AUTOMATED CLASSIFICATION OF TURKISH MOBILE APPLICATION REVIEWS

by

Güray Baydur

B.S., Computer Engineering, Bilkent University, 2018

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
2024

AUTOMATED CLASSIFICATION OF TURKISH MOBILE APPLICATION REVIEWS

APPROVED BY:

Assist. Prof. Fatma Başak Aydemir (Thesis Supervisor)	....
Prof. Tunga Güngör	....
Assist. Prof. Furkan Kıraç	....

DATE OF APPROVAL: 22.01.2024

ACKNOWLEDGEMENTS

I would like to express my gratitude to all those who have contributed to the completion of my master's thesis.

Finally, I would like to express my deepest gratitude to my family. I always felt their support and encouragement during this intense period of my life.

ABSTRACT

AUTOMATED CLASSIFICATION OF TURKISH MOBILE APPLICATION REVIEWS

App reviews provide insights into user behavior, ideas, complaints, and overall experience. Product Managers need rapid feedback to ensure customer satisfaction and long-term stay. Hence, app review analysis has emerged as a crucial topic to investigate. Nevertheless, the manual classification of such reviews is costly and demands significant effort. To reduce this cost and effort, automatic classification of app reviews is widely studied in many different languages. Studies focused on detecting Bug Reports, Feature Requests, User Experience, and User Ratings. To the best of our knowledge, our study is the first one that covers these labels and employs the automatic app review classification task in the Turkish language. To accomplish such a task, we manually crawled the Google Play Store and Apple App Store and obtained a dataset of size 5004 from different apps and categories. Additionally, according to the feedback from the industry, we introduce a new label, OPERATIONAL, to identify reviews that are not related to the technical functionality but to the operational processes of the application. Regarding classification, we utilize traditional machine learning techniques with various features, word embeddings, and BERT-based deep language models. Our results show that BERT-based models outperform traditional machine learning techniques. We also applied sampling techniques and observed improvements in the minority classes' performance. Furthermore, we explore the effect of generative artificial intelligence models, specifically GPT-3.5, on our task. We augmented data for a minority class and observed performance improvement with less data creation than sampling techniques. Additionally, we applied prompt engineering (Few-Shot and Chain-of-Thought) and fine-tuning with GPT-3.5 for a minority class. We observed high recall values with Chain-of-Thought prompting and the best F_1-score via fine-tuning.

ÖZET

TÜRKÇE KULLANICI YORUMLARININ OTOMATIK SINIFLANDIRILMASI

Uygulama yorumları, kullanıcıların davranışı, fikirleri, şikayetleri, genel deneyimleri hakkında bilgi sağlamaktadır. Ürün yöneticileri, müşteri memnuniyetini ve uzun süreli kalışı sağlamak için hızlı geri bildirime ihtiyaç duymaktadırlar. Bu nedenle, uygulama yorum analizi, araştırılması önemli bir konu olarak karşımıza çıkmaktadır. Ancak bu yorumların elle sınıflandırılması maliyetli ve efor gerektiren bir süreçtir. Bu eforu ve maliyeti azaltmak adına, yorumların otomatik sınıflandırılması başka dillerde geniş kapsamda çalışılmıştır. Bu çalışmalar, yorumları Hata Raporu, Yeni Özellik İsteği, Kullanıcı Deneyimi ve Değerlendirmesi olarak sınıflandırmaya odaklanmıştır. Bizim çalışmamız, Türkçe dilinde uygulama yorumlarının otomatik sınıflandırma konusunu bu etiketlerle ele alan ilk çalışmadır. Çalışmamız kapsamında Google ve Apple Uygulama Mağaza yorumlarını tarayıp çeşitli kategorilerden ve uygulamalardan 5004 örneklik bir veri kümesi elde ettik. Bunlara ek, sektörden gelen geri bildirimleri dikkate alarak, iş süreçlerini dikkate alan Operasyonel isminde yeni bir etiket tanımladık. Yorumların sınıflandırılması sırasında geleneksel makine öğrenmesi ve kelime temsili yöntemlerinin yanı sıra BERT tabanlı derin dil modelleri kullanılmıştır. Sonuçlar, BERT tabanlı modellerin geleneksel makine öğrenmesi ve kelime temsili yöntemlerini geride bıraktığını göstermektedir. Az sayıda örnek içeren etiketlerdeki düşük performans, örnekleme yöntemleriyle iyileştirilmiştir. Bunun yanında, üretici yapay zeka modellerinin sınıflandırmaya olan etkisi GPT-3.5 ile denenmiştir. Düşük verili bir sınıfta örnekleme yöntemlerine kıyasla daha az veri arttırımı kullanılarak performansın arttığı gözlemlenmiştir. Son olarak, az örnekli bir etikette istem mühendisliği ve ince ayar teknikleri kullanılmıştır. Fikir dizisi yöntemi yüksek duyarlılık sergilerken, en yüksek F_1-Skor ince ayar ile elde edilmiştir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1. App Reviews

Mobility plays a crucial role in the technology era. Users benefit from mobile applications in smartphones, tablets, even smartwatches to do almost every task. In 2021, there are 1.96 million apps available for download in Apple App Store, while there are 2 million apps on the Google Play Store [1]. Google states that over 2 billion people visit Google Play Store to download apps in a month [2]. Application stores are reaching out to millions of users and they refer to a huge community. To exemplify, Apple App Store reaches users across 175 countries and regions [3].

Requirements for mobile applications in such platforms are mainly elicited during sessions with stakeholders. During these sessions, different experts from various domains share their ideas, concerns, and priorities. They are expected to provide all the details from their perspective and make clear explanations. The ultimate goal is to reach a consensus for the final version of the application.

After gathering requirements, the application is implemented by developers aligned with the agreed scope and functionalities [4]. However, due to the fast evolution of software, obstacles may occur, causing severe application problems such as security risks to mitigate, bugs to fix and others [5]. Moreover, users may require new app functionalities to improve their experience [6]. Such requests and information are needed to be monitored by application providers to take necessary actions rapidly. At this point, the primary communication source between users and developers is app store reviews [7]. Al-Subaihin et al. [8] highlight that these reviews provide insights to developers about how well the application performs and handles user requests. Hence, detailed studies are conducted by many researchers to classify the reviews into relevant topics such as feature requests, bug reports, user experiences, and text ratings [9]

1.2. Motivation and Objectives

The need for app review analysis is significant in terms of app success. A recent survey on users' reasoning behind installing and deleting applications demonstrates that user reviews play an important role when they make their decision [10]. Another study points out the influence of app reviews [11]. Moreover, the impact of app reviews is investigated in many studies [12, 13].

App review classification problem is focused on many languages such as English [14], Italian [15] and Indonesian [16]. In Turkish, reviews are studied under user perceptions [17], user opinion [18] and sentiment analysis [19, 20]. But to the best of our knowledge, none of the studies focus on the automatic classification of app reviews. Since Turkish is a rich, agglutinated language that is spoken by more than 75 million people [21] and considering there is no previous work on this specific task in the Turkish language, this study will focus on the automatic classification of Turkish user reviews from app stores. In this thesis, our main objective is to apply the state-of-art methods to categorize and hence provide insights about Turkish app reviews.

1.3. Contributions

It is important to note that contributions mentioned in this thesis are pioneering in the Turkish language. Our primary contributions are as follows:

(i) We created and shared a brand new dataset for the Turkish app review classification problem. For this, we crawled the two most used app stores (Apple App Store and Google Play Store) and obtained 5004 app reviews. Reviews are from six categories: News, E-Commerce, Education, Health and Fitness, Food & Grocery, and Finance. We annotated the dataset using an annotation guideline based on previous studies. Each review in our dataset can have multiple labels.

(ii) In addition to the labels commonly used in previous studies, namely Feature Request, Bug Report, User Experience, and Rating, according to feedback from industry, we added a new informative label called OPERATIONAL. This label is not related to the technical functionality of the application, yet is about business processes that the application follows, such as delivery problems for food-service applications, shipment issues for e-commerce applications, or pricing plans for health and fitness applications. We observed that this new label covers a significant number of reviews.

(iii) The automatic classification of app reviews is accomplished by first utilizing traditional machine learning techniques with manually extracted features, well-known word embeddings. We use different types of classifiers to identify the relevant category of the review. We compared those techniques with each other and reported the results.

(iv) In addition to traditional machine learning techniques, BERT-based deep language models are used to observe the effects on our classification task. We employed several BERT models based on different settings. We analyzed the results of each setting and compared the results with traditional machine learning techniques.

(v) Finally, to observe the effects of trending generative artificial intelligence models on our classification task, we experimented data augmentation to improve the performance for a minority class, as well as prompt engineering and fine-tuning techniques to compare the effort and analyze the trade-off between each method.

1.4. Outline

This thesis is structured as follows. Chapter 2 reviews the related work. Chapter 3 represents the dataset. Chapter 4 explains the approach used. Chapter 5 discusses the results in detail. Finally, Chapter 6 gives a summary of the work, possible extensions and concludes.

2. RELATED WORK

This section discusses critical works from different areas, focusing specifically on classification problems in RE, app reviews, NLP for RE, Turkish NLP, and app review related works.

2.1. Classification problems in Requirements Engineering

Classification is one of the most significant tasks in Requirements Engineering [22]. Requirements can be classified as functional and non-functional requirements to distinguish the functionality and quality aspects of a system (i.e. reliability, maintainability, security, availability) [23]. For the categorization of requirements, Kurtanovic et al. [24] practiced the Support Vector Machines (SVM) to detect Functional Requirements (FR), Non-Functional Requirements (NFR), and their respective subcategories. They preferred the PROMISE repository in their study, which contains an imbalance of functional and non-functional requirements. To resolve this issue, they crawled Amazon product reviews and added them to the primary dataset to create a balanced distribution. This contribution led to better results in precision and recall rates over 90% in automatically identifying FRs and NFRs, including specific NFRs detection.

Non-functional requirements can be classified within themselves [25]. A review of machine learning algorithms for categorizing non-functional requirements is given by Binkhonain and Zhao [26]. For identifying and classifying non-functional requirements (NFRs) in requirements papers, they examined twenty-four machine learning (ML) techniques. This analysis demonstrates that 16 different machine learning algorithms are used, with Support Vector Machines (SVM) being the most prevalent approach. Moreover, the review identifies seven different performance metrics, with precision and recall as the widely utilized measures. However, open challenges still exist due to a lack of common datasets and a standard categorization and definition system for NFRs.

Another topic in this subject is distinguishing between requirements and non-requirements. Fahmi and Siahaan [27] approach this by focusing on non-requirements as noise. Their custom classification model is built upon the semantic features of non-requirement texts. They conduct a comparative analysis of the five traditional machine learning algorithms: Naïve Bayes (NB), Random Forest (RF), Decision Tree, Support Vector Machine (SVM), and k-Nearest Neighbor (kNN). The objective is to find which method outperforms others for non-requirements classification. Analysis results indicate that the SVM model is the most effective model, with an average accuracy of 96%. Considering the same subject, Le et al. [28] study an innovative method to detect requirement statements for construction contract documents automatically. Their approach utilizes supervised machine learning to develop a binary text classifier for categorizing requirement and non-requirement texts. They evaluate the results conducted on a dataset containing 100 construction contract documents. They report their achievement with an accuracy rate of 95%.

To consider other aspects of requirement classification problem, Sommerville [29] approaches requirement statements from various levels. He used the term user requirements to refer to the broadscaled, abstract requirements, while system requirements denote what the system should achieve. Lauesen [30] classify software requirements by following what the specifications should include and what the software aims to accomplish, emphasizing especially the goal-design level. Wiegers [31] investigates the Product, User and Business perspectives of the requirement classification task.

Recently, the classification of user feedback is receiving attention from many researchers [22]. Tavakoli et al. [32] utilize ML and NLP techniques to identify useful information from user feedback. They discriminate comments related to requirements from any other comments. Developer feedback is another area studied by Nazar et al. [33] where authors reviewed and summarized various data sources such as issues for bugs, mailing lists, and developer forums to extract requirements-related information. We further discuss application reviews in following section.

2.2. App Reviews

App reviews provide extensive insights as well as better visibility and overall interest for an application [34]. Users benefit from reviews to choose between similar applications [9]. Not only users are referring to such reviews but also developers utilize such reviews for app success [9]. There are two reviews by Santos et al. [35, 36] that show the importance of app review classification for users and developers. Pagano and Maalej [7] point out app reviews contain information related to requirements in terms of bugs or issues. App reviews also include overall user experience [37], requests for improvements [38], and even new feature requests [39]. Based on these mentioned studies, our goal is to apply classification of app reviews into several categories that give insights for requirement engineering.

Considering classification of app reviews into different types, Maalej et al. [9] aim to distinguish app reviews into four: bug reports, feature requests, user experiences, and text ratings. The paper creates manually extracted features by using Bag of Words (BoW) and NLP concepts such as bigrams, lemmatization, tense information, stopword removal, and others. They experiment on reviews from different apps and use the Naive Bayes ML algorithm to make the classification. Their precision and recall values for all four review types are significant, ranging from 89% up to 99%. However, they also advise the careful selection of features and preprocessing steps to achieve significant results.

A recent study from Marlo et al. [40] investigate app reviews and match them with the bugs reported in issue trackers. Their proposed tool DeepMatcher first extracts word embeddings from both app reviews and bug reports and then calculates how similar they are by using cosine similarity. The results indicate their solution can assist developers with early bug detection, enhancements of bug reports with feedback, and finding similar bugs that are close to each other.

Chen et al. [41] demonstrate that app reviews can be noisy and sometimes irrelevant. To avoid such unnecessary data, they developed a tool called AR-Miner which uses machine learning and Latent Dirichlet Allocation (LDA) topic classification algorithm to mine descriptive user reviews instead of noisy and irrelevant ones. They used a large pool of user reviews in App markets. Another work on this topic is from Lu and Liang [42] where authors augment the reviews by replacing similar words which have similar word2Vec embeddings.

Another important aspect of app reviews is their benefits for competitor analysis. Dalpiaz and Parente [43] bring attention to the fact that app reviews can uncover requirements based on explicit comparisons of competing apps. Jin et al. [44] also study this problem and authors extract and compare sentences for the same feature from multiple reviews for competitor analysis.

App reviews can also contain information about user decisions regarding the app [45]. Kunaefi and Aritsugi [46] study different decisions such as buying, recommending, and rating to assist app success. Authors use manually extracted features and try to classify decisions accordingly. They outperformed the state-of-the-art models in terms of F1-score for discriminating argumentative and non-argumentative reviews. They also classify decision types in reviews.

2.2.1. Techniques Used For App Review Classification

Table 2.1 lists different techniques used for app review classification tasks in the literature, considering rule-based, traditional machine learning, deep learning, and hybrid methods. These techniques are widely studied standalone and together. Vu et al. [47] introduce a keyword-based solution called MARK for review analysis. Asghar et al. [48] uses a sentence-level emotion detection framework by analyzing the semantic and syntactic features of sentences from app reviews. Rana and Cheah [49] introduce a hybrid approach that leverages sequential patterns and Normalized Google Distance (NGD) to extract explicit and implicit aspects in application reviews for categorization.

Table 2.1. Techniques used for App Reviews in the literature.

Reference	**Year**	**Rule Based**	**Machine Learning**	**Deep Learning**
Vu et al. [47]	2015	✓		
Guzman et al. [50]	2015		✓	
Maalej et al. [9]	2016		✓	
Shah et al. [51]	2018		✓	✓
Aslam et al. [52]	2018			✓
Al-Hawari et al. [53]	2020	✓	✓	
Araujo et al. [54]	2020		✓	✓
Rustam et al. [55]	2020		✓	
Triantafyllou et al. [56]	2020		✓	
Hadi and Fard [57]	2022			✓
Zhou et al. [58]	2022			✓
Kaur and Kaur [59]	2023			✓

Most of the studies compare a technique with one another. Al-Hawari et al. [53] introduce associative classification algorithms to extract class association rules and traditional machine learning techniques. Rustam et al. [55] use traditional machine learning techniques with several text features to classify app reviews. Shah et al. [51] experiment with the Bag-of-Words (BoW) feature extraction method and compare the machine learning results with other models used in other studies. They benefit from Convolutional Neural Network (CNN) models and conclude that, due to low data, CNN does not perform better than the traditional ML model with the BoW method. Araujo et al. [54] investigate pre-trained language models for traditional machine learning techniques and report that the BoW model with careful text pre-processing still performs similarly to pre-trained language models. Aslam et al. [52] also utilize a CNN-based classifier by extracting textual and non-textual information as embedding along with the emotions from the app reviews. To compare performance, Guzman et al. [50] experiment with traditional machine learning classifiers in isolation and combination. They demonstrate that the performance of hybrid classifiers is better than single classifiers. Furthermore, a novel feature extraction method called DEVMAX.DF is introduced by Triantafyllou et al. [56]. They use these features to conduct the traditional machine learning models. They perform better than the other feature extraction methods. Hadi and Fard [57] study the pre-trained models' performance compared to traditional ML models. According to their findings, fine-tuned custom pre-trained models achieve the highest scores in all settings. Zhou et al. [58] introduce semantics with extensions to categorize app reviews with pre-trained models, and they perform better than the state-of-the-art baselines. Kaur and Kaur [59] leverage transfer learning with a Bidirectional Encoder Representation for Transformer (BERT) model and Recurrent-Convolutional Neural Network (RCNN). Their joint BERT-RCNN framework surpasses other state-of-the-art models. Based on these studies mentioned above, the prevalence of combining various techniques pioneers our approach.

2.2.2. Datasets Used For App Review Classification

We employed extensive research on datasets created for the app review classification task to understand the contribution of our work. Considering datasets, Gu and Kim [60] obtained one, which harnasses reviews for 17 widely used Android apps from Google Play Store across 16 various categories, social media and entertainment applications included. The dataset comprises 34,000 reviews, with 2,000 reviews for each app, which are manually labeled. The authors classified the reviews into five categories based on their guidelines: Aspect Evaluation (5,937), Praise (8,112), Feature Request (2,323), Bug Report (2,338), and Others (15,290).

Another set of data from Stanik et al. [15] consists of 6,406 app reviews gathered from the Google Play Store, and 10,364 tweets were filtered from a pool of 5 million English tweets. Three classes are used for classification: Problem Report, Inquiry, and Irrelevant. In numbers, Google Play Store contributed Problem Report (1,437), Inquiry (1,100), and Irrelevant (3,869) records, while the Twitter data contains 2,933 Problem Report records, 1,405 Inquiry records, and 6,026 Irrelevant records.

Lu and Liang [42] acquired a dataset where authors analyzed reviews from the two widely preferred apps, namely iBooks from the books category on the Apple App Store and WhatsApp from the communication category on the Google Play Store. After they obtained raw reviews from each category, they filtered out 4,000 reviews and manually classified them. The classification results in six different categories: Usability (432), Reliability (587), Portability (119), Performance (121), Functional Requirements (558), and Others (2,183).

Maalej et al. [9] furnished a dataset with 2,000 manually labeled reviews extracted from popular apps randomly selected across various categories— 1,000 from the Google Play Store and 1,000 from the Apple App Store. The authors provided a balanced distribution of 200 reviews for each star rating within each review in the dataset. Additionally, they included 2,400 more manually labeled reviews for three randomly selected Android apps and three iOS apps from the top 100 apps, totaling 400 reviews for each app. In total, the dataset was finalized with 4,400 labeled reviews, categorized into Bug Report (378), Feature Request (299), User Experience (737), and Rating (2,721).

Guo et al. [61] created another dataset, combining 1,500 app reviews manually labeled as User Action (428), App Problem (399), and Neither (673). Authors carefully select the reviews from a large pool of over 5.8 million samples covering 151 apps from the Apple App Store.

A dataset comprising reviews for three apps from the Apple App Store and four apps from Google Play was retrieved by Guzman et al. [50]. Sampling 260 reviews per app, the authors asked for the input of five annotators, sampling 260 reviews each app to label 1,820 reviews in total manually. The dataset includes seven categories: Bug Report (990), Feature Strength (644), Feature Shortcoming (1,281), User Request (404), Praise (1,703), Complaint (277), and Usage Scenario (593).

Table 2.2. Labels used for App Reviews in the literature.

Label	**Publication**
Bug Report	[6], [9], [15], [52], [50], [62], [63]
Feature Request	[5], [6], [9], [15], [52], [50], [62], [63], [46]
Informative	[9], [46], [64]
Praise	[6], [38], [52], [50], [46]
Rating	[9], [46]
Other Labels	[6], [9], [15], [52], [50], [62]
Operational	This work

The datasets mentioned above demonstrate that Google Play Store and Apple App Store are two primary data sources for the app review classification task. Studies show that although naming conventions are different, they all try to classify app reviews into similar categories like finding bugs, requesting new features, and extracting user emotions and intentions. Considering the labels, Table 2.2 summarizes that Bug Reports and Feature Requests are commonly used as well as Informative, Praise, and Rating. In this work, we align with the vastly preferred labels and choose Bug Report and Feature Requests. We also observe that the Informative label covers the user's experience with the application. Therefore, we include this kind of label in the User Experience. We consider positive and negative comments (Complaint, Praise) from the user together and put them inside the Rating label. Additionally, we introduce a new label, Operational, which is about operational processes of application, and we explain it in detail in the following chapters.

2.3. Turkish NLP and App Reviews

Turkish natural language processing is widely studied for over 25 years [65]. The studies include semantic and sentiment analysis, named entity recognition, term extraction, morphological analysis [66]. Considering Turkish NLP tools and resources, several libraries are developed such as Turkish NLP, Zemberek, Turkish Stemmer, Turkish Lemmatizer ve ITU Turkish NLP Pipeline [67]. These tools provide Turkish language preprocessing steps (lemmatization, stemming, stopword removal, dependency parsing, normalization) to assist NLP researchers. In addition to preprocessing, text classification is also a prevalent focus area. BERTurk [68] is an extension of the BERT model [69] which is based on deep bidirectional transformers to be used for such tasks as well as any other type of classification and NLP related tasks.

Table 2.3. Natural languages studied for App Reviews in the literature.

Language	**Publication**
English	[5], [6], [9], [15], [52], [50], [62], [63], [46], [70]
Chinese	[70], [71], [72], [73]
Italian	[63], [74]
German	[9], [63], [74]
Other Languages	[75]
Turkish	This work

Turkish app reviews have received attention in recent years [20,76]. Furthermore, app reviews are not only gathered from app stores but they can be also obtained from tweets [77]. To exemplify, a user opinion analysis study from Kebabci and Diri [18] investigate the reviews from municipality Twitter accounts. Ogul and Ercan [20] tried to classify hotel reviews from users by applying both dictionary-based methods and machine learning methods such as tf-idf features with Naive Bayes, SVM, and Random Forest. Machine learning methods perform better than dictionary-based methods. Sigirci et al. [19] focus on analyzing sentiment of Google App Store Reviews from more than 2 million reviews. They used BERTurk as their model and achieved significant results in terms of accuracy (over 80% accuracy in different train-test splits).

Table 2.3 lists different languages that are studied in the literature so far. It is important to note that although there exist several works related to Turkish app reviews, none of them emphasize automatic classification of the reviews.

3. DATASET

This chapter focuses on the dataset used in this thesis in detail. As mentioned in previous chapters and as Table 2.3 indicates, we have observed that the app review classification task is widely studied in many languages, such as English, Italian and German, and manually annotated datasets exist in those languages. To exemplify, Maalej et al. [9] crawled the web to classify English app reviews into Bug Reports, Feature Requests, User Experience, and Ratings. Nevertheless, the classification task with the same tags has yet to be studied in Turkish, and no dataset for such task exists. Hence, we have created a Turkish dataset and manually annotated it.

3.1. Data Collection

Apple App Store and Google Play Store are the most dominant mobile application stores among others as of the last quarter of 2023 [78]. Hence, we chose these app stores as our two primary data sources. As illustrated in Table 3.1, our dataset focuses on six major categories, one application from each. We select these applications according to their popularity in that category. We chose Trendyol, a Turkish e-commerce mobile shopping app, as our first application. It has an average rating of 4.6 and over 100 million downloads. Secondly, we select Duolingo, a language application for new learners. Duolingo has been downloaded over 500 million times from the Google Play Store and the Apple App Store combined and has an average rating of 4.7 stars as of October 2023. Our third choice is Yapı Kredi Mobil, the mobile banking application of Yapı Kredi Bank in Turkey. As of 2023, it has been downloaded over 15 million from the Apple App Store and the Google Play Store. The app has an average rating of 4.5 stars on the App Store and 4.3 stars on the Google Play Store. Yemeksepeti, one of Turkey's most popular food delivery apps, with over 15 million active users, was chosen as the fourth application. As of the last quarter of 2023, Yemeksepeti has been downloaded over 50 million times from the Google Play Store and the Apple App Store. The app has an average rating of 4.7 stars on the Google Play Store and 4.8

stars on the App Store. Our fifth app, Meditopia, is a popular meditation and sleep app with over 10 million downloads worldwide. As of October 2023, Meditopia has over 10 million downloads from the Google Play Store and the Apple App Store. The app has an average rating of 4.9 stars on the App Store and 4.8 stars on the Google Play Store. Finally, Bundle, a popular news aggregator app, was selected. It has over 10 million downloads. As of 2023, it has been downloaded over 10 million times from the Apple App Store and the Google Play Store. Bundle News averages 4.7 stars on the Apple App Store and 4.5 stars on the Google Play Store.

Table 3.1. The list of applications.

Name	**Category**	**Description**
Trendyol	E-commerce	Online shopping app
Duolingo	Education	Language learning app
Yapı Kredi Mobil	Finance	Mobile banking app
Yemeksepeti	Food & Grocery	Food delivery app
Meditopia	Health & Fitness	Meditation tool
Bundle	News	News and events aggregator

In order to retrieve the aforementioned app review data, we employ Python libraries designed for scraping the app stores [79,80]. We crawl both the App Store and Google Play Store. Our crawling concludes with a total of 10,190 reviews conducted in May 2022. The dataset covers reviews from the time range from May 11, 2017, to April 24, 2022.

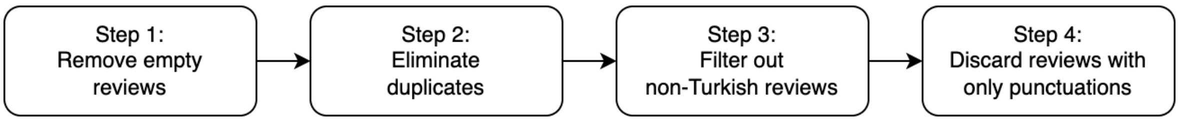
Step 1:
Remove empty reviews
Step 2:
Eliminate duplicates
Step 3:
Filter out non-Turkish reviews
Step 4:
Discard reviews with only punctuations

Figure 3.1. The filtering process.

In Figure 3.1, we can see the filtering process. We filter out non-Turkish reviews from our dataset. Additionally, we reduced our dataset size by eliminating empty reviews. We also observed that some reviews only contain standalone punctuations, which do not represent clear ideas about the content. Hence, we removed those items. For short reviews, we only eliminate duplicates because short reviews may indicate a sign of rating reviews. After this process, 8324 app reviews remained.

Table 3.2. Scraped applications and their raw and filtered review counts.

App Name	**Source**	**Raw Count**	**Included**
Trendyol	Google Play Store	988	775
	AppStore	732	578
Duolingo	Google Play Store	901	812
	AppStore	654	632
Yapı Kredi Mobil	Google Play Store	903	612
	AppStore	703	655
Yemeksepeti	Google Play Store	820	701
	AppStore	655	521
Meditopia	Google Play Store	765	731
	AppStore	1234	1022
Bundle	Google Play Store	855	632
	AppStore	980	653
Total	Both stores	10190	8324

3.2. Data Annotation

3.2.1. Annotation Schema

Based on our investigation into app review classification data and a thorough review of the existing literature, we have honed in on five specific labels: Bug report, Feature request, Rating, User experience and Operational. The first four of these labels have been extensively explored in previous studies [6, 9]. Seeking additional insights, we consulted a co-founder of a software consultancy company and, based on their feedback, introduced a novel label, OPERATIONAL, focusing on user discussions about operational processes rather than solely the application itself. With these five labels established, we have adopted a multi-label labeling framework for the reviews. Subsequently, we provide a detailed breakdown of each label's characteristics.

BUG REPORT reviews express issues about app performance, crashes, and erroneous behavior, which the developer should mitigate. It denotes a technical issue linked to the functionality of the application. For example, *"Son güncellemeden sonra uygulamaya giriş yapamıyorum. Ekran sürekli donuyor."* is a BUG REPORT indicating that user can not login to the application after software update. It is important to note that *"Çok fazla reklam çıkıyor lütfen reklam sayısını azaltın!"* is not a BUG REPORT since it is not related with the functionality of the app but with the advertisement amount.

FEATURE REQUEST reviews express missing functionality (e.g., provided by other apps) or missing content (e.g., in catalogs and games) and share ideas on improving the app in future releases by adding or changing features. *"Yemek siparişlerinde iptal seçeneğinin bulunmamasından birçok kez istemediğim şekilde ücreti vermek durumda kaldım. İşyeri hazırlık aşamasına geçtiğinden iptali yapamazken müşteri de iptal seçeneği olmadığından müşteri hizmetlerini beklemek zorunda kalıyor"* is an example of this label since user requests for order cancellation option from the app instead of using call center.

RATING reviews are simple text reflections of the numeric star rating. Ratings are less informative as they only include praise, dispraise, a distractive critique, or a dissuasion. *"Çok kötü ve yavaş acilen geliştirilmesi lazım diğer uygulamalara göre 1 puan çok bile!!!"* is a RATING review with negative sentiment. *"Uygulama zaten çok yaygın. Diyecek söz bulamıyorum. Memnunum. Emeğinize sağlık!"* is a positive RATING review example.

User experience reviews provide descriptions of the app in action. These are helpfulness, which captures use cases where the application has proven helpful, and feature information, which includes descriptions of application features and user interface. An example can be *"Düşüncelerimi özgürce ifade edebiliyorum, haberleri takip edebiliyorum ve insanlarla fikir alış verişi yapabiliyorum."*, where the app contributes user's daily life by following news and discussing with other people.

Operational reviews focus on aspects of the service or delivery process rather than the application's functionality itself. Examples include meal delivery problems, shipment delays, cancellation of orders, payment policies, or issues with the content provided (i.e., biased news) within the application. For instance, the statement *"Kargom hiçbir sebep belirtilmeden iptal oldu. Lütfen biri benimle iletişime geçebilir mi?"* exemplifies this label due to its association with a shipment problem. Similarly, *"Yemeğim 2 saat oldu gelmedi. Bir daha bu uygulamayı kullanmayacağım."* also falls under the Operational label, highlighting a delay in food delivery. Additionally, the statement *"Seçmememe rağmen karşıma sürekli yanlı haber sunan kanalları çıkarıyorsunuz."* indicates biased content in a news application, unrelated to the app's functionality.

It is important to note that our study allows reviews to have multiple labels. To exemplify, *"Kurye daha siparişimi getiremedi. Uygulamadan müşteri hizmetlerine bağlanmaya çalışıyorum hata veriyor, bu uygulama çok kötü kullanmayın."* states that delivery has not arrived yet, which indicates Operational issue. In the meantime, user complains about customer support section of the app is not working properly, pointing a Bug report. And finally, user states that application is worse than the others, pointing out a Rating review.

3.2.2. Annotation Process

In total, four annotators contributed to the annotation process. At least two annotators annotated each review. All annotators have a computer engineering background and voluntarily participated. Together, we built annotation guidelines that covered our label descriptions, annotation policies, and illustrative examples. We provide strategies to prevent fatigue, such as taking regular breaks. To construct the ground truth, we selected 20 reviews at random for each app–source pair, 200 reviews in total. We annotated this training set and discussed disagreements in a resolution session. As the next step, the author conducted training sessions for other annotators, reminding label definitions and guidelines. New annotators were tasked with annotating a subset of the training set, and we assessed agreement using Cohen's kappa [81] and Krippendorff's alpha [82] coefficients. This iterative process is repeated until the scores reach to 0.8 or higher, indicating substantial agreement. After training, new reviews were assigned to annotators, each receiving annotations from at least one other. Resolution sessions were conducted after each annotation cycle to resolve conflicts. Overall, we annotated 5004 reviews, which were limited by resource availability. Table 3.3 provides the list of annotated reviews per application and source.

Table 3.3. Annotated review counts.

App Name	**Category**	**Source**	**Count**
Trendyol	E-commerce	Google Play Store	417
		AppStore	417
Duolingo	Education	Google Play Store	417
		AppStore	417
Yapı Kredi Mobil	Finance	Google Play Store	417
		AppStore	417
Yemeksepeti	Food & Grocery	Google Play Store	417
		AppStore	417
Meditopia	Health & Fitness	Google Play Store	417
		AppStore	417
Bundle	News	Google Play Store	417
		AppStore	417
Total	All categories	Both stores	5004

3.3. Statistics of Dataset

The details of the dataset are as follows. Among 5004 annotated reviews, Figure 3.2 summarizes that 764 are categorized as Bug report, 363 as Feature request, 2511 as Operational, 2937 as Rating, and 2244 as User experience.

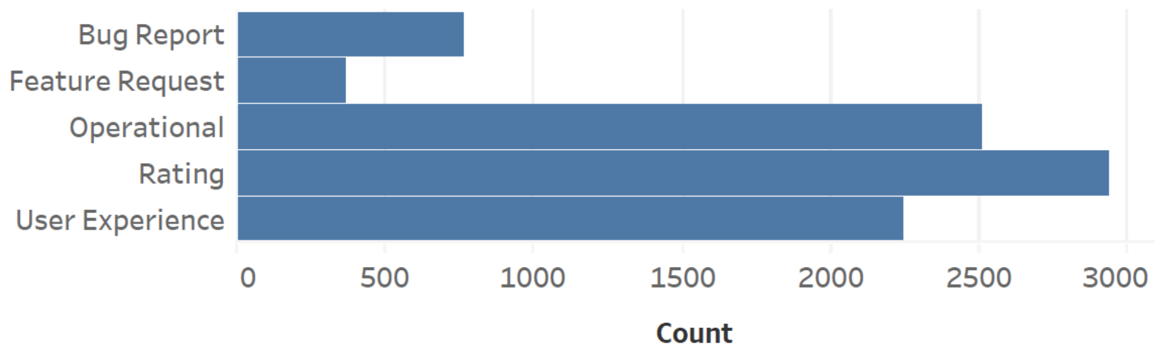

Figure 3.2. Label counts of annotated reviews

Rating reviews are the most common type of reviews in the dataset. It is followed by Operational, User Experience, Bug Report and Feature Request, respectively. In total, there are 8819 labels that are used in the annotation process. Figure 3.2 shows there exists data imbalance problem in this dataset. We try to overcome this issue in the following chapters.

Figure 3.3 shows the label count for each application. Overall, Meditopia contains most amount of labels (1608), it is followed by Duolingo, Yemeksepeti, Yapı Kredi Mobil, Bundle and Trendyol, respectively. In Bundle, Rating labels largest in number, same goes for Duolingo, Meditopia and Yapı Kredi Mobil. Trendyol and Yemeksepeti have the most Operational labels compared to other labels. Considering that both applications are using delivery service, users are commented more about the delivery process. Feature Request label have the least amount in all applications. It demonstrates that users did not emphasize requesting new features. Bug Reports are the second lowest label among all applications.

There are 3604 labels in Apple App Store and 5215 labels in Google Play Store. As seen in Figure 3.4, Apple App Store contains 1156 Operational (highest), 1026 Rating, 793 User Experience, 411 Bug Report and 218 Feature Request reviews. Google Play Store contains 1911 Rating (highest), 1451 User Experience, 1355 Operational, 353 Bug Report, 145 Feature Request reviews.

If the same labels in both platforms are compared with each other, Apple App Store passes the Google Play Store by Bug Report and Feature Request Reviews in terms of number. Google Play Store has more Operational, Rating, and User Experience reviews than Apple App Store.

Google Play Store has almost twice as many User Experience reviews. The number of Bug Report Reviews is close to each other on both platforms. The difference between Operational Reviews on both platforms is almost 200. Amount of rating reviews is much higher in Google Play Store. Google Play Store contains slightly more Feature Request reviews.

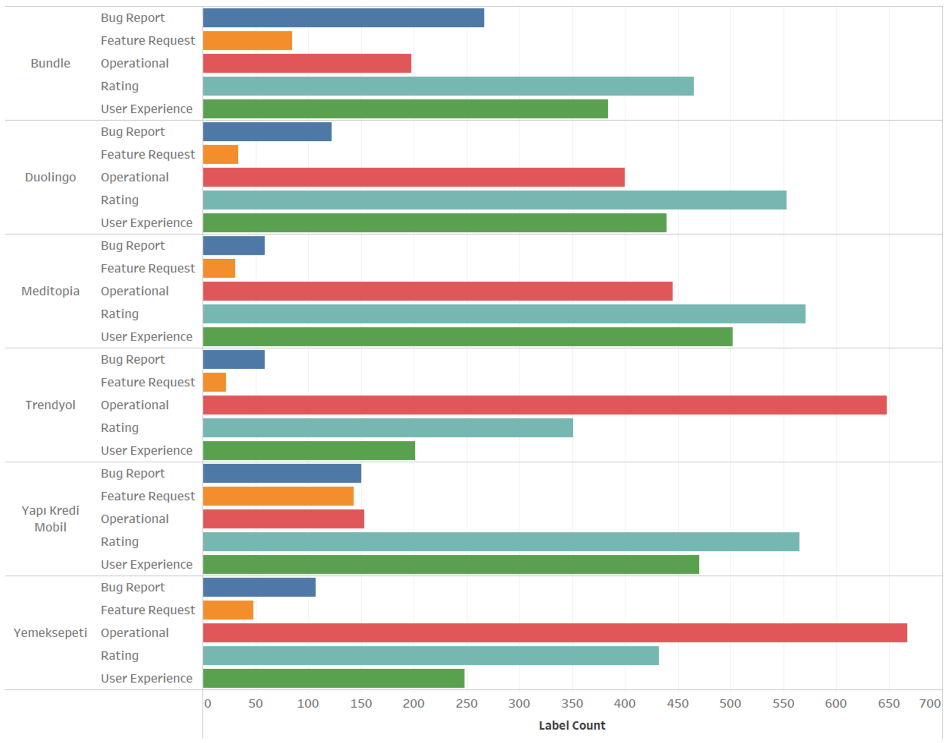

Figure 3.3. Label counts of each application.

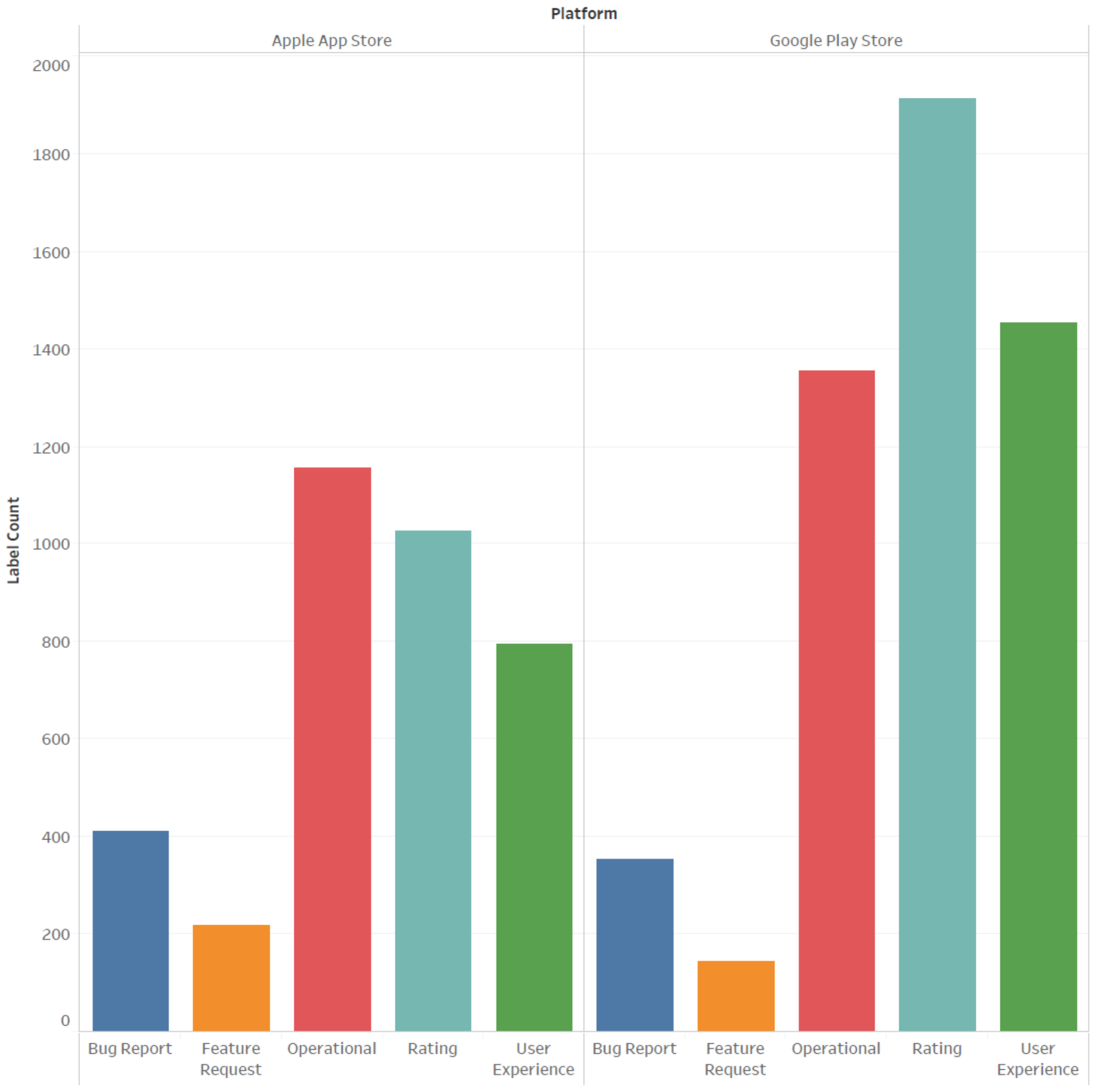

Figure 3.4. Label counts based on app stores.

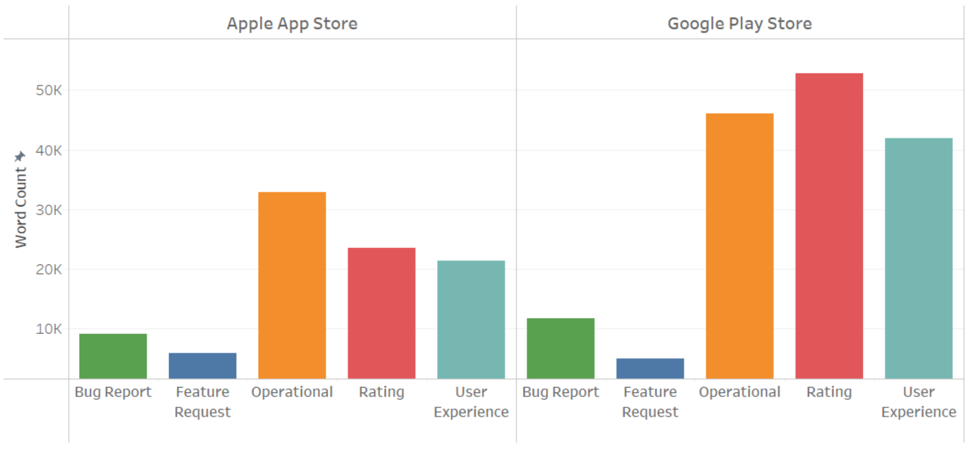

Figure 3.5. Word counts for each label for each platform.

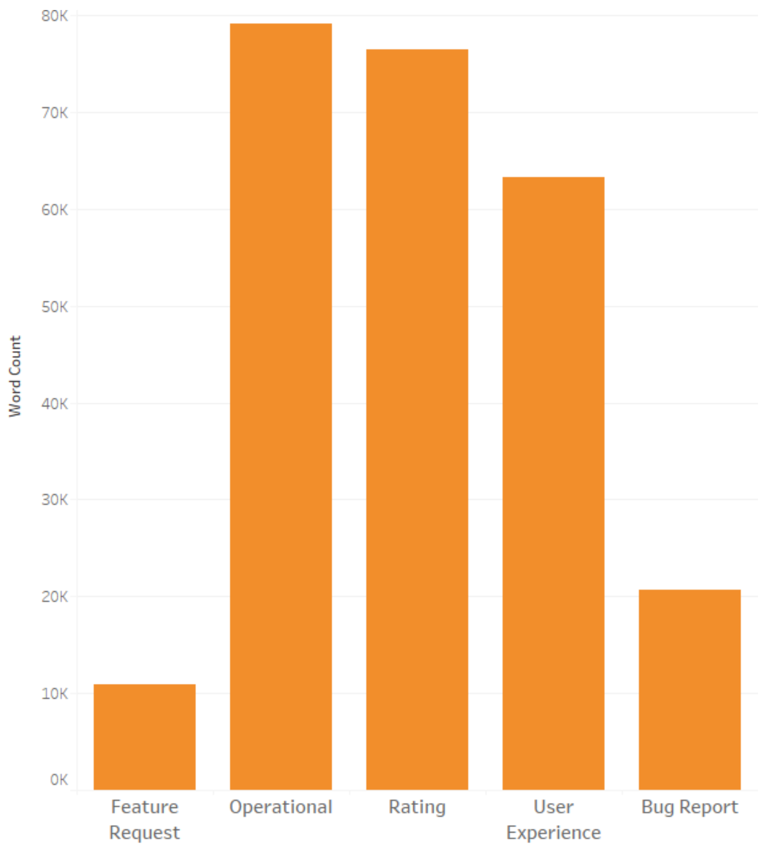

Figure 3.6. Word counts for each label.

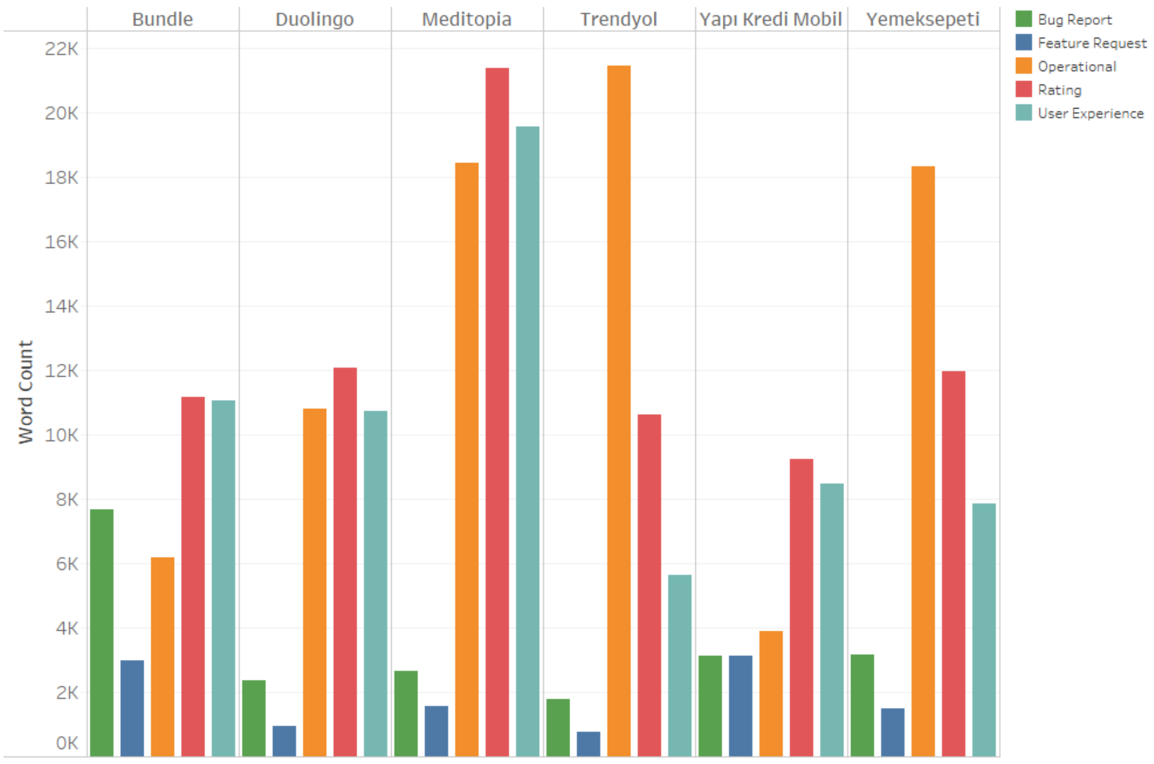

Figure 3.7. Word counts for each label for each app.

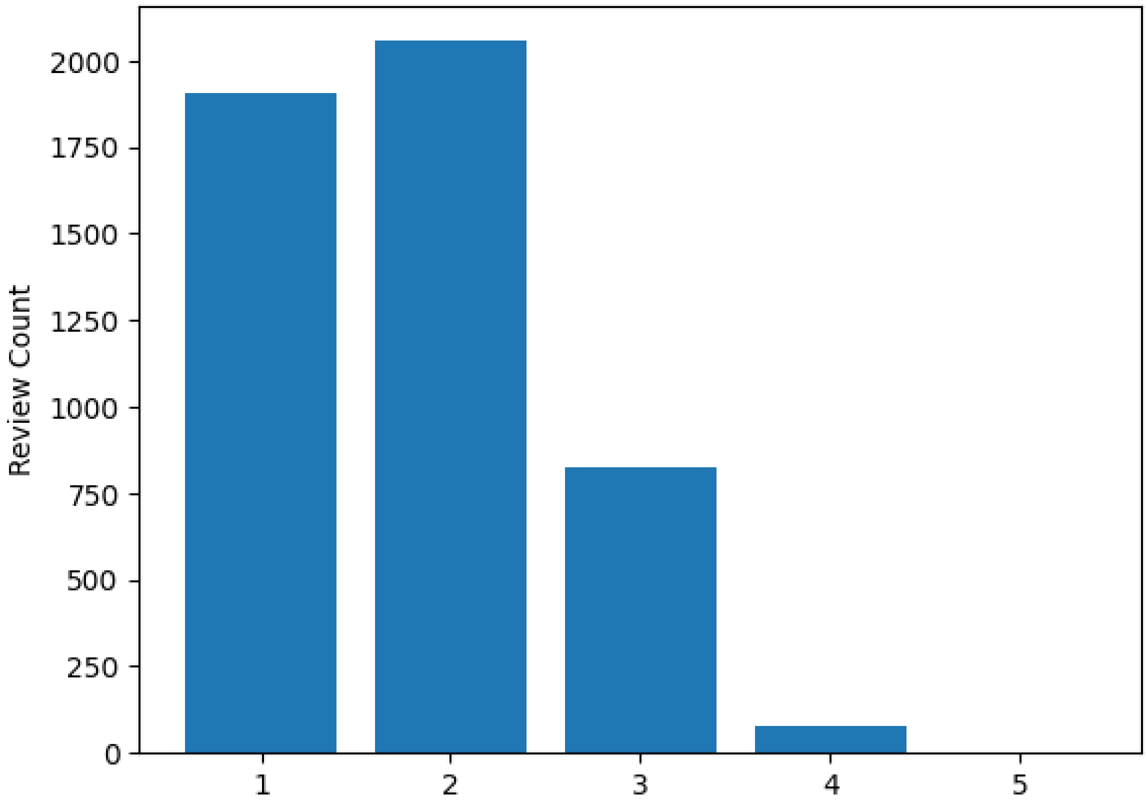

Figure 3.8. Distribution of number of labels.

4. APPROACH

To automatically classify application reviews, we use both traditional machine learning techniques and deep learning. First, we utilize standard feature extraction techniques, namely bag of word representations and term frequency-inverse document frequency. Afterward, we incorporate manually extracted features to improve performance. We also benefit from widely used word embedding representations to conduct comparative analysis. Our deep learning approach is based on transformer-based models, specifically BERT. We make use of several BERT models for comparison. In the following sections, we also explore the usage of large language models, GPT-3.5.

4.1. Traditional Machine Learning Techniques

4.1.1. Shallow Models

To categorize user reviews in mobile application stores, shallow machine learning techniques have been widely studied [83]. In our study, we also use shallow models to gain insights into the data and make comparisons with deep language models. Our experiments include three feature extraction methods: Bag-Of-Words, term frequency-inverse document frequency (Tf-Idf), and manually obtained feature-enhanced representations of the reviews, respectively. We employ Logistic Regression, Random Forest, Naive Bayes, and Support Vector Machines (Linear SVC) as classifiers. We also perform grid-search for optimal hyperparameters and k-fold cross-validation techniques.

We preprocess data by applying several NLP techniques. In particular, we perform the following steps:

(i) We lower the case of the review text.
(ii) We tokenize the review text using the Stanza library.
(iii) We remove the stopwords from the review text using the NLTK library.

After initial preprocessing, we utilized the bag-of-words (BoW) model for feature extraction. BoW is a feature extraction method for representing text as a numerical vector. It counts the number of time a word appears in a text and basically ignores the word order. However, BoW lacks the ability to capture the meaning or context of words. In our study, we prefer BoW as a baseline for feature extraction. For the BoW implementation, we utilized the CountVectorizer function from the sklearn [84]. We conduct experiments with different n-gram settings, where an n-gram stands for a contiguous sequence of n items from a given sequence of text. We ultimately found that unigrams (1-gram) produced the most optimal results. We share the results in subsequent sections.

As the second method for feature extraction, we utilized Term Frequency-Inverse Document Frequency (TF-IDF). It is computed by multiplying two values: TF, which calculates the frequency of a term in a document, and IDF, which incorporates how rare the word occurs across the entire corpus. TF-IDF gives importance to terms that are common in a text but relatively rare in the corpus, trying to highlight relevant and distinct words for a particular document against the more extensive collection. We used TfIdfVectorizer function from the sklearn library.

4.1.2. Feature Enhancements

Third, we employ other feature engineering methods to improve performance. We further apply the following to the data:

(i) We perform part-of-speech tagging using the Stanza library.
(ii) We put the information on whether a POS tag occurs in a review or not.
(iii) We compute POS-grams (unigrams & bigrams) for each review.
(iv) We put the information on whether a POS-gram occurs in a review or not.
(v) We lemmatize the text using the BOUN Morphological Analyzer.
(vi) We create unigrams and bigrams based on lemmas extracted.
(vii) We put the information whether a n-gram occurs in a review or not.

Our research in the literature demonstrates that metadata information contributes to the model's performance [9]. Hence, we extracted information that could be important for our classification task. We employ the following:

(i) We compute the number of characters, words, and sentences in the review.
(ii) We count the number of adjectives, nouns, verbs, adverbs, and numbers in the review and normalize it with review length.
(iii) We calculate the tree height of the review as well as the number of subtrees for the longest sentence in that review.

4.1.3. Results

We obtain the results for aforementioned methods in terms of Precision, Recall and F_1-score. Due to class imbalance problem, we also take weighted average of each metric for our ML model's performance among all labels. We also calculate weighted standard deviation to measure how much our metrics varies across different labels.

In the BoW representation case, Table 4.1 indicates that Operational and Rating labels have the highest F_1-scores with 84%. Both labels contain more samples than others, as seen in Figure 3.2, which is aligned with high performance. They are followed by Bug Report and User Experience labels with 72%. These two labels have the highest number of samples after the Operational and Rating labels. Feature Request suffers from a low amount of data, producing a low F_1-score of 43%. On weighted average, Linear SVC performs the best weighted average F_1-score with 77%. It has the lowest weighted standard deviation with 9%, indicating that performances for each label are closer to the weighted average, hence more precise results with lower variety and volatility. Due to significant Linear SVC performance, we observe linear relationships between the features of the reviews and the target labels. The second model that performs best is Logistic Regression, which supports our idea that linear relationships exist between the features and the target variable.

Table 4.1. Shallow machine learning with Count Vectorizer.

	Operational			**Bug Report**			**Feature Request**			**User Experience**			**Rating**			**Micro Average**			**Std Dev**
Model	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**F1**
Logistic Regression	0.85	0.78	0.82	0.73	0.71	**0.72**	0.44	0.32	0.37	0.75	0.68	0.72	0.86	0.81	0.83	0.80	0.74	0.77	0.10
Random Forest	0.85	0.65	0.73	0.65	0.56	0.60	0.46	0.25	0.32	0.77	0.66	0.71	0.88	0.70	0.78	0.81	0.64	0.71	0.10
Naive Bayes	0.84	0.83	**0.84**	0.59	0.70	0.64	0.33	0.36	0.34	0.66	0.77	0.71	0.81	0.83	0.82	0.74	0.78	0.76	0.11
Linear SVC	0.85	0.77	0.81	0.71	0.62	0.66	0.51	0.37	**0.43**	0.77	0.67	**0.72**	0.87	0.81	**0.84**	0.81	0.73	**0.77**	**0.09**

Considering TF-IDF representation, Table 4.2 demonstrates the F_1-scores across all labels. Operational and Rating labels reveal the highest F_1-scores at 85% and 83%, respectively. The high number of samples demonstrate better performance. Compared to TF-IDF, Bag-of-Words (BoW) representations show an improvement in the F_1-score for the Operational label but lower F_1-score for the Rating label. The difference between F_1-scores is noted as 1% which is relatively small. Following these, for TF-IDF, the User Experience and Bug Report categories achieve 72% and 71%, respectively. Analyzing the comparison with BoW, the Bug Report label exhibits a decrease in F_1-score with an amount of 1%, whereas the User Experience performance remains unchanged. Once again, the Feature Request label records the lowest F_1-score at 47%. Logistic Regression shows the best performance with a weighted average F_1-score of 78%. Logistic Regression also outputs the lowest weighted standard deviation at 9%, indicating lower variability across all classes. From above, we observe there is no significant difference between TF-IDF and BoW representations.

Table 4.3 presents the F_1-scores for each label to see the outcomes of enhanced features. Operational and Rating labels display the highest F_1-scores at 87% and 85%, respectively. These scores outperforms previous representations for each label. Moreover, the User Experience and Bug Report labels follow those labels, achieving 74% and 73% of F_1-scores, respectively. These two labels outperform other two methods in F_1-score. Additionally, the Feature Request label displays the lowest F_1-score at 49% but the performance is increased compared to previous representations. Considering the weighted average, the Linear SVC model shows high performance with an 80% weighted average F_1-score. It also has the lowest weighted standard deviation at 9%, implying a smaller variation among all labels.

Overall, the findings indicate that our specialized feature engineering techniques (POS tags, POS-grams, lemmatized n-grams etc.) enhance performance compared to shallow models utilizing standard feature representations.

Table 4.2. Shallow machine learning with Tf-Idf Vectorizer.

	Operational			**Bug Report**			**Feature Request**			**User Experience**			**Rating**			**Micro Average**			**Std Dev**
Model	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**F1**
Logistic Regression	0.85	0.84	0.84	0.79	0.65	**0.71**	0.79	0.32	0.45	0.76	0.69	**0.72**	0.87	0.80	0.83	0.83	0.75	**0.78**	**0.09**
Random Forest	0.84	0.68	0.75	0.65	0.61	0.63	0.60	0.25	0.35	0.75	0.66	0.70	0.89	0.73	0.80	0.81	0.67	0.73	0.09
Naive Bayes	0.83	0.85	0.84	0.63	0.58	0.60	0.28	0.12	0.17	0.71	0.70	0.70	0.80	0.86	**0.83**	0.75	0.76	0.75	0.14
Linear SVC	0.85	0.84	**0.85**	0.78	0.65	0.71	0.76	0.34	**0.47**	0.76	0.67	0.71	0.89	0.79	0.83	0.83	0.74	0.78	0.09

Table 4.3. Traditional machine learning models with feature enhancements.

	Operational			Bug Report			Feature Request			User Experience			Rating			Micro Average			Std Dev
Model	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**F1**
Logistic Regression	0.87	0.85	0.86	0.80	0.66	0.72	0.80	0.35	**0.49**	0.78	0.70	**0.74**	0.88	0.82	**0.85**	0.84	0.76	**0.80**	**0.09**
Random Forest	0.76	0.74	0.75	0.64	0.63	0.63	0.57	0.34	0.43	0.71	0.71	0.71	0.88	0.73	0.79	0.77	0.70	0.73	0.09
Naive Bayes	0.83	0.79	0.81	0.53	0.76	0.63	0.32	0.75	0.45	0.68	0.74	0.71	0.87	0.76	0.81	0.76	0.76	0.75	0.09
Linear SVC	0.86	0.87	**0.86**	0.77	0.70	**0.73**	0.78	0.36	0.49	0.78	0.69	0.73	0.90	0.80	0.85	0.84	0.77	**0.80**	**0.09**

4.1.4. Sampling Results

As seen in Figure 3.2, Feature Request and Bug Report labels suffer from insufficient samples, reflecting the models' low performance. We employ oversampling and undersampling techniques for Traditional Machine Learning with Feature Enhancements, Word Embeddings, and BERT models [85] to improve performance. We used *RandomOverSampler* and *RandomUnderSampler* utilities of scikit-learn library [86,87]. For oversampling, we choose random samples from the minority class and increase the number until it reaches the number of the majority class. For undersampling, we randomly choose samples from the majority class and extract those samples until we decrease the number to the amount of the minority class. We share the Precision, Recall, and F_1-score results for each label and model.

First, we employ oversampling to our best traditional machine learning approach in which we manually extracted our custom features as described in Section 4.1.2. Table 4.4 exhibits the results all labels. We observe that the best F_1-score of the Operational label is increased up to 87%. For the Bug Report label, the best F_1-score is moved up to 75%. The best F_1-score of the Feature Request label is increased up to 51%. The User Experience label's best F_1-score is also improved to 75%. The best F_1-score of the Rating label is reported as 86%. Overall, we noted an improvement in the weighted F_1-score and weighted standard deviation. This indicates that oversampling enhances the performance of our technique.

Regarding undersampling, Table 4.5 lists the results. Except the Bug Report and Feature Request labels, the best F_1-score of the labels are improved. On the other hand, since the labels with low amount of samples decreases F_1-scores significantly, the best weighted F_1-score is decreased down to 79%. Additionally, weighted standard deviation is increased which demonstrates further distribution of F_1-scores from the weighted mean with less precise results.

Table 4.4. Feature enhancements with oversampling.

	Operational			Bug Report			Feature Request			User Experience			Rating			Micro Average			Std Dev
Model	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**F1**
Logistic Regression	0.88	0.86	**0.87**	0.78	0.70	0.74	0.69	0.41	**0.51**	0.79	0.71	**0.75**	0.88	0.84	**0.86**	0.84	0.78	**0.81**	**0.08**
Random Forest	0.76	0.74	0.75	0.57	0.64	0.60	0.35	0.41	0.38	0.70	0.70	0.70	0.86	0.70	0.78	0.74	0.69	0.72	0.09
Naive Bayes	0.84	0.79	0.81	0.58	0.72	0.64	0.31	0.55	0.40	0.68	0.73	0.71	0.86	0.78	0.82	0.76	0.76	0.76	0.10
Linear SVC	0.87	0.87	**0.87**	0.79	0.72	**0.75**	0.79	0.38	**0.51**	0.79	0.70	0.74	0.90	0.82	**0.86**	0.85	0.78	**0.81**	**0.08**

Table 4.5. Feature enhancements with undersampling.

	Operational			**Bug Report**			**Feature Request**			**User Experience**			**Rating**			**Micro Average**			**Std Dev**
Model	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**F1**
Logistic Regression	0.87	0.88	0.87	0.45	0.78	0.57	0.22	0.82	**0.35**	0.80	0.76	**0.78**	0.87	0.84	0.85	0.79	0.82	**0.79**	0.12
Random Forest	0.75	0.74	0.75	0.45	0.69	0.55	0.18	0.64	0.28	0.71	0.71	0.71	0.87	0.71	0.78	0.73	0.71	0.71	**0.11**
Naive Bayes	0.84	0.79	0.81	0.45	0.84	**0.59**	0.18	0.84	0.30	0.69	0.75	0.72	0.86	0.76	0.81	0.75	0.78	0.75	**0.11**
Linear SVC	0.88	0.89	**0.88**	0.33	0.86	0.48	0.17	0.88	0.29	0.80	0.70	0.75	0.90	0.84	**0.87**	0.79	0.82	0.78	0.15

4.2. Word Embeddings

We employ word embeddings as another technique to apply feature engineering. Word embeddings correspond to numerical vector representations of words in a document [88]. They incorporate syntactic and semantic relationships of words in a corpus and, hence, provide more enhanced features [89]. They are widely used in the literature and are still being evolved [90]. In this work, we utilize well-known word embeddings, which are Word2Vec, GloVe, FastText, and ELMo.

Word2Vec is a prevalent word embedding technique where each word representation includes contextual and syntactical information from a text corpus [91]. Word2Vec vector is constructed by a neural network model based on two primary tasks: Skip-gram and Continous Bag-of-Words (CBOW). Skip-gram predicts the context given the current word, whereas CBOW predicts the current word given the context. In our study, we use Turkish Word2Vec embeddings [92] that are trained on Turkish Wikipedia pages.

Another essential word embedding is GloVe (Global Vectors for Word Representation). The idea behind this word embedding is that words with similar meanings are more likely to be close to each other in the same contexts [**?**]. GloVe constructs a log-bilinear regression model to obtain semantic information between words. We use Turkish GloVe embedding [93], which is uncased and contains 253.000 words in its corpus with a 300-dimensional vector.

We also use FastText, which improves word embeddings by investigating words as character n-grams, allowing us to handle out-of-vocabulary words and morphologically variations of them [94]. We obtained the pre-trained FastText embeddings from official website which is trained on Common Crawl and Wikipedia [95]. The model is trained on CBOW with position-weights, with a window of size 5 and 10 negatives. It has 300-dimensions. The character n-grams's length is 5 during training.

Finally, we conduct experiments with ELMo (Embeddings from Language Models) word embeddings. It captures the representation of a word by focusing its surrounding, hence learns the contextual information [96]. It uses deep bidirectional language model to construct the word embeddings. The difference from aforementioned word embeddings is that ELMo does not generate vector representations with fixed size. For that reason, for each review, we get ELMo word embeddings and calculate the sentence embedding by first padding each word embedding with zero to the longest word embedding length in that review and then calculate the average of all word embeddings. We use Turkish ELMo embeddings from Che et al. which is trained on Common Crawl and Wikipedia data [97].

4.2.1. Results

We apply the same machine learning algorithms used in previous sections with the word embeddings we acquired. In this section, we evaluate the performance of each word embedding. The comparison between different techniques will be discussed in the following sections.

Table 4.6 shows the results for Word2Vec embeddings. F_1-scores of Operational (78%) and Rating (79%) labels are better than other labels. One possible reason for this could be the large number of samples for these labels compared to other labels as can be seen in Figure 3.2. User Experience has the third largest samples and it has the F_1-score 69%. It is followed by Bug Report (56%) and Feature Request (36%). On micro average, Logistic Regression model performs best compared to others with a F_1-score of 71%. It also demonstrates low standard deviation compared to other algorithms (10%) which indicates more precise and significant results for this word embedding.

Table 4.6. Traditional machine learning models with Word2vec embedding.

	Operational			Bug Report			Feature Request			User Experience			Rating			Micro Average			Std Dev
Model	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**F1**
Logistic Regression	0.78	0.77	**0.78**	0.45	0.75	**0.56**	0.24	0.67	**0.36**	0.67	0.69	0.68	0.85	0.71	0.77	0.72	0.72	**0.71**	**0.10**
Random Forest	0.75	0.78	0.76	0.70	0.34	0.46	0.64	0.19	0.29	0.71	0.67	**0.69**	0.78	0.78	0.78	0.74	0.69	0.70	0.12
Naive Bayes	0.82	0.15	0.26	0.41	0.61	0.49	0.11	0.96	0.19	0.46	1.00	0.63	0.59	0.99	0.74	0.59	0.72	0.53	0.21
Linear SVC	0.75	0.81	**0.78**	0.41	0.80	0.54	0.10	0.97	0.18	0.56	0.88	0.68	0.70	0.91	**0.79**	0.63	0.87	0.71	0.13

Table 4.7. Traditional machine learning models with FastText embedding.

	Operational			**Bug Report**			**Feature Request**			**User Experience**			**Rating**			**Micro Average**			**Std Dev**
Model	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**F1**
Logistic Regression	0.80	0.80	0.80	0.51	0.81	**0.62**	0.00	0.00	0.00	0.69	0.73	0.71	0.86	0.71	0.78	0.73	0.72	0.72	0.16
Random Forest	0.83	0.80	**0.81**	0.82	0.45	0.58	0.57	0.11	0.18	0.76	0.70	**0.73**	0.83	0.78	**0.80**	0.80	0.71	**0.74**	**0.13**
Naive Bayes	0.75	0.68	0.72	0.49	0.63	0.55	0.15	0.93	**0.26**	0.54	0.88	0.67	0.62	0.98	0.76	0.61	0.84	0.69	**0.11**
Linear SVC	0.64	0.96	0.77	0.00	0.00	0.00	1.00	0.01	0.03	0.86	0.03	0.05	0.71	0.94	0.80	0.68	0.59	0.50	0.36

Table 4.7 demonstrates FastText embeddings result. Compared to Word2Vec embedding scores, except for Feature Request label, FastText outperforms Word2Vec in terms of F_1-score. It aligns with the fact that FastText is an extension of Word2Vec by using character n-grams instead of words as tokens [94]. The F_1-score of the Operational label (81%) is better than the Rating label (80%). One possible reason for this could be the large number of words (larger character n-grams) for Operational label compared to the Rating label, as seen in Figure 3.6. User Experience has the third largest sample, with an F_1-score 73%. It is followed by Bug Report (62%) and Feature Request (26%). On micro average, the Random Forest model performs best compared to others with an F_1-score of 74%. Naive Bayes gives the lowest standard deviation, but it does not have better results except for Feature Request. There is only a difference of 2% between Random Forest and Naive Bayes. We also observe that Random Forest gives better results for labels with many samples than others.

Table 4.8 displays results for ELMo embeddings. We observe the best F_1-score in the Operational label with 80%. Rating label follows it with 75% along with User Experience label 67%. Bug Report and Feature Request labels again give the worst performance, possibly due to low data with 27% and 16%, respectively. Regarding standard deviation, Naive Bayes and Linear SVC give the lowest percentages with 16%. One possible reason for this could be that they were able to produce true positives in labels with the low amount of data compared to other classifiers. Although Logistic Regression outperforms other models in the F_1-score, since it could not predict any true positive for the labels with the low number of samples, it gave a high standard deviation. We will focus on this issue using the oversampling technique and report the results in the subsequent sections.

Table 4.9 showcases GloVe embedding results. Compared to previous embedding scores, the F_1-scores of the Operational (81%) and Rating (81%) labels are similar. Random Forest outputs the best results in all labels except the Operational label. The lowest standard deviation is obtained from the Random Forest label, which indicates that results are close to the weighted mean and more precise.

Overall, GloVe embeddings perform best across all labels compared to other embedding types. Before the experiments, we expected ELMo to be effective due to its ability to capture contextual information. However, we observe that ELMo can not even predict when it encounters labels with fewer words in its reviews. One possible reason for this could be the padding with zero and then taking the average of the word vectors might dilute the contextual information. For the labels with many words and more contextual information (Operational and Rating), ELMo performs similarly to the best models, which supports this idea.

Regarding FastText, it outperforms Word2Vec as expected due to its consideration of character n-grams instead of words as tokens. FastText challenges GloVe when labels with large samples (Operational, Rating, and User Experience) are investigated but fall behind in labels with less samples (Feature Request and Bug Report). When we focus on the differences between these two word embeddings, GloVe is only trained on Commoncrawl data, and FastText is trained both on Commoncrawl and Wikipedia data. One possible reason for FastText's low performance could be the involvement of Wikipedia data during training. Additionally, the co-occurrence matrix of GloVe might indicate better feature representations than character n-grams of FastText embeddings for Feature Request and Bug Report labels.

Table 4.8. Traditional machine learning models with ELMo embedding.

	Operational			Bug Report			Feature Request			User Experience			Rating			Micro Average			Std Dev
Model	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**F1**
Logistic Regression	0.78	0.81	**0.80**	0.00	0.00	0.00	0.00	0.00	0.00	0.64	0.70	**0.67**	0.80	0.71	**0.75**	0.65	0.65	**0.65**	0.25
Random Forest	0.68	0.74	0.71	0.67	0.08	0.14	0.25	0.01	0.03	0.60	0.56	0.58	0.70	0.75	0.72	0.65	0.61	0.60	0.20
Naive Bayes	0.51	0.99	0.67	0.16	0.99	**0.27**	0.07	1.00	0.14	0.45	1.00	0.62	0.60	0.97	0.74	0.48	0.99	0.62	**0.16**
Linear SVC	0.59	0.98	0.73	0.15	1.00	0.27	0.09	0.95	**0.16**	0.46	1.00	0.63	0.84	0.62	0.71	0.58	0.87	0.63	**0.16**

Table 4.9. Traditional machine learning models with GloVe embedding.

	Operational			**Bug Report**			**Feature Request**			**User Experience**			**Rating**			**Micro Average**			**Std Dev**
Model	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**F1**
Logistic Regression	0.80	0.83	**0.81**	0.32	0.65	0.43	0.00	0.00	0.00	0.71	0.71	0.71	0.86	0.71	0.78	0.72	0.71	0.71	0.18
Random Forest	0.82	0.79	0.80	0.79	0.56	**0.65**	0.62	0.32	**0.42**	0.75	0.72	**0.73**	0.85	0.78	**0.81**	0.80	0.73	**0.76**	**0.09**
Naive Bayes	0.88	0.27	0.41	0.43	0.75	0.55	0.32	0.58	0.41	0.51	0.97	0.67	0.59	0.99	0.74	0.63	0.74	0.60	0.14
Linear SVC	0.71	0.91	0.80	0.27	0.94	0.42	0.11	0.97	0.19	0.55	0.94	0.69	0.69	0.95	0.80	0.60	0.94	0.71	0.15

4.2.2. Sampling Results

We conduct experiments with oversampling and undersampling techniques for the word embeddings. Table 4.10 reports the oversampling results for the best performing word embedding for our task, which is Glove embeddings. When compared with no sampling case in Table 4.9, we observe that the best F_1-score of Operational label is improved up to 82% from 81%. The best F_1-score of the Bug Report label is remained the same at 65%. The best F_1-score of the Feature Request label is increased up to %43 from %42. The User Experience label is also improved up to 74% from 73%. The best F_1-score of the Rating label is remained the same at 81%. Overall, the best weighted F_1-score is remained the same at 76% after oversampling but weighted standard deviation is improved up to 8% from 7%. This indicates better distribution around the weighted mean and hence more precise results than no sampling case.

Considering undersampling, we note performance improvement in the best F_1-scores of the Operational and User Experience labels by 1% when compared with no sampling scenario. Nevertheless, the Bug Report, Feature Request and Rating labels' performance is decreased. On weighted average, the best F_1-score is decreased down to 74% and the best weighted deviation is increased up to 10% from 9%. When undersampling and oversampling techniques are compared, oversampling outperforms undersampling in almost all labels with the exception of the Operational and User Experience F_1-scores, where the results remained the same. One reason for worse performance of undersampling is that labels with a low amount of data could not train the model enough to learn and classify.

Table 4.10. Word embeddings with oversampling.

	Operational			**Bug Report**			**Feature Request**			**User Experience**			**Rating**			**Micro Average**			**Std Dev**
Model	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**F1**
Logistic Regression	0.80	0.83	**0.82**	0.49	0.76	0.60	0.24	0.70	0.36	0.71	0.72	0.72	0.84	0.73	0.78	0.74	0.76	0.74	0.10
Random Forest	0.81	0.80	0.81	0.69	0.61	**0.65**	0.52	0.37	**0.43**	0.74	0.74	**0.74**	0.85	0.75	0.80	0.78	0.73	**0.76**	**0.08**
Naive Bayes	0.88	0.27	0.41	0.42	0.75	0.54	0.27	0.62	0.37	0.50	0.97	0.66	0.59	0.99	0.74	0.62	0.74	0.59	0.14
Linear SVC	0.71	0.91	0.80	0.26	0.95	0.41	0.09	1.00	0.16	0.55	0.94	0.69	0.71	0.94	**0.81**	0.60	0.93	0.72	0.16

Table 4.11. Word embeddings with undersampling.

	Operational			**Bug Report**			**Feature Request**			**User Experience**			**Rating**			**Micro Average**			**Std Dev**
Model	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**F1**
Logistic Regression	0.80	0.83	**0.82**	0.45	0.82	**0.58**	0.21	0.89	0.34	0.69	0.71	0.70	0.86	0.70	0.77	0.74	0.76	0.73	0.11
Random Forest	0.80	0.80	0.80	0.41	0.82	0.55	0.22	0.90	0.35	0.73	0.76	**0.74**	0.89	0.69	**0.78**	0.75	0.76	**0.74**	**0.10**
Naive Bayes	0.87	0.27	0.41	0.42	0.78	0.54	0.52	0.21	0.29	0.51	0.96	0.67	0.59	0.99	0.74	0.63	0.73	0.59	0.15
Linear SVC	0.69	0.92	0.79	0.19	0.99	0.32	0.33	0.56	**0.41**	0.55	0.93	0.69	0.61	0.99	0.76	0.57	0.94	0.70	0.14

4.3. BERT-based Models

We employ Turkish versions of BERT, RoBERTa, and DistilBERT models from the transformers library provided by HuggingFace [98]. Figure 4.1 demonstrates the pipeline of our BERT-based models. The pipeline approaches the task as one vs all classification, meaning that distinct models are constructed for each category.

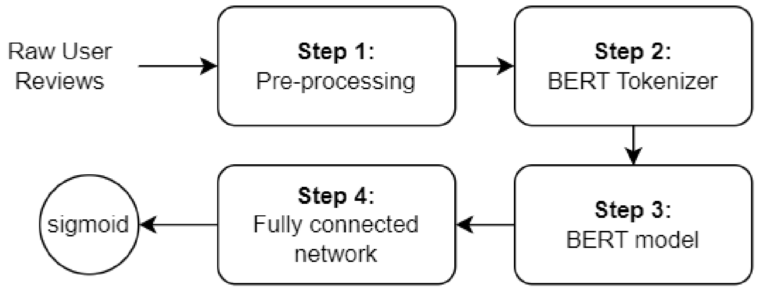

Figure 4.1. Steps of the proposed method.

Initially, we preprocess the data to obtain better tokenization performance. First, we extract emojis from the annotated reviews since the Turkish BERT models are not trained with such symbols. We observed that some reviews contain many new line and whitespace characters; hence, we converted them into a single whitespace. Since BERT-based models can be case sensitive, depending on the case sensitivity of each model, we convert the reviews into lowercase.

In Step 2, we input the pre-processed reviews to BERT Tokenizer in which we apply WordPiece algorithm [99]. During tokenization, the special tokens *[CLS], [SEP], and [PAD]* are added into relevant positions by BERT Tokenizer. Finally, the tokenizer returns input IDs with attention masks. We conduct experiments with different *max_ length* values (64 and 128), which performed poorly. At the end, we choose *max_ length* parameter value as 280 which gives the best model performance, since our reviews contain maximum of 280 tokens.

In Step 3, we utilize the BERT, RoBERTa, and DistilBERT models from HuggingFace [100] to create the hidden state vectors of tokenized reviews. Hidden state vectors are generated for every token after passing through the encoder layers. We ended up with the output vector of size 768 by taking the first token's hidden state.

In Step 4, we apply a fully connected classification layer to map the dimensions of the output vector into one dimension. Finally, we give one-dimensional vectors into sigmoid function to decide if the label suits for the input review.

During training, we fine-tuned our BERT model. We employed *Binary Cross Entropy* loss function and *2e-5* learning rate. We used both Adam [101] and AdamW [102] optimizers. AdamW optimizer resulted in better performance. We experimented with different numbers of epochs and calculated the precision, recall, and F_1-scores in each epoch. We found that the optimal number of epochs is 10 by observing F_1-score values in each epoch. Overfitting occurred when we applied large epoch values, and low epoch values caused worse performance.

4.3.1. Results

Table 4.12 demonstrates the results for the BERT-based models. Similar to previous sections, we observe that categories with many samples (Operational, Rating and User Experience) shows greater F_1-score values than categories with less samples (Bug Report and Feature Request). Rating label has the highest F_1-score value 89%, followed by Operational label with 88%. User Experience has the third highest F_1-score value with 81%. Feature Request follows it with F_1-score of 72%.

Overall, the best model can be deduced as BERT-128k-uncased, which outperforms other BERT-based models with greater F_1-score value in almost all labels. The only exception is Bug Report label, where the BERT-uncased model performed better with a higher F_1-score value of 77%. The lowest standard deviation values are obtained form BERT-uncased, BERT-cased, and BERT-128k-uncased models which shows that the results are more close to weighted mean and precise compared to other models. However, the best weighted average F_1-score value is obtained from the BERT-128k-uncased model which makes the model is the perfect choice for the optimal Turkish BERT-based model for our app review classification task.

To evaluate the results, we need to understand the details of the BERT models. BERT-uncased, BERT-cased, BERT-128k-uncased and BERT-128k-cased models has the final training corpus of size 35GB and 4.404.976.662 tokens [103,104]. The difference is that 128k-models contain a vocabulary of size 128.000, meaning that more unique and comprehensive set of tokens are learned and covered during the training phase. It is aligned with high performance of 128k-models compared to others.

The RoBERTa model is a modified version of the BERT model, where the model is trained with ten times more data than the original model [105]. It employs a dynamic masking pattern instead of static masking, full sentences without Next Sentence Prediction (NSP) loss, a larger Byte-Pair Encoding (BPE), and large mini-batches during training [105]. Despite these improvements, our results have shown that the RoBERTa model did not produce better results than other BERT models. One reason could be that the Turkish RoBERTa model is also trained with news websites where the context is not similar to the app review domain [106].

The DistilBERT model is a lighter, faster and cheaper implementation of the BERT model due to its fewer parameters and model size. The Turkish version of this model was trained on 7GB of the Turkish BERT-cased model [107] which corresponds to 20% of the original model. For the labels with large number of samples, DistilBERT only differs original BERT-models from 3% or 4%. Considering labels with few samples, in Bug Report, DistilBERT is at most only 3% behind other BERT models. It only performs much worse (from 9% up to 18%) in terms of F_1-score for Feature Request label, which demonstrates that model could not learn well due to low amount of examples.

Table 4.12. Performance metrics for BERT models.

	Operational			**Bug Report**			**Feature Request**			**User Experience**			**Rating**			**Micro Average**			**Std Dev**
Model	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**F1**
BERT-uncased	0.87	0.88	**0.88**	0.66	0.92	**0.77**	0.54	0.84	0.66	0.80	0.80	0.80	0.95	0.78	0.85	0.85	0.83	0.83	0.05
BERT-cased	0.87	0.87	0.87	0.63	0.93	0.75	0.74	0.70	**0.72**	0.80	0.79	0.80	0.90	0.86	0.88	0.84	0.84	0.84	0.05
BERT-128k-uncased	0.86	0.91	**0.88**	0.62	0.92	0.74	0.64	0.81	**0.72**	0.81	0.81	**0.81**	0.92	0.86	**0.89**	0.84	0.86	**0.85**	**0.05**
BERT-128k-cased	0.88	0.85	0.87	0.66	0.90	0.76	0.50	0.85	0.63	0.81	0.78	0.79	0.93	0.84	0.88	0.84	0.83	0.83	0.06
RoBERTa-uncased	0.82	0.83	0.82	0.66	0.82	0.73	0.52	0.84	0.64	0.78	0.77	0.77	0.90	0.80	0.85	0.81	0.80	0.80	0.05
DistilBERT-cased	0.83	0.84	0.84	0.67	0.84	0.74	0.52	0.56	0.54	0.80	0.75	0.77	0.87	0.83	0.85	0.81	0.80	0.80	0.07

4.3.2. Sampling Results

After oversampling for the BERT models, in Table 4.13, we observe performance improvement in all labels regarding the best F_1-scores, except the Operational label, where the best F_1-score remained the same. The best F_1-score of the Bug Report label increased up to 80% from 77%. The best F_1-score of the Feature Request label moved up to 78% from 72%. The User Experience label's best F_1-score value is increased up to 82% from 81% and Rating label's best F_1-score is also moved up to 90% from 89%. Before oversampling, we reported that the BERT-128k-uncased model outperformed other models with the best-weighted F_1-score and weighted standard deviation. When we applied oversampling, we noted that the best-performing model changed to BERT-cased with the best weighted standard deviation and the best-weighted F_1-score. Although the best-weighted F_1-score is not improved, the best standard deviation is reduced to 3%, indicating a much closer distribution to the weighted mean and more stable results. Our main goal was to improve the performance, especially for the labels with a low amount of data (Bug Report and Feature Request), and we achieved better performance for those labels.

We also applied undersampling to observe performance improvement. Our results in Table 4.14 demonstrates that undersampling did not produce better results than oversampling. Additionally, it performs worse than original BERT model without sampling in Rating, Bug Report and Feature Request labels, 1%, 2% , 24% decrease in the best weighted F_1-scores respectively. The best weighted F_1-score of Operational label is remained the same with 88%. The only improvement is in the best weighted F_1-score of User Experience label, where it moved up to 82% from 81%. Overall, the best weighted F_1-score average is reduced down to 83% and the best weighted standard deviation is increased up to 9% from 5%. To compare the results with oversampling, undersampling technique did not outperform oversampling in any of the labels. The best weighted F_1-score of Operational and User Experience label stayed the same. From the results, we deduce that undersampling did not produce better performance, especially for the labels with low number of samples.

Table 4.13. Performance metrics for BERT models with oversampling.

	Operational			**Bug Report**			**Feature Request**			**User Experience**			**Rating**			**Micro Average**			**Std Dev**
Model	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**F1**
BERT-uncased	0.87	0.86	0.87	0.87	0.73	0.79	0.78	0.73	0.75	0.77	0.86	0.81	0.94	0.83	0.88	0.86	0.83	**0.85**	0.04
BERT-cased	0.81	0.92	0.86	0.87	0.72	0.79	0.87	0.71	**0.78**	0.80	0.83	**0.82**	0.93	0.84	0.88	0.85	0.84	**0.85**	**0.03**
BERT-128k-uncased	0.85	0.91	**0.88**	0.81	0.78	**0.80**	0.80	0.62	0.70	0.76	0.87	0.81	0.92	0.85	0.88	0.84	0.86	**0.85**	0.05
BERT-128k-cased	0.87	0.89	**0.88**	0.88	0.72	0.79	0.80	0.66	0.72	0.86	0.73	0.79	0.90	0.89	**0.90**	0.88	0.82	**0.85**	0.05
RoBERTa-uncased	0.85	0.87	0.86	0.79	0.78	0.78	0.85	0.48	0.61	0.76	0.77	0.77	0.90	0.80	0.85	0.84	0.80	0.82	0.06
DistilBERT-cased	0.83	0.86	0.84	0.77	0.77	0.77	0.75	0.55	0.63	0.79	0.74	0.76	0.90	0.82	0.86	0.83	0.80	0.81	0.06

Table 4.14. Performance metrics for BERT models with undersampling.

	Operational			**Bug Report**			**Feature Request**			**User Experience**			**Rating**			**Micro Average**			**Std Dev**
Model	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**F1**
BERT-uncased	0.89	0.87	**0.88**	0.64	0.91	**0.75**	0.28	0.95	0.43	0.81	0.79	0.80	0.93	0.84	**0.88**	0.84	0.85	**0.83**	**0.09**
BERT-cased	0.83	0.91	0.87	0.60	0.95	0.73	0.27	0.96	0.42	0.88	0.70	0.78	0.92	0.83	0.87	0.83	0.84	0.82	0.10
BERT-128k-uncased	0.83	0.93	**0.88**	0.58	0.88	0.70	0.29	0.86	0.43	0.82	0.81	**0.82**	0.91	0.85	**0.88**	0.81	0.87	**0.83**	0.10
BERT-128k-cased	0.83	0.93	**0.88**	0.55	0.92	0.69	0.31	0.90	0.46	0.82	0.76	0.79	0.93	0.84	**0.88**	0.81	0.85	0.82	0.10
RoBERTa-uncased	0.84	0.86	0.85	0.50	0.92	0.65	0.33	0.89	**0.48**	0.79	0.75	0.77	0.95	0.74	0.83	0.81	0.80	0.79	**0.09**
DistilBERT-cased	0.85	0.85	0.85	0.47	0.87	0.61	0.19	0.82	0.31	0.77	0.73	0.75	0.90	0.81	0.85	0.79	0.81	0.78	0.12

4.4. Generative Artificial Intelligence Models for App Review Classification

Generative Artificial Intelligence, shortly Generative AI, is a subfield of artificial intelligence that produces unique content by augmenting human creativity and innovation [108]. Outputs created by Generative AI models are from various areas such as audio, text, image, video, and so on [109]. These models are trained with massive amounts of data and learn patterns to generate new instances of the same kind. In the last two years, the interest in Generative AI models has increased rapidly [110].

To address the impact of Generative AI models for our classification task, we utilize the Generative Pre-Trained Transformer 3.5 model, GPT-3.5, which has over 175 billion parameters [111]. Due to its large model size, GPT-3.5 can process intricate linguistic nuances and generate logical and contextually appropriate responses. In our study, we specifically used the gpt-3.5-turbo model [112] that was recommended by OpenAI at the time this study was conducted. It is important to note that it is the most cost-efficient model that is suggested and, hence, the most appropriate model for our study [113]. We followed the OpenAI's *Chat Completions* endpoint usage guidelines to generate responses [114].

During our study, we noticed that our data is highly imbalanced, which results in low performance of our models for the minority classes. To overcome this issue, we employed sampling techniques in the aforementioned sections. Nevertheless, we observed minor improvements in the weighted average F_1-scores and the weighted standard deviations. To further enhance the performance, we employ data augmentation with GPT-3.5. We mainly have two minority classes, Bug Report and Feature Request, but we were able to apply this technique to only one label due to budget limitations. The reason we choose Bug Report over Feature Request is that Bug Report contains more samples than Feature Request to augment data. The following section will explain the details and the results of this technique.

In addition to data augmentation, we apply prompt engineering method for our classification task. Prompt engineering is a practice of designing and crafting prompts for large language models (LLMs) to produce desired outputs [115]. Recently, it has gained popularity in many research areas and utilized in wide range of applications [116]. We employ different techniques of this method which are Few-shot and Chain-of-Thought Prompting. We further give details about the techniques used and share the results in the following sections. Again, since GPT-3.5 serves as paid endpoints, due to budget limitations, we only applied these techniques to only one minority class, Bug Report.

4.4.1. Data Augmentation

Our goal in this section is to augment minority class data (Bug Report) to observe its effects on our models' performance. For this purpose, we first split our data into train and test datasets. Then, we feed GPT-3.5 with our train data to augment one or more reviews from single review. For user input, formally user content in the OpenAI documentation, we cover each review with Extensible Markup Language (XML) tags to seperate from each other. For system instructions, namely system content, we experimented with several prompts to guide GPT-3.5 to generate similar Bug Report review. One example prompt was *"Yorum etiketi içinde verilen hata raporu yorumlarına benzer yorumlar üret. Üretilen yorumları yorum etiketi içinde ver."*, which simply tells the model to produce similar Bug Report reviews and output them properly between xml tags. We experienced that the model did not understand these instructions well as it produced responses for some reviews like customer support. *"En kısa zamanda iletişime geçeceğiz"* and *"Bu sorunla karşılaştığınız için üzgünüz."* were some of the replies which tells the customer to be contacted soon and sorry for the inconvenience. Hence, we detailed our prompts and finalized with the following: *"Verilen yorumları anlamları bozulmadan değiştirerek, cümleleri devrikleştirerek ya da benzer kelimeler kullanarak veri arttırımı uygula. Bir yorum için birden fazla veri arttırımı örneği üretebilirsin. Sonuçlar yorum etiketinin içinde olsun."*. This prompt gives clues to the model about how to augment the data by advising synonym usage and reordering part of speech.

Table 4.15. Augmented data example.

Review Content	**Review Type**
Yüklenme sorunları var.	Original
Yüklenme sorunları yaşıyorum.	Augmented
Uygulamanın yüklenmesi sorunlu.	Augmented
Uygulama yüklenirken sorun yaşıyorum.	Augmented

As can be seen in Table 4.15, our prompt resulted in consistent augmented reviews. Table 4.15 outputs the example of synonym usage and word order changes. It also produces new words to enrich the context. Overall, the augmentation with GPT-3.5 introduced 816 augmented Bug Report reviews. We obtained 358 reviews for Google Play Store and 458 reviews for Apple App Store. Considering the original 767 Bug Report reviews, the data augmentation created more than double of the original number of reviews. It is important to note that the total amount is still not enough to overcome data imbalance problem but due to budget limitations we could not proceed with further augmentation. We report the results for the overall 1583 Bug Report reviews.

4.4.2. Results

We share the results for the aforementioned methods that we used, which are traditional machine learning with feature enhancements, word embeddings and BERT-based models. Table 4.16 displays the results for traditional machine learning technique with manually crafted features. When we apply our feature engineering and add augmented data, we observed that F_1-score is increased for Logistic Regression from 72% to 73%. Similarly, F_1-score of LinearSVC is improved from 73% to 74%, when compared with the results with no augmentation.

Table 4.17 lists the results for our best performing word embedding, GloVe. It is observed that F_1-score is increased for Logistic Regression from 43% to 61%. For Linear LinearSVC classifier, the F_1-score is increased from %42 to %59. For Naive Bayes, F_1-score remained the same. For Random Forest, the F_1-score is slightly decreased from %64 to 61%.

For BERT-based models, we tokenized augmented data with our BERT Tokenizer and fed to our system. Table 4.18 presents the outputs for the experiment. We obtained performance improvement in terms of F_1-scores for all models, except BERT-128k-cased, which is decreased slightly from %74 to %73. Compared with no augmentation case, the best F_1-score is improved from %77 to 80% in BERT-uncased model.

Overall, data augmentation technique with GPT-3.5 improved the best F_1-score performance for the traditional machine learning with feature enhancements and BERT-based models. The results are promising considering we were only able to increase the number of samples twice as many from the original amount due to budget limitations. Further augmentation can be beneficial for those methods and experiments can be conducted in future studies.

Table 4.16. Feature enhancements with augmented data for Bug Report.

Model	**Precision**	**Recall**	**F1-score**
Logistic Regression	0.79	0.68	0.73
Random Forest	0.55	0.63	0.59
Naive Bayes	0.57	0.71	0.63
Linear SVC	0.75	0.73	**0.74**

Table 4.17. Word embeddings for augmented Bug Report.

Model	**Precision**	**Recall**	**F1-score**
Logistic Regression	0.53	0.73	**0.61**
Random Forest	0.66	0.56	**0.61**
Naive Bayes	0.51	0.59	0.55
Linear SVC	0.50	0.71	0.59

Table 4.18. BERT models with augmented data for Bug Report.

Model	**Precision**	**Recall**	**F1-score**
BERT-uncased	0.76	0.84	**0.80**
BERT-cased	0.79	0.80	0.79
BERT-128k-uncased	0.64	0.85	0.73
BERT-128k-cased	0.77	0.81	0.79
RoBERTa-uncased	0.72	0.76	0.74
DistilBERT-cased	0.75	0.78	0.77

4.4.3. Classification with Generative Artificial Intelligence models

In this section, we utilize different prompt engineering techniques, namely Few-shot and Chain-of-Thought prompting, for our app review classification task. Additionally, we took one step forward and fine-tuned the GPT-3.5 model to compare the results with prompt engineering. It is important to note that, due to budget limitations, we applied these methods to only the Bug Report category to compare with data augmentation and other techniques.

First, we employed Few-shot learning, feeding the system with a few unique reviews from each application and then waiting for correct tag output [117]. We tried with several samples separately and concluded that a total of 12 samples is the most appropriate amount. We selected two unique samples from our six different applications. Table 4.19 demonstrates a few examples from the data that we used for input.

We covered each review with XML tags for the model to understand the input's beginning and end. To guide the model for proper output, we added a single instruction that tells the model to output the label between the same XML tags of the input review.

Table 4.19. Few-shot data examples.

Review Content	**Bug Report**
Çok kötü, düzgün çalışmıyor, bildirime tıklıyorsun haberi açmıyor	✓
İndirim kodlarını kabul etmiyor.	✓
30 yıldır bankada bir olumsuzluk yaşamadım.	✗
Siparişi yanlış bir adrese gönderdiler.	✗

Second, we applied Chain-of-Thought prompting [118], in which we detailed the logical process of our reasoning step-by-step with few sentences for our model. We explained both Bug Report and non-Bug Report reviews during the process. In total, we introduced 11 distinct and descriptive reviews for our model to classify.

Table 4.20. Chain-of-Thought process for a single sample.

Type	**Content**
Review	Adresimi kayıt etmeme rağmen sipariş veremiyorum
Step 1	Kullanıcı uygulamanın sipariş özelliğini kullanamıyor.
Step 2	Bu teknik bir probleme işaret ediyor.
Step 3	Bu yüzden bu bir Hata Raporu'dur.

Third, we fine-tuned the GPT-3.5 model to observe the performance compared to prompt engineering. Fine-tuning involves further training the pre-trained model to focus on our particular task [119]. We accomplished this by feeding the system with our specific data, which adjusts the parameters accordingly. Due to budget limitations, we were able to fine-tune the model for only one epoch.

Table 4.21 demonstrates the results for Few-shot, Chain-of-Thought prompting, and fine-tuning with GPT-3.5. Considering Few-shot prompting, the results are not promising since the F_1-score for Bug Report label is below traditional machine learning, word embeddings and BERT-based models that we used previously. We investigated the reasons for this and observed that Operational reviews are mostly labeled as Bug Report, hence this may indicate worse performance. When we try to explain the reasoning with Chain-of-Thought, we obtained better Recall scores compared to Few-Shot technique, but still suffered in terms of Precision and F_1-score. However, we noticed that Recall score of 85% is better than most of the traditional machine learning methods that we used. Furthermore, with fine-tuning, we observed the best precision score among all methods that we used with 86%. Only BERT-based models in which we used oversampling (BERT-cased and BERT-128k-cased models) surpasses this score with 87% and 88%, respectively. Although the recall score for fine-tuning is 75%, we obtained an F_1-score of 80%, which is above all techniques that we used previously. We only received the same F_1-score for that label when we applied oversampling for BERT-based models (BERT-128k-cased) and when we augment Bug Report data for BERT-uncased model. These findings indicate that fine-tuning with GPT-3.5 model gives same performance in terms of F_1-score without using data sampling and data augmentation methods. In other words, it performs better with lower amount of data compared to traditional machine learning techniques and BERT-based models, which is significant to note.

Table 4.21. ChatGPT with different methods.

Model	**Precision**	**Recall**	**F1-score**
Few Shot	0.55	0.65	0.60
Chain of Thought	0.42	**0.85**	0.56
Fine Tuning	**0.86**	0.75	**0.80**

5. DISCUSSION

In this section, we analyze the results of our approaches, namely traditional machine learning with feature enhancements, word embeddings, and BERT-based models, between each other. When we compare feature enhancements with the word embeddings technique, we observed that feature enhancements performed better in terms of the best F_1-scores for each label, with a weighted average of F_1-score 80% and weighted standard deviation of 9%. For the best-performing word embedding, the best weighted F_1-score average is 76%, and its weighted standard deviation is 9%. This indicates that our manually crafted features provided more information than word embedding techniques. It is important to note that for the categories with low data (Bug Report and Feature Request), both models perform worse than the labels with more data (Operational, Rating, and User Experience).

```
{
    'name': 'Random Forest’,

    'grid_search_parameters’: {

        'n_estimators': [50, 100, 200],
        'max_depth': [3, 5, 10],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4],
        'random_state': [42],
        'n_jobs': [-1],
        'class_weight': ["balanced"]

    }
}
```

Figure 5.1. Grid search parameters of Random Forest classifier

Considering hyperparameters used in traditional machine learning classifiers, we applied grid search to find the optimal parameters for each classifier. We investigated previous studies to obtain possible hyperparameters for our app review classification task.

Figure 5.1 demonstrates the possible hyperparameters used for the Random Forest classifier. To investigate further, we look at one of our techniques, traditional machine learning with FastText word embeddings. After these parameters are fed into grid search, we obtain the following optimal hyperparameters. Our *n_estimators* parameter, which indicates the number of trees in the forest, resulted in 200. Having more trees generally improves the model's performance up to a certain point. 200 trees is a reasonable value that should provide a good balance between accuracy and computational efficiency. Our *max_depth* parameter, which controls the maximum depth of each decision tree in the forest, is 10. A lower value like 10 can help prevent overfitting by limiting the depth of the individual trees. It essentially restricts how many questions the model can ask about the data during the learning process. Our *min_samples_leaf* parameter is 1, which is the minimum number of samples required to be at a leaf node. A small value like 1 means that each leaf node can have only one sample. This can lead to more fine-grained decisions and potentially overfitting, but it might work well in combination with other parameters. Our *min_samples_split* parameter, which is the minimum number of samples required to split an internal node, is 2. A value of 2 means that a node will only be split if it has at least two samples. This can help avoid small splits that capture noise in the data.

It is important to note that we obtained better results than other classifiers in word embedding techniques for the Random Forest classifier, and the results are promising. Additional hyperparameters like *criterion, min_weight_fraction_leaf* and *max_features* can further improve the classifier's performance.

Regarding BERT-based models with other techniques, we observed that BERT-based models outperform feature enhancements and word embedding techniques in all labels regarding the best weighted F_1-score and the best weighted standard deviation. The best weighted F_1-score is 85%, and the best weighted standard deviation is 5% for the best BERT-based model, which is BERT-128k-cased. Due to its large vocabulary of size 128k, it was able to predict better in almost all labels except Bug Report, where the BERT-uncased model performed better with 77% F_1-score.

Considering the effect of sampling techniques, the undersampling technique underperforms regarding the best weighted F_1-score since we reduced the majority class samples, which causes information loss, especially for classes with low samples. On the other hand, the oversampling technique improved the results for feature enhancements. For feature enhancements, the best weighted F_1-score is increased up to 81% from 80%. For BERT-based models, the best weighted F_1-score remains the same, but the best weighted standard deviation is decreased from 5% to 3%, which shows a closer distribution to the weighted mean and, hence, more consistent results. Similarly, for word embeddings, the same best-weighted F_1-score is obtained, but the best-weighted standard deviation is decreased from 5% to 3%, which indicates closer distribution to the weighted mean and, therefore, more precise results. Although the oversampling technique improved the performance within one approach, it does not cause one approach to outperform the other compared to no sampling case.

Overall, we conclude that although some labels contain a low number of samples, BERT-based models outperformed other models in terms of the best-weighted F_1-score and standard deviation. Furthermore, our oversampling and augmentation techniques prove that with more data, the results of our best-performing model can be further improved.

5.1. Investigation of Confusion Matrices

This section focuses on confusion matrices obtained by our best-performing technique, *BERT with RandomOverSampling.* We report the best-performing BERT model's confusion matrix for each label. Figure 5.2 and the following figures demonstrate the confusion matrices for Operational, Bug Report, Rating, Feature Request, and User Experience labels, respectively. It is important to note that we used an 80:20 train-test split ratio, and confusion matrices are from the test set. When we investigated the Operational, User Experience, and Rating labels' confusion matrices, we observed that true positives and true negatives are close to each other in terms of number. Bug Report and Feature Request labels need more samples and contain fewer true positives.

Another critical remark is that we try to use a stratified split to ensure the train split contains the same sample ratio as the test set.

To explore the model's performance further, we focused on some predictions that the model made. For instance, *"Her güncellemede reklamlar çoğalıyor reklamları güncelliyoruz galiba."* is an example that explains that new updates introduce more advertisements, which annoys the user. Since the decision to include more ads comes from the business side of the application, the review is an Operational review. However, our model predicted this as a Bug Report, which is a wrong classification. One of the reasons for this may be that the model learns the update-related negative reviews as Bug Reports rather than Operational.

Another example is *"Arayüz gereksiz detaylarla dolu. Kullanıcı dostu değil. Sadelestirilmesi gerekiyor."* which demonstrates user's complaints about the user interface, indicating a User Experience review. The user also requests a more straightforward graphical interface, marking the review as a Feature Request. Nevertheless, our model classified the review as a Bug Report. This classification may be because the user interface experience is related to the application's technical features by the model. When we investigated similar reviews about the user interface, the model again made some wrong classifications for those reviews. Possible solutions for this problem are to improve the data for Bug Report and Feature Request with more user interface-related reviews and to retrain our model.

Final example is *"Çok sıkıcı bir uygulama. Hiç memnun kalmadım."* which demonstrates the user's overall rating, indicating a Rating review. Nevertheless, our model classified the review as a User Experience. This classification may be because the user interface experience is related to the user's satisfaction with the application. When we investigated similar reviews about the rating, the model again made some wrong classifications for some of the reviews.

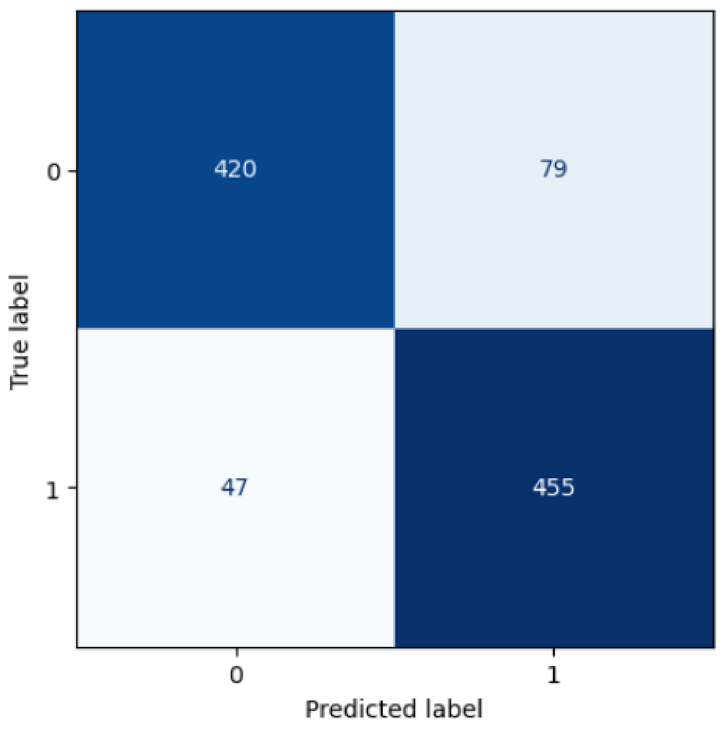

Figure 5.2. Confusion matrix for Operational label.

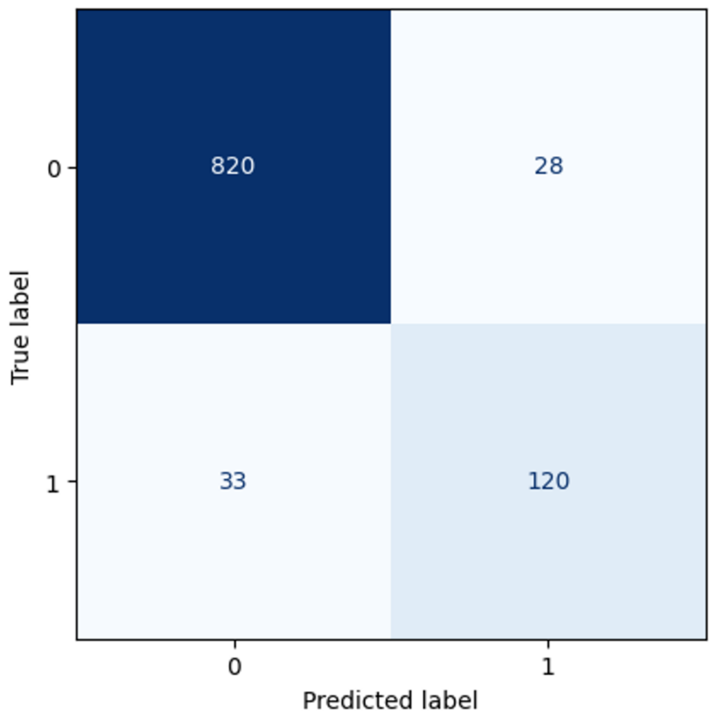

Figure 5.3. Confusion matrix for Bug Report label.

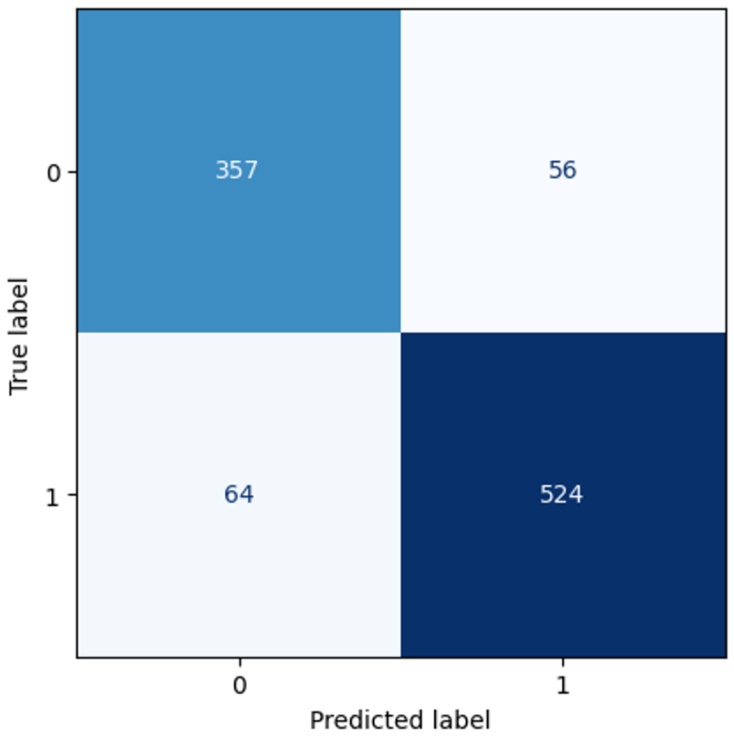

Figure 5.4. Confusion matrix for Rating label.

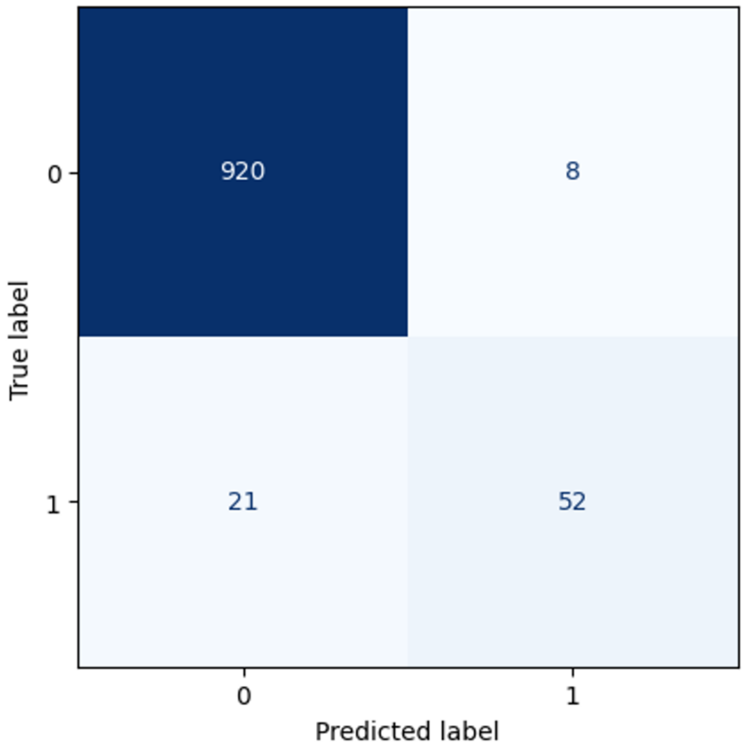

Figure 5.5. Confusion matrix for Feature Request label.

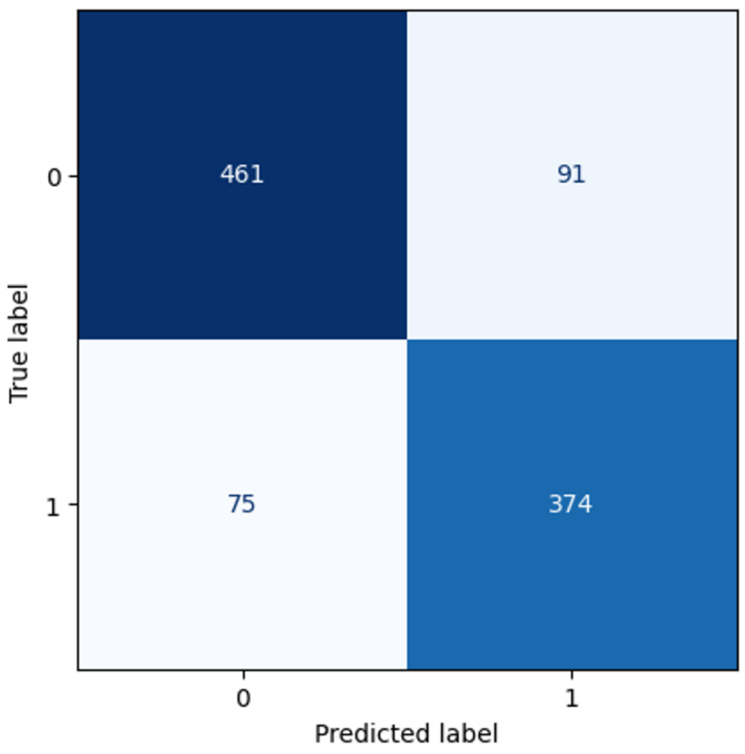

Figure 5.6. Confusion matrix for User Experience label.

5.2. Comparison with Previous Studies

As stated before, our work is the only one that employs app review classification tasks in Turkish. Hence, we can not compare our results with other studies in the same language. Nevertheless, there are several English studies where we can compare our findings with the same or similar labels. Since we obtained our four labels, Bug Report, Feature Request, User Experience and Rating from Maalej et al. [9], we can directly compare our results with this study. Regarding the Bug Report category, Maalej et al. [9] reached 88% F_1-score whereas our study remained below at 80%. Considering Feature Requests, they obtained 85% F_1-score, and our study, due to class imbalance, reached only 78% F_1-score. For the User Experience label, their best model resulted in 92% F_1-score, whereas ours stayed at 82%. Our study surpasses their study in the Rating label, where they obtained 87% F_1-score, and our study reached 90%. They used traditional machine learning methods with several features. Their results show that more specific features can contribute to our traditional machine-learning pipeline. Another study from Gu and Kim [60] classifies with Request and Bug labels, corresponding to our Feature Request and Bug Report labels, respectively. Their study reached 59% F_1-score for the Request label, which is below ours, 78%. For the Bug label, they reached 65% F_1-score, whereas we obtained 80% F_1-score. Stanik et al. [15] crawled app reviews and tweets for such a task. They labeled their data as Problem Report, which corresponds to our Bug Report label of ours. They also tagged Feature Request reviews as Inquiry. They applied deep learning and traditional machine learning techniques. They obtained the best F_1-score in the Problem Report label as 79%. We surpass this score with 80% in our study. They obtained 74% F_1-score for the Inquiry label, which is again below ours, 78%. Guzman et al. [50] classified reviews similar to our study, with Bug Report and User Request labels. They received 81% best F_1-score in the Bug Report label when they applied ensemble methods, which is 1% higher than our study. For the User Request label, they obtained 51% best F_1-score, which is below our results, %78. Finally, Panichella et al. [14] applied Random Forest for such task and obtained 34% best F_1-score for the Feature Request label in which we obtained 78% for the same label. For the Problem Discovery label, which corresponds to our Bug Report label, they obtained 82% best F_1-score, which is above ours, %80.

To summarize, studies in the literature demonstrate different scores for our labels. We surpassed the scores in many studies regarding Bug Report, Feature Request, and Rating labels. Since Turkish is a morphologically rich agglutinative language, more carefully extracted features may contribute to the performance of the traditional machine learning classifiers.

5.3. Implications on Practice

Our study may contribute to the mobile application development process as a tool for review analysis. Bug Report reviews might provide instant feedback for development teams. Product managers, product owners, and requirements engineers may utilize the Feature Request reviews. Operational reviews might be critical for the operations department. In fact, we constructed this label according to the feedback obtained from a software consultancy company. We held a session with the directors of this company, and they advised creating this label to improve the functionality and usability of the tool. They stated that shipment problems, delivery issues, and customer support cases are severe topics that companies should handle. The presence of an Operational tag might give vital information to teams that are responsible for those situations. Additionally, User Experience reviews may provide insights about how customers benefit from the app. Rating reviews demonstrate user satisfaction with the applications. Application owners can watch the ratio and the content of the Rating reviews regularly, observe the users' reactions during that period, and act accordingly.

5.4. Threats to Validity

This study contributes to the literature by providing a new manually annotated dataset for Turkish application reviews. We need to mention that the bias of the annotators during the annotation process can be a threat to *internal validity*. To alleviate this threat, we constructed written guidelines that describe our labels in detail with examples and held several training sessions. Furthermore, some of our labels introduced complexity for annotators to annotate. To mitigate this, we also highlighted and covered intricate samples. We used Cohen's kappa and Krippendorff's alpha coefficients to reach satisfactory agreement levels for novice annotators while annotating the training dataset. To avoid individual bias, we ensured that at least two annotators had annotated every data sample and held resolution sessions to resolve conflicts. When uncertainty occurred, an additional annotator joined the conversation to address the solution.

The generalizability of our findings is the concern of *external validity*. We obtained reviews from the two most prevalent application stores: Google Play Store and Apple App Store. Other mobile application stores also exist. Depending on the application store, the characteristics of user reviews may vary. Moreover, Turkish culture and customs may impact the user review characteristics [120]. Furthermore, the reviews we crawl come from six distinct applications in six categories. It is noted that the characteristics of user reviews are subject to change, as seen in Figure 5.7. We need further research for a definitive conclusion.

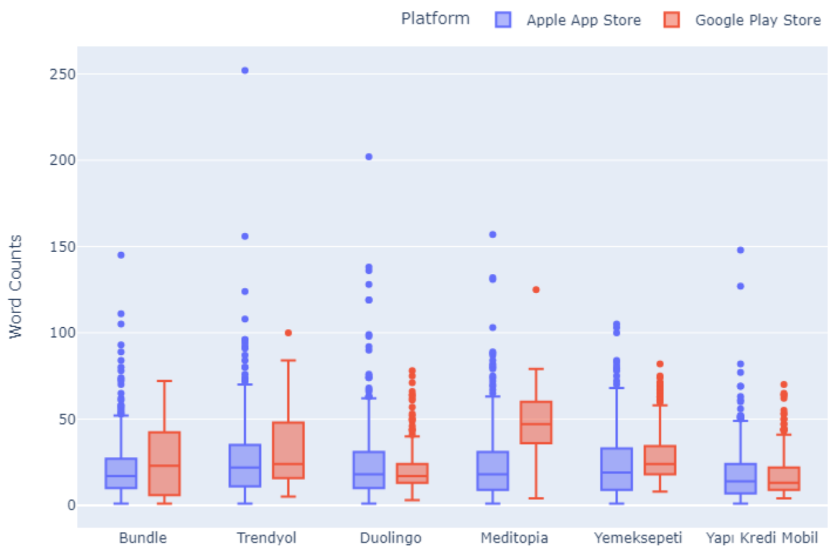

Figure 5.7. Word counts per application and store.

The imbalanced dataset may threaten the *conclusion validity*. We applied sampling techniques and data augmentation to reduce the impact of the threat. This approach was critical, especially for minority classes, due to the low performance of such label classifications. Manually annotated small datasets in this field typically require assistance addressing these issues. When utilizing large datasets, this problem is less likely to be encountered.

We utilize transfer learning by employing deep language models specifically designed to perform general-purpose natural language processing tasks. As the problem of application review classification is inherently a text classification task, our approach is not affected by any factors that could threaten *construct validity*. We utilize statistical measures like weighted standard deviation and weighted F_1-score average to ensure the accuracy of our conclusions regarding our classifiers' performance.

6. CONCLUSIONS AND FUTURE WORK

Mobile application reviews provide significant information for the industry. This study aims to utilize and gain deeper insights from that information. For this purpose, we propose an annotation guideline explaining the state-of-the-art labels (Bug Report, Feature Request, Rating, User Experience) in detail. Additionally, according to feedback from the industry, we introduce a new label called Operational that describes the operational processes (shipment problems, delivery issues), which differ from the mobile application's technical functionality. Overall, we contribute with a multi-label annotation schema and a manually constructed dataset of size 5004. We made our dataset publicly available for further research.

For the app review classification task, we build classification pipelines that employ different techniques. First, we apply traditional machine learning methods. We use Naive Bayes, Random Forest, Logistic Regression, and Support Vector Machines for the machine learning classifiers. As feature extraction baselines, we utilize bag of word representations and term frequency-inverse document frequencies. Furthermore, to improve performance, we manually extract features such as metadata, lemma, and part-of-speech tag occurrences. We measure the performance in terms of precision, recall, and F_1-score. We observe performance improvement after custom feature extraction.

Second, we process app reviews to experiment with widely used word embeddings, which are Word2Vec, FastText, GloVe, and ELMo. FastText shows better performance than Word2Vec, aligned with the fact that it is an extension of Word2Vec. ELMo demonstrates low performance as it can not learn contextual information from short reviews. Overall, GloVe produces the best results among all embeddings. However, word embeddings underperform compared to our custom feature extraction technique, indicating carefully handcrafted features can surpass embeddings for the Turkish app review classification task. It is important to note that these embeddings are mainly pre-trained with Commoncrawl data. Further pre-training is needed to be conclusive.

We also apply commonly used deep language models such as BERT, RoBERTa, and DistilBERT. We employ distinct versions of BERT models with different case sensitivity and vocabulary sizes. We observed that BERT-based models outperform traditional machine learning techniques with custom feature extraction and word embedding features.

Another topic we focus on is the effect of generative AI models on our classification task. For this purpose, we utilize the GPT-3.5 model. To the best of our knowledge, this thesis is the first study that uses GPT-3.5 for the Turkish app review classification task. We first employ fundamental prompt engineering techniques: Few-shot and Chain-of-Thought prompting. Additionally, we apply fine-tuning to compare the results with prompting methods. Due to budget limitations, we could only apply these methods to a single label, Bug Report. Regarding Few-shot prompting, the results do not outperform our traditional machine learning and deep language models. Nevertheless, we observe high recall values after Chain-of-Thought prompting, which challenges our traditional machine-learning techniques with custom feature extraction. This shows that, with the help of GPT-3.5, one can quickly identify Bug Report reviews without any fine-tuning, yet takes many false positives into consideration. On the other hand, when we fine-tuned with our train data, we also observed improvement in Precision score and, hence, better F_1-score compared to our traditional machine learning and deep language models. To conclude, it is seen that fine-tuning is the best approach to obtain better results. However, it needs an extra training phase compared to prompting techniques.

One concern is the imbalance of our dataset. Bug Report and Feature Request labels contain low amount of samples compared to others. This causes low performance in all of our classifiers. To mitigate that issue, we first employed sampling techniques. Undersampling resulted in low performance since it decreases majority class data which causes loss of information. On the other hand, oversampling improved the performance, especially for the minority classes. Due to class imbalance, we also reported weighted F_1-score average and weighted standard deviation to evaluate overall performance.

Additionally, we investigate the effects of data augmentation on the class imbalance problem. For this purpose, we used GPT-3.5 to generate augmented data. Due to cost restrictions, we applied this technique to only the Bug Report label. Because of the same reason, we were only able to double the data compared to sampling techniques, where we sampled the minority class data more, up to the number of majority class samples. We observed that GPT-3.5 can produce syntactically and semantically meaningful data. When we feed our models with such data, we observe performance improvement for traditional machine learning with feature enhancements and BERT-based models. It indicates that with the help of generative AI models for data augmentation, we can obtain better results than other sampling techniques with less data generation. Further augmentation would benefit those methods, and with a larger budget, experiments can be extended in future studies.

We also investigate the usefulness of our study. To focus on that topic, we held a session with Product Managers (PMs) from industry and asked for their feedback. They stated that such app review classifier tool would benefit their processes. They suggested introducing a new label called OPERATIONAL, which can help companies gain insights about business processes besides the technical functionality of their applications. After the session, we considered their advice and constructed our dataset including OPERATIONAL label. We observed significant portion of data contains OPERATIONAL reviews aligned with the PMs' suggestions and our classifiers were able to detect them with success.

Overall, our classification pipelines contribute to the industry by providing deeper insights. PMs will save significant time and human effort by utilizing recall and precision metrics. They have the flexibility to select any label they want to concentrate on and make decisions based on user feedback. For instance, PMs can examine USER EXPERIENCE and RATING labels to observe customer satisfaction and overall experience. They can investigate OPERATIONAL reviews to understand how business processes work. Developers can benefit from BUG REPORT and FEATURE REQUEST labels to identify problems and missing features.

Despite the benefits of our proposed pipelines and its significant outcomes, there exists several challenges related with our app review classification task and analysis. The process of manual annotation poses another considerable challenge. Dataset creation by web crawling and manual labeling requires significant time and effort. Additionally, human annotators carefully read and categorize each review, which may introduce inconsistencies due to subjectivity. This process would be complicated for more extensive dataset construction due to increased manual effort. Furthermore, we annotated reviews of size 5004 from a specific period, which is a significant contribution, yet it may only cover some variations of reviews. A larger dataset from a broader time range would reveal more profound insights.

Classifying user reviews presents another challenge due to the evolution of language and changes in user sentiment. Language conventions, user behavior, and context are all subject to vary over time. Hence, ongoing monitoring and updates are needed to keep up with the new trends and topics via adapting the classification pipelines.

Another challenge is integrating such classification pipelines into the product's development lifecycle. User-friendly tools with enhanced analytical capabilities are needed for smooth integration. Such a tool's scalability, maintainability, and security are other essential topics that must be covered.

To further expand upon the outcomes of our work, future studies might focus on extending the dataset for minority classes. Achieving this would result in better classification performance and robustness for our models. Furthermore, experiments with other applications from different categories would contribute to the generalizability and reliability of our findings.

REFERENCES

1. BuildFire, "Mobile App Download and Usage Statistics", https://buildfire.com/app-statistics, accessed on February 11, 2023.

2. Google, "How Google Play Works", https://play.google.com/about/howplayworks, accessed on February 11, 2023.

3. Apple, "Apple Developers Grow App Store Ecosystem Billings And Sales", https://apple.com/newsroom/2021/06/apple-developers-grow-app-store-ecosystem-billings-and-sales-by-24-percent-in-2020/, accessed on July 11, 2021.

4. Burnay, C., I. J. Jureta, and S. Faulkner, "What Stakeholders Will or Will Not Say: A Theoretical and Empirical Study of Topic Importance in Requirements Engineering Elicitation Interviews", *Information Systems*, Vol. 46, No. 1, pp. 61–81, 2014.

5. Johann, T., C. Stanik, A. M. Alizadeh B., and W. Maalej, "SAFE: A Simple Approach for Feature Extraction from App Descriptions and App Reviews", *IEEE 25th International Requirements Engineering Conference*, Vol. 25, pp. 21–30, Hannover, Germany, 2017.

6. Maalej, W. and H. Nabil, "Bug Report, Feature Request, or Simply Praise? On Automatically Classifying App Reviews", *IEEE 23rd International Requirements Engineering Conference*, Vol. 23, pp. 116–125, Hannover, Germany, 2015.

7. Pagano, D. and W. Maalej, "User Feedback in the AppStore: An Empirical Study", *21st IEEE International Requirements Engineering Conference*, Vol. 21, pp. 125–134, Hannover, Germany, 2014.

8. Al-Subaihin, A. A., F. Sarro, S. Black, L. Capra, and M. Harman, “App Store Effects on Software Engineering Practices”, *IEEE Transactions on Software Engineering*, Vol. 47, No. 2, pp. 300–319, 2021.

9. Maalej, W., Z. Kurtanović, H. Nabil, and C. Stanik, “On the Automatic Classification of App Reviews”, *Requirements Engineering*, Vol. 21, pp. 311–331, 2016.

10. Ickin, S., K. Petersen, and J. Gonzalez-Huerta, “Why Do Users Install and Delete Apps? A Survey Study”, *Software Business: 8th International Conference Proceedings*, Vol. 8, pp. 186–191, Essen, Germany, 2017.

11. Li, H., L. Zhang, L. Zhang, and J. Shen, “A User Satisfaction Analysis Approach for Software Evolution”, *IEEE International Conference on Progress in Informatics and Computing*, Vol. 2, pp. 1093–1097, 2010.

12. Hao, L., X. Li, Y. Tan, and J. Xu, “The Economic Role of Rating Behavior in Third-Party Application Market.”, *26th ICIS Proceedings*, Vol. 26, pp. 100–104, 2011.

13. Hyrynsalmi, S., M. Seppänen, L. Aarikka-Stenroos, A. Suominen, J. Järveläinen, and V. Harkke, “Busting Myths of Electronic Word of Mouth: The Relationship between Customer Ratings and the Sales of Mobile Applications”, *Journal of Theoretical and Applied Electronic Commerce Research*, Vol. 10, pp. 1–18, 2015.

14. Panichella, S., A. Di Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, and H. C. Gall, “How Can I Improve My App? Classifying User Reviews for Software Maintenance and Evolution”, *IEEE International Conference on Software Maintenance and Evolution*, Vol. 12, pp. 281–290, Bogota, Colombia, 2015.

15. Stanik, C., M. Haering, and W. Maalej, “Classifying Multilingual User Feedback Using Traditional Machine Learning and Deep Learning”, *IEEE 27th International Requirements Engineering Conference Workshops*, Vol. 27, pp. 220–226,

Jeju Island, South Korea, 2019.

16. Ekanata, Y. and I. Budi, "Mobile Application Review classification for the Indonesian Language Using Machine Learning Approach", *4th International Conference on Computer and Technology Applications*, Vol. 2, pp. 117–121, Istanbul, Turkey, 2018.

17. Bertan, S., M. Bayram, and N. Benzergil, "The Evaluation of Thermal Hotels' Online Reviews", *Tourism: An International Interdisciplinary Journal*, Vol. 63, No. 1, pp. 53–65, 2015.

18. Kebabcı, K. and B. Diri, "User Opinion Analysis On Turkish Tweets", *Mugla Journal of Science and Technology*, Vol. 12, No. 1, pp. 35–38, 2017.

19. Siğirci, O., H. Özgür, A. Oluk, H. Uz, E. Çetiner, H. U. Oktay, and K. Erdemir, "Sentiment Analysis of Turkish Reviews on Google Play Store", *5th International Conference on Computer Science and Engineering*, Vol. 1, pp. 314–315, 2020.

20. Ogul, B. and G. Ercan, "Sentiment classification on Turkish Hotel Reviews", *24th Signal Processing and Communication Application Conference*, Vol. 3, pp. 497–500, Istanbul, Turkey, 2016.

21. University, B., "Why Study Turkish?", `https://www.bu.edu/wll/home/why-study-turkish/`, accessed on February 11, 2023.

22. Zhao, L., W. Alhoshan, A. Ferrari, K. J. Letsholo, M. A. Ajagbe, E.-V. Chioasca, and R. T. Batista-Navarro, "Natural Language Processing for Requirements Engineering: A Systematic Mapping Study", *ACM Computing Surveys*, Vol. 54, No. 3, pp. 1–41, 2021.

23. Glinz, M., "On Non-Functional Requirements", *15th IEEE International Requirements Engineering Conference*, Vol. 4, pp. 21–26, Delhi, India, 2007.

24. Kurtanović, Z. and W. Maalej, "Automatically Classifying Functional and Non-functional Requirements Using Supervised Machine Learning", *IEEE 25th International Requirements Engineering Conference*, Vol. 6, pp. 490–495, 2017.

25. Hussain, I., L. Kosseim, and O. Ormandjieva, "Using Linguistic Knowledge to Classify Non-functional Requirements in SRS Documents", *Natural Language and Information Systems: 13th International Conference on Applications of Natural Language to Information Systems*, Vol. 7, pp. 287–298, 2008.

26. Binkhonain, M. and L. Zhao, "A Review Of Machine Learning Algorithms For Identification And Classification Of Non-Functional Requirements", *Expert Systems with Applications*, Vol. 1, pp. 100–120, 2019.

27. An'im Fahmi, A. and D. Siahaan, "Algorithms Comparison for Non-Requirements Classification Using the Semantic Feature of Software Requirement Statements", *IPTEK The Journal for Technology and Science*, Vol. 31, No. 3, pp. 343–352, 2021.

28. Le, T., C. Le, H. David Jeong, S. B. Gilbert, and E. Chukharev-Hudilainen, "Requirement Text Detection from Contract Packages to Support Project Definition Determination", *Advances in Informatics and Computing in Civil and Construction Engineering: Proceedings of the 35th CIB Conference: IT in Design, Construction, and Management*, Vol. 7, pp. 569–576, Springer, 2019.

29. Sommerville, I., *Software Engineering*, Addison-Wesley, Harlow, England, 9th Edition, 2010.

30. Lauesen, S., *Software Requirements: Styles and Techniques*, Addison-Wesley, Copenhagen, Denmark, 2002.

31. Wiegers, K. E. and J. Beatty, *Software Requirements 3*, Microsoft Press, USA, 2013.

32. Tavakoli, M., L. Zhao, A. Heydari, and G. Nenadic, "Extracting Useful Software Development Information from Mobile Application Reviews: A Survey of Intelligent Mining Techniques and Tools", *Expert Systems with Applications*, Vol. 113, pp. 45–61, 2018.

33. Nazar, N., Y. Hu, and J. He, "Summarizing Software Artifacts: A Literature Review", *Journal of Computer Science and Technology*, Vol. 31, pp. 883–909, 2016.

34. Harman, M., Y. Jia, and Y. Zhang, "App Store Mining and Analysis: MSR for App Stores", *9th IEEE Working Conference on Mining Software Repositories*, pp. 108–111, 2012.

35. Santos, R., E. C. Groen, and K. Villela, "A Taxonomy for User Feedback Classifications", *Requirements Engineering: Foundation for Software Quality Workshops*, Vol. 2, pp. 10–21, 2019.

36. Santos, R., E. C. Groen, and K. Villela, "An Overview of User Feedback Classification Approaches", *Requirements Engineering: Foundation for Software Quality Workshops*, Vol. 7, pp. 20–42, 2019.

37. Guzman, E. and W. Maalej, "How Do Users Like This Feature? A Fine Grained Sentiment Analysis of App Reviews", *IEEE 22nd International Requirements Engineering Conference*, Vol. 14, pp. 153–162, 2014.

38. Iacob, C. and R. Harrison, "Retrieving and Analyzing Mobile Apps Feature Requests from Online Reviews", *10th Working Conference On Mining Software Repositories*, Vol. 10, pp. 41–44, 2013.

39. Galvis, L. and K. Winbladh, "Analysis of User Comments: An Approach for Software Requirements Evolution", *35th International Conference on Software Engineering*, Vol. 3, pp. 582–591, San Francisco, CA, USA, 2013.

40. Haering, M., C. Stanik, and W. Maalej, “Automatically Matching Bug Reports With Related App Reviews”, *IEEE 43rd International Conference on Software Engineering*, Vol. 7, pp. 970–981, Madrid, Spain, 2021.

41. Chen, N., J. Lin, S. C. H. Hoi, X. Xiao, and B. Zhang, “AR-miner: Mining Informative Reviews for Developers from Mobile App Marketplace”, *Proceedings of the 36th International Conference on Software Engineering*, Vol. 9, p. 767–778, Hyderabad, India, 2014.

42. Lu, M. and P. Liang, “Automatic Classification of Non-Functional Requirements from Augmented App User Reviews”, *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, Vol. 9, p. 344–353, Karlskrona, Sweden, 2017.

43. Dalpiaz, F. and M. Parente, “RE-SWOT: From User Feedback to Requirements via Competitor Analysis”, *Proceedings of the 25th International Working Conference on Requirements Engineering: Foundation for Software Quality*, Vol. 2, pp. 55–70, Hannover, Germany, 2019.

44. Jin, J., P. Ji, and R. Gu, “Identifying Comparative Customer Requirements from Product Online Reviews for Competitor Analysis”, *Engineering Applications of Artificial Intelligence*, Vol. 49, pp. 61–73, 2016.

45. Genc-Nayebi, N. and A. Abran, “A Systematic Literature Review: Opinion Mining Studies from Mobile App Store User Reviews”, *Journal of Systems and Software*, Vol. 125, pp. 30–43, 2016.

46. Kunaefi, A. and M. Aritsugi, “Characterizing User Decision based on Argumentative Reviews”, *IEEE/ACM International Conference on Big Data Computing, Applications and Technologies*, Vol. 12, pp. 15–21, Leicester, UK, 2020.

47. Phong, M. V., T. T. Nguyen, H. V. Pham, and T. T. Nguyen, “Mining User

Opinions in Mobile App Reviews: A Keyword-Based Approach", *30th IEEE/ACM International Conference on Automated Software Engineering*, Vol. 7, pp. 749–759, Madrid, Spain, 2015.

48. Asghar, M. Z., A. Khan, A. Bibi, F. M. Kundi, and H. Ahmad, "Sentence-level Emotion Detection Framework Using Rule-Based Classification", *Cognitive Computation*, Vol. 9, pp. 868–894, 2017.

49. Rana, T. A. and Y.-N. Cheah, "Hybrid Rule-Based Approach for Aspect Extraction and Categorization from Customer Reviews", *9th International Conference on IT in Asia*, Vol. 9, pp. 1–5, Bangkok, Thailand, 2015.

50. Guzman, E., M. El-Haliby, and B. Bruegge, "Ensemble Methods for App Review Classification: An Approach for Software Evolution", *30th IEEE/ACM International Conference on Automated Software Engineering*, Vol. 30, pp. 771–776, Amsterdam, Netherlands, 2015.

51. Shah, F. A., K. Sirts, and D. Pfahl, "Simple App Review Classification with Only Lexical Features", *International Conference on Software and Data Technologies*, Vol. 9, pp. 146–153, Vienna, Austria, 2018.

52. Aslam, N., W. Y. Ramay, K. Xia, and N. Sarwar, "Convolutional Neural Network Based Classification of App Reviews", *IEEE Access*, Vol. 8, pp. 185619–185628, 2020.

53. Al-Hawari, A., H. Najadat, and R. Shatnawi, "Classification of Application Reviews into Software Maintenance Tasks Using Data Mining Techniques", *Software Quality Journal*, Vol. 29, pp. 667–703, 2021.

54. Araujo, A., M. Golo, B. Viana, F. Sanches, R. Romero, and R. Marcacini, "From Bag-of-Words to Pre-Trained Neural Language Models: Improving Automatic Classification of App reviews for Requirements Engineering", *Anais do XVII En-*

contro Nacional de Inteligência Artificial e Computacional, Vol. 2, pp. 378–389, 2020.

55. Rustam, F., A. Mehmood, M. Ahmad, S. Ullah, D. M. Khan, and G. S. Choi, "Classification of Shopify App User Reviews Using Novel Multi Text Features", *IEEE Access*, Vol. 8, pp. 30234–30244, 2020.

56. Triantafyllou, I., I. C. Drivas, and G. Giannakopoulos, "How to Utilize My App Reviews? A Novel Topics Extraction Machine Learning Schema for Strategic Business Purposes", *Entropy*, Vol. 22, pp. 1310–1322, 2020.

57. Hadi, M. A. and F. H. Fard, "Evaluating Pre-Trained Models for User Feedback Analysis in Software Engineering: A Study on Classification of App Reviews", *Empirical Software Engineering*, Vol. 28, No. 4, pp. 88–100, 2023.

58. Zhou, W., Y. Wang, Y. Qu, and L. Li, "Automating App Review Classification Based on Extended Semantic", *9th International Conference on Dependable Systems and Their Applications*, Vol. 9, pp. 106–115, Kuala Lumpur, Malaysia, 2022.

59. Kaur, K. and P. Kaur, "BERT-RCNN: An Automatic Classification of App Reviews Using Transfer Learning Based RCNN Deep Model", 2023.

60. Gu, X. and S. Kim, "What Parts of Your Apps Are Loved by Users?", *30th IEEE/ACM International Conference on Automated Software Engineering*, Vol. 4, pp. 760–770, Lisbon, Portugal, 2015.

61. Guo, H. and M. P. Singh, "Caspar: Extracting and Synthesizing User Stories of Problems from App Reviews", *IEEE/ACM 42nd International Conference on Software Engineering*, Vol. 4, pp. 628–640, Prague, Czech Republic, 2020.

62. Khalid, H., E. Shihab, M. Nagappan, and A. E. Hassan, "What Do Mobile App Users Complain About?", *IEEE Software*, Vol. 32, pp. 1–10, 2014.

63. Henao, P., J. Fischbach, D. Spies, J. Frattini, and A. Vogelsang, "Transfer Learning for Mining Feature Requests and Bug Reports from Tweets and App Store Reviews", *IEEE 29th International Requirements Engineering Conference Workshops*, Vol. 3, pp. 80–86, Athens, Greece, 2021.

64. Kurtanović, Z. and W. Maalej, "Mining User Rationale from Software Reviews", *IEEE 25th International Requirements Engineering Conference*, Vol. 25, pp. 61–70, Hannover, Germany, 2017.

65. Oflazer, K., "Turkish and its Challenges for Natural Language Processing", *Language Resources and Evaluation*, Vol. 48, p. 639–653, 2014.

66. Tohma, K. and Y. Kutlu, "Challenges Encountered in Turkish Natural Language Processing Studies", *Natural and Engineering Sciences*, Vol. 5, pp. 204–211, 2020.

67. Yilmaz, H. and S. Yumusak, "Açık Kaynak Doğal Dil İşleme Kütüphaneleri", *İstanbul Sabahattin Zaim Üniversitesi Fen Bilimleri Enstitüsü Dergisi*, 2021.

68. Schweter, S., "BERT models for Turkish", `https://doi.org/10.5281/zenodo.3770924`, accessed on April 11, 2023.

69. Kenton, J. D. M.-W. C. and L. K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", *17th Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Vol. 1, p. 4171–4186, Minneapolis, USA, 2019.

70. Duan, S., J. Liu, and Z. Peng, "RCBERT an Approach with Transfer Learning for App Reviews Classification", *Computer Supported Cooperative Work and Social Computing*, Vol. 1, pp. 444–457, 2022.

71. Wu, H., W. Deng, X. Niu, and C. Nie, "Identifying Key Features from App User Reviews", *IEEE/ACM 43rd International Conference on Software Engineering*, Vol. 43, pp. 922–932, Bangkok, Thailand, 2021.

72. Zhang, L., K. Hua, H. Wang, G. Qian, and L. Zhang, "Sentiment Analysis on Reviews of Mobile Users", *Procedia Computer Science*, Vol. 34, pp. 458–465, 2014.

73. Fan, X., X. Li, F. Du, X. Li, and M. Wei, "Applying Word Vectors for Sentiment Analysis of App Reviews", *3rd International Conference on Systems and Informatics*, Vol. 3, pp. 1062–1066, Shanghai, China, 2016.

74. Di Rosa, E. and A. Durante, "App2Check: A Machine Learning-Based System for Sentiment Analysis of App Reviews in Italian Language", *International Workshop on Social Media World Sensors and International Conference on Language Resources and Evaluation*, Vol. 3, pp. 8–13, Paris, France, 2016.

75. Carrouel, F., B. d. S. de Vigneulles, D. Bourgeois, B. Kabuth, N. Baltenneck, F. Nusbaum, V. Burge, S. Roy, S. Buchheit, M.-L. Carrion-Martinaud, *et al.*, "Mental Health Mobile Apps in the French App Store: Assessment Study of Functionality and Quality", *JMIR mHealth and uHealth*, Vol. 10, No. 10, pp. 41282–41292, 2022.

76. Parlar, T. and S. A. Özel, "A New Feature Selection Method for Sentiment Analysis of Turkish Reviews", *International Symposium on Innovations in Intelligent Systems and Applications*, Vol. 2, pp. 1–6, IEEE, 2016.

77. Karahoca, A., D. Karahoca, and E. Evirgen, "Sentiment Analysis of Turkish Tweets by Data Mining Methods", Vol. 10, pp. 915–925, 01 2019.

78. Mobiloud, "List of Mobile App Stores", `https://www.mobiloud.com/blog/list-of-mobile-app-stores`, accessed on November 18, 2023.

79. Carrera, I., "Google Play Scraper", `https://pypi.org/project/google-play-scraper`, accessed on November 18, 2023.

80. Casas, O., "App Store Scraper", `https://pypi.org/project/app-store-scraper`, accessed on November 18, 2023.

81. McHugh, M. L., “Interrater Reliability: The Kappa Statistic”, *Biochemia medica*, Vol. 22, No. 3, pp. 276–282, 2012.

82. Krippendorff, K., *Content Analysis: An Introduction to Its Methodology*, Sage publications, 2018.

83. Dąbrowski, J., E. Letier, A. Perini, and A. Susi, “Analyzing App Reviews for Software Engineering: A Systematic Literature Review”, *Empirical Software Engineering*, Vol. 27, pp. 1310–1322, 2022.

84. 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, pp. 2825–2830, 2011.

85. Mohammed, R., J. Rawashdeh, and M. Abdullah, “Machine Learning with Oversampling and Undersampling Techniques: Overview Study and Experimental Results”, *11th International Conference on Information and Communication Systems*, pp. 243–248, Paris, France, 2020.

86. Scikit-Learn, “RandomOverSampler”, `https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.html`, accessed on November 18, 2023.

87. Scikit-Learn, “RandomUnderSampler”, `https://imbalanced-learn.org/stable/references/generated/imblearn.under_sampling.html`, accessed on November 18, 2023.

88. Rodriguez, P. L. and A. Spirling, “Word Embeddings: What Works, What Doesn’t, and How to Tell the Difference for Applied Research”, *The Journal of Politics*, Vol. 84, No. 1, pp. 101–115, 2022.

89. Torregrossa, F., R. Allesiardo, V. Claveau, N. Kooli, and G. Gravier, "A Survey on Training and Evaluation of Word Embeddings", *International Journal of Data Science and Analytics*, Vol. 11, pp. 85–103, 2021.

90. Incitti, F., F. Urli, and L. Snidaro, "Beyond Word Embeddings: A Survey", *Information Fusion*, Vol. 89, pp. 418–436, 2023.

91. Mikolov, T., K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space", *1st International Conference on Learning Representations*, pp. 23–35, Scottsdale, AZ, USA, 2013.

92. Köksal, A., "Turkish-Word2Vec Repository", `https://github.com/akoksal/Turkish-Word2Vec`, accessed on November 18, 2023.

93. Inzva, "Turkish-GloVe Repository", `https://github.com/inzva/Turkish-GloVe`, accessed on November 18, 2023.

94. Bojanowski, P., E. Grave, A. Joulin, and T. Mikolov, "Enriching Word Vectors with Subword Information", *Transactions of the Association for Computational Linguistics*, Vol. 5, pp. 135–146, 2017.

95. FastText, "FastText Documentation Crawl Vectors", `https://fasttext.cc/docs/en/crawl-vectors.html`, accessed on November 18, 2023.

96. Peters, M. E., M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep Contextualized Word Representations", *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Vol. 1, pp. 2227–2237, Seattle, United States, 2018.

97. Che, W., Y. Liu, Y. Wang, B. Zheng, and T. Liu, "Towards Better UD Parsing: Deep Contextualized Word Embeddings, Ensemble, and Treebank Concatenation", *Proceedings of the CoNLL Shared Task: Multilingual Parsing from Raw*

Text to Universal Dependencies, pp. 55–64, Brussels, Belgium, 2018.

98. Wolf, T., L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, *et al.*, "Transformers: State-of-the-Art Natural Language Processing", *Proceedings of the Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, Lisbon, Portugal, 2020.

99. Wu, Y., M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, *et al.*, "Google's Neural Machine Translation System: Bridging the Gap Between Human and Machine Translation", *arXiv:1609.08144*, 2016.

100. Face, H., "Hugging Face Documentation", `https://huggingface.co`, accessed on December 12, 2023.

101. Kingma, D. and J. Ba, "Adam: A Method for Stochastic Optimization", *International Conference on Learning Representations*, 2015.

102. Loshchilov, I. and F. Hutter, "Decoupled Weight Decay Regularization", *International Conference on Learning Representations*, 2019.

103. Face, H., "BERT Model for Turkish", `https://huggingface.co/dbmdz/bert-base-turkish-128k-uncased`, accessed on December 12, 2023.

104. Face, H., "BERT Model for Turkish cased", `https://huggingface.co/dbmdz/bert-base-turkish-cased`, accessed on December 12, 2023.

105. 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", *Computing Research Repository*, 2019.

106. Aytan, B., "Turkish RoBERTa Model", `https://huggingface.co/burakaytan/`

`roberta-base-turkish-uncased`, accessed on December 12, 2023.

107. Face, H., “DistilBERT Model for Turkish”, `https://huggingface.co/dbmdz/distilbert-base-turkish-cased`, accessed on December 12, 2023.

108. Aydın, and E. Karaarslan, “Is ChatGPT Leading Generative AI? What is Beyond Expectations?”, *Academic Platform Journal of Engineering and Smart Systems*, Vol. 11, pp. 125–136, 2023.

109. Sætra, H. S., “Generative AI: Here to stay, but for good?”, *Technology in Society*, Vol. 75, pp. 372–386, 2023.

110. Kumar, S., D. Musharaf, S. Musharaf, and A. K. Sagar, “A Comprehensive Review of the Latest Advancements in Large Generative AI Models”, *International Conference on Advanced Communication and Intelligent Systems*, pp. 90–103, Prague, Czech Republic, 2023.

111. Brown, T. B., B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, “Language Models are Few-Shot Learners”, *arXiv:2005.14165*, 2020.

112. OpenAI, “GPT-3.5 Turbo: A More Powerful and Efficient Version of GPT-3”, `https://platform.openai.com/docs/models/gpt-3-5`, accessed on December 12, 2023.

113. OpenAI, “Completions API Documentation”, `https://platform.openai.com/docs/guides/text-generation/completions-api`, accessed on December 12, 2023.

114. OpenAI, “Chat Completions API Documentation”, `https://platform.openai.com/docs/guides/text-generation/chat-completions-api`, accessed on December 12, 2023.

115. Giray, L., "Prompt Engineering with ChatGPT: A Guide for Academic Writers", *Annals of Biomedical Engineering*, pp. 1–5, 2023.

116. White, J., Q. Fu, S. Hays, M. Sandborn, C. Olea, H. Gilbert, A. Elnashar, J. Spencer-Smith, and D. C. Schmidt, "A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT", 2023.

117. Bao, Y., M. Wu, S. Chang, and R. Barzilay, "Few-shot Text Classification with Distributional Signatures", *arXiv:1908.06039*, 2019.

118. Wei, J., X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, *et al.*, "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models", *Advances in Neural Information Processing Systems*, Vol. 35, pp. 24824–24837, 2022.

119. Wortsman, M., G. Ilharco, J. W. Kim, M. Li, S. Kornblith, R. Roelofs, R. G. Lopes, H. Hajishirzi, A. Farhadi, H. Namkoong, *et al.*, "Robust Fine-Tuning of Zero-Shot Models", *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Vol. 3, pp. 7959–7971, Paris, France, 2022.

120. Guzman, E., L. Oliveira, Y. Steiner, L. C. Wagner, and M. Glinz, "User Feedback in the App Store: A Cross-Cultural Study", *IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Society*, Vol. 13, pp. 13–22, Hannover, Germany, 2018.

APPENDIX A: USE OF VISUALS

The images that were produced during the research for this thesis and have been copyrighted by the publisher, have been used in the thesis book in compliance with the publisher's "publication policy for reusing the author's own written works and graphics," which can be found on the publisher's website.