MANCHESTER 1824
The University of Manchester

Classification of App Reviews for Requirements Engineering using Deep Learning Models

A dissertation submitted to The University of Manchester for the degree of
Bachelor of Science in Computer Science
in the Faculty of Science and Engineering

Year of submission
2023

Student ID
10634648

School of Engineering

Contents

Contents **2**

List of figures **4**

List of tables **5**

Abstract **6**

Declaration of originality **7**

Intellectual property statement **8**

1 Introduction **9**
1.1 Context and Motivation 9
1.2 Aims and Objectives 9
1.3 Report Structure 9

2 Background and Related Work **10**
Background 10
2.1 App Reviews 10
2.2 Machine Learning Models 11
2.3 Supervised Learning 11
2.4 Neural Networks 12
2.5 Machine Learning Model Evaluation 12
2.6 Transformer-Based Language Models 14
Related Work 17
2.7 On the automatic classification of app reviews 17
2.8 How Can I Improve My App? Classifying User Reviews for Software Maintenance and Evolution 18
2.9 Convolutional Neural Network Based Classification of App Reviews 18
2.10 SAFE: A Simple Approach for Feature Extraction from App Descriptions and App Reviews 19
2.11 Ensemble Methods for App Review Classification: An Approach for Software Evolution 19

3 Research Methodology **20**
3.1 Collecting Data 20
3.2 Data Preprocessing 21
3.3 Defining Classification Tasks 22
3.4 Model Selection 22
3.5 Selection of Implementation Technologies 23
3.6 Model Building and Training 24

3.7 Model Evaluation and Performance Measures 27

4 Result Analysis **28**
4.1 Results 28
4.2 Comparison of Model Performances 30

5 Research Evaluation **32**
5.1 Comparison to Related Work 32
5.2 Reflection and Lessons Learned 33

6 Conclusion **33**
6.1 Achievements 34
6.2 Future Work 34

References **36**

Word count: 10094

List of figures

1 A Plot to Show the Average Accuracies of Different Machine Learning Algorithms Using 75%-25% Split 29
2 A Plot to Show the Average Accuracies of Different Machine Learning Algorithms Using 10-Fold Cross Validation 29

List of tables

1 Final Dataset 22
2 Precision, Recall, F1-score and AUROC score received from training using 75%-25% Train-Test Split 28
3 Precision, Recall, F1-score and AUROC score received from training using 10-Fold Cross Validation 28

Abstract

App stores, such as Google Play and the Apple Store, have revolutionised the way billions of users interact with apps by enabling downloads, usage, and reviews. These platforms facilitate the exchange of app-related information between users and developers. However, a significant proportion of app reviews are non-informative, comprising of generic praises or irrelevant details. This report addresses this issue by proposing an automatic classification system that utilises advanced deep language models like BERT, ROBERTa, and DistilBERT to classify app reviews into five distinct categories: problem discovery, feature request, user experience feedback, text ratings, and information giving. Moreover, the report acknowledges that app reviews can also be informative, encompassing categories such as app functionality, design, performance, and usability, which provide valuable insights for developers to improve their apps and enhance user satisfaction. We describe two different experiments to train and classify the deep learning models, one where we perform a 75%-25% train-test split on the dataset, and another where we perform a 10-Flod Cross Validation on the dataset. We evaluated the performance of the deep language models on 3735 reviews that were combined from two separate datesets. Overall, the BERT model trained using a 10-Fold Cross Validation had a better performance than the rest of the deep language models, with an average precision of 0.88 and 0.76 recall.

Declaration of originality

I hereby confirm that this dissertation is my own original work unless referenced clearly to the contrary, and that no portion of the work referred to in the dissertation has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Intellectual property statement

i The author of this thesis (including any appendices and/or schedules to this thesis) owns certain copyright or related rights in it (the "Copyright") and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.

ii Copies of this thesis, either in full or in extracts and whether in hard or electronic copy, may be made *only* in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has from time to time. This page must form part of any such copies made.

iii The ownership of certain Copyright, patents, designs, trademarks and other intellectual property (the "Intellectual Property") and any reproductions of copyright works in the thesis, for example graphs and tables ("Reproductions"), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.

iv Further information on the conditions under which disclosure, publication and commercialisation of this thesis, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see `http://documents.manchester.ac.uk/DocuInfo.aspx?DocID=24420`), in any relevant Dissertation restriction declarations deposited in the University Library, and The University Library's regulations (see `http://www.library.manchester.ac.uk/about/regulations/_files/Library-regulations.pdf`).

1 Introduction

1.1 Context and Motivation

App stores like Google Play and the Apple Store have transformed the way billions of users interact with apps. They enable users to download, use, and review apps and also facilitate the exchange of app-related information between users and developers. However, a significant proportion of app reviews are non-informative, consisting of generic praises or irrelevant details. These types of reviews can make it challenging for developers to understand user feedback and improve their apps to enhance user satisfaction. Additionally, the report acknowledges that app reviews can also be informative, encompassing categories such as app functionality, design, performance, and usability, which provide valuable insights for developers to improve their apps. By automating the classification process, developers can more efficiently analyse and utilise these informative reviews to improve their apps further.

1.2 Aims and Objectives

The aim of this report is to propose an automatic classification system that utilises advanced deep language models like BERT, ROBERTa, and DistilBERT to classify app reviews into five distinct categories: problem discovery, feature request, user experience feedback, text ratings, and information giving. The objectives of this report are to:

- Address the issue of non-informative app reviews that comprise of generic praises or irrelevant details
- Provide a solution that enables developers to classify app reviews efficiently and effectively
- Enhance user satisfaction by improving the quality of the apps through the analysis of informative app reviews
- Demonstrate the effectiveness of deep learning models in classifying app reviews in a multi-class classification task.

1.3 Report Structure

The report is structured as follows:

- **Background and Related Work** provides an overview of the context and motivation for this study, as well as an examination of the existing research in the field of app review classification.

- **Research Methodology** outlines the specific methods and techniques used in the study, including data collection and pre-processing, model development and evaluation, and training procedures.
- **Results Analysis** presents the findings of the study and evaluates the performance of the developed models.
- **Research Evaluation** reflects on the strengths and limitations of the study and comparison to related works.
- **Conclusion** summarises the key insights and contributions of the study and discusses potential avenues for future research.

2 Background and Related Work

Background

2.1 App Reviews

The number of available mobile apps is continuously growing, with over 3 million apps as of July 2014 [1]. Mobile apps are easily accessible through various means such as the App Store, Google Play Store, and Windows Phone Store with hundreds of billions of downloads between 2011 and 2016 [1]. Anyone with access to a smartphone or tablet can easily search, download and install apps with just a few clicks on their screen. Alongside downloading apps, users have the option to leave a review by giving a star rating and providing textual feedback on their experience using the app, for the benefit of other users or the developers themselves.

Research has highlighted the importance of app reviews for the success of apps [2]. Applications with a greater number of reviews obtain a higher ranking in search results and tire lists. This then results in increased visibility, higher sales, and a greater number of downloads [3]. These reviews are not only used by other users, to make an informed decision if they want to use a particular app, but also by developers themselves to gain beneficial insights about their app.

Many apps receive over a thousand reviews per day [4], some of which include valuable information related to requirements, such as bug reports [4], summaries of user experience with certain features [5], enhancement requests [6], and even ideas for potential new features [4], [7]. However, it is important to note that not all app reviews are of high quality. There are often low-quality reviews that contain meaning less information, insulting comments, spam, or simply repeat the star rating in words. With the sheer volume of reviews submitted daily for popular apps, it can be challenging for developers to filter and process the useful information from the reviews.

2.2 Machine Learning Models

According to Arthur Samuel Machine learning is defined as the field of study that gives computers the ability to learn without being explicitly programmed [8]. Machine learning models are algorithms or mathematical representations that are designed to automatically learn from and make predictions or decisions based on a set of data. They are a subset of artificial intelligence that uses statistical techniques to enable computers to learn from and improve their performance on a specific task without being explicitly programmed.

Machine learning models work by being exposed and trained on a large dataset, which contains examples of input data and their corresponding output or target values. During training, the model analyses the data, identifies patterns, and adjusts its internal parameters or weights in order to minimise the difference between its predictions and the actual target values. Once the model is trained, it can then be used to make predictions or decisions on new, unseen input data.

Machine learning has successfully been applied in various areas, including pattern recognition, natural language processing, computer vision, spacecraft engineering, finance, entertainment, and computational biology, as well as biological and medical applications [9]. Machine learning models require careful selection of algorithms, appropriate training data, feature engineering, and model evaluation to ensure their effectiveness and accuracy.

It's crucial to bear in mind that machine learning models might include flaws including biases, overfitting, and interpretability problems [10]. Consequently, it is essential to carefully assess and validate machine learning models before using them in real-world applications. Additionally, it is important to regularly monitor their performance to make sure they continue to be reliable as well as efficient.

2.3 Supervised Learning

Supervised learning is a widely used type of machine learning where the model learns to predict or classify output variables based on a set of labelled input variables. This involves learning a mapping between a set of input variables, denoted as X, and an output variable, denoted as Y [11], and then applying this mapping to make predictions for unseen data. One common formulation of supervised learning is the classification problem, where the learner approximates the behaviour of a function that maps a vector into one of several classes by looking at input-output examples of the function [11].

The workflow of supervised machine learning algorithms typically involves dividing the input dataset into a train dataset and a test dataset. The train dataset contains the labelled output variable that needs to be predicted or classified, and the algorithms learn patterns from this training dataset. These learned patterns are then applied to the test dataset for prediction or classification [8].

There are several commonly used models in supervised learning [12]. Linear regression is a simple model that models the relationship between input features and a continuous target variable. Decision trees are tree-like structures used for both regression and classification tasks. Support vector machines (SVMs) are powerful models that aim to find a hyperplane that best separates the data into different classes or predicts the target variable. Neural networks are highly flexible and powerful models inspired by the human brain, capable of handling a wide range of tasks. Ensemble methods, such as bagging, boosting, and stacking, combine multiple base models to make predictions or decisions, often resulting in improved performance.

2.4 Neural Networks

Neural networks, also known as artificial neural networks (ANNs), are a type of computational model that provides a range of powerful techniques for solving problems in pattern recognition, data analysis, and control [13]. ANNs are comprised of interconnected computational nodes, referred to as neurons, which work in a distributed fashion to collectively learn from input data and optimise their final output.

The basic structure of an ANN typically includes an input layer where the input data, usually in the form of a multidimensional vector, is inputted. This input data is then distributed to the hidden layers of the network. The hidden layers make decisions based on the input from the previous layer and weigh up how a stochastic change within themselves affects the final output, which is referred to as the process of learning. [14]

Neural networks have shown remarkable success in various applications, such as image recognition [15], speech recognition [16], natural language processing [17], and many more. They are capable of handling complex patterns and relationships within data, making them highly flexible and powerful models for solving challenging problems. The ability to learn from data and optimise their output based on learned patterns makes neural networks well-suited for tasks such as pattern recognition and data analysis.

2.5 Machine Learning Model Evaluation

2.5.1 Cross Validation

Model evaluation is a crucial stage in machine learning to evaluate a trained model's performance. One common technique used for model evaluation is K-Fold Cross Validation (CV). The method consists of partitioning the dataset into numerous folds, training the model on different subsets of the data, then assessing the model's performance on the remaining folds. To generate a reliable eval-

uation of a model's performance in a variety of machine learning tasks, including natural language processing (NLP), K-Fold CV is frequently used.

The process of K-Fold CV can be summarised as follows [18]:

- Dataset Partitioning: The original dataset is divided into K equally sized or nearly equally sized folds. Typically, K is set to a value of 5 or 10 [19], but it can be adjusted based on the size of the dataset and the computational resources available.
- Training and Evaluation: The model is trained K times, each time using K-1 folds as the training data and the remaining fold as the evaluation data. The model is then evaluated on the evaluation fold to obtain a performance metric, such as accuracy or F1 score.
- Average Performance: The performance metrics obtained from the K evaluations are averaged to obtain a single performance estimate for the model. This average performance is considered as the overall performance of the model.

K-Fold CV offers a number of advantages for model evaluation. First, by averaging a model's performance across numerous folds, it offers a more reliable evaluation of a model's performance while minimising the effects of data variability and noise. Second, because each fold is utilised for both training and evaluating the model, different data points are used for each fold's evaluation, ensuring that the model is evaluated on different data points in each fold. Thirdly, because the model is evaluated using various subsets of the data, it offers insights into the model's stability.

In NLP tasks, where the performance of a model can be influenced by a variety of factors such as the quality of the data, the selection of hyperparameters, and the inclusion of language-specific subtleties [20], K-Fold CV can be very helpful. Researchers and practitioners may choose the best-performing model and aid in making decisions regarding the model's deployment by utilising K-Fold CV to get a more precise evaluation of a model's performance.

It must be acknowledged that K-Fold CV has several drawbacks. Smaller values of K result in increased variance, whereas bigger values of K result in higher computing overhead, which might have an influence on the performance estimate. Furthermore, K-Fold CV might not be appropriate in other circumstances, such as when the dataset is extensive or when the data involves temporal dependencies. In such cases, alternative model evaluation techniques, such as time-based validation or stratified sampling, may be more suitable.

2.5.2 Performance Measure Metrics

When performing classification tasks, it's crucial to evaluate a trained model's effectiveness in terms of its capacity to accurately divide instances into several classes. The Area Under the Receiver Operating Characteristic (AUROC) score is a commonly used statistic for assessing the performance of classification models. The AUROC score is a graphical depiction of a classification model's performance that may be used to assess its overall prediction accuracy.

The Receiver Operating Characteristic (ROC) curve, which is a plot of the True Positive Rate (TPR) against the False Positive Rate (FPR) at various classification thresholds [21], is the foundation upon which the AUROC score is calculated. The proportion of true positive instances that the model correctly predicts is known as the TPR, often referred to as Sensitivity or Recall, whereas the proportion of false positive instances that the model mistakenly predicts as positive is known as the FPR.

The area under the ROC curve, which ranges from 0 to 1, is used to generate the AUROC score. A higher AUROC score indicates a better performance of a model, where 1 denotes a perfect classification and 0.5 denotes a random classification.

The AUROC score offers a number of advantages as a measurement tool for evaluating classification models. A fair evaluation of the model's performance is provided by taking into account both the true positive rate and the false positive rate. In many real-world classification problems, class imbalance is a problem that is frequently present. The AUROC score is insensitive to this problem. It also gives a visual depiction of the model's performance, making it simple to compare several models and understand their performance.

The AUROC score in NLP tasks can be especially helpful in evaluating the performance of models in tasks like sentiment analysis [22], spam detection [23], and medical diagnosis [24], where it is vital to correctly classify positive and negative instances. Additionally, it can be combined with other measures like recall, accuracy, and precision to get a more complete picture of the model's performance.

2.6 Transformer-Based Language Models

Transformers are a groundbreaking type of neural network architecture that has revolutionised natural language processing (NLP) and other fields of artificial intelligence. One of the key features of transformers is their ability to selectively attend to relevant parts of the input sequence, which results in high accuracy.

Unlike traditional models that process input sequentially, transformers use self-attention mechanisms to dynamically weigh the importance of each word/token in the input sequence. This allows the model to focus only on the relevant parts of the input to make its prediction, which contributes to their high accuracy [25]. This ability to selectively address relevant information makes transformers highly effective at capturing complex patterns and dependencies in data, leading to a state-of-the-art performance in many NLP tasks.

Transformers use continuous representations in their decoder to create an output sequence [26]. This allows for a smoother generation of sequences, as the decoder can generate output tokens based on the continuous representations learned from the input sequence. This continuous representation enables the model to generate coherent and meaningful output sequences, which is particularly beneficial for tasks such as text generation and machine translation.

In addition, the architecture of a transformer allows it to scale with the training data size and use parallel training, making it highly efficient for large-scale tasks. The self-attention layers used in transformers enable the model to capture long-range sequence features, allowing it to model complex dependencies even in lengthy input sequences [27]. This scalability and ability to capture long-range dependencies make transformers well suited for tasks that require processing of large amounts of data, such as language translation, document summarisation, and speech recognition.

However, it is worth noting that the computational requirements of transformers can be significant, and extensive training data is often required to achieve optimal performance. Fine-tuning and hyperparameter tuning are critical for achieving the best results with transformer models.

2.6.1 BERT

BERT, Bidirectional Encoder Representations from Transformers, is a revolutionary language representation model that has achieved remarkable performance in a wide range of natural language processing tasks. BERT has set new benchmarks in tasks such as question answering, text classification, named entity recognition, and sentiment analysis.

One of the key innovations of BERT is its bidirectional training approach, which allows it to capture contextual information from both left and right contexts of a token in the input sequence. Unlike traditional language models that process text sequentially in one direction, BERT uses a masked language model (MLM) objective during pre-training, where it randomly masks out some tokens in the input and the model is trained to predict the masked tokens [28]. This bidirectional training approach enables BERT to better understand the contextual relationships between words in a sentence, leading to improved language representation capabilities.

BERT also employs a transformer architecture, which consists of multiple layers of self-attention and feed-forward neural networks. The self-attention mechanism allows the model to weigh the importance of different tokens in the input sequence based on their contextual relevance, while the feed-forward neural network captures local patterns in the data. The combination of self-attention and feed-forward networks enables BERT to capture both local and global contextual information, making it highly effective for a wide range of NLP tasks [29].

One of the strengths of BERT is its ability to leverage large-scale unlabelled text data for pre-training, followed by fine-tuning on task-specific labelled data [26]. This allows BERT to transfer knowledge learned from a large corpus of text data to specific downstream tasks with limited labelled data, which is a common challenge in many NLP tasks. Fine-tuning BERT on task-specific data further refines its language representation capabilities, leading to state-of-the-art performance in various tasks.

However, BERT also has some limitations. The large-scale pre-training and fine-tuning processes of BERT can be computationally expensive and may require substantial computing resources [30]. The

model's size can also be large, which may limit its deployment in resource constrained environments.

2.6.2 RoBERTa

Robustly Optimised BERT Approach (RoBERTa) is a variant of BERT that builds upon the success of BERT and introduces further optimisations to enhance its performance. RoBERTa was developed by Facebook AI Research (FAIR) in 2019 [31] and has achieved advanced results in various natural language processing tasks, surpassing the performance of BERT in many cases.

One of the key optimisations in RoBERTa is the modification of the training approach. RoBERTa employs a larger training corpus compared to BERT, including more data from sources such as books, websites, and Wikipedia. This larger corpus allows RoBERTa to learn from a broader range of text, improving its language representation capabilities. RoBERTa also uses longer training sequences and removes the next sentence prediction (NSP) objective used in BERT, as studies have shown that NSP does not contribute significantly to the model's performance [32]. These modifications to the training approach in RoBERTa result in a more robust and optimised language representation model.

RoBERTa also introduces dynamic masking during pre-training, where the masked tokens are randomly re-selected during each epoch of training [31]. This dynamic masking helps RoBERTa to better adapt to different masking patterns and improves its ability to handle MLM training, where the model is trained to predict the masked tokens. This dynamic masking approach helps RoBERTa to further enhance its language representation capabilities and leads to improved performance in downstream NLP tasks.

Another notable optimisation in RoBERTa is the use of larger model sizes compared to BERT. RoBERTa introduces variations of model sizes, such as RoBERTa-base, RoBERTa-large, and RoBERTa-xlarge, which have more layers and hidden dimensions compared to BERT. These larger model sizes allow RoBERTa to capture more complex language patterns and representations, resulting in enhanced performance.

RoBERTa also retains the transformer architecture used in BERT, which consists of multiple layers of self-attention and feed-forward neural networks. This architecture allows RoBERTa to capture both local and global contextual information from the input sequence, enabling it to model complex language relationships effectively.

2.6.3 DistilBERT

DistilBERT, short for Distillation-based BERT, is a compressed and smaller version of the original BERT model that aims to reduce the computational resources required for training and inference while maintaining high performance. DistilBERT was developed by Hugging Face, a leading NLP research organisation, in 2019 [33].

DistilBERT's major goal is to take what has been learned from the huge BERT model and apply it to a more compact and effective model. A process known as "distillation" is utilised to train DistilBERT, which makes use of the knowledge from the bigger BERT model to direct the training of the smaller model. The smaller model may capture the same language representations with less complexity and computation by mirroring the output of the larger BERT model during training.

The primary benefit of DistilBERT over BERT is its reduced size. DistilBERT is made to have fewer hidden dimensions and layers, which leads to a smaller model that uses fewer training and inference resources. DistilBERT, therefore, becomes more suited for deployment in environments with restricted resources or on computing limited hardware, such as mobile or edge devices.

DistilBERT performs at a high level despite being more compact. According to studies, DistilBERT performs similarly to BERT or even slightly better on a variety of NLP tasks while requiring significantly fewer calculations and parameters [34]. Because of this, DistilBERT is a desirable alternative for applications like real-time processing and web applications which require quick and efficient inference.

DistilBERT also inherits the same transformer architecture used in BERT, which allows it to capture both local and global contextual information from the input sequence. This architecture enables DistilBERT to model complex language relationships and representations effectively, similar to BERT.

The main drawback of DistilBERT is that it has the potential to fall short of BERT on tasks that require a high level of fine-grained language comprehension or on tasks with little training data. DistilBERT's performance in some cases may be impacted by the possible loss of information from the original BERT model during the distillation process. However, given DistilBERT's decreased computational needs, the performance trade-off can often be acceptable.

Related Work

2.7 On the automatic classification of app reviews

The work reported by Maalej [1] introduced several probabilistic techniques for classifying app reviews into four types. They compared three different machine learning algorithms, namely Naïve Bayes, Decision Tree, and Maximum Entropy, using two classification techniques: bag of words and bag of words + metadata.

The experiment data for the study was collected by crawling the Apple AppStore and Google Play stores, and selecting the top apps in each category. The dataset included a total of 1.1 million reviews for 1100 apps from the Apple store and 146,057 reviews for 80 apps from the Google store. The dataset had an equal representation of paid and free apps in both stores, and it included review text, title, app name, category, store, submission date, username, and star rating.

In comparison to multiclass classifiers, binary classifiers were found to be more accurate at predicting various types of reviews. The fact that each binary classifier employed two training sets, one where the appropriate type is observed and one where it is not, could be one explanation. With the highest average score of 0.79 among the binary classifiers, Naive Bayes surpassed the competition. The multiclass classifiers had the highest average score of 0.50.

2.8 How Can I Improve My App? Classifying User Reviews for Software Maintenance and Evolution

This study by Panichella [35] presented a taxonomy to classify app reviews into categories relevant to software maintenance and evolution. To automatically categorise app reviews into the suggested categories, they suggested a method that combines three techniques: Natural Language Processing (NLP), Text Analysis (TA), and Sentiment Analysis (SA). The study evaluated five different machine learning algorithms for classification, including Bayes, Support Vector Machine (SVM), Logistic Regression, J48, and Alternating Decision Tree.

A collection of reviews compiled by Guzman and Maalej [5] was used to evaluate the research methodology. This collection of reviews covered the TripAdvisor, PicsArt, Pinterest, and WhatsApp applications for Android as well as the AngryBirds, Dropbox, and Evernote apps from Apple's App Store. This wide range of applications made it possible to classify reviews that used various vocabulary and were written by various user audiences in order to assess the integrity of the approach.

The results showed that the NLP+TA+SA configuration with the J48 algorithm achieved the best results among all possible feature inputs and classifiers, with a precision of 75% and a recall of 74%.

2.9 Convolutional Neural Network Based Classification of App Reviews

The study by Aslam [36] proposed a deep learning based approach for the classification of app reviews. The approach began by extracting both textual and non-textual data from each app review. It then pre-processed the textual data, computed the sentiment of app reviews using Senti4SD, and determined the reviewer's history, including the number of reviews they have submitted overall and their submission rate for the related app. A digital vector was then created for each app review. Finally, a CNN-based deep learning multi-class classifier was developed to train and classify app reviews.

The same dataset extracted by Maleej et al. [5] was used in this report. The dataset included 146,057 reviews for 80 Google Store applications and 1,126,453 reviews for 1100 Apple Store apps. The text, title, app name, category, store, date of submission, reviewer-id, and star rating were all included in each review.

The proposed CNN-based classifier's performance was evaluated in comparison to that of other classifiers, such as LSTM, Random Forest (RF), VM, Multi-nomial Naive Bayes (MNB), and Naive Bayes. The proposed classifier achieved the highest performance among the selected deep learning classifiers. It improved the average precision from 75.72% to 95.49%, average recall from 69.40% to 93.94%, and F-measure from 72.41% to 94.71%, respectively.

2.10 SAFE: A Simple Approach for Feature Extraction from App Descriptions and App Reviews

In the study [37] Johann investigates the task of feature extraction from app descriptions and reviews. They attempt to pinpoint the app features that are most frequently mentioned in both the description and user reviews. 18 part-of-speech patterns and 5 sentence patterns that are often used in text related to app features were carefully developed by the writers. Their method has the significant benefit of not requiring an extensive amount of training and configuration data.

The authors constructed a dataset by crawling the descriptions and reviews of 10 apps from the Apple App Store in order to assess their strategy. They chose the top five free and top five premium applications (as of January 2017) in the Apple App Store's Productivity category. They obtained the description page for each app using the iTunes Search API3. They collected 44 values of metadata, including the name, version, description, category, and price, from the app description page. The app reviews were collected programmatically using a custom tool that connects to an internal iTunes API. Ten values make up a review, including the name of the reviewer, the title, the description, the rating, and the date. For further analysis, the data was persisted inside a MongoDB database. The authors published the dataset on their website to enable replication.

According to the evaluation's findings, this approach has an average accuracy of 87% across the 10 examined apps. Additionally, 87% of the features extrapolated from user evaluations, and the app descriptions matched. In summary, this study proposes a straightforward yet efficient method for feature extraction from app descriptions and reviews that does not require a lot of training data or extensive configurations.

2.11 Ensemble Methods for App Review Classification: An Approach for Software Evolution

The study presented by Guzman [38] was a taxonomy for categorising app reviews into categories relevant to software evolution, and an approach that examined how well individual machine learning algorithms and their ensembles performed for automatically classifying app reviews. The performance of four distinct classifiers: Naive Bayes, Support Vector Machines (SVMs), Logistic Regression,

and Neural Networks, as well as combinations of their predictions were compared by the researchers. The performance of the machine learning techniques on 4550 reviews that were systematically labelled using content analysis methods.

The results showed that the Logistic Regression and Neural Network classifiers had better precision than the Naive Bayes and SVM models. The Neural Network model had the highest F-measure average among all individual classifiers, whereas the SVM and Neural Network models had the highest recall values. Furthermore, the ensembles had a better performance than the individual classifiers, with an average precision of 0.74 and 0.59 recall.

3 Research Methodology

My project has followed a systematic approach, consisting of the following steps:

- Step 1: Data Collection
- Step 2: Data Preprocessing
- Step 3: Defining Classification Tasks
- Step 4: Model Selection
- Step 5: Implementation Technologies
- Step 6: Model Building and Training
- Step 7: Model Evaluation and Performance Measures

These steps are detailed in the following sections:

3.1 Collecting Data

The dataset [39] produced by Hawari was used in this study. The dataset has been obtained from two different sources. The first dataset, referred to as 'Pan Dataset' contains reviews of various apps such as AngryBirds, Dropbox, Evernote, TripAdvisor, PicsArt, Pinterest, and WhatsApp. It consists of 1390 reviews, which were classified into four classes related to software engineering's maintenance tasks, including 192 reviews as Feature Request (FR), 494 reviews as Problem Discovery (PD), 603 reviews as Information Gaing (IG), and 101 reviews as Information Seeking (IS).

The second dataset, referred to as 'Maalej Dataset' contains 3691 reviews from different Google Play and Apple App Store apps. All reviews were classified into four classes related to software engineering's maintenance tasks, including 252 reviews as FR, 370 reviews as PD, 607 reviews as User Experience (UE), and 2461 reviews as Rating (RT).

The categories of app reviews are defined below:

- Feature Request: sentences expressing ideas, suggestions or needs for improving or enhancing the app or its functionalities.
- Problem Discovery: sentences describing issues with the app or unexpected behaviours.
- Information Giving: sentences that inform or update users or developers about an aspect related to the app.
- Information Seeking: sentences related to attempts to obtain information or help from other users or developers.
- User Experience: sentences summarising user experience with features in the app.
- Ratings: sentences that reflect the numeric star rating.

Both datasets were used to train and evaluate the performance of the machine learning models used in this research. The combination of these datasets provides a diverse range of reviews from various sources, which can help improve the generalisation of the models. The reviews were preprocessed before being fed into the models, and the details of the preprocessing steps are explained in the next section.

3.2 Data Preprocessing

Data preprocessing is an essential step in any machine learning project. In this study, we first combined the two datasets obtained to increase the total number of reviews and improve our model's performance. However, before combining the datasets, we removed the 'information seeking' class from the Pan dataset since these reviews were not relevant or meaningless to software engineering requirements. We also relabelled the class names to be consistent between the two datasets, which will be defined in the next section. This step was necessary to ensure that the datasets could be easily combined into a single dataset.

The combined dataset has a total of 4980 reviews, which were classified into five classes related to software engineering requirements, including 864 reviews as PD, 2452 reviews as RT, 444 reviews as FR, 607 reviews as UE, and 603 reviews as IG.

Most of the preprocessing of individual reviews is done in the BERT preprocessor layer. The BERT preprocessor layer tokenises the review and converts it into a format that can be understood by the BERT model. It lowercases the text, splits words into tokens, adds special tokens, and converts each token into its corresponding ID from the BERT vocabulary. It also pads and truncates the sequences to ensure that they are all the same length. This process helps to normalise the data and make it suitable for model training.

One issue we encountered was a severe imbalance in the number of reviews per class. To address this issue, we carried out undersampling. Undersampling is a technique used to balance the dataset by randomly removing samples from the majority class, in this case, the RT class, to match the number of samples in the minority class FR, PD, UE, and IG. This was achieved by selecting a balanced number

of reviews from each class to ensure that there was an equal representation of each class in the final dataset. By doing so, the model would not be biased towards predicting the majority class and would be able to learn more effectively from the data.

Software Engineering Tasks	Number of Reviews
problem discovery	637
rating	1856
feature request	342
user experience	451
information giving	449
Total	**3735**

Table 1. Final Dataset

3.3 Defining Classification Tasks

In this study, we address two different multiclass classification tasks to evaluate the performance of the models. The first task involves training the model on 75% of the dataset and testing it on the remaining 25%. This approach is commonly used to evaluate the performance of models on unseen data. The second approach we used was a model trained using 10-fold cross-validation, which means splitting the dataset into 10 equal parts, using 9 parts for training, and 1 part for testing. This process was repeated 10 times, with each part being used for testing once.

For both classification tasks, the models were trained to predict the software engineering requirement class of each review. We considered five software engineering requirement classes: Feature Request, Problem Discovery, User Experience, Rating, and Information Giving. The FR class corresponds to reviews that suggest adding new features to the app. The PD class includes reviews that report issues or problems with the app. The UE class contains reviews that express the user's overall experience with the app. The RT class contains reviews that rate the app, and the IG class includes reviews that inform developers about an aspect related to the app.

Our aim was to build a deep learning language model that could accurately classify each review into one of the five software engineering requirement classes, as this information can help software engineers understand the needs and expectations of their users.

3.4 Model Selection

In this study, we used the software engineering requirement reviews dataset to conduct classification tasks using three distinct deep learning language models, namely BERT, RoBERTa, and DistilBERT. These models were chosen because they excelled in a number of NLP tasks, such as text classification, sentiment analysis, and question answering. These models can capture complicated language

patterns in the dataset since they include millions of parameters and have been pre-trained on a vast quantity of data.

The pre-trained model BERT, formerly referred to as Bidirectional Encoder Representations from Transformers, was created by Google AI Language. The model can examine the complete input sequence rather than just the left or right context because it employs a transformer design that enables bidirectional language modelling. On a number of NLP tasks, such as the GLUE benchmark, SQuAD, and sentiment analysis, BERT has demonstrated outstanding performance.

Another pre-trained model based on the BERT architecture is called RoBERTa, which stands for Robustly Optimised BERT Approach. However, RoBERTa outperformed BERT on a number of NLP tasks since it was trained on a bigger corpus and using superior training methods. On the GLUE benchmark as well as a number of other benchmarks, such as SuperGLUE and ReCoRD, RoBERTa has produced superior results. DistilBERT, which stands for Distilled BERT, is a variant of BERT that is lighter and quicker while preserving the same performance as the original BERT model. DistilBERT is easier to train and utilise for inference since it has fewer parameters than BERT. However, the model's performance in comparison to the original BERT model is also impacted by the fewer parameters.

For text classification tasks on the software engineering maintenance reviews dataset, the choice of these three models offers a thorough comparison of the most recent deep learning models. To decide which model is most suited for the classification tasks, the effectiveness of each model will be assessed and evaluated.

3.5 Selection of Implementation Technologies

For this study, we used a combination of different open source libraries and tools to implement our deep learning models and perform data analysis. We used Pandas [40] and NumPy [41] for data preprocessing and manipulation. These libraries provide efficient and flexible data structures for working with large datasets. PyTorch [42], an open-source machine learning framework that provides a Python interface for building and training neural networks, was used for building and training the models. We utilised the Transformers models from the Hugging Face library, which provided us with pre-trained models for BERT [43], RoBERTa [44], and DistilBERT [45]. For evaluation and comparison purposes, we used the scikit-learn library [46]. Sklearn is a popular machine learning library in Python that provides a range of tools for data preprocessing, model selection, and evaluation. Matplotlib [47] is a powerful plotting library in Python that provides a variety of plotting functions for creating different types of visualisations. We used this library to visualise and compare the performance of our models.

These technologies have the advantage of being extensively supported and employed, which makes it simpler to locate resources and documentation when problems emerge. When creating custom models with PyTorch, you have a lot of freedom, which is especially helpful when using ad-

vanced deep learning architectures like BERT, RoBERTa, and DistilBERT. Additionally, a wide collection of pre-trained models are available for these architectures in the Hugging Face Transformers library, making it simpler to fine-tune them for our specific application.

3.6 Model Building and Training

3.6.1 Model Architecture

The machine learning model architecture is a critical component of any machine learning project, as it determines the structure and behaviour of the model. In this project, we use a PyTorch implementation of a transformer-based neural network BERT, RoBERTa or DistilBERT for classifying app review. The model consists of a BERT model followed by a linear classifier layer, which is responsible for predicting the output classes.

The BERT model takes as input the text of a given app review, represented as a sequence of word embeddings, along with attention masks that indicate which tokens in the sequence are padding tokens. The purpose of the BERT model is to generate a sequence of contextualised embeddings for each token in the input sequence, which captures the meaning and context of the text. These embeddings are learned through the pre-training process, where the BERT model is trained on a large corpus of text to predict missing words in the input sequences.

The linear classifier layer is responsible for predicting the output classes. It takes as input the contextualised representation of the input tokens from the BERT model and maps them to the output classes using a linear transformation. Specifically, the linear layer processes the output of the BERT model to generate classification outputs for each of the five software engineering task classes. The purpose of this layer is to map the learned representations from the BERT model to the output space of the classification task.

The sigmoid activation function is applied to the output of the linear layer to produce binary classification outputs for each of the five classes. The purpose of the sigmoid function is to transform the linear output to a probability value between 0 and 1, which can be interpreted as the predicted probability of each label being present in the app review.

3.6.2 Objective Function Selection

We decided to use the binary cross-entropy loss as our objective function for our multiclass classification task. Binary cross-entropy loss may be implemented for multiclass classification tasks even though it is primarily employed for binary classification tasks. In our situation, we treated each class

individually and trained a different binary classifier for each class in order to adjust the binary cross-entropy loss to handle multiclass classification. This categorisation is often referred to as ”one-vs-all” or ”one-vs-rest”.

Because binary cross-entropy loss is well-suited for unbalanced datasets, which are frequently the case in NLP applications, we picked it over other more commonly used loss functions like categorical cross-entropy loss. Additionally, binary cross-entropy loss evaluates the difference between true labels and predicted probabilities for each class independently, and it also has a clear interpretation. The performance of the model is therefore simple to analyse and understand.

Ultimately, binary cross-entropy loss helped us successfully train our model for multiclass classification and obtain high accuracy on the test set.

3.6.3 Optimiser Selection

The choice of optimiser is crucial in determining the convergence and generalisation performance of the machine learning model. In this work, we selected the AdamW optimiser, a variant of the Adam optimiser that incorporates weight decay during the weight update step. This helps to prevent overfitting and improve generalisation performance.

The learning rate for the optimiser was set to 2e-5, following the fine-tuning recommendations from the original BERT paper [24]. To further improve convergence and prevent divergence, we employed a linear learning rate scheduler with warmup. The scheduler was configured to gradually increase the learning rate during the initial warmup phase and then decrease it linearly towards the end of the training phase. The number of warmup steps and training steps were determined based on the size of the dataset and the complexity of the model.

In addition, we used the early stopping technique to prevent overfitting and achieve optimal generalisation performance. The early stopping criterion was set to monitor the validation loss, and the training process was terminated when the validation loss did not improve for two epochs.

The selected optimisation strategy helped to achieve a fast and stable convergence of the model while preventing overfitting and improving generalisation performance.

3.6.4 Hyperparameter Optimisation

Hyperparameters are parameters that are set prior to training and cannot be learned from the data. The optimal choice of hyperparameters can significantly impact the performance of the machine learning model. In this work, we performed hyperparameter optimisation using an automated technique called Grid Search. Grid search is a popular technique for hyperparameter tuning, which involves testing a pre-defined set of hyperparameters to identify the optimal combination.

We defined a grid of hyperparameters, including the learning rate, batch size, number of epochs, number of warmup and training steps, and searched over all possible combinations of these hyperparameters. We used the validation set to evaluate the performance of each combination and selected the hyperparameters that produced the highest validation accuracy.

We also employed early stopping to prevent overfitting during hyperparameter optimisation. Specifically, we monitored the validation loss during the search and stopped the search if the validation loss did not improve after 2 epochs.

Through grid search, we were able to identify the optimal hyperparameters for our model, which resulted in improved performance on the test set.

3.6.5 Model Training

We applied two alternative techniques to train our models:

Approach 1: Train-Test Split: In the first approach, we randomly split the combined dataset into a training set (75%) and a testing set (25%). We then trained our deep learning models on the training set using the Adam optimiser with a learning rate of 2e-5 for 10 epochs. The batch size used was 8. We used a binary cross-entropy loss function, and early stopping was employed if the validation loss did not improve after 2 epochs.

Approach 2: 10-Fold Cross Validation: For the second approach, we used 10-fold cross-validation to train and evaluate our models. We divided the combined dataset into 10 folds, with each fold containing an equal proportion of samples from each class. We then trained our models on each of the 10 folds, using the same hyperparameters as in approach 1. For each fold, we computed the performance measures of interest (accuracy, AUROC score, precision, recall, and F1-score) and averaged them across all folds.

To monitor the model's development and prevent overfitting, we utilised the performance on the validation set during training. In order to stop the model from continuing to train when it converged on the training set, we also used early stopping.

In both approaches, we used the PyTorch library and the Transformers models from Hugging Face. The training was done in Google Colab using the free licence and limited resources.

We employed these rigorous training procedures to ensure that our models were well-trained and robust, and we evaluated their performance using various metrics to provide a comprehensive analysis of their effectiveness.

3.7 Model Evaluation and Performance Measures

To evaluate the performance of our models, we used a combination of performance measures commonly used in multi-class classification tasks. These metrics offer different perspectives on the models' performance and usefulness in various contexts. The measures we used include Accuracy, AUROC score, Precision, Recall, and F1-score.

In multiclass classification, there are four possible outcomes when a model makes a prediction for a given class: True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN). A true positive is when the model correctly predicts a positive instance, a false positive is when the model incorrectly predicts a positive instance, a true negative is when the model correctly predicts a negative instance, and a false negative is when the model incorrectly predicts a negative instance.

When evaluating the performance of our models, we also considered two additional metrics: True Positive Rate (TPR) and False Positive Rate (FPR). TPR is the proportion of actual positive instances that are correctly identified as positive, while FPR is the proportion of actual negative instances that are incorrectly identified as positive.

Now, let's discuss each of the performance measures used in our evaluation:

- **Accuracy**: This is the proportion of correct predictions made by the model over the total number of predictions. In other words, it measures how often the model correctly predicts the correct class. While accuracy is a useful metric for evaluating model performance, it can be misleading in situations where the data is imbalanced. In our study, we used accuracy as the primary measure for comparing the performance of our models.
- **AUROC Score**: The Area Under the Receiver Operating Characteristic Curve is a measure of the trade-off between TPR and FPR across different probability thresholds. It provides a useful summary of the overall performance of a model across all possible thresholds. We used AUROC score as a secondary measure to assess the models' performance as this allows us to see how well our model it at classifying each category of software engineering tasks.
- **Precision**: Precision is the proportion of true positive predictions over the total number of positive predictions. It measures how often the model correctly predicts the positive class when it makes a prediction. Precision is useful when the cost of false positives is high. In our study, we used precision to evaluate the model's ability to correctly identify positive instances.
- **Recall**: This is the proportion of true positive classifications over the total number of actual positive instances. Recall is useful when the cost of false negatives is high. In our study, we used recall to evaluate the model's ability to identify all positive instances.
- **F1-score**: This is the harmonic mean of precision and recall. It provides a balance between the two measures, making it a useful metric when both precision and recall are important.

Each of these performance measures provides a different perspective on the model's performance and allow us to obtain a more comprehensive understanding of their performance. While

accuracy is a useful metric, it can be misleading in situations where the data is imbalanced. AUROC score provides a more nuanced understanding of model performance across different thresholds. Precision and recall are useful metrics for evaluating the model's performance on positive instances, while the F1-score provides a balance between the two.

4 Result Analysis

The following section begins by presenting the findings from this study and then goes on to analyse the results from both experiments.

4.1 Results

	BERT				RoBERTa				DistilBERT			
	precision	recall	F1-score	AUROC	precision	recall	F1-score	AUROC	precision	recall	F1-score	AUROC
problem discovery	0.75	0.80	0.78	0.93	0.71	0.66	0.68	0.88	0.84	0.48	0.61	0.91
rating	0.94	0.56	0.70	0.90	0.88	0.67	0.76	0.90	0.90	0.60	0.72	0.90
feature request	0.51	0.45	0.48	0.91	0.40	0.55	0.46	0.91	0.40	0.52	0.45	0.89
user experience	0.38	0.68	0.49	0.84	0.35	0.37	0.36	0.79	0.34	0.49	0.40	0.80
information giving	0.70	0.84	0.77	0.96	0.66	0.71	0.68	0.92	0.74	0.45	0.58	0.92

Table 2. Precision, Recall, F1-score and AUROC score received from training using 75%-25% Train-Test Split

	BERT				RoBERTa				DistilBERT			
	precision	recall	F1-score	AUROC	precision	recall	F1-score	AUROC	precision	recall	F1-score	AUROC
problem discovery	0.88	0.93	0.91	0.99	0.74	0.83	0.78	0.95	0.65	0.70	0.67	0.92
rating	0.90	0.93	0.91	0.97	0.85	0.77	0.81	0.92	0.74	0.92	0.82	0.90
feature request	0.89	0.45	0.60	0.98	0.57	0.67	0.62	0.94	0.63	0.25	0.36	0.90
user experience	0.68	0.46	0.55	0.93	0.43	0.41	0.42	0.87	0.53	0.08	0.14	0.84
information giving	0.93	0.96	0.95	0.99	0.83	0.68	0.75	0.96	0.73	0.49	0.58	0.92

Table 3. Precision, Recall, F1-score and AUROC score received from training using 10-Fold Cross Validation

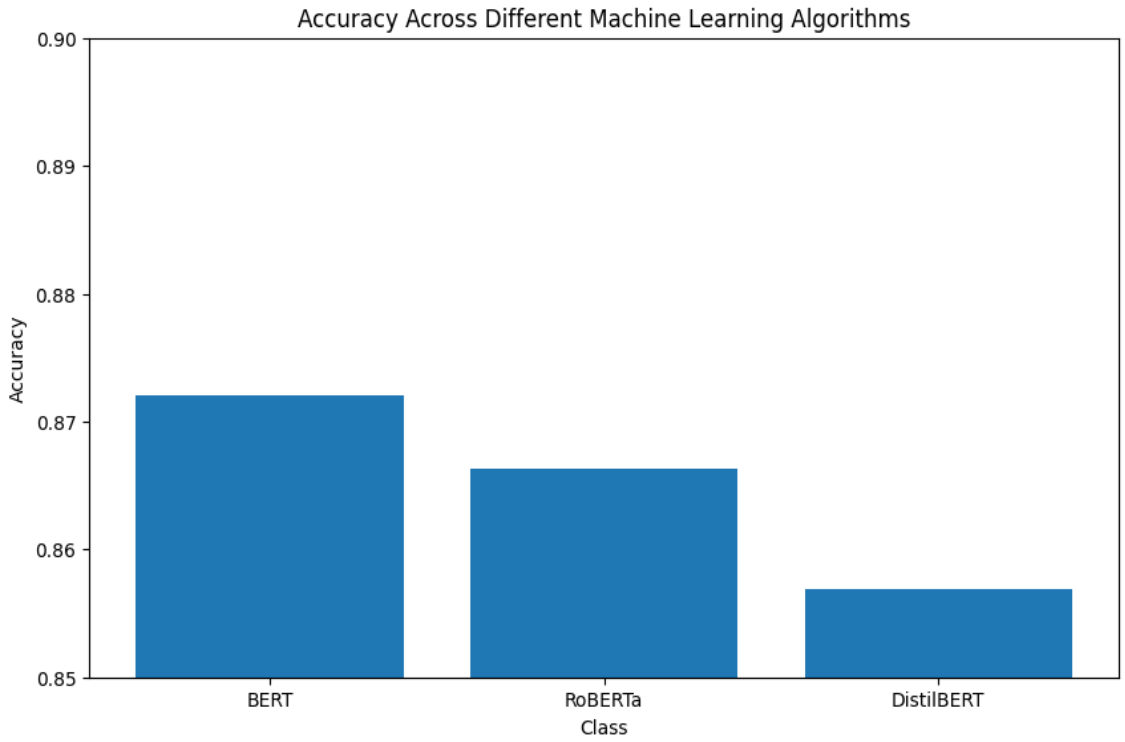

Fig. 1. A Plot to Show the Average Accuracies of Different Machine Learning Algorithms Using 75%-25% Split

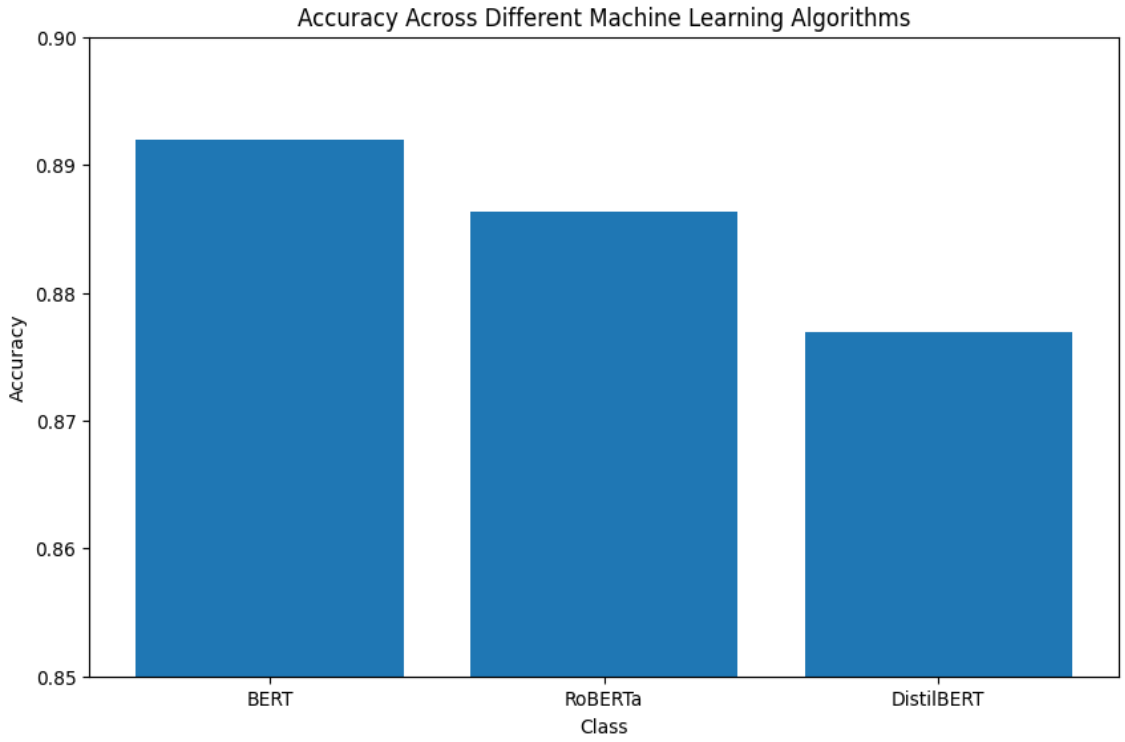

Fig. 2. A Plot to Show the Average Accuracies of Different Machine Learning Algorithms Using 10-Fold Cross Validation

4.2 Comparison of Model Performances

4.2.1 75%-25% Train-Test Split

In this section, we compare the performance of three popular pre-trained language models, BERT, RoBERTa, and DistilBERT, on five different software engineering tasks. The models were trained using a 75-25 train-test split, and the results were averaged over 10 trials.

The performance of each model varied depending on the task. Looking at Table 2 and Figure 1, BERT performed the best with an accuracy of 87%, and achieved the highest F1-scores and AUROC scores on several of the tasks, while RoBERTa and DistilBERT achieved lower scores. However, the performance of the models was not consistent across all tasks. For instance, RoBERTa achieved the highest recall score on the FR task, and DistilBERT achieved the highest recall score on the RT task.

One possible reason for the variation in performance is the differences in the architectures and training techniques used for each model. RoBERTa is a variation of BERT that was trained on a bigger corpus of data using extra pre-training approaches. BERT and RoBERTa have a similar architecture. Contrarily, DistilBERT is a distilled variation of BERT that was learned on a smaller dataset and with fewer training parameters.

The tasks' inherent nature is another factor for the difference in performance. The PD task, for instance, may require a model to find particular words or phrases that signify a problem, but the UE may require for a model to find more subtle patterns in user input. The difficulty of each task may require different strengths from each model, leading to variations in performance.

The variation in performance may also be influenced by the evaluation approach's weaknesses. For instance, the use of a single train-test split may not accurately represent the model's performance on unseen data, and the imbalance in the number of reviews across tasks has the potential to have an impact on the models' ability to make predictions.

4.2.2 10-Fold Cross Validation

In this experiment, we fine tune the BERT, RoBERTa, and DistilBERT models using 10-fold cross-validation and evaluated their performance on five different software engineering tasks, namely PD, RT, FR, UE, and IG.

Compared to the previous experiment, we observe a significant improvement in the performance of all three models when evaluated using 10-fold cross-validation, shown in Table 3. The average F1-scores for each task and model have improved substantially, indicating that cross-validation has helped improve the model's generalisation capability. For instance, the F1-score for PD has increased from 0.78 to 0.91 for BERT, from 0.68 to 0.83 for RoBERTa, and from 0.45 to 0.67 for DistilBERT.

Similarly, the F1-score for FR has increased from 0.48 to 0.60 for BERT, from 0.46 to 0.67 for RoBERTa, and from 0.45 to 0.36 for DistilBERT. This is further emphasised by the fact that the accuracies of all three models have greatly increased from the previous experience, shown in Figure 2.

The reason for this improvement is that 10-fold cross-validation allows us to train the model on a larger amount of data by iteratively using different parts of the dataset for training and testing. This approach reduces the risk of overfitting the model to the training data and improves its generalisation capability. Additionally, cross-validation also helps in identifying potential issues with the dataset and the model architecture, which can be addressed to improve the model's performance further.

When classifying reviews that belong to UE, all models had relatively low scores across all metrics. This suggests that the task may be more challenging or require a more nuanced understanding of user feedback. One possible reason why the UE task resulted in low scores across all metrics is that this type of feedback can be highly subjective and context dependent. The quality of the user experience is influenced by a wide range of factors such as usability, aesthetics, functionality, and emotional appeal. Therefore, it can be difficult to capture the essence of user feedback accurately without a deep understanding of the domain and the context in which the feedback was provided. Additionally, the UE task may require a more nuanced approach to modelling, as the meaning of certain phrases or expressions can be highly dependent on the context in which they are used. Furthermore, users may use a wide range of vocabulary and expressions to describe their experience, making it difficult for models to generalize effectively.

To improve the performance of the models on the UE task, it may be necessary to incorporate domain specific knowledge and contextual information into the training process. Also, additional pre-processing techniques such as sentiment analysis or topic modelling may be required to extract more meaningful information from user feedback.

Across both experiments, DistilBERT continuously under performed the other models. This could be due to its smaller size and reduced complexity compared to BERT and RoBERTa. DistilBERT is a smaller and faster version of BERT that uses knowledge distillation to compress the original BERT model while maintaining a similar level or performance. While this approach is effective in reducing the size and computational requirements of the model, it may also result in a loss of information and nuance in the learned representations. This loss of information can be particularly pronounced for our smaller dataset and potentially complex task, that require a more robust understanding of the underlying patterns in the data.

DistilBERT's small size may also make it less capable to identify the intricate connections between words and phrases, especially for more difficult tasks. This helps to explain why DistilBERT does not perform as well on tasks like FR and UE, which may call for a more complex understanding of the data.

Overall, these results demonstrate the importance of evaluating model performance on multiple tasks and using different evaluation approaches to gain a more comprehensive understanding

of their strengths and weaknesses. Additionally, it highlights the need for continued research and improvement in natural language processing models to better tackle complex software engineering tasks.

5 Research Evaluation

The section reflects on the strengths and limitations of the study and comparison to other related works.

5.1 Comparison to Related Work

In comparison to the related work by Maalej [1], our study achieved higher average F1-scores on all tasks, with the lowest average score of 0.52 still higher than the highest average score of 0.50 achieved by the multiclass classifiers in the related work. It is important to note that the datasets used in both studies were different, with our study including an additional subset of app reviews to Maalej's dataset. Additionally, our study used pre-trained transformer models, which have been shown to outperform traditional machine learning algorithms such as Naïve Bayes and Decision Trees.

As a result, our work highlights the potential for additional research in this field and shows the usefulness of pre-trained transformer models in categorising software engineering reviews.

In comparison to the study by Panichella [35] our model outperformed their approach in terms of precision, recall, and F1-score for all categories except for the FR category in recall and F1-score. However, it is important to note that the method by Panichella used a different set of techniques and algorithms, so a direct comparison may not be completely fair.

In comparison to the study by Aslam [36] our BERT model achieved an average accuracy of 0.873, which is lower than the performance of the proposed classifier in the study by Aslam. However, it is important to note that our study used a different approach and dataset.

In contrast to our study, the study by Johanna [37] does not perform any classification or sentiment analysis on the reviews. Instead, they aim to identify the app features that are most frequently mentioned in both the app description and user reviews. The results show that their approach has an average accuracy of 87% across the 10 examined apps, and 87% of the features extracted from user evaluations and the app descriptions matched. While the focus of their study is different from ours, their approach could be used in conjunction with ours to provide additional insights into the most frequently mentioned features in the reviews.

5.2 Reflection and Lessons Learned

In this study, we explored the effectiveness of using deep learning models to classify app reviews for software engineering requirements. Through the analysis of the results, we were able to draw several conclusions about the performance of the models and the effectiveness of our approaches.

The results indicate that the use of deep learning models can lead to high accuracy in classifying app reviews for software engineering requirements. The best performing model was found to be BERT with an accuracy of 89.2%.

However, the results also revealed several limitations of the study. One of the main limitations was the available resources to train the models. Due to the limited computational resources available, we were only able to train the models for a limited number of epochs, which may have affected their overall performance. Also, due to the limited size of the dataset, it may be difficult to generalise the results to other app reviews.

Through the course of this study, I learned several valuable lessons about the use of deep learning models for the automatic classification of app reviews. One of the main lessons learned was the importance of rigorous training procedures, including the use of cross validation and hyperparameter tuning, to ensure optimal model performance. I learned the importance of addressing class imbalance through techniques such as undersampling, which helped to improve the accuracy of the models.

In summary, the study highlights the potential benefits of using deep learning models for the automatic classification of app reviews for requirements engineering and provides valuable insights into the performance of different models in this context. However, further research is needed to explore the generalisability of the results to other app reviews and to investigate the potential for further improving model performance through additional training and optimisation techniques.

6 Conclusion

The primary objective of this study was to investigate the performance of several deep learning models for categorising app reviews for requirement engineering. According to our findings, the DistilBERT model performed worse than the BERT and RoBERTa models in terms of accuracy, precision, recall, and f1-score. Furthermore, our research showed that the models' performance was enhanced by the fine-tuning and undersampling processes.

However, this study is not without limitations. One limitation is the available resources to train the models. It is possible that with more computational power, the models' performance could be further improved. Additionally, the dataset used in this study was limited to reviews of software engineering tools and may not generalize to other domains.

In conclusion, the findings of this study suggest that BERT and RoBERTa models can be effective in classifying software reviews. Future research could explore the performance of these models in other domains and investigate the impact of different pre-training methods on their performance. Ultimately, the results of this study provide valuable insights for software engineers and developers looking to automate the process of classifying app reviews into requirement engineering tasks.

6.1 Achievements

Throughout the development of this project, several achievements were made which have enhanced my knowledge and skills in the field of machine learning. Firstly, I gained a deeper understanding of the intricate parts of designing and developing a machine learning model, including data preparation, model architecture, and hyperparameter tuning. Training the model on a real world scenario has also allowed me to better understand the challenges and limitations of applying machine learning to practical problems.

Furthermore, I have expanded my knowledge in the field of machine learning, which will undoubtedly be beneficial in my future career pursuits. The skills I have acquired throughout this project will enable me to apply machine learning techniques to a wide range of problems and improve my problem solving abilities.

Lastly, I have also developed my skills in academic research and writing. Through the process of writing this report, I have learned how to successfully structure an academic paper, how to effectively communicate complex ideas, and how to incorporate feedback from peers and mentors to improve my work. These skills will be valuable not only in future research projects but also in my professional career.

6.2 Future Work

The results of this study demonstrate the potential of deep learning models, specifically BERT, RoBERTa, and DistilBERT, in the classification of app reviews for requirement engineering. While the focus of this study was on app reviews for requirement engineering, these models can be adapted for use in a wide range of domains, such as e-commerce, healthcare, and finance, where sentiment analysis and classification of text data are important tasks. For example, online retailers can use sentiment analysis to understand customer sentiment about their products, pricing, and customer service. This can help them improve their offerings and better understand customer needs.

One potential avenue for future work is the development of a complete application that utilises these models for real time classification of app reviews. The classification of app reviews can assist in directing different types of reviews to suitable software project members, such as developers, testers, requirements analysts, and user experience experts. This application can be used to automate the

triage process for incoming reviews, leading to more efficient and effective handling of customer feedback.

Another area for future work is the exploration of ensemble learning methods, which involve combining the predictions of multiple models to improve overall performance. Additionally, the development of new deep learning models and techniques can further advance the cutting-edge sentiment analysis and classification of text data.

Overall, this study provides a solid foundation for future research in the application of deep learning models for the classification of app reviews for requirements engineering, as well as other domains.

References

[1] W. Maalej, Z. Kurtanović, H. Nabil, and C. Stanik, “On the automatic classification of app reviews,” *Requirements Engineering*, vol. 21, pp. 311–331, 2016 (cited on pp. 10, 17, 32).

[2] H. Li, L. Zhang, L. Zhang, and J. Shen, “A user satisfaction analysis approach for software evolution,” in *2010 IEEE International Conference on Progress in Informatics and Computing*, IEEE, vol. 2, 2010, pp. 1093–1097 (cited on p. 10).

[3] A. Finkelstein, M. Harman, Y. Jia, W. Martin, F. Sarro, and Y. Zhang, “App store analysis: Mining app stores for relationships between customer, business and technical characteristics,” *RN*, vol. 14, no. 10, pp. 1–24, 2014 (cited on p. 10).

[4] D. Pagano and W. Maalej, “User feedback in the appstore: An empirical study,” in *2013 21st IEEE international requirements engineering conference (RE)*, IEEE, 2013, pp. 125–134 (cited on p. 10).

[5] E. Guzman and W. Maalej, “How do users like this feature? a fine grained sentiment analysis of app reviews,” in *2014 IEEE 22nd international requirements engineering conference (RE)*, Ieee, 2014, pp. 153–162 (cited on pp. 10, 18).

[6] C. Iacob and R. Harrison, “Retrieving and analyzing mobile apps feature requests from online reviews,” in *2013 10th working conference on mining software repositories (MSR)*, IEEE, 2013, pp. 41–44 (cited on p. 10).

[7] L. V. G. Carreño and K. Winbladh, “Analysis of user comments: An approach for software requirements evolution,” in *2013 35th international conference on software engineering (ICSE)*, IEEE, 2013, pp. 582–591 (cited on p. 10).

[8] B. Mahesh, “Machine learning algorithms-a review,” *International Journal of Science and Research (IJSR).[Internet]*, vol. 9, pp. 381–386, 2020 (cited on p. 11).

[9] I. El Naqa and M. J. Murphy, *What is machine learning?* Springer, 2015 (cited on p. 11).

[10] G. C. Cawley and N. L. Talbot, “On over-fitting in model selection and subsequent selection bias in performance evaluation,” *The Journal of Machine Learning Research*, vol. 11, pp. 2079–2107, 2010 (cited on p. 11).

[11] P. Cunningham, M. Cord, and S. J. Delany, “Supervised learning,” *Machine learning techniques for multimedia: case studies on organization and retrieval*, pp. 21–49, 2008 (cited on p. 11).

[12] A. Singh, N. Thakur, and A. Sharma, “A review of supervised machine learning algorithms,” in *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, Ieee, 2016, pp. 1310–1315 (cited on p. 12).

[13] C. M. Bishop, “Neural networks and their applications,” *Review of scientific instruments*, vol. 65, no. 6, pp. 1803–1832, 1994 (cited on p. 12).

[14] K. O’Shea and R. Nash, “An introduction to convolutional neural networks,” *arXiv preprint arXiv:1511.08458*, 2015 (cited on p. 12).

[15] S. Hijazi, R. Kumar, C. Rowen, *et al.*, “Using convolutional neural networks for image recognition,” *Cadence Design Systems Inc.: San Jose, CA, USA*, vol. 9, p. 1, 2015 (cited on p. 12).

[16] L. Deng, G. Hinton, and B. Kingsbury, “New types of deep neural network learning for speech recognition and related applications: An overview,” in *2013 IEEE international conference on acoustics, speech and signal processing*, IEEE, 2013, pp. 8599–8603 (cited on p. 12).

[17] M. M. Lopez and J. Kalita, “Deep learning applied to nlp,” *arXiv preprint arXiv:1703.03091*, 2017 (cited on p. 12).

[18] D. Anguita, L. Ghelardoni, A. Ghio, L. Oneto, and S. Ridella, “The’k’in k-fold cross validation.,” in *ESANN*, 2012, pp. 441–446 (cited on p. 13).

[19] T.-T. Wong and P.-Y. Yeh, “Reliable accuracy estimates from k-fold cross validation,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 32, no. 8, pp. 1586–1594, 2019 (cited on p. 13).

[20] H. B. Moss, D. S. Leslie, and P. Rayson, “Using jk fold cross validation to reduce variance when tuning nlp models,” *arXiv preprint arXiv:1806.07139*, 2018 (cited on p. 13).

[21] T. Yang and Y. Ying, “Auc maximization in the era of big data and ai: A survey,” *ACM Computing Surveys*, vol. 55, no. 8, pp. 1–37, 2022 (cited on p. 14).

[22] X. Lin, “Sentiment analysis of e-commerce customer reviews based on natural language processing,” in *Proceedings of the 2020 2nd International Conference on Big Data and Artificial Intelligence*, 2020, pp. 32–36 (cited on p. 14).

[23] K. L. Goh and A. K. Singh, “Comprehensive literature review on machine learning structures for web spam classification,” *Procedia Computer Science*, vol. 70, pp. 434–441, 2015 (cited on p. 14).

[24] T. A. Koleck, C. Dreisbach, P. E. Bourne, and S. Bakken, “Natural language processing of symptoms documented in free-text narratives of electronic health records: A systematic review,” *Journal of the American Medical Informatics Association*, vol. 26, no. 4, pp. 364–379, 2019 (cited on p. 14).

[25] B. Cui, Y. Li, M. Chen, and Z. Zhang, “Fine-tune bert with sparse self-attention mechanism,” in *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on*

natural language processing (EMNLP-IJCNLP), 2019, pp. 3548–3553 (cited on p. 14).

[26] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017 (cited on pp. 14, 15).

[27] T. Wolf, L. Debut, V. Sanh, *et al.*, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, 2020, pp. 38–45 (cited on p. 15).

[28] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018 (cited on p. 15).

[29] M. Koroteev, "Bert: A review of applications in natural language processing and understanding," *arXiv preprint arXiv:2103.11943*, 2021 (cited on p. 15).

[30] F. A. Acheampong, H. Nunoo-Mensah, and W. Chen, "Transformer models for text-based emotion detection: A review of bert-based approaches," *Artificial Intelligence Review*, pp. 1–41, 2021 (cited on p. 15).

[31] Y. Liu, M. Ott, N. Goyal, *et al.*, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019 (cited on p. 16).

[32] M. D. Deepa *et al.*, "Bidirectional encoder representations from transformers (bert) language model for sentiment analysis task," *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, vol. 12, no. 7, pp. 1708–1721, 2021 (cited on p. 16).

[33] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: Smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019 (cited on p. 16).

[34] S. Casola, I. Lauriola, and A. Lavelli, “Pre-trained transformers: An empirical comparison,” *Machine Learning with Applications*, vol. 9, p. 100 334, 2022 (cited on p. 17).

[35] S. Panichella, 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,” in *2015 IEEE international conference on software maintenance and evolution (ICSME)*, IEEE, 2015, pp. 281–290 (cited on pp. 18, 32).

[36] N. Aslam, W. Y. Ramay, K. Xia, and N. Sarwar, “Convolutional neural network based classification of app reviews,” *IEEE Access*, vol. 8, pp. 185 619–185 628, 2020 (cited on pp. 18, 32).

[37] T. Johann, C. Stanik, W. Maalej, *et al.*, “Safe: A simple approach for feature extraction from app descriptions and app reviews,” in *2017 IEEE 25th international requirements engineering conference (RE)*, IEEE, 2017, pp. 21–30 (cited on pp. 19, 32).

[38] E. Guzman, M. El-Haliby, and B. Bruegge, “Ensemble methods for app review classification: An approach for software evolution (n),” in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, IEEE, 2015, pp. 771–776 (cited on p. 19).

[39] A. Hawari, *A dataset of mobile application reviews for classifying reviews into software engineering’s maintenance tasks using data mining techniques*, Mendeley Data, version V1, 2019. DOI: 10.17632/5fk732vkwr.1 (cited on p. 20).

[40] T. pandas development team, *Pandas-dev/pandas: Pandas*, version latest, Feb. 2020. DOI: 10.5281/zenodo.3509134. [Online]. Available: https://doi.org/10.5281/zenodo.3509134 (cited on p. 23).

[41] C. R. Harris, K. J. Millman, S. J. van der Walt, *et al.*, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. DOI: 10.1038/s41586-020-2649-2. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2 (cited on p. 23).

[42] A. Paszke, S. Gross, F. Massa, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf (cited on p. 23).

[43] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018. arXiv: 1810.04805. [Online]. Available: http://arxiv.org/abs/1810.04805 (cited on p. 23).

[44] Y. Liu, M. Ott, N. Goyal, *et al.*, "Roberta: A robustly optimized BERT pretraining approach," *CoRR*, vol. abs/1907.11692, 2019. arXiv: 1907.11692. [Online]. Available: http://arxiv.org/abs/1907.11692 (cited on p. 23).

[45] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: Smaller, faster, cheaper and lighter," *ArXiv*, vol. abs/1910.01108, 2019 (cited on p. 23).

[46] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011 (cited on p. 23).

[47] J. D. Hunter, “Matplotlib: A 2d graphics environment,” *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007. DOI: 10.1109/MCSE.2007.55 (cited on p. 23).