AUTOMATED ASSIGNMENT AND CLASSIFICATION OF SOFTWARE ISSUES

by

Büşra Tabak

B.S., Computer Engineering, Izmir University of Economics, 2019

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2023

ACKNOWLEDGEMENTS

First and foremost, I extend my heartfelt gratitude to my supervisor, Assist. Prof. Fatma Başak Aydemir, for not only accepting me into the program but also for her exceptional guidance, expertise, and unwavering support throughout every step of this research journey. Her profound insights and constructive feedback have played a pivotal role in shaping the direction and elevating the quality of this work. I am deeply appreciative of her mentorship and unwavering commitment to my academic and personal growth, which have been invaluable assets on this path to success.

I extend my heartfelt appreciation to the members of my thesis committee, Assoc. Prof. Arzucan Özgür and Assist. Prof. Gözde Gül Şahin for their time and reviews.

Moreover, I would like to express my gratitude to my former directors at Vestek for their unwavering support throughout the semester. They consistently encouraged me to balance my academic commitments and allowed me the flexibility to attend lectures. Their provision of valuable data for this research and their insightful feedback on my work have been instrumental in its development. I am sincerely thankful for their ongoing support and belief in my abilities.

ABSTRACT

AUTOMATED ASSIGNMENT AND CLASSIFICATION OF SOFTWARE ISSUES

Software issues contain units of work to fix, improve or create new threads during the development and facilitate communication among the team members. Assigning an issue to the most relevant team member and determining a category of an issue is a tedious and challenging task. Wrong classifications cause delays and rework in the project and trouble among the team members. This thesis proposes a set of carefully curated linguistic features for shallow machine learning methods and compares the performance of shallow and ensemble methods with deep language models. Unlike the state-of-the-art, we assign issues to four roles (designer, developer, tester, and leader) rather than to specific individuals or teams to contribute to the generality of our solution. We also consider the level of experience of the developers to reflect the industrial practices in our solution formulation. We employ a classification approach to categorize issues into distinct classes, namely bug, new feature, improvement, and other. Additionally, we endeavor to further classify bugs based on the specific type of modification required. We collect and annotate five industrial data sets from one of the top three global television producers to evaluate our proposal and compare it with deep language models. Our data sets contain 5324 issues in total. We show that an ensemble classifier of shallow techniques achieves 0.92 for issue assignment and 0.90 for issue classification in accuracy which is statistically comparable to the state-of-the-art deep language models. The contributions include the public sharing of five annotated industrial issue data sets, the development of a clear and comprehensive feature set, the introduction of a novel label set and the validation of the efficacy of an ensemble classifier of shallow machine learning techniques.

ÖZET

YAZILIM SORUNLARININ OTOMATİK ATANMA VE SINIFLANDIRILMASI

Yazılım sorunları, geliştirme sırasında düzeltmek, iyileştirmek veya yeni iş parçacıkları oluşturmak ve ekip üyeleri arasındaki iletişimi kolaylaştırmak için iş birimleri içerir. Bir sorunu en alakalı ekip üyesine atamak ve bir sorunun kategorisini belirlemek sıkıcı ve zorlu bir iştir. Yanlış sınıflandırmalar, projede gecikmelere, yeniden çalışmalara ve ekip üyeleri arasında sıkıntıya neden olur. Bu makale, sığ makine öğrenimi yöntemleri için özenle derlenmiş bir dizi dilsel özellik önermekte ve sığ ve topluluk yöntemlerinin performansını derin dil modelleriyle karşılaştırmaktadır. Son teknolojiden farklı olarak, çözümümüzün genelliğine katkıda bulunmak için sorunları belirli kişiler veya ekipler yerine dört role (tasarımcı, geliştirici, testçi ve lider) atıyoruz. Çözüm formülasyonumuzdaki endüstriyel uygulamaları yansıtmak için geliştiricilerin deneyim düzeyini de dikkate alıyoruz. Sorunları, hata, yeni özellik, iyileştirme ve diğer gibi farklı sınıflara ayırmak için bir sınıflandırma yaklaşımı kullanıyoruz. Ek olarak, gereken özel değişiklik türüne göre hataları daha fazla sınıflandırmaya çalışıyoruz. Tasarımızı değerlendirmek ve derin dil modelleriyle karşılaştırmak için en büyük üç küresel televizyon üreticisinden birinden beş endüstriyel veri seti topluyor ve bunları etiketliyoruz. Veri setlerimiz toplamda 5324 sorun içermektedir. Sığ tekniklerin bir topluluk sınıflandırıcısının, en gelişmiş derin dil modelleriyle istatistiksel olarak karşılaştırılabilir olan hatırlama ve doğrulukta sorun ataması için 0.92 ve sorun sınıflandırması için 0.90'a ulaştığını gösteriyoruz. Katkılarımız, beş adet etiketlenmiş endüstriyel sorun veri setinin kamuya açık paylaşımını, açık ve kapsamlı bir özellik setinin geliştirilmesini, yeni bir etiket setinin tanıtılmasını ve sığ makine öğrenme teknikleriyle oluşturulan bir topluluk sınıflandırıcının etkinliğinin doğrulanmasını içermektedir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

f	Variable being calculated
i	Index variable used in the summation
j	Index variable used in the nested summation
p_i^j	Value of p raised to the power of j for the i-th term
s_i	Value of s for the i-th term
$\sum$	Summation, adding values within a specified range
$\cup$	Union, the combination of multiple elements
$\vee$	Logical OR, the disjunction of two or more conditions

LIST OF ACRONYMS/ABBREVIATIONS

BC	Bug Classification
BOW	Bag-of-Words
B2B	Business-to-Business
DA	Developer Assignment
DT	Decision Tree
FE	Feature Extraction
FJB	Feature-Jira-Boolean
FJC	Feature-Jira-Categorical
FJN	Feature-Jira-Numerical
FTB	Feature-Text-Boolean
FTC	Feature-Text-Categorical
FTN	Feature-Text-Numerical
FTNP	Feature-Text-Numerical-POS
IA	Issue Assignment
IC	Issue Classification
ICE	Issue Classification - English
ICT	Issue Classification - Turkish
ITS	Issue Tracking Systems
kNN	k-Nearest Neighbors
LR	Logistic Regression
LSTM	Long Short Term Memory
ML	Machine Learning
MLP	Multilayer Perceptron
NB	Naive Bayes
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
OSS	Open Source Software
POS	Part-of-Speech

PROP	Proprietary Software
RF	Random Forest
RNN	Recurrent Neural Networks
SES	Software Engineering Student
SGD	Stochastic Gradient Descent
SSE	Senior Software Engineer
SVM	Support Vector Machine
TA	Team Assignment
TF-IDF	Term Frequency-Inverse Document Frequency
UI	User Interface
UX	User Experience
WE	Word Embedding

1. INTRODUCTION

Software project development refers to the process of creating a software product from start to finish, including planning, designing, coding, testing, and maintenance. It involves a team of developers, often with different specializations, working together to produce a working software product. Software project management involves overseeing the development process, ensuring that the project is completed on time, within budget, and to the expected quality standards [1]. This includes managing resources, schedules, and budgets, as well as communicating with stakeholders and ensuring that the project meets its objectives. Effective project management is necessary for successful software development.

One of the primary responsibilities of a project manager is to identify and address software issues as they arise throughout the development process [2]. These issues can include technical challenges, quality assurance problems, or unexpected delays. The project manager must work with the development team to find solutions to these issues, prioritize tasks, and make adjustments to the project plan as needed. By effectively managing software issues, the project manager can help ensure that the development process stays on track, the software product is delivered on time and to the expected quality standards, and that the project stays within budget.

Issue Tracking Systems (ITS) are designed to help development teams track and manage software issues throughout the development process. These systems allow developers to identify, report, and prioritize software issues and assign them to team members for resolution [3]. Issue tracking systems often include features such as issue tracking, bug reporting, status tracking, and reporting tools, enabling developers to manage issues effectively and ensure that they are resolved in a timely manner. Issues can be created by users with different roles such as software developers, team leaders, testers, or even customer support teams in these tools. Bertram et al. [1] carry out a qualitative study of ITS as used by small software development teams. They demon-

strate that ITS plays a key critical role in the communication and collaboration within software development teams based on their interviews with various stakeholders.

Text classification is an important problem that is the task of assigning a label to a given text [4]. Text classification has started to be used as a tool to produce solutions in many studies in various fields due to the abundance and diversity of data known as big data. Issue classification is the separation of issues into distinct facets that classify various aspects, resources, and characteristics of each issue. It entails analyzing issues to decide their existence and form, as well as identifying common facets that can be used to construct issue groups or divisions. This thesis tackles the problem of classifying issues as a multi-class classification problem [5].

The main focus of this thesis is to address the issue classification problem through two distinct approaches. The first approach involves issue category classification, where we aim to categorize the issues into specific groups or classes. The second approach is issue assignment, where we assign the identified issues to appropriate team members or departments for further resolution. To accomplish this, we treat the problem as a text classification challenge. We leverage machine learning algorithms and natural language processing techniques to analyze and classify the text data of the issues. By applying these techniques, we are able to extract relevant information from the issue descriptions, such as the issue severity, context, and other important details. Overall, by tackling the issue classification problem through these two distinct approaches, we aim to provide a more comprehensive and effective solution for issue management and resolution.

Through the issue category classification approach, we are able to identify the type of issue. The categories include Bug – CodeBased and TextBased, NewFeature, Improvement, and Other. This allows us to better understand the nature of the issues and prioritize them based on their impact and urgency. Wrong types of issues can have various negative consequences, depending on how the issue tracking system is used [6]. If all issues must be fixed before the next release, a feature request or improvement

that is mistakenly categorized as a bug can cause the project to be delayed. In the exact opposite situation, it could lead to the project's bugs being missed, resulting in dissatisfaction and problems in the following release. Developers might also be harmed by incorrect classification. For example, a developer working on an issue that is opened by a tester with the wrong issue type and without enough information may focus on adding new features instead of fixing existing bugs. This results in a waste of time and effort. The testers may reopen the issue that the developer declared as resolved, which may have a negative impact on the developer's performance score. Antoniol et al. [7] investigate a substantial number of issues and found that more than half of issues marked as bug might not actually refer to bug. Herzig et al. [8] also state that "every third bug is not a bug".

On the other hand, the issue assignment approach enables us to allocate the issues to the most suitable team members or departments. This helps to streamline the resolution process and ensure that the issues are addressed by the right people, thereby improving the overall efficiency and effectiveness of the support system. We decide that assigning issues to groups of employees who can perform the same activities is preferable to the individuals. Because some employees in the issue history may not have been able to complete the task that is automatically assigned to them in that planning time due to a variety of factors, including seasonal spikes in workload, illness, or employee turnover [9]. To effectively manage the employees in our data set, we have grouped them based on the fields they work in. This approach has resulted in the identification of four main teams in the data set, namely the software developer, UI/UX designer, software tester, and team leader. The software developer team represents the majority of the data set, making them a crucial focus of our analysis. To improve time management and issue resolution, it is important to assign the right issues to the right developers. To achieve this, we have categorized the Software Developers using sub-labels that are generally accepted in the industry, such as senior, mid, and junior software developer levels. This categorization helps us identify the experience level and skill set of each developer, allowing us to allocate the most appropriate tasks to each team member. These teams may differ according to the project or the company. For

example, new teams such as business analysts, and product owners can be added or some teams can be split or removed. At this point, we expect the side that will use the system to separate the individuals according to the teams. After a newly opened issue is automatically classified among these classes, it can be randomly assigned to the individuals in the relevant team, or the individuals in the team or the team leader can make this assignment manually.

In our study, we use a closed-source data set for our analysis contrary to the majority of studies in the literature. We obtain five projects from the company's Jira interface for analysis. For our first study, a software engineering student and the main senior software developer of the project classify the issues according to their categories and translate them into English. We conduct our issue category classification study for both English and Turkish using a single project. Correctly classifying issues according to their categories is a crucial step in issue classification, as it allows for more accurate and effective machine learning models. However, this task requires knowledge about the project and can be time-consuming. By focusing on a single project, we are able to devote more time and attention to each issue and develop a deep understanding of its category. This approach allow us to create a high-quality data set that accurately reflected the categories and characteristics of the issues in the project. For our second study, we use all five projects and focus exclusively on the main language of the issues. To prepare the data set for this study, we determine the label values by changing the people assigned to the issue according to the fields they work in, based on information we receive from the company.

ITS often contain a wealth of valuable data related to software issues. In our study, we set out to analyze this data using NLP methods, with the goal of creating a feature set that would be simpler and more successful than the word embedding methods typically used in text classification. To create our feature set, we use a range of NLP techniques to analyze the language used in software issues like part-of-speech tagging and sentiment analysis. We then compare our feature set with commonly used word embedding methods and apply a range of machine learning, ensemble learning,

and deep-learning techniques to our annotated data set. This allows us to evaluate the efficiency of our approach using a range of standard metrics, including accuracy, precision, recall, and F1-score.

We have made several significant contributions to the state of the art in issue classification.

- Data set: We provide a closed-source issue data set from the industry that is manually annotated by its main developer according to the category and assignee in both Turkish and English. This data set is publicly available for further research, and to the best of our knowledge, there is no shared commercial issue data set for both languages in the literature.
- Feature set: We develop an understandable feature set that is extracted from the information in the issues, which can be applied to all issue classification procedures with high accuracy and low complexity.
- Label set: We introduce novel labels for issue assignment and issue classification. By incorporating these new labels, we expand the boundaries of current research and offer unique insights into the underlying themes, contributing to a more comprehensive understanding of the domain.

The remainder of this thesis is structured as follows: Chapter 2 describes the background of this study including the structure of software issues in issue tracking systems, natural language processing tasks, and automated text classification methods. In Chapter 3, we present our experimental setup and approach, followed by our results and analysis in Chapter 4. In Chapter 5, we discuss the threats to validity and user evaluation. In Chapter 6, we discuss related work and similar classification endeavors with Turkish issue reports. Chapter 7 concludes our work and discusses future work.

2. BACKGROUND

To understand our approach, it is essential to have a solid understanding of the structure of software issues in issue tracking systems (ITS), as well as the fundamentals of natural language processing (NLP) tasks and automated text classification processes.

2.1. Software Issues in ITS

This section explores software issues within ITS, covering their anatomy, life cycle, and significance in software projects.

2.1.1. Issue Tracking Systems (ITS)

An issue tracking system (ITS) is a software application that manages and maintains lists of issues. It provides a database of issue reports for a software project. Members of the software development team stay aligned and collaborate faster with the help of these tools. Users can log in with a password and post a new issue report or comment on an already-posted issue. There are various issue-tracking tools that are used in software development projects, such as JIRA, Bugzilla, and Azure DevOps. Although there are some differences among these tools, they all share a similar basic structure. We developed our approach with a data set of a company that uses JIRA software.

2.1.2. Anatomy of an Issue

The issue report in ITS contains various fields, each of which contains a different piece of information. The following details [10] could typically be included in the generic process of reporting an issue:

- Summary: Title of the issue.
- Description: Issue details including its cause, location, and timing.
- Version: Version of the project to which the issue belongs.
- Component: The predetermined component on which the issue depends.
- Attachment: Attaches such as pictures, videos, and documents uploaded to the issue.
- Priority: Urgency level of the issue.
- Severity: Frequent occurrence and systemic effects.
- Status: Current status of the issue.
- Reporter: Person who creates the issue.
- Assignee: Person to whom the issue is assigned.
- Revision History: Historical changes of the issue.
- Estimated time: Estimated time spent to develop or solve the issue.
- Comments: Additional details that can be used to understand the issue.

The summary, description, and comments all contain textual details about the issue. In our methodology, we extract numerical features using the language structure of summary and description. Version is the current version of the project which differs with each release. The issue can have a predefined tag or component added to it. (e.g. project_v1.1.0) Users can upload files to help others understand the issue, an attachment refers to it. Priority, severity, and status are the categorical features of the issue. Priority is the urgency level, severity is the frequency of occurrence and status is the current status of the issue such as committed or done. People play different roles as they interact with reports in ITS. The person who submits the report is the reporter and the assignee is the person to whom the issue is assigned. If the issue has been reported previously, historical changes are shown in the revision history. The estimated time is the time spent on the development and varies depending on the effort metric. The fields offered by each ITS vary, and not all fields are filled out for each project. Our strategy makes use of the fields that are largely filled in.

2.1.3. Life Cycle of an Issue

The series of phases and phase changes that an issue experiences throughout its life cycle is referred to as a "workflow." Workflows for issues typically represent development cycles and business processes. Figure 2.1 shows a standard workflow of JIRA. The following stages of the JIRA workflow must be monitored as soon as an issue is created:

- Open: The issue is open after creation and can be assigned to the assignee to begin working on it.
- In Progress: The assignee has taken the initiative to begin working on the issue.
- Resolved: The issue's sub-tasks and works have all been finished. The reporter is currently waiting to confirm the matter. If verification is successful, the case will be closed or reopened depending on whether any additional changes are needed.
- Reopened: Although this issue has already been solved, the solution is either flawed, omitted some important details, or needed some modifications. Issues are classified as assigned or resolved at the Reopened stage.
- Closed: The issue is now regarded as resolved, and the current solution is accurate. Issues that have been closed can be reopened at a later time as needed.

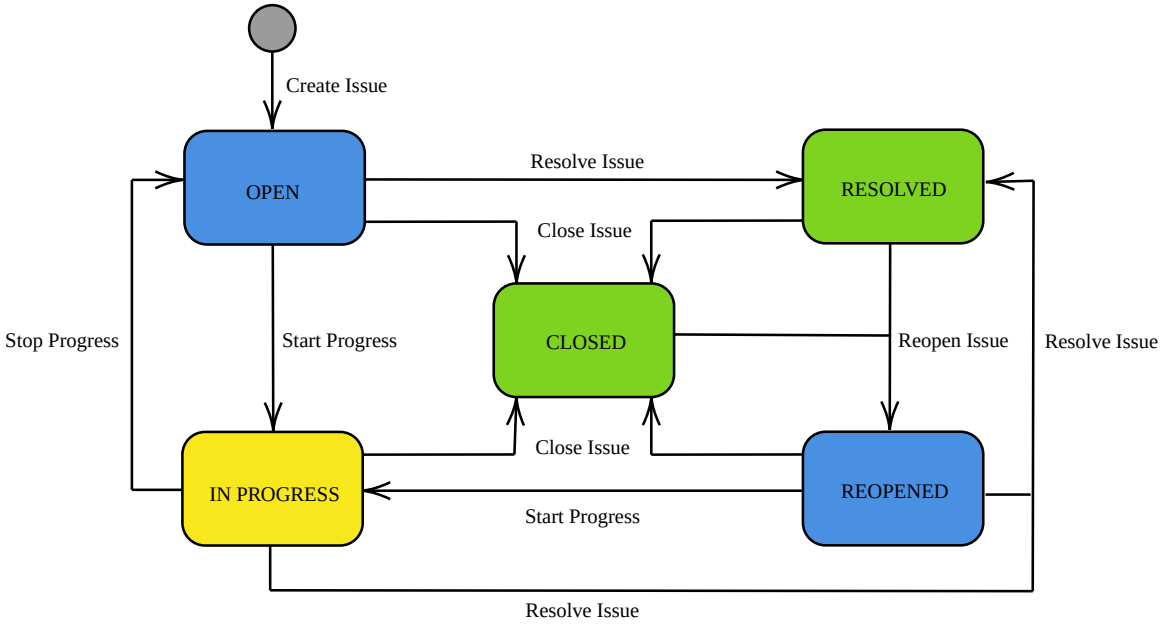

Figure 2.1. Jira workflow.

2.2. Natural Language Processing Tasks

The field of NLP encompasses various tasks aimed at understanding and processing human language using computational techniques. In this section, we focus on two fundamental NLP tasks applied in this thesis: lemmatization and part-of-speech (POS) tagging.

2.2.1. Lemmatization

Lemmatization is a crucial NLP technique that aims to transform words to their base or dictionary form, known as the lemma. Unlike stemming, which removes affixes to obtain word roots, lemmatization ensures that the resulting lemma is a valid word with proper meaning. This process is particularly important in tasks where understanding the grammatical properties of words is crucial for accurate analysis. When comparing Turkish and English lemmatization, we encounter notable differences due to the linguistic characteristics of each language. In Turkish, a highly agglutinative

language, words can contain multiple suffixes that carry grammatical information [11]. Lemmatizing Turkish text requires a deep understanding of its morphological rules. For example, the verb 'okuyorum' (I am reading) can be lemmatized to 'oku' (read), where the suffix '-yor' indicates the present continuous tense and '-um' indicates first-person singular. In contrast, English has a simpler inflectional system, and lemmatization is relatively straightforward. For instance, the verb 'running' can be lemmatized to 'run,' preserving its base form. While both Turkish and English benefit from lemmatization in NLP tasks, the linguistic complexity of Turkish necessitates more intricate lemmatization algorithms and resources tailored to its specific morphological rules.

2.2.2. Part-of-speech (POS) tagging

Part-of-speech (POS) tagging is a fundamental task in NLP that involves assigning grammatical tags to individual words in a given text. These tags represent the syntactic category or grammatical function of each word, such as noun, verb, adjective, or adverb, enabling deeper linguistic analysis. POS tagging is crucial for various NLP applications, including text analysis, information retrieval, and machine translation. However, POS tagging can present unique challenges when applied to different languages. In the case of Turkish and English, notable differences arise due to their distinct grammatical structures. Turkish is an agglutinative language, where words are formed by adding affixes to a base, resulting in complex word forms [11]. This complexity poses challenges for POS tagging, as the tagger needs to accurately identify and assign the appropriate POS tags to each morpheme. On the other hand, English, an inflectional language, relies more on word order and function words for grammatical cues. Therefore, the POS tagging task for English texts tends to focus on disambiguation between different parts of speech based on contextual clues. Understanding these linguistic nuances and the specific challenges posed by Turkish and English in the POS tagging process is essential for developing effective POS taggers and interpreting the results of the subsequent analyses conducted in this thesis.

2.3. Automated Text Classification

Automated text classification, also known as text categorization or document classification, involves automatically assigning predefined categories or labels to text documents based on their content. The goal of automated text classification is to develop models or algorithms that can accurately classify large volumes of text data efficiently and effectively.

The process of automated text classification typically involves the following steps:

- Data Preparation: The text data is collected and preprocessed to remove noise, and irrelevant information, and standardize the text. This step may include tasks such as tokenization, removing stopwords, and handling special characters or punctuation.
- Feature Extraction: Relevant features are extracted from the text documents to represent their content in a numerical or vectorized format. Feature extraction techniques used in this study include Bag-of-Words (BOW), Term Frequency-Inverse Document Frequency (TF-IDF), Word2Vec, and manual approach. TF-IDF is a numerical statistic that reflects the importance of a word in a document within a collection or corpus. It calculates the product of two factors: term frequency (TF), which measures the frequency of a term in a document, and inverse document frequency (IDF), which measures the rarity of the term across the entire corpus. BoW, on the other hand, represents text as a "bag" or collection of words without considering their order or grammar. It creates a vector representation of a document by counting the occurrences of words in the document and ignoring their sequence. Each word becomes a feature, and the feature vector represents the presence or absence of words in the document. The goal of Word2Vec is to identify words' semantic and contextual meanings based on how frequently they appear in a corpus. It offers vector representations that are useful for word arithmetic (e.g., "king" - "man" + "woman" = "queen"), comparing the similarity of various words, and even identifying word connections. Manual feature

extraction involves the manual identification and extraction of specific linguistic or domain-specific features from the text. These features can be syntactic, semantic, or domain-specific characteristics that are relevant to the task at hand. Manual feature extraction requires expertise and domain knowledge to identify and select the most informative features for a particular problem.

- Training Data Creation: A labeled training data set is created, where each text document is associated with a predefined category or label. This data set is used to train the text classification model.
- Model Training: In our study, we employ a diverse set of classifiers for our task, grouped into different categories. Traditional machine learning models include Support Vector Machine (SVM), Logistic Regression (LR), Naive Bayes (NB), Multilayer Perceptron (MLP), Stochastic Gradient Descent (SGD), Decision Tree (DT), Random Forest (RF), and K-Nearest Neighbors (KNN). These models are relatively simpler and have fewer parameters to tune. Linear models like LR and SGD are computationally efficient and work well for linearly separable data. Tree-based models like DT and RF capture non-linear relationships and are robust to outliers. Probabilistic models like NB use probability theory to make predictions. They assume feature independence given the class and calculate the probability of each class based on the input features. Neural networks like MLP can learn complex patterns but require more data and computational resources. SVM finds optimal decision boundaries in high-dimensional feature spaces. Ensemble methods include One vs Rest, Voting, RF with Boosting, Bagged DT, Extra Trees, and Stacking. These methods combine multiple base classifiers to improve overall performance. Voting combines the predictions of multiple models using a majority vote, while One vs Rest constructs multiple binary classifiers to handle multi-class classification. RF with Boosting, Bagged DT, and Extra Trees are ensemble models that use variations of decision trees to create a diverse set of classifiers. Deep learning models include DistilBert, Roberta, and Electra. They are state-of-the-art deep-learning models pre-trained on large-scale corpora. They utilize transformer architectures to capture contextual information and have achieved impressive results in various classification tasks.

- Model Evaluation: The trained model is evaluated using separate test data to measure its performance and accuracy. Evaluation metrics such as precision, recall, F1-score, or accuracy are used to assess the model's performance. Precision measures the correctness of positive predictions. Recall measures the ability to capture actual positive instances. F1-Score balances precision and recall in a single metric. Finally, accuracy measures the overall correctness of predictions.
- Model Deployment: Once the model has been trained and evaluated, it can be deployed to classify new, unseen text documents automatically. The deployed model takes the text as input and assigns the most appropriate category or label based on its learned patterns and features.

Automated text classification has a wide range of applications, including spam detection, sentiment analysis, topic classification, news categorization, customer feedback analysis, and content filtering. It enables efficient handling and organization of large volumes of textual data by automatically categorizing documents, improving information retrieval, and supporting decision-making processes.

3. APPROACH

This chapter outlines the methodology employed in this study to address the research objectives. It is divided into several sections, each focusing on a crucial step in the process. The chapter begins with an overview of the data collection, which details the sources and methods used to gather the necessary data for analysis. This is followed by the manual issue annotation, where a team of experts systematically categorizes and labels the collected issues based on predefined criteria. The preprocessing describes the steps taken to clean and transform the raw data into a format suitable for further analysis. Next, feature extraction explains the techniques used to extract relevant features from the preprocessed data, capturing essential information for the subsequent classification. Finally, the classification discusses the algorithms and models employed to classify the issues based on their assigned labels.

3.1. Data Collection

Our raw data come from the issues of five industrial projects documented on Jira software of a company that offers solutions in the fields of business-to-business (B2B) display systems for televisions, namely, hotel TV systems and advertising and promotional display systems. The company is a home and professional appliance manufacturer with 18 subsidiaries specializing in electronics, large appliances, and information technology and it is Europe's leading television manufacturer, accounting for a quarter of the European market with over eight million units sold in a year.

Table 3.1. Data sets used in the experiments.

ID	Name	# Issues	Timespan	Team Size
P1	Ip Hotel Tv	1287	2011-2021	35
P2	Rf Hotel Tv	2004	2017-2021	15
P3	Hospital Tv	202	2017-2021	28
P4	Vsync	126	2017-2021	7
P5	Html Hotel Tv	1705	2018-2021	16

Table 3.1 summarizes the raw data. We use issue reports from five web application projects, all of which are two-sided apps with a management panel and a TV-accessible interface. The mission of the Ip Hotel Tv project is a browser-based interactive hospitality application used by hotel guests in their rooms. There is also a management application for managing the content that will be displayed. This is the company's first hospitality project, which began in 2011 and is still in use today by many hotels. The project Hospital Tv is a hospital-specific version of the Ip Hotel Tv application. It is compatible with the Hospital Information Management System (HIMS), which is an integrated information system for managing all aspects of a hospital's operations, including medical, financial, administrative, legal, and compliance. The Rf Hotel Tv project is a version of the Ip Hotel Tv application that can be used in non-intranet environments. A coax cable is used for communication with the server. The Html Hotel Tv project is a cutting-edge hospitality platform. It will eventually replace the Ip Hotel Tv project. Instead of using an intranet, this version is a cloud application that works over the Internet. A central system oversees the management of all customers. Customers now have access to new features such as menu design and theme creation. The project Vsync is a signage application that synchronizes the media content played by televisions. Televisions play the media through their own players.

These projects are developed in different programming languages. The project Rf Hotel Tv is written in Python using the Django framework while the project Vsync is written in C# and JavaScript using the Angular framework. The rest of the projects

are written in pure Javascript and C# using Microsoft technologies such as .Net and .Net Core frameworks.

The number of issues per project ranges from 126 to 2004, and the total number of issues is 5324. The data set contains issue reports submitted between 2011 and 2021, all related to different versions of the applications. The issues are created by users with different roles such as software developers, team leaders, testers, or even customer support teams in the data. Then, they are assigned to workers with different roles and experiences. The number of employees in the projects varies between seven and 35 when the "Creator", "Reporter", and "Assignee" columns in the data set are combined.

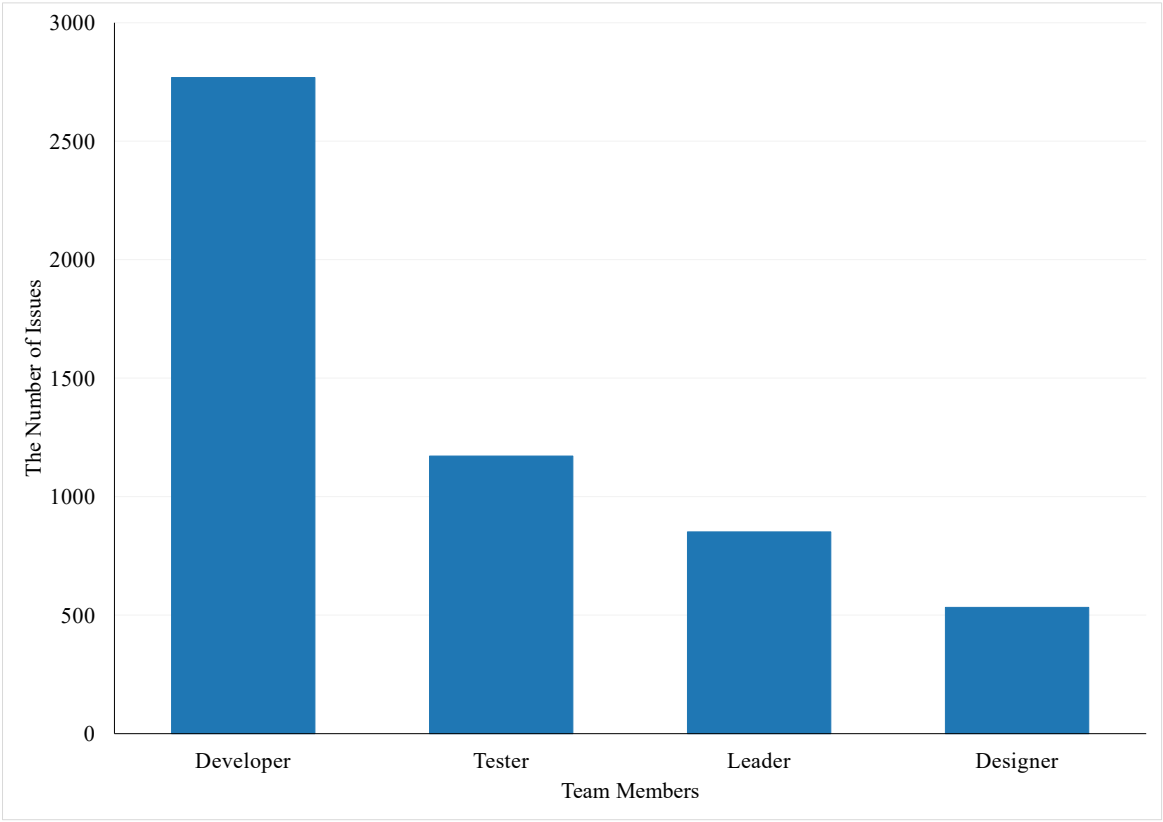

Figure 3.1. Team members role distribution.

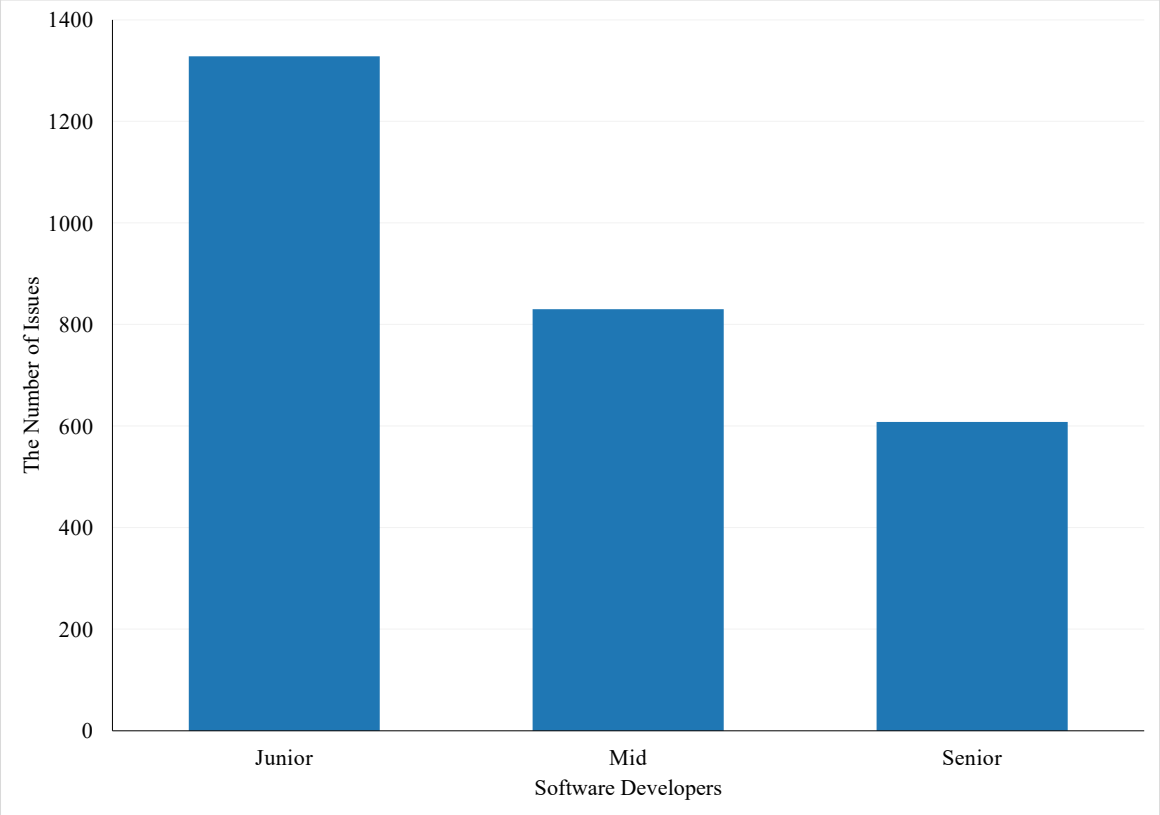

Figure 3.2. Software developers experience distribution.

In the original data, the issues are assigned to the individual employees. We removed the names of the individuals to preserve their privacy and inserted their roles in the development team as a new column with the possible values "Software Developer", "UI/UX Designer", "Test Engineer", and "Team Leader". For the developers only, we also assigned another column indicating their level of expertise as Junior, Mid, and Senior. We inserted this new information in collaboration with the company. Figure 3.1 depicts the distribution of the assignees over the entire data set. As the first chart shows we can observe that team leaders receive the least number of issues and software engineers receive the majority of them. According to Figure 3.2, the distribution of experience among software developers, the issues are primarily assigned to junior-level developers, at most, and senior-level developers, at least.

Table 3.2. Part of issue from our data set.

Project	**Summary**	**Description**
P1	Orders placed on the room service page do not send e-mails	Although the success message is displayed on the screen, the order e-mail does not come.
P2	Server v1.0.9 Test Request	Please test it.
P3	Making a mother-baby record distinction	The mother's data should come into the room, since it is the same protocol number as the baby, it should be separated.
P4	Multiple video wall synchronization support	Multiple video wall setup should be added to the system and it should be synchronized independently.
P5	Version Filter MacId Problem	Problem experienced due to different incoming Mac address in wifi and ethernet connections.

We directly export the data from the company's Jira platform in Excel format, including all columns. Table 3.2 is a small portion of the massive amount of the data available. Although most columns are empty for each row, the tables have a total of 229 columns. To create the issue, required fields like "Summary," "Description" and "Assignee" are filled, but fields like "Prospect Effort" and "Customer Approval Target Date" are left blank because they aren't used in the project management.

The issues are originally written in Turkish with some English terms for technical details and jargon. We also translate the issues to English and share our data publicly [12] in our repository.

3.2. Manual Issue Annotation

The manual issue classification approach relies solely on the judgment of human classifiers; the ground truth labels to compare against are not available. The classification is performed independently by software engineering student (SES) and senior software engineer (SSE), no automated tool is used. SES is an undergraduate software engineering student who assists us with our study by classifying and translating data set into English. SSE is a senior software developer having five years of experience in software development and two years of experience as a main developer of our case study project.

We provide participants with general details about the project and labeling in accordance with Table 3.3 so they can make informed decisions. This information includes the project's goal, a description, and a demonstration of the application to get familiar with the project. In this study, we consider five types of labels, namely BugCodeBased, BugTextBased, NewFeature, Improvement, and Other. If an issue requires modifying the source code on a code-based basis, it is categorized as Bug-Code Based. For instance, there is a code block in the project that behaves improperly, problematically or fails to function as it should. Issues that demand text-based changes to the source code are labeled as Bug-Text Based. This class should be chosen if the application needs to change the color, font, size, translation, or capitalization of a letter. If the issue is a New Feature request, new development of the source code is required. This includes adding integration, building a structure, adding a new page, etc. It is an Improvement request if it calls for changes to the currently usable portion of the source code. For example, adding a scroll to a text field, closing an opened bar with a timeout, etc. are examples of developing or editing existing features. Finally, if the issue does not require modifying the source code, such as during testing, project installation, or work involving documentation, their label is Other. We analyze our proprietary data set to create our labels, and unlike other studies, we split the bug label into two sub-labels depending on whether it is a text-based or a code-based bug. Table 3.4 shows one issue for each label in the data set with the summary and description sections.

Table 3.3. Description of our labels.

Label	**Description**	**Example**
BugCodeBased	Necessitate code-based modifications to the source code	An error occurs in the project, a code block that works incorrectly/problematically or does not work when it should work, etc.
BugTextBased	Necessitate text-based modifications to the source code	The necessity of changing color, icon, font, size, translation, capitalizing the letter in the application, etc.
NewFeature	Necessitate new development to the snippet of the source code	Creating a new page, adding integration, building a structure, etc.
Improvement	Necessitate improvement to the available snippet of the source code	Developing, and editing existing features such as adding a scroll to the text field, closing the opened bar with a timeout, etc.
Other	Do not necessitate a modification to source code	Testing, installation of the project, works related to documentation, etc.

Table 3.4. The example of labeled issues from our data set.

Label	**Issue Summary**	**Issue Description**
BugCodeBased	Recipient e-mail address group does not exist.	Only one e-mail address can be entered in the e-mail recipient and e-mail cc options in SMTP settings. More than one e-mail address should be entered in these sections.
BugTextBased	Added channel logos are half displayed in the admin panel.	Long channel logos added to the admin panel are half visible. Scaling should be done. The sample is attached.
NewFeature	Information on the log page should be able to be exported.	Information on the log page should be able to be exported.
Improvement	Room number, IP, and MAC addresses should be displayed on the TV management page.	Only MAC addresses are displayed on the TV management page, in addition, room number, IP, and MAC addresses should be displayed in the list.
Other	TV setup.	TV setup should be done.

The degree of agreement among diverse observers who label, code, or rate the same phenomenon is known as inter-rater reliability [13]. There are several statistics that can be used to assess inter-rater reliability, from which the most common are Cohen's Kappa [14] or Krippendorf's Alpha [15] for two annotators. We perform the Cohen's Kappa test. K values range from 0 to 1, with higher values indicating greater and lower values indicating less agreement between raters. We use the guideline provided by Landis and Koch [16] for interpreting kappa values is as follows:

- <0 as poor,
- 0.01–0.20 as slight,
- 0.21–0.40 as fair,
- 0.41–0.60 as moderate,
- 0.61–0.80 as substantial,
- 0.81–1.00 as almost perfect agreement.

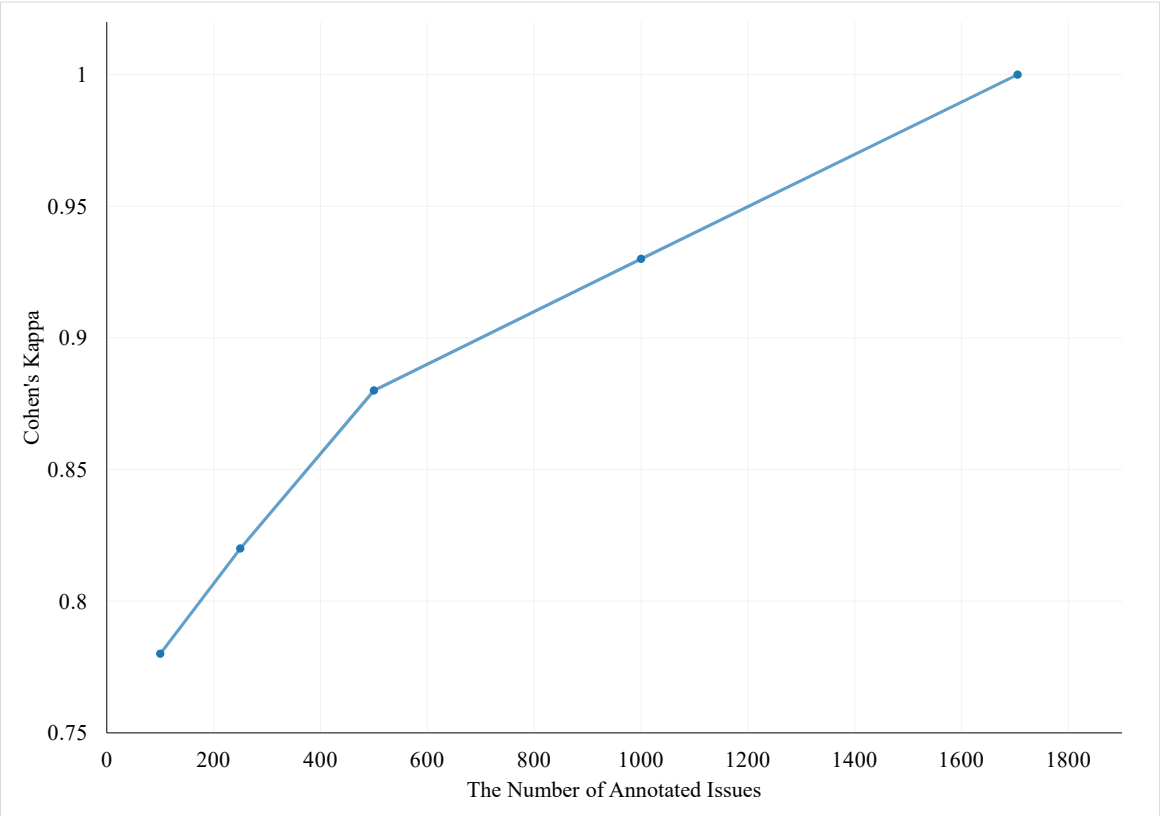

Figure 3.3. Cohen's Kappa values after each annotation step.

Firstly, SES and SSE separately annotate 100 random issues as part of the annotation process. For these 100 issues, the initial Kappa value is calculated to be 0.78. According to the aforementioned rule, the first agreement value gets off to a good start at a substantial level because of the information provided prior to annotation. They get together to discuss issues classified as different and come to a consensus. In the second phase, labeling studies are carried out separately for 150 random issues and

when the Kappa value is calculated again, an almost perfect agreement is reached with a value of 0.82 among the participants. The annotating process takes five steps to complete. After each annotation step, Cohen's Kappa is calculated and reassembled to decide the common tag value for differently tagged issues. The number of separately tagged issues is increased after each step, thus tagging 100, 250, 500, 1000, and 1705 issues overall, respectively. Figure 3.3 shows the number of annotated issues in relation to Cohen's Kappa values after each step.

SES and SSE report the entire amount of time spent on the manual annotation process. Table 3.5 lists the number of annotated reports, independently and collaboratively time spent in minutes, as well as the total hours spent after each step. The manual classification in the case study took shorter for SSE as she worked on the project for two years. Time spent by annotators collaboratively indicates the elapsed time in the meeting for different annotated issues. The manual classification of software issues in the case study took 17.48 hours for SES and SSE, but the overall time spent by the human on the data set took longer. The other tasks for the overall data set study include the preparation of the set-up (two days), analysis of the data and generation of appropriate labels (one week), translation of the issue reports' summaries and descriptions (28 hours), the introduction of the project in the case study (two hours) and the overall reporting (one day), which took about two weeks in total.

Table 3.5. Total time spent on annotation.

Step	Annot.	# Reports	Indep. (min)	Collab. (min)	Total (hours)
1	SES SSE	100	63 35	28	2.1
2	SES SSE	150	58 37	33	2.13
3	SES SSE	250	113 72	36	3.68
4	SES SSE	500	137 98	32	4.45
5	SES SSE	705	166 102	39	5.12

3.3. Preprocessing

Preprocessing is the first step in machine learning, and it involves preparing raw data for analysis as shown in Figure 3.4. The process starts with exporting all the issues from the selected projects in the Jira tracker system to an Excel file. Once exported, the data needs to be cleaned up to eliminate any rows that have all empty columns. For the issue assignment approach, we eliminate the rows that contain empty assignee columns. Another issue that needs to be addressed during preprocessing is dealing with missing values. In the Jira tracker system, if the numerical data that will be used as a feature, such as reopen count, is not assigned, it appears as NaN (Not a Number) in the data set. To avoid this problem, the missing values are changed to zero in the entire data set.

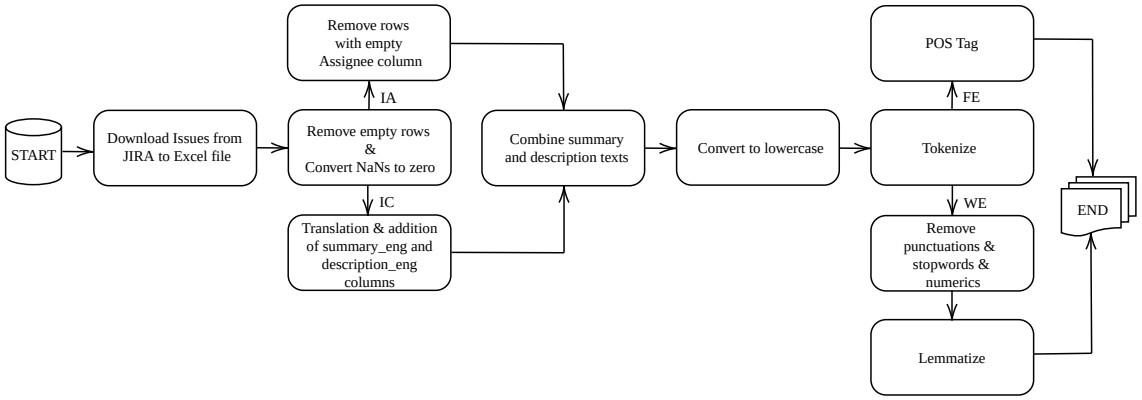

Figure 3.4. Preprocessing steps (IA: Issue Assignment, IC: Issue Classification, FE: Feature Extraction, WE: Word Embedding).

We translate the summary and the description columns of the project which we use in the approach using English issues and add new columns called summary_eng and description_eng. Note that these two fields are available for each issue when an issue report is submitted. We concatenate these two textual parts (summary and description) into new metadata, which we refer to as issue text. We apply a lowercase transformation to ensure consistency in the issue text. This step involves converting all uppercase characters to their corresponding lowercase characters. After the transformation, we tokenize the text into words by splitting it into spaces between words.

For our feature extraction methodology, we do not perform additional text cleaning steps as every word's feature is essential for our process. We perform Part-of-Speech (POS) tagging after the tokenization step. It involves assigning a POS (such as a noun, verb, adjective, etc.) to each word in a given text. We use Zeyrek library [17] for issue texts in Turkish because it is trained on a large corpus of Turkish text. We use the popular preprocessing library namely Natural Language Toolkit (NLTK) [18] for issue texts in English.

For mostly used word embedding methods, we perform additional text cleaning steps to reduce the dimensionality of the data, remove redundant information, and

further improve the accuracy. We eliminate all numeric characters and punctuation marks from issue texts. Stop words are words that do not carry significant meaning in a given context and can be safely ignored without sacrificing the overall meaning of a sentence. Examples of stop-words in English include “the”, “and”, “is”, “are”, etc. Similarly, in Turkish, examples of stop-words include “ve”, “ile”, “ise”, “ama”, etc. We use NLTK which provides a built-in list of stop-words for many languages, including English and Turkish to remove them from issue texts. The last step is lemmatization which is a crucial step in NLP that involves reducing words to their base or dictionary form, known as the “lemma”. The resulting lemma retains the essential meaning of the original word, making it easier to process and analyze text data. We use the same library to lemmatize words in both languages.

3.4. Feature Extraction

This section describes the feature selection steps for the vectors we created and two popular word embedding methods to compare them. The data set obtained from Jira contains over a hundred important fields that we can potentially use as features. However, a significant number of these fields are empty as they are not filled out by the project’s team. To avoid this issue, we have narrowed down the selection of features to only those fields that are either non-empty for each issue or automatically populated by the Jira system when an issue is opened.

The columns of the issue tracking system are utilized as the initial feature set in our study, as presented in Table 3.6. FJN indicates the numerical features from Jira ITS. We consider the data numbers in the columns for these values. Watchers are the users who have expressed interest in a particular issue and want to receive notifications about its progress. They can receive notifications whenever a comment is added or any changes are made to the issue. Typically, multiple individuals subscribe to problematic issues in order to receive notifications upon closure or new comments. Images column is used to attach relevant screenshots, diagrams, or other images to an issue. This helps in better understanding and resolving the issue. When a bug cannot be easily

identified or located, it is common practice for test engineers to include an image of the bug as a reference for developers. This serves as a visual aid to help the developers understand the issue and resolve it more effectively. Reopen Count column tracks the number of times an issue has been reopened after being marked as resolved or closed. It provides insight into the recurring nature of the issue and can help identify if the issue is resolved properly or not. This feature serves to distinguish problematic issues that persist even after the developer has addressed them. Reassign Count column keeps track of how many times an issue has been reassigned to different users or teams. It can help in analyzing the workflow and identifying any inefficiencies. There are various reasons why an issue may be assigned to someone other than the initially assigned individual. These reasons include cases where the assigned person is unavailable or unable to resolve the issue. The linked issues column allows users to link related issues together. It helps in identifying dependencies and tracking progress across multiple issues. The sub-tasks column allows users to break down larger issues into smaller sub-tasks. It helps in better managing and tracking complex issues. The components column specifies the different modules or components of the software that are affected by the issue. It helps in identifying the root cause of the issue and assigning it to the appropriate team or individual.

We only consider whether or not there is a value present in the column for columns that are mostly empty across the issues and do not have diversity in the data to separate each other. We call these boolean features FJB. Reported by customer column indicates if a customer or an internal team member reports the issue. It helps in prioritizing and resolving customer-reported issues quickly. The tested versions column indicates the versions of the software in which the issue is tested. It helps in identifying the specific version where the issue is first detected. The test execution type column specifies the type of test execution, such as Manual or Automated. It helps in tracking the progress and success of the testing phase. The approval type column is used to indicate the type of approval required for the issue, such as Manager Approval or Technical Approval. It helps ensure that the issue is reviewed and approved by the appropriate stakeholders before being resolved. Affects versions column indicates the

versions of the software that are affected by the issue. It helps in identifying the scope of the issue and prioritizing it accordingly.

Several features in our feature set are categorical as FJC, and in order to use them in our analysis, we replaced them with numerical values using label encoding. This process assigns a unique numerical value between 0 and the number of classes minus one to each category, allowing us to use them in our computations. The issue type column defines the type of issue being reported, such as Bug, Improvement, Task, etc. It helps in categorizing and prioritizing issues based on their type. The reporter column indicates the user who reported the issue. It can help in contacting the user for additional information or to gather feedback. The priority column indicates the relative importance of an issue. It can be set to High, Medium, Low, or any other custom value based on the severity of the issue and its impact on the project. The frequency column tracks how often the issue occurs. It helps in identifying patterns and trends in the occurrence of the issue. The bug category column allows users to categorize the issue based on its root cause, such as Performance, Security, Usability, etc. It helps in prioritizing and assigning the issue to the appropriate team or individual. The labels column allows users to add descriptive tags to an issue. It helps in categorizing and searching for issues based on common themes or topics.

Issue texts are utilized to extract features using NLP techniques, as detailed in Table 3.7. The FTN column indicates the numerical features extracted from the text fields. The Summary Words and Description Words columns indicate the number of words in the corresponding issue text columns. To analyze the sentiments of the issue texts, the TextBlob library [19] is used for sentiment analysis. Polarity represents the emotional tone of a piece of text, typically characterized as positive, negative, or neutral. The polarity score ranges from minus one (most negative) to one (most positive). Subjectivity, on the other hand, refers to the degree to which a piece of text expresses opinions or personal feelings, as opposed to being factual or objective. The subjectivity score ranges from zero (most objective) to one (most subjective). As described in Section 3.3, each word in the issue text is classified with known lexical

categories using POS tagging for the morpheme-related features in both Turkish and English. The number of available tags, such as adjective, adverb, conjunction, verb, numeral, etc., is added as a new feature column for each issue. However, not all tags are added to the table. The most effective POS tags as features are discussed in Section 4.2. The FTB column indicates the boolean features extracted from the text fields. The issue text is searched for Bug Words, such as "error", "null", "bug", "server", and "undefined" to determine if there is a bug or for the developers. Test Words, such as "test" and "request" are searched for issues created for the test engineers. Document Words, such as "document(ation)" and "write" are searched for the team leaders, and Design Words, such as "design", "icon", and "logo" are searched for the designers. The negative verb is a boolean value that checks for negative verbs in the issue text. It is assumed that bugs would be more likely to have negative verbs in their definitions rather than being by design or a new task opened. The necessity verb is a boolean value that checks for the verb for necessity in the issue text (e.g., "should" verb in English, "-meli/-malı" suffix in Turkish).

Table 3.6. Features from the columns of the issue tracking system.

Feature	**Name**	**Description**
FJN1	Watchers	The number of users following the issue.
FJN2	Images	The number of the images that have been attached to the issue.
FJN3	ReopenCount	The number of times an issue has been reopened.
FJN4	ReassignCount	The number of times an issue has been reassigned to different users.
FJN5	LinkedIssues	The number of linked related issue keys to the issue. (i.e. ProjectName-2037)
FJN6	SubTasks	The number of added sub-issue keys to the issue. (i.e. ProjectName-2037)

Table 3.6. Features from the columns of the issue tracking system (cont.).

Feature	**Name**	**Description**
FJN7	Components	The number of different components of the software that are affected by the issue (i.e. cloud)
FJB1	ReportedByCustomer	The customer who reports the issue. (i.e X Hotel)
FJB2	TestedVersions	The tested versions of the software in which the issue is tested. (i.e. ProjectName 9.4.x)
FJB3	TestExecutionType	The type of test execution (i.e. manual)
FJB4	ApprovalType	The type of approval required for the issue. (i.e. P1-Pilot)
FJB5	AffectsVersions	The versions of the software that are affected by the issue. (i.e. ProjectName 9.4.x)
FJC1	IssueType	The type of issue. (Story, Epic, Request, Bug, Test Request, Technical task)
FJC2	Reporter	The user who reported the issue.
FJC3	Priority	The relative importance of an issue. (Low, Medium, High, Showstopper)
FJC4	Frequency	The frequency of the issue occurs. (i.e. always-2/2, sometimes-2/4)
FJC5	BugCategory	Category of the issue based on its root cause (i.e. General, Functional)
FJC6	Labels	Added descriptive tags to an issue. (i.e. Admin)

Word Embedding techniques are used to represent text as vectors. To create vectors, we utilize the preprocessed combination of title and description parts of issues. There are various forms of word embeddings available, with some of the most popular ones being Bag of Words (BoW) [20], Term Frequency-Inverse Document Frequency (TF-IDF) [21] and, Word2Vec [22]. We have implemented Tf-Idf and BOW algorithms using the Sklearn library [23] and the Word2Vec algorithm using the Gensim library

[24]. We have tested both BoW unigram and bigram models separately and together. The unigram model stores the text as individual tokens, while the bigram model stores the text as pairs of adjacent tokens. Based on our experiments, the BoW unigram model outperformed the bigram model. This is attributed to the unigram model's superior ability to capture essential text features.

Table 3.7. Features extracted from issue texts.

Feature	**Name**	**Description**
FTN1	SummaryWords	The number of words in the issue summary.
FTN2	DescriptionWords	The number of words in the issue description.
FTN3	PolarityScore	Emotional tone of the issue text. (ranges from minus one (most negative) to one (most positive))
FTN4	SubjectivityScore	Issue text expresses opinions or personal feelings. (ranges from zero (most objective) to one (most subjective))
FTNP	PosTags	The number of every POS tag in the issue text.
FTB1	BugWords	Check for "error, null, bug, server, undefined" words in the issue text.
FTB2	TestWords	Check for "test, request" words in the issue text.
FTB3	DocumentWords	Check for "document/ation, write" words in the issue text.
FTB4	DesignWords	Check for "design, icon, logo" words in the issue text.
FTB5	NecessityVerb	The boolean value that checks for a verb for necessity in the issue text. (i.e. "should" verb in English, "-meli/-malı" suffix in Turkish)
FTB6	NegativeVerb	The boolean value that checks for negative verbs in the issue text. (i.e. "n't, not" in English, "-me, -ma" in Turkish)

3.5. Classification

We can train a classifier to attempt to predict the labels of the issues after we have our features. We experiment with various algorithms and techniques when working on a supervised machine learning problem with a given data set in order to find models that produce general hypotheses, which then make the most precise predictions about future instances, possible. We start with using machine learning techniques that Scikit-learn includes several variants of them to automatically assign issue reports to the developers. We try best-known ML models i.e. Support Vector Machine (SVM), Decision Tree, Random Forest, Logistic Regression, k-Nearest Neighbors (kNN), and Naive Bayes (NB). We use Multinomial and Gaussian NB which are the most suitable variants for text classification. The multinomial model offers the capability of classifying data that cannot be numerically represented. The complexity is greatly decreased, which is its main benefit. We test the one-vs-rest model with SVM, a heuristic technique for multi-class binary classification algorithms. The multi-class data set is divided into various binary classification issues. Scikit-learn offers a high-level component called CountVectorizer that will produce feature vectors for us. The work of tokenizing and counting is done by CountVectorizer, while the data is normalized by TfidfTransformer. In order to combine this tool with other machine learning models, we supply it the title and description fields that we combined.

Most machine learning algorithms do not produce optimal results if their parameters are not properly tuned so we use grid search with cross-validation to build a high-accuracy classification model. We use the GridSearchCV tool from Sklearn library [23] to perform hyperparameter tuning in order to determine the optimal values for a given model. In particular, we use a 10-fold cross-validation. We first split the issues data set into 10 subsets. We train the classifier on nine of them and one subset is used as testing data. Several hyper-parameter combinations are entered, then we calculate the accuracy and the one with the best cross-validation accuracy is chosen and used to train a classification method on the entire data set.

We also try ensemble learning methods [25] which combine the results of multiple machine learning algorithms to produce weak predictive results based on features extracted from a variety of data projections, and then fuse them with various voting mechanisms to achieve better results than any individual algorithm. First, we use the hard-voting classifier which can combine the predictions of each classifier to determine which class has the most votes. Soft voting based on the probabilities of all the predictions made by different classifiers is also an option. Second, we try a classification method called extra trees, which combines the predictions of multiple decision trees. Finally, we combine machine learning methods with bagging, boosting, and stacking ensemble learning techniques. While boosting and stacking aim to create ensemble models that are less biased than their components, bagging will primarily focus on obtaining an ensemble model with less variance than its components [26].

The majority of classification studies using the issue data set do not use or have limited success with deep learning-based text mining techniques. Herbold et al. [6] believe that they lack sufficient (validated) data to train a deep neural network and deep learning should instead be used for this task once the necessary data requirements are satisfied, such as through pre-trained word embeddings based on all issues reported at GitHub. We try some bidirectional language models: DistilBert, Roberta, and Electra to provide empirical evidence. DistilBert [27] is developed using the Bert [28] model. In comparison to pre-trained Bert on the same corpus, this model is quicker and smaller in size. Roberta [29] is retraining BERT with improved training methodology, more data and compute power. Electra [30] uses less computation than Bert to pre-train transformer networks. In 2022, Guven [31] compares language models for the Turkish sentiment analysis approach and the best performance has been achieved by training the Electra language model. These models are pre-trained with a Turkish data set for Turkish approaches [32].

4. EXPERIMENTS AND RESULTS

This chapter presents the findings and outcomes of the conducted experiments, providing a comprehensive analysis of the collected data. We critically assess the performance and effectiveness of the proposed methodology by employing various evaluation metrics and techniques. We compare the extracted features from different perspectives, evaluating their individual contributions to the classification task. Lastly, we present a thorough statistical examination of the obtained results, employing appropriate statistical tests and measures to validate the significance and reliability of the findings.

4.1. Evaluation

Table 4.1 presents the experiment results for the issue classification. The classification models are evaluated on both Turkish and English data sets. Among the models, Support Vector Machine achieved an accuracy of 0.78 for Turkish and 0.81 for English. Logistic Regression showed high accuracy for Turkish (0.88) but slightly lower accuracy for English (0.79). Naive Bayes performed well for both languages, with accuracies of 0.83 for Turkish and 0.82 for English. The Stacking model, combining Random Forest and Linear SVC, outperformed other models, achieving the highest accuracy of 0.90 for Turkish and 0.93 for English.

Table 4.2 presents the experiment results for the issue assignment. The models are evaluated on Team Assignment (TA) and Developer Assignment (DA). Similar to the issue classification task, the Stacking model (RF and Linear SVC) achieved the highest accuracy, with values of 0.92 for TA and 0.89 for DA. Other models, such as Support Vector Machine, Logistic Regression, and Random Forest, also showed good performance with accuracies ranging from 0.86 to 0.88. In both classification and assignment tasks, the transformer-based models, including DistilBert, Roberta, and Electra, demonstrated competitive accuracies, with Roberta and Electra achieving the

highest scores in some cases.

Table 4.3 displays the performance metrics for each class in the Stacking algorithm used for issue classification. In the Turkish classification task, the Stacking algorithm achieved high precision values for the Bug class (0.87) and the NewFeature class (0.73), indicating a low rate of false positives. The Recall scores for the Bug class (0.94) and the Other class (0.62) demonstrate the algorithm's ability to correctly identify the majority of instances belonging to these classes. The F1 scores, which consider both precision and recall, show balanced performance across most classes. In the English classification task, the Stacking algorithm exhibits similar trends with high precision values for the Bug class (0.87) and the Improvement class (0.75). The Recall scores for the Bug class (0.92) and the NewFeature class (0.94) indicate effective identification of instances from these classes. The F1 scores reflect the overall performance of the algorithm, with values ranging from 0.84 to 0.91.

Table 4.4 provides the performance metrics for each class in the Stacking algorithm for issue assignment. Under the Team Assignment (TA) approach, the Stacking algorithm achieved a high precision value of 0.90 for the Developer class, indicating a low rate of false positive assignments. The Recall score of 0.99 for the Developer class demonstrates the algorithm's ability to correctly identify the majority of instances assigned to developers. The Tester class shows a balance between precision (0.95) and recall (0.71), indicating accurate assignments with a relatively high rate of false negatives. The Designer class exhibits similar trends with a precision of 0.92 and a recall of 0.67. The Leader class has relatively lower precision and recall scores, indicating more challenging assignments for the algorithm.

Under the Developer Assignment (DA) approach, the Stacking algorithm achieved high precision values for the Senior class (0.90) and the Mid class (0.89), indicating accurate assignments with low rates of false positives. The Mid class also demonstrates a high recall score of 0.94, indicating effective identification of instances assigned to this class. The Junior class shows a lower precision (0.62) and recall (0.53) compared

to the other classes, suggesting potential challenges in accurately assigning instances to this class.

We also test our classification algorithms using the most popular word embedding techniques to determine how well our features work. Figure 4.1 illustrates the comparison of our feature set with Tf-Idf and BOW methods for the issue assignment. Despite the potential of Word2Vec as a word embedding algorithm, the accuracy results in my approach do not yield comparable outcomes. The issue classification approach also yields similar results. We use the accuracy score as the comparison metric. The graph demonstrates that using our feature set yields superior results while using Tf-Idf and BOW yields comparable results.

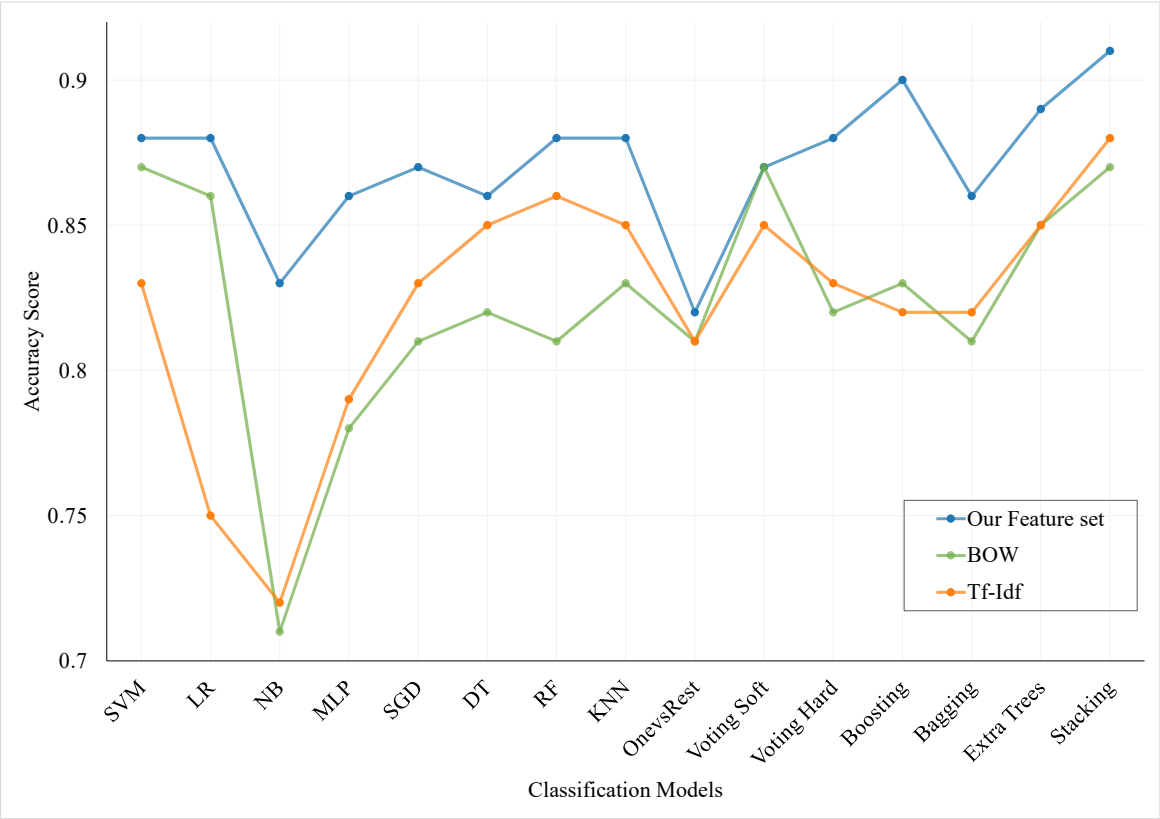

Figure 4.1. Comparison of our feature set with word embedding methods for issue assignment.

Table 4.1. Experiment results for issue classification (IC: Issue Classification, BC: Bug Classification).

Classification Model	**Turkish**		**English**	
	Acc_IC	**Acc_BC**	**Acc_IC**	**Acc_BC**
Support Vector Machine	0.83	0.81	0.76	0.78
Logistic Regression	0.88	0.79	0.81	0.78
Naive Bayes	0.75	0.68	0.83	0.82
Multilayer	0.86	0.81	0.86	0.84
Stochastic Gradient Descent	0.87	0.80	0.88	0.88
Decision Tree	0.86	0.80	0.86	0.88
Random Forest	0.88	0.89	0.88	0.88
KNN	0.88	0.87	0.88	0.88
One vs Rest	0.82	0.77	0.82	0.83
Voting Soft	0.87	0.80	0.87	0.88
Voting Hard	0.88	0.83	0.88	0.88
RF with Boosting	0.90	0.88	0.90	0.89
Bagged DT	0.86	0.80	0.86	0.88
Extra Trees	0.89	0.87	0.89	0.86
Stacking (RF and Linear SVC)	**0.90**	**0.93**	**0.92**	**0.93**
DistilBert	0.88	0.82	0.88	0.89
Roberta	0.90	0.84	0.91	0.90
Electra	0.90	0.84	0.91	0.90

Table 4.2. Experiment results for issue assignment (TA: Team Assignment, DA: Developer Assignment).

Classification Model	**Acc_TA**	**Acc_DA**
Support Vector Machine	0.88	0.86
Logistic Regression	0.88	0.85
Naive Bayes	0.83	0.78
Multilayer	0.86	0.75
Stochastic Gradient Descent	0.87	0.82
Decision Tree	0.86	0.86
Random Forest	0.88	0.86
KNN	0.88	0.88
One vs Rest	0.82	0.81
Voting Soft	0.87	0.87
Voting Hard	0.88	0.88
RF with Boosting	0.90	0.88
Bagged DT	0.86	0.88
Extra Trees	0.89	0.88
Stacking (RF and Linear SVC)	**0.92**	**0.89**
DistilBert	0.88	0.87
Roberta	0.91	0.88
Electra	0.91	0.88

Table 4.3. Performance metrics for each class in the Stacking algorithm for issue classification.

Task	**Class**	**Turkish**			**English**		
		Precision	**Recall**	**F1**	**Precision**	**Recall**	**F1**
IC	Bug	0.87	0.94	0.90	0.87	0.92	0.89
	Improvement	0.72	0.64	0.75	0.75	0.62	0.72
	NewFeature	0.73	0.97	0.83	0.72	0.94	0.84
	Other	0.83	0.62	0.71	0.90	0.92	0.91
BC	CodeBased	**0.96**	**0.94**	**0.95**	**0.94**	**0.92**	**0.93**
	TextBased	0.89	0.88	0.89	0.92	0.90	0.92

Table 4.4. Performance metrics for each class in the Stacking algorithm for issue assignment.

Task	**Class**	**Precision**	**Recall**	**F1**
TA	Developer	0.90	0.99	0.94
	Tester	**0.95**	**0.71**	**0.83**
	Designer	0.92	0.67	0.80
	Leader	0.65	0.57	0.62
DA	Senior	0.90	0.67	0.80
	Mid	0.89	0.94	0.91
	Junior	0.62	0.53	0.57

4.2. Feature Comparison

Reducing the number of redundant and irrelevant features is an effective way to improve the running time and generalization capability of a learning algorithm [33]. Feature selection methods are used to choose a subset of relevant features that contribute to the intended concept. These methods can employ a variety of models and techniques to calculate feature importance scores. One simple approach [34] is

to calculate the coefficient statistics between each feature and the target variable. This method can help to identify the most important features of a given problem and discard the redundant or irrelevant ones. By reducing the number of features used for training a model, the running time of the algorithm can be significantly reduced without sacrificing accuracy. Moreover, feature selection can also improve the interpretability of the model, as it helps to identify the key factors that influence the target variable. We present the coefficient values of each feature for issue assignment in Figure 4.2, for issue classification in English in Figure 4.3, and for issue classification in Turkish in Figure 4.4.

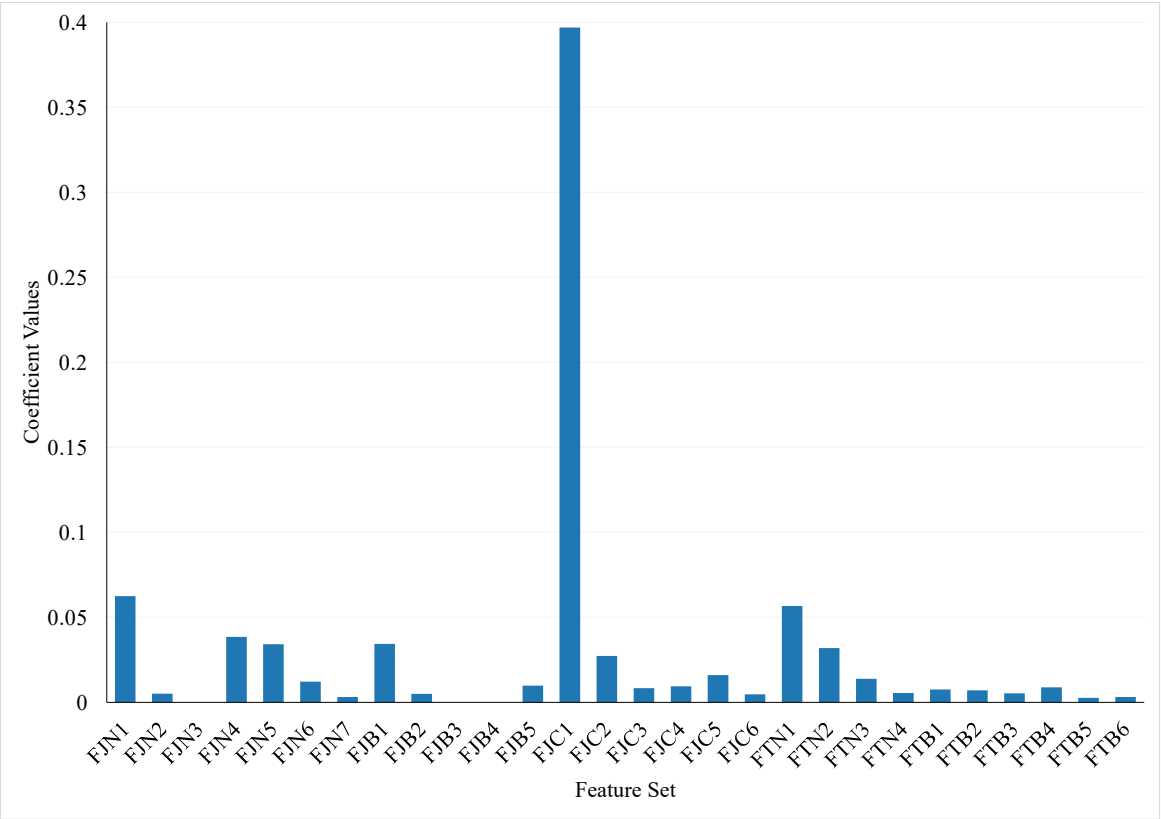

Figure 4.2. Coefficient values of each feature for issue assignment.

In our study on issue assignment, we find that Issue Type, namely FJC1, emerges as the most influential feature from our feature set. Apart from the Issue Type, we discover that the features Watchers and Summary Words also exhibit significant effec-

tiveness in our analysis. Conversely, features such as Reopen Count, Test Execution Type, and Approval Type demonstrate no impact on our issue assignment process. In Figure 4.5, we present the effective POS tags in Turkish, highlighting the most influential ones among all POS tags. Notably, the number of unknown words, verbs, and nouns emerge as the most impactful features. Following the rigorous selection of the best features, we proceed to employ Scikit-learn's [23] SelectFromModel, a powerful meta-transformer designed to choose features based on their importance weights, to retrain our models. Through this process, we carefully identify and select the top eight features that exhibited the highest significance, as determined by the module. Remarkably, leveraging this refined feature subset allows us to achieve optimal performance and attain the most favorable outcome in our experiments.

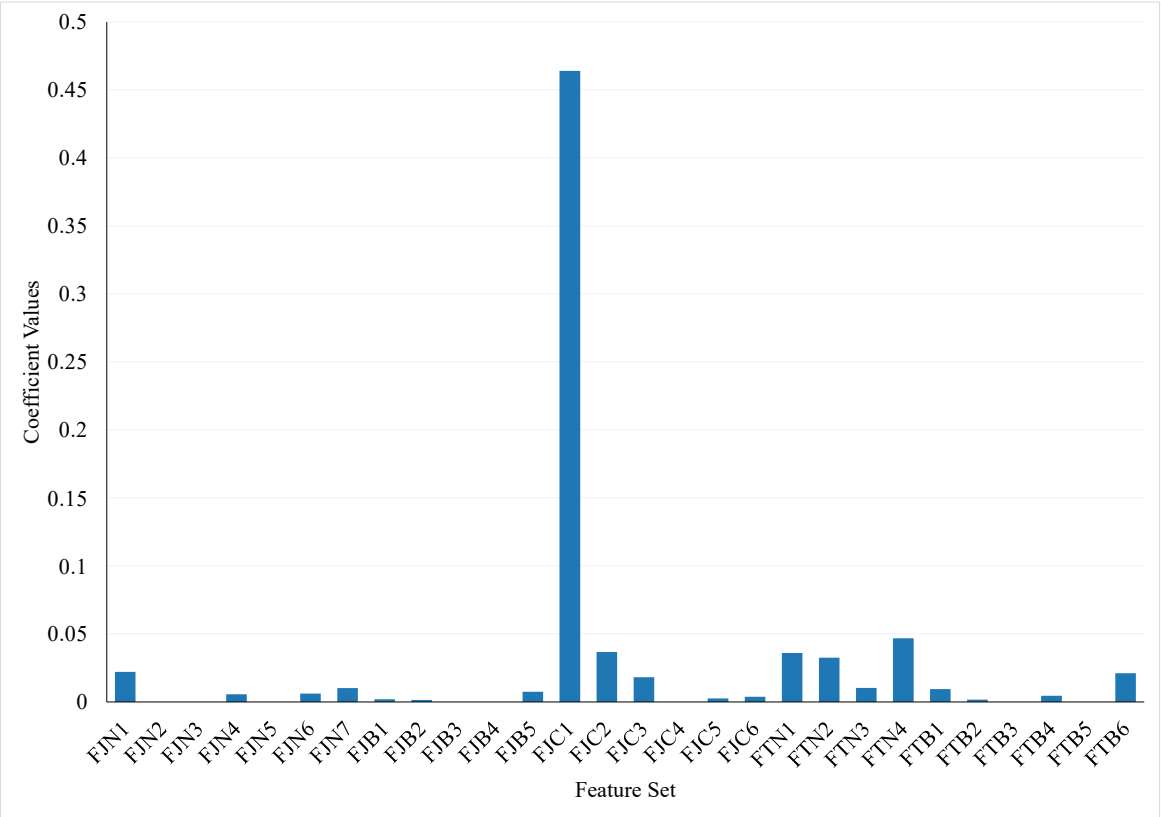

Figure 4.3. Coefficient values of each feature for issue classification in English.

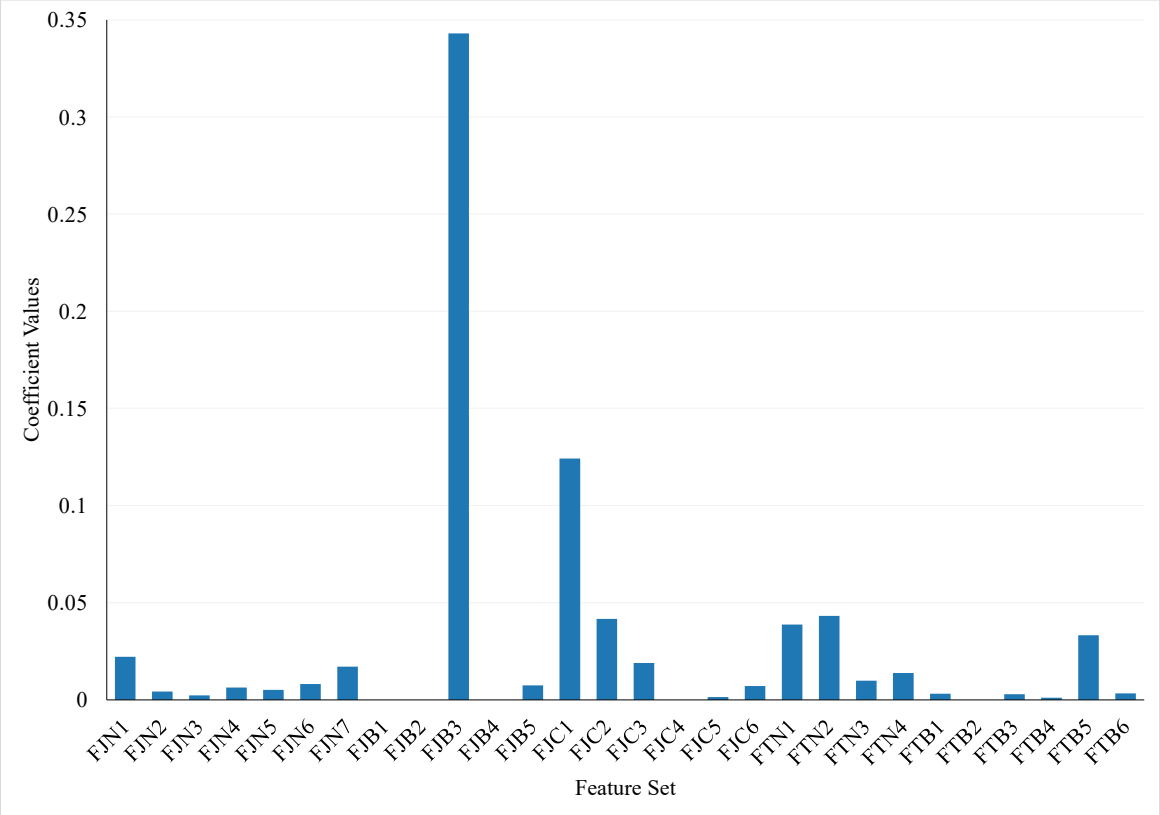

Figure 4.4. Coefficient values of each feature for issue classification in Turkish.

In our comprehensive study on issue classification, we conduct a thorough analysis of two distinct feature sets for both Turkish and English languages. For the Turkish issue classification process, we identify the Test Execution Type (specifically FJB3) as the most influential feature, which has no impact on the issue assignment task. We discover that the features Issue Type and Description Words display significant effectiveness in our analysis. On the other hand, attributes such as Reported by Customer, Tested Versions, Approval Type, Frequency, and TestWords show no influence on the issue classification process in Turkish. Figure 4.5 illustrates the prominent POS tags in Turkish, highlighting the most influential ones among all the POS tags. Mirroring the findings of the issue assignment study, the number of unknown words, verbs, and nouns emerge as the most impactful features.

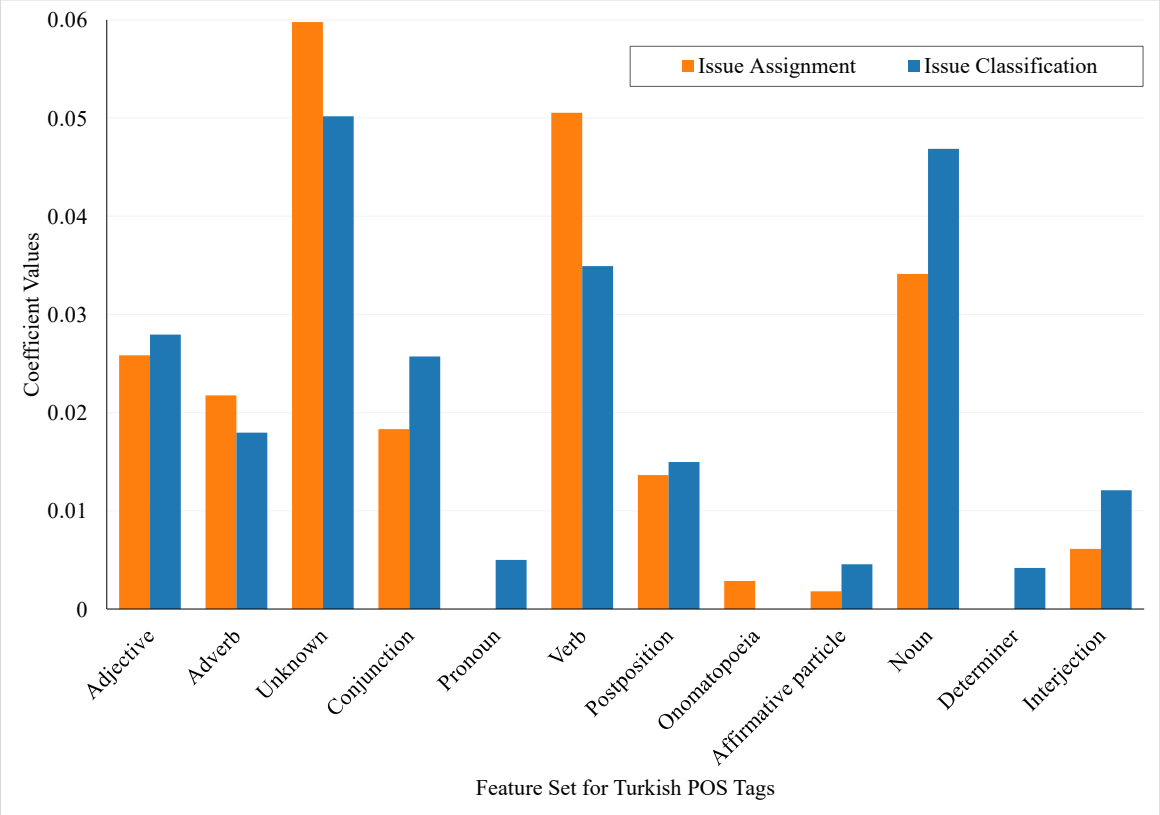

Figure 4.5. Coefficient values of each POS tag in Turkish.

Moving on to the English issue classification process, we found that the Issue Type (specifically FJC1) emerges as the most influential feature from our feature set, similar to the issue assignment study. Subjectivity Score, Reporter, and Summary Words closely follow the Issue Type in terms of effectiveness. Conversely, we observe that several features exhibit no impact on the issue classification in English. Figure 4.6 showcases the influential POS tags in English, highlighting the most significant ones among all the POS tags. The number of modals, verbs, and nouns emerge as the most effective POS tags. To further refine our models, we utilize the SelectFromModel module, which identifies the top eleven features for the English study and the top twelve features for the Turkish study, as they demonstrate the highest significance in the classification process.

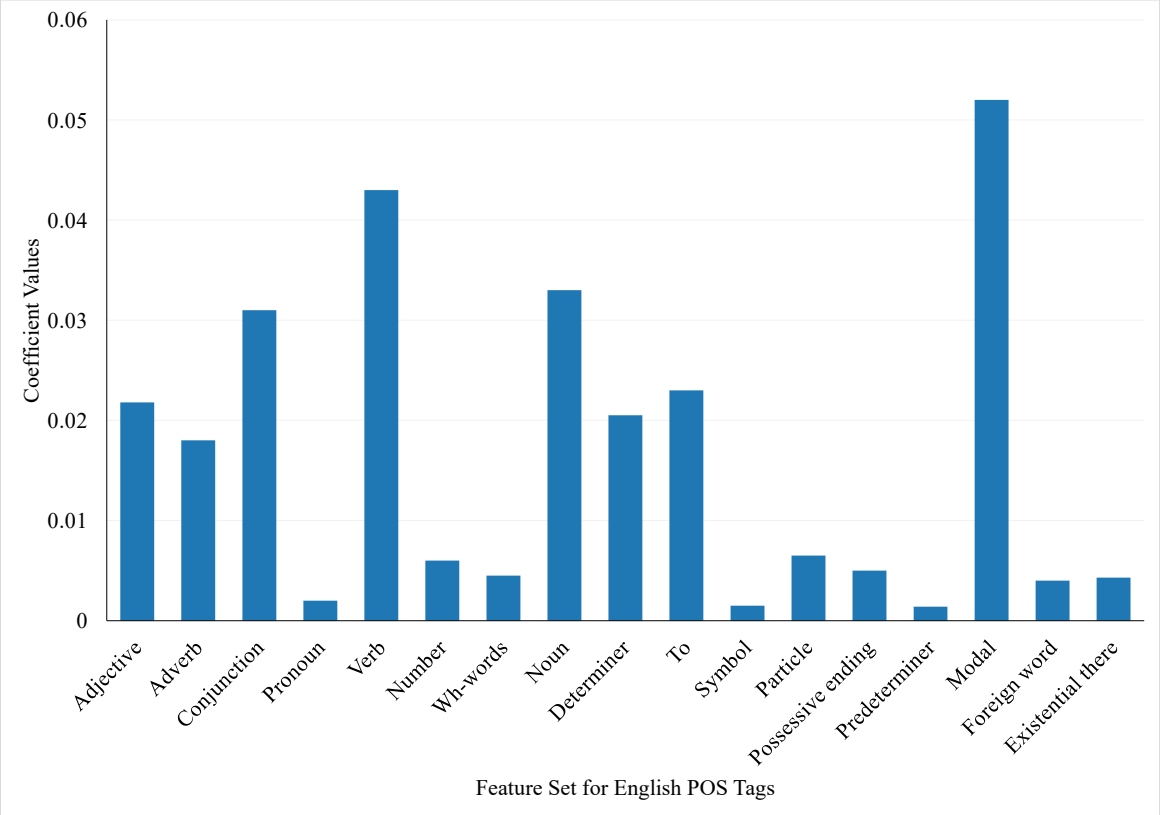

Figure 4.6. Coefficient values of each POS tag for issue classification in English.

4.3. Statistical Analysis

Statistical significance tests are conducted to compare classification methods for determining whether one learning algorithm outperforms another on a particular learning task. Dietterich [35] reviews five approximate statistical tests and concludes that McNemar's test and the 5x2 cv t-test, both have low type I error and sufficient power. In our study, we combine all data sets into a single data set for the classification algorithm. Dietterich [35] recommends using a 5x2 t-test to statistically compare two classifiers on a single data set. The 5x2 f-test, which is also suggested as the new standard by the original authors above, is further expanded upon by Alpaydın [36]. The following lists the null hypothesis and the alternative hypothesis. The null hypothesis is that the probabilities are the same, or in simpler terms, neither of the two models outperforms the other. The alternative hypothesis, therefore, holds that the

performances of the two models are not equivalent.

Accordingly, we apply the 5x2 f-test implemented by Alpaydın [36] which is an extension of the 5x2 cv t-test as stated above. We create the matrix for all pairwise comparisons of learning algorithms. In this test, the splitting process (50% training and 50% test data) is repeated five times. A and B are fitted to the training split and their performance on the test split in each of the five iterations is assessed. The training and test sets are then rotated (the training set becomes the test set, and vice versa), and the performance is computed again, yielding two performance difference measures. Then, the mean and variance of the differences are estimated and the f-statistic proposed by Alpaydın is calculated as

$$f = \frac{\sum_{i=1}^{5} \sum_{j=1}^{2} (p_i^j)^2}{2 \sum_{i=1}^{5} s_i^2}, \tag{4.1}$$

where p_i^j is the difference in error rates between the two classifiers on fold $j = \{1, 2\}$ of replication $i = \{1, ..., 5\}$ and s_i^2 is estimated variance. We reject the null hypothesis that the two models' performances are equal if the p-value is less than our chosen significance level (p-value $< \alpha = 0.05$) and accept that the two models are significantly different. Table 4.5 presents the results of the statistical F test analysis comparing the Stacking algorithm to other classification models across three different tasks: Issue Assignment, Issue Classification in English, and Issue Classification in Turkish. The table provides the p-values and corresponding hypothesis decisions for each classification model. Based on the p-values compared to the chosen alpha value, we can accept the null hypothesis that there are no significant differences among all ensemble learning and deep-learning techniques. However, when comparing these techniques with machine learning methods, the majority of cases result in rejecting the null hypothesis, indicating significant performance variations. This observation is evident from the table, which highlights the substantial rejection of the null hypothesis in most comparisons between ensemble learning, deep learning, and machine learning methods.

Table 4.5. Analysis of the Stacking algorithm's statistical test results in comparison to others.

Classification Model	**IA**		**ICE**		**ICT**	
	p-val	**Hypo.**	**p-val**	**Hypo.**	**p-val**	**Hypo.**
SVM	0.0076	**Reject**	0.0269	**Reject**	0.0077	**Reject**
Logistic Regression	0.0412	**Reject**	0.2207	Accept	0.4497	Accept
Naive Bayes	0.0413	**Reject**	0.0005	**Reject**	1.8142	Accept
Multilayer	0.1336	Accept	0.1814	Accept	0.0736	Accept
SGD	0.0412	**Reject**	0.0	**Reject**	0.0094	**Reject**
Decision Tree	0.0265	**Reject**	0.2482	Accept	0.6831	Accept
Random Forest	0.4795	Accept	0.0	**Reject**	0.0	**Reject**
KNN	0.0412	**Reject**	0.0162	**Reject**	0.0404	**Reject**
One vs Rest	0.0025	**Reject**	0.0269	**Reject**	0.0077	**Reject**
Voting Soft	0.0736	Accept	0.1336	Accept	0.4795	Accept
Voting Hard	0.1336	Accept	0.6831	Accept	0.2482	Accept
RF with Boosting	0.4795	Accept	0.0	Accept	0.4795	Accept
Bagged DT	0.4795	Accept	0.4795	Accept	0.6831	Accept
Extra Trees	0.2482	Accept	0.6831	Accept	0.4795	Accept
DistilBert	0.1376	Accept	0.2482	Accept	0.2482	Accept
Roberta	0.3675	Accept	0.4795	Accept	0.6831	Accept
Electra	0.3675	Accept	0.4795	Accept	0.6831	Accept

5. DISCUSSION and QUALITATIVE ANALYSIS

In this chapter, we focus on the validity of threats that could impact the reliability and generalizability of our study results. We discuss potential sources of bias, confounding variables, and other factors that may affect the validity of our study design, data collection, and analysis. We also describe our experiment for user evaluation in the company, which is aimed at investigating the effectiveness of our approach for issue assignment. We explain the methodology we use to gather feedback from users, such as surveys or interviews, and how we plan to analyze the results.

5.1. Threats to Validity

In this section, we discuss the validity threats to our study concerning internal validity, external validity, construct validity, and conclusion validity. (Wohlin et al. [37])

Internal validity pertains to the validity of results internal to a study. It focuses on the structure of a study and the accuracy of the conclusions drawn. To avoid creating a data set with inaccurate or misleading information for the classification, the corporate employees labeled the employees by fields in the data set. We attempt to use well-known machine learning libraries during the implementation phase to prevent introducing an internal threat that can be brought on by implementation mistakes. All of our classification techniques specifically make use of the Python Sklearn [38] package. Sklearn and NLTK are used to preprocess the text of the issues and Sklearn metrics are used to determine accuracy, precision, and recall. We think that the application that can cause the most internal threat is when allocating Turkish issues to part of speech tags. Since Turkish is not as common as English and is an agglutinative language, it is more difficult to find a highly trained POS tagger library that provides high precision. We decided to use the Turkish_pos_tagger [39] library by comparing many parameters such as data numbers, accuracy percentages, and usage popularity among many Turkish POS tagger libraries. Turkish_pos_tagger library includes 5110 sentences

and the data set originally belongs to Turkish UD treebank. For 10-fold validation, the accuracy of the model is 95%.

We employ manual issue annotation as a part of this study. The project's senior software engineer, to whom the majority of the issues are assigned, and a software engineering student who is unrelated to the project are working together to label issues in order to lessen the threat posed by wrong annotation. Labeling is frequently paused while various assignments are discussed and a consensus is reached. But since we lack ground truth labels, the incorrect classification may still have happened and affected the outcomes.

External validity involves the extent to which the results of a study can be applied beyond the sample. We use the data set of five different applications with thousands of issues. These projects use different software languages and various technologies at the front-end and back-end layers, including restful services, communication protocols such as gRPC and WebSocket, database systems, and hosting servers. The issue reports cover a long period of time from 2011 to 2021. However, all the projects we get from the issue reports are mainly concerned with the development of web projects made to run in the browser on the TV. Issues contain many TV-specific expressions, such as application behaviors that occur as a result of pressing a button on the remote or resetting the TV. We take great effort to ensure that the features we design to prevent external validity concerns are not particular to the data we utilize. For our classification analysis to be applicable in other fields, we believe that it will be sufficient to replicate it using other data sets from various fields.

Construct validity refers to the degree to which a test or experiment effectively supports its claims. The performance of our automated issue assignment system is evaluated using the well-known accuracy metric. We additionally back it up with two other well-known metrics, namely recall, and precision. Tasks in the organization where we use the data set can only be given to one employee, and that employee is also in charge of the sub-tasks that make up the task. This makes assigning the issue report to

a single employee group as a binary classification, an appropriate classification method. However, it could be necessary for a business analyst, a product owner, and a software developer to open the same task in different project management or different software teams. For this kind of data set, the binary classification research we conducted is not a suitable approach.

Conclusion validity refers to the extent to which our conclusion is considered credible and accurately reflects the results of our study. All the issue data we used are real issue reports collected from the software development team. We use issue reports in 10 years time span, but according to the information we received from within the company, the turnover rate is low compared to other companies, and especially the team leaders and testers, who usually created the tasks, are generally people who worked for the company for 10+ years. This may have caused a high similarity in the language, namely the text, of the opened tasks and created a conclusion threat at the accuracy rate. To assess how well the accuracy values we find are consistent among themselves, we used statistical significance tests as outlined in Section 4.3. By proving our hypotheses in this manner, we showed the consistency of the outcomes we discovered and the effectiveness of our methods.

5.2. User Evaluation for Issue Assignment

An employee of the company who served as a Senior Software Developer and created the architecture of the projects where we use the data set as well as an employee who serves as the Team Leader of the three projects we have are interviewed for this section about our application. On all the projects where we use the data set, a software developer has been employed by this company for three years. Three of the projects are being led by the team leader, who has been employed by the company for ten years. After spending about 15 minutes outlining our application, we evaluate the outcomes by conducting a few joint experiments.

In the first section, we find an issue that has been done and our model assigns a different assignee value than the one made on ITS, and we talk about it. This issue is assigned to the team leader in ITS, but our model assigns it to the junior software developer. We want to know what team members think about this example of a wrong assignment. Our model assigns it to a different employee, both in terms of experience and field. First off, they state that from the test team or the customer support team, an incorrectly tested issue that is not actually a problem or issues that should not be developed can be opened. In this case, the team manager can take the issue and bring it to Won't Fix status. This is also the case in this issue. In fact, this is something that should not be done. They state that for such situations, the team manager must decide.

In the second part, we assign an idle issue that has not yet been assigned by our classification method. The model labels the issue as the junior software developer. We are asking for their opinion to find out if this is the correct assignment. Considering the scope of the job, both team members state that it is appropriate to assign this job to a junior friend, as the requirement for seniority is quite low. Assigning it to mid or senior employees would not be a problem either, but they would not consider assigning this issue to more experienced employees.

In the next section, we give a data set consisting of 20 issues assigned and closed in ITS to the senior software developer and team leader in the company. They label these issues according to the labels we set. We compare the tags made with the assigned values in the issue's data set and the assignments made by our best working system. Table 5.1 shows the results of this comparison.

First, we compare the labels of Senior Software Developer and Team Leader with the assigned values on the ITS and the label values of our best model and find the issues where all four are the same. The least number of intersections are 11 common labels with values where all four of them are the same. In order to understand whether this difference is due to mislabeling of our model or due to labeling differences between

employees and ITS, in combination 2, we check the values where the labels made by our model and all three of the labels made by the employees intersect. Here the total intersection turns out to be 13. We show the labels that these two issues are assigned differently on ITS to the employees. Both employees gave the same tag value, but a different employee type seems to have closed the issue in ITS. They think that if it's a problem that an employee has dealt with before, they may have taken it for that reason and it could be both types of labels. In the third combination, we find the values that our model has labeled in common with at least one employee to see if there are labels that they think differently among the employees or if our model has assigned completely different assignments from the two. Here, the number of common tags increases to 18. We find five issues that two employees tagged differently, and we ask the employees what they think about these differences. After the exchange of ideas between each other, in two of the different tagged issues, the developer thinks that the tag value of the leader is more appropriate, and in the other two, the leader thinks that the tag value of the developer is more accurate. In an issue, they cannot reach a common decision. Finally, we add the values from the ITS to the combination and find the values where our model coincides with at least one of the labels of the two employees and the label from the ITS. Thus, we see that our model and ITS have the same label value with that undecided issue.

Table 5.1. Comparison of label results.

Combination	**# labels**
LabelByDeveloper $\cup$ LabelByLeader $\cup$ Model $\cup$ ITS	11
LabelByDeveloper $\cup$ LabelByLeader $\cup$ Model	13
Model $\cup$ (LabelByDeveloper $\vee$ LabelByLeader)	18
Model $\cup$ (LabelByDeveloper $\vee$ LabelByLeader $\vee$ ITS)	19

In the last section, we direct the questions we prepared to the employees to get an idea about the system. We ask whether they would prefer such a system to be used in business life. They state that if they are converted into an application and

the necessary features are added, for example, if an interface is provided where they can enter the current number of personnel, and their experiences, add and remove employees who are currently on leave, and if they turn into a plugin that integrates with the Jira interface, they will want to use it. Afterward, we ask if you find the system reliable and do you trust the assignments made. They say that they cannot completely leave the assignment to the application, and they will want to take a look at the assignments made by the application. The team leader adds that if there is a feature to send his approval, for example, by sending an e-mail before making the appointment, he will take a look and approve it, except for exceptional cases, and his work will be accelerated. As a result of the assignments made with the system, we address the question: Do you think that the average task solution time will decrease? It can reduce the average task resolution time, but they state that they think that if similar tasks are constantly sent to similar employee groups, this may have undesirable consequences for employee happiness and development. Next, we ask if you think using the system will reduce planning time. There are times when they talked at length in team planning meetings about who would get the job and who would be more suitable. At least, they think that if they have a second data, it can be a savior in cases where they are undecided. Finally, we would like to know your suggestions to improve the system. They state that if this system is going to turn into an application, they will want to see the values that the application pays attention to, to be able to edit and remove or add new ones. They think that if it has a comprehensive and user-friendly interface, it will still be suitable for use in business processes.

6. RELATED WORKS

Several studies in the literature have focused on issue classification, which has addressed a variety of objectives, including issue assignment, effort estimation, issue prioritization, and so on. In this section, we briefly give details regarding issue classification and issue assignment studies in general and all Turkish-language issue classification studies in particular.

6.1. Issue Classification

We summarize the papers mentioned in this section, presenting the methods employed in Table 6.1 and their data and labels in Table 6.2. OSS refers to open-source software and PROP refers the proprietary software. Most of the existing literature in issue classification has focused on open-source software issues, with only a limited number of studies highlighting commercial software issues, which is the main focus of our study. GitHub, being the most popular ITS, has been extensively used in previous works [40–45] due to its built-in tags that can be utilized for issue labeling without the need for manual annotation. GitHub offers seven default labels for issue classification including bug, enhancement, question, help-wanted, duplicated, wont-fix, and invalid. In addition, members can create and modify labels based on their project requirements [46]. The amount of GitHub issues used in previous works [40, 43, 44] is considerably larger compared to other studies, which can be attributed to the vast availability of open-source projects on GitHub.

Most studies [6,7,40,45,47–51] model issue label recommendation as a binary classification problem that determines whether or not it is a bug. Fewer studies [8,41–44,52] employ it as a multi-class problem. In 2013, Herzig et al. [8] introduce six different issue labels – bug, feature request, improvement request, documentation request, refactoring request, and others. Herbold et al. [53] use a similar schema, but they also add the category tests and combine the categories request for improvement, feature request,

and refactoring into a single category called improvement. The labels “bug”, “feature”, “improvement”, and “other” are all used, and the bug category has two distinct sub-labels in our study. We classify documentation requests under the other category and there are no issues in our data set that we could classify as refactoring requests. Kallis et al. [41, 42] and Siddiq et al. [44] select Bug, Enhancement, and Question labels and Izadi et al. [43] select Bug, Enhancement, and Support/Documentation labels for the GitHub-based issues. Otoom et al. [51] refer to labels in different ways as perfective and corrective, but the logic behind them is the same: perfective refers to bug fixing while corrective refers to new features or enhancements. From a different perspective, Köksal et al. [52] use and adapt the bug classification schema defined by Seaman [54] in 2008 with four bug categories: assignment/initialization, an external interface, an internal interface, and other. Assignment/initialization refers to a variable or data item that is assigned a value incorrectly or is not initialized properly. Internal interface errors affect the interfaces between system components, while external interface errors affect the user interface [54].

To the best of our knowledge, Antoniol et al. [7] published the first approach in this field. In their work, they utilized a Term Frequency Matrix (TFM) to describe the features. Since then, many researchers have adopted this approach [7, 47, 51], as well as other techniques, such as the use of TF-IDF vectors [6, 40, 43] and N-gram IDF [49] derived from the title and description of the issues, to create the feature set for analysis. These feature sets are commonly used in the field and have proven effective in various applications of issue report analysis. Some of the studies [6,41,45,52] suggest that FastText is a powerful tool for issue classification and can be particularly effective for analyzing short text data, such as issue reports. Its ability to handle out-of-vocabulary words and capture sub-word information also makes it well-suited for this task. Different machine learning algorithms have been used for issue classification, with varying degrees of success. BERT and RoBERTa have shown promise in recent studies [43–45], while commonly used machine learning models like Decision Trees, Naive Bayes, and Random Forest have also been used successfully in some cases [7, 40, 47, 49, 51, 52].

Table 6.1. Comparison of the methods in the related works.

Ref.	**Annotation**	**Techniques Used**
[7]	Yes	TFM, Alternating Decision Trees, Naive Bayes, Logistic Regression
[8]	Yes	-
[47]	Yes	TFM, Multinomial Naive Bayes, Bayesian Net
[48]	[8]	Naive Bayes, Linear Discriminant Analysis, K-Nearest Neighbors, Support Vector Machine, Decision Tree, Random Forest
[40]	No	TF-IDF, Support Vector Machine, Naive Bayes, Logistic Regression, Random Forest
[49]	[8]	N-gram IDF, Logistic Regression, Random Forest
[50]	[8]	Long Short-Term Memory
[53]	Mixed	SZZ
[41]	No	FastText
[51]	Yes	TFM, Naïve Bayes, Support Vector Machines, Random Trees
[6]	Mixed	FastText, TF-IDF, Multinomial NB, RF
[52]	Yes	FastText, Naïve Bayes, Support Vector Machines, K-Nearest Neighbors, Logistic Regression, Decision Tree, Random Forest
[42]	No	FastText
[43]	No	TF-IDF, RoBERTa, Random Forest
[44]	No	BERT
[45]	Mixed	FastText, RoBERTa, Logistic Regression, Decision Trees, Naive Bayes, K-Nearest Neighbours
Our study	Yes	TF-IDF, BOW, Word2Vec, SVM, LR, NB, Multilayer, SGD, DT, RF, KNN, One vs Rest, Voting, RF with Boosting, Bagged DT, Extra Trees, Stacking, DistilBert, Roberta, Electra

Table 6.2. Comparison of the data set in the related works.

Year	**Paper**	**Labels**	**Type**	**Data set(s)**	**Issues**
2008	Antoniol et al. [7]	Bug, Nonbug	OSS	Eclipse, Mozilla and JBoss	1,800
2013	Herzig et al. [8]	Bug, Feature, Improvement, Documentation, Refactoring, Other	OSS	HTTPClient, Jackrabbit, Lucene-Java, Rhino, Tomcat5	7,401
2016	Zhou et al. [47]	Bug, Non-Bug	OSS	Mozilla, Eclipse, JBoss, Firefox and OpenFOAM	3,220
2017	Pandey et al. [48]	BUG, NUG	OSS	HttpClient, Lucene, Jackrabbit	5,587
2017	Fan et al. [40]	Bug, Nonbug	OSS	80 popular projects in GitHub	252,084
2017	Terdchanakul et al. [49]	Bug, Nonbug	OSS	Jackrabbit, Lucene and HttpClient	5,590
2018	Qin et al. [50]	Bug, Nonbug	OSS	Jackrabbit, Lucene and HttpClient	5,591

Table 6.2. Comparison of the data set in the related works (cont.).

Year	Paper	Labels	Type	Data set(s)	Issues
2019	Herbold et al. [53]	Bug, Improvement, Test, Documentation, Other	OSS	38 Apache projects	11,154
2020	Herbold et al. [6]	Bug, Nonbug	OSS	mixed from Jira and Bugzilla	641,855
2019	Kallis et al. [41]	Bug, Enhancement, Question	OSS	GitHub projects	30,000
2019	Otoom et al. [51]	Perfective, Corrective	OSS	AspectJ, Tomcat and SWT	5,800
2021	Köksal et al. [52]	Assignment/ Initialization, External Interface, Internal Interface, Other	PROP	Proprietary Bug Data set	504
2021	Kallis et al. [42]	Bug, Enhancement, Question	OSS	GitHub projects	30,000
2022	Izadi et al. [43]	Bug, Enhancement, Support/ Documentation	OSS	60,958 repositories from GitHub	817,743

Table 6.2. Comparison of the data set in the related works (cont.).

Year	Paper	Labels	Type	Data set(s)	Issues
2022	Siddiq et al. [44]	Bug, Enhancement, Question	OSS	GitHub projects	803,417
2023	Afric et al. [45]	Bug, Non-Bug	OSS	seven popular repositories from GitHub	7,436
2023	Our study	Bug-CodeBased, Bug-TextBased, NewFeature, Improvement, Other	PROP	Hospitality Tv Project	1,705

6.2. Issue Assignment

Several types of research have been conducted in order to automate the time-consuming task of issue assignment. In 2017, Goyal et al. [55] review and categorize 75 research papers on the automated bug assignment area. They identify seven categories: machine learning [9, 38, 56–59], information retrieval [60], auction [61], social network [62], tossing graph [63], fuzzy set [64] and operational research based [65] techniques. They capture the fact that for automatic bug report assignment, machine learning and information retrieval techniques are the most popular ones. In recent years, deep learning algorithms have also been successfully applied in this field, which has recently revolutionized the idea of word sequence representation and demonstrated encouraging advancements in a number of classification tasks [66]. In this section, we restrict our focus to machine learning and deep learning architectures used to train issue assignment

systems.

The machine learning algorithms use historical bug reports to build a supervised or unsupervised machine learning classifier, which is then used to choose appropriate developers for new bug reports. Naive Bayes is the most widely used classifier in machine learning-based approaches according to prior studies [38, 58, 67–69], and it has been extensively tested [55] in the bug reports of open-source projects. Most studies use Eclipse [57, 58, 70–73] and Mozilla [58, 71, 73, 74] projects to validate their proposals. Machine learning models in most approaches [57, 59, 71] use only summary and description as textual features of the issues. Jonsson et al. [9] use the combined title and description as textual features and version, type, priority, and submitter columns as nominal features. Sharma et al. [75] consider bug attributes, namely, severity, priority, component, operating system, and the bug assignee.

To estimate the value of terms, most of the approaches [9, 38, 71, 76] in the literature employ term-weighting techniques like Tf-Idf. Jonsson et al. [9] represent textual parts in the bug reports as the 100 words with the highest Tf-Idf. Shokripour et al. [38] use time metadata in Tf-Idf (Time-Tf-Idf). To determine the value of terms in a document and corpus, the Tf-Idf technique only considers their frequency. However, in determining the weight, time-based Tf-Idf considers the time spent using the term in the project. The developer's recent use of the term is taken into account when determining the value of the developer's expertise. They rank the developers according to their calculated term expertise, and the first developer on the list is assigned to fix the new bug.

However, prior studies focused on open-source projects only but rarely [9, 59] attempted in industrial environments like our study. Jonsson et al. [9] use ensemble learner Stacked Generalization, which is our best method also, that combines several machine learning classifiers on data from the automation and telecommunication company. In their approach, the different classes correspond to the development teams. Oliveira et al. [59] also use the data set of a large electronic company. They create

a model that can distribute new issues according to the responsibilities of the teams using a variety of machine learning techniques and the WEKA [77] tool.

To improve prediction accuracy, some researchers use incremental learning methods. Bhattacharya et al. [58] use various machine learning algorithms and achieve the best results using the NB classifier in combination with the product-component features, tossing graphs, and incremental learning in mostly used large projects: Mozilla and Eclipse. Xia et al. [56] offer the multi-feature topic model (MTM), a specialized topic modeling approach that extends Latent Dirichlet Allocation (LDA) for the bug assignment. To map the term space to the subject space, their approach takes into account product and component information from issue reports. Then, they suggest an incremental learning mechanism that uses the topic distribution of a new bug report to assign an appropriate developer based on the reports that the developer has previously fixed.

The deep learning algorithms are attempted first in 2017 [78] for bug report assignment recommendation, to the best of our knowledge. Gupta et al. [79] describe the popular deep learning approaches applied to the domain of bug reports and Recurrent Neural Networks (RNN) and Long Short Term Memory (LSTM) are a few famous approaches being used for the deep learning-based approaches [78, 80]. Mani et al. [80] use title and description parts and Florea et al. [78] use the component id, product id, and bug severity fields as one-hot-encoded categorical variables in addition to title, description, and comments to represent the issues. In 2022, Feng et al. [81] use four transformers models BERT and RoBERTa along with their distilled counterparts DistilBERT, DistilRoBERTa in an ensemble using a resolver team, resolver person, and description columns of the issues.

6.3. Turkish Issue Reports

In research using Turkish issue reports, there are limited studies available in a few fields. The reason may be the agglutinative nature of the Turkish language and

the absence of a shared data set for Turkish issues in the literature. Aktas et al. [82] classified the issues they gathered from the banking industry among various software development teams. They use the Jira issue reports for their research like our study. They use SVC, CNN, and LSTM models to solve the classification problem, and they represent the summary and description columns of the issue reports as ordered vectors of Tf-Idf scores. The linear SVC classifier offers the best assignment accuracy for their research with a 0.82 score. Koksal et al. [52] present an automated bug classification approach using a commercial proprietary bug data set. They apply several machine learning algorithms and the SVM classifier is the best algorithm with 0.72 accuracy. In 2022, Tunali [83] prioritize the software development demands of a private insurance company in Turkey. He proposes several deep-learning architectures and a pre-trained transformer model called distilbert-base-turkish-cased based on DistilBERT to achieve the highest accuracy of 0.82.

7. CONCLUSION

This study focuses on automated issue assignment and issue classification using proprietary issue reports obtained from the electronic product manufacturer's issue tracking system. The objective of the issue assignment approach is to assign issues to appropriate team members based on their respective fields. The team members are categorized into Software Developer, Software Tester, Team Leader, and UI/UX Designer. Among these categories, the majority of the data set consists of developers. Efficiently allocating issues to developers is critical for effective time management. To achieve this, we further classify developers into Senior, Mid, and Junior levels, which are widely accepted labels in the industry. Regarding the issue classification approach, two annotators, namely SES and SSE, manually annotate and translate issues into English. The issues are categorized into Bug, New Feature, Improvement, and Other. Given the high density of the Bug class in the data set, we aim to refine the classification by dividing it into two sub-classes: Code-based and Text-based bugs. This finer categorization provides more specific insights into the nature of bugs and facilitates targeted resolutions.

Our focus lies in extracting features from the filled Jira columns, as well as the title and description texts of the issues, utilizing NLP techniques. These features serve as inputs to our learning methods, enabling us to analyze and classify the issues effectively. Additionally, we employ other commonly used word embedding methods which are Tf-Idf, BOW, and Word2Vec to generate feature vectors from the text fields. This step, implemented using the Sklearn and Gensim library, allows us to compare the performance of our feature set against alternative approaches. Furthermore, to assess the effectiveness of our overall methodology, we incorporate widely adopted deep-learning techniques, namely DistilBert, Roberta, and Electra.

Following the production of feature vectors, we proceed to implement the proposed system utilizing established machine learning techniques. With the aim of en-

hancing predictive performance, we employ ensemble methods that leverage a diverse range of machine-learning algorithms. To evaluate the effectiveness of our system, we employ widely recognized metrics such as accuracy, precision, recall, and F1-score which serve as indicators of its performance. To further refine our predictions, we employ a robust technique known as 10-fold cross-validation. In order to conduct a thorough statistical analysis, we construct a matrix to compare and contrast the effectiveness of our proposed strategies. This matrix allows us to assess the performance of our system across different algorithms, ensemble techniques, and evaluation metrics.

Our future endeavors involve the development of a versatile tool applicable to diverse software team models. To fortify our work, we actively engage in discussions and pursue collaborations to acquire data set from businesses operating across various domains, such as game development and banking applications. This broadened data set will enable us to enhance our model's capabilities for multi-class classification, accommodating different roles within software teams, including product owners and business analysts. Furthermore, we are committed to ensuring compatibility and flexibility by incorporating various business branches into our data set. By incorporating real-world data obtained directly from industry sources, both in English and Turkish, we will conduct comprehensive evaluations through diverse studies. Expanding on the existing features, we intend to utilize the same data set for future research endeavors, such as effort estimation [84, 85], further solidifying the value and applicability of our work in the field.

REFERENCES

1. Bertram, D., A. Voida, S. Greenberg and R. Walker, "Communication, Collaboration, and Bugs: The Social Nature of Issue Tracking in Small, Collocated Teams", *Proceedings of the 2010 ACM Conference on Computer Supported Cooperative Work*, pp. 291–300, New York, USA, 2010.

2. Stellman, A. and J. Greene, *Applied Software Project Management*, O'Reilly Media, California, 2005.

3. Merten, T., M. Falis, P. Hübner, T. Quirchmayr, S. Bürsner and B. Paech, "Software Feature Request Detection in Issue Tracking Systems", *IEEE 24th International Requirements Engineering Conference (RE)*, pp. 166–175, Beijing, China, 2016.

4. Sebastiani, F., "Machine Learning in Automated Text Categorization", *ACM Computing Surveys (CSUR)*, Vol. 34, No. 1, pp. 1–47, 2002.

5. Tsoumakas, G. and I. Katakis, "Multi-Label Classification: An Overview", *International Journal of Data Warehousing and Mining (IJDWM)*, Vol. 3, No. 3, pp. 1–13, 2007.

6. Herbold, S., A. Trautsch and F. Trautsch, "On the Feasibility of Automated Prediction of Bug and Non-Bug Issues", *Empirical Software Engineering*, Vol. 25, No. 6, pp. 5333–5369, 2020.

7. Antoniol, G., K. Ayari, M. Di Penta, F. Khomh and Y.-G. Guéhéneuc, "Is It a Bug or an Enhancement? A Text-Based Approach to Classify Change Requests", *Proceedings of the 2008 Conference of the Center for Advanced Studies on Collaborative Research: Meeting of Minds*, pp. 304–318, New York, USA, 2008.

8. Herzig, K., S. Just and A. Zeller, "It's not a Bug, It's a Feature: How Misclassifi-

cation Impacts Bug Prediction", *35th International Conference on Software Engineering (ICSE)*, pp. 392–401, San Francisco, USA, 2013.

9. Jonsson, L., M. Borg, D. Broman, K. Sandahl, S. Eldh and P. Runeson, "Automated Bug Assignment: Ensemble-Based Machine Learning in Large Scale Industrial Contexts", *Empirical Software Engineering*, Vol. 21, No. 4, pp. 1533–1578, 2016.

10. Singh, V. and K. K. Chaturvedi, "Bug Tracking and Reliability Assessment System (BTRAS)", *International Journal of Software Engineering and Its Applications*, Vol. 5, No. 4, pp. 1–14, 2011.

11. Oflazer, K. and M. Saraçlar, *Turkish Natural Language Processing*, Springer, Cham, 2018.

12. Tabak, B., "Automated Assignment and Classification of Software Issues", 2021, `https://github.com/busrat/automated-software-issues`, accessed on June 10, 2023.

13. Gwet, K. L., "Computing Inter-Rater Reliability and Its Variance in the Presence of High Agreement", *British Journal of Mathematical and Statistical Psychology*, Vol. 61, No. 1, pp. 29–48, 2008.

14. Cohen, J., "A Coefficient of Agreement for Nominal Scales", *Educational and Psychological Measurement*, Vol. 20, No. 1, pp. 37–46, 1960.

15. Krippendorff, K., "Computing Krippendorff's Alpha-Reliability", *Computational Linguistics*, Vol. 37, No. 2, pp. 323–325, 2011.

16. Landis, J. R. and G. G. Koch, "An Application of Hierarchical Kappa-type Statistics in the Assessment of Majority Agreement among Multiple Observers", *Biometrics*, Vol. 33, No. 2, pp. 363–374, 1977.

17. Bulat, O., "Zeyrek: A Turkish Morphological Analyzer and Disambiguator", 2020, `https://github.com/obulat/zeyrek`, accessed on June 10, 2023.

18. Bird, S., E. Klein and E. Loper, *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*, O'Reilly Media, California, 2009.

19. Loria, S., "TextBlob", 2020, `https://textblob.readthedocs.io`, accessed on June 10, 2023.

20. McCallum, A. and K. Nigam, "BOW: A Toolkit for Statistical Language Modeling, Text Retrieval, Classification, and Clustering", *Journal of Machine Learning Research*, Vol. 2, No. 3, pp. 557–559, 2000.

21. Robertson, S., "Understanding Inverse Document Frequency: On Theoretical Arguments for IDF", *Journal of Documentation*, Vol. 60, No. 5, pp. 503–520, 2004.

22. Mikolov, T., K. Chen, G. Corrado and J. Dean, "Efficient Estimation of Word Representations in Vector Space", ArXiv:1301.3781 [cs], 2013.

23. Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay, "Scikit-Learn: Machine Learning in Python", *Journal of Machine Learning Research*, Vol. 12, No. 85, pp. 2825–2830, 2011.

24. Rehurek, R. and P. Sojka, "Gensim - Python Framework for Vector Space Modelling", *Masaryk University NLP Centre*, Vol. 3, No. 2, p. 2, 2011.

25. Zhou, Z.-H., *Ensemble Methods: Foundations and Algorithms*, CRC Press, Florida, 2012.

26. Odegua, R., "An Empirical Study of Ensemble Techniques (Bagging, Boosting, and Stacking)", *Proceedings of the Deep Learning IndabaX*, pp. 25–31, Nairobi,

Kenya, 2019.

27. Sanh, V., L. Debut, J. Chaumond and T. Wolf, "DistilBERT, a Distilled Version of BERT: Smaller, Faster, Cheaper, and Lighter", ArXiv:1910.01108 [cs], 2019.

28. Devlin, J., M.-W. Chang, K. Lee and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", ArXiv:1810.04805 [cs], 2018.

29. Liu, Y., M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer and V. Stoyanov, "RoBERTa: A Robustly Optimized BERT Pretraining Approach", ArXiv:1907.11692 [cs], 2019.

30. Clark, K., M.-T. Luong, Q. V. Le and C. D. Manning, "ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators", ArXiv:2003.10555 [cs], 2020.

31. Guven, Z. A., "The Comparison of Language Models with a Novel Text Filtering Approach for Turkish Sentiment Analysis", *ACM Transactions on Asian and Low-Resource Language Information Processing*, Vol. 22, No. 2, pp. 1–16, 2022.

32. Schweter, S., "BERTurk - BERT Models for Turkish", 2020, `https://github.com/stefan-it/turkish-bert`, accessed on September 17, 2022.

33. Dash, M. and H. Liu, "Feature Selection for Classification", *Intelligent Data Analysis*, Vol. 1, No. 1-4, pp. 131–156, 1997.

34. Brownlee, J., "How to Choose a Feature Selection Method for Machine Learning", 2019, `https://machinelearningmastery.com/feature-selection-withreal-and-categorical-data/`, accessed on September 17, 2022.

35. Dietterich, T. G., "Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms", *Neural Computation*, Vol. 10, No. 7, pp. 1895–1923,

1998.

36. Alpaydın, E., "Combined 5× 2 CV F Test for Comparing Supervised Classification Learning Algorithms", *Neural Computation*, Vol. 11, No. 8, pp. 1885–1892, 1999.

37. Wohlin, C., P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell and A. Wesslén, *Experimentation in Software Engineering*, Springer, Berlin, 2012.

38. Shokripour, R., J. Anvik, Z. M. Kasirun and S. Zamani, "A Time-Based Approach to Automatic Bug Report Assignment", *Journal of Systems and Software*, Vol. 102, pp. 109–122, 2015.

39. Yilmaz, O., "Turkish POS Tagger", 2015, `https://github.com/onuryilmaz/turkish-pos-tagger`, accessed on June 10, 2023.

40. Fan, Q., Y. Yu, G. Yin, T. Wang and H. Wang, "Where is the Road for Issue Reports Classification Based on Text Mining?", *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pp. 121–130, Toronto, Canada, 2017.

41. Kallis, R., A. Di Sorbo, G. Canfora and S. Panichella, "Ticket Tagger: Machine Learning Driven Issue Classification", *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 406–409, Cleveland, USA, 2019.

42. Kallis, R., A. Di Sorbo, G. Canfora and S. Panichella, "Predicting Issue Types on GitHub", *Science of Computer Programming*, Vol. 205, No. 3, p. 102598, 2021.

43. Izadi, M., K. Akbari and A. Heydarnoori, "Predicting the Objective and Priority of Issue Reports in Software Repositories", *Empirical Software Engineering*, Vol. 27, No. 2, p. 50, 2022.

44. Siddiq, M. L. and J. C. Santos, "BERT-Based GitHub Issue Report Classification", *IEEE/ACM 1st International Workshop on Natural Language-Based Software En-*

gineering (NLBSE), pp. 33–36, Pittsburgh, USA, 2022.

45. Afric, P., D. Vukadin, M. Silic and G. Delac, "Empirical Study: How Issue Classification Influences Software Defect Prediction", *IEEE Access*, Vol. 11, pp. 11732–11748, 2023.

46. Cabot, J., J. L. C. Izquierdo, V. Cosentino and B. Rolandi, "Exploring the Use of Labels to Categorize Issues in Open-Source Software Projects", *IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pp. 550–554, Montreal, Canada, 2015.

47. Zhou, Y., Y. Tong, R. Gu and H. Gall, "Combining Text Mining and Data Mining for Bug Report Classification", *Journal of Software: Evolution and Process*, Vol. 28, No. 3, pp. 150–176, 2016.

48. Pandey, N., D. K. Sanyal, A. Hudait and A. Sen, "Automated Classification of Software Issue Reports Using Machine Learning Techniques: An Empirical Study", *Innovations in Systems and Software Engineering*, Vol. 13, No. 4, pp. 279–297, 2017.

49. Terdchanakul, P., H. Hata, P. Phannachitta and K. Matsumoto, "Bug or Not? Bug Report Classification Using N-gram IDF", *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 534–538, Shanghai, China, 2017.

50. Qin, H. and X. Sun, "Classifying Bug Reports into Bugs and Non-Bugs Using LSTM", *Proceedings of the 10th Asia-Pacific Symposium on Internetware*, pp. 1–4, New York, USA, 2018.

51. Otoom, A. F., S. Al-jdaeh and M. Hammad, "Automated Classification of Software Bug Reports", *Proceedings of the 9th International Conference on Information Communication and Management*, pp. 17–21, New York, USA, 2019.

52. Köksal, Ö. and B. Tekinerdogan, "Automated Classification of Unstructured Bilingual Software Bug Reports: An Industrial Case Study Research", *Applied Sciences*, Vol. 12, No. 1, p. 338, 2021.

53. Herbold, S., A. Trautsch, F. Trautsch and B. Ledel, "Problems with SZZ and Features: An empirical study of the state of practice of defect prediction data collection", ArXiv:1911.08938 [cs], 2019.

54. Seaman, C. B., F. Shull, M. Regardie, D. Elbert, R. L. Feldmann, Y. Guo and S. Godfrey, "Defect Categorization: Making Use of a Decade of Widely Varying Historical Data", *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, pp. 149–157, New York, USA, 2008.

55. Goyal, A. and N. Sardana, "Machine Learning or Information Retrieval Techniques for Bug Triaging: Which Is Better?", *e-Informatica Software Engineering Journal*, Vol. 11, No. 1, pp. 117–141, 2017.

56. Xia, X., D. Lo, Y. Ding, J. M. Al-Kofahi, T. N. Nguyen and X. Wang, "Improving Automated Bug Triaging with Specialized Topic Model", *IEEE Transactions on Software Engineering*, Vol. 43, No. 3, pp. 272–297, 2016.

57. Anvik, J., L. Hiew and G. C. Murphy, "Who Should Fix This Bug?", *Proceedings of the 28th International Conference on Software Engineering*, pp. 361–370, New York, USA, 2006.

58. Bhattacharya, P., I. Neamtiu and C. R. Shelton, "Automated, Highly-Accurate, Bug Assignment Using Machine Learning and Tossing Graphs", *Journal of Systems and Software*, Vol. 85, No. 10, pp. 2275–2292, 2012.

59. Oliveira, P., R. M. Andrade, I. Barreto, T. P. Nogueira and L. M. Bueno, "Issue Auto-assignment in Software Projects with Machine Learning Techniques",

IEEE/ACM 8th International Workshop on Software Engineering Research and Industrial Practice (SER&IP), pp. 65–72, Madrid, Spain, 2021.

60. Yang, X., D. Lo, X. Xia, L. Bao and J. Sun, "Combining Word Embedding with Information Retrieval to Recommend Similar Bug Reports", *IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 127–137, Ottawa, Canada, 2016.

61. Hosseini, H., R. Nguyen and M. W. Godfrey, "A Market-Based Bug Allocation Mechanism Using Predictive Bug Lifetimes", *16th European Conference on Software Maintenance and Reengineering*, pp. 149–158, Szeged, Hungary, 2012.

62. Yang, G., T. Zhang and B. Lee, "Utilizing a Multi-Developer Network-Based Developer Recommendation Algorithm to Fix Bugs Effectively", *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, pp. 1134–1139, Gyeongju, Republic of Korea, 2014.

63. Su, Y., Z. Xing, X. Peng, X. Xia, C. Wang, X. Xu and L. Zhu, "Reducing Bug Triaging Confusion by Learning from Mistakes with a Bug Tossing Knowledge Graph", *36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 123–128, Melbourne, Australia, 2017.

64. Panda, R. R. and N. K. Nagwani, "Topic Modeling and Intuitionistic Fuzzy Set-Based Approach for Efficient Software Bug Triaging", *Knowledge and Information Systems*, Vol. 64, No. 11, pp. 3081–3111, 2022.

65. Karim, M. R., G. Ruhe, M. M. Rahman, V. Garousi and T. Zimmermann, "An Empirical Investigation of Single-Objective and Multiobjective Evolutionary Algorithms for Developer's Assignment to Bugs", *Journal of Software: Evolution and Process*, Vol. 28, No. 12, pp. 1025–1060, 2016.

66. Goodfellow, I., Y. Bengio and A. Courville, *Deep Learning*, MIT Press, Cambridge,

2016.

67. Xuan, J., H. Jiang, Z. Ren, J. Yan and Z. Luo, "Automatic Bug Triage Using Semi-Supervised Text Classification", *International Conference on Software Engineering and Knowledge Engineering*, pp. 209–214, San Francisco, USA, 2010.

68. Banitaan, S. and M. Alenezi, "TRAM: An Approach for Assigning Bug Reports Using Their Metadata", *Third International Conference on Communications and Information Technology (ICCIT)*, pp. 215–219, Beirut, Lebanon, 2013.

69. Mahajan, G. and N. Chaudhary, "Bug Classifying and Assigning System (BCAS): An Automated Framework to Classify and Assign Bugs", *Smart Systems: Innovations in Computing: Proceedings of SSIC*, pp. 537–547, Jaipur, India, 2022.

70. Aljarah, I., S. Banitaan, S. Abufardeh, W. Jin and S. Salem, "Selecting Discriminating Terms for Bug Assignment: A Formal Analysis", *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, pp. 1–7, New York, USA, 2011.

71. Sureka, A., "Learning to Classify Bug Reports into Components", *Objects, Models, Components, Patterns: 50th International Conference*, pp. 288–303, Prague, Czech Republic, 2012.

72. Alenezi, M., K. Magel and S. Banitaan, "Efficient Bug Triaging Using Text Mining.", *Journal of Software*, Vol. 8, No. 9, pp. 2185–2190, 2013.

73. Peng, X., P. Zhou, J. Liu and X. Chen, "Improving Bug Triage with Relevant Search", *International Conference on Software Engineering and Knowledge Engineering*, pp. 123–128, Pittsburgh, USA, 2017.

74. Hernández-González, J., D. Rodriguez, I. Inza, R. Harrison and J. A. Lozano, "Learning to Classify Software Defects from Crowds: A Novel Approach", *Applied Soft Computing*, Vol. 62, No. 1, pp. 579–591, 2018.

75. Sharma, M., A. Tandon, M. Kumari and V. Singh, "Reduction of Redundant Rules in Association Rule Mining-Based Bug Assignment", *International Journal of Reliability, Quality and Safety Engineering*, Vol. 24, No. 06, p. 1740005, 2017.

76. Sajedi-Badashian, A. and E. Stroulia, "Vocabulary and Time Based Bug-Assignment: A Recommender System for Open-Source Projects", *Software: Practice and Experience*, Vol. 50, No. 8, pp. 1539–1564, 2020.

77. Hall, M., E. Frank, G. Holmes, B. Pfahringer, P. Reutemann and I. H. Witten, "The WEKA Data Mining Software: An Update", *ACM SIGKDD Explorations Newsletter*, Vol. 11, No. 1, pp. 10–18, 2009.

78. Florea, A.-C., J. Anvik and R. Andonie, "Parallel Implementation of a Bug Report Assignment Recommender Using Deep Learning", *International Conference on Artificial Neural Networks*, pp. 64–71, Cham, Switzerland, 2017.

79. Gupta, S. and S. Gupta, "Bug Reports and Deep Learning Models", *International Journal of Computer Science and Mobile Computing*, Vol. 10, No. 12, pp. 21–26, 2021.

80. Mani, S., A. Sankaran and R. Aralikatte, "DeepTriage: Exploring the Effectiveness of Deep Learning for Bug Triaging", *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data*, pp. 171–179, New York, USA, 2019.

81. Feng, L., J. Senapati and B. Liu, "TaDaa: Real-Time Ticket Assignment Deep Learning Auto Advisor for Customer Support, Help Desk, and Issue Ticketing Systems", ArXiv:2207.11187 [cs], 2022.

82. Aktas, E. U., R. Yeniterzi and C. Yilmaz, "Turkish Issue Report Classification in Banking Domain", *28th Signal Processing and Communications Applications Conference (SIU)*, pp. 1–4, Gaziantep, Turkey, 2020.

83. Tunali, V., "Improved Prioritization of Software Development Demands in Turkish With Deep Learning-Based NLP", *IEEE Access*, Vol. 10, pp. 40249–40263, 2022.

84. Weiss, C., R. Premraj, T. Zimmermann and A. Zeller, "How Long Will It Take to Fix This Bug?", *Fourth International Workshop on Mining Software Repositories (MSR'07: ICSE Workshops)*, p. 1, Minneapolis, USA, 2007.

85. Zhang, H., L. Gong and S. Versteeg, "Predicting Bug-Fixing Time: An Empirical Study of Commercial Software Projects", *35th International Conference on Software Engineering (ICSE)*, pp. 1042–1051, San Francisco, USA, 2013.