. This work also performs various trials with intelligent hyperparameters tuning strategies for each model to obtain the highest performance in all the models.
- **Model findings:** With all the above experiments performed in the three areas discussed, we contribute this knowlegde gained to be later applied in the context of real time projects and find usefulness from the user perspective.

5.2 Limitations

The limitations of this work are in the process of model creation. During the data preparation phase, it was found that tasks are more specific to a particular project rather than being shared within all the projects in the organization. Based on this specific knowledge of the data it was decided to train the models for each project. Due to this, it is only possible for any new task to have prediction within a project. The preprocessing steps works based on language specification. For this work, we worked with language as "English". However the trained model is capable of working with other language, it does not require any modifications. But the performance of the model without the cleaning steps for a different language can drastically reduce the performance. Hence if the model pipeline is required to work with other languages, it is suitable to configure the language in cleaning steps and then retrain the model with the same training setup. It is more logical in our case to have predictions for tasks within a project since teams may work closely within a project hence predictions obtained more specific to the project can be more meaningful. But it would be thoughtful for business to decide the approach to be taken. Hence we reserve this note as future work. Another limitation to the process lies in the data preparation phase. The data preprocessing and transformation techniques rely on a specific JIRA database model used within the organization and cannot be applied for a different database model. If the process is required to adapt for a different database structure, modifications in the data preparation stage.

However, the modeling process and optimization process does not need any modifications. The process followed in this work can only be applied inside organizations having a project structure with tasks and epics closely related to a specific project.

5.3 Future Work

- **Proposal 1:** Improvements can be made to the feature engineering steps to also consider n-grams (bigrams, trigrams) and involve additional text preprocessing methods to improve the performance of the model.
- **Proposal 2:** This module can be integrated with the Jira tool used by the different projects to test the usefulness from a business perspective.
- **Proposal 3:** To test the usefulness, a possibility can be to introduce a placeholder in the project dashboard used for sprint activities. For each task created by the user, prediction results from this "Jira task classification" module can be displayed and track the acceptance records from the user for each task. Insights can be gained from the feedback and improvements can be made to the existing model based on the user suggestion.

References

[1] Muhammad Abbas, Kamran Ali, Abdul Jamali, Kamran Ali Memon, and Abdul Aleem Jamali. Multinomial naive bayes classification model for sentiment analysis an efficient power splitting ratio based relay selection in cooperative relaying network under swipt framework view project classification for sentiment analysis view project multinomial naive bayes classification model for sentiment analysis. *IJCSNS International Journal of Computer Science and Network Security*, 19:62, 2019.

[2] Akiko Aizawa. An information-theoretic perspective of tf–idf measures. *Information Processing & Management*, 39(1):45–65, 2003.

[3] Himani Bansal, Gulshan Shrivastava, Nguyen Nhu, and Loredana STANCIU. *Social Network Analytics for Contemporary Business Organizations*. 03 2018.

[4] Daniel Berrar. Cross-validation, 1 2018.

[5] Gérard Biau and Gerard Biau@upmc Fr. Analysis of a random forests model, 2012.

[6] Christopher J C Burges and Chris J C Burges. Simplified support vector decision rules simpliied support vector decision rules, 1997.

[7] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *CoRR*, abs/1603.02754, 2016.

[8] Shovan Chowdhury and Marco P. Schoen. Research paper classification using supervised machine learning techniques. Institute of Electrical and Electronics Engineers Inc., 10 2020.

[9] Mike Cohn. *User Stories Applied: For Agile Software Development*. Addison Wesley Longman Publishing Co., Inc., USA, 2004.

[10] Ieee Copyright, P Abrahamsson, and J Koskela. Extreme programming: Empirical results from a controlled case study, acm, 2004.

[11] Shadi Diab. Optimizing stochastic gradient descent in text classification based on fine-tuning hyper-parameters approach. A case study on automatic classification of global terrorist attacks. *CoRR*, abs/1902.06542, 2019.

[12] Fabian Pedregosa FABIANPEDREGOSA, Vincent Michel, Olivier Grisel OLIVIERGRISEL, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Jake Vanderplas, David Cournapeau, Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Bertrand Thirion, Olivier Grisel, Vincent Dubourg, Alexandre Passos, Matthieu Brucher, Matthieu Perrot andÉdouardand, andÉdouard Duchesnay, and FRÉdouard Duchesnay EDOUARDDUCHESNAY. Scikit-learn: Machine learning in python gaël varoquaux bertrand thirion vincent dubourg alexandre passos pedregosa, varoquaux, gramfort et al. matthieu perrot, 2011.

[13] Tin Kam Ho. Random decision forests. In *Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1) - Volume 1*, ICDAR '95, page 278, USA, 1995. IEEE Computer Society.

[14] Thorsten Joachims. Text categorization with support vector machines. 1998.

[15] Matthias Jurisch, Stephan Böhm, and Toby James-Schulz. Applying machine learning for automatic user story categorization in mobile enterprises application development. *International Journal of Interactive Mobile Technologies (iJIM)*, 14:81, 09 2020.

[16] Sang-Woon Kim and Joon-Min Gil. Research paper classification systems based on tf-idf and lda schemes.

[17] Vladimer B. Kobayashi, Stefan T. Mol, Hannah A. Berkers, Gábor Kismihók, and Deanne N. Den Hartog. Text classification for organizational researchers: A tutorial. *Organizational Research Methods*, 21:766–799, 7 2018.

[18] Kamran Kowsari, Kiana Jafari Meimandi, Mojtaba Heidarysafa, Sanjana Mendu, Laura Barnes, and Donald Brown. Text classification algorithms: A survey. *Information (Switzerland)*, 10, 2019.

[19] Maria Lusky, Matthias Jurisch, Stephan Böhm, and Katharina Kahlcke. *Evaluating a User Story Based Recommendation System for Supporting Development Processes in Large Enterprises*.

[20] Hossin M and Sulaiman M.N. A review on evaluation metrics for data classification evaluations. *International Journal of Data Mining & Knowledge Management Process*, 5:01–11, 3 2015.

[21] Marco Ortu, Giuseppe Destefanis, Mohamad Kassab, and Michele Marchesi. Measuring and understanding the effectiveness of jira developers communities. volume 2015-August, pages 3–10. IEEE Computer Society, 8 2015.

[22] Heidar Pirzadeh, Andre oliveira, and Sara Shanian. Reuse: A recommendation system for implementing user stories. 08 2016.

[23] Erik R. Ranschaert, Sergey Morozov, and Paul R. Algra. *Artificial intelligence in medical imaging: Opportunities, applications and risks*. Springer International Publishing, 1 2019.

[24] Martin Riekert, Matthias Riekert, and Achim Klein. Simple baseline machine learning text classifiers for small datasets. *SN Computer Science*, 2, 5 2021.

[25] Xavier Schmitt, Sylvain Kubler, Jeremy Robert, Mike Papadakis, and Yves Letraon. A replicable comparison study of ner software: Stanfordnlp, nltk, opennlp, spacy, gate. pages 338–343. Institute of Electrical and Electronics Engineers Inc., 10 2019.

[26] Marina Sokolova, Nathalie Japkowicz, and Stan Szpakowicz. Beyond accuracy, f-score and roc: A family of discriminant measures for performance evaluation. volume WS-06-06, pages 24–29, 2006.

[27] M. Thangaraj and M. Sivakami. Text classification techniques: A literature review. *Interdisciplinary Journal of Information, Knowledge, and Management*, 13:117–135, 2018.

[28] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, Berlin, Heidelberg, 1995.

[29] R. Wirth and J. Hipp. Crisp-dm: towards a standard process modell for data mining. 2000.