Department of Computer Science

CLASSIFICATION OF APP REVIEWS FOR REQUIREMENTS ENGINEERING USING SUPERVISED MACHINE LEARNING

A DISSERTATION SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF MASTER OF SCIENCE
IN THE FACULTY OF SCIENCE AND ENGINEERING

Student: Bushui Zhang

UID: 10407579

Supervisor: Dr. Liping Zhao

Degree: MSc ACS: Artificial Intelligence

2022

Contents

Word Count: 15051

List of Figures

List of Tables

List of Equations

Acronyms

Adam Adaptive moment estimation. 35, 36

ANN Artificial Neural network. 32, 38, 86

BERT Bidirectional Encoder Representations from Transformers. 5–7, 15, 44, 46, 51, 59, 73, 74, 77, 79, 83–87

Bi-LSTM Bidirectional Long Short-Term Memory Network. 5–7, 15, 40–42, 59, 69–72, 77–79, 82–84, 86, 87

BOW Bag of Words. 5, 23, 24, 59, 66–68, 74, 81, 82, 84

CNN Convolutional Neural Network. 51, 88

FN False Negative. 48

FP False Positive. 48

IDF Inverse Document Frequency. 24, 25

LSTM Long Short-Term Memory Network. 5, 39–41

NB Naive Bayes. 5–7, 29, 59, 67–69, 74, 76, 77, 81, 82, 84, 87

NLP Natural Language processing. 14, 16–18, 52, 53, 62, 80, 86

NLTK Natural Language Toolkit. 62

ReLU rectified linear activation unit. 36

RNN Recurrent Neural Network. 5, 38, 39, 41, 43, 44, 86

SVM Support Vector Machine. 5–7, 29–31, 59, 68, 69, 76, 77, 82, 84, 87

TF Term Frequency. 24, 25

TF-IDF Term Frequency-Inverse Document Frequency. 5, 6, 24, 25, 59, 66–69, 74, 76, 77, 81, 82, 84

TN True Negative. 48

TP True Positive. 48

VADER Valence Aware Dictionary and sentiment Reasoner. 62

Abstract

In this digital age, the software is gradually migrating from PCs to mobile phones. Nearly every aspect of our everyday lives can be accessed through mobile applications made available by app stores like Google Play and the Apple AppStore. At the same time, feedback such as app ratings and reviews provided by the customers on these app stores can indicate the strengths and weaknesses of the apps, enabling developers and vendors to improve them. Recent studies have shown that user review is one of the most crucial aspects in deciding whether or not an app will succeed in the future. As user reviews may include information about bugs, ideas for new features, ratings on the apps, and experiences of using these apps, as illustrated by some previous studies, manual categorization of these reviews could be relatively laborious and time-consuming.

This project, which was mainly based on NTLK, Scikit-Learn, Keras, and Python programming, focused on the application of various supervised machine learning techniques to the classification of app reviews. These methods include conventional machine learning approaches, such as Support Vector Machine and Naive Bayes, and cutting-edge deep learning approaches (Bidirectional Encoder Representations from Transformers, etc.). Due to the lack of sufficient data with associated labels, a dataset with a total of 12000 reviews with corresponding labels was created, with 3000 entries in each category. The pipeline for natural language processing was strictly followed throughout the entire project. A series of experiments were conducted to compare these techniques' accuracy, recall, precision, and F1 scores. In addition, rule-based sentiment analysis was leveraged to generate an extra feature, the compound score, thus improving the performance of different models. Regarding the evaluation of results, the BERT model performed the best. It achieved the highest F1 score and accuracy, which were 0.826 and 0.825, respectively.

Declaration

No portion of the work referred to in this 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.

Copyright

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, trade marks 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 Thesis restriction declarations deposited in the University Library, The University Library's regulations (see `http://www.library.manchester.ac.uk/about/regulations/`) as well as in The University's policy on presentation of Theses

Acknowledgement

I would like to express my gratitude to my family. It is their encouragement that enables me to successfully accomplish this challenging project.

I also want to thank Dr Liping Zhao for her patience and support. She gave me a lot of inspiring suggestions.

Chapter 1

Introduction

1.1 Project Description and Motivation

Software is gradually migrating from computers to mobile phones in this digital era. Mobile apps are provided for nearly every field of our daily lives by app stores such as the Google Play Store and Apple AppStore. According to the statistics made by Statista [1], with 3.48 million and 2.22 million apps, respectively, Google's Play Store and Apple's app store were the top two largest app stores in the first quarter of 2021. Such app stores allow users to explore, download and write reviews for different apps. At the same time, developers can collect users' reviews and improve their apps. It is demonstrated by Mobile App Daily [2] that user review is one of the most important factors in determining whether or not an app will succeed in the market. ReviewTrackers, in a report [3] published in 2022, points out that a negative review has discouraged 94% of users from starting to use that app. With better reviews, the ranking of apps may increase, as illustrated by Finkelstein et al. [4]. Hence, analyzing users' reviews and gaining insight into their demands and experiences is significant for improving an app's quality and preventing negative reviews. However, a fairly large number of the reviews are uninformative, including meaningless information, spam, or simply stating star ratings in words [5]. Besides, thousands of reviews are submitted each day[6]. These issues make manual classifications critical and time-consuming for analysts and developers. Therefore, it is necessary to automatically classify app reviews into several categories in order to save these people from this laborious process. Classifying app reviews into several classes can be regarded as a text classification problem in the natural language processing (NLP) domain since each app review

is composed of one or several sentences.

Machine learning, a crucial technology in text classification, can be divided into supervised and unsupervised learning to analyze labeled and unlabelled data accordingly. Supervised machine learning needs to be applied with a labeled apps' review dataset, which is required to teach models to learn which category each review is from. More details will be covered in Section 2.7. Over the past few decades, much research has adopted different supervised machine learning methodologies for app reviews' classification tasks. However, the majority of them concentrated on the application of traditional machine learning; few took into accounts more advanced ones such as Bidirectional Long Short-Term Memory Network (Bi-LSTM) and Bidirectional Encoder Representations from Transformers (BERT). Details of that will be covered in Section 2.9. As a result, automatic classification of app reviews utilizing different supervised machine learning algorithms, from traditional machine learning to deep learning, is proposed. Their performance will be examined and compared. As suggested by the paper [7], four basic types (bug reports, feature requests, user experience, and ratings) for categorizing reviews will be used.

1.2 Project Aim and Objectives

The project aim can be summarised as the development of automatic classification classifiers using supervised machine learning algorithms ranging from traditional machine learning to deep learning to categorize app reviews into four main categories: bug reports, feature requests, user experience, and ratings:

1. `Bug/Problem reports (PD)`:with the application that needs to be fixed, such as crashes, erroneous behavior, or performance issues.

2. `Feature requests (FR)`: more functionalities or content that users ask for or suggestions on making the app better by adding or changing features.

3. `User experience (UE)`: users' experiences with the app and its features in specific scenarios.

4. `Ratings (RT)`: Text which reflects users' sentiments towards an app, such as praise and dispraise.

As the project allows for a variety of approaches to be pursued, a collection of objectives should be considered to achieve the goal as mentioned above. As a result, the following objectives have acted as a guide for the project's development:

- Gain a thorough understanding of some NLP techniques and basic supervised machine learning algorithms such as support vector machine and Nave Bayes.
- Obtain some knowledge of advanced deep learning networks such as recurrent neural networks and bidirectional encoder representations from transformers.
- Research different evaluation metrics and find appropriate metrics to evaluate classifiers from various aspects.
- Investigate the related literature, and discuss their main weaknesses.
- Design the research methodology and experiments.
- Collect app reviews from Google Play and the Apple AppStore and categorize them into the four categories listed above.
- Utilize some text preprocessing techniques to tidy up the collected text and vectorize sentences.
- Choose suitable machine learning algorithms and implement these models.
- Make experiments, evaluations, and comparisons among these models.

1.3 Dissertation Structure

The remainder of this dissertation is structured as follows.

- Chapter 2 demonstrates this project's background and literature reviews to provide a thorough grasp of the techniques utilized. The natural language processing pipeline, some methods related to text preprocessing and feature engineering in the field of NLP, various machine learning algorithms, as well as relevant evaluation metrics, are clearly presented. The related research and critical analysis are explored.

- Chapter 3 presents research methodology based on the NLP pipeline, along with experimental design, to ensure that experiments will be carried out according to appropriate instructions. Finally, it outlines the experimental implementation details which follow the experimental design.
- Chapter 4 first shows the results of various experiments. Then the effect of the additional feature is illustrated. The findings of these experiments are finally clarified.
- Chapter 5 summarises what has been achieved and the limitations of the experiments. Based on that, possible future investigations are also highlighted.

Chapter 2

Background and Related Work

Because the purpose of this project is to develop classifiers to categorize app reviews into four classes, which is in the area of natural language processing, this chapter will investigate the natural language processing pipeline and related techniques that can be utilized in each step of the NLP pipeline. In addition, many significant and representative research publications will be discussed, and their defects will be analyzed.

2.1 NLP and Text Classification

Ben illustrates that NLP is the ability of a computer program to interpret natural language which refers to both spoken and written human language [8]. It is a significant area in artificial intelligence. NLP combines computational linguistics with statistical machine learning and deep learning models [9]. Text classification, a key application area of NLP, is the processing of tagging unstructured texts with their relevant labels from a set of predetermined categories [10].

2.2 NLP Pipeline

An NLP pipeline is a set of steps used to produce end-to-end NLP software by using machine learning or deep learning in a more structural manner [11]. There are 8 main steps in the NLP pipeline as shown in Figure 2.1

1. Data Collection

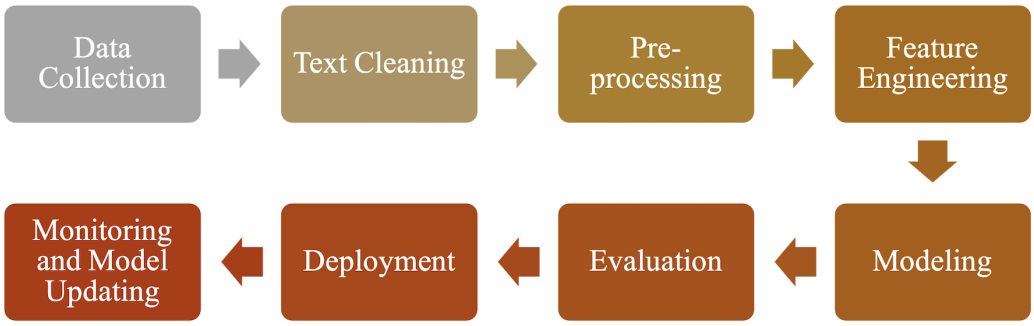

Figure 2.1: NLP Pipeline[12]

Text dataset collection is simply a method of obtaining enough text or text-like data from multiple resources for training and testing purposes [13].

2. Text Cleaning

   Texting cleaning means removing unwanted text from the data [14]. For instance, when data is obtained from a website, a variety of HTML tags need to be removed from the original text to get meaningful textual data.

3. Pre-processing

   Text processing is a technique to tidy up data so that it can be fed into models effectively [15]. Text data comprises noise in numerous forms, such as punctuation, stop words, and text written in numerical or other special character forms, since they do not add to the overall meaning of text [16].

4. Feature Engineering

   Feature engineering is one of the most important steps in NLP [17]. Since machines can only understand numbers, feature engineering is demanded to transform the raw text into numerical form.

5. Modelling

   In the modeling step, models are made based on data. Depending on volumes of data, different models could be used to get better results [18]. If only a small amount is collected, a heuristic approach (but will not cover in this project) should be used. Traditional machine learning algorithms can be utilized if a good amount of data is obtained. Deep learning models can be built if a large amount of data is available.

6. Evaluation

   Evaluation is the process of analyzing machine learning models' performance as well as their strengths and weaknesses, using different evaluation metrics [19]. Besides, to evaluate models efficiently, suitable resampling techniques, which draw samples from train data and fit a model to check the variability of the model, need to be chosen [20].

7. Deployment, Monitoring and Model Updating

   After selecting an appropriate model, it is deployed in a production environment. Then, system performance is regularly monitored. Furthermore, the model is updated with new data available [21]. Because the project's models were never used in real life, these procedures will not be illustrated further.

2.3 Data Collection

From section 2.2, the beginning phase of the NLP pipeline, data collection, is introduced. In that phase, practitioners are faced with the challenge that data used to train the model is not enough [22]. Besides, a severe problem called imbalanced data is also discussed here. Data augmentation can be exploited to grow the dataset to mitigate these two problems.

2.3.1 Imbalanced Data

Imbalanced data occurs during training when there is an unequal distribution of the training data between its classes. The class distribution of a dataset could reveal anything from a slight bias to a severe imbalance. For a small bias, this problem could be ignored. However, when the problem of imbalance becomes more serious, specialized techniques are required to avoid inaccurate predictions of minority classes [23]. Resampling is a widely used approach to deal with highly imbalanced datasets [24]. It entails removing samples from the majority classes (under-sampling) or adding more samples to the minority classes (over-sampling). A balanced dataset can be obtained after these approaches.

2.3.2 Data Augmentation

The data augmentation technique allows us to artificially enhance training data size by producing multiple versions of genuine datasets without actually acquiring more data [25]. The

model performs better when more data is available. However, the distribution of augmented data should neither be too similar nor too different from the original [26]. Otherwise, it may result in overfitting which leads to poor performance. Therefore, effective data augmentation should strive for balance [27]. There are three different ways to augment text data [28].

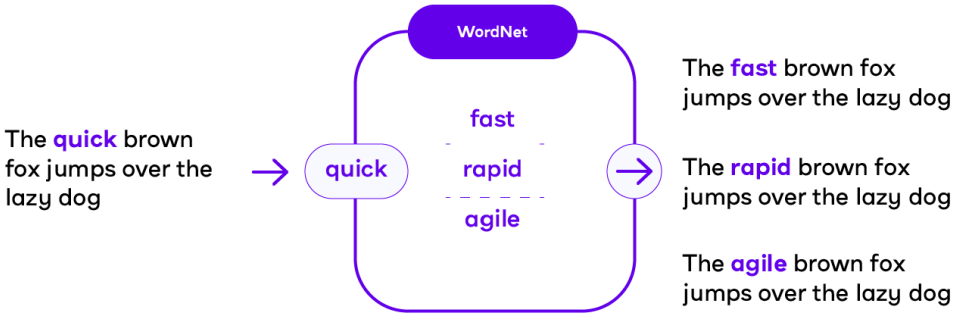

Figure 2.2: Data Augmentation Using WordNet[29]

1. Thesaurus-based substitution: Substitute synonyms for a few words. WordNet is one such example, a manually crafted database that connects words with their synonyms. That is depicted in Figure 2.2.

2. Word-Embeddings Substitution: In this technique, we use pre-trained word embeddings such as Word2Vec and GloVe. Then, some words in the sentence are substituted with the nearest neighboring words in the embedding space. For example, the embedding space of "awesome" is shown in Figure 2.3. The original word "awesome" can be replaced with the three most similar words described in Figure 2.4.

3. Back Translation: Translate sentences from the original language into other languages and then back to the source language. An example is shown in Figure 2.5.

2.4 Text Preprocessing

Text preprocessing is the next step of data cleaning, as mentioned in section 2.2. It converts the text into a more digestible format, allowing the machine learning algorithm to perform better [32]. Long text input is typically split down into smaller chunks so that system can process the input in subsequent stages [33]. Text preprocessing may contain the subprocesses listed below.

Figure 2.3: Embedding Space for the Word Awesome [30]

Figure 2.4: Three Variations of the Original Text Using Word-Embeddings Substitution

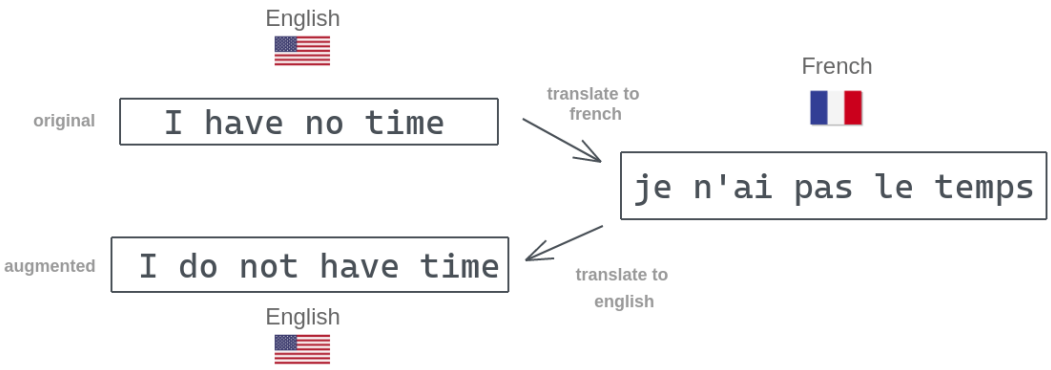

Figure 2.5: Data Augmentation Using Back Translation[31]

1. Tokenization: sentences are splitting into words. Punctuation marks are always discarded during this step. [34]

2. Lowercaseing: each word converted into lower case (Books -> books). Although words like "books" and "books" mean the same, they are represented as two different words in the vector space model if these words are not converted into lowercase. [35]

3. Stop words removal: Stop words (a, an, etc.) are frequently used tokens in sentences. As they do not assist in distinguishing two documents, these terms can always be removed with ease. [36]

4. Lemmatization: it is the process of removing a component of the word or reducing the word into its root. [37]

2.5 Feature Engineering

As discussed in the section 2.2, the process of extracting meaningful information to make it useable for machine learning models is known as feature engineering [38]. In the field of NLP, there are three major categories of feature extraction methods: basic, statistical, and vectorized. One example will be introduced in the following sections for each of them.

2.5.1 Bag of Words

Bag of Words (BOW) is a commonly used basic feature extraction method [39]. It is a text representation that specifies the occurrence of words in a document [40], involving two key steps:

1. Create the vocabulary of known words.

   For instance, two documents are obtained after the text preprocessing step:

   - Document 1: learn svm algorithm
   - Document 2: learn english article

     We can end up with making the vocabulary containing all the words in both documents, shown in Figure 2.6.

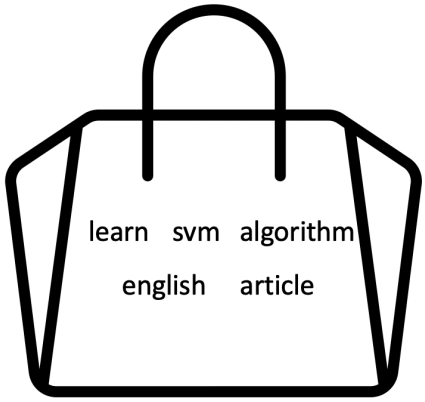

Figure 2.6: A Vocabulary Example

2. Measure the presence of existing words.

   Then, as shown in Figure 2.7, records of the occurrence of the vocabulary words in each specific sentence are kept.

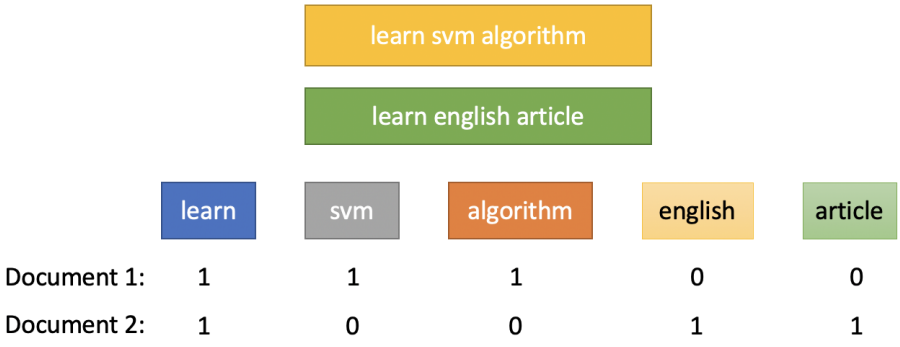

Figure 2.7: Word occurrence records in example documents

2.5.2 Term Frequency-Inverse Document Frequency

Term Frequency-Inverse Document Frequency (TF-IDF) extracts features from original documents using concepts from statistics and probability [41]. Unlike calculating the count of each word in the vocabulary for every document in BOW, TF-IDF uses two components: Term-Frequency (TF) and Inverse Document Frequency (IDF) to measure the numerical representation of each document. The TF of a word can be calculated by using Equation 2.1 where $tf(t,d)$ means the Term-Frequency of term t appearing in document d, and $freq(t,d)$ is the

occurrence of term t in document d.

$$tf(t,d) = log(1 + freq(t,d)) \tag{2.1}$$

The IDF of a set of documents measures how common or rare a word is in the entire document set. The closer to zero, the more common a word is. It is calculated by utilizing Equation 2.2 where N is the total number of documents and $count(d \in D : t \in d)$ is the number of documents that contain term t

$$idf(t,D) = log(\frac{N}{count(d \in D : t \in d)}) \tag{2.2}$$

Finally, the TF-IDF score for a word can be calculated by multiplying TF and IDF together, as shown in equation 2.3.

$$tfidf(t,D) = tf(t,d) \times idf(t,D) \tag{2.3}$$

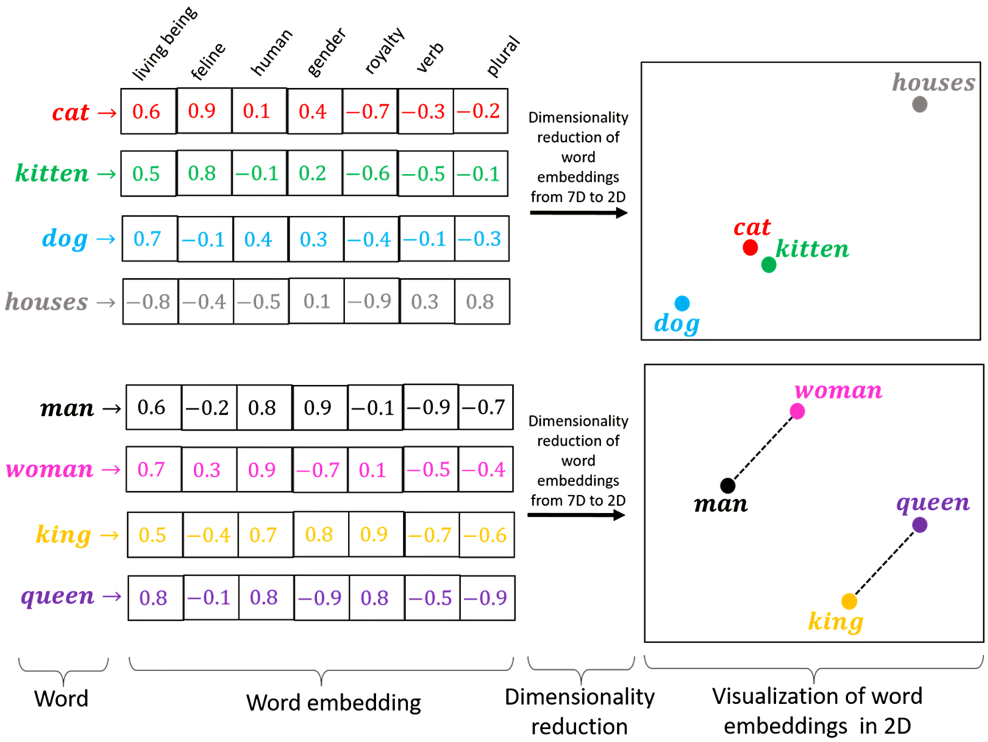

Figure 2.8: An example to show that word embedding can store word semantics [42]

2.5.3 Word Embeddings

This technique aims to map a word to a fixed-length vector, which can store the semantics of that word[43]. To generate word embeddings, neural networks can be leveraged with any type of text input [44]. Figure 2.8 shows how the word embedding can store semantics. The words (cat, kitten, etc.) with similar meanings appear close to each other when they are mapped into vector space.

2.6 Feature Selection

When creating supervised machine learning models, feature selection can be employed to reduce the number of input variables [45]. It is preferable to limit the number of input variables to decrease the computation cost of modeling. In some situations, it can also improve models' performance. Two main types of approaches can be leveraged in feature selection: filter methods and wrapper methods [46].

2.6.1 Filter Methods

Filter methods are generally utilized before building machine learning models, introduced in Section 2.2. These techniques' processes are independent of any machine learning algorithm [47]. Instead, features are selected based on their performance in different statistical tests, which calculate their correlation with the outcome variable [48]. To define which correlation coefficient is applied in distinct conditions, the following table 2.1 is presented.

Feature\Response	Continuous	Categorical
Continuous	Pearson's Correlation	LDA
Categorical	Anova	Chi-Square

Table 2.1: Correlation Coefficient in Various Situations

As the numerical representation of text (feature) in BOW and labels (response) are categorical, Chi-Square will be used in this project. Hence, it will be discussed in the following section.

2.6.1.1 Chi-Square in Filter Method

Chi-Square calculates the deviation between the expected count E and observed count O by using Equation 2.4, where O_i is the observed count for samples i, E_i is the expected count for samples i, and X_c is Chi-Square value of a feature [49]. In feature selection, features that are highly related to the response are required to be selected. When a feature and the response are dependent, the difference between the observed and expected counts is large. Hence, the higher the Chi-Square value, the more closely the feature relates to the response, which should be selected for model training.

$$X_c^2 = \sum \frac{(O_i - E_i)^2}{E_i} \tag{2.4}$$

2.6.2 Wrapper Methods

Wrapper methods attempt to employ a subset of features to train a model. On the basis of the inferences drawn from the previous model, features are added or removed from the subset [50]. This process is depicted in Figure 2.9.

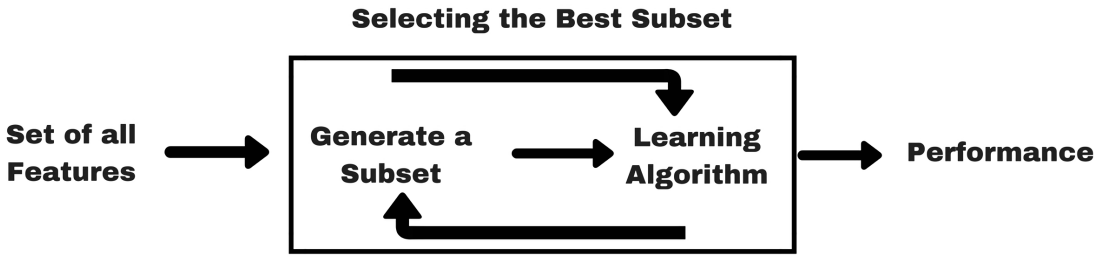

Figure 2.9: Process of Wrapper Methods

2.6.2.1 Forward Feature Selection

Forward feature selection is a widely used wrapper method [51]. It is an iterative strategy in which the model is trained with no features at first. A new feature that can most effectively improve the model is added until the model's performance can no longer be improved by adding a new feature. Figure 2.10 depicts the processing of the forward feature selection for five features. Each of the five features is utilized for training the model in the first iteration. If

Feature 3 produces the best results, it is kept in the next iteration, and the other four features are added. This procedure is repeated iteratively. Assume that the addition of Feature 1 or Feature 2 does not improve the model's performance in the fourth iteration, and the process halts. As a result, Feature 2, Feature 3, and Feature 4 are chosen for training.

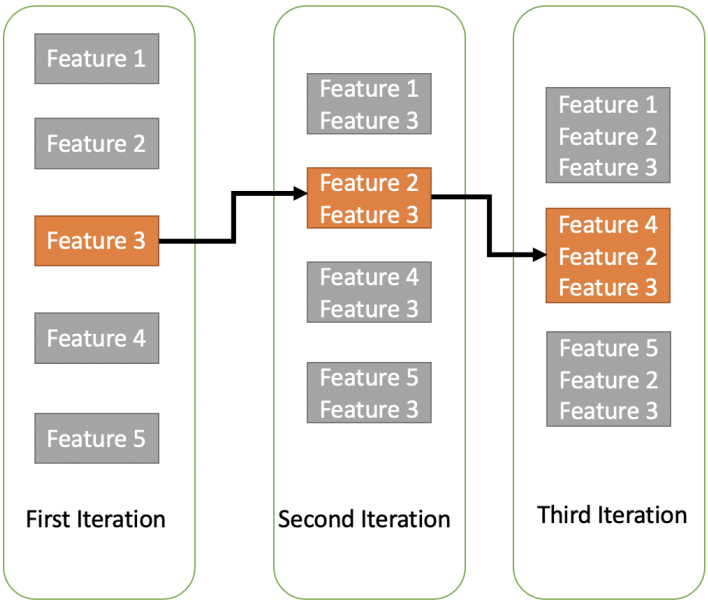

Figure 2.10: A Forward Feature Selection Example

2.7 Machine Learning Methods

In the section 2.2, modeling is the process of making models based on data. That always refers to the training process in machine learning [52]. The training process, the most essential step in machine learning, is the time that prepared data is fed into the selected machine learning model to find patterns and make predictions [53]. The model learns from the data and is able to complete the predefined goal via this process. There are two different techniques for model training, including supervised machine learning and unsupervised machine learning [54]. Since supervised machine learning is relevant to the project, it will be illustrated in the following section.

2.7.1 Supervised Machine Learning

Supervised machine learning utilizes data that is well labeled to train the machine [55]. By doing this, a mathematical representation of the relationship between the source features and target labels is created [56]. When new data comes in, its corresponding labels can be predicted based on the mathematical representation. There are two main types of supervised machine learning: regression and classification, which can be differentiated by the target variable type [57]. In classification, the target variable is categorical, while the target is continuous in cases of regression [58]. As source features (obtained from app reviews) and targets (4 categories) are identified in this project, it needs to employ supervised machine learning techniques. More specifically, the project is a task of classification, as targets are categorical.

2.7.2 Traditional Machine Learning

The majority of the applied features in traditional machine learning techniques must be defined by a domain expert in order to reduce the complexity of the data and make patterns more obvious for the learning algorithm to work [59] Problems need to be broken down into different components first, and results are combined at the final step. A traditional machine learning algorithm can be as simple as linear regression [60]. A big advantage of these techniques is that they take a short time to run and can give acceptable outcomes [61]. As illustrated by Destin [62], Naïve Bayes (NB) and Support Vector Machine (SVM) can give fairly good results for classification. Therefore, they will be presented in the following two sections, respectively.

2.7.2.1 Naive Bayes

The Naïve Bayes algorithm is a classification algorithm based on the Bayes Theorem with assumptions of independent predictors (features) [63]. The probability of the occurrence of y, given that X has occurred, can be calculated using the Bayes theorem shown in Equation 2.5. The Nave Bayes algorithm interprets y as a class label and X as a set of features. X is given as $X = (x_1, x_2, x_3, \cdots, x_n)$. Equation 2.6 can be gotten by the property of independent predictors. The denominator in Equation 2.6 is a constant. Therefore, $P(y|x_1, \cdots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i|y)$. Then, class y can be selected with maximum probability, as shown in Equation 2.7. The major advantages of NB are that it can be both highly scalable in terms of the number of predictors

and make real-time predictions [64].

$$p(y|X) = \frac{P(X|y)P(y)}{P(X)} \tag{2.5}$$

$$p(y|x_1,\cdots,x_n) = \frac{P(x_1|y)P(x_2|y)\cdots P(x_n|y)P(y)}{P(x_1)P(x_2)\cdots P(x_n)} \tag{2.6}$$

$$y = argmax_y P(y)\prod_{i=1}^{n} P(x_i|y). \tag{2.7}$$

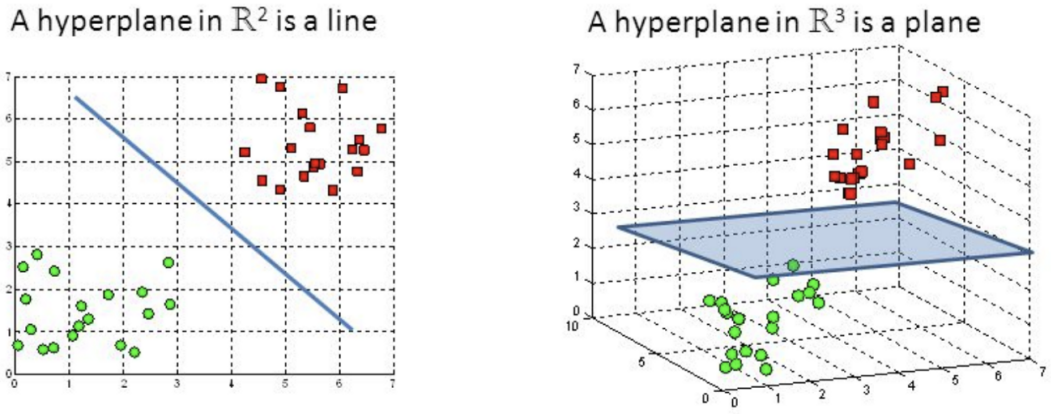

Figure 2.11: Hyperplanes in 2D and 3D Feature Space [65]

2.7.2.2 Support Vector Machine

SVM is a supervised machine learning algorithm that can be employed in both classification, and regression problems [66]. However, it is generally used in classification [67]. Each data item is plotted in n (number of features) dimensional space in the SVM algorithm. Then, an optimal hyperplane is found to separate these data points into different classes [68]. An example of hyperplanes in 2D and 3D feature space is shown in Figure 2.11. This approach is based on the notion that an ideal boundary should be located as far away from each class's closest data point as possible [69]. For non-linear problems, the SVM has a technique called

kernel trick to take low dimensional input space to higher dimensional output space so that a non-linear problem is converted to a separable problem [70]. The primary advantages of SVM are its good generalization (less overfitting) and relatively good scalability on high-dimensional data [71].

2.7.3 Deep Learning

As opposed to the traditional machine learning method discussed in section 2.7.2, deep learning approaches aim to incrementally learn high-level data, which eliminates the demand for domain expertise and hardcore feature extraction [59]. That can be depicted in Figure 2.12. It tends to solve problems end-to-end. Deep learning algorithms can also be thought of as the sophisticated structural, and complicated mathematical evolution of machine learning algorithms [72].

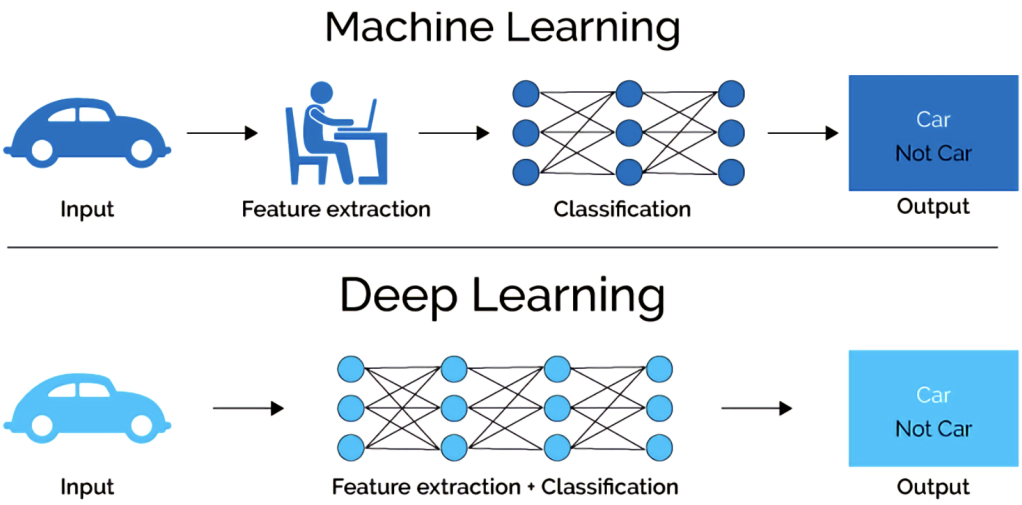

Figure 2.12: (Traditional) Machine Learning Vs Deep Learning [40]

2.7.3.1 Loss Function

The loss function in deep learning quantifies the disparity between the output produced by the algorithms and the expected outcome [72]. The loss function boils down all the positive and negative characteristics of a potentially complicated system to a single scalar value, enabling comparison and ranking of potential solutions [73]. Furthermore, a loss function must be chosen to calculate the model's error during the optimization process, specified in 2.7.3.3 [74].

Cross entropy loss can measure the performance of a classification model, defined as Equation 2.8 ,where N is the total number of classes, $\hat{y}_i$ is the probability of an instance belonging to the $i-th$ class, and y_i is its corresponding target class [75]. This loss can be calculated as the variation between the predicted probability distribution of a machine learning model and the expected distribution [76]. The minus sign makes sure that the loss gets smaller when the difference gets closer [77].

$$Loss = -\sum_{i=1}^{n} y_i \cdot log\hat{y}_i \tag{2.8}$$

- Binary Cross Entropy

  Binary Cross Entropy Loss can be used for binary classification, where the observations are segregated into any of two labels. It formula is explained in Equation 2.9, where $\hat{y}$ is the predicted probability, and y is the indicator (0 or 1) [78].

$$Loss = -(ylog(\hat{y}) + (1-y)log(1-\hat{y})) \tag{2.9}$$

- Categorical Cross Entropy vs Sparse Categorical Cross Entropy

  Categorical cross entropy and Sparse cross entropy can both be presented using Equation 2.8. The only difference is the encoding of truth labels [64]. Sparse categorical cross entropy employs the label encoding method to directly assign a number to each label. In contrast, one hot encoding is utilized for categorical cross entropy to create a one-hot vector form for each category. An example is depicted in Figure 2.13.

2.7.3.2 Neural Network

A neural network, often known as an artificial neural network ANN, is a type of computational learning system that employs a network of functions to convert data input in one form to the intended output, typically in another form [79]. It was inspired by human biology and how the brain's neurons work together to process inputs from the senses [80]. A neural network usually comprises three components: input layers, hidden layers, and output layers [81]. The Figure

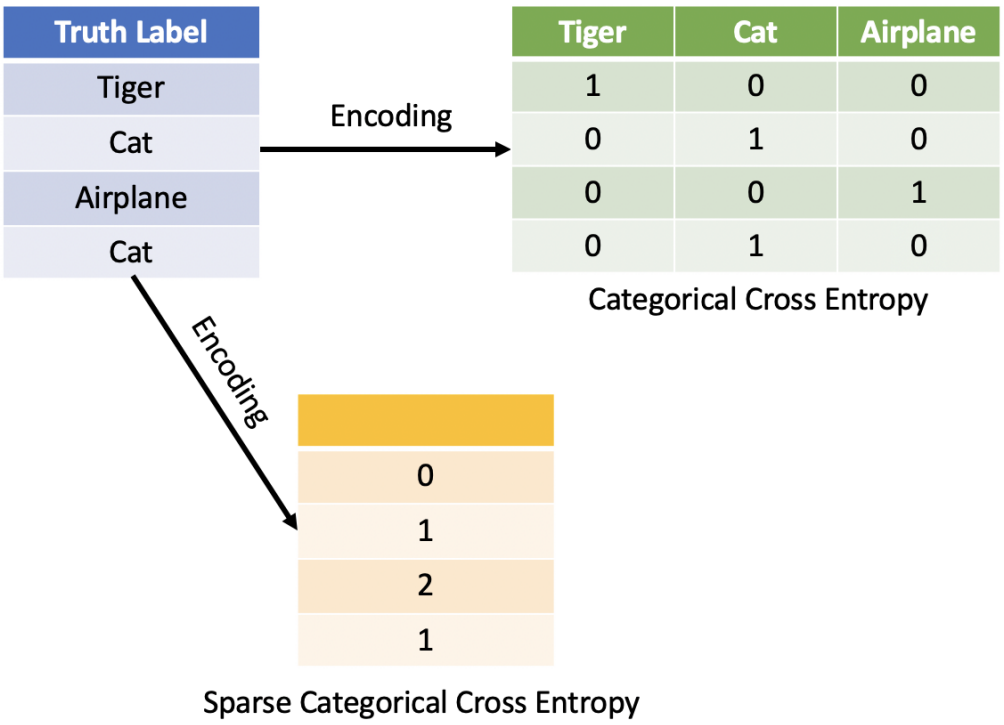

Figure 2.13: Categorical Cross Entropy VS Sparse Categorical Cross Entropy

2.14can describe this architecture. An input layer, known as a bunch of input nodes, is where the input data is given and motivates the model to start learning. After that, the information from the input layer is passed to the hidden layers. A hidden layer consists of a group of neurons where all the computation on the input data is done. A neuron network can have any number of hidden layers. An output layer contains one or more nodes from which the output of a model is derived. Based on different classification problems, the number of output neurons is different.

A perceptron, a simple neural network, has only one layer where all of the mathematical computations are done [83], as shown in Figure 2.15.This model includes five steps: First, data inputs are fed into the perceptron. Then, each input is multiplied by its weight. After that, the results from the last step are summed up. An extra bias is added to the summation. Furthermore, the activation function, which will be discussed in Section 2.7.3.4, is applied. Finally, the output of the perceptron model is triggered to either 0 or 1. However, this model can also learn linear functions, demonstrated by Marvin and Seynour [84]. Multilayer perceptrons, also known as artificial neural networks, involve more than one perceptron being assembled to form a multiple-layer neural network [85]. This type of network has more power to cope with

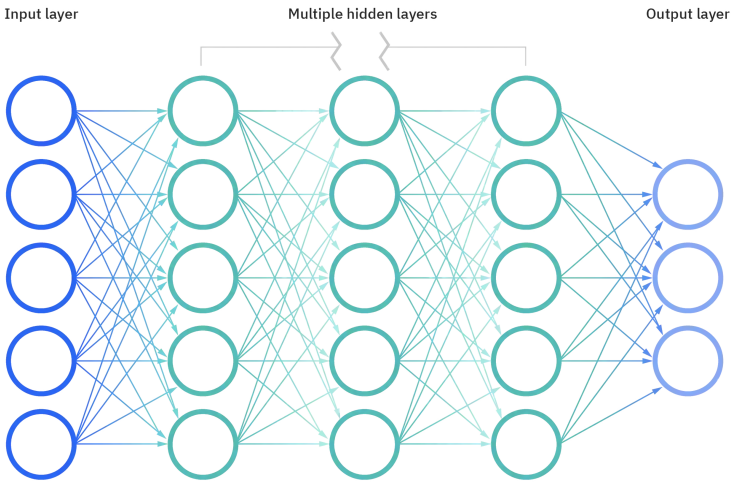

Figure 2.14: An Example of Neural Network[82]

learning non-linear functions [86].

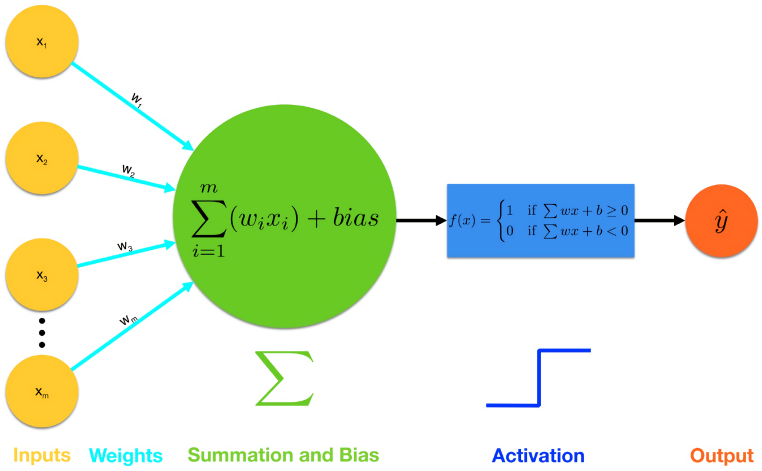

Figure 2.15: Architecture of A Perceptron [87]

2.7.3.3 Optimization Method

As shown in Figure 2.14, $\sum_{i=1}^{n}(w_i x_i) + bias$ is performed in a perceptron model's hidden neuron. For a neural network, there will be more such calculations performed. However, these parameters (weights and bias) that could minimize the model's total loss (cost) cannot be determined manually [88]. Hence, optimization algorithms that can minimize the difference between the predictions and the expected values are desired.

Gradient descent, which is an iterative first-order optimization algorithm, is based on the

idea of updating parameters in the opposite direction of the gradient of the objective function [89]. Equation 2.10 explains this, where θ represents a set of parameters to optimize, $J(\theta)$ represents an objective function, and α is the learning rate. The objective function is the one to optimize, whose value can be either maximized or minimized [90]. However, the method has the problem of a fixed learning rate. A high learning rate might lead to skipping the optimal solution, while a small learning rate could result in too many iterations to reach the best values [91].

$$\theta = \theta - \alpha \dot{\nabla}_\theta J(\theta) \tag{2.10}$$

Adaptive moment estimation (Adam) can adopt an adaptive step size for each parameter using a decaying average of partial derivatives [92]. As suggested by Kingma et al. [93], the essential update rules are defined as:

$$g_t = \nabla_\theta f_t(\theta_{t-1}) \tag{2.11}$$

$$m_t = \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t \tag{2.12}$$

$$v_t = \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2 \tag{2.13}$$

$$\hat{m_t} = \frac{m_t}{1-\beta_1^t} \tag{2.14}$$

$$\hat{v_t} = \frac{v_t}{1-\beta_2^t} \tag{2.15}$$

$$\hat{\theta_t} = \hat{\theta_{t-1}} - \frac{\alpha \cdot \hat{m_t}}{\sqrt{\hat{v_t}} + \varepsilon} \tag{2.16}$$

where θ is a set of parameters that minimizes the cost function, $f_t(\theta)$ implies the stochastic objective function at timestep t, m is the biased first-moment estimate, n indicates the biased second-moment estimate, $\hat{m}$ signifies the bias-corrected first-moment estimate, $\hat{n}$ denotes the bias-corrected second raw moment estimate. Besides, g_t^2 indicates the elementwise square of g_t. β_1^t and β_2^t denote the the powers of t of β_1 and β_2 , respectively. β_1, β_2 and ε are three hyperparameters. According to the Adam paper [93], good default settings are $\varepsilon = 10^{-8}, \beta_1 = 0.9$, and $\beta_2 = 0.999$. Except for β_1 and β_2, all variables with the subscript t represent their values at timestep t. As β_1 and β_2 are set to values nearly equal to 1, $(1-\beta_1)\cdot g_1$ and $(1-\beta_2)\cdot g_1^2$ in Equation 2.12 and Equation 2.13 can be considered as 0 separately. Then, they are simplified as $m_t = \beta_1 \cdot m_{t-1}$ and $v_t = \beta_2 \cdot v_{t-1}$ individually, which have a tendency to be skewed towards 0. Therefore, Equation 2.14 and Equation 2.15 are applied to fix this problem. Finally, parameters can be updated with the usage of Equation 2.16.

2.7.3.4 Activation Function

A neural network's activation function describes how a node or nodes in a layer of the network convert the weighted sum of the input into an output [94]. It can be leveraged both in hidden layers and in output layers. This section explains the rectified linear unit (ReLU) for hidden layers as well as Sigmoid and Softmax for output layers.

ReLU function is currently the most successful and widely-used activation function in deep learning [95]. That is calculated via Equation 2.17. It is widespread because it is both easy to implement and less prone to the vanishing gradient problem. The vanishing problem may occur during the training process in deep learning. In Section 2.7.3.3, partial derivatives are calculated during the optimization process. When there are so many layers in a network, the product result of derivatives reduces. At some point, the loss function's partial derivative may get close to zero. That can prevent the parameters that are required to optimize from updating, which may result in the training process stopping [96].

$$f(x) = max(0, x) \tag{2.17}$$

The sigmoid function in Equation 2.18, also known as the logistic function, accepts any

real value as input and outputs values between 0 and 1. The output value will be closer to 1.0 the larger the input, and the output value will be closer to 0.0 the smaller the input. It is always used in binary classification together with binary cross-entropy loss [97].

$$f(x) = \frac{1}{1+e^{-x}} \tag{2.18}$$

The Softmax function generates a vector of values that add up to 1. That vector can represent the probability of class membership. [98]. Equation 2.19 is used to calculate it. Here, z with a subscript denotes the value from the related neuron of the output layer. The exponential achieves a non-linear property. To do normalization, the value from each neuron in the output layer is exponentiated and then divided by the sum of these exponential values. This process converts each value into a probability.

$$softmax(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{2.19}$$

2.7.3.5 Regularization

Regularization is a method for decreasing the variance in the validation set. In doing so, overfitting, which occurs when a model learns too much detail and noise in a training dataset, is mitigated or even avoided. As a result, the ability of models to generalize to new samples is improved [99]. There are several strategies that exist for regularization in deep learning. This section will mostly explain two of them: dropout and L2 regularization.

L2 regularization can solve the problem with large weights in deep learning. As the network is trained continuously, the weights will become specialized and overfit the training data since the weights will expand in size to address the details of the samples presented [73]. In order to solve this problem, an extra term is added to the loss function so that large weight values are penalized, as shown in Equation 2.20. λ, which refers to the regularization parameter, is a hyperparameter. m indicates the total number of weights, and w denotes the weight vector.

$$Cost = Loss + \frac{\lambda}{2m}\sum ||w||_2^2 \tag{2.20}$$

Dropout is a technique where a subset of neurons in the network is selected to be ignored while training [100]. That can be demonstrated in Figure 2.16. The right part is a standard neural network, and the left is the network's structure after applying dropout. This implies that contributions from the selected neurons to the activation of downstream neurons are subsequently erased, and weights associated with these neurons are not updated during optimization [101]. By doing this, there is a decrease in the sensitivity of the model to the particular neurons' weights. As a result, the network is able to better generalize and is less prone to overfitting the training set of data.

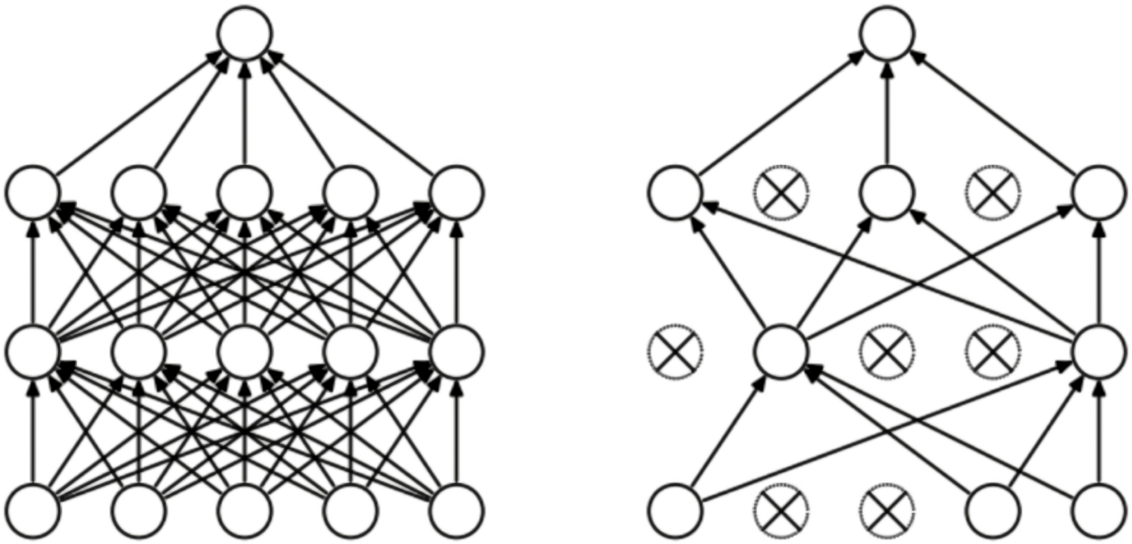

Figure 2.16: Illustration of The Dropout Technique [100]

2.7.3.6 Recurrent Neural Network

Recurrent Neural Network (RNN) is a deep learning architecture that can handle time-series data or data involving sequences [102]. In ANN, all of the input and output are independent of one another, which means that the information from previous steps will not be memorized [103]. In comparison, RNN has internal loops that introduce recursive dynamics to cause delayed activation dependencies across processing elements [104]. That can be shown in Figure 2.17, whereas A, B, and C are the weights to be learned.

A represents weights used for hidden state computation from the input. B denotes weights utilized for output computation. Weights from the previous hidden state are used for the com-

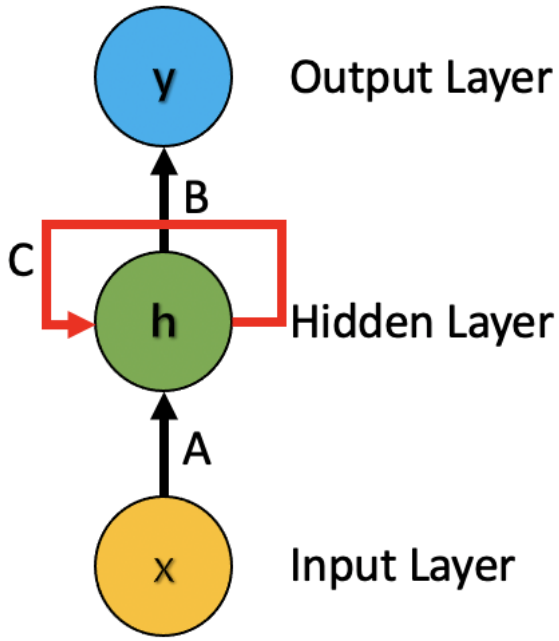

Figure 2.17: RNN Architecture

putation of the current hidden state, represented by C. x, y, and h represent the input, output, and state of the hidden unit separately. In doing so, it not only receives new information but also adds a weighted version of the previous information at each time step [105]. As a result, these neurons can memorize the previous inputs they had. However, this type of network suffers from the problem of vanishing gradient [106]. In other words, as time elapses and much data is fed into the network, it starts to lose the memory of the information that it has previously viewed. It indicates that RNN only has a good short-term memory.

2.7.3.7 Long Short Term Memory

(Unidirectional) Long Short-Term Memory Network (LSTM), a variant of RNN, can solve the problem discussed above. It has internal mechanisms called gates to control the information flow. These gates are capable of learning which data should be kept or forgotten in a sequence. Due to this, the related information is kept from being disturbed by invalid inputs [107].

Three important gates in a LSTM cell are the core of the LSTM. Its structure is shown in Figure 2.18. Especially, C represents the cell state, and h denotes a hidden state. Subscripts are used to indicate the various timesteps that a variable is at. The forget gate determines whether or not a piece of information is kept. Information from the previous hidden state and

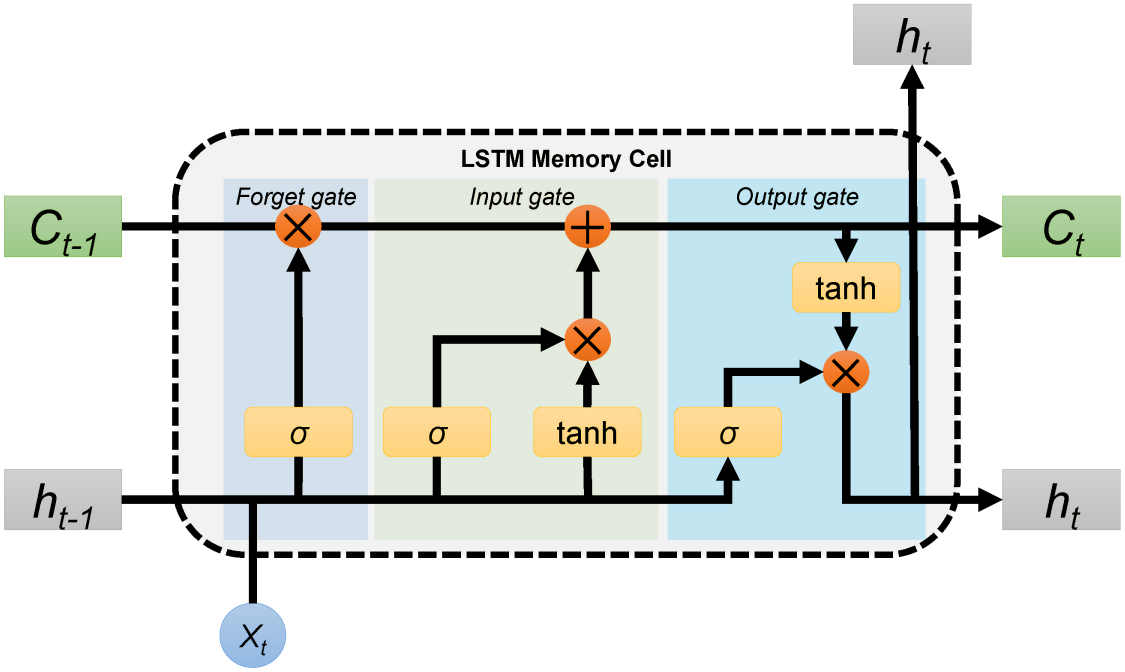

Figure 2.18: LSTM Cell Structure [108]

the current input are combined and passed into the sigmoid function. The closer it is to 0, the more likely it is to forget, and the closer it is to 1, the more likely it is to keep. The input state and output state are used to update the cell state. In the input gate, the importance of the new information supplied by the input is measured by the sigmoid function [109]. The output of the tanh function measures if the information in the tanh function is new or not. The outputs of the sigmoid and tanh functions are multiplied afterward, and the result is utilized to update the cell state. Finally, what information should carry is specified by the output gate. As inputs flow in only one direction, it can only make predictions based on the forward information. Bidirectional Long-Short Term Memory (Bi-LSTM) is an enhanced type of LSTM, which can solve this problem to make the network have both backward and forward information at each timestep [110]. This can make context information better learned [111]. The comparison between the LSTM architecture and the Bi-LSTM architecture is presented in Figure 2.19.

2.7.3.8 Bahdanau Attention Mechanism

Attention is a component of the network's architecture that controls and measures the degree of interdependence [113]. Generally speaking, it is inspired by how humans correlate words in one sentence for natural language processing [114]. In deep learning, attention can be in-

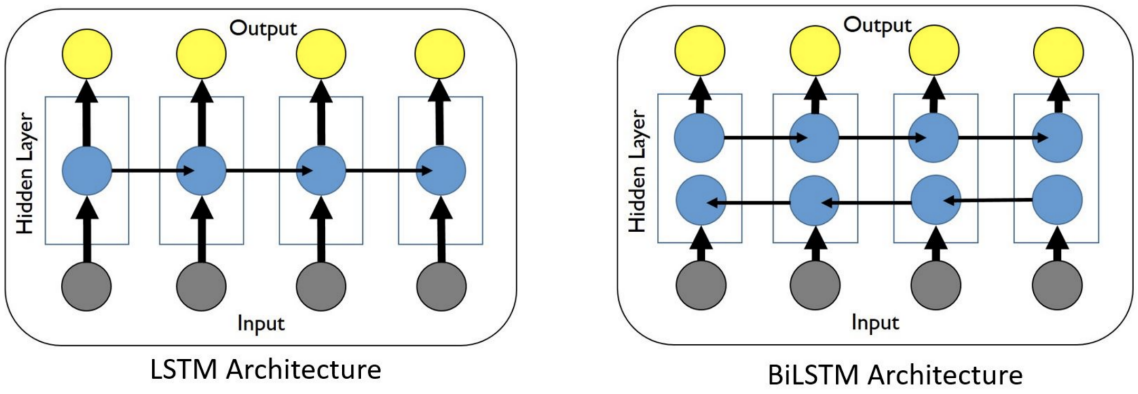

Figure 2.19: LSTM Architecture VS Bi-LSTM Architecture [112]

terpreted as a vector of important weights [115]. Using the attention vector, we can determine how strongly one element, such as a pixel in an image or a word in a sentence, "attends" to other elements. Attention mechanisms, in particular, focus on the most relevant things and discard things that are not so relevant [113]. It was first proposed by Bahdanau et al. as the additive attention in the encoder-decoder model in 2014 for machine translation [116]. It consists of two main components: an encoder and a decoder. The encoder's task is to produce an annotation, $\boldsymbol{h}_i$ for each input word x_i. The decoder is leveraged to generate the target words by concentrating the source sentence's most crucial information with the use of an attention mechanism. The model's architecture is illustrated in Figure 2.20.

First of all, the input and output of the model are defined. Input sequence $\boldsymbol{x}$ of length n is defined as $\boldsymbol{x} = [x_1, x_2, \ldots, x_n]$, and output sequence $\boldsymbol{y}$ of length m is shown as $\boldsymbol{y} = [y_1, y_2, \ldots, y_m]$. Then, the encoder is a bidirectional RNN (or another recurrent network configuration). It has two hidden states: a forward one $\overrightarrow{\boldsymbol{h}}_i$, and a backward one $\overleftarrow{\boldsymbol{h}}_i$. The annotation, $\boldsymbol{h}_i$, is indicated by the concentration of these two, which is represented in Equation 2.21. After that, in order to generate attention scores, each annotation together with the previous hidden decoder's state $\boldsymbol{s}_{t-1}$ are fed into the alignment model, which demonstrates how closely the inputs near location j and the output at position i, using Equation 2.22. As the alignment model is implemented by combining $\boldsymbol{s}_{t-1}$ and $\boldsymbol{h}_i$ with the method of addition, this attention mechanism is called "additive attention." Equation 2.23 depicts this, where $\boldsymbol{U}_a$, $\boldsymbol{W}_a$, and $\boldsymbol{v}_a$ are learnable attention parameters. A single-layer multilayer perceptron is used here to reduce the computation. Furthermore, the corresponding weight $\alpha_{i,j}$ is obtained by applying the softmax function to each attention score. It is explained by Equation 2.24. Finally, the context vector, which is inserted into the

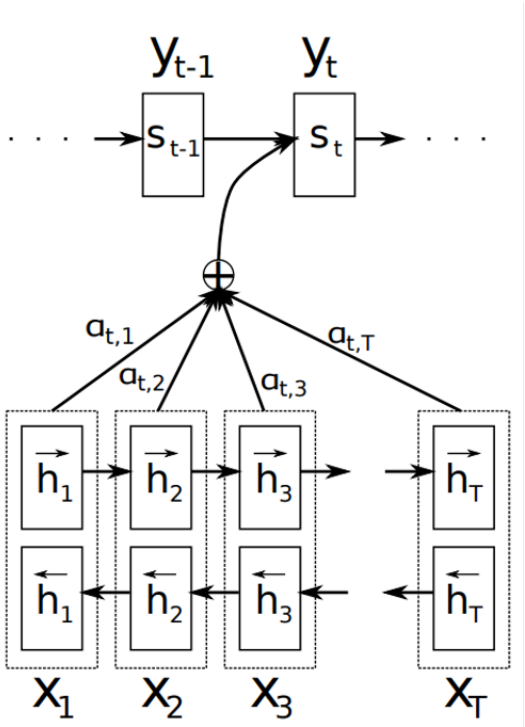

Figure 2.20: Encode-Decoder with Additive Attention Mechanism [116]

decoder, can be obtained as the weighted sum of annotations, as indicated in Equation 2.25.

The attention mechanism has been applied to text classification in recent years. The Sequence representation obtained from Bi-LSTM are fed into the attention layer to replace the $\boldsymbol{s}_{t-1}$ in original Equation 2.22 and 2.23 [117]. This process can determine which features are highly correlated to the final classification.

$$\boldsymbol{h}_i = [\overrightarrow{\boldsymbol{h}}_i^\top; \overleftarrow{\boldsymbol{h}}_i^\top]^\top, i = 1, \ldots, n \tag{2.21}$$

$$e_{i,j} = \mathrm{a}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_j) \tag{2.22}$$

$$\mathrm{a}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_j) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a \boldsymbol{s}_{t-1} + \mathbf{U}_a \boldsymbol{h}_j) \tag{2.23}$$

$$\alpha_{i,j} = \frac{\exp(e_{i,j})}{\sum_{k=1}^{n} \exp(e_{i,k})} \tag{2.24}$$

$$\boldsymbol{c}_i = \sum_{j=1}^{n} \alpha_{i,j} \boldsymbol{h}_j \tag{2.25}$$

2.7.3.9 Transformer

Transformers are encoder-decoder type architectures based solely on the attention mechanism, implying that any RNN is excluded from this architecture [118]. It is depicted in Figure 2.21. The encoder is in the left half of the figure, while the decoder is in the right half. At a high level, an encoder converts a sequence of input data into a continuous abstract representation that contains all the information that was previously learned from the input [119]. Then, the decoder takes that representation and previously generated symbols to produce a sequence of output symbols step by step [120].

In the beginning, the inputs are fed into input embedding to map each word to a vector of continuous values so that textual representation is converted into a numerical representation. Then, the positional encoding is done to insert positional information into the numerical representation, as recurrence does not exist. The encoder consists of N_x identical layers that are stacked. Each layer is composed of two sublayers: a multi-headed attention sublayer and a position-wise feed-forward sublayer [121]. The multi-head attention sublayer is applied with the self-attention mechanism, which enables the model to aggregate all the information from other words for each word in the input [120]. The output of the multi-head attention layer is added to the numerical representation of the input with positional information. Its result is then normalized and goes into the position-wise feed-forward sublayer for further processing. The decoder has similar sublayers as the encoder, which behave similarly to those of the encoder except for multi-headed attention layers [122]. For the first multi-headed attention, it only associates each word with the words previously generated since future words are not produced at this time. For the multi-headed attention layer, it compares the encoder's input and the input from the decoder. This process enables the decoder to select the relevant encoder input to

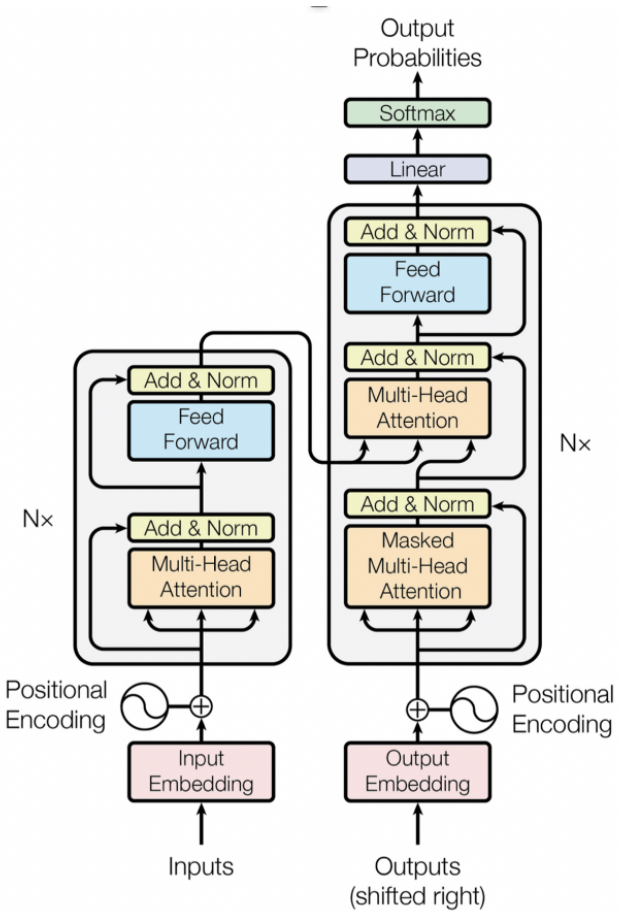

Figure 2.21: Transformer Architecture [119]

focus on. Finally, a linear layer and a softmax layer are used for classification. The number of neurons in the linear layer is equal to the number of words in the vocabulary. The Softmax layer is used to transform outputs from the linear layer into probabilities. The predicted word should be the one with the highest probability. The main advantage of this architecture is parallelization. The word-by-word sequential processing of input by RNN makes parallelization difficult. The Transformer employs a self-attention mechanism to substitute recurrence, allowing for parallelization and significantly reducing training time [123].

2.7.3.10 Bidirectional Encoder Representations from Transformers

A recently proposed language representation model called Bidirectional Encoder Representations from Transformers (BERT) aims to pre-train deep bidirectional representations to extract context-sensitive characteristics from input text [124]. It was firstly proposed in 2018 and achieved state-of-the-art accuracy on many NLP tasks [125]. The biggest innovation in BERT

is its two pretraining tasks, namely the masked language model and next sentence prediction [126].

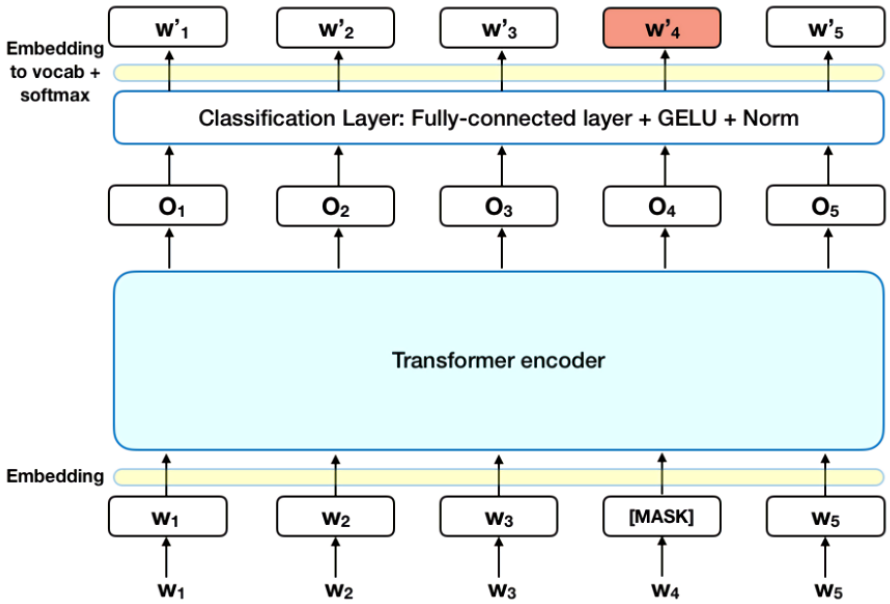

Figure 2.22: Masked Language Modelling Illustration [127]

For the first pretrained task, in each word sequence, 15% of the words are altered by a [Mask] token randomly before being fed into the BERT, as described in Figure 2.22. Taking into account the context given by non-mask words in the sequence, the original values of covered words are predicted by the model. For the second pretrained task, the model receives a sentence pair and learns to predict if the second sentence in the pair is the subsequence sentence of the first one. 50% of the sentence pairs are selected so that the second sentence in a pair just comes after the first one. For the other 50%, the second sentence of a pair is chosen at random. To make the representation of inputs signify both single sentences and sentences in pairs, a new input representation is presented, as shown in Figure 2.23. It has several characteristics. First of all, the [CLS] token and the [SEP] token are used to represent the start of a sentence and the end of a sentence separately. Then, sentence pairs are combined into a single sequence, and a learned segment embedding is added to each token to denote which sentence it belongs to. Finally, each token is given a positional embedding to indicate its position in the sequence.

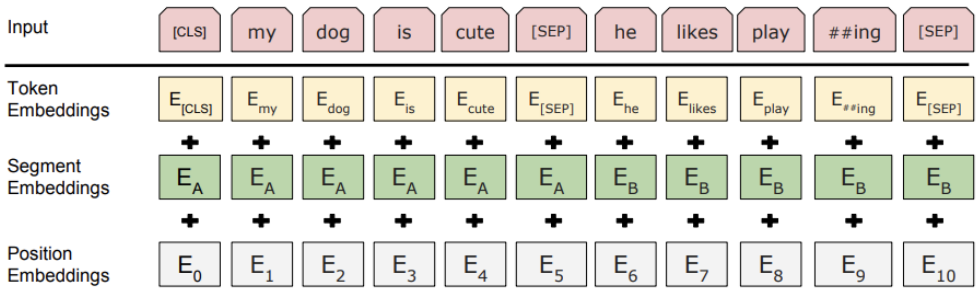

Figure 2.23: Input Representation of BERT [125]

2.7.4 Model Evaluation

2.7.4.1 Cross-Validation

In supervised machine learning projects, models are trained with the aim of getting their optimal values of weights and biases using labeled data samples. With utilizing the same dataset for both training and testing, models suffer from the problem of overfitting so that they cannot perform well on unseen data [128]. That problem can be depicted in Figure 2.24. Hence, some methods are required to fit more accurate models.

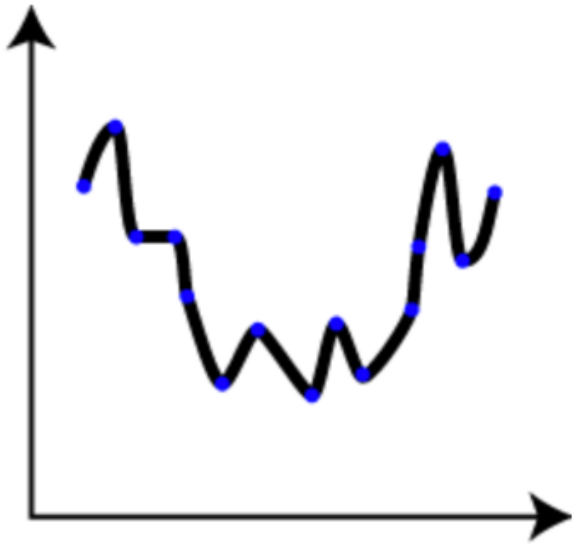

Figure 2.24: An Example to illustrate overfitting

Cross-validation, also namely rotation estimation is a measurement of assessing the performance of a predictive model and how the results of statistical analysis will generalize to an independent dataset [129]. It tackles the overfitting problem by partitioning the whole set into

multiple train-test splits so that different, unseen partitions of the dataset can be chosen as the test set to validate the model iteratively [130]. In typical cross-validation, the training and test sets must cross over in subsequent rounds such that each data point gets a chance to be validated against [131]. There are different types of cross-validation techniques. In this section, K-fold cross-validation is introduced. In K-fold cross-validation, the entire dataset is divided into k equal-size segments. Each segment is also referred to as a "fold." The technique is called "K-fold" since there are k segments. One segment is used as a validation set while the other $k-1$ segments are leveraged for training. The process is repeated k times until each segment is used for validation. Figure 2.25 can illustrate the 5-fold cross-validation.

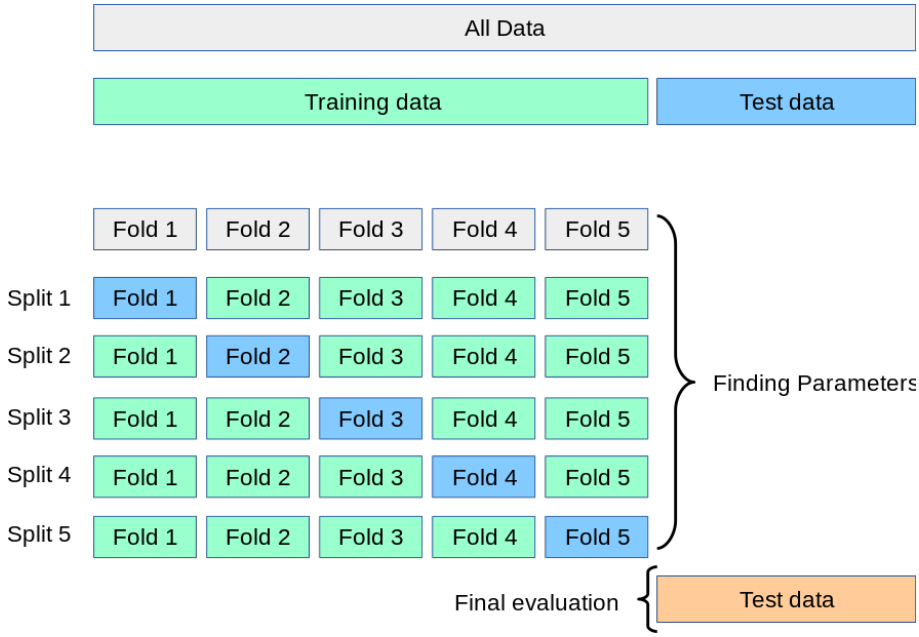

Figure 2.25: 5-Fold Cross-Validation [132]

2.7.4.2 Evaluation Metrics

It is essential to evaluate the models' performance in machine learning projects. In this section, several evaluation metrics which assess the performance of models are discussed.

In general, confusion matrices are $N \times N$ matrices, where N represents the number of classes, as presented in the evaluation of scientific models in many areas such as natural language processing and computer vision [133]. However, only confusion matrices for binary classification are discussed here because a multiclass classification problem can be reduced into several binary classification problems in many different ways [134]. A confusion matrix

for binary classification has two rows and two columns, as plotted in Figure 2.26. Predicted and true values are represented in confusion matrices.

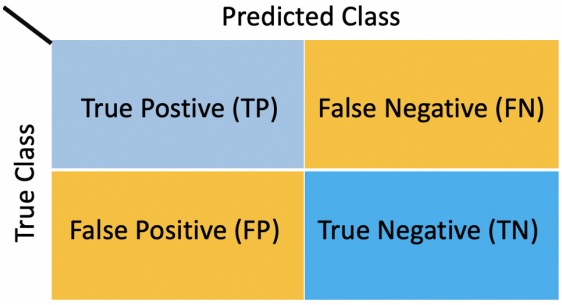

Figure 2.26: Confusion Matrix for Binary Classification

There are four important terms:

- True Positive (TP): The number of positive class samples which are predicted correctly.
- False Positive (FP): The number of positive class samples which are predicted falsely.
- False Negative (FN): The number of negative class samples that are predicted incorrectly.
- True Negative (TN): The number of negative class samples that are predicted correctly.

These terms can be used to describe other evaluation metrics. Accuracy can describe the ratio of correctly predicted samples among all the samples, as described in Equation 2.26. Recall can explain the ratio of correctly predicted samples among all the positive samples, as shown in Equation 2.27. Precision can explain the fraction of true positive predictions among all the positive predictions made by the model. Equation 2.28 can present it. The F-measure gives the harmonic mean of precision and recall values, as illustrated in Equation 2.29. It is one of the most important metrics since it combines information about the precision and the recall [135].

$$Accuracy = \frac{TP+FN}{TP+FN+FP+TN} \tag{2.26}$$

$$Recall = \frac{TP}{TP+FN} \tag{2.27}$$

$$Precision = \frac{TP}{TP+FP} \tag{2.28}$$

$$F1 = (\frac{recall^{-1} + precision^{-1}}{2})^{-1} \tag{2.29}$$

2.8 Related Research

According to the chaos report [136], the users' involvement and their continuous feedback have been proven to be critical factors for software projects. Bano and Zowghi found that although user involvement has positive effects on projects, it must be managed with great attention. Otherwise, more problems can be caused as opposed to benefit [137]. Besides, a report highlights that user feedback can provide developers with valuable information so that not only the quality of the software is improved, but also missing functionality can be discovered. With the development of requirements engineering, the focus of researchers gradually shifts from traditional methods to cloud-based ones. As discussed in the report [138], it is significant to have tools to analyze the users' feedback with scalability issues in the event of several users. This point is also emphasized in the article [139]. Seyff et al. proposed utilizing user feedback from mobile devices to continuously collect user requirements, such as implicit information from apps [140]. Bug repositories are one of the most studied tools for collecting user feedback in the development and maintenance of software projects [141].

Previous studies focused on the use of machine learning algorithms for app review filtering. Fu et al. implemented a system to remove inconsistent reviews whose sentiments are different from their ratings [142]. Chen et al. developed a novel framework to filter out irrelevant reviews from raw text reviews [143].

Furqan used feature engineering techniques combined with some supervised machine learning models, such as random forest, AdaBoost classifier, and the logistic regression model, to classify reviews as happy or unhappy [144]. Standard evaluation metrics like precision, recall, and F1 score were leveraged. As a result, they discovered that logistics regression combined with TF-IDF and Chi2 outperformed the others.

Maalej and Nabil used probabilistic techniques to classify app reviews into four types: bug reports, feature requests, user experiences, and rating [145]. Different types of classifiers (binary vs multiclass) were implemented to classify app reviews. Each classifier was built utilizing different classification algorithms, including Naïve Bayes, decision trees, and maximum entropy, combined with various classification techniques for document classification, metadata, and combinations of these two. The evaluation was done by using the standard metrics of precision, recall, and F1 score. They found that the binary classifier built with Naïve Bayes clearly outperformed the other classifiers. Concerning classification techniques, no "best" one exists for all review types.

Researchers utilised ensemble models to categorize app reviews into classes [146] [147]. Especially, Guzman et al. experimented with ensemble methods to classify app reviews into categories relevant to software evolution [148]. They first investigated the performance of four individual machine learning algorithms (Naïve Bayes, Support Vector Machine, Logistic Regression, and Neural Networks) that have previously performed well at categorizing text. Then ensemble methods, which enhance the strength and mitigate the deficiencies of individual classifiers, were utilized to combine their predictions. They concluded that the classifier with the neural network performed the best among classifiers implemented by four different machine learning approaches. Ensemble methods generally worked better than individual machine algorithms in app review classification.

Al Kilani et al. used machine learning methods with natural language techniques to categorize a collection of healthcare-domain app reviews into multiple classes (bug reports, new feature requests, application performance, and user interface) [149]. The performance of four machine learning methods, such as Naïve Bayes, Multinomial Naïve Bayes, Random Forest, and Support Vector Machine, was examined. In their project, the effects of data labels' confidence and the problem of an imbalanced dataset were also explored. In their experiments, the performance of the multi-nominal Naïve Bays algorithm was shown to be the best. The overall performance was improved when the high-confidence dataset, where two experts agreed on a label for a given review, was applied. Furthermore, the imbalanced dataset problem was alleviated by applying resampling techniques, which resulted in a rise in the accuracy score.

The simplest textual machine learning approach utilizing only lexical information (the bag-of-word approach) was compared with the more advanced one, a deep learning model (Con-

volutional Neural Network) with different word embedding settings, explored by Kairit et al. [150]. In their study, standard evaluation metrics like precision, recall, and F1 score were leveraged. They discovered that pretrained word embeddings with a fine-tuning option outperform the other three options for deep learning models (random, freeze, and fine-tuning). When the training set is small, Convolutional Neural Network (CNN) models don't perform better than BOW models on average. If the training set could be extended, CNN could take a big advantage. However, Aslam et al. proposed a deep learning approach (CNN) for app review classification that leverages statistics of reviews as non-textual information for app review classification [151]. Using this approach, the F1 score was changed from 72.41 percent to 94.71 percent. This tangible result indicates the approach has dramatically improved the state of the art.

A BERT-based classifier was designed and tested successfully to achieve a state-of-the-art accuracy of 87 percent on common tasks for user feedback analysis [152]. From a vast number of online product reviews, a domain-oriented deep learning strategy for discovering the most crucial customer requests, such as new app product features and bug reports, was presented [153].

2.9 Critical Analysis

On the basis of the historical research, certain flaws and gaps have been identified:

- Limited supervised learning methods: The majority of past researchers only focused on traditional machine learning algorithms such as support vector machine, Naïve Bayes, and decision tree. More advanced ones, such as recurrent neural networks and Bert, were rarely taken into account.

- Limited attention to the characteristics of the dataset: Few researchers cared about the characteristics of their training dataset, which was often imbalanced. This made the model fail to classify minority classes accurately.

- Limited work on the comparison between the traditional machine learning models and more advanced models such as BERT in the field of app review classification.

2.10 Summary

In the chapter, the background and related research are discussed. As this project is in the field of NLP, the NLP pipeline (data collection, text cleaning, preprocessing, feature engineering, modeling, and evaluation) was first introduced. Then, a technique, data augmentation, is presented to overcome the problem that the data collected is not enough for training. After that, techniques relevant to text preprocessing and feature engineering are illustrated, respectively. As features are not always pertinent to the target, some feature selection approaches involving filter methods and wrapper methods are mentioned. Besides, some machine learning approaches are highlighted to create models with high performance. Methods of evaluation are also specified to ensure models are implemented properly. Furthermore, literature associated with the project is investigated and analyzed to clarify the current state of the field. In the end, flaws in previous research that are intended to be improved in this project are revealed.

Chapter 3

Design and Implementation

This chapter is separated into four parts. First of all, this chapter will specify an appropriate methodology required for successful experiments. At the same time, the experiments are designed under the proposed methodology. The tools that were used in the project will then be clarified. After that, the project's implementation, which strictly follows the experimental design, is going to be detailed. To end this section, this chapter will be summarized.

3.1 Design

To accomplish the objectives of this project and also take into account the NLP pipeline discussed in Section 2.2, five main phases, including background research, data preparation, classification, and evaluation, are designed as plotted in Figure 3.1. Background research that has been completed is also shown in the figure, just to make the methodology complete.

3.1.1 Background Research

At the stage of background research, two subtasks, project understanding, and literature review, should be accomplished. To get a fundamental grasp of this project, several points should be considered carefully, such as the input and output of this project (Section 1.1), the sorts of labels that models will be trained on (Section 1.2), the specific machine learning techniques that can be utilized (Section 2.7), and several evaluations metrics that could be leveraged (Section 2.7.4.2). After thinking about these aspects, the student could move on to the next task, the literature review. To end this task, recent work done by researchers should be investigated and

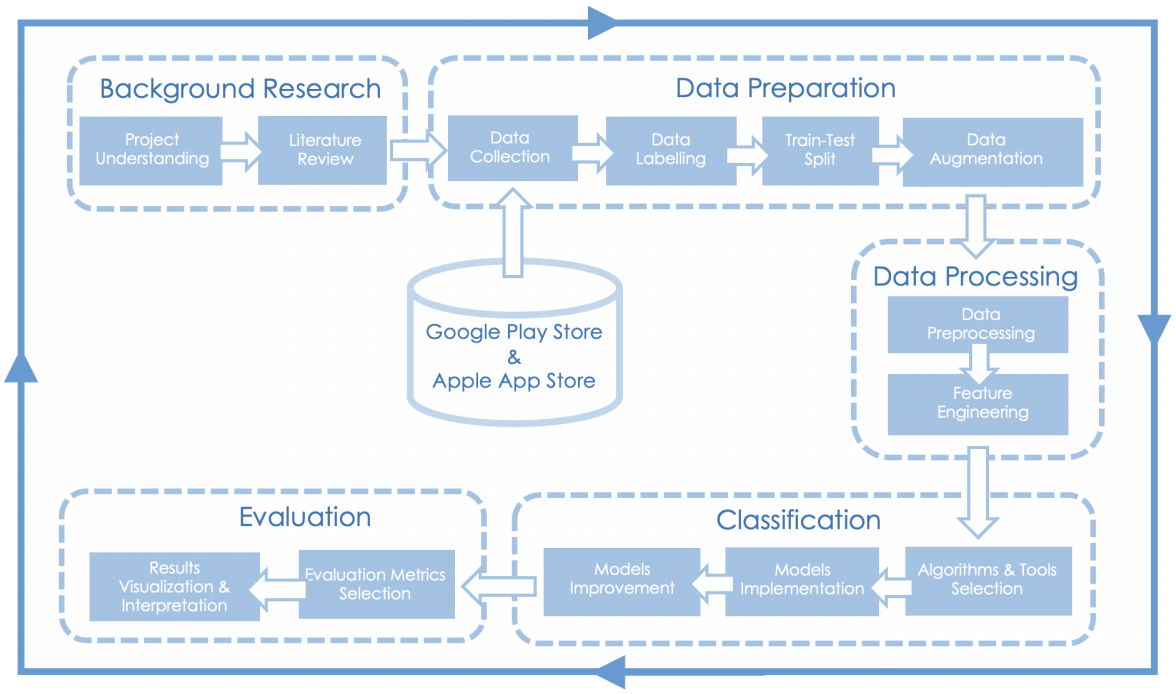

Figure 3.1: Research Methodology

analyzed to clarify the current state of this field (Section 2.8). In particular, any weaknesses. in their studies should be noted so that issues that should be emphasized in this project are identified (Section 2.9). These tasks can serve as a good starting point for this project and lay a solid basis for future work.

3.1.2 Data Preparation

The stage of data preparation will focus on four main tasks: data collection, data labeling, train-test split, and data augmentation. Since a dataset is not given for this project, a suitable one should be found or even made. If a proper dataset can be found with enough labeled app reviews, this step can be skipped. Otherwise, raw text files that contain enough app reviews from the Google Play Store or the Apple App Store should be found. Then, app reviews ought to be gathered and labeled manually. After that, the data needs to be split into two parts: a training set and a testing set, which will be used to train and test the models separately. This step is crucial since the subset of training data might be used to test the model accidentally, which results in the problem of data leakage. The models will learn something that should not be known, and this will invalidate the evaluation process. To make models perform better, data augmentation could be leveraged on the training set to generate more training data, thus

improving the models' performance.

In accordance with the idea illustrated, a more detailed data preparation process is designed, as shown in Figure 3.2.

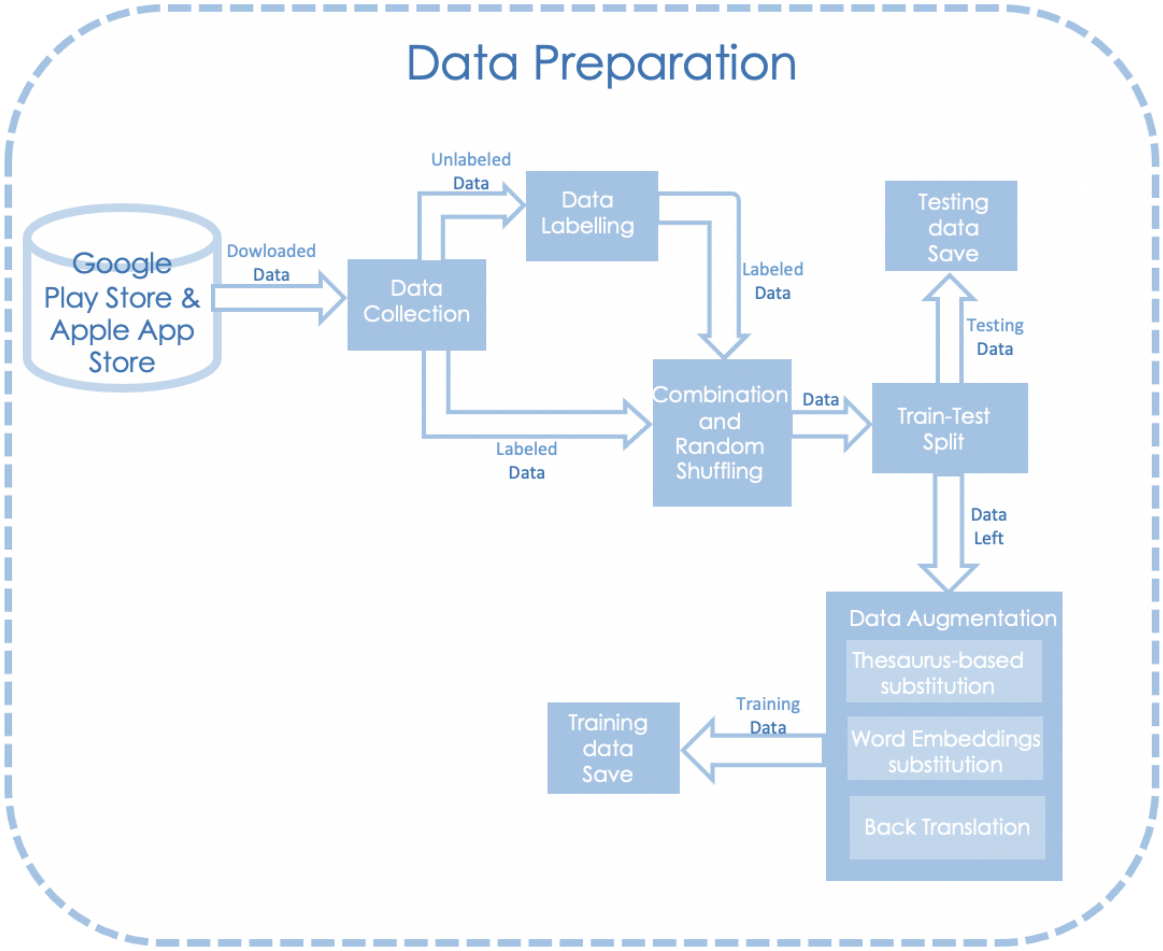

Figure 3.2: Design of Data Preparation

Data Collection

A dataset (`https://data.mendeley.com/datasets/5fk732vkwr/1`) contains 3691 reviews with associated labels was discovered, as shown in Figure 3.3. Data is not enough to train deep learning models. Other datasets found only comprise raw text or data with irrelevant labels. Hence, some of the datasets still need to be labeled.

A	B	C	D	E	F	G	H	I
id_num	review	id	title	task	dataSo	appId	date	
1	Besides th	264	Almost pe	PD	RE2014_a			
2	This could	111	Take a pho	PD	AppStore_	38269856		
3	I can't ope	210	Won't ope	PD	RE2014_a			
4	Use to lov	53	Not Worki	PD	AppStore_	40429986		
5	UrrrrmAft	13		PD	PlayStore_	#3_Need f	12:44	
6	This app s	83	Needs wo	PD	AppStore_	30311312		
7	Album fun	3		PD	PlayStore_	#10_Perfe	26:07:13:08:31:49	
8	I'm very di	337	Excel files	PD	RE2014_a			
9	NEEDS TO	1		PD	PlayStore_	#8_Wo ist	07:35	
10	It's good if	122	It's good t	PD	AppStore_	43887595		
11	CrashCrasl	229	Crashes	PD	RE2014_a			
12	Was great	323		PD	RE2014_a			
13	The white	340	Yup It's ug	PD	RE2014_a			
14	Would like	12		PD	PlayStore_	#10_Fotoc	25:11:13:02:26:01	
15	This use to	341	?? VERY B	PD	RE2014_a			
16	The app w	265	Started ou	PD	RE2014_a			
17	After the l	272	Crash	PD	RE2014_a			
18	I liked very	217	PDF Proble	PD	RE2014_a			
19	Can't upda	190	problem	PD	RE2014_a			
20	It says erro	131	It won't le	PD	AppStore_	50999351		
21	ok serious	11		PD	PlayStore_	#2_flipagr	11:01:14:06:00:35	
22	The updat	222	Dropbox u	PD	RE2014_a			
23	I love the	67	Hats	PD	AppStore_	45402127		
24	EffectPlea	331		PD	RE2014_a			
25	It's really s	76	Meh	PD	AppStore_	28488221		
26	Can't use i	200	Problems	PD	RE2014_a			
27	Won't ope	96	Waste of t	PD	AppStore_	33801082		
28	COME ON	144	Still CAN'T	PD	RE2014_a			
29	So frustrat	95	Just doesn	PD	AppStore_	33638199		
30	Bookmark	100	Buggy	PD	AppStore_	35018983		
31	Recent up	354	Upgrade c	PD	RE2014_a			
32	I like Trip A	284	5? informa	PD	RE2014_a			
33	Search do	305		PD	RE2014_a			
34	Plz more t	17		PD	PlayStore_	#10_Fotoc	14:01:14:16:46:44	

Figure 3.3: Dataset with Relevant Labels

Data Labelling

Since data are labeled manually, it is extremely likely that mistakes are made by a single student throughout the labeling process, resulting in unreliable classification. To reduce this risk, a fellow student needs to be found, and the definitions of categories will be explained to him. Then the same labeling task can be conducted by each person individually. Cross-checking should be done after completing the task to ensure that the category of each review in the dataset is agreed upon by both of them. After that, these manually labeled data are merged into the labeled data downloaded online, and the whole dataset is randomly shuffled to make different types of data evenly distributed. Finally, a labeled dataset with 13000 entries is expected to be attained.

Train-Test Split and Data augmentation

Train-Test split will be done using the common ratio: 70% for training and 30% for testing. As some types of reviews (`PD` and `FR`) appear more frequently than others [7], I first plan to get balanced testing set with 900 entries for each class, which ensures a classifier's performance

for each class is evaluated unbiasedly. Then, it is extremely likely that the rest of the data is imbalanced. To mitigate this problem, over-sampling will be applied to the minority classes so that each class could have 2100 samples by using data augmentation techniques. To balance the strengths and weaknesses of each technique, different augmenters that implement distinct approaches will be employed and selected randomly to do oversampling. Besides, samples are removed from the majority classes to get 2100 samples in each class. It is worthwhile to emphasize that the train-test split is done before the data augmentation since the data leakage problem could occur if the order of these two steps is reversed.

3.1.3 Data Processing

The text processing stage aims to tidy up the text and convert the data into numerical representations. There are two tasks in this stage, data preprocessing and feature engineering. To convert the text into a more digestible format, subprocesses discussed in Section 2.4, such as tokenization, lowercaseing, stop word removal, and lemmatization, are applied to the collected data. Then, feature engineering is leveraged to vectorize the text and make it ready to be fed into the models.

As stated by the major points discussed above, the data processing phase is designed, as depicted in Figure 3.4.

Text Preprocessing

The training data and testing data are separately applied with cleaning, tokenization, lowercasing, and lemmatization in order. Primarily, punctuations, URLs, and stop words are removed from the original text for data cleaning since these things are redundant for the machine to understand the meaning of the text. Then, each data sample is separated into a list of words prepared for feature engineering. Finally, each word in the list is lowercased and converted into its root for consistency.

Feature Engineering

Three different feature engineering techniques, including the bag of words, Term Frequency-Inverse Document Frequency, and word embeddings, will be applied to the lists of words so

that we can get three different kinds of features. In particular, word embeddings will appear as an embedding layer in a neural network.

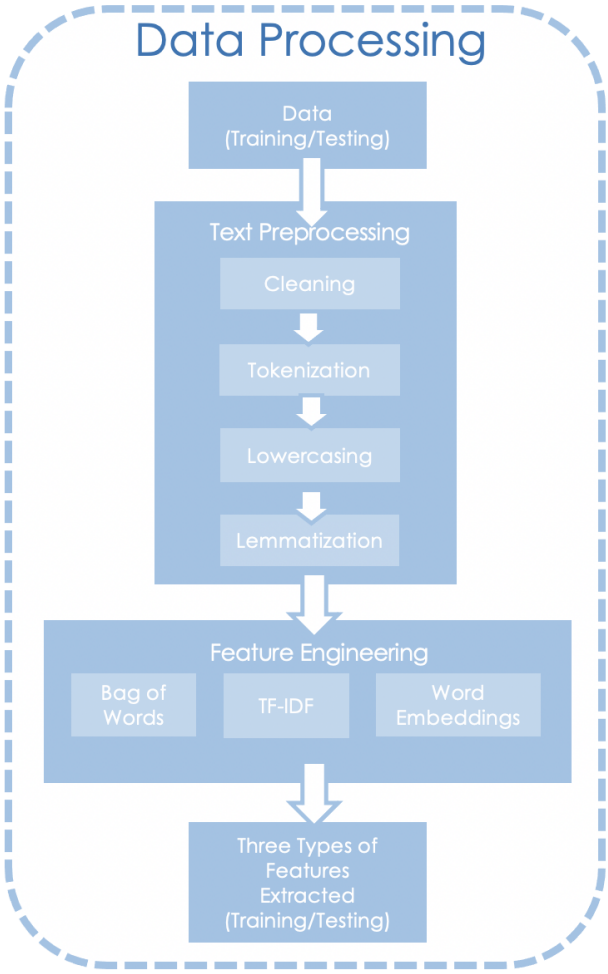

Figure 3.4: Design of Data Processing

3.1.4 Classification

Classification is a pivot point in the development of the project. Algorithms shown to have good results in the literature discussed in Section 2.8 are chosen. Tools and packages that resulted in high implementation efficiency and demonstrated excellent usability and robust functionality should be selected and learned. Then, various classifiers with chosen algorithms will be implemented and trained using the selected tools and packages. Ultimately, different ways should be attempted to improve the performance of the models.

As indicated by the main aspects addressed, the data processing step is designed, as demonstrated in Figure 3.5.

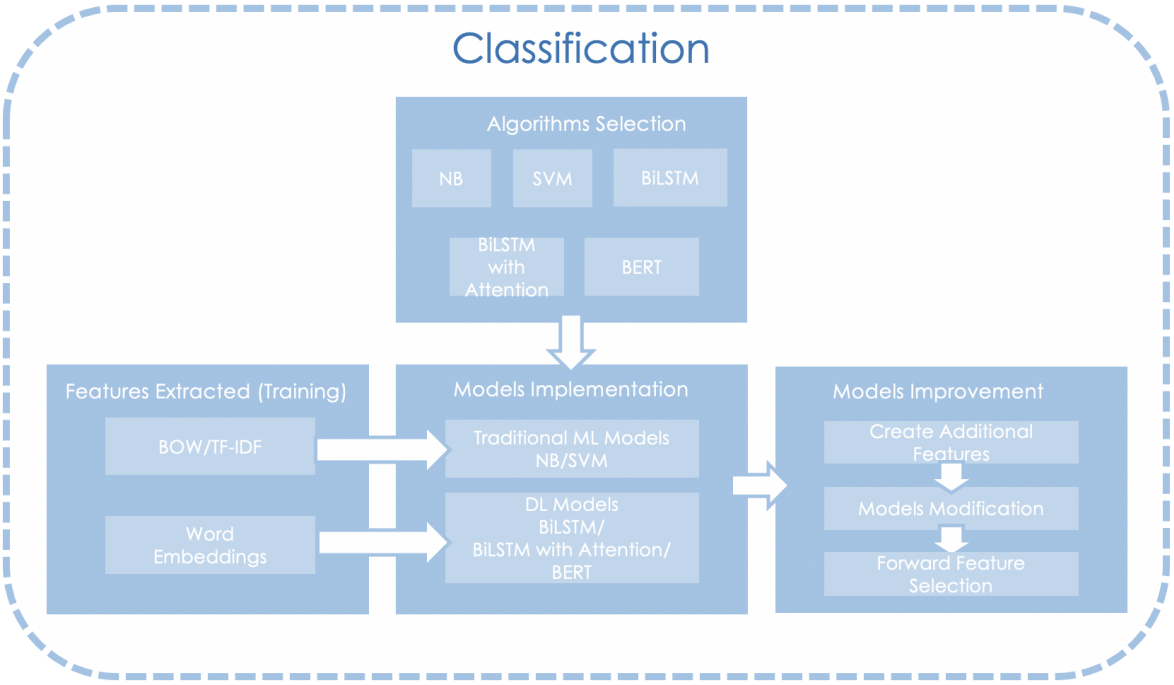

Figure 3.5: Design of Classification

Algorithms Selection

The selection of NB and SVM is made because of their capabilities for classification. Additionally, they were used to classify app reviews in the earlier study, with positive outcomes detailed in Section 2.8. Bi-LSTM, Bi-LSTM-attention and BERT are selected deep learning algorithms, which are frequently utilized in other text classification problems.

Models Implementation

Traditional machine learning algorithms such as NB and acrshort SVM, will be implemented and trained using BOW or TF-IDF..Pretrained word embeddings will be used to train Bi-LSTM, Bi-LSTM-attention and BERT. These different feature engineering techniques and different machine learning methods are discussed in Section 2.5, and Section 2.7 respectively.

Models Improvement

Additional features, including each review's length, polarity, and subjectivity, which might aid the classification, will be generated. Then models will be modified to receive different numbers of features. Finally, forward feature selection, discussed in Section 2.6.2.1, will be

used to select a subset of features that can improve the models' performance the most.

3.1.5 Evaluation

The Evaluation stage entails two tasks: selecting evaluation metrics and visualizing and analyzing the results. First of all, proper evaluation metrics should be chosen to evaluate different aspects of the classifiers. Classifiers will then be measured against the testing set. Related evaluation metrics are going to be computed. Finally, results should be plotted using different visualization approaches and interpreted according to them.

Based on the main points illustrated, Figure 3.6 is plotted to describe the design of the evaluation.

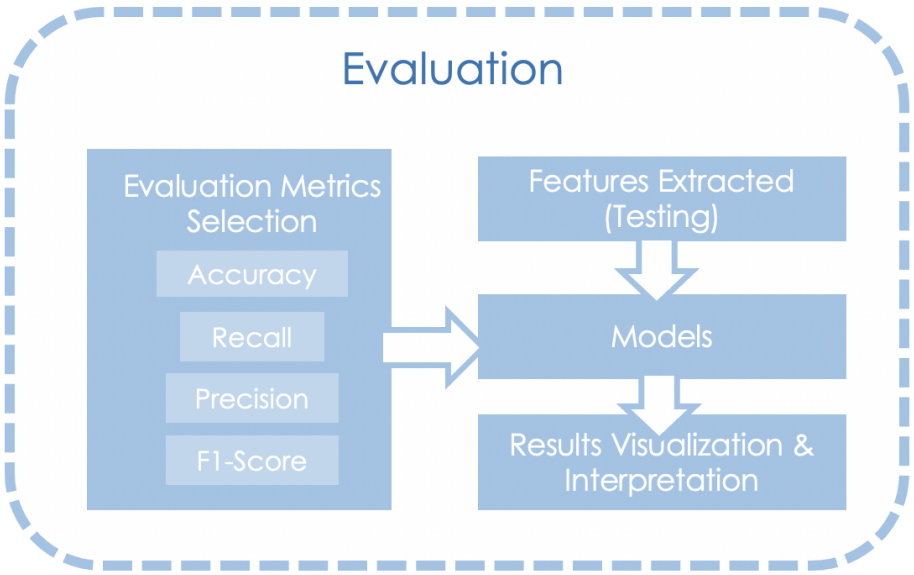

Figure 3.6: Design of Evaluation

Evaluation Metrics Selection

Accuracy, Recall, Precision, and F1-score, explained in Section 2.7.4.2, are selected to measure the performance of the classifiers from different perspectives. Accuracy reveals how frequently a machine learning model is overall correct. The precision is a measure of how well a model can predict a particular category. The recall can indicate how frequently a particular category is identified. By calculating the harmonic mean of a classifier's precision and recall, the F1-score integrates both into a single metric.

Results visualization and Interpretation

The models implemented can be used to predict the labels of features extracted for testing. Calculations for the chosen evaluation metrics can be made based on predictions and expectations. In order to make it simple to compare the performance of different models, a large table containing all of the results from various models will be created at the end.

3.2 Utilised Tools and Libraries

Jupyter Notebook

To create and share documents with well-formed equations, adequate visualizations, neatly arranged text, and real-time code, a server-client application can be used, named Jupyter Notebook. Though it needs to be edited through a web browser, it does not mean that it has to be executed by accessing the network. It can also be utilized on a computer without any Internet access. It supports different kinds of programming languages. The project was developed in this environment with Python programming language.

Numpy

Numpy is one of Python's core scientific computing libraries. It can provide a multi-dimensional object, numerous derived objects, and a variety of methods for fast operations on arrays. These operations vary from basic linear algebra operations to discrete Fourier transforms. The ndarray object, the core of the NumPy package, incorporates n-dimensional arrays of homogeneous data types with various operations carried out in compiled code for speed. In addition, only a minority of the package parts are written in Python. Most parts that demand fast computation are written in C and C++ and optimized to work with current CPU architectures. This library is utilized for matrix operations in this project.

Pandas

A Python Package called pandas offers quick, adaptable, and expressive data structures intended to make handling relational data straightforward. It seeks to serve as the essential,

high-level building block to analyze data efficiently in Python. In order to store and manage 2-D tabular data, a data structure named Data Frame is utilized. This package offers a wide range of features for the DataFrame, including customized indexing, data alignment, label-based slicing, and data aggregation. It is used to read the data files and convert them into 2-D tabular form.

TextAttack

A Python framework called TextAttack is used to conduct adversarial attacks against NLP models. Besides, it can also be utilized for data augmentation and model training. A search method, target function, transformation, and constraints set are the four blocking blocks upon which TextAttack constructs attacks. The modular structure of TextAttack makes it simple to adapt it to new NLP models and attack approaches. At the moment, TextAttack supports attacks against entailment, classification, and translation models. In this project, the data are augmented using this package.

NLTK

NLTK (Natural Language Toolkit), one of the most popular platforms, can be utilized to develop Python programmes in order to handle the data of human language. Along with a collection of text processing packages for tokenization, tagging, parsing, stemming, and semantic reasoning, it offers simple interfaces to lexical resources and more than 50 corpora, including WordNet. In addition, an active discussion forum is also provided to report bugs and share ideas. This project is used to preprocess the text.

VaderSentiment

VADER (Valence Aware Dictionary and sentiment Reasoner), which is known as a lexicon and rule-based sentiment analysis tool, is customized precisely to sentiments expressed on social media. VADER makes use of a variety of sentiment lexicons, each of which is a collection of lexical features that are often labeled as either positive or negative depending on their semantic orientation. It informs us of the compound score to specify the extent of positivity or negativity in sentences. It is used to get compound scores of each sample, which is used as an extra feature to improve models.

TextBlob

A python package called TextBlob is utilized to process textual data. A straightforward API is supplied to cope with typical natural language processing tasks involving non-phrase extraction, sentiment analysis, and part-of-speech tagging. It can analyze the sentences' subjectivity, which measures how much factual and personal information is present in the text. It is used to get the subjectivity score for each sample. That can be considered another feature to improve models. It is used to read the data files in 2-D tabular form.

Sklearn

Sklearn, also known as Scikit-Learn, is the most effective and reliable Python machine learning library. Through interfaces provided, it offers a variety of practical tools for statistical modeling and machine learning, such as regression and classification. This library, which is primarily written in Python, is mainly based on Numpy, Scipy, and Matplotlib. This library is employed to build traditional machine learning models and calculate evaluation metrics.

TensorFlow

TensorFlow was created mainly for doing deep learning and is open-sourced by Google. Traditional machine learning is also supported. It can handle substantial numerical computations efficiently and reliably. Tensors, which are multi-dimensional arrays with more dimensions, are the only type of data TensorFlow takes. When operating extensive datasets, multi-dimensional arrays come in quite useful. Data flow graphs with nodes and edges serve as the foundation for TensorFlow's operation. It is considerably simpler to distribute the execution of TensorFlow code using GPUs across a cluster of computers because the execution mechanism takes the form of graphs. This library is used for building deep learning models and handling some numerical computations while building these models.

Git

Git is a distributed version control system that is free and open source and is made to handle projects of all sizes quickly and efficiently. In this project, Git is used for backing up the latest version's source code and data files.

3.3 Implementation

According to the design illustrated in the last section, the relevant experiments were implemented.

3.3.1 Data Preparation

3.3.1.1 Data Collection

Data were collected from four different resources. The first dataset (`https://guxd.github.io/srminer/srminer.html`) was produced by developers from the Hong Kong University of Science and Technology with the goal of summarizing product evaluations from the user reviews collected from the Google Play Store. The second dataset (`https://ase.in.tum.de/lehrstuhl_1/images/publications/Emitza_Guzman_Ortega/truthset.tsv`) was from a thesis that investigates applying ensemble methods to app review classification. The third dataset (`https://data.mendeley.com/datasets/5fk732vkwr/1`) was also from a paper that discusses employing machine learning techniques to automatically classify app reviews for the purposes of requirement engineering. The last dataset (`https://www.kaggle.com/datasets/prakharrathi25/google-play-store-reviews`) was obtained from Kaggle and is a collection of app reviews from the Google Play Store. Among them, only the third dataset has relevant labels. Therefore, the other three datasets need to be labeled manually.

3.3.1.2 Data Labelling

A roommate of mine, a computer science student at the University of Manchester, helped categorize the dataset. Before doing this task, relevant definitions of the labels had been provided to him. During the labeling, we discussed the data that needed to be labeled each day. Then, we labeled these datasets in separate rooms. At the end of the day, we did cross-checking to ensure we got the same labeling results. Otherwise, discussions between us were done to reach a consensus. After that, the dataset that had already been labeled was combined with the manually labeled dataset. Finally, as shown in Figure 3.7, a labeled dataset with 13000 entries was obtained.

	A	B	C
1	raw text	y	
2	"Just need sc	FR	
3	- The Godfat	RT	
4	tilt should be	FR	
5	Network ero	PD	
6	Awesome (y	RT	
7	I like it	RT	
8	"It's is great,	FR	
9	This is some	FR	
10	¬†:/ I like thi	UE	
11	Keyboard shu	PD	
12	¬†I'm in love	FR	
13	Awful What	UE	
14	There are 3 p	PD	
15	¬†Handy and	UE	
16	Ok what	UE	
17	Okay App wc	FR	
18	Text size sho	FR	
19	LOVE IT!! D	RT	
20	Great app. G	RT	
21	¬†Cool app D	UE	
22	Please fix thi	PD	
23	Can ad an op	FR	
24	Awesome ne	UE	

Figure 3.7: Dataset for This Project

3.3.1.3 Train Test Split

The dataset was read via the $read\_excel()$ method of the Pandas package. Samples of each class were selected via $df.loc[df['y'] == class\_name]$. Then, each group was converted into a NumPy array by using $to\_numpy()$. In addition, the method $np.random.shuffle()$ randomly shuffled the NumPy array. The first 900 samples were retrieved in each class using the slicing technique. They were combined using $np.concatenate$ to obtain a single NumPy array. It was converted back to a pandas data frame and saved into a CSV file with the usage of $to\_csv()$. Furthermore, the imbalanced problem must be mitigated for the rest of the data.

3.3.1.4 Data Augmentation

Three data augmenters (easy data augmenter, embedding augmenter, and back-translation augmenter) were created and put in a list. The rest of the data was put into a dictionary whose keys were the names of the classes. The corresponding value was a list of samples that belonged to that class. Then, the first 2100 samples were kept if the length of a list exceeded 2100. Otherwise, the gap between 2100 and the length was the number of samples that needed to be augmented. This number o of samples was selected for being augmented. For each of

the samples that needed to be augmented, a random integer ranging from 0 to 2 was generated. By taking this integer as the index of the augmenters' list, a random data augmenter can be accessed to augment each selected sample. The results of the augmentations were appended to the end of the corresponding list. After this process, the dictionary was converted into a data frame, which was then randomly shuffled and saved into a CSV file. This file is used as the training dataset.

3.3.2 Data Processing

3.3.2.1 Text Preprocessing

The first step for data preprocessing was data cleaning. We can remove the URL within the text by employing the knowledge of regular expressions. The pattern $'(https|http)?:\backslash\backslash/(\backslash w|\backslash.|\backslash/|\backslash?|\backslash=|\backslash\&|\backslash\%)*\backslash b'$ which can represent URLs were replaced by a whitespace. By removing the pattern $['!"\#\$\%\&\backslash'()*+,-./:;<=>?@[\backslash\backslash]\hat{}\_`|']$. In each sample, general punctuation can be removed. A list of stop words in English can be obtained via *nltk.corpus.stopwords.words*("*english*"). Each sample can be split by whitespace to get a list of words that make up the sample. Words in the stop-word list were removed to achieve stop-word removal. Tokenization can be easily done by using $split()$ to split each sample into a list. $lower()$ was applied to each word of a sample list to finish lowercasing. Finally, lemmatization was accomplished by utilizing the $lemmatize()$ method of a word net lemmatizer of the nltk package.

3.3.2.2 Feature Engineering

BOW vectorizer can be made via $feature\_extraction.text.CountVectorizer()$, while TF-IDF vectorizer can be obtained via $feature\_extraction.text.TfidfVectorizer()$. The parameter $max\_features$ of both vectorizers was set to 10000. Then Chi-Square was used to select some of the features that were relevant to labels. Features whose p-values were beyond 0.99 were kept. Finally, 2289 features were attained. For the method of word embeddings, a file (`https://nlp.stanford.edu/projects/glove/`) comprising the pre-trained word vectors with 100 dimensions was downloaded. Then, these word vectors were read from the file and stored in a dictionary. Its keys were the textual form of each word, and corresponding values were its numerical form. A *kprocessing.text.Tokenizer* was created to fit onto the training

dataset, thus obtaining the vocabulary. Each sample was then transformed into a sequence via *texts_to_sequences* function. To make each sequence have the same length as the longest sequence, the *kprocessing.sequence.pad_sequences* method was utilized to add extra zeros to each. Depending on the type of dataset that was turned into the sequences, these obtained sequences were utilized to either train or test the models. Moreover, the numerical form of each word in the vocabulary was found in the dictionary. A two-dimensional array used to assign weights to embedding layers in later steps can be attained by gathering these numerical forms.

3.3.3 Experimental Setup

The performance of five different algorithms was compared in the experiments. More specifically, the performance of Naïve Bayes, Support Vector Machine, Bidirectional Long Short Term Memory, Bidirectional Long Short Term Memory with Attention, and Bidirectional Encoder Representations from Transformers was compared. For traditional machine learning algorithms, including Naïve Bayes and Support Vector Machine, each can be fed features from either BOW or TF-IDF. In addition, the classification can be done either by a multiclass classifier or four binary classifiers. Each review can be assigned to several classes at once by a multiclass classifier, while each binary classifier decides whether each review belongs to a specific category or not. For deep learning algorithms, different classification strategies (binary or multiclass) could be examined by employing different loss functions. By using sparse categorical cross-entropy, the assignment of reviews can be obtained directly. That can simulate multiclass classification. An array with four elements, each of which predicts if a review is from that class, can simulate the usage of the binary classification strategy by leveraging the binary cross-entropy. All the deep learning models used Adam optimizer and a very small learning rate 3×10^{-5}. Each was trained for 800 epochs, and we selected the one with the minimum validation loss. All the deep learning models used the Adam optimizer and had a very small learning rate of 3×10^{-5}. Each model was trained for 800 epochs, and the one with the minimum validation loss was selected.

3.3.3.1 NB-Based Experiments

According to the different types of features, classifiers were implemented according to different multiclass classification strategies. A NB Multiclass classifier was created by using

naive_bayes.MultinomialNB() in *sklearn* package. For using the binary classifiers for multi-class classification, a method *OneVsRestClassifier*() under the model *sklearn.multiclass* was used, which fitted one classifier for each class. Each class was fitted against other classes for each classifier. This method was implemented using the code *OneVsRestClassifier*(*naive_bayes.MultinomialNB*()). With each feature vectorizer and each NB-based multiclass classification strategy, a machine learning pipeline that automated the workflow of machine learning was built due to the method *pipline*. *Pipeline*() within the *sklearn* package. To illustrate it more clearly, a pipeline that consists of a BOW vectorizer and One-vs-rest multiclass strategy is shown in Figure 3.8. *a_pipline.fit*() was used to fit training samples on target labels. After fitting, the pipelines employed the *predict*() method to make the predictions of the testing samples.

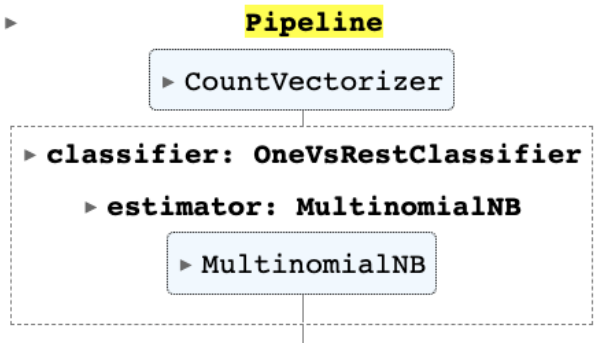

Figure 3.8: Pipeline Consisting of BOW Vectorizer and One-Vs-Rest Multiclass Strategy on NB Model

3.3.3.2 SVM-Based Experiments

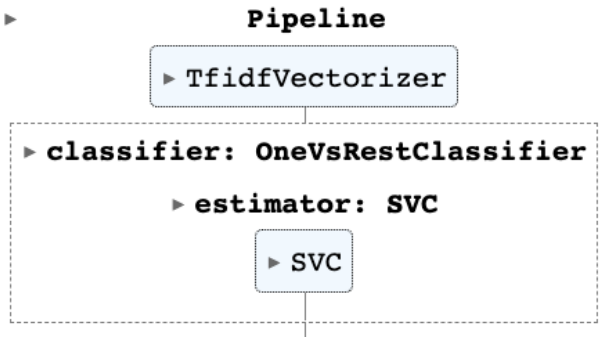

Figure 3.9: Pipeline Consisting of TF-IDF Vectorizer and One-Vs-Rest Multiclass Strategy on SVM Model

SVM-based experiments were quite similar, except the classifiers used were different. They leveraged SVM classifiers instead of NB classifiers. A pipeline that consists of a TF-IDF vectorizer and a "one-vs-rest" multiclass strategy on SVM is shown in Figure 3.9.

3.3.3.3 BiLSTM-Based Experiments

Layer Name	Parameter Name	Parameter Value
embedding_1	input_dim	7767
	output_dim	100
	trainable	True
dropout_4	rate	0.3
bidirectional_2(lstm_2)	units	64
	dropout	0.7
	return_sequences	True
dropout_5	rate	0.3
bidirectional_3(lstm_3)	units	32
	dropout	0.7
dropout_6	rate	0.5
dense_2	units	20
	activation	relu
	kernel_regularizer	l2
dropout_7	rate	0.3
dense_3	units	5
	activation	sigmoid/softmax
	kernel_regularizer	l2
	loss	binary_crossentropy/ sparse_categorical_crossentropy
	optimizer	Adam
	metrics	['accuracy']

Table 3.1: Parameter Setting of Bi-LSTM Models

Bi-LSTM-based Experiments examined the performance of models using different loss functions and activation functions of the output layer (multiclass classification strategies). Sequences attained from Section 3.3.2.2 were used to train or test the models, depending on the dataset transformed into the sequences. With the help of *keras* package, the Bi-LSTM models were built, and their architectures are shown in Figure 3.10. Different multiclass classification strategies were implemented by varying the activation function of the output layer (dense_3) and the corresponding loss function. To achieve the general multiclass classification, the output layer used the softmax activation function, and binary cross entropy was leveraged as the model's loss function. The output layer employed the sigmoid activation function to imple-

ment binary classifiers for multiclass classification. At the same time, the model used sparse categorical cross entropy as the loss function. The parameters set for the models are shown in Table 3.1.

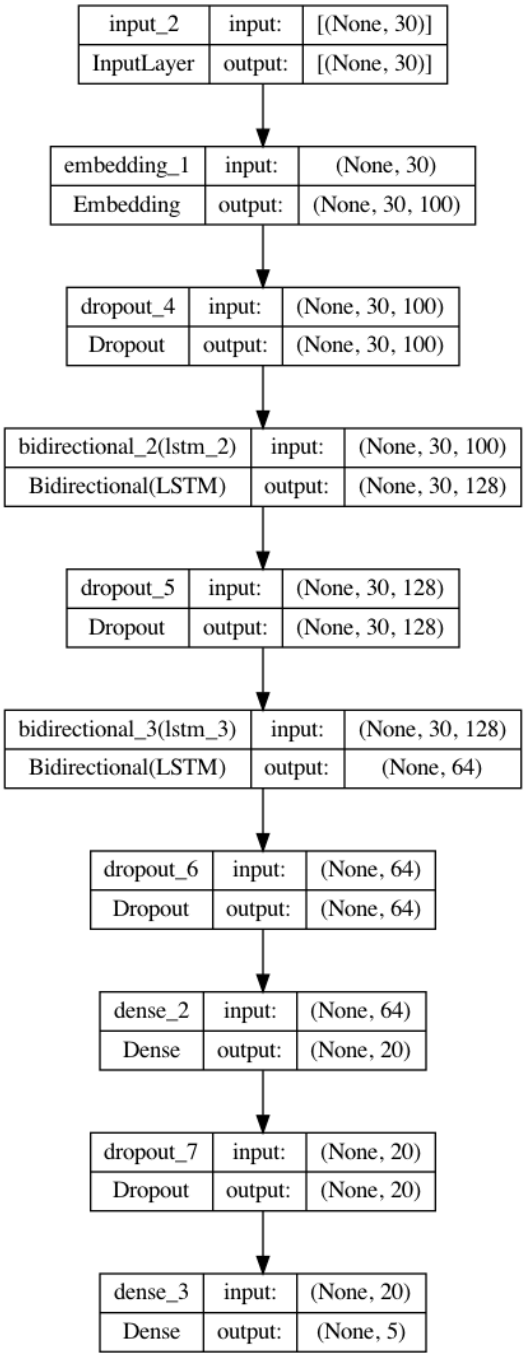

Figure 3.10: Architecture of Bi-LSTM Models

3.3.3.4 BiLSTM-Attention-Based Experiments

Bi-LSTM-Attention-Based experiments did things similar to what has been discussed in the last part, except Bi-LSTM-attention models were utilized. The architecture of each of these models added one more attention layer compared to each Bi-LSTM model, as depicted in Figure 3.11. This layer can give relevant elements more emphasis. To implement the attention layer, the bidirectional_17(lstm_17) states were returned. The backward hidden state and the forward hidden state were combined as the annotation in the concatenate_13 layer. Then, the hidden state of all the time steps and the annotation were passed into the attention layer, defined in Figure 3.12. The parameter setting was illustrated in Table 3.2.

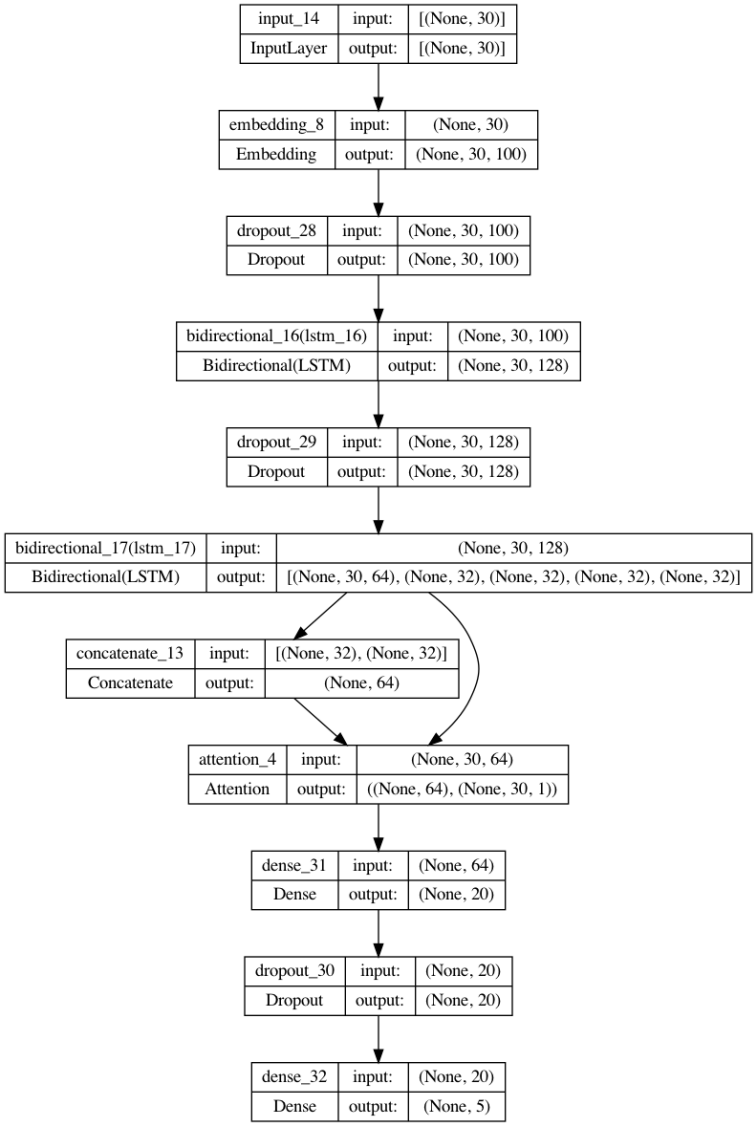

Figure 3.11: Architecture of Bi-LSTM-Attention Models

Layer Name	Parameter Name	Parameter Value
embedding_8	input_dim	7767
	output_dim	100
	trainable	True
dropout_28	rate	0.3
bidirectional_16(lstm_16)	units	64
	dropout	0.7
	return_sequences	True
dropout_29	rate	0.3
bidirectional_17(lstm_17)	units	32
	dropout	0.7
	return_state	True
	return_sequences	True
attention_4	units	10
dense_31	units	20
	activation	relu
	kernel_regularizer	l2
dropout_30	rate	0.3
dense_32	units	5
	activation	sigmoid/softmax
	kernel_regularizer	l2
	loss	binary_crossentropy/ sparse_categorical_crossentropy
	optimizer	Adam
	metrics	['accuracy']

Table 3.2: Parameter Setting of Bi-LSTM-Attention Models

```
#Attention layer class is defined
class Attention(tf.keras.Model):
    def __init__(self, units, **kwargs):
        super(Attention, self).__init__(**kwargs)
        #Intialize the learnable attention parameters Wa, Ua, and va
        self.Wa = tf.keras.layers.Dense(units)
        self.Ua = tf.keras.layers.Dense(units)
        self.va = tf.keras.layers.Dense(1)

    def call(self, features, hidden):
        #Expand the shape of the annotations which has comhined the forward
        #hidden state and the backward hidden state
        hidden = tf.expand_dims(hidden, 1)

        #Calculate the additive attention score
        atten_scores = self.va(tf.nn.tanh(self.Wa(features) + self.Ua(hidden)))

        #The corresponding attention weights are calculated
        atten_weights = tf.nn.softmax(atten_scores, axis=1)

        #The context vector is obtained by obtaining the weightd
        #sum of annotations.
        con_vec = atten_weights * features
        con_vec = tf.reduce_sum(con_vec, axis=1)

        return con_vec, atten_weights
```

Figure 3.12: Definition of Attention Layer

3.3.3.5 BERT-Based Experiments

For BERT-based experiments, the distilBert model, a smaller version of BERT was utilized. It runs 60% faster and uses 40% fewer parameters than Bert-base-uncased while maintaining over 97% of BERT's performance on the GLUE language understanding benchmark [154]. First, the pre-trained BERT tokenizer was utilized to preprocess the samples. By applying the tokenizer to each sentence, indices of elements that constituted the sentence in the vocabulary and attention masks that recorded whether tokens were masked or not were obtained. Then, the models which used pre-trained distilBert ('distilbert-base-uncased') to produce sentence embeddings and a neural network to perform multiclass classification were built. Their architecture is demonstrated in Figure 3.13. Different multiclass classification strategies were still investigated by utilizing these models. Their parameters set for these models are shown in Table 3.3.

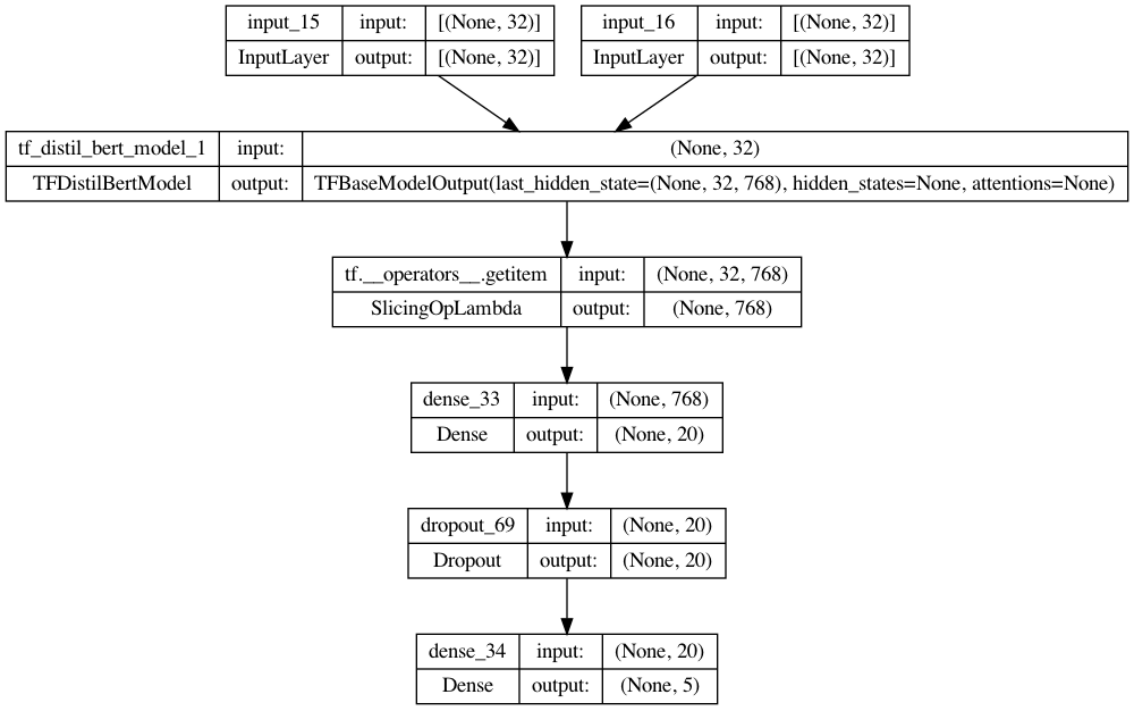

Figure 3.13: Architecture of BERT Models

Layer Name	Parameter Name	Parameter Value
tf_distil_bert_model_1	model_name	distilbert-base-uncased
dense_33	units	20
	activation	relu
	kernel_regularizer	l2

dropout_69	rate	0.5
dense_34	units	5
	activation	sigmoid/softmax
	kernel_regularizer	l2
	loss	binary_crossentropy/ sparse_categorical_crossentropy
	optimizer	Adam
	metrics	['accuracy']

Table 3.3: Parameter Setting of BERT Model

3.3.4 Improvement of Models

As discussed in Section 3.1.4, extra features had been planned to be extracted to improve the models. Three types of features were retrieved, including the compound score for sentiment, the score of subjectivity, and the length of sentences. Compound scores was gotten by leverage sentiment intensity analyzer in *vaderSentiment.vaderSentiment* module. It had the method *polarity_scores* to obtain a sentiment dictionary. Compound scores were attained from these dictionaries. *textblob.TextBlob* was used to retrieve subjectiveness scores. A TextBlob instance was created for each sample. The subjective score was gotten via its *sentiment.polarity* attribute. In the end, more diverse datasets, which contained more features, were generated. An example can be shown in Figure 3.14.As discussed in Section 2.6.2.1, forward feature selection was leveraged to select a set of appropriate features for each model. It was found that models trained with text and compound scores could lead to the best accuracy. Hence, the following discussion on the models' modification will focus on how to make models receive these two features as inputs.

3.3.4.1 Traditional Machine Learning Models

For traditional machine learning models, the machine learning pipelines were modified. I first defined two selectors to select a textual data column and a numerical data column separately from a data frame. Their definitions are shown in Figure 3.15. Then, I implemented a pipeline for the textual feature. The first step was to select the 'clean_text' column in the data frame. The second step was to extract textual features using some techniques (BOW/TF-IDF).After that, a pipeline for the numerical features was generated. The first step was to use *NumberSelector* to select the 'polarity' column. As NB did not receive negative inputs, these were transformed

Galaxy S4 e:	RT	galaxy s4 ex	0.55	0.675	9
. Nice	RT	nice	0.6	1	2
Check Depo	PD	check depos	0	0	4
Need impro	FR	need improv	0	0	2
Great Ap.	RT	great ap	0.8	0.75	2
This app is	FR	app great to	0.520833	0.683333	105
So slow The	UE	slow conten	-0.08	0.51	28
Can't custor	FR	cant custom	-0.1	0.4	3
love this ap	UE	love app sup	0.422222	0.7	9
I can't find t	PD	cant find no	0	0	9
"Love that p	UE	love place st	0.416667	0.633333	12
PLEASE PLE	FR	please pleas	0	0	17
What happe	PD	happened u	0.166667	0.383333	27
Great progr	RT	great progra	0.8	0.75	2
love this gar	UE	love game pl	0.125	0.35	23
Simply the b	RT	simply best	1	0.3	3
Downhill, ev	UE	downhill eve	0.069444	0.430556	50
Can't get sin	PD	cant get sing	0.12619	0.279762	15
Hate the iPa	UE	hate ipad lay	0.016667	0.458333	50
Hey! Where	FR	hey ukrainia	0	0	18
Addicting! T	FR	addicting be	0.078571	0.392857	57
deposit stop	PD	deposit stop	0	0	3
Please fix!	PD	please fix	0	0	2
Carol Can't f	UE	carol cant fi	0.136364	0.454545	6
The only thi	FR	thing need c	-0.125	0.55	18
fix it please.	PD	fix please	0	0	3
Great.	RT	great	0.8	0.75	1
"No month	FR	month caler	-0.3	0.5	7
Frequent cra	PD	frequent cra	0.1	0.3	2
Loveeeee Al	RT	loveeeee abs	-0.1	0.75	6
It's a really g	FR	really good k	0.7	0.6	15
It needs to b	FR	need like tab	0	0	8
Potentially N	UE	potentially r	0.318182	0.477273	42
"No visual c	FR	visual cut le	0.125	0.125	15
BOOO ADDS	UE	booo add ar	-0.22813	0.571875	30
It s a great p	PD	great progra	0.2	0.345833	20
This is very	RT	great workin	0.65	0.625	7
AWESOME M	RT	awesome go	0.366667	0.598148	34
Please inclu	FR	please inclu	-0.1	0.4	9
like candor r	UE	like candor r	0.446667	0.506667	43
go to app su	UE	go app supe	0.334091	0.69697	15
but now cra	PD	crash every	0	0.5	7
Love TripAd	UE	love tripadvi	0.55	0.75	13
good ap	RT	good ap	0.7	0.6	2
Very good V	UE	good good r	0.666667	0.733333	8
"""There wa	PD	error conne	0.416667	0.5	17
Please fix!	PD	please fix	0	0	2
Please show	FR	please show	-0.05	0.233333	8
has lot of w	UE	lot workout	0.4	0.8	8
Warning do	PD	warning upc	0	0.283333	18

Figure 3.14: Dataset with Additional Features

into values ranging from 0 to 10 by using $FunctionTransformer(transformation)$. The other function called "transformation" returned $5 * input + 5$. To rejoin the textual and numerical pipeline, *FeatureUnison* was utilized. A pipeline was also generated for the union of features. The final pipeline was used for classification. The first step was to enter the features' union pipeline. The second step was a multiclass classification strategy. A improved pipeline consisting of TF-IDF vectorizer and one-vs-rest multiclass strategy on NB model is shown in Figure 3.16. A improved pipeline consisting of TF-IDF vectorizer and one-vs-rest multiclass strategy on SVM model is shown in Figure 3.17.As discussed in Section 3.1.4, extra features had been planned to be extracted to improve the models. Three features were retrieved, including the compound score for sentiment, the subjectivity score, and the sentences' length. Compound scores was gotten by leverage sentiment intensity analyzer in *vaderSentiment.vaderSentiment* module. It used the method *polarity_scores* to obtain a sentiment dictionary. Compound scores were attained from these dictionaries. *textblob.TextBlob* was used to retrieve subjectiveness scores. A TextBlob instance was created for each sample. The subjective score was gotten via its *sentiment.subjectivity* attribute.

```
from sklearn.base import BaseEstimator, TransformerMixin

class TextSelector(BaseEstimator, TransformerMixin):
    """
    Transformer to select a single column from the data
    frame to perform additional transformations on
    textual data
    """
    def __init__(self, key):
        self.key = key

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        return X[self.key]

class NumberSelector(BaseEstimator, TransformerMixin):
    """
    Transformer to select a single column from the data
    frame to perform additional transformations on numerical
    data
    """
    def __init__(self, key):
        self.key = key

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        return X[[self.key]]
```

Figure 3.15: Definitions of Text Selector and Number Selector

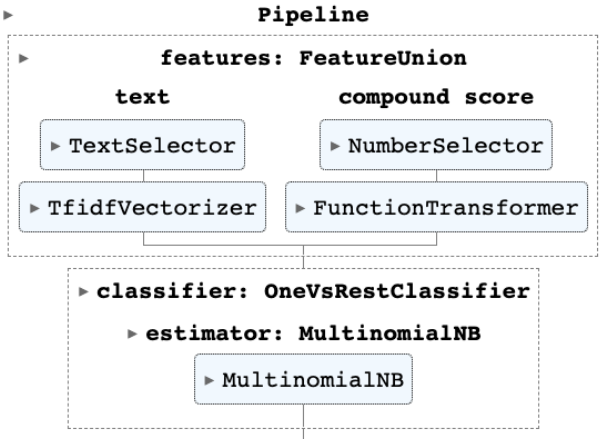

Figure 3.16: Modified Pipeline Consisting of TF-IDF Vectorizer and One-Vs-Rest Multiclass Strategy on NB Model

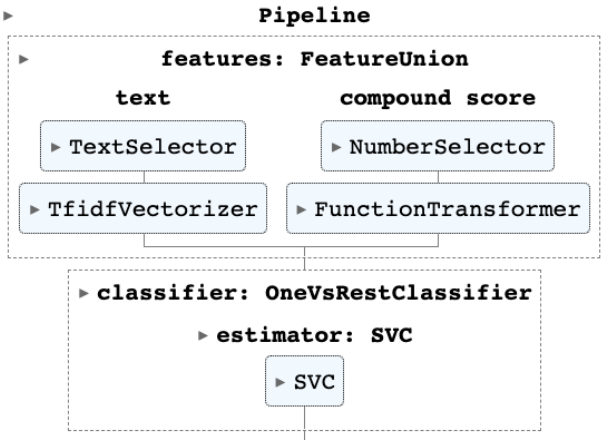

Figure 3.17: Modified Pipeline Consisting of TF-IDF Vectorizer and One-Vs-Rest Multiclass Strategy on SVM Model

3.3.4.2 Deep Learning Models

Each deep learning model generated an extra input layer to receive the additional compound scores. The output of the second Bi-LSTM layer, which can be regarded as each review's encoding, was fed to a dropout layer to reduce overfitting in Bi-LSTM models. Then, the output of the dropout layer was concatenated to the newly added input, which was then entered into the artificial neural network for further classification. Its architecture is shown in Figure 3.18.Similarly, the output of the attention layer was combined with input in Bi-LSTM-attention models, while the input was merged to the distilBert layer's output in BERT models. Their architectures are shown in Figure 3.19 and Figure 3.20 respectively.

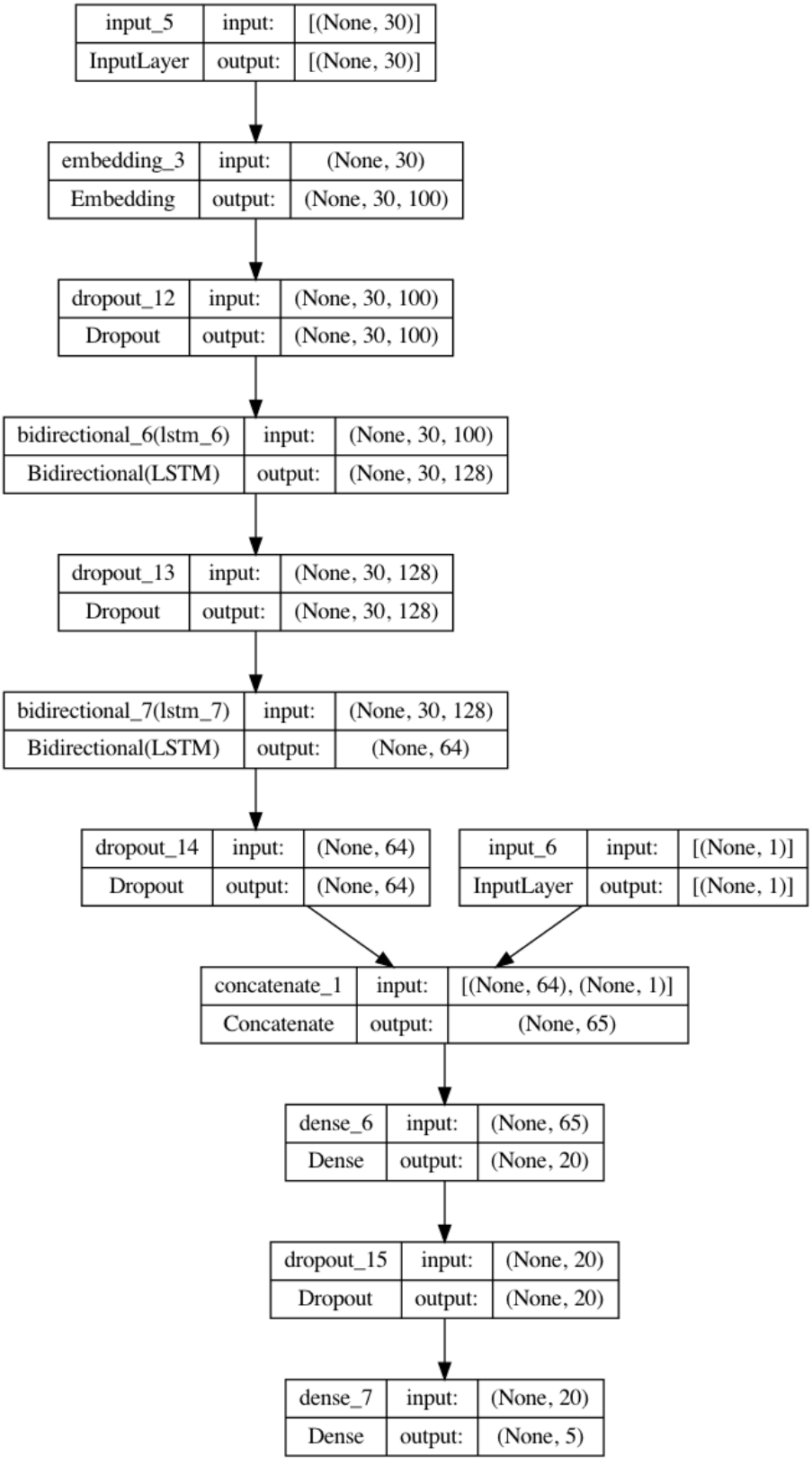

Figure 3.18: Modified Bi-LSTM Models Architecture

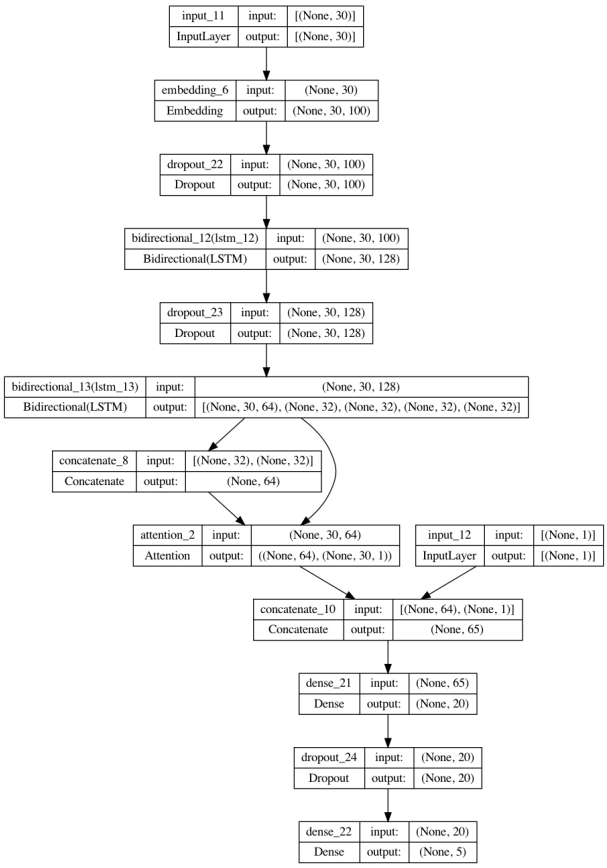

Figure 3.19: Modified Bi-LSTM-Attention Models Architecture

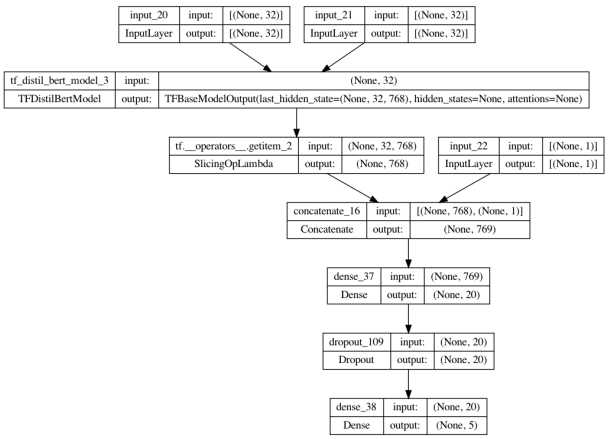

Figure 3.20: Modified BERT Models Architecture

3.4 Summary

This chapter has discussed the design and implementation of experiments in this project. First of all, the research methodology was designed based on the NLP pipeline, providing overall instructions for this project. Then, each phase (data preparation, processing, classification, and evaluation) was designed according to the proposed methodology. The details of each step within each phase were included specifically. After that, all the utilized tools and libraries were illustrated, involving Jupyter Notebook, Numpy, Pandas, TestAttack, NLTK, VaderSentiment, TextBlob, Sklearn, TensorFlow, and Git. Lastly, the implementation of each phase, including all the different kinds of model-based experiments, was described. In particular, extracting additional features and modifying the models are demonstrated to improve the models' performance.

Chapter 4

Results and Discussion

In this chapter, each type of experiment's results will be first discussed to assess the performance of classifiers. The effect of the additional feature, compound score, is then illustrated. Furthermore, the insights within the results are specified.

4.1 NB-Based Experiments

The results of acrshortnb-based experiments are shown in Table 4.1. It can be found that the highest F1 score of bug report was achieved by using multiple binary classifiers with BOW. That was also a case for rating. Multiclass classifiers with BOW and binary classifiers with TF-IDF performed the best in the scope of feature requests and user experiences, respectively. No model achieved the best results for all four types of classification. All of these models accomplished their own best performance (all the F1 scores are above 0.8) on the classification of bug reports. At the same time, all of them performed the worst (the highest F1 score is 0.61) on user experience classification. Hence, it can be estimated that the bug report is the easiest to classify, while the user experience classification is the hardest. In addition, using binary classifiers for multiclassification might be better than employing direct multiclassification since F1 scores or accuracies of binary ones were always higher than or equal to their multiclass counterparts. It can be easily discovered that using the BOW features was always better than leveraging TF-IDF features. Hence, using TF-IDF technique to extract features is better when the NB algorithm is utilised. Finally, the accuracies and F1 scores for NB-based models were around 0.705, which was fairly ordinary.

Type of Classifier	Technique	Bug Reports			Feature Requests			User Experiences			Rating			Overall	
		P	R	F1	P	R	F1	P	R	F1	P	R	F1	A	F1
Naive Bayes															
Multiclass	BOW	0.795	0.857	0.825	0.768	0.701	0.733	0.591	0.572	0.582	0.695	0.723	0.709	0.713	0.712
Binary	BOW	0.812	0.851	0.831	0.747	0.714	0.730	0.614	0.554	0.583	0.675	0.738	0.705	0.714	0.712
Multiclass	TF-IDF	0.786	0.834	0.809	0.733	0.660	0.695	0.556	0.598	0.576	0.730	0.702	0.716	0.699	0.699
Binary	TF-IDF	0.798	0.830	0.814	0.694	0.692	0.693	0.605	0.614	0.610	0.734	0.694	0.713	0.708	0.707

Table 4.1: Results of NB-Based Experiments

4.2 SVM-Based Experiments

Leveraging binary classifiers for multiclass classification with BOW achieved the best results on the classifications of the bug reports, user experiences, and ratings. General multiclass classification strategy with BOW obtained the best outcomes on the feature requests classification according to the F1 scores. All these models also got the best outcomes on bug report classification, and user experience was also the hardest to classify by taking precision, recall, and F1 scores into account. It can be concluded that bug reports were the easiest to classify and user experience was the hardest to classify for SVM. Apart from the classification of user experience, the models with the BOW technique were always better than the corresponding models with TF-IDF. Apart from the case of feature request classification, the binary classifier strategy always performed better than the multiclass classifier strategy from the aspect of the F1 score. Finally, the overall accuracies and F1 scores were around 0.725, which were better than those of NB.

Type of Classifier	Technique	Bug Reports			Feature Requests			User Experiences			Rating			Overall	
		P	R	F1	P	R	F1	P	R	F1	P	R	F1	A	F1
Support Vector Machine															
Multiclass	BOW	0.834	0.861	0.847	0.773	0.750	0.761	0.606	0.571	0.588	0.693	0.731	0.711	0.728	0.727
Binary	BOW	0.860	0.838	0.849	0.749	0.772	0.760	0.607	0.634	0.621	0.732	0.696	0.713	0.735	0.736
Multiclass	TF-IDF	0.832	0.832	0.832	0.747	0.733	0.740	0.599	0.591	0.595	0.677	0.699	0.688	0.714	0.714
Binary	TF-IDF	0.859	0.821	0.840	0.772	0.730	0.750	0.588	0.648	0.616	0.696	0.693	0.694	0.723	0.725

Table 4.2: Results of SVM-Based Experiments

4.3 BiLSTM-Based Experiments

For Bi-LSTM models, using direct multiclass classification strategy was worse than using binary classification strategy for multiclass classification depending on the overall accuracy and F1 score. Besides, Bi-LSTM-based models performed the best on bug report classification

based on the three evaluation metrics (precision, recall, and F1 score). Although the performance of user experience classification improved compared to the traditional machine learning models, their outcomes were still the worst. These models' accuracies were around 0.768, and F1 scores were around 0.760, which were better than traditional machine learning models.

Type of Classifier	Technique	Bug Reports			Feature Requests			User Experiences			Rating			Overall	
		P	R	F1	P	R	F1	P	R	F1	P	R	F1	A	F1
BiLSTM															
Multiclass	Word Embedding	0.824	0.871	0.847	0.857	0.672	0.753	0.609	0.789	0.687	0.838	0.722	0.776	0.764	0.766
Binary	Word Embedding	0.878	0.857	0.867	0.835	0.691	0.756	0.654	0.728	0.684	0.759	0.811	0.784	0.772	0.773

Table 4.3: Results of Bi-LSTM-Based Experiments

4.4 BiLSTM-Attention-Based Experiments

By using Bi-LSTM-attention models, the binary classification strategy was better than the multiclass strategy since the binary classification strategy's overall accuracy and F1 score was higher. Bug report reviews were the easiest ones to classify as their precision, recall, and F1 were highest among these four types of classification. It was still the hardest to classify user experience reviews for these models. The performance of this type of model and that of Bi-LSTM were tied in terms of overall accuracy and F1 score.

Type of Classifier	Technique	Bug Reports			Feature Requests			User Experiences			Rating			Overall	
		P	R	F1	P	R	F1	P	R	F1	P	R	F1	A	F1
BiLSTM-Attention															
Multiclass	Word Embedding	0.872	0.868	0.870	0.825	0.691	0.752	0.645	0.700	0.671	0.753	0.841	0.783	0.768	0.769
Binary	Word Embedding	0.831	0.889	0.859	0.852	0.678	0.755	0.633	0.763	0.682	0.812	0.753	0.782	0.771	0.772

Table 4.4: Result of Bi-LSTM-Attention-Based Experiments

4.5 BERT-Based Experiments

The findings of bug report classification, user experience classification, and the effect of different classification strategies were also valid for BERT. It made significant progress on classifying reviews of user experience, which were increased by about 0.09 compared to Bi-LSTM-based models. Besides, feature request and rating classification were also improved to different extents. Finally, BERT-based models had a huge advantage in almost all types of classification,

excluding bug report categorization, over other models made. They got a similar performance on bug report categorization compared to Bi-LSTM-attention-based models

Type of Classifier	Technique	Bug Reports			Feature Requests			User Experiences			Rating			Overall	
		P	R	F1	P	R	F1	P	R	F1	P	R	F1	A	F1
BERT															
Multiclass	Word Embedding	0.873	0.853	0.862	0.804	0.809	0.806	0.784	0.762	0.772	0.851	0.822	0.826	0.818	0.819
Binary	Word Embedding	0.877	0.862	0.869	0.806	0.809	0.807	0.786	0.770	0.778	0.837	0.832	0.834	0.821	0.822

Table 4.5: Result of BERT-Based Experiments

4.6 Effect of the Additional Feature

As the additional feature had a similar effect on different models, only the best set of models (BERT) with the highest overall F1 score will be described. Except for the bug reports, the F1 score for all other types of classification was more or less raised, thus making BERT-based models perform the best. The /acrshortbert model with the binary classification strategy for multiclass classification achieved the highest overall accuracy and F1 score, which were 0.825 and 0.826, respectively.

Type of Classifier	Technique	Bug Reports			Feature Requests			User Experiences			Rating			Overall	
		P	R	F1	P	R	F1	P	R	F1	P	R	F1	A	F1
BERT															
Multiclass	Word Embedding	0.873	0.824	0.848	0.811	0.831	0.821	0.792	0.763	0.777	0.861	0.838	0.849	0.823	0.824
Binary	Word Embedding	0.872	0.823	0.847	0.822	0.836	0.829	0.801	0.768	0.784	0.872	0.817	0.844	0.825	0.826

Table 4.6: Result of improved BERT-Based Experiments

4.7 Research Findings

For traditional machine learning models, models with SVM algorithms performed better than those with NB algorithms. Employing the BOW technique for feature extraction was usually better than leveraging the TF-IDF technique. In our experiments, deep learning models always got higher F1 scores and accuracies than traditional machine learning models; thus, it can be concluded that employing deep learning algorithms was better than traditional machine learning algorithms in the scope of the project. Based on the results of experiments, it can also be investigated that using a binary classification strategy (implementing binary classifiers

physically or logically) for multiclass classification is better than employing a direct multiclass strategy. For almost all the models, bug report reviews were the easiest to classify. In contrast, all the classifiers performed their worst on user experience classification. That might be because user experience reviews have more kinds of patterns than other ones. The corpus of this project may not be enough to find all the patterns. The user experience classification improved significantly by utilizing BERT models. That might be due to the fact that the BERT layer, which was pre-trained with large corpora of data, aided in finding these patterns. With the additional feature compound score, the BERT model utilizing logical binary classifiers for multiclass classification achieved the highest accuracy and F1 score, which are 0.825 and 0.826, respectively.

4.8 Summary

This chapter presents the results of experiments discussed in 3.3.3. The primary discovery was that BERT-based models had the best performance compared to other models. The effect of the additional feature compound score was explained by taking the additional feature as an extra input to these models. Finally, the research findings were illustrated and analyzed, including the influence of types of machine learning algorithms, different feature engineering techniques, and different classification strategies.

Chapter 5

Conclusion

The dissertation will be concluded with three major sections: achievements, limitations, and future work. What has been achieved in this project will be illustrated in the achievements section. Then, some potential risks that limit models' performance will be discussed in the limitation section. Lastly, the future work section demonstrates what is planned to be done to improve these models in the future.

5.1 Achievements

I will demonstrate the achievements of three aspects: learning and research, design and implementation, and evaluation in accordance with the project's objectives discussed in Section 1.2. A NLP pipeline was learned, including seven steps. In particular, details about some techniques of text preprocessing, text augmentation, and feature engineering were discussed in Section 2.4, Section 2.3.2, and Section 2.5 respectively. These techniques were all related to NLP. It can be proved that traditional machine learning techniques were fully understood in Section 2.7.2. Hence, the first objective of this project was achieved. Some deep learning techniques such as ANN, RNN, Bi-LSTM, attention, and BERT were introduced in Section 2.7.3. Therefore, the objective of learning deep learning techniques was accomplished. Evaluation metrics were discussed in Section 2.7.4.2, thus completing the objective of evaluation metrics research. Section 3.1.5 specified the details about evaluation metrics selection and why they were selected, proving the third objective's completion. The related publications were discussed in Section 2.8, and their main weaknesses were clarified in Section 2.9. That means the fourth objective

was completely finished. Therefore, it can be concluded that the achievement associated with the learning and research aspects has been achieved.

From the perspective of design and implementation, Section 3.1 designed the research methodology and experiments, which showed the completion of the fifth objective. Data collection and labeling implementation were described in 3.3.1, thus obtaining a labeled dataset with 12000 entries. Therefore, the sixth objective was achieved. The accomplishment of the seventh objective was shown in Section 3.3.2, including text preprocessing and feature engineering. The implementation of all the models was depicted in Section 3.3.3. As a result, the achievement related to design and implementation was obtained.

Experiments were evaluated, and comparisons were made between different models in section 4. In conclusion, all the objectives proposed were accomplished. As the project's aim is to develop automatic classification classifiers for app review classification, Five different kinds of classifiers including NB classifiers, SVM classifiers, Bi-LSTM classifiers, Bi-LSTM-Attention classifiers and BERT classifiers were designed and implemented. A BERT classifier achieved the best performance among them. Its F1 score and accuracy were 0.826 and 0.825, respectively.

5.2 Limitations

Even though the data was labeled utilizing the strategy of cross-checking, it was possible that some data were mislabeled, making classification unreliable. Besides, low-quality reviews that made mistakes in spelling or were even meaningless text cannot totally be prevented in the dataset, which also decreased the performance of the classifiers. Besides, as labeling data was quite a time-consuming task, there was not much time for employing strategic hyperparameter tuning approaches such as grid search and random search, consequently harming models' performance. In the end, the additional feature, compound score, improved the models slightly. That might be caused by the inaccuracy of the rule-based sentiment analyzer.

5.3 Future Work

There are already a lot of datasets for sentiment analysis on the Internet. Hence, a model will be built to generate more accurate compound scores based on these datasets. Other features, such

as location-based features, might also improve the performance of classifiers. These data will be collected and utilized to train the classifiers in the future. Some hyperparameter tuning techniques, such as grid search, will be employed in the future. Finally, other popular algorithms, such as Text-CNN are going to be attempted in the future.

Bibliography

[1] Statista. 2022. *Biggest app stores in the world 2021*. 2022. URL: %7Bhttps://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/%7D (visited on 04/19/2022).

[2] MobileMarketing. *Top 5 Reasons Showing the Importance of Ratings and Reviews for Your Mobile App*. 2021. URL: %7Bhttps://www.mobileappdaily.com/importance-of-mobile-app-reviews%7D (visited on 04/19/2022).

[3] ReviewTrackers. *Online Reviews Statistics and Trends: A 2022 Report by ReviewTrackers*. 2022. URL: %7Bhttps://www.reviewtrackers.com/reports/online-reviews-survey/%7D (visited on 04/19/2022).

[4] Anthony Finkelstein et al. *App Store Analysis: Mining App Stores for Relationships between Customer, Business and Technical Characteristics*. 2014.

[5] Leonard Hoon et al. "An Analysis of the Mobile App Review Landscape: Trends and Implications". In: 2013.

[6] Dennis Pagano and W. Maalej. "User feedback in the appstore: An empirical study". In: *2013 21st IEEE International Requirements Engineering Conference (RE)* (2013), pp. 125–134.

[7] W. Maalej et al. "On the automatic classification of app reviews". In: *Requirements Engineering* 21 (2016), pp. 311–331.

[8] Ben Lutkevich and Ed Burns. *What is natural language processing? an introduction to NLP*. Mar. 2021. URL: https://www.techtarget.com/searchenterpriseai/definition/natural-language-processing-NLP.

[9] IBM Cloud Education. *What is natural language processing?* URL: https://www.ibm.com/cloud/learn/natural-language-processing.

[10] ProjectPro. *Machine learning NLP text classification algorithms and Models*. June 2022. URL: https://www.projectpro.io/article/machine-learning-nlp-text-classification-algorithms-and-models/523#mcetoc_1fle0r0755.

[11] *An end to end guide on NLP pipeline*. June 2022. URL: https://www.analyticsvidhya.com/blog/2022/06/an-end-to-end-guide-on-nlp-pipeline/.

[12] *Python notes for Linguistics*. URL: https://alvinntnu.github.io/python-notes/nlp/nlp-pipeline.html.

[13] Patrik Hrkút et al. "Data collection for Natural Language Processing Systems". In: *Communications in Computer and Information Science* (2020), pp. 60–70. DOI: 10.1007/978-981-15-3380-8_6.

[14] CrazyTechie. *Step by step explanation of NLP Data Pipeline*. Apr. 2022. URL: https://www.crazytechie.com/2022/01/end-end-NLP-pipeline-explained.html#4.

[15] *Must known techniques for text preprocessing in NLP*. Aug. 2022. URL: https://www.analyticsvidhya.com/blog/2021/06/must-known-techniques-for-text-preprocessing-in-nlp/.

[16] Vijaysinh Lendave. *Complete tutorial on text preprocessing in NLP*. Oct. 2021. URL: https://analyticsindiamag.com/complete-tutorial-on-text-preprocessing-in-nlp/.

[17] Tapas Das. *How to perform feature engineering from text data*. Feb. 2022. URL: https://betterprogramming.pub/beginners-to-advanced-feature-engineering-from-text-data-c228047a4813.

[18] *What is NLP modeling? 1 process for active learning*. Aug. 2022. URL: https://nlpsure.com/what-is-nlp-modeling/.

[19] *What is model evaluation?: Domino data science dictionary*. URL: https://www.dominodatalab.com/data-science-dictionary/model-evaluation.

[20] Joshgun Guliyev. *Resampling methods in machine learning: Cross-validation*. Jan. 2021. URL: https://medium.com/analytics-vidhya/resampling-methods-in-machine-learning-cross-validation-677485fa1b4d.

[21] Elton. *The NLP pipeline*. Dec. 2021. URL: https://pythonwife.com/the-nlp-pipeline/.

[22] Kurtis Pykes. *Tips & Tricks: Augmenting data for NLP tasks*. Apr. 2021. URL: https://towardsdatascience.com/tips-tricks-augmenting-data-for-nlp-tasks-983e33ad55a7.

[23] Jason Brownlee. *A Gentle Introduction to Imbalanced Classification*. 2019. URL: %7Bhttps://machinelearningmastery.com/what-is-imbalanced-classification/%7D (visited on 04/22/2022).

[24] Saikat Mazumder. *5 Techniques to Handle Imbalanced Data For a Classification Problem*. 2021. URL: %7Bhttps://www.analyticsvidhya.com/blog/2021/06/5-techniques-to-handle-imbalanced-data-for-a-classification-problem/%7D (visited on 04/22/2022).

[25] *Text data augmentation in natural language processing with Texattack*. Feb. 2022. URL: https://www.analyticsvidhya.com/blog/2022/02/text-data-augmentation-in-natural-language-processing-with-texattack/.

[26] Raj Sangani. *Powerful text augmentation using NLPAUG !* June 2021. URL: https://towardsdatascience.com/powerful-text-augmentation-using-nlpaug-5851099b4e97.

[27] Saurabhk. *5 data augmentation techniques for text classification*. Nov. 2020. URL: https://saurabhk30.medium.com/5-data-augmentation-techniques-for-text-classification-d14f6d8bd6aa.

[28] Pema Grg. *NLP data augmentation*. Feb. 2022. URL: https://pemagrg.medium.com/nlp-data-augmentation-a346479b295f.

[29] Vladimir Ilievski. *The delicacy of data augmentation in Natural Language Processing (NLP)*. July 2020. URL: https://medium.com/ideas-at-igenius/the-delicacy-of-data-augmentation-in-natural-language-processing-nlp-2ef07e9ad1c0.

[30] Amit Chaudhary. *A visual survey of data augmentation in NLP*. Oct. 2020. URL: https://amitness.com/2020/05/data-augmentation-for-nlp/#2-back-translation.

[31] Amit Chaudhary. *Back translation for text augmentation with Google Sheets*. Sept. 2020. URL: https://amitness.com/2020/02/back-translation-in-google-sheets/.

[32] Jiahao Weng. *NLP text preprocessing: A practical guide and template*. Jan. 2021. URL: https://towardsdatascience.com/nlp-text-preprocessing-a-practical-guide-and-template-d80874676e79.

[33] S. Vijayarani, J. Ilamathi, and S. Nithya. "Preprocessing Techniques for Text Mining-An Overview Dr". In: 2015.

[34] Ronen Feldman, James Sanger, et al. *The text mining handbook: advanced approaches in analyzing unstructured data*. Cambridge university press, 2007.

[35] Harshith. *Text preprocessing in Natural Language Processing using python*. July 2022. URL: https://towardsdatascience.com/text-preprocessing-in-natural-language-processing-using-python-6113ff5decd8.

[36] Chetna Khanna. *Text pre-processing: Stop words removal using different libraries*. Feb. 2021. URL: https://towardsdatascience.com/text-pre-processing-stop-words-removal-using-different-libraries-f20bac19929a.

[37] Sunny Srinidhi. *Stemming of words in natural language processing, what is it?* May 2020. URL: https://towardsdatascience.com/stemming-of-words-in-natural-language-processing-what-is-it-41a33e8996e2.

[38] Eshban Suleman. *Feature engineering in NLP*. Feb. 2022. URL: https://eshban9492.medium.com/feature-engineering-in-nlp-7d89bf47f7ae.

[39] *Bow model and TF-IDF for creating feature from text*. Dec. 2020. URL: https://www.analyticsvidhya.com/blog/2020/02/quick-introduction-bag-of-words-bow-tf-idf/.

[40] Jason Brownlee. *A gentle introduction to the bag-of-words model*. Aug. 2019. URL: https://machinelearningmastery.com/gentle-introduction-bag-words-model/.

[41] *Supervised and unsupervised learning*. July 2022. URL: https://www.geeksforgeeks.org/supervised-unsupervised-learning/.

[42] Hariom Gautam. *Word embedding*. Mar. 2020. URL: https://medium.com/@hari4om/word-embedding-d816f643140.

[43] Lisa A. Chalaguine. *NLP 101 1/3-feature engineering and word embeddings*. June 2021. URL: https://towardsdatascience.com/nlp-101-%5C%E2%5C%85%5C%93-feature-engineering-and-word-embeddings-f10dffd67bb0.

[44] Shachi Kaul. *Text feature extraction (3/3): Word embeddings model*. Jan. 2022. URL: https://medium.com/geekculture/text-feature-extraction-3-3-word-embeddings-model-e98f3d270dce.

[45] Jason Brownlee. *How to choose a feature selection method for machine learning*. Aug. 2020. URL: https://machinelearningmastery.com/feature-selection-with-real-and-categorical-data/.

[46] *Feature selection methods: Machine learning*. Oct. 2020. URL: https://www.analyticsvidhya.com/blog/2016/12/introduction-to-feature-selection-methods-with-an-example-or-how-to-select-the-right-variables/.

[47] Rahil Shaikh. *Feature selection techniques in machine learning with python*. Oct. 2018. URL: https://towardsdatascience.com/feature-selection-techniques-in-machine-learning-with-python-f24e7da3f36e.

[48] *General - Feature Selection*. URL: https://h2o.ai/wiki/feature-selection/.

[49] Sampath Kumar Gajawada. *Chi-square test for feature selection in machine learning*. Oct. 2019. URL: https://towardsdatascience.com/chi-square-test-for-feature-selection-in-machine-learning-206b1f0b8223.

[50] *Feature selection using Wrapper method - python implementation*. June 2022. URL: https://www.analyticsvidhya.com/blog/2020/10/a-comprehensive-guide-to-feature-selection-using-wrapper-methods-in-python/.

[51] *Forward feature selection: Implementation of Forward Feature Selection*. Apr. 2021. URL: https://www.analyticsvidhya.com/blog/2021/04/forward-feature-selection-and-its-implementation/.

[52] Will Koehrsen. *Modeling: Teaching a machine learning algorithm to deliver business value*. Nov. 2018. URL: https://towardsdatascience.com/modeling-teaching-a-machine-learning-algorithm-to-deliver-business-value-ad0205ca4c86.

[53] Simplilearn. *The Complete Guide to Machine Learning Steps*. July 2022. URL: https://www.simplilearn.com/tutorials/machine-learning-tutorial/machine-learning-steps.

[54] David Weedmark. *Machine learning model training: What it is and why it's important*. July 2022. URL: https://www.dominodatalab.com/blog/what-is-machine-learning-model-training.

[55] IBM Cloud Education. *What is supervised learning?* URL: https://www.ibm.com/cloud/learn/supervised-learning#:~:text=Supervised%5C%20learning%5C%2C%5C%20also%5C%20known%5C%20as,data%5C%20or%5C%20predict%5C%20outcomes%5C%20accurately..

[56] *Understanding TF-IDF (term frequency-inverse document frequency)*. July 2022. URL: https://www.geeksforgeeks.org/understanding-tf-idf-term-frequency-inverse-document-frequency/.

[57] Grzegorz Mrukwa and Netguru. *Supervised and unsupervised machine learning - types of ML*. June 2022. URL: https://www.netguru.com/blog/supervised-machine-learning.

[58] Telefonica Tech ayuda a sus clientes en su proceso de transformación digital unificando las capacidades de IoT AI of Things Desde la unidad de negocio de AI of Things. *The 2 types of learning in machine learning: Supervised and unsupervised*. June 2021. URL: https://business.blogthinkbig.com/the-2-types-of-learning-in-machine-learning-supervised-and-unsupervised/.

[59] Sambit Mahapatra. *Why deep learning over traditional machine learning?* Jan. 2019. URL: https://towardsdatascience.com/why-deep-learning-is-needed-over-traditional-machine-learning-1b6a99177063.

[60] *Deep Learning vs. machine learning – what's the difference?* URL: https://levity.ai/blog/difference-machine-learning-deep-learning.

[61] *Traditional machine learning methods vs deep learning in retail*. June 2022. URL: https://cognira.com/blog/retail-ai-ml-data-science/retailers-deep-learning-and-traditional-machine-learning/.

[62] Destin Gong. *Top 6 machine learning algorithms for classification*. July 2022. URL: https://towardsdatascience.com/top-machine-learning-algorithms-for-classification-2197870ff501.

[63] Rohith Gandhi. *Support Vector Machine - introduction to machine learning algorithms*. July 2018. URL: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47.

[64] Simplilearn. *Naive Bayes classifier - machine learning [updated]: Simplilearn*. Aug. 2022. URL: https://www.simplilearn.com/tutorials/machine-learning-tutorial/naive-bayes-classifier.

[65] Rohith Gandhi. *Naive Bayes classifier*. May 2018. URL: https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c.

[66] Roberto Basili, Alessandro Moschitti, and Maria Teresa Pazienza. "NLP-driven IR: Evaluating Performances over a Text Classification task". In: *IJCAI*. 2001, pp. 1286–1294.

[67] *SVM: Support Vector Machine Algorithm in machine learning*. Aug. 2021. URL: https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/.

[68] Lijuan Cai and Thomas Hofmann. "Hierarchical Document Categorization with Support Vector Machines". In: CIKM '04. Washington, D.C., USA: Association for Computing Machinery, 2004, pp. 78–87. ISBN: 1581138741. DOI: 10.1145/1031171.1031186. URL: https://doi.org/10.1145/1031171.1031186.

[69] Ingo Steinwart and Andreas Christmann. *Support vector machines*. Springer Science & Business Media, 2008.

[70] E. Alpaydin. *Introduction to Machine Learning*. Adaptive computation and machine learning. MIT Press, 2004. ISBN: 9780262012119. URL: https://books.google.com.hk/books?id=1k0%5C_-WroiqEC.

[71] Statinfer. *204.6.8 SVM : Advantages disadvantages and applications*. Apr. 2019. URL: https://statinfer.com/204-6-8-svm-advantages-disadvantages-applications/.

[72] Posted by Seb, About Author Seb, and Related Posts. *An introduction to neural network loss functions*. Apr. 2022. URL: https://programmathically.com/an-introduction-to-neural-network-loss-functions/.

[73] Russell Reed and Robert J MarksII. *Neural smithing: supervised learning in feedforward artificial neural networks*. Mit Press, 1999.

[74] Jason Brownlee. *Loss and loss functions for training Deep Learning Neural Networks*. Oct. 2019. URL: https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/.

[75] Saurav Maheshkar. *What is cross entropy loss? A tutorial with code*. Sept. 2021. URL: https://wandb.ai/sauravmaheshkar/cross-entropy/reports/What-Is-Cross-Entropy-Loss-A-Tutorial-With-Code--VmlldzoxMDA5NTMx.

[76] Jason Brownlee. *A gentle introduction to cross-entropy for Machine Learning*. Dec. 2020. URL: https://machinelearningmastery.com/cross-entropy-for-machine-learning/.

[77] Kiprono Elijah Koech. *Cross-entropy loss function*. July 2022. URL: https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e.

[78] *Multi-hot sparse categorical cross-entropy*. URL: https://cwiki.apache.org/confluence/display/MXNET/Multi-hot+Sparse+Categorical+Cross-entropy.

[79] Y.-S. Park and S. Lek. "Chapter 7 - Artificial Neural Networks: Multilayer Perceptron for Ecological Modeling". In: *Ecological Model Types*. Ed. by Sven Erik Jørgensen. Vol. 28. Developments in Environmental Modelling. Elsevier, 2016, pp. 123–140. DOI: https://doi.org/10.1016/B978-0-444-63623-2.00007-4. URL: https://www.sciencedirect.com/science/article/pii/B9780444636232000074.

[80] Michael A Nielsen. *Neural networks and deep learning*. Vol. 25. Determination press San Francisco, CA, USA, 2015.

[81] Sujin Pyo et al. "Predictability of machine learning techniques to forecast the trends of market index prices: Hypothesis testing for the Korean stock markets". In: *PLOS ONE* 12 (Nov. 2017), e0188107. DOI: 10.1371/journal.pone.0188107.

[82] By: IBM Cloud Education. *What are neural networks?* URL: https://www.ibm.com/cloud/learn/neural-networks.

[83] *Artificial Neural Network: Beginners guide to ann*. May 2021. URL: https://www.analyticsvidhya.com/blog/2021/05/beginners-guide-to-artificial-neural-network/.

[84] M. Minsky and S. Papert. *Perceptrons; an Introduction to Computational Geometry*. MIT Press, 1969. ISBN: 9780262630221. URL: https://books.google.com.hk/books?id=Ow1OAQAAIAAJ.

[85] R.A. Dunne, Murdoch University. Division of Science, and Engineering. *Multi-layer Perceptron Models for Classification*. Murdoch University, 2003. URL: https://books.google.com.hk/books?id=JeW3tgAACAAJ.

[86] Hind Taud and JF Mas. "Multilayer perceptron (MLP)". In: *Geomatic approaches for modeling land change scenarios*. Springer, 2018, pp. 451–455.

[87] Nahua Kang. *Introducing deep learning and neural networks - deep learning for rookies (1)*. Feb. 2019. URL: https://towardsdatascience.com/introducing-deep-learning-and-neural-networks-deep-learning-for-rookies-1-bd68f9cf5883.

[88] Raniah Zaheer and Humera Shaziya. "A study of the optimization algorithms in deep learning". In: *2019 third international conference on inventive systems and control (ICISC)*. IEEE. 2019, pp. 536–539.

[89] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: (Sept. 2016).

[90] Milos Simic. *Difference between the cost, loss, and the objective function*. Mar. 2022. URL: https://www.baeldung.com/cs/cost-vs-loss-vs-objective-function.

[91] Aditya Rakhecha. *Understanding learning rate*. July 2019. URL: https://towardsdatascience.com/https-medium-com-dashingaditya-rakhecha-understanding-learning-rate-dd5da26bb6de.

[92] Jason Brownlee. *Code adam optimization algorithm from scratch*. Oct. 2021. URL: https://machinelearningmastery.com/adam-optimization-from-scratch/.

[93] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[94] Santanu Ganguly. “Neural Networks”. In: *Quantum Machine Learning: An Applied Approach: The Theory and Application of Quantum Machine Learning in Science and Industry*. Berkeley, CA: Apress, 2021, pp. 99–139. ISBN: 978-1-4842-7098-1. DOI: 10.1007/978-1-4842-7098-1_3. URL: https://doi.org/10.1007/978-1-4842-7098-1_3.

[95] Prajit Ramachandran, Barret Zoph, and Quoc Le. “Searching for Activation Functions”. In: 2018. URL: https://arxiv.org/pdf/1710.05941.pdf.

[96] Sunitha Basodi et al. “Gradient Amplification: An Efficient Way to Train Deep Neural Networks”. In: *Big Data Mining and Analytics* 3 (Sept. 2020), pp. 196–207. DOI: 10.26599/BDMA.2020.9020004.

[97] Zhou (Joe) Xu. *Understanding sigmoid, logistic, Softmax functions, and cross-entropy loss (log loss)*. July 2022. URL: https://towardsdatascience.com/understanding-sigmoid-logistic-softmax-functions-and-cross-entropy-loss-log-loss-dbbbe0a17efb.

[98] Jason Brownlee. *Softmax activation function with python*. June 2020. URL: https://machinelearningmastery.com/softmax-activation-function-with-python/.

[99] Ekaba Bisong. “Regularization for Deep Learning”. In: *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*. Berkeley, CA: Apress, 2019, pp. 415–421. ISBN: 978-1-4842-4470-8. DOI: 10.1007/978-1-4842-4470-8_34. URL: https://doi.org/10.1007/978-1-4842-4470-8_34.

[100] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15 (June 2014), pp. 1929–1958.

[101] Jason Brownlee. *Dropout regularization in deep learning models with Keras*. Aug. 2022. URL: https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/.

[102] Robert DiPietro and Gregory D. Hager. “Deep learning: RNNs and LSTM”. English (US). In: *Handbook of Medical Image Computing and Computer Assisted Intervention*. Elsevier, Jan. 2019, pp. 503–519. DOI: 10.1016/B978-0-12-816176-0.00026-0.

[103] Aditi Mittal. *Understanding RNN and LSTM*. Aug. 2021. URL: https://aditi-mittal.medium.com/understanding-rnn-and-lstm-f7cdf6dfc14e.

[104] Sajid A. Marhon, Christopher J. F. Cameron, and Stefan C. Kremer. “Recurrent Neural Networks”. In: *Handbook on Neural Information Processing*. Ed. by Monica Bianchini, Marco Maggini, and Lakhmi C. Jain. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 29–65. ISBN: 978-3-642-36657-4. DOI: 10.1007/978-3-642-36657-4_2. URL: https://doi.org/10.1007/978-3-642-36657-4_2.

[105] Dinesh. *Beginner’s Guide to RNN & LSTMs*. Dec. 2019. URL: https://medium.com/@humble_bee/rnn-recurrent-neural-networks-lstm-842ba7205bbf.

[106] Y. Bengio, P. Simard, and P. Frasconi. “Learning long-term dependencies with gradient descent is difficult”. In: *IEEE Transactions on Neural Networks* 5.2 (1994), pp. 157–166. DOI: 10.1109/72.279181.

[107] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. eprint: https://direct.mit.edu/neco/article-pdf/9/8/1735/813796/neco.1997.9.8.1735.pdf. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

[108] Hongxiang Fan et al. “Comparison of Long Short Term Memory Networks and the Hydrological Model in Runoff Simulation”. In: *Water* 12.1 (2020). ISSN: 2073-4441. DOI: 10.3390/w12010175. URL: https://www.mdpi.com/2073-4441/12/1/175.

[109] *LSTM: Introduction to LSTM: Long short term memor*. Mar. 2021. URL: https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/.

[110] Raghav Aggarwal. *Bi-LSTM*. July 2019. URL: https://medium.com/@raghavaggarwal0089/bi-lstm-bc3d68da8bd0.

[111] Hanane Elfaik and El Habib Nfaoui. “Deep Bidirectional LSTM Network Learning-Based Sentiment Analysis for Arabic Text”. In: *Journal of Intelligent Systems* 30.1 (2021), pp. 395–412. DOI: doi:10.1515/jisys-2020-0021. URL: https://doi.org/10.1515/jisys-2020-0021.

[112] Arvind T Mohan and Datta V Gaitonde. “A deep learning based approach to reduced order modeling for turbulent flow control using LSTM neural networks”. In: *arXiv preprint arXiv:1804.09269* (2018).

[113] Shailja Gupta et al. “Application-based Attention Mechanisms in Deep Learning”. In: *2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*. 2020, pp. 363–366. DOI: 10.1109/ICRITO48877.2020.9197981.

[114] Kenneth K. Fletcher. “An Attention Model for Mashup Tag Recommendation”. In: *Services Computing – SCC 2020*. Ed. by Qingyang Wang et al. Cham: Springer International Publishing, 2020, pp. 50–64.

[115] Peyman Tehrani and Marco Levorato. “Frequency-based Multi Task learning With Attention Mechanism for Fault Detection In Power Systems”. In: *2020 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*. 2020, pp. 1–6. DOI: 10.1109/SmartGridComm47815.2020.9302968.

[116] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. “Neural machine translation by jointly learning to align and translate”. In: *arXiv preprint arXiv:1409.0473* (2014).

[117] Beakcheol Jang et al. “Bi-LSTM Model to Increase Accuracy in Text Classification: Combining Word2vec CNN and Attention Mechanism”. In: *Applied Sciences* 10.17 (2020). ISSN: 2076-3417. URL: https://www.mdpi.com/2076-3417/10/17/5841.

[118] Lei Wang et al. “A novel approach to ultra-short-term multi-step wind power predictions based on encoder–decoder architecture in natural language processing”. In: *Journal of Cleaner Production* 354 (2022), p. 131723. ISSN: 0959-6526. DOI: https://doi.org/10.1016/j.jclepro.2022.131723. URL: https://www.sciencedirect.com/science/article/pii/S0959652622013361.

[119] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[120] Shiyu Zhou et al. “Syllable-based sequence-to-sequence speech recognition with the transformer in mandarin chinese”. In: *arXiv preprint arXiv:1804.10752* (2018).

[121] Yiling Wu et al. “Learning fragment self-attention embeddings for image-text matching”. In: *Proceedings of the 27th ACM International Conference on Multimedia*. 2019, pp. 2088–2096.

[122] Michael Phi. *Illustrated guide to transformers- step by step explanation*. June 2020. URL: https://towardsdatascience.com/illustrated-guide-to-transformers-step-by-step-explanation-f74876522bc0.

[123] Abdul Waheed et al. “Domain-Controlled Title Generation with Human Evaluation”. In: *International Conference on Innovative Computing and Communications*. Springer. 2022, pp. 461–474.

[124] Evangelina Gogoulou. *Using Bidirectional Encoder Representations from Transformers for Conversational Machine Comprehension*. 2019.

[125] Jacob Devlin et al. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805* (2018).

[126] Jiajia Duan et al. “A Study of Pre-trained Language Models in Natural Language Processing”. In: *2020 IEEE International Conference on Smart Cloud (SmartCloud)*. 2020, pp. 116–121. DOI: 10.1109/SmartCloud49737.2020.00030.

[127] Rani Horev. *Bert explained: State of the art language model for NLP*. Nov. 2018. URL: https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270.

[128] *Top 7 cross validation techniques with Python Code*. Aug. 2022. URL: https://www.analyticsvidhya.com/blog/2021/11/top-7-cross-validation-techniques-with-python-code/.

[129] Ping Jiang and Jiejie Chen. “Displacement prediction of landslide based on generalized regression neural networks with K-fold cross-validation”. In: *Neurocomputing* 198 (2016), pp. 40–47.

[130] Thineswaran Gunasegaran and Yu-N Cheah. “Evolutionary cross validation”. In: *2017 8th International Conference on Information Technology (ICIT)*. 2017, pp. 89–95. DOI: 10.1109/ICITECH.2017.8079960.

[131] Payam Refaeilzadeh, Lei Tang, and Huan Liu. “Cross-Validation”. In: *Encyclopedia of Database Systems*. Ed. by Ling Liu and M. Tamer Özsu. New York, NY: Springer New York, 2016, pp. 1–7. ISBN: 978-1-4899-7993-3. DOI: 10.1007/978-1-4899-7993-3_565-2. URL: https://doi.org/10.1007/978-1-4899-7993-3_565-2.

[132] *3.1. cross-validation: Evaluating estimator performance*. URL: https://scikit-learn.org/stable/modules/cross_validation.html.

[133] Damir Krstinić et al. “Multi-label classifier performance evaluation with confusion matrix”. In: *Comput Sci Inf Technol* 10 (2020), pp. 1–14.

[134] Erin L Allwein, Robert E Schapire, and Yoram Singer. “Reducing multiclass to binary: A unifying approach for margin classifiers”. In: *Journal of machine learning research* 1.Dec (2000), pp. 113–141.

[135] Zeya LT. *Essential things you need to know about F1-score*. Feb. 2022. URL: https://towardsdatascience.com/essential-things-you-need-to-know-about-f1-score-dbd973bf1a3.

[136] standish group. *chaos report 2015*. 2016. URL: %7Bhttps://standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf%7D (visited on 04/22/2022).

[137] Muneera Bano and Didar Zowghi. “A systematic review on the relationship between user involvement and system success”. In: *Information and Software Technology* 58 (2015), pp. 148–169. ISSN: 0950-5849. DOI: https://doi.org/10.1016/j.infsof.2014.06.011. URL: https://www.sciencedirect.com/science/article/pii/S0950584914001505.

[138] Dennis Pagano and Bernd Bruegge. “User involvement in software evolution practice: A case study”. In: *2013 35th International Conference on Software Engineering (ICSE)*. 2013, pp. 953–962. DOI: 10.1109/ICSE.2013.6606645.

[139] Eduard C. Groen, Joerg Doerr, and Sebastian Adam. “Towards Crowd-Based Requirements Engineering A Research Preview”. In: *Requirements Engineering: Foundation for Software Quality*. Ed. by Samuel A. Fricker and Kurt Schneider. Cham: Springer International Publishing, 2015, pp. 247–253.

[140] Timo Johann and Walid Maalej. “Democratic mass participation of users in Requirements Engineering?” In: *2015 IEEE 23rd International Requirements Engineering Conference (RE)*. 2015, pp. 256–261. DOI: 10.1109/RE.2015.7320433.

[141] Nicolas Bettenburg et al. “What makes a good bug report?” In: *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. 2008, pp. 308–318.

[142] Bin Fu et al. “Why people hate your app: making sense of user feedback in a mobile app store”. In: *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining* (2013).

[143] Ning Chen et al. “AR-miner: mining informative reviews for developers from mobile app marketplace”. In: *Proceedings of the 36th International Conference on Software Engineering* (2014).

[144] Furqan Rustam et al. “Classification of Shopify App User Reviews Using Novel Multi Text Features”. In: *IEEE Access* 8 (2020), pp. 30234–30244. DOI: 10.1109/ACCESS.2020.2972632.

[145] Walid Maalej and Hadeer Nabil. “Bug report, feature request, or simply praise? On automatically classifying app reviews”. In: *2015 IEEE 23rd International Requirements Engineering Conference (RE)*. 2015, pp. 116–125. DOI: 10.1109/RE.2015.7320414.

[146] Liming Fu et al. “A Machine Learning Based Ensemble Method for Automatic Multiclass Classification of Decisions”. In: *Evaluation and Assessment in Software Engineering* (2021).

[147] Bin Guo et al. “Enhancing Mobile App User Understanding and Marketing With Heterogeneous Crowdsourced Data: A Review”. In: *IEEE Access* 7 (2019), pp. 68557–68571. DOI: 10.1109/ACCESS.2019.2918325.

[148] Emitza Guzman, Muhammad El-Haliby, and Bernd 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)*. 2015, pp. 771–776. DOI: 10.1109/ASE.2015.88.

[149] Nadeem Al Kilani, Rami Tailakh, and Abualsoud Hanani. “Automatic Classification of Apps Reviews for Requirement Engineering: Exploring the Customers Need from Healthcare Applications”. In: *2019 Sixth International Conference on Social Networks Analysis, Management and Security (SNAMS)*. 2019, pp. 541–548. DOI: 10.1109/SNAMS.2019.8931820.

[150] Faiz Ali Shah, Kairit Sirts, and Dietmar Pfahl. “Simplifying the Classification of App Reviews Using Only Lexical Features”. In: *Software Technologies*. Ed. by Marten van Sinderen and Leszek A. Maciaszek. Cham: Springer International Publishing, 2019, pp. 173–193.

[151] Naila Aslam et al. “Convolutional Neural Network Based Classification of App Reviews”. In: *IEEE Access* 8 (2020), pp. 185619–185628. DOI: 10.1109/ACCESS.2020.3029634.

[152] Rohan Reddy Mekala et al. “Classifying User Requirements from Online Feedback in Small Dataset Environments using Deep Learning”. In: *2021 IEEE 29th International Requirements Engineering Conference (RE)*. 2021, pp. 139–149. DOI: 10.1109/RE51729.2021.00020.

[153] Zhilei Qiao et al. “Deep Learning-Based User Feedback Classification in Mobile App Reviews”. In: (2020).

[154] Esraa Karam, Wedad Hussein, and Tarek F. Gharib. “Detecting needs of people in a crisis using Transformer-based question answering techniques”. In: *2021 Tenth International Conference on Intelligent Computing and Information Systems (ICICIS)*. 2021, pp. 348–354. DOI: 10.1109/ICICIS52592.2021.9694136.