StretchVADER – A Rule-based Technique to Improve Sentiment Intensity Detection using Stretched Words and Fine-Grained Sentiment Analysis

by

Muhammad Naveed Jokhio
B.E., Mehran University of Engineering and Technology, Pakistan, 2015

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Applied Science

in the Department of Electrical and Computer Engineering

StretchVADER – A Rule-based Technique to Improve Sentiment Intensity Detection using Stretched Words and Fine-Grained Sentiment Analysis

by

Muhammad Naveed Jokhio
B.E., Mehran University of Engineering and Technology, Pakistan, 2015

Supervisory Committee

Dr. T. A. Gulliver, Supervisor
(Department of Electrical and Computer Engineering)

Dr. M. Sima, Departmental Member
(Department of Electrical and Computer Engineering)

Supervisory Committee

Dr. T. A. Gulliver, Supervisor
(Department of Electrical and Computer Engineering)

Dr. M. Sima, Departmental Member
(Department of Electrical and Computer Engineering)

ABSTRACT

Watching a horror movie and someone shouts "HEEEELLLPPPPPPPPP" or someone replies to your joke with a huge "HAHAHAHAHAHAHAHAHAHAHA" is known as word stretching. Word stretching is not only an integral part of spoken language but is also found in many texts. Though it is very rare in formal writing, it is frequently used on social media. Word stretching emphasizes the meaning of the underlying word, changes the context and impacts the sentiment intensity of the sentence.

In this work, a rule-based fine-grained approach to sentiment analysis named StretchVADER is introduced that extends the capabilities of the rule-based approach called VADER. StretchVADER detects sentiment intensity using textual features such as stretched words and smileys by calculating a StretchVADER Score (SVS). This score is also used to label the dataset. It has been observed that many tweets contain stretched words and smileys, e.g. 28.5% in a randomly extracted dataset from Twitter. A dataset is also generated and annotated using SVS which contains detailed features related to stretched words and smileys. Finally, Machine Learning (ML) models are evaluated using two different data encoding techniques, e.g. TF-IDF and Word2Vec. The results obtained show that the XGBoost algorithm with 1500 gradient-boosted trees and TF-IDF data encoding achieved a higher accuracy, precision, recall and F1-score than the other ML models, i.e. 91.24%, 91.11%, 91.24% and 91.08%, respectively.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

Above all others, I praise our Lord and God, Allah Almighty, the Most Gracious and the Most Merciful, for His countless blessings upon me during my studies and whole life. Secondly, I give respect, credit, and salutations upon our Holy Prophet, and Spiritual Leader Muhammad e Mustafa (P.B.U.H.) whose life acted as a light for me and gave me strength during the hardships while completing this work.

I would definitely like to thank:

Prof. Dr. T. Aaron Gulliver, for his immense support and encouragement. His mentoring, guidance, and patience made it possible for me to complete this work.

My lovely and beautiful wife Dr. Sughra Jokhio, for her love, prayers, and support during the ups and downs of my life.

My father Prof. Dr. Muhammad Hayat Jokhio and whole family, for believing in me, their prayers, support, and all the sacrifices they have made.

Salahuddin Jokhio and my sister Marvi Jokhio, for encouraging me to gain admission to UVic and supporting me financially throughout my studies.

DEDICATION

My spiritual master Sufi Muhammad Nasir Naqshbandi Saifi Mujaddidi Muhammadi m.a. and my spiritual brother Sufi Ali Anwar Bhatti Saifi m.a.

My miscarried babies and my beloved elder sister who passed away during the pandemic.

Chapter 1

Introduction

Social media has made the internet one of the primary sources of information and knowledge. Internet users are continuously accessing social media platforms, i.e. Twitter, Facebook and Reddit to express their sentiments about different topics [4]. The use of these platforms is not limited to the expression of sentiments but also to aid in consultation and decision-making [5]. A few decades ago, individuals would typically seek advice from their friends or rely on critic reviews when buying products or services because the information available was limited [5]. However, the growing popularity of social media and the internet, the explosion of big data and the availability of various tools to extract information have made the decision-making process easier [5]. These tools are used to seek decisions ranging from recommendations on products, brands, services, politics, religion, entertainment, economics and many other topics [5].

Sentiment analysis is a technique that has gained popularity in recent years. It is used by researchers, businesses, organizations, and even governments to extract knowledge, i.e. opinion or sentiment (positive, negative, or neutral), from texts/tweets [6], [7]. Figure 1.1 shows the top 5 words associated with emotions. Sentiment analysis becomes more complex and generates more precise results when the classification is more granular, i.e. sentiments (strongly negative, weakly negative, neutral, weakly positive, strongly positive) or correlated emotions (grief with sadness, pride with admiration, nervousness with fear) as shown in Figure 1.1 [1].

Stretching words [8] or letter repetitions [9] is known as word-lengthening [2], word-stretching, or expressive lengthening [10], [11]. Stretched words are integral

parts of spoken languages and their use has increased significantly [12]. Stretching a word modifies the context of the base word, i.e. strengthening or emphasizing the base word (e.g. huuuuuge, sooooo good), showing excitement (e.g. yeeeeeeeees), intense laughter (e.g. hahahahahah), or sarcasm (e.g. sssuuuuure), or even indicating danger (e.g. nnnnooooooo, heeellllppppp) [12].

love	joy	excitement	anger	fear	disgust
love	happy	excited	f***	scared	disgusting
loved	glad	happy	hate	afraid	awful
favorite	enjoy	cake	f**ing	scary	worst
loves	enjoyed	wow	angry	terrible	worse
like	fun	interesting	dare	terrifying	interesting

confusion	curiosity	surprise
confused	curious	wow
why	what	surprised
sure	why	wonder
what	how	shocked
understand	did	omg

Figure 1.1: The top 5 words associated with emotions [1].

1.1 Motivation and Problem Statement

Stretched words can be found in many novels [13] but not in formal English dictionaries, i.e. the Oxford English dictionary does not have “lolzzzzz” or ”hahahah” [14]. There is a scarcity of dictionaries, literature and techniques that can help researchers understand, process and benefit from stretched words in order to extract information and knowledge.

In this thesis, a new approach called StretchVADER is introduced to take advantage of textual features found in social media. It is an extension of the rule-based algorithm called Valence Aware Dictionary for sEntiment Reasoning (VADER). It is sensitive to both the polarity (positive/negative) and intensity (strength) of emotions. The following factors are important for sentiment analysis.

1. Sentiment analysis techniques should be general and process every type of text, i.e. simple texts and emoji-based texts or tweets [6].

2. Sentiment-bearing textual features such as
   (a) smileys, i.e. emoticons :) and emojis 😀,
   (b) social media-specific terms and acronyms, e.g. gonna, wanna, lmao, lolz, roflz, u, good ni8, hahahaha, smh, idk,
   (c) slang words, i.e. BS, *** and
   (d) capitalization and punctuation, e.g. OMG!!!
3. The ability to process comparative expressions, for example
   (a) this iPhone model A is much better than Samsung model B and
   (b) the location was truly disgusting…. but the people there were glorious.
4. Stretched words because
   (a) stretched words are not used arbitrarily, but rather are applied to words to strengthen the sentiment or emotion they convey [2],
   (b) vowel lengthening and consonant lengthening (called gemination), is a feature of some languages and can change a word including its meaning [15], and
   (c) there is a connection between the amount of stretch and intensity, i.e. so sad and soooooo sad, omg and OMG!!!!, hahahaha, reallllllllllly satisfying, aweawwwesome and heellppp.

1.2 Related Work

Machine Learning (ML) algorithms such as Naïve Bayes, Maximum Entropy (MaxEnt) and Support Vector Machines (SVM) have been used to detect sentiment from movie reviews [16]. The performance was evaluated using traditional n-gram techniques along with Parts-of-Speech (POS) information. An accuracy of 82.9% was achieved using SVM with unigrams. However, the dataset was small and the classification was not fine-grained. Contrasting and comparative sentences can easily confuse classifiers and result in misclassification and so were ignored. Further, stretched words were not considered.

Table 1.1: List of emoticons [1].

Positive Emoticons	**Negative Emoticons**
:)	:(
:-)	:-(
:)	: (
:D	
=)	

The publicly available dataset Sentiment140 was used for sentiment analysis in [4]. Distant supervision (rule-based) was used on emoticons to weakly label the dataset. Weakly label means the dataset was labeled based on just the presence of emoticons in the tweet. Two classes were used, positive and negative. The positive and negative emoticons are shown in Table 1.1. The tweets which contain a happy emoticon from the list were labeled as positive, while the tweets which contain a sad emoticon from the list were labeled as negative. However, what if the context of the tweet contrasts with the emoticons being used in the tweet, e.g. sarcasm? A very small number of emoticons were mapped to happy and sad emoticon. There is no mention of emojis in the list of emoticons, i.e. 😀, ☹, or special characters, i.e. =×, ❤, <3. After labeling, the emoticons were removed from the dataset. It was not possible to validate their labeling using emoticons because all emoticons were removed from the dataset after labeling. Further, in the preprocessing, stretched words were removed from the tweets. It was concluded that ML classifiers, i.e. Naïve Bayes, MaxEnt and SVM provided accuracies of 81.3%, 80.5% and 82.2% compared to 81.0%, 80.4% and 82.9% for Naïve Bayes, MaxEnt and SVM, respectively [16].

Smileys and special characters are crucial in sentiment analysis as they can change the context, i.e. criticism or sarcasm [17], [18], [19]. Emoticons in text were used to detect sentiment in [20]. A manually created emoticon sentiment dictionary was used with lexical-based techniques on 2,080 Dutch tweets and forum messages. The results obtained show that applying emoticons with existing lexical-based text classification improved both the F1-score and accuracy from 22% to 94% at paragraph level, but the F1-score was 66% and accuracy was 59% when applied at sentence level. Furthermore, the approach is limited to the size of the emoticon dictionary and it is language

specific.

In [5], sentiment analysis was performed using traditional n-gram techniques with POS tags. An accuracy of 76.6% was obtained for SVM-lite with POS tags on bigrams used as TF-IDF features. However, important textual features such as fine-graining, stretched words and emoticons were ignored. The results showed that using trigrams instead of bigrams improved the accuracy.

In recent years, different aspects of languages have been examined [2], [11], [21–35]. Stretched words were studied in [2, 9–11, 13, 30–32, 36–39], but text processing is difficult when it contains stretched words. Some of the issues with currently available methods for text normalization were discussed in [10]. The differences between formal and informal language writing patterns of Twitter users who include emoticon faces with noses and without noses were determined in [11]. It was shown that users who use face emoticons without noses have informal writing patterns with many stretched words and vice versa [20]. In [2], it was observed that there are a significant number of stretched words in a dataset of half a million tweets. Stretched words were found in 17.44% of tweets. Stretched words are not arbitrarily used but have a strong relationship with their root words and this relationship has been used to find and classify new sentiment-bearing words [7]. Table 1.2 shows the reduction of stretched words to condensed forms and root words. In [2], stretched words were reduced to their root words using the steps described in Table 1.2. Single-letter repetitions were first removed from the stretched words, e.g. reeeeeallllly so there is a single occurrence, e.g. e, l in order to reach the condensed form realy. All words along with their frequencies in the dataset which lead to the same condensed form were found and recorded, e.g. realy, really, realllly,… have same condensed form realy. Then, the word which has the highest number of occurrences in the dataset, e.g. really is chosen as the root word for the corresponding stretched words. The problem with this approach is that it ignores paired duplicates, i.e. aweawesome, hahaha (awe and ha are repeated in pairs) and it is corpus-limited which means new words not in the corpus are ignored. Further, the relationship between stretch and sentiment is not considered. In [12], approximately 100 billion tweets were examined. Two parameters known as balance and stretch were introduced for stretched words. Balance measures the balance of the stretched word, e.g. hahahaha has a balance of 1 while hahahaa has a balance close to 1. Stretch is the number of repetitions of letters inside a word [12].

Table 1.2: Reduction of stretched words to condensed forms and root words [2].

1. For every word in the vocabulary, extract the condensed form, where sequences of repeated letters are replaced with a single instance of that letter.

   $niiiice \rightarrow nice$, $realllly \rightarrow realy$

2. Create sets of words sharing the same condensed form.

   $\{nice, niiice, nicccccceee...\}, \{realy, really, realllly, ...\}$

3. Remove sets which do not contain at least one length three repetition.

   $\{committee, committe, commitee\}$

4. Find the most frequently occurring condensed form in the dataset and select it as the root word for all the condensed forms in the set.

   $\{\mathbf{nice}, niiice, nicccccceee...\}, \{realy, \mathbf{really}, realllly, ...\}$

In [40], a simple rule-based algorithm called VADER was introduced to perform sentiment analysis. VADER consists of a long list of lexical features with their corresponding sentiment intensity, fine-tuned for text and words found on microblogs like Twitter. Five general rules consisting of grammatical and syntactical conventions were used to express sentiment intensity. The algorithm was evaluated using 4,200 tweets and the results compared with various benchmarks including Linguistic Inquiry and Word Count (LIWC), SentiWordNet, Naïve Bayes, MaxEnt and SVM. It was found that VADER provides the best performance with an F1-score of 96% and an accuracy of 84%.

1.3 Contributions

The contributions of this thesis are summarized as follows.

1. A large dataset containing a continuous stream of approximately 688,881 tweets was extracted from Twitter randomly over 24 hours from 2011-09-28 to 2011-09-29 using the JSON API Spritzer. The JSON data was then converted into tabular form to create a dataframe for Python. It contained raw text data and many other unrelated columns. The dataset was then preprocessed to remove all columns except the text column and then the data preprocessing mentioned

in Section 2.4.3 was performed. Next, several data columns related to stretched words and smileys were derived from the text column as outlined below.

- contains_stretched indicates whether the tweet contains stretched words or not.
- stretched_words is a list of all stretched words found in the tweet.
- root_words is a list of all root words of the stretched words in the tweet.
- in_dict is a binary column that indicates whether the root word was found in the English dictionary or not.
- contains_smileys is a binary column that indicates whether smileys were found in the tweet or not.
- smileys is a list of all smileys found in the tweet.
- VADER_compound is the sum of all VADER scores and indicates the sentiment intensity and polarity of the tweet.
- Stretch_score is the sum of all the stretch scores in the tweet.
- StretchVADER_Score is the sum of all stretch and VADER scores and indicates the improved sentiment intensity and polarity of the tweet.
- StretchVADER_label is the class label for the tweet.

2. A rule-based technique named StretchVADER is proposed as an extension of the existing rule-based technique VADER to improve sentiment intensity using stretched words and fine-grained sentiment analysis. VADER is fine-tuned for social media-specific tasks. It considers 5 rules that embody grammatical and syntactical conventions used by humans when expressing or emphasizing sentiment intensity. Rules are punctuation (good!), capitalization (GOOD!), degree modifiers (extemely good), contrasting sentences, and the tri-gram preceding a sentiment-bearing feature (The food here is not really all that great). The stretched words and smileys are extracted, reduced to their roots words and processed by the extension StretchVADER proposed here in this thesis.

3. New parameters for stretched words are introduced, i.e. stretch, stretch score and StretchVADER Score (SVS). SVS is calculated from the stretch and stretch score. It is added to the sentiment intensity provided by VADER to obtain an improved sentiment intensity for the text.

4. After the dataset is labelled using StretchVADER, it is encoded with two data encodings, i.e. TF-IDF and Word2Vec (W2V). Then five ML algorithms are implemented with these data encodings to determine which combinations perform better. ML algorithms are considered because they provide adaptability, scalability and automation.

1.4 Thesis Organization

- **Chapter 1** provided an introduction to the thesis and presented the problem statement and motivation. The related literature was reviewed and discussed. The contributions of this research were outlined and the thesis organization was given.

- **Chapter 2** provides an introduction to sentiment analysis including its levels, types and approaches. The relationship between word stretching and sentiment intensity is examined. Furthermore, the proposed framework with StretchVADER is given. The dataset is introduced along with its labeling using the SVS.

- **Chapter 3** provides a detailed discussion of ML algorithms. The implementation of ML algorithms with StretchVADER is also given.

- **Chapter 4** presents the results for the ML algorithms with TF-IDF and W2V data encoding. The performance of the ML algorithms is evaluated using accuracy, precision, recall, F1-score and execution time. Hyperparameter-tuning results are also given to demonstrate their effect on performance.

- **Chapter 5** concludes the thesis and provides some suggestions for future work.

Chapter 2

Sentiment Analysis and StretchVADER

Sentiment analysis, sometimes called opinion mining or opinion analysis, is an area of Natural Language Processing (NLP) [20, 41]. It is used to study opinions, sentiments and emotions in text. It is a challenging but crucial NLP task because of its importance in practical applications. Typically, sentiment analysis is used to either detect the polarity of text (positive, negative, or neutral) or subjectivity of opinion (personal opinion or factual information). However, it can also be used to detect feelings and emotions (joy, happiness, anger, sarcasm and sadness), urgency (urgent, not urgent) and interest or intention (interested or not interested). Depending on the scope and application, sentiment analysis can be divided into different types and levels as discussed below.

2.1 Sentiment Analysis Levels

2.1.1 Document-level Sentiment Analysis

Sentiment analysis can be applied at the document level to assign polarity to complete documents. Figure 2.1 shows the levels of sentiment analysis along with the applications for each level. The applications include detecting emotions from news articles, books or chapters, pages or sections of a book as shown in Figure 2.1. Both supervised and unsupervised ML approaches have been used at this level [42].

2.1.2 Sentence-level Sentiment Analysis

Sentence-level sentiment analysis extracts sentiments or emotions from individual sentences rather than documents. Individual sentence polarities can be aggregated to determine the sentiment of a document. When a document has mixed sentiment-bearing sentences, sentence-level sentiment analysis can be more effective than document-level sentiment analysis [43]. One of the applications of sentence-level sentiment analysis is to detect subjective sentences (sentences having emotions rather than factual information) [44].

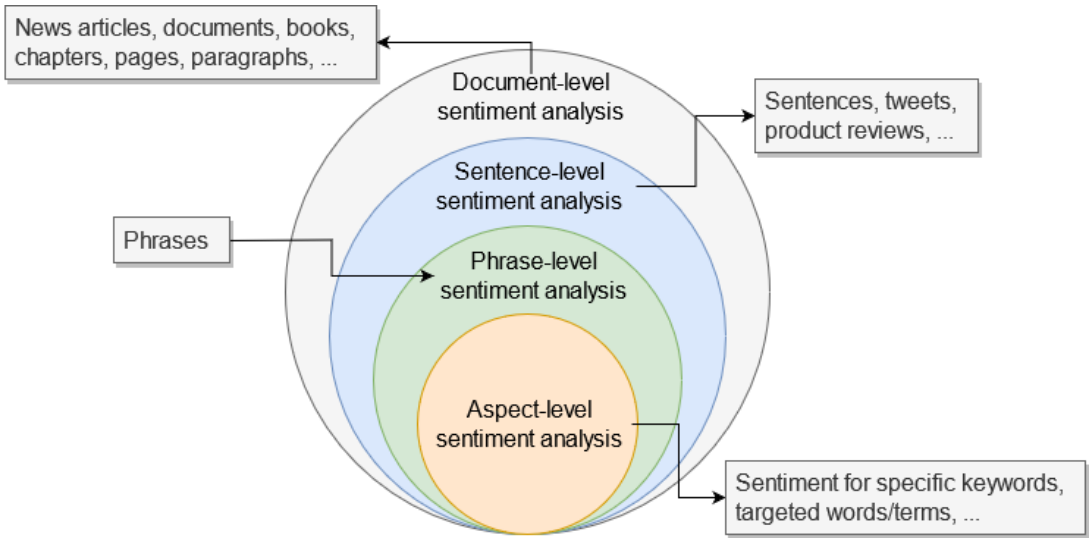


Figure 2.1: Levels of sentiment analysis.

2.1.3 Phrase-level Sentiment Analysis

Phrase-level sentiment analysis has many practical applications in product reviews and text containing sarcasm and criticism. At this level, phrases rather than complete sentences are assigned polarities.

2.1.4 Aspect-level Sentiment Analysis

A phrase in a sentence can have more than one sentiment. The goal of aspect-level sentiment analysis is to detect sentiment polarity for specific terms in a sentence [45]. It is also called target-level or word-level sentiment analysis.

2.2 Types of Sentiment Analysis

Sentiment analysis can be divided into two types, binary and multi-class, fine-grained or graded. Binary sentiment analysis has two classes, e.g. positive and negative, subjective and factual, or sarcastic and not sarcastic. This is useful in many cases but fails when the text contains mixed sentiments, multiple sentiment-bearing sentences (phrases inside sentences with differing sentiments), comparative analysis, or sentiment towards a specific term. Multi-class sentiment analysis is used when there are more than two classes. In recent years, research has increased in this area because it can provide more precise and accurate results [46]. The most frequently used multi-class classifications are 5-class and 7-class. The labels and classes depend on the problem. The most common 5-class labels are strongly negative, weakly negative, neutral, weakly positive and strongly positive.

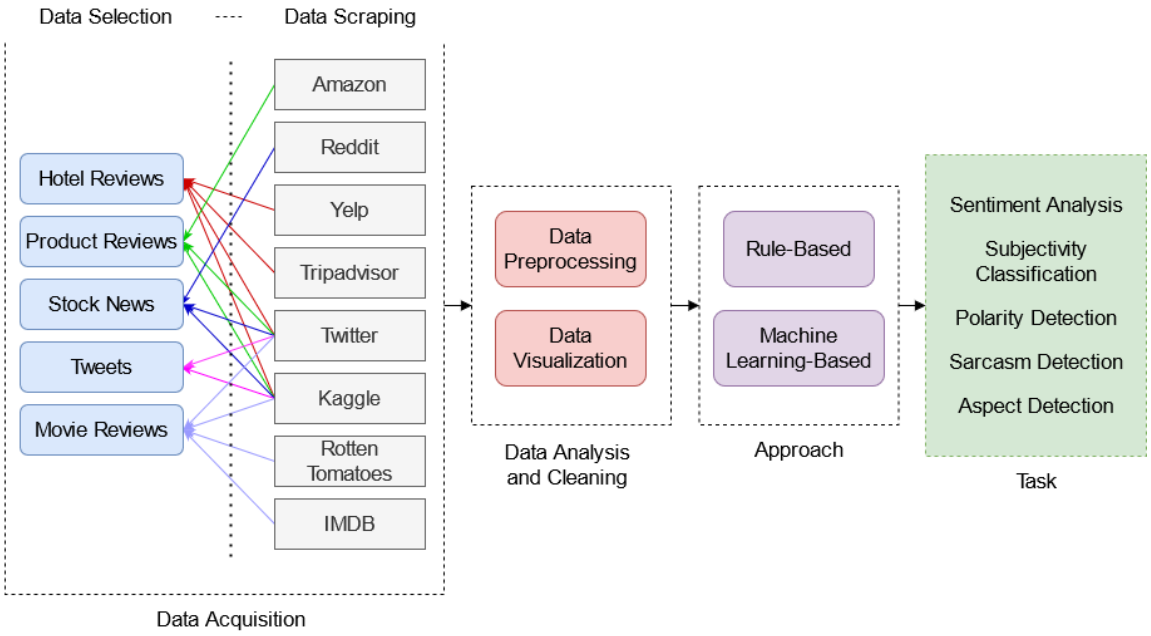


Figure 2.2: General procedure for sentiment analysis [3].

2.3 Sentiment Analysis Approaches

Text has morphological, grammatical, syntactical and semantic-based features. These features are used in sentiment analysis to detect sentiment from text. There are many

approaches to detect sentiment from text. The two approaches related to this research are described below.

2.3.1 Rule-based Approaches

The features mentioned in Section 2.3 can be used with pattern matching to create rules either manually or automatically. These rules can then be used to determine the sentiment of text [40]. The main advantage of this approach is that it does not require labeled data and performs very well on unstructured text.

2.3.2 Machine Learning-based Approaches

ML algorithms can be used to detect sentiments from text. Supervised ML approaches use labeled data to detect sentiment while unsupervised ML approaches use algorithms to automatically identify sentiment without predefined rules and are based on hidden patterns.

2.4 StretchVADER

StretchVADER is a rule-based technique to improve sentiment intensity detection using stretched words and fine-grained sentiment analysis. It extends the capabilities of the existing rule-based algorithm VADER which is fine-tuned for social media-specific text. The sentiment score and polarity of the stretched word is calculated using StretchVADER. One of the reasons for using StretchVADER instead of transformer-based models is because rule-based models are more transparent and explainable than complex ML models, provide more control over logic and deeper domain-specific knowledge, and require fewer resources.

2.4.1 Data Acquisition

Figure 2.2 shows the general procedure for sentiment analysis and Figure 2.3 gives the proposed framework with StretchVADER. Data acquisition is the first and most crucial step in sentiment analysis as shown in Figures 2.2 and 2.3. It employs data selection and data scraping. The data selected, e.g. tweets, Amazon reviews, web articles, hotel reviews and/or stock news depends on the research problem. The extraction of data from various sources, e.g. Twitter, Kaggle, Rotten Tomatoes,

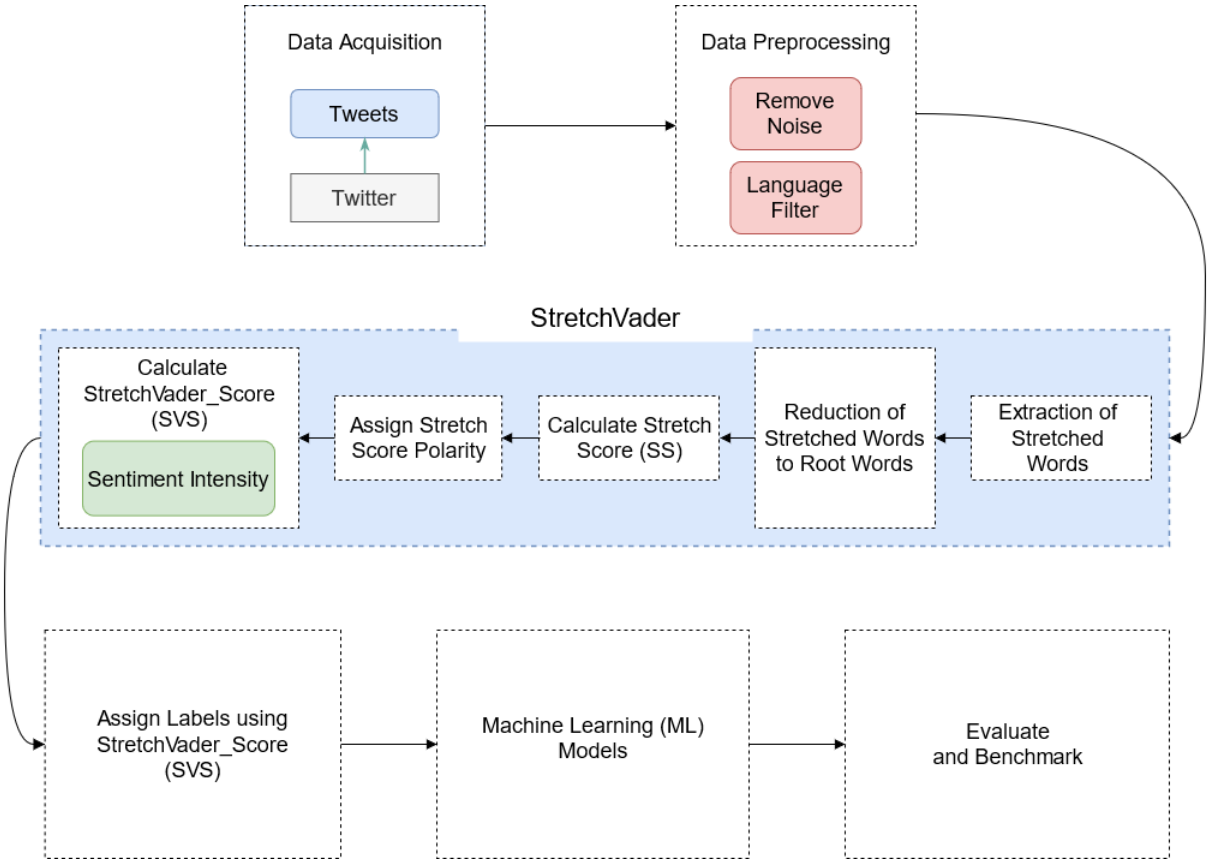


Figure 2.3: The proposed framework with StretchVADER.

Reddit, Amazon, Tripadvisor and/or Yelp is called data scraping. As this research considers stretched words in sentiment analysis and stretched words are found in abundance on Twitter [2], Twitter data is used here.

2.4.2 Dataset Analysis

NG-StV_Senti dataset is created in this study. It contains tweets randomly sampled over 24 hours from 2011-09-28 to 2011-09-29. NG-StV_Senti is a general and a non-biased dataset because it contains a continuous stream of tweets from diverse regions, languages and topics. Table 2.1 shows the distribution of tweets in the dataset.

2.4.3 Data Preprocessing

Data preprocessing is a crucial step in sentiment analysis. Without careful preprocessing, essential information within the data may be lost and noise such as name handles,

Table 2.1: NG-StV_Senti dataset contents and distribution.

Type	Number of tweets	Percentage
Tweets with text only without smileys and stretched words	492,420	71.5%
Tweets with all textual features e.g. stretched words and smileys	196,461	28.5%
Tweets with stretched words only without smileys	113,476	16.5%
Tweets with smileys only without stretched words	60,844	8.8%
Tweets with both smileys and stretched words	22,141	3.2%

hashtags and URLs in the data can result in inaccurate results. The preprocessing steps used in this research are as follows.

1. Name handles, hashtags and URLs are removed.
2. Tweets in languages other than English are removed.
3. Numeric values including random digits, phone numbers, dates and years, are removed from the data. However, numbers combined with letters to create meaningful words such as good ni8 and smileys such as <3 are not removed.

2.5 Word Stretching

As discussed in Section 1.1, word stretching is not arbitrary but rather is applied to base words to change their context. Word stretching can be used to strengthen the meaning of base words [2]. Thus, stretched words should be considered to detect the sentiment of text.

2.5.1 Stretched Words

Stretched words were extracted from the tweets using the following steps.

1. Tweets containing stretched words were filtered using the following regular expression

$$r`(\backslash b\backslash w * (\backslash w)(\backslash w)(? : \backslash 2|\backslash 3)\{2, \}\backslash w * \backslash b)' \tag{2.1}$$

   A regular expression is a sequence of characters used in string-searching algorithms. It uses pattern matching in text and a matched string is known as a token. The above expression will extract any word that has a single character (sooo) or paired characters (awawesome) repeated at least twice.

2. After the stretched words were collected, each word was reduced to its root word [2], [12].

2.5.2 Reduction of Stretched Words to Root Words

Consider a word with some letters repeated to form a stretched word such as reaaaaaally/lolzzzz and aweawesome/hahaha. For instance, the example in Table 2.2 that consists of both single-letter and paired duplicates Awawawaaawwwwwessssssommmmmeeeeeee. The reduction steps are as follows and a flowchart is given in Figure 2.4.

1. Consecutive repeated letters are removed, e.g. Awawaw[a][w]e[s]o[m][e] and recorded separately, e.g. {repeated letters = a, w, s, m, e}.

2. Consecutive paired duplicates are removed, e.g. (aw)e[s]o[m][e] to obtain a kernel and a condensed form of the stretched word. A **kernel** is an expression of the stretched word with brackets showing repetition of single or paired letters and a **condensed form** of the stretched word is generated from its kernel without brackets which has only a single occurrence of every letter.

3. The condensed form is matched with the English dictionary. If it is not found, then consecutive repeated letters that were recorded separately, are inserted back, one by one in condensed form to generate a **root word** until it is found in the English dictionary.

Table 2.2 gives four examples of stretched words and their reduction to root words. The first line of each example shows the extracted stretched words. The next lines starting with $\rightarrow$ are the reduction steps. Single letter reduction is shown in square brackets, e.g. [a] and [l] in re[a][l]y and paired reduction is represented by round

Table 2.2: Reduction of stretched words to their root words in StretchVADER.

1.	Hahahahahaha $\rightarrow (ha)$ - matched with the dictionary **= ha**
2.	Goooooooaaaaaaal $\rightarrow g[o][a]l$ {repeated letters = o, a} - matched with the dictionary **= goal**
3.	Awawawaaawwwwwesssssommmmmeeeeee $\rightarrow Awawaw[a][w]e[s]o[m][e]$ {repeated letters = a, w, s, m, e} $\rightarrow (aw)e[s]o[m][e]$ {repeated letters = s, m, e} – matched with the dictionary **= awesome**
4.	Reaaallllllly $\rightarrow re[a][l]y$ {repeated letters = a, l} – does not match with the dictionary, so the repeated letters, i.e. "a,l" are inserted back one by one - matched with the dictionary (no need to reduce further) **= really**

brackets, e.g. (ha) in Hahahahahaha. Repeated letters are shown in curly brackets, e.g. {repeated letters = a,l} in re[a][l]y. The bold words in each example are the root words. Figure 2.5 shows the top 5 (head) and bottom 5 (tail) examples of the stretched words and their reduction to root words from the NG-StV_Senti dataset.

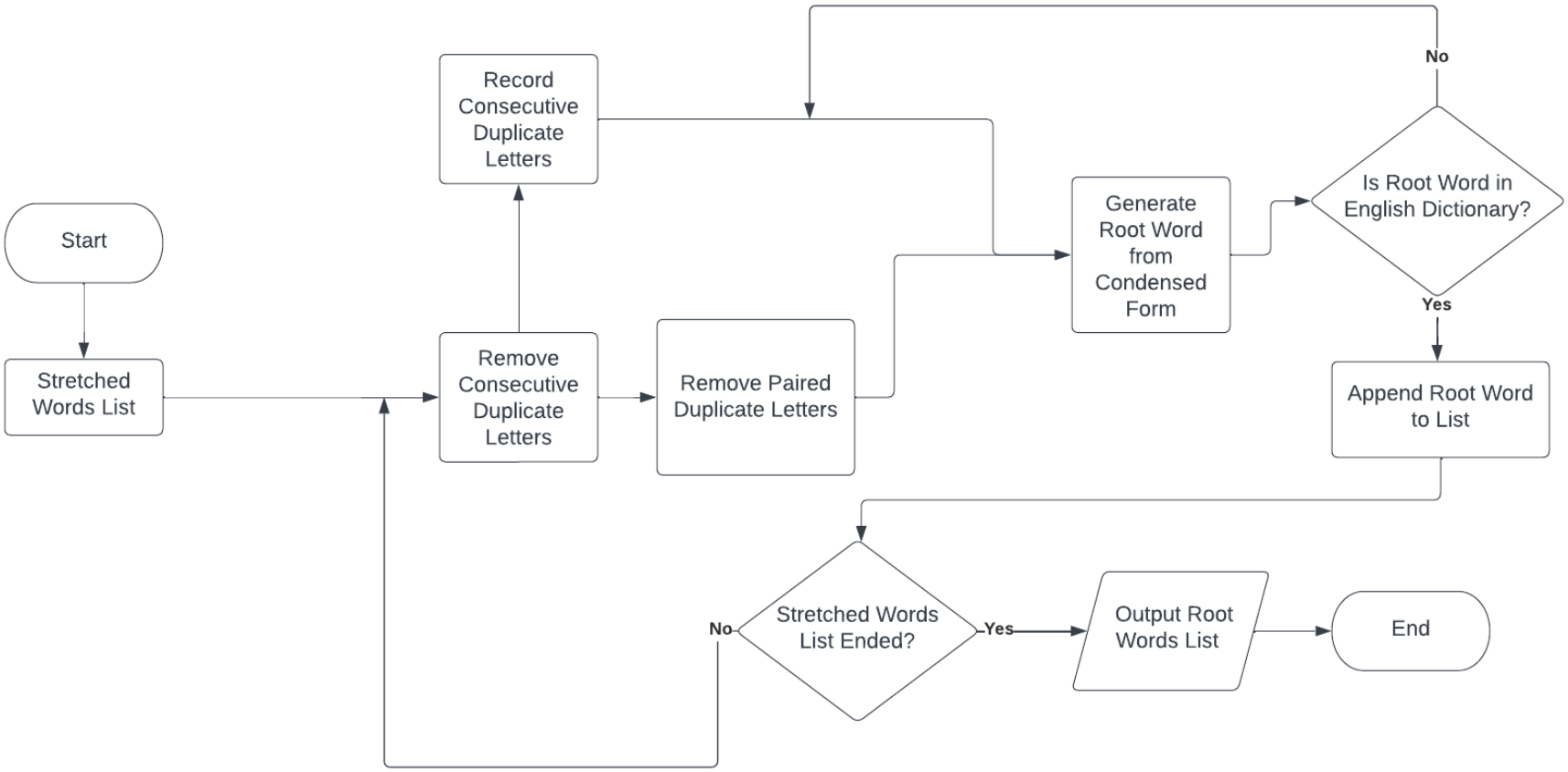


Figure 2.4: Flowchart of the reduction of stretched words to root words.

	contains_stretched	stretched_words	in_dict	condensed_form	contains_paired_duplicates	Consecutive_Duplicate_Letters	root_words
0	1	ewwwwwwwwww	[0]	ew	0	w	ew
1	1	goneee	[0]	gone	0	e	gone
2	1	hahahaha	[0]	ha	1		ha
3	1	folllowbck	[0]	folowbck	0	l	folowbck
4	1	_____	[0]	_	0	_____	_
...	...	...	...	...	...	...	...
67305	1	shitttttt,GOOOOOO,babyyyyy	[0, 0, 0]	shit,go,baby	0	t,O,y	shit,GO,baby
67306	1	muahahahahhahahahaha	[0]	muaha	1	h	muaha
67307	1	yepppp,illinois	[0, 1]	yep,ilinois	0	p,l	yep,illinois
67308	1	proceeeeeed	[0]	proced	0	e	proceed
67309	1	whoooo	[0]	who	0	o	who

67310 rows × 7 columns

Figure 2.5: Top 5 and bottom 5 examples from the NG-StV_Senti dataset showing stretched words and their reduction to root words.

2.5.3 Stretch Score (SS)

Stretch is the number of repetitions of letters in a word. For example, the stretch for Sooo sad is 2 because o is repeated twice. The stretch for aweawesooome is 3 because awe is repeated once and o is repeated twice. A sentence can have multiple stretched words. A stretched word can have positive or negative meaning. Similarly, it can have polarity, e.g. positive or negative. Polarity is assigned according to the root word in the sentence. A sentence can be a simple sentence, e.g. I love Mercedes, or a compound sentence connected using conjunctions (but, and, or, because, since, yet), having multiple polarities. Compound sentences can have same polarity clauses or different polarity clauses, e.g. I love this product and it is worth buying or I love this product but people hate it. VADER is used to detect the polarity of a complete sentence and also individual clauses in the case of a compound sentence. Assigning polarities in a simple sentence is easy, as a stretched word in a positive simple sentence gets positive polarity and similarly for a negative simple sentence. Consider the following examples.

- iPhone 13 pro camera is sooooooooo good → positive polarity.
- I don't loveeeeeee this product → loveeeeeee gets negative polarity. It is a positive word but it is used here to create negative sentiment.
- Ewwwww this hotel is not worth living in → negative polarity.

In the case of compound sentences, stretched words are assigned polarities based on their occurrence in the sentence depending on the clause. Compound sentences have more than one clause connected together using conjunctions. A coordinating conjunction joins two or more words, main clauses or sentences which have the same syntactic importance. There are seven coordinating conjunctions, i.e. for, and, nor, but, or, yet and so. In this thesis, coordinating conjunctions are used instead of other conjunctions because they are used to generate compound sentences and contrasting sentences. POS tag, i.e. Coordinating Conjunction (CC) is used to find conjunctions in compound sentences and they are used to split the sentences into clauses. Then, the sentiment of each clause is detected using the VADER algorithm. Finally, the polarity of the clause is assigned. Consider the following examples.

- I loveeeeeee iPhone but some people do not like it.

  The polarity for loveeeeeee depends on the clause, not the complete sentence, because it resides in the positive part (clause) of the sentence. So, loveeeeeee is assigned positive polarity.

- This product is very cheap but people are stilllllllllll using it.

  This sentence has overall negative polarity but it is a compound sentence, so the polarity for stilllllllllll depends on the clause not the complete sentence. The person does not like the product but most people are still using it, so stretching reduces the negative sentiment of the sentence. The polarity for stilllllllllll will be positive because it is used inside the positive clause.

After assigning polarities, the stretch score for a word is

$$ss = \pm \frac{stretch}{total\,length\,of\,stretched\,word} \tag{2.2}$$

A single sentence or a tweet can have multiple stretched words, so the stretch scores are added to produce an overall Stretch Score (SS) given by

$$SS = \sum_{i=1}^{n} ss_i \tag{2.3}$$

2.6 Sentiment Intensity (SI)

A sentiment score consists of polarity and intensity [47]. Sentiment intensity quantifies the positive or negative sentiment as a number, e.g. +0.987 or -0.766. Each stretched word is replaced by its root word in the text to generate a clean text. The Sentiment Intensity (SI) of the clean text of a tweet is calculated using the VADER algorithm and added to the SS to obtain an improved sentiment intensity, i.e. StretchVADER Score (SVS). The SI of text is obtained by adding the sentiment scores of the individual words from the VADER dictionary where each word is assigned an emotion intensity. The SVS is given by

$$SVS = SI + SS \tag{2.4}$$

Figure 2.6 shows the target class labels with their respective SVS ranges. There are five classes, i.e. Strongly Negative (SN), Weakly Negative (WN), Neutral (N), Weakly Positive (WP) and Strongly Positive (SP). Initially, the SVS values were within the range -1 to +1. The text with no sentiment has SVS value 0 and is assigned to the N class. The remaining four classes include two positive classes, i.e. WP and SP with SVS values greater than 0 and two negative classes, i.e. WN and SN with SVS values less than 0. The break points for both positive and negative classes were determined by dividing the maximum values, i.e. +1 and -1 by 2 in order to split the SVS ranges into four equal bins. This is called equal-width bining. Then, the SVS values were assigned to respective classes. The SVS ranges for strong sentiments, i.e. SN and SP, have open intervals as shown in Figure 2.6 and expressed in (2.5). If the SVS is greater than 0 and less than or equal to +0.5, then the text is assigned to the WP class. If the SVS is greater than +0.5, then the text is assigned to the SP class. Negative sentiments are similarly assigned.

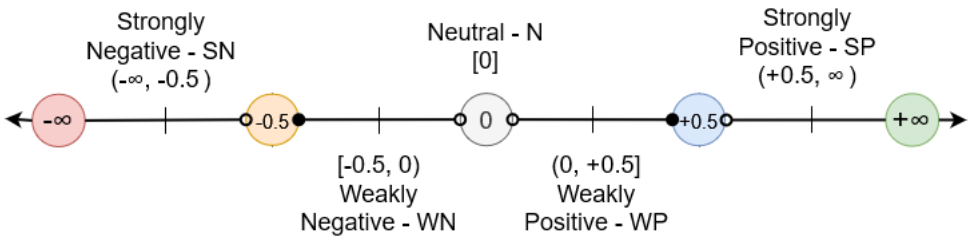


Figure 2.6: Target class labels with their respective SVS ranges.

$$StretchVADER\ \ Label = \begin{cases} SN, & SVS < -0.5 \\ WN, & -0.5 \leq SVS < 0 \\ N, & SVS = 0 \\ WP, & 0 < SVS \leq 0.5 \\ SP, & 0.5 < SVS \end{cases} \tag{2.5}$$

Chapter 3

Machine Learning with StretchVADER

ML is a branch of Artificial Intelligence (AI) that enables computers to learn without being explicitly programmed. It has gained significant attention because of publically available datasets and growing computing and storage capacities [48]. Datasets are used by ML algorithms to either learn from labelled data or to find hidden and important patterns from data that was not labelled. ML has roots in data mining, statistics and computational mathematics.

3.1 Types of ML Algorithms

ML algorithms can be categorized into the following types.

1. Supervised Machine Learning
2. Unsupervised Machine Learning
3. Semi-supervised Machine Learning
4. Reinforcement Learning

3.1.1 Supervised Machine Learning

Supervised ML uses known or labelled data to predict unknown or new data. The data fed into an ML algorithm contain features and the outputs are called output

variables. Supervised ML algorithms determine the relationship between the features and the output variables [49]. Based on the output variables, supervised ML can be divided into two types.

1. **Regression**: If the algorithm predicts continuous values, e.g. sales, stocks and prices.

2. **Classification:** If the algorithm predicts discrete or limited values, e.g. fraudulent or legitimate, spam or not spam and positive or negative. Classification problems can be binary (two classes) or multi-class (more than two classes).

There are many supervised ML algorithms but the most commonly used are Naïve Bayes, Linear Regression, Decision Trees, K-Nearest Neighbors (KNN), Random Forest, SVM and Neural Networks [50].

3.1.2 Unsupervised Machine Learning

Unsupervised ML is used to detect hidden patterns in data that has not been labelled. It is useful when a user does not know what to look for in the data [49]. Unsupervised ML algorithms are used for pattern recognition, descriptive modeling, association rule mining, clustering, dimensionality reduction and anomaly detection [51]. The results are not specific classes or labels but rather are rules, associations, patterns, or groups (clusters) in the data. Some commonly used unsupervised ML algorithms are K-Means Clustering, Hierarchical Clustering, Apriori Algorithm (association rule mining), Principal Component Analysis (PCA) and Hidden Markov Model (HMM).

3.2 Machine Learning with StretchVADER

In this section, ML with StretchVADER is discussed in detail and the steps are shown in Figure 3.1.

3.2.1 Label Encoding

Label encoding is the process of encoding target classes which are in textual form to numerical form. It is an essential step when dealing with text data. Figure 3.2 shows the label encoding of the sentiment classes. The sentiment classes, e.g. SN, WN, N, WP and SP are manually encoded as -2, -1, 0, +1 and +2, respectively. These are called StretchVADER labels.

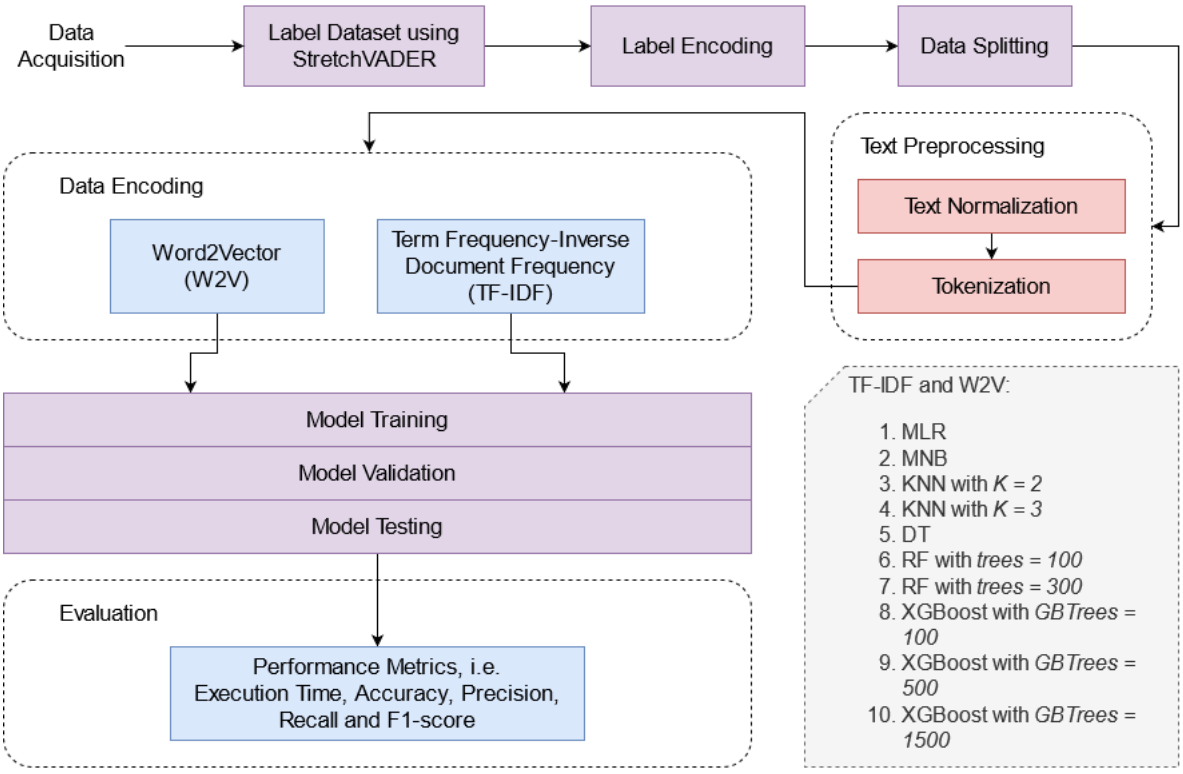


Figure 3.1: ML with StretchVADER.

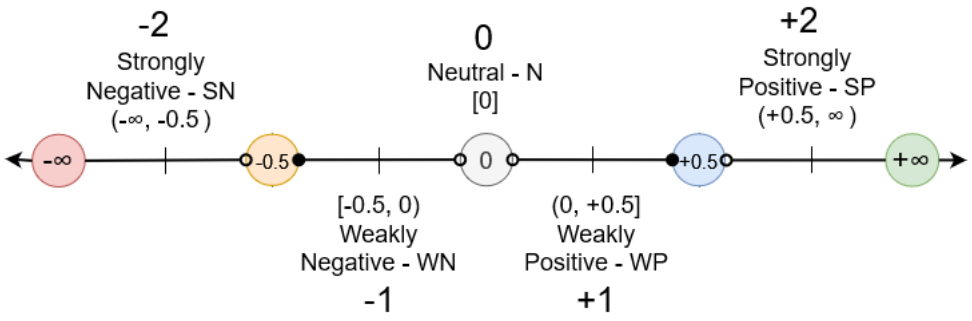


Figure 3.2: Label encoding

3.2.2 Data Spliting

Data splitting is the process of dividing a dataset into training and testing datasets. In [52], the training dataset was split into training and validation sets. This division allows for early evaluation of model performance using the validation set, which is a subset of the training dataset [52].

In this thesis, the dataset is divided into three sets. Initially, it is split into

two subsets, i.e. the training dataset and the testing dataset. The training dataset constitutes 70% of the original dataset, while the remaining 30% is allocated for testing the ML algorithms. To incorporate validation, the training dataset is further divided into two subsets, i.e. the training set and the validation set. The training set encompasses 80% of the training dataset which will be used for training the ML algorithms, and the remaining 20% is assigned to the validation set.

All the tweets in the dataset were shuffled before being split to ensure the sets are unbiased. The number of tweets in each set after applying data splitting are as follows.

- The number of tweets in the dataset is 688,881.
- The number of tweets in the training set is 385,772.
- The number of tweets in the validation set is 96,444.
- The number of tweets in the test set is 206,665.

The distribution for each target class of the NG-StV_Senti dataset is given in Figure 3.3. This shows that the highest number of tweets are neutral, i.e. 226,746. The NG-StV_Senti dataset contains 169,769 tweets that are strongly positive and 131,697 that are weakly positive. The number of negative tweets is lower than the number of positive and neutral tweets, i.e. 86,684 are weakly negative and 73,985 are strongly negative.

3.2.3 Text Preprocessing

In Section 2.4.3, data preprocessing was discussed. The dataset contained urls, hashtags, username handles, noise, anomalies, different languages and other unnecessary data. Thus, it was cleaned, filtered and processed to generate clean text. Stretched words were also processed and reduced to their root words. Text preprocessing is then applied to transform the text data into a structured format suitable for analysis, involving steps such as text normalization and tokenization.

3.2.3.1 Text Normalization

Text normalization involves transforming text data to a standard format. The major advantage of text normalization is to reduce the overhead of processing multiple

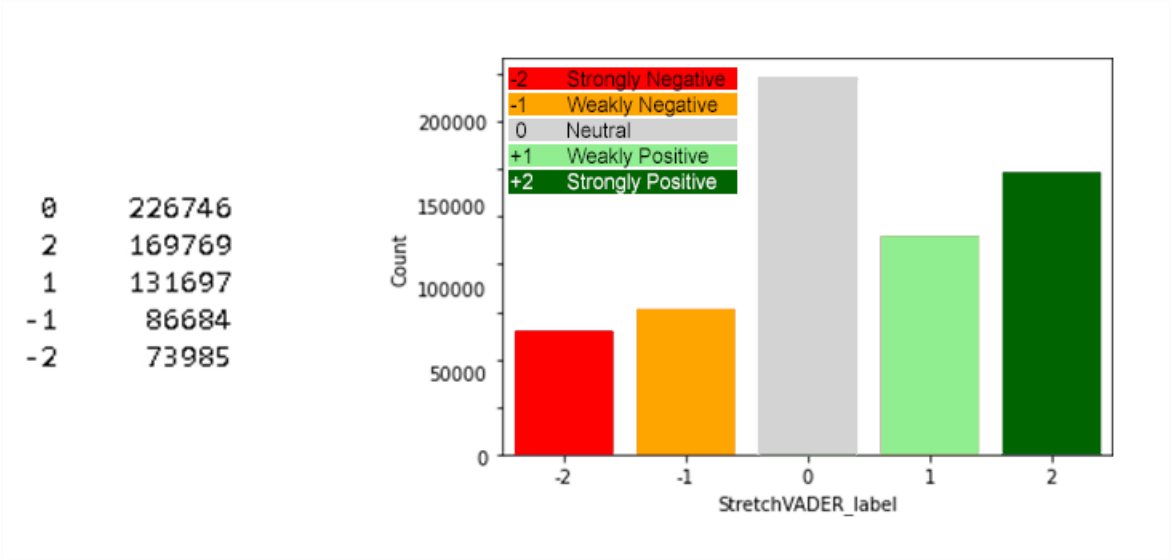


Figure 3.3: Target class distribution for the NG-StV_Senti dataset.

forms of the same words [53]. There are two methods of text normalization, stemming and lemmatization. In the proposed framework, lemmatization is used instead of stemming because lemmatization reduces words to their base form considering the language dictionary and context while stemming does not consider context and removes characters from the words which may result in incorrect meanings and spelling errors. In lemmatization, the base form of a set of words is known as the lemma. The steps for text normalization are as follows.

1. Stop words are removed using the Python Natural Language ToolKit (NLTK) library, i.e. determiners (the, a, an, another), coordinating conjunctions (for, and, but, nor, neither, nor, or, yet) and prepositions (in, on, under, before).

2. The open-source WordNet lemmatization library is used to reduce the words to their lemmas.

3.2.3.2 Tokenization

Tokenization is the process of extracting smaller units from the text called tokens. A token can be a sentence, word, sub-word, or character. Simple tokenization can be done using a delimiter, e.g. a space or punctuation. In NLP, it is also used to create a vocabulary from a corpus by extracting unique tokens or frequently occurring tokens. Here, word-level tokenization is applied on the 3 data inputs (normalized text,

SVS and smileys), separately and then combined to form a larger tokenized array as described below.

- Normalized text is tokenized using space as a delimiter and the result is a list of words.
- SVS is a single value. Thus, no tokenization is required.
- Smileys are tokenized using space as a delimiter and the result is a list of smileys.

3.2.4 Data Encoding

The text is next converted into a form that can be processed by machines. In NLP, various techniques are used to convert text data into numbers. The following data encoding techniques are used here in combination with the ML algorithms to determine which combinations perform better.

3.2.4.1 Term Frequency Inverse Document Frequency (TF-IDF)

Term Frequency-Inverse Document Frequency (TF-IDF) is used to determine the importance of a term in a collection of documents. A document is a unit of text, such as a tweet, product review, article or webpage and a term is a single word or combination of words in a document. Term Frequency (TF) is the frequency of a term within a document, indicating how often it appears in relation to other words in the document. Inverse Document Frequency (IDF) is a measure of how significant a term is across an entire collection of documents. It quantifies the rarity of a term by considering the ratio of the total number of documents to the number of documents containing the term. A sparse matrix is a matrix with a large number of zero elements relative to the total number of elements.

TF-IDF is the term frequency multiplied by the inverse document frequency. It reflects the importance of a term in a document relative to its importance in the document corpus. TF-IDF is applied on the tokenized data to generate a sparse matrix of TF-IDF features.

- There are 385,772 documents and 129,396 terms in the TF-IDF training set feature matrix, so the dimensions of the TF-IDF feature matrix for the training set are 385,772 $\times$ 129,396.

- There are 96,444 documents and 129,396 terms in the TF-IDF validation set feature matrix, so the dimensions of the TF-IDF feature matrix for validation set are 96,444 × 129,396.
- There are 206,665 documents and 129,396 terms in the TF-IDF test set feature matrix, so the dimensions of the TF-IDF feature matrix for test set are 206,665 × 129,396.

3.2.4.2 Word Embedding

Word embedding is a technique employed to convert words into numerical representations suitable for input to ML algorithms. Each word is transformed into a dense vector consisting of numerical values. These vectors capture characteristics of words including semantic and contextual information. Words that are closer in the vector space have similar meaning. These vectors enable the detection of similarities between words. Commonly used word embedding models include Word2Vec, GloVe, Flair embeddings, BERT, fastText and Gensim [54]. Semantics refers to the meaning and interpretation of words within a given context while context is the surrounding words and their influence on the meaning of a particular word.

Word2Vec (W2V)

Word2Vec (W2V) is a word embedding technique that is used to convert text data into numerical vectors and combine the vectors having similar meaning. In this thesis, W2V has been applied on the tokenized data in order to transform words into vectors [54].

- There are 385,772 documents and 100 word-embedded vectors in the W2V training set feature matrix, so the dimensions of the W2V feature matrix for the training set are 385,772 × 100 as shown in Figure 3.4.
- There are 96,444 documents and 100 word-embedded vectors in the W2V validatin set feature matrix, so the dimensions of the W2V feature matrix for the validation set are 96,444 × 100 as shown in Figure 3.5.
- There are 206,665 documents and 100 word-embedded vectors in the W2V test set feature matrix, so the dimensions of the W2V feature matrix for the test set are 206,665 × 100 as shown in Figure 3.6.

```
array([[-0.7251954 ,  0.8792501 , -0.14927697, ..., -0.3581486 ,
         0.6511727 , -0.20691629],
       [-0.53956413,  1.6246858 ,  0.03992929, ..., -1.0249279 ,
         1.024131  , -0.21681824],
       [-0.8351458 ,  0.61087346,  0.3189466 , ..., -0.9683943 ,
         0.89668316,  0.16516005],
       ...,
       [-0.26030686,  0.7583959 ,  0.1152641 , ..., -1.0294359 ,
         0.2045738 ,  0.31819487],
       [-1.3510176 ,  0.8152084 , -0.04826832, ..., -1.2174158 ,
         0.71285766,  0.64893514],
       [-1.0730107 ,  0.84474266,  0.18567902, ..., -0.01610129,
         0.7990663 , -0.01700371]], dtype=float32)
```

Figure 3.4: Training set W2V feature matrix

```
array([[-0.6599701 ,  1.101579  ,  0.42392012, ..., -0.4773441 ,
         1.3197378 ,  0.6482947 ],
       [-1.1378812 ,  0.17858484,  0.15480857, ..., -0.70273095,
         0.7583524 ,  0.2530216 ],
       [-0.7924258 ,  0.5475055 , -0.13528886, ..., -0.39777648,
         0.97946906,  0.05921941],
       ...,
       [-0.86455005,  0.39087525,  0.14060414, ..., -0.6055637 ,
         0.6475664 ,  0.24009968],
       [-0.6485841 ,  0.48885295,  0.1895776 , ..., -0.84172827,
         0.6326868 ,  0.22236781],
       [-0.83235836,  1.0436615 ,  0.8297693 , ..., -1.2931612 ,
        -0.04461909,  0.30315363]], dtype=float32)
```

Figure 3.5: Validation set W2V feature matrix

```
array([[-0.99310434,  1.3870375 ,  0.10383858, ..., -0.43559602,
         1.3685472 , -0.28972554],
       [-0.4675504 , -0.08811706,  0.02100156, ..., -0.16683593,
         0.788203  , -0.6055704 ],
       [-0.705775  ,  0.72105104,  0.46277076, ..., -0.52957886,
         0.7044777 , -0.25794384],
       ...,
       [-0.6213321 ,  0.85850793,  0.45872557, ..., -0.66179395,
         0.7412517 ,  0.44783318],
       [-1.1588755 ,  0.7389167 , -0.04580651, ..., -0.4824406 ,
         0.66479063,  0.10811888],
       [-0.81381   ,  0.6320487 ,  0.27406174, ..., -1.029807  ,
         0.5172934 ,  0.4314326 ]], dtype=float32)
```

Figure 3.6: Test set W2V feature matrix

3.2.5 Model Training, Validation and Testing

Model training, validation and testing is the process of training an ML algorithm on labeled data, evaluating its performance and determining how effective the model is with new and unseen data. In this thesis, each ML algorithm is trained, validated and

tested with two data encodings, i.e. frequency-based TF-IDF and neural network-based W2V in order to determine which combinations perform better. The ML algorithms considered are Multinomial Logistic Regression (MLR), Multinomial Naïve Bayes (MNB), K-Nearest Neighbor (KNN), Decision Trees (DT), Random Forest (RF) and XGBoost. These algorithms are presented in the next section.

3.3 Machine Learning Algorithms

3.3.1 Multinomial Logistic Regression

MLR is a statistical technique used to model the relationship between dependent variables, i.e. outputs, and independent variables, i.e. inputs. In MLR, there are multiple but mutually exclusive and categorical (discrete numbers or text) outputs while there can be one or more continuous (numeric) or categorical inputs [55]. In this thesis, the output contains multiple, mutually exclusive and categorical values, i.e. multiple classes.

3.3.2 Multinomial Naïve Bayes

MNB is a supervised ML classification algorithm that is used for multi-class classification problems. It is based on Bayes Theorem. It calculates the probability of a data point, i.e. tweet belonging to a specific output, i.e. class. It is called naïve because it assumes that the effect of a feature of a tweet on a given class is independent of the other features of the same tweet [56]. It is widely used in the document classification, spam detection and sentiment analysis. In this thesis, MNB is used because the problem is multi-class classification.

3.3.3 K-Nearest Neighbors

The KNN algorithm is a commonly used supervised ML algorithm for both regression and classification problems. This algorithm classifies new data based on the information provided by its neighbors. When classifying a new data instance, KNN considers the class labels of its K nearest neighbors in the feature space. The distances (usually Euclidean distances) between the new data and its K neighbors are calculated. For classification, a majority vote between the class labels of its K nearest neighbors is

conducted. The new data point is assigned with the most frequently occurring class of its K neighbors [57].

3.3.4 Decision Tree

DT is a supervised ML algorithm that employs rule-based learning. It is also known as the Classification and Regression Tree (CART) algorithm. The dataset features are split into a tree-like structure where each internal node represents a feature, each branch represents a decision and each leaf node represents the final class label or numerical value for that specific data point [58–61].

3.3.5 Random Forest

RF is a supervised ML algorithm used for both classification and regression. It is an ensemble learning method which means it employs multiple classifiers to solve complex problems and improve overall performance. RF employs a technique known as bagging which uses multiple decision trees on different subsets of the dataset called bootstraps. For classification, the class assigned to a data point is determined by the majority of trees, but for regression, the output is the average of the predictions from the decision trees. RF overcomes the problem of overfitting so the model should perform better when applied to unseen data.

3.3.6 XGBoost

Boosting is an ensemble learning method that is used in ML to make strong classifier from weak classifiers by combining the weak classifiers in series. The Gradient Boosting Decision Tree (GBDT) algorithm uses a series of models (trees) trained on the residual errors of their predecessor models (trees) until the residual error is reduced or a specified number of models (trees) are reached, as determined by the hyperparameters. The Extreme Gradient Boosting algorithm (XGBoost) is a scalable gradient boosting algorithm that employs parallel computation to build decision trees simultaneously, resulting in faster training and improved performance. Thus, in this thesis, XGBoost is used.

Chapter 4

Performance Metrics and Evaluation

This chapter presents and discusses performance of the ML algorithms, i.e. MLR, MNB, KNN, DT, RF and XGBoost with two data encodings, i.e. TF-IDF and W2V. The objective is to determine the most effective combination of ML algorithm and data encoding.

4.1 Performance Metrics

Several performance metrics are used to evaluate the ML algorithms. These metrics are used to adjust the hyperparameters. The performance metrics used here are given below.

Execution time is the computation time to perform training, validation and testing.

A confusion matrix presents the possible outcomes based on the predicted class and actual class. It is an $n \times n$ square matrix where n is the number of classes. For two classes, it is a 2×2 matrix. In this thesis, the number of classes is 5, so the confusion matrix has dimensions 5×5 as shown in Table 4.1. The parameters True Positive (TP), False Positive (FP), True Negative (TN) and False Negative (FN) for multi-class classification are defined as follows.

1. True Positive (TP) for a class is the number of predicted values that are the same as the actual class.

2. False Positive (FP) for a class is the sum of the values in the corresponding column excluding the TP value.

3. False Negative (FN) for a class is the sum of the values in the corresponding class row excluding the TP value.

4. True Negative (TN) for a class is the sum of the values in all the columns and rows excluding the corresponding class row and column which means excluding the TP, FP and FN values.

In Table 4.1, each row represents the actual class, each column represents the predicted class and SN, WN, N, WP and SP are the five classes. TP_{SN}, TP_{WN}, TP_{N}, TP_{WP} and TP_{SP} are the true positive values for each class, repectively. $E_{Actual-Predicted}$ is the misclassification value, e.g. E_{SN-WN} is the value when the actual class is SN but it is misclassified as WN.

		Predicted				
		SN	**WN**	**N**	**WP**	**SP**
Actual	**SN**	TP_{SN}	E_{SN-WN}	E_{SN-N}	E_{SN-WP}	E_{SN-SP}
	WN	E_{WN-SN}	TP_{WN}	E_{WN-N}	E_{WN-WP}	E_{WN-SP}
	N	E_{N-SN}	E_{N-WN}	TP_{N}	E_{N-WP}	E_{N-SP}
	WP	E_{WP-SN}	E_{WP-WN}	E_{WP-N}	TP_{WP}	E_{WP-SP}
	SP	E_{SP-SN}	E_{SP-WN}	E_{SP-N}	E_{SP-WP}	TP_{SP}

Table 4.1: The 5×5 confusion matrix for five classes.

Precision is the percentage of correct positive class predictions. It is the ratio of TP to the number of positive predictions, i.e. sum of TP and FP

$$Precision = \frac{TP}{(TP + FP)} \tag{4.1}$$

Recall is the ratio of TP to the sum of TP and FN

$$Recall = \frac{TP}{(TP + FN)} \tag{4.2}$$

Accuracy is the ratio of correct classifications to all classifications

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{4.3}$$

F1-score is the harmonic mean of recall and precision. It emphasizes smaller values by taking reciprocals which increases their impact on the result. The F1-score is calculated as

$$F1_{score} = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} \tag{4.4}$$

$$= \frac{2TP}{2TP + FP + FN} \tag{4.5}$$

4.2 Performance of the ML Algorithms

This section presents the performance of the ML algorithms with data encoded using TF-IDF and W2V.

4.2.1 Performance of MLR with TF-IDF and W2V

The performance of MLR with TF-IDF and W2V data encoding is presented in Tables 4.2 and 4.3. Table 4.2 gives the execution times for training, validation and testing using MLR with TF-IDF and W2V. Table 4.3 gives the validation and testing results using MLR with TF-IDF and W2V data encoding. Figure 4.1 shows the confusion matrices using MLR with TF-IDF and W2V data encoding. With TF-IDF data encoding, MLR takes 42.0 s for training and 0.29 s for validation and has a 90.7% validation accuracy. For testing, MLR takes 0.55 s and has a test accuracy of 90.8%. With W2V data encoding, MLR takes 107 s for training and 0.11 s for validation and has an 82.8% validation accuracy. For testing, MLR takes 0.23 s and has a test accuracy of 82.9%. To further assess the performance of MLR with TF-IDF and W2V data encoding, the first 500 tweets from the validation and test sets were chosen. The actual and predicted classes for these tweets visualized through scatter and line plots are shown in Figures 4.2, 4.3, 4.4 and 4.5. Blue denotes the actual class while red denotes the predicted class. Some data points and curves are not overlapped which means these are misclassifications. These results show that MLR performs better with TF-IDF.

	Training Time	**Validation Time**	**Testing Time**
TF-IDF	42.0 s	0.29 s	0.55 s
W2V	107 s	0.11 s	0.23 s

Table 4.2: Execution times for training, validation and testing using MLR with TF-IDF and W2V.

	MLR (TF-IDF)		**MLR (W2V)**	
	Validation	**Testing**	**Validation**	**Testing**
Accuracy	90.7%	90.8%	82.8%	82.9%
Precision	90.6%	90.7%	82.6%	82.2%
Recall	90.7%	90.8%	82.8%	82.4%
F1-score	90.6%	90.7%	82.5%	82.1%

Table 4.3: Validation and testing results using MLR with TF-IDF and W2V.

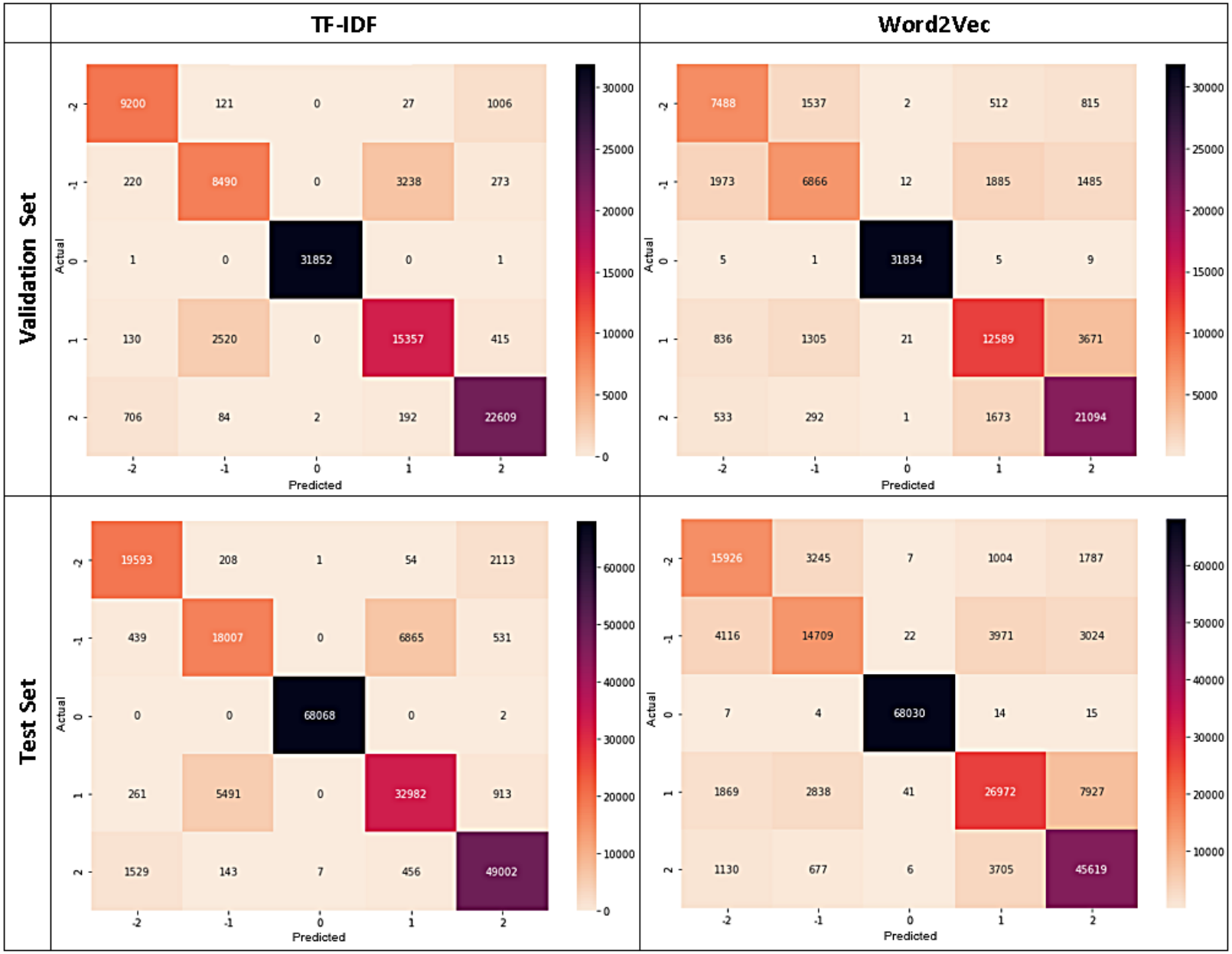


Figure 4.1: Confusion matrices for validation and testing using MLR with TF-IDF and W2V.

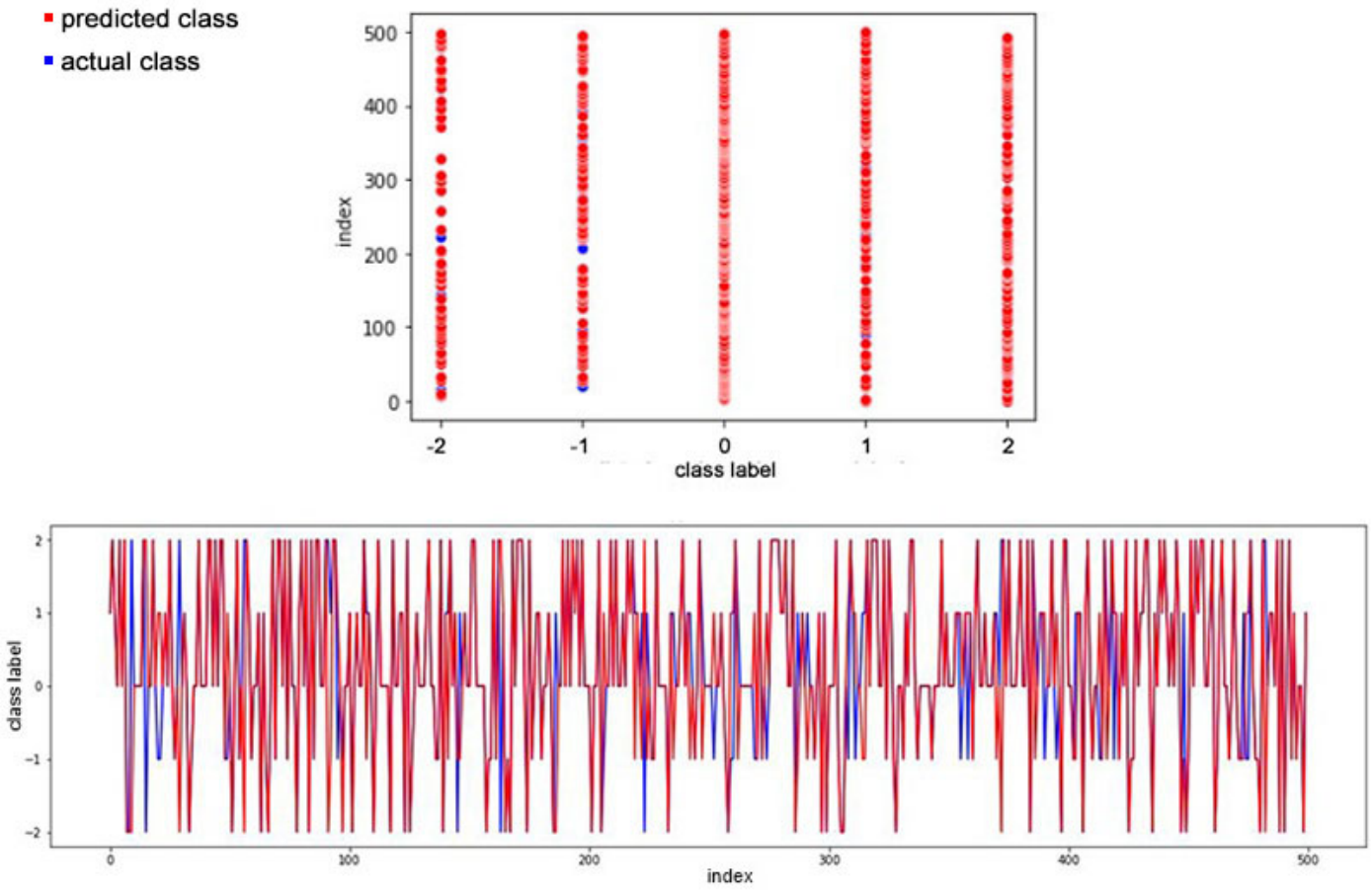


Figure 4.2: Overlapped scatter and line plots for the first 500 tweets from the validation set, showing actual and predicted classes using MLR with TF-IDF data encoding.

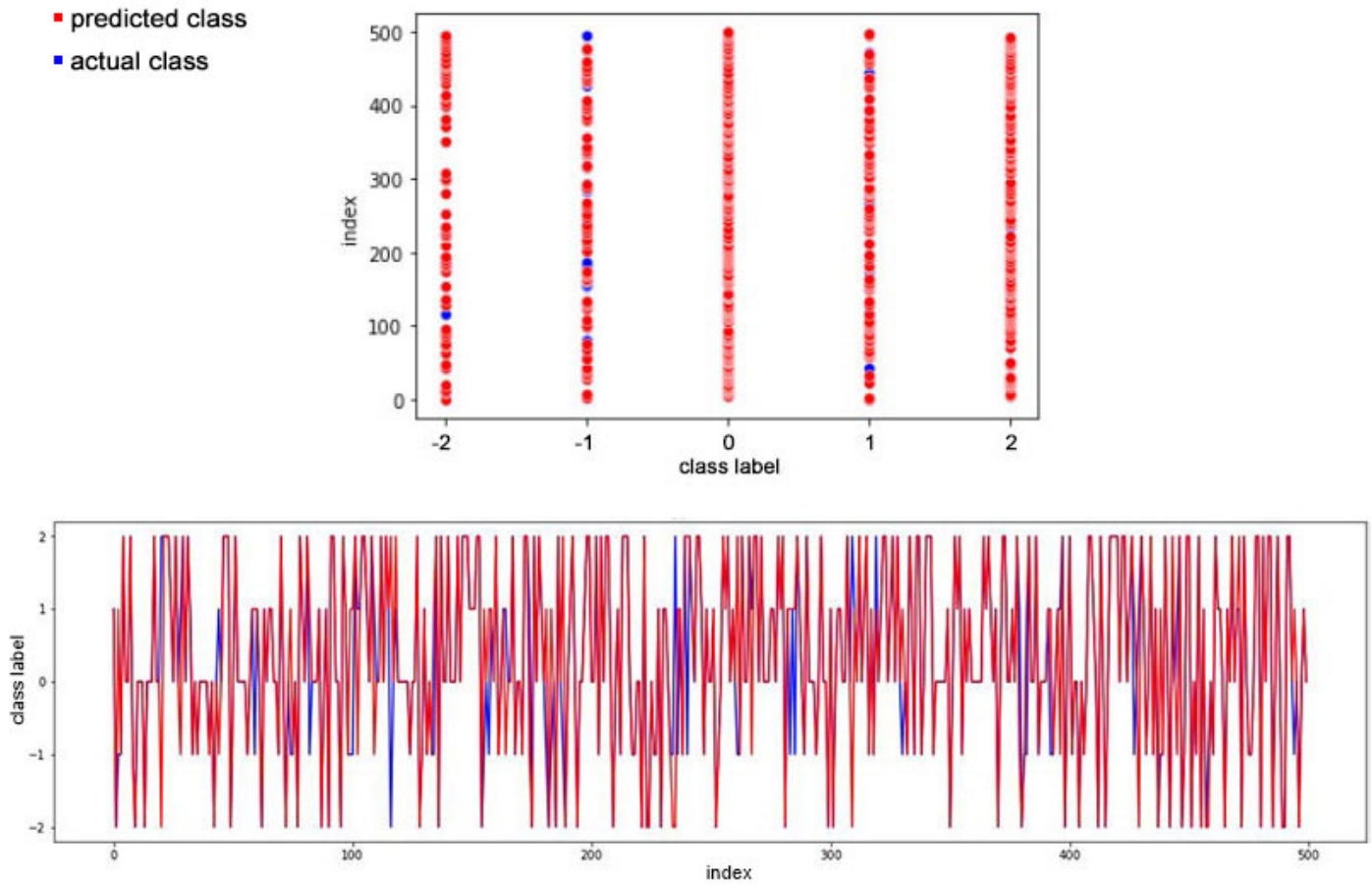


Figure 4.3: Overlapped scatter and regression plots for the first 500 tweets from the test set, showing actual and predicted classes using MLR with TF-IDF data encoding.

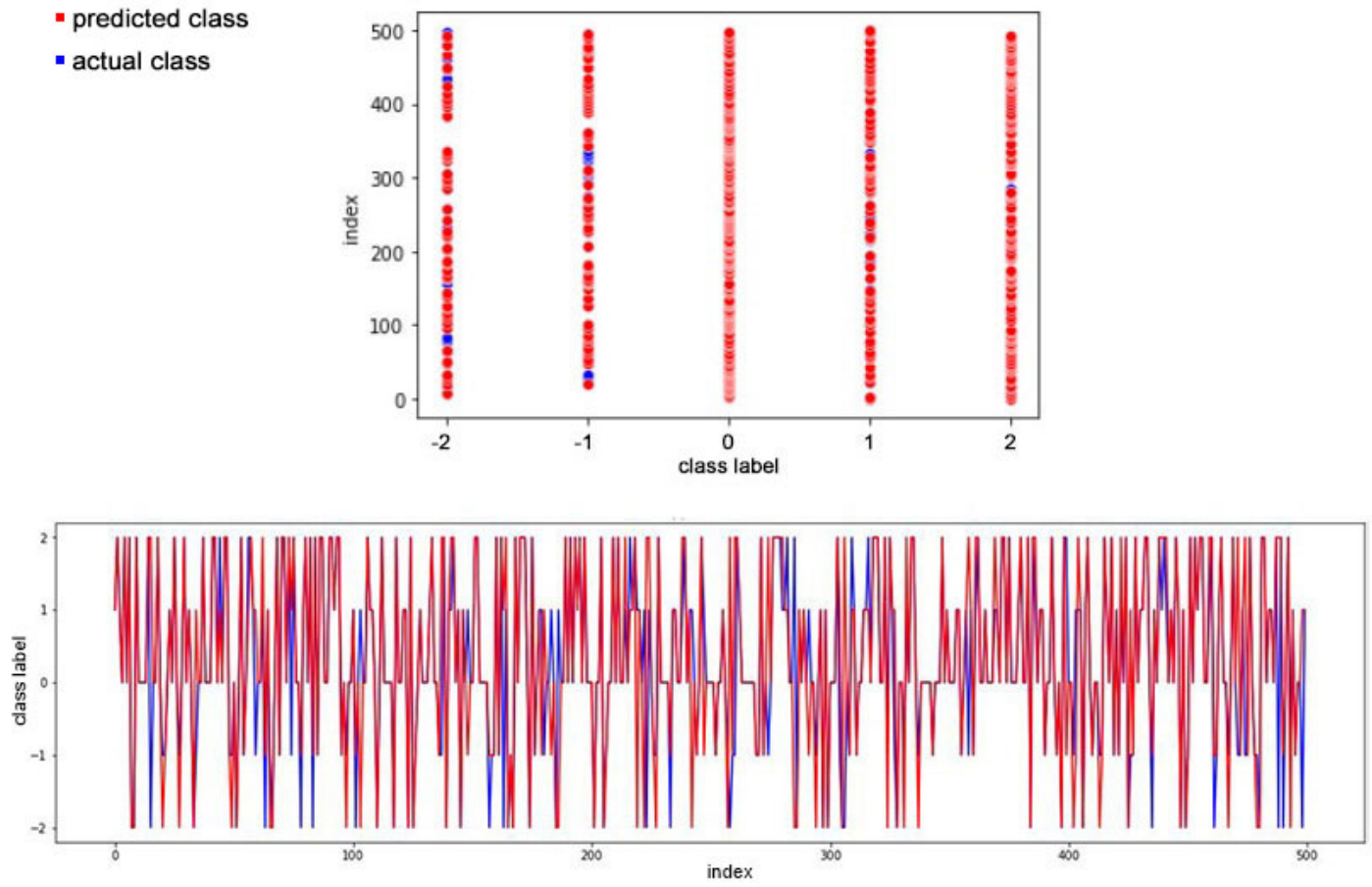


Figure 4.4: Overlapped scatter and regression plots for the first 500 tweets from the validation set, showing actual and predicted classes using MLR with W2V data encoding.

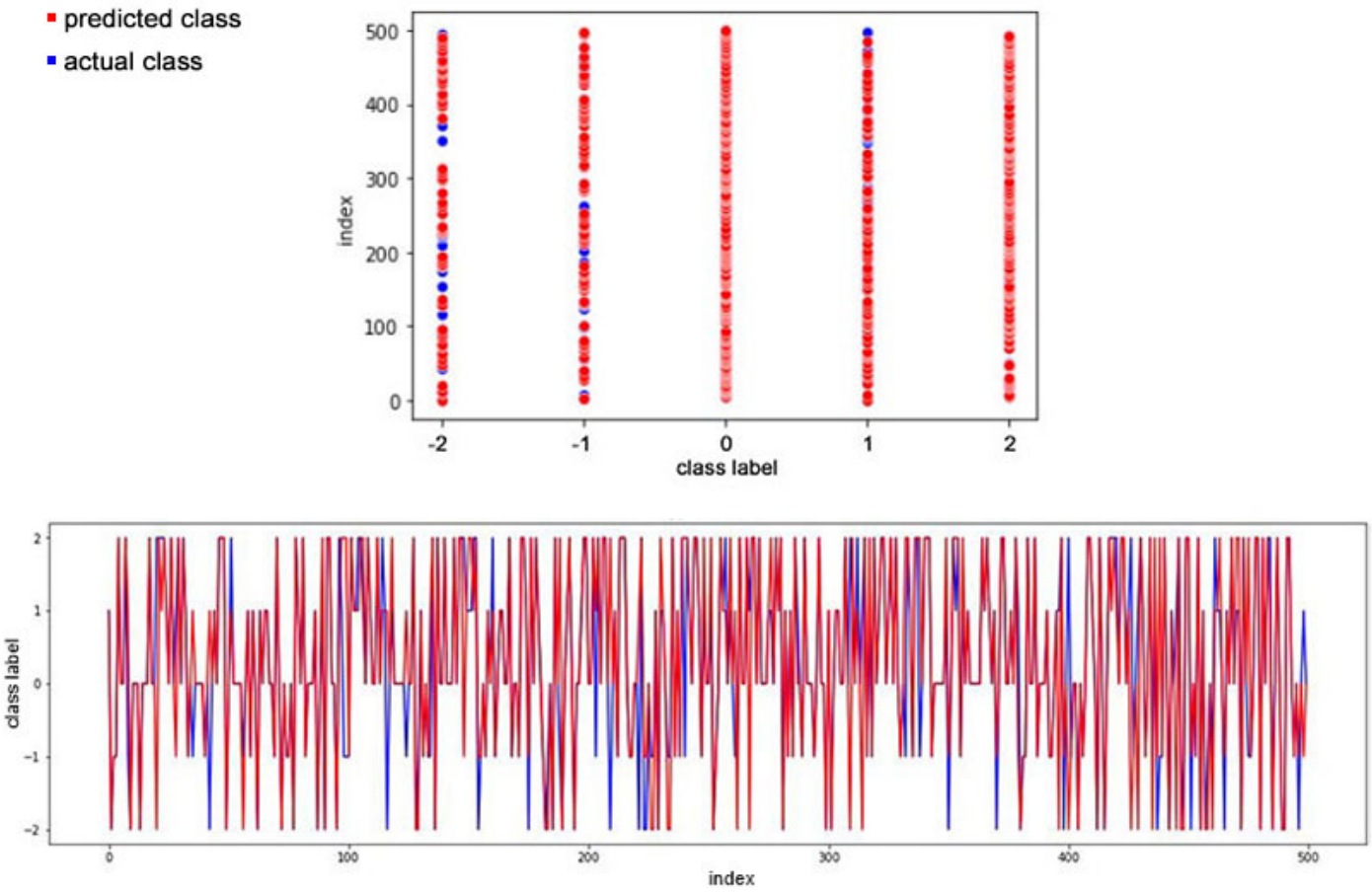


Figure 4.5: Overlapped scatter and regression plots for the first 500 tweets from the test set, showing actual and predicted classes using MLR with W2V data encoding.

4.2.2 Performance of MNB with TF-IDF and W2V

The performance of MNB with TF-IDF and W2V data encoding is presented in Tables 4.4 and 4.5. Table 4.4 gives the execution times for training, validation and testing using MNB with TF-IDF and W2V. Table 4.5 gives the validation and testing results using MNB with TF-IDF and W2V data encoding. Figure 4.6 shows the confusion matrices using MNB with TF-IDF and W2V data encoding. With TF-IDF data encoding, MNB takes 0.18 s for training and 0.24 s for validation and has an 81.2% validation accuracy. For testing, MLR takes 0.48 s and has a test accuracy of 81.4%. With W2V data encoding, MLR takes 0.59 s for training and 0.12 s for validation and has a 49.0% validation accuracy. For testing, MLR takes 0.21 s and has a test accuracy of 49.1%. These results show that MNB performs better with TF-IDF.

	Training Time	**Validation Time**	**Testing Time**
TF-IDF	0.18 s	0.24 s	0.48 s
W2V	0.59 s	0.12 s	0.21 s

Table 4.4: Execution times for training, validation and testing using MNB with TF-IDF and W2V.

	MNB (TF-IDF)		**MNB (W2V)**	
	Validation	**Testing**	**Validation**	**Testing**
Accuracy	81.2%	81.4%	49.0%	49.1%
Precision	82.5%	82.7%	82.6%	41.5%
Recall	81.2%	81.4%	49.0%	49.1%
F1-score	78.6%	78.9%	37.0%	37.1%

Table 4.5: Validation and testing results using MNB with TF-IDF and W2V.

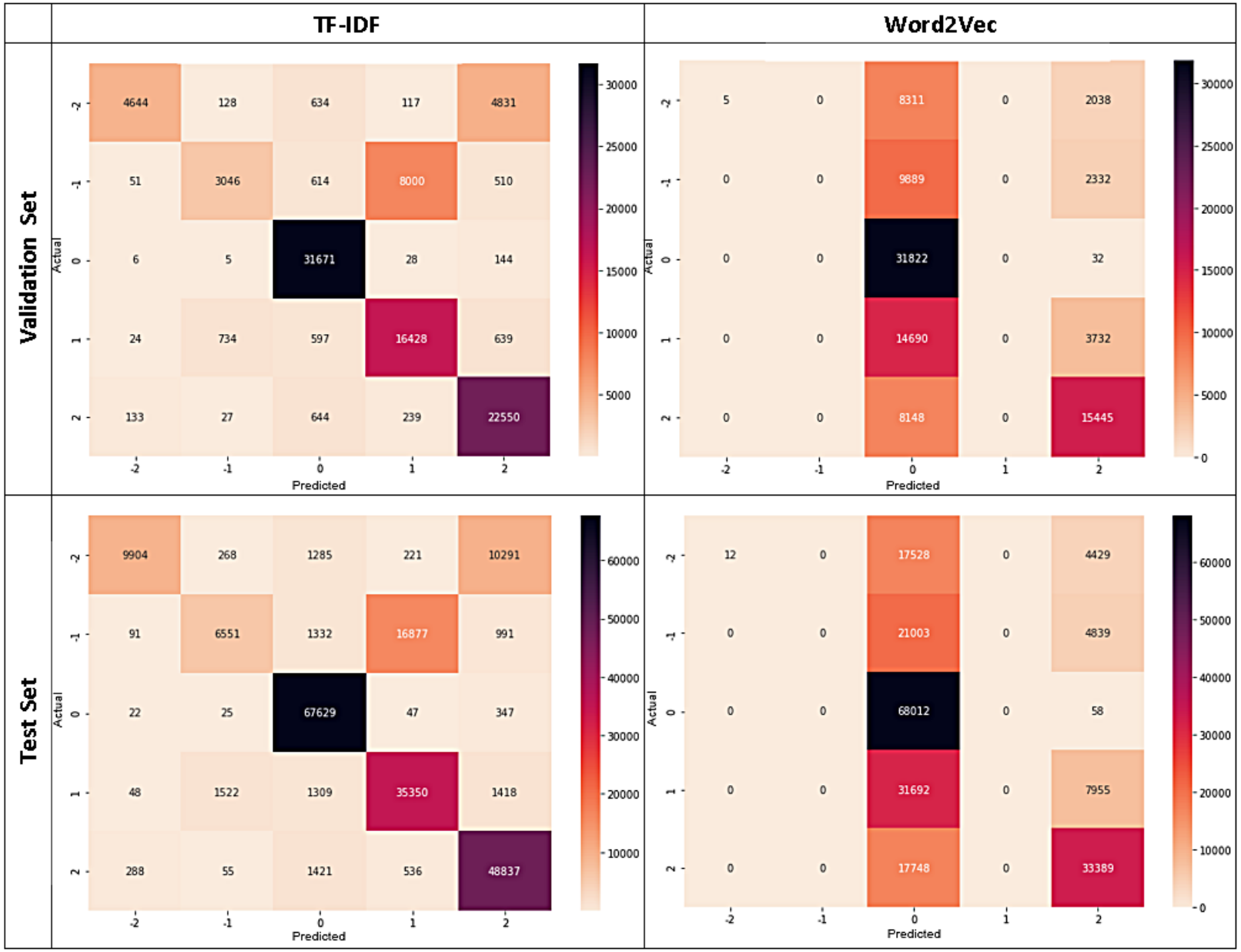


Figure 4.6: Confusion matrices for validation and testing using MNB with TF-IDF and W2V.

4.2.3 Performance of KNN with $K = 2$, TF-IDF and W2V

The performance of KNN with $K = 2$, TF-IDF and W2V data encoding is presented in Tables 4.6 and 4.7. Table 4.6 gives the execution times for training, validation and testing using KNN with $K = 2$, TF-IDF and W2V. Table 4.7 gives the validation and testing results using KNN with $K = 2$, TF-IDF and W2V data encoding. While Figure 4.8 shows the confusion matrices using KNN with $K = 2$, TF-IDF and W2V data encoding. With the TF-IDF data encoding and $K = 2$, KNN takes 0.18 s for training and 767 s for validation and has a 58.1% validation accuracy. For testing, KNN with $K = 2$ and TF-IDF takes 1,583 s and has a test accuracy of 58.1%. With W2V data encoding and $K = 2$, KNN takes 0.84 s for training and 7,593 s for validation and has a 73.8% validation accuracy. For testing, KNN with $K = 2$ and W2V takes 16,305 s and has a test accuracy of 73.8%. These results show that KNN with $K = 2$ and W2V performs better.

	Training Time	**Validation Time**	**Testing Time**
TF-IDF	0.18 s	767 s	1,583 s
W2V	0.84 s	7,593 s	16,305 s

Table 4.6: Execution times for training, validation and testing using KNN with $K = 2$, TF-IDF and W2V.

	KNN with $K = 2$ (TF-IDF)		**KNN with $K = 2$ (W2V)**	
	Validation	**Testing**	**Validation**	**Testing**
Accuracy	58.1%	58.1%	73.8%	73.8%
Precision	66.4%	66.8%	75.8%	76.0%
Recall	58.1%	58.1%	73.8%	73.8%
F1-score	55.9%	56.0%	73.9%	74.0%

Table 4.7: Validation and testing results using KNN with $K = 2$, TF-IDF and W2V.

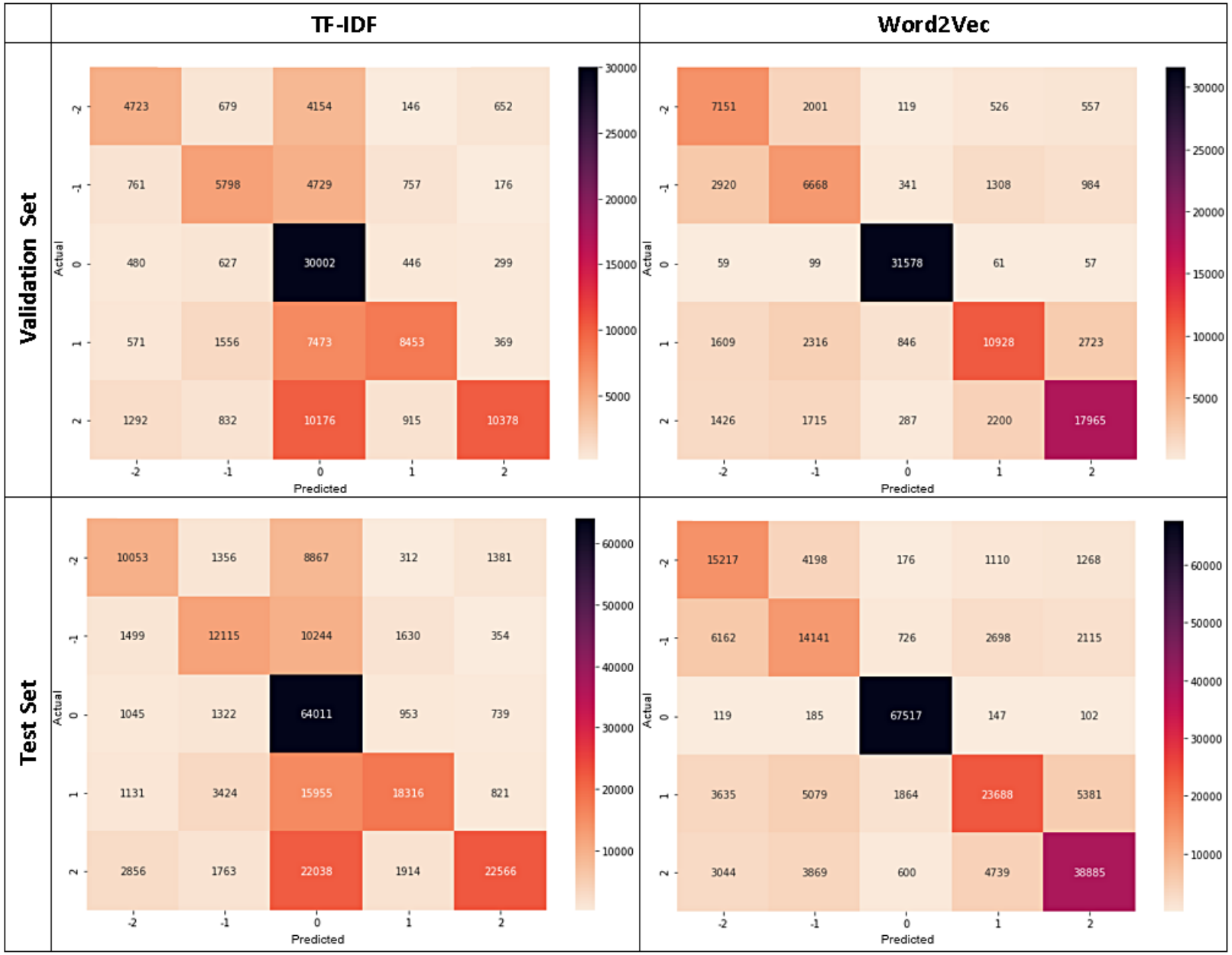


Figure 4.7: Confusion matrices for validation and testing using KNN with $K = 2$, TF-IDF and W2V.

4.2.4 Performance of KNN with $K = 3$, TF-IDF and W2V

The performance of KNN with $K = 3$, TF-IDF and W2V data encoding is presented in Tables 4.8 and 4.9. Table 4.8 gives the execution times for training, validation and testing using KNN with $K = 3$, TF-IDF and W2V. Table 4.9 gives the validation and testing results using KNN with $K = 3$, TF-IDF and W2V data encoding. While Figure 4.8 shows the confusion matrices using KNN with $K = 3$, TF-IDF and W2V data encoding. With the TF-IDF data encoding and after changing the number of neighbors from $K = 2$ to $K = 3$, the accuracy improved from 58.1% to 61.5% with the increase in total execution time from 2,350 s to 5,705 s. While with W2V, the accuracy changes from 73.8% to 77.1% with the increase in total execution time from 23,899 s (6.6 hr) to 57,698 s (16 hr). These results show that KNN with $K = 3$ and W2V performs better. It is also observed that it is not worth increasing K above 3 because it will take much more time to achieve accuracy higher than the best performing algorithm, i.e. MLR.

	Training Time	**Validation Time**	**Testing Time**
TF-IDF	0.54 s	1,812 s	3,893 s
W2V	25.5 s	17,830 s	39,842 s

Table 4.8: Execution times for training, validation and testing using KNN with $K = 3$, TF-IDF and W2V.

	KNN with $K = 3$ (TF-IDF)		**KNN with $K = 3$ (W2V)**	
	Validation	**Testing**	**Validation**	**Testing**
Accuracy	61.5%	61.5%	77.0%	77.1%
Precision	68.1%	68.2%	77.4%	77.7%
Recall	61.5%	61.5%	77.0%	77.1%
F1-score	60.1%	60.1%	77.0%	77.2%

Table 4.9: Validation and testing results using KNN with $K = 3$, TF-IDF and W2V.

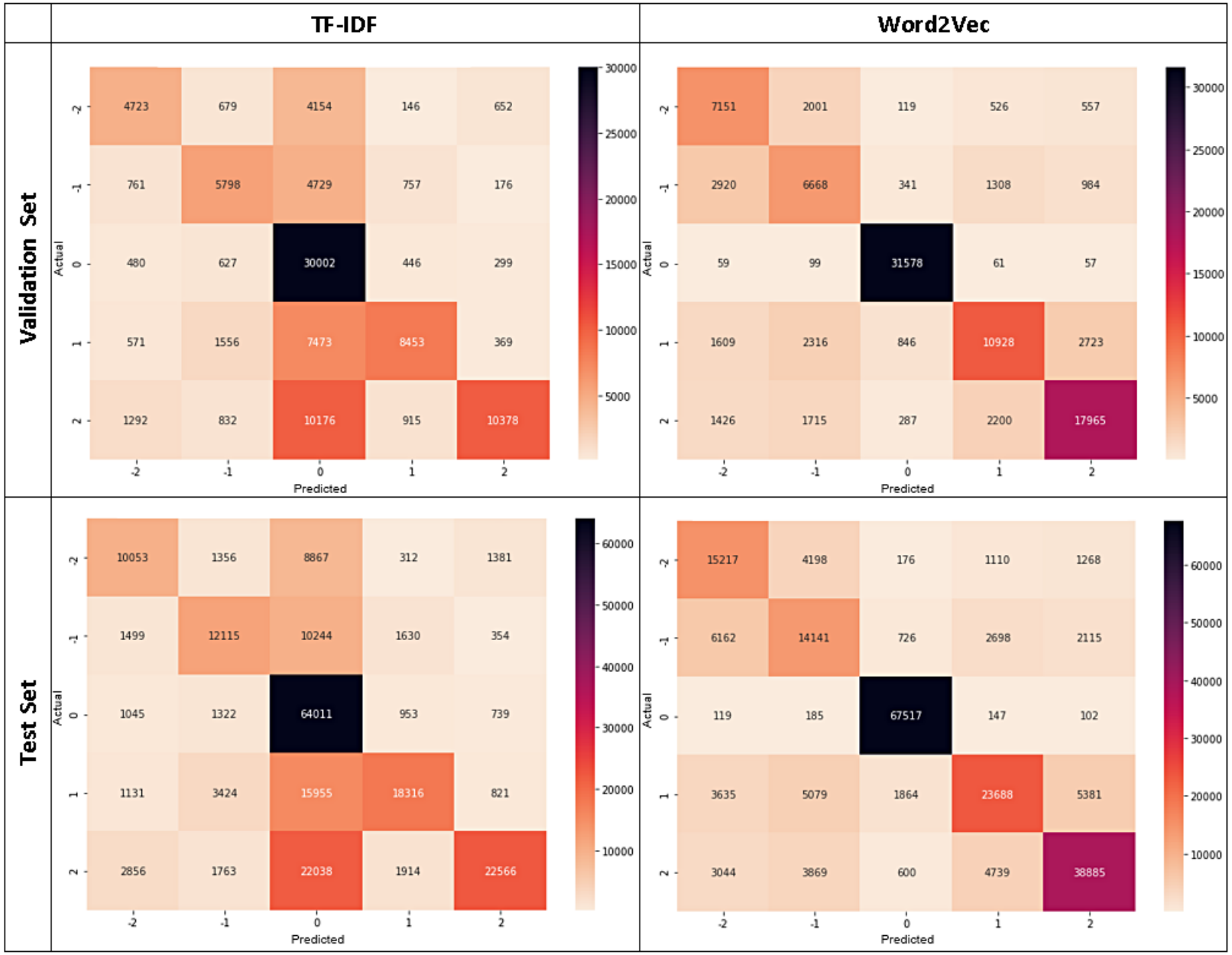


Figure 4.8: Confusion matrices for validation and testing using KNN with $K = 3$, TF-IDF and W2V.

4.2.5 Performance of DT with TF-IDF and W2V

The performance of DT with TF-IDF and W2V data encoding is presented in Tables 4.10 and 4.11. Table 4.10 gives the execution times for training, validation and testing using DT with TF-IDF and W2V. Table 4.11 gives the validation and testing results using DT with TF-IDF and W2V data encoding. Figure 4.9 shows the confusion matrices using DT with TF-IDF and W2V data encoding. With TF-IDF data encoding, DT takes 491 s for training and 0.11 s for validation and has an 87.5% validation accuracy. For testing, DT takes 0.20 s and has a test accuracy of 87.5%. With W2V data encoding, DT takes 136 s for training and 0.64 s for validation has a 68.3% validation accuracy. For testing, DT takes 0.14 s and has a test accuracy of 68.1%. The results show that DT performs better with TF-IDF.

	Training Time	**Validation Time**	**Testing Time**
TF-IDF	491 s	0.11 s	0.20 s
W2V	136 s	0.64 s	0.14 s

Table 4.10: Execution times for training, validation and testing using DT with TF-IDF and W2V.

	DT (TF-IDF)		**DT (W2V)**	
	Validation	**Testing**	**Validation**	**Testing**
Accuracy	87.5%	87.5%	68.3%	68.1%
Precision	87.4%	87.4%	68.2%	68.5%
Recall	87.5%	87.5%	68.3%	68.1%
F1-score	87.4%	87.4%	68.3%	68.4%

Table 4.11: Validation and testing results using DT with TF-IDF and W2V.

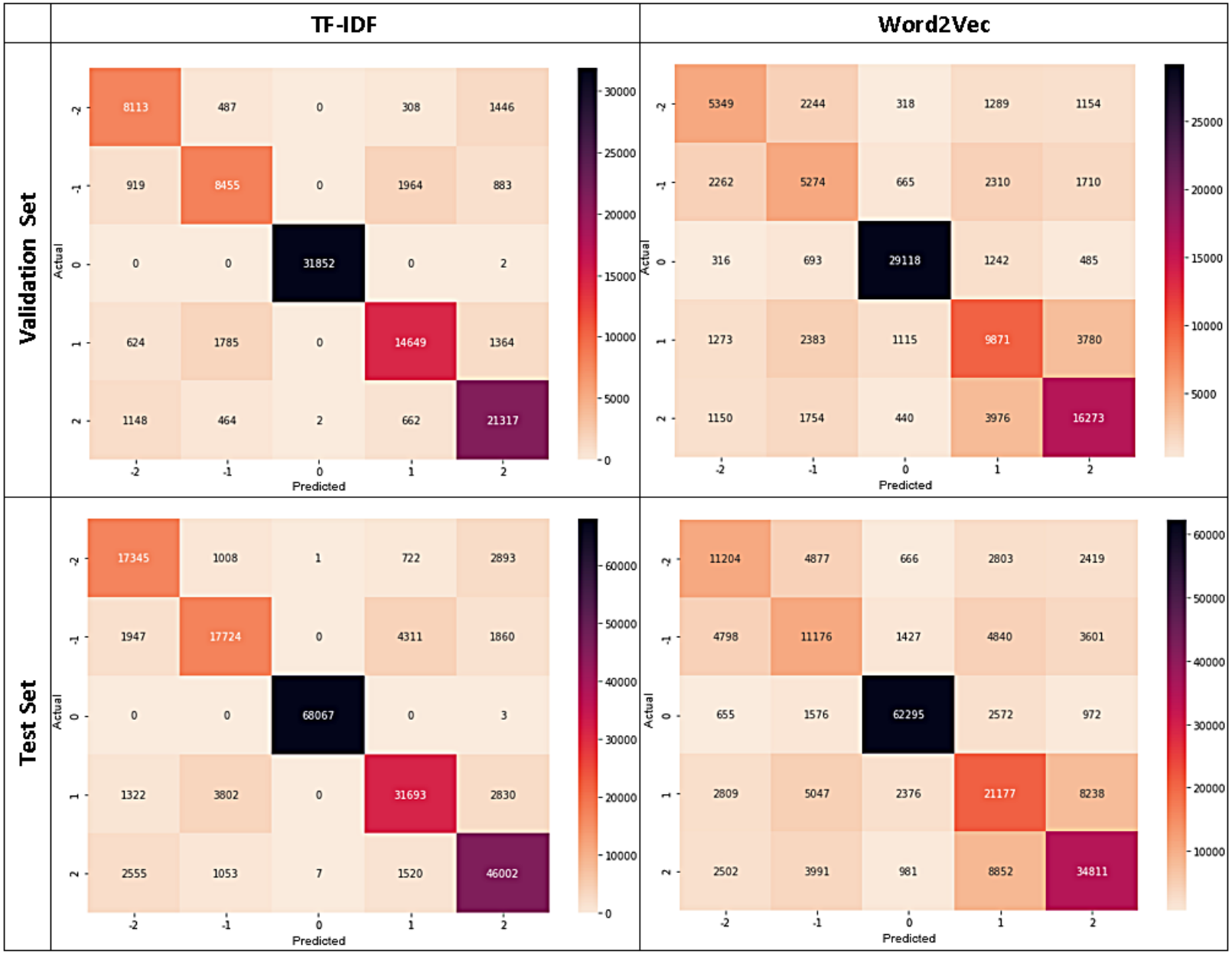


Figure 4.9: Confusion matrices for validation and testing using DT with TF-IDF and W2V.

4.2.6 Performance of RF with 100 Trees, TF-IDF and W2V

As discussed in the Section 3.3.5, the RF employs ensemble learning, which means it is a combination of more than one classifier model and has more than one decision tree. RF has $trees = 100$ as the default parameter in the Scikit-learn library. The performance of RF with $trees = 100$, TF-IDF and W2V data encoding is considered and presented in Tables 4.12 and 4.13. Table 4.12 gives the execution times for training, validation and testing using RF with $trees = 100$, TF-IDF and W2V. Table 4.13 gives the validation and testing results using RF with $trees = 100$, TF-IDF and W2V data encoding. Figure 4.10 shows the confusion matrices using RF with $trees = 100$, TF-IDF and W2V data encoding. With TF-IDF data encoding and $trees = 100$, RF takes 5,845 s for training and 22.1 s for validation and has an 88.9% validation accuracy. For testing, RF with $trees = 100$ and TF-IDF takes 43.7 s and has a test accuracy of 89.1%. With W2V data encoding and $trees = 100$, RF takes 826 s for training and 5.38 s for validation and has an 80.9% validation accuracy. For testing RF with $trees = 100$ and W2V takes 9.77 s and has a test accuracy of 81.1%. These results show that RF with $trees = 100$ and TF-IDF performs better.

	Training Time	**Validation Time**	**Testing Time**
TF-IDF	5,845 s	22.1 s	43.7 s
W2V	826 s	5.38 s	9.77 s

Table 4.12: Execution times for training, validation and testing using RF with $trees = 100$, TF-IDF and W2V.

	RF with *trees* = 100 (TF-IDF)		**RF with *trees* = 100 (W2V)**	
	Validation	**Testing**	**Validation**	**Testing**
Accuracy	88.9%	89.1%	80.9%	81.1%
Precision	89.0%	89.2%	80.6%	80.8%
Recall	88.9%	89.1%	80.9%	81.1%
F1-score	88.7%	88.9%	80.7%	80.9%

Table 4.13: Validation and testing results using RF with $trees = 100$, TF-IDF and W2V.

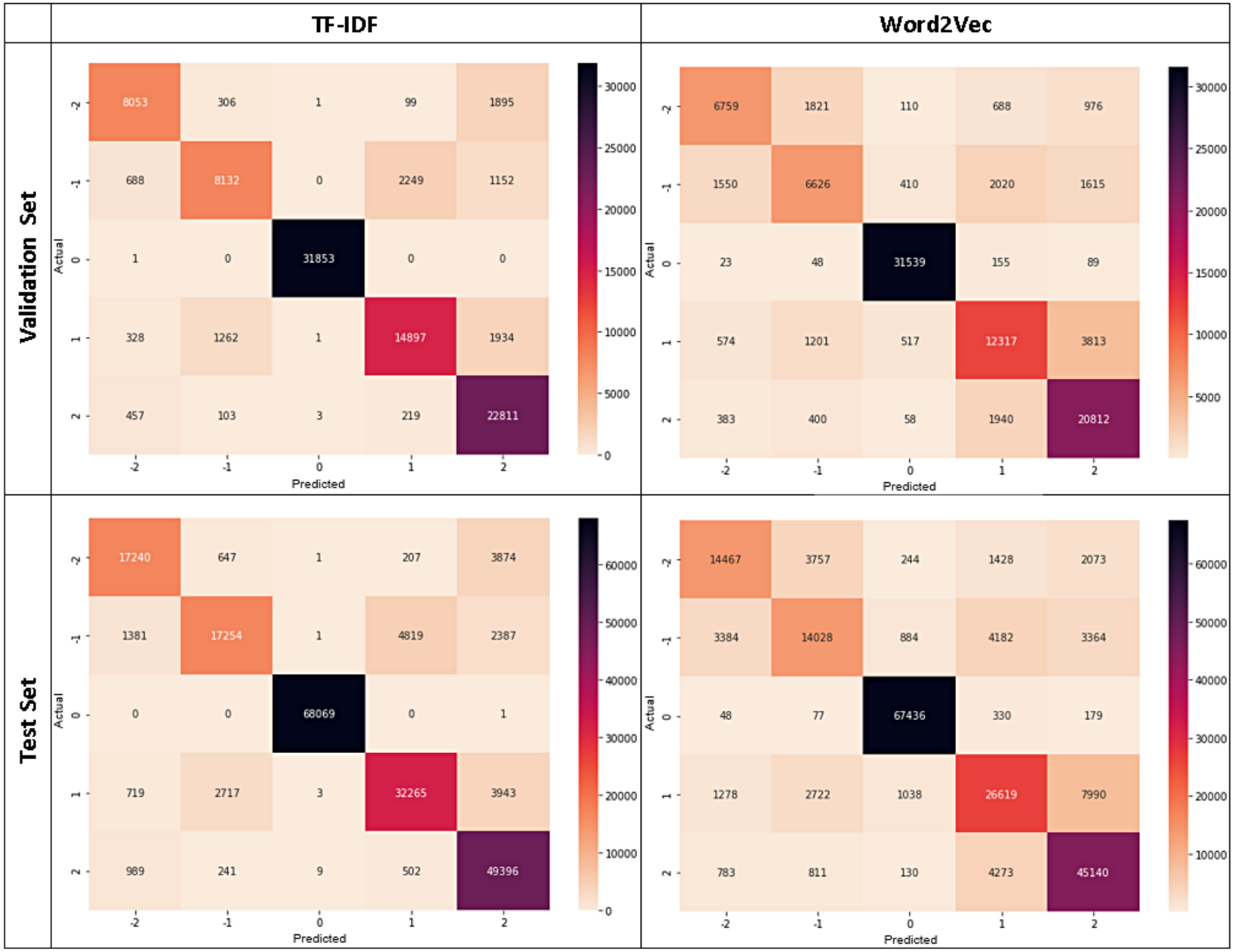


Figure 4.10: Confusion matrices for validation and testing using RF with $trees = 100$, TF-IDF and W2V.

4.2.7 Performance of RF with 300 Trees, TF-IDF and W2V

The performance of RF with *trees* = 300, TF-IDF and W2V data encoding is presented in Tables 4.14 and 4.15. Table 4.14 gives the execution times for training, validation and testing using RF with *trees* = 300, TF-IDF and W2V. Table 4.15 gives the validation and testing results using RF with *trees* = 300, TF-IDF and W2V data encoding. Figure 4.11 shows the confusion matrices using RF with *trees* = 300, TF-IDF and W2V data encoding. With TF-IDF data encoding and after changing the number of trees for RF from 100 to 300, the accuracy improved from 89.1% to 89.4% with the increase in training time from 5,845 s to 31,239 s. With W2V, the accuracy changes from 81.1% to 81.9% which is not a significant improvement. These results show that increasing the number of trees does not significantly improve the accuracy but there is a significant increase in execution time. Considering the execution time and improvement in accuracy, it is not worth increasing the number of trees above 300.

	Training Time	**Validation Time**	**Testing Time**
TF-IDF	31,239 s	65.5 s	116 s
W2V	2,307 s	11.2 s	24.0 s

Table 4.14: Execution times for training, validation and testing using RF with *trees* = 300, TF-IDF and W2V.

	RF with *trees* = 300 (TF-IDF)		**RF with *trees* = 300 (W2V)**	
	Validation	**Testing**	**Validation**	**Testing**
Accuracy	89.1%	89.4%	81.6%	81.9%
Precision	89.2%	89.5%	81.3%	81.4%
Recall	89.1%	89.4%	81.7%	81.9%
F1-score	88.8%	89.1%	81.2%	81.4%

Table 4.15: Validation and testing results using RF with *trees* = 300, TF-IDF and W2V.

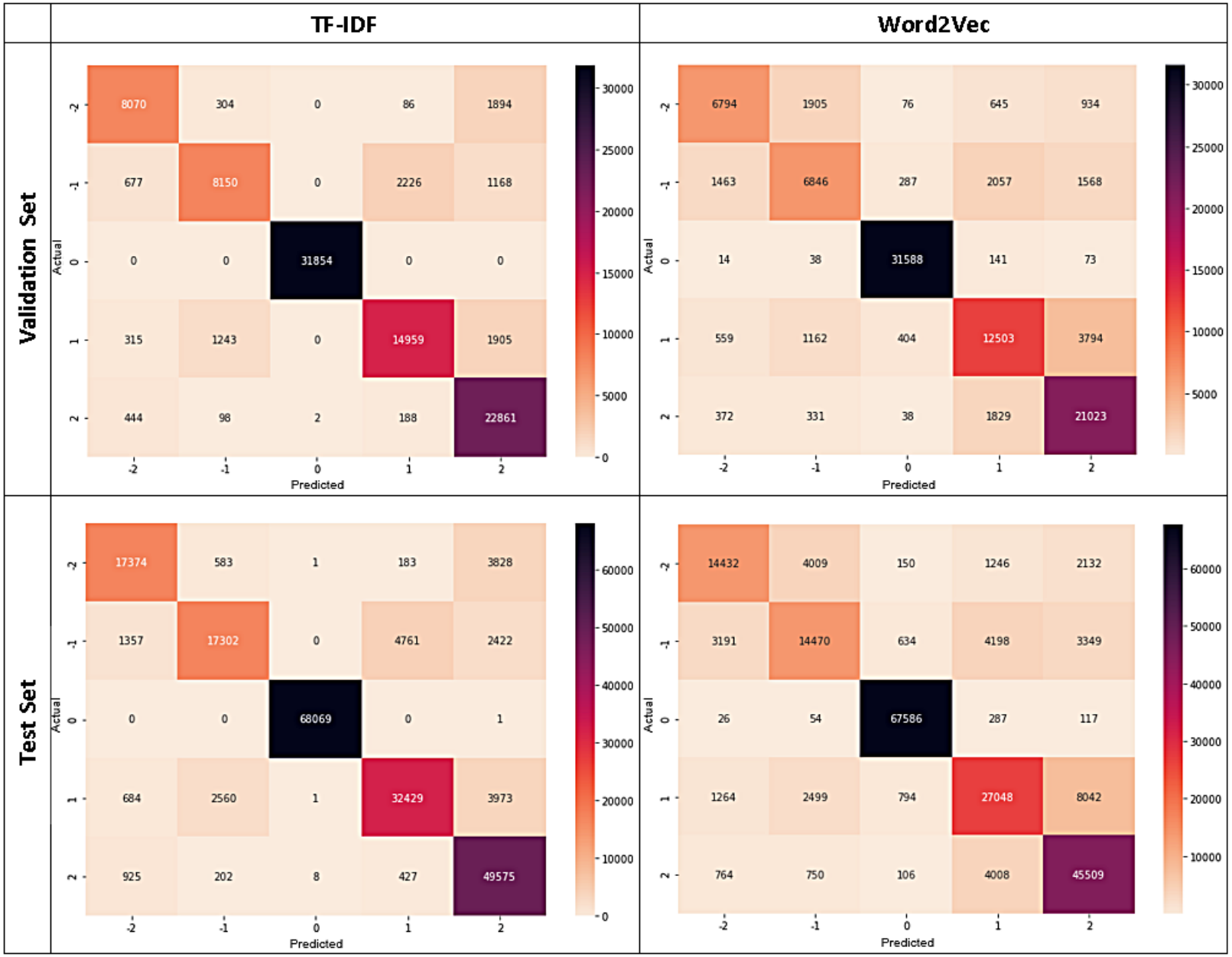


Figure 4.11: Confusion matrices for validation and testing using RF with $trees = 300$, TF-IDF and W2V.

4.2.8 Performance of XGBoost with 100 Gradient-boosted Trees, TF-IDF and W2V

The performance of XGBoost with $GBTrees = 100$ (number of gradient-boosted trees), TF-IDF and W2V data encoding is presented in Tables 4.16 and 4.17. Table 4.16 gives the execution times for training, validation and testing using XGBoost with $GBTrees = 100$, TF-IDF and W2V. Table 4.17 gives the validation and testing results using XGBoost with $GBTrees = 100$, TF-IDF and W2V data encoding. Figure 4.12 shows the confusion matrices using XGBoost with $GBTrees = 100$, TF-IDF and W2V data encoding. With TF-IDF data encoding and $GBTrees = 100$, XGBoost takes 255 s for training and 0.75 s for validation and has an 83.8% validation accuracy. For testing, XGBoost with $GBTrees = 100$ and TF-IDF takes 1.46 s and has a test accuracy of 84.1%. With W2V data encoding and $GBTrees = 100$, XGBoost takes 2,066 s for training and 0.55 s for validation and has an 83.9% validation accuracy. For testing, XGBoost with $GBTrees = 100$ and W2V takes 1.72 s and has a test accuracy of 84.1%. These results show that XGBoost with $GBTrees = 100$ and TF-IDF performs better.

	Training Time	**Validation Time**	**Testing Time**
TF-IDF	255 s	0.75 s	1.46 s
W2V	2,066 s	0.55 s	1.72 s

Table 4.16: Execution times for training, validation and testing using XGBoost with $GBTrees = 100$, TF-IDF and W2V.

	XGBoost with $GBTrees = 100$ (TF-IDF)		**XGBoost with $GBTrees = 100$ (W2V)**	
	Validation	**Testing**	**Validation**	**Testing**
Accuracy	83.8%	84.1%	83.9%	84.1%
Precision	84.4%	84.6%	83.7%	83.6%
Recall	83.9%	84.1%	83.9%	84.1%
F1-score	83.0%	83.3%	83.7%	83.6%

Table 4.17: Validation and testing results using XGBoost with $GBTrees = 100$, TF-IDF and W2V.

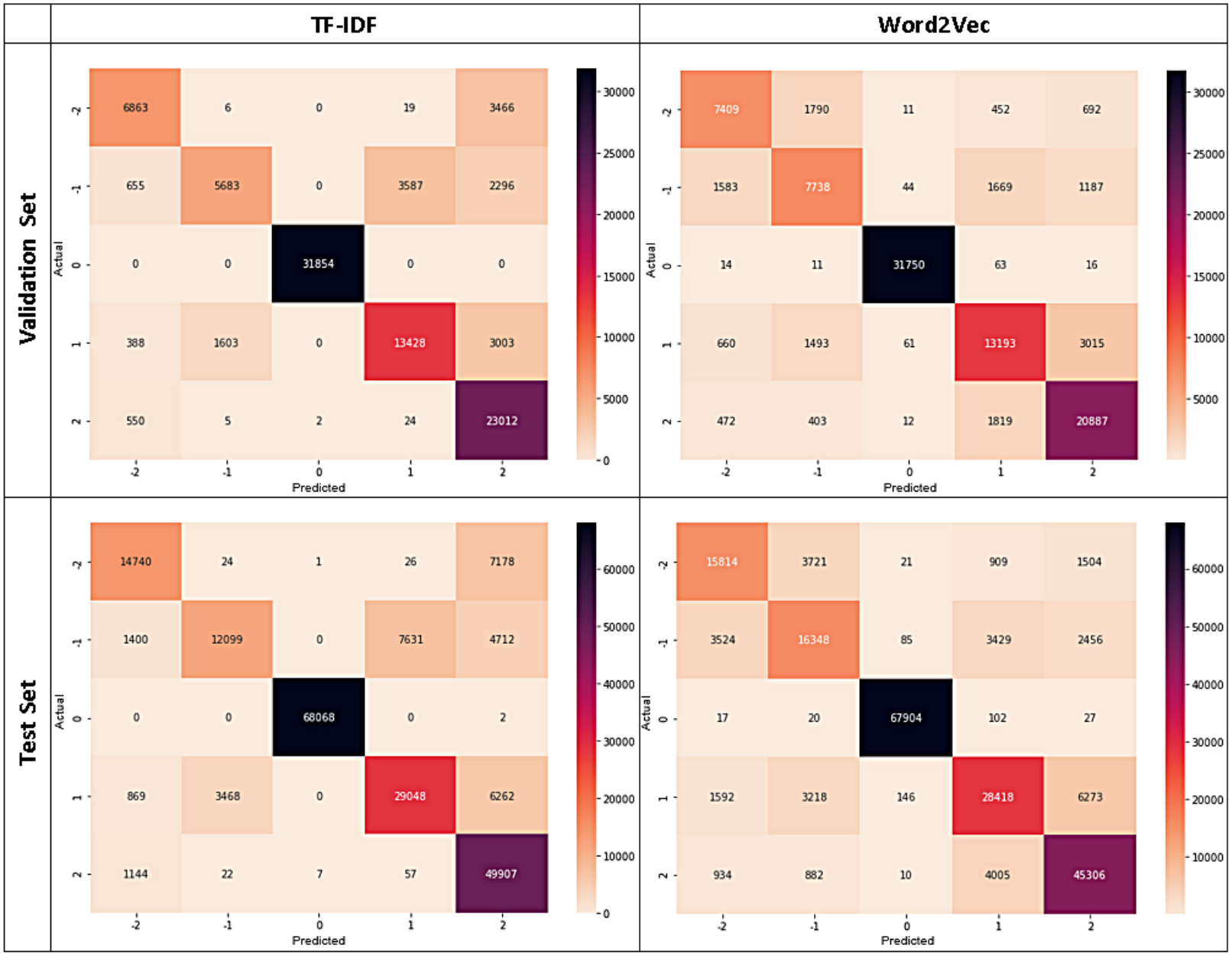


Figure 4.12: Confusion matrices for validation and testing using XGBoost with $GBTrees = 100$, TF-IDF and W2V.

4.2.9 Performance of XGBoost with 500 Gradient-boosted Trees, TF-IDF and W2V

The performance of XGBoost with $GBTrees = 500$ (gradient-boosted trees), TF-IDF and W2V data encoding is presented in Tables 4.18 and 4.19. Table 4.18 gives the execution times for training, validation and testing using XGBoost with $GBTrees = 500$, TF-IDF and W2V. Table 4.19 gives the validation and testing results using XGBoost with $GBTrees = 500$, TF-IDF and W2V data encoding. Figure 4.13 shows the confusion matrices using XGBoost with $GBTrees = 500$, TF-IDF and W2V data encoding. When the number of GBTrees in XGBoost is changed from the default value, i.e. 100 to 500, and with TF-IDF data encoding, it takes 1,128 s for training and 3.94 s for validation and has an 88.8% validation accuracy. For testing, XGBoost with $GBTrees = 500$ and TF-IDF takes 7.94 s and has a test accuracy of 89.1%. With W2V data encoding and $GBTrees = 500$, XGBoost takes 9,071 s for training and 2.31 s for validation and has an 85.7% validation accuracy. For testing, XGBoost with $GBTrees = 500$ and W2V takes 4.57 s and has a test accuracy of 85.8%. These results show that XGBoost with $GBTrees = 500$ and TF-IDF performs better, and also that changing the number of gradient-boosted trees from its default value, i.e. 100 to 500, improved the accuracy with some increase in execution time.

	Training Time	**Validation Time**	**Testing Time**
TF-IDF	1,128 s	3.94 s	7.94 s
W2V	9,071 s	2.31 s	4.57 s

Table 4.18: Execution times for training, validation and testing using XGBoost with $GBTrees = 500$, TF-IDF and W2V.

	XGBoost with $GBTrees = 500$ (TF-IDF)		**XGBoost with $GBTrees = 500$ (W2V)**	
	Validation	**Testing**	**Validation**	**Testing**
Accuracy	88.8%	89.1%	85.7%	85.8%
Precision	88.7%	89.0%	85.6%	85.7%
Recall	88.8%	89.1%	85.7%	85.8%
F1-score	88.4%	88.8%	85.6%	85.7%

Table 4.19: Validation and testing results using XGBoost with $GBTrees = 500$, TF-IDF and W2V.

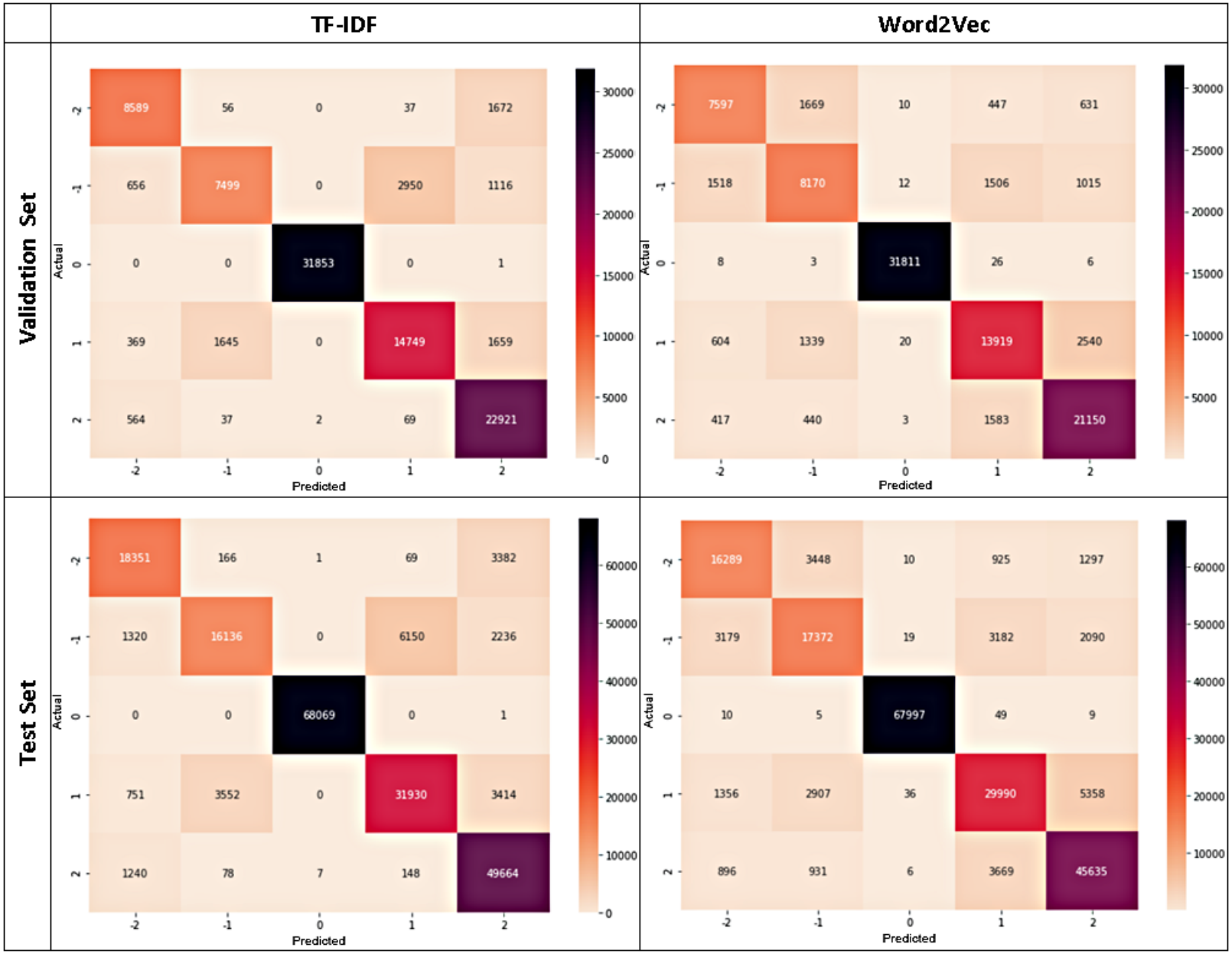


Figure 4.13: Confusion matrices for validation and testing using XGBoost with $GBTrees = 500$, TF-IDF and W2V.

4.2.10 Performance of XGBoost with 1500 Gradient-boosted Trees, TF-IDF and W2V

The performance of XGBoost with $GBTrees = 1500$ (gradient-boosted trees), TF-IDF and W2V data encoding is presented in Tables 4.20 and 4.21. Table 4.20 gives the execution times for training, validation and testing using XGBoost with $GBTrees = 1500$, TF-IDF and W2V. Table 4.21 gives the validation and testing results using XGBoost with $GBTrees = 1500$, TF-IDF and W2V data encoding. Figure 4.14 shows the confusion matrices using XGBoost with $GBTrees = 1500$, TF-IDF and W2V data encoding. When the number of GBTrees in XGBoost is changed from the default value, i.e. 100 to 500, it achieves an accuracy of 89.1% with TF-IDF. When GBTrees is increased from 500 to 1500, it outperforms MLR by 0.44% achieving 91.2% of accuracy. There is a significant improvement in accuracy from 84.1% to 91.2% when the the number of gradient-boosted trees is increased from 100 to 1500 with TF-IDF data encoding.

	Training Time	**Validation Time**	**Testing Time**
TF-IDF	3,811 s	30.9 s	73.8 s
W2V	27,627 s	5.9 s	12.5 s

Table 4.20: Execution times for training, validation and testing using XGBoost with $GBTrees = 1500$, TF-IDF and W2V.

	XGBoost with $GBTrees = 1500$ (TF-IDF)		**XGBoost with $GBTrees = 1500$ (W2V)**	
	Validation	**Testing**	**Validation**	**Testing**
Accuracy	91.0%	91.2%	86.3%	86.4%
Precision	90.8%	91.1%	86.1%	86.3%
Recall	91.0%	91.2%	86.3%	86.4%
F1-score	90.8%	91.1%	86.1%	86.3%

Table 4.21: Validation and testing results using XGBoost with $GBTrees = 1500$, TF-IDF and W2V.

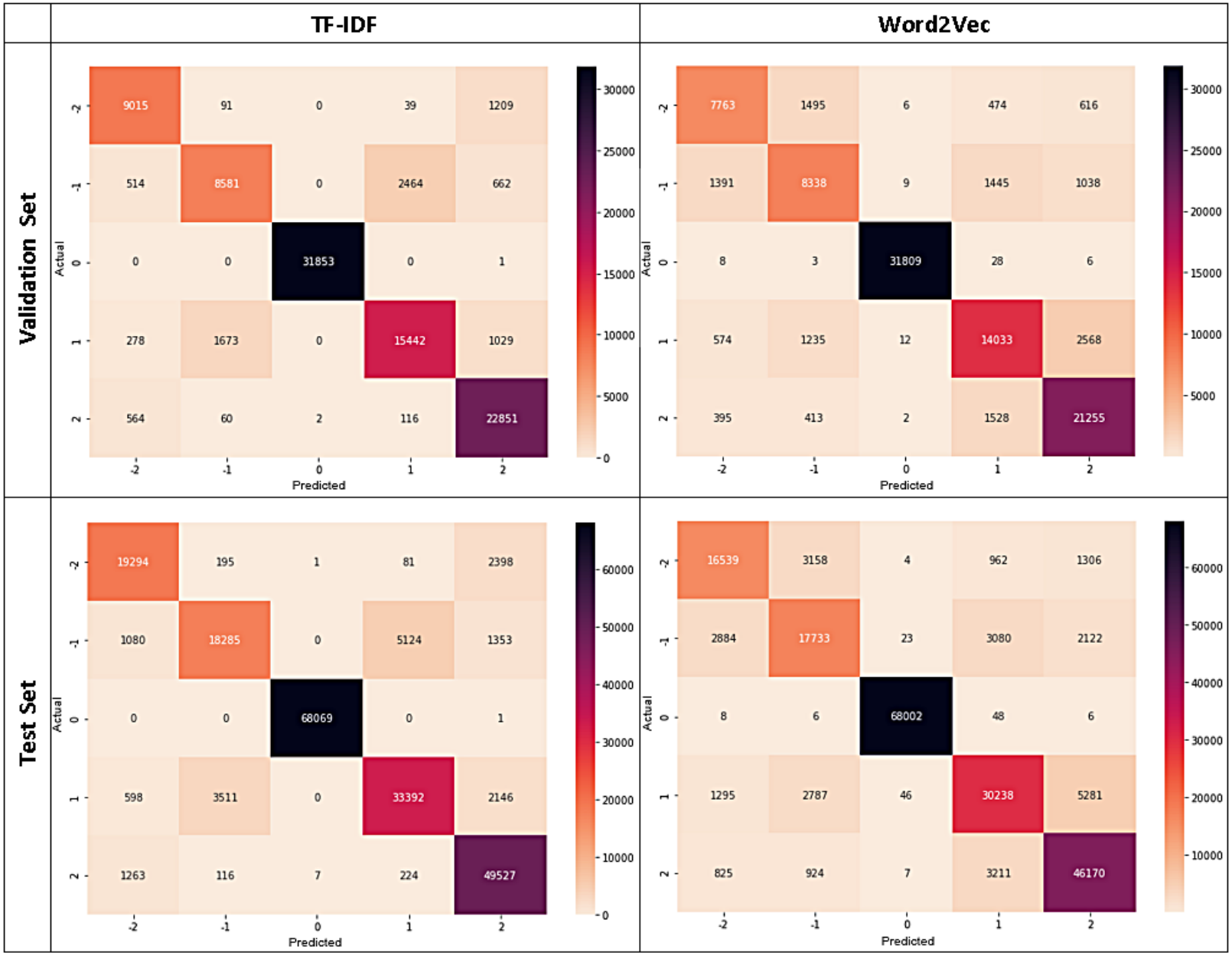


Figure 4.14: Confusion matrices for validation and testing using XGBoost with $GBTrees = 1500$, TF-IDF and W2V.

4.3 Discussion

This section discusses the ML algorithm results obtained with TF-IDF and W2V data encoding.

4.3.1 TF-IDF

Five ML algorithms were trained, validated and tested with TF-IDF data encoding. The best performing algorithm considering accuracy and execution time is MLR. It achieved an accuracy of 90.8% and total execution time including training, validation and testing was 42.8 s. Three of the ML algorithms, i.e. KNN, RF and XGBoost, were tuned to achieve higher accuracy. KNN with $K = 3$ and RF with $trees = 300$ did not achieve higher accuracy than MLR and they required more time for execution. XGBoost was also tuned with 100, 500 and 1500 GBTrees. XGBoost with 1500 GBTrees achieved an accuracy higher than MLR, i.e. 91.2% with less execution time than the other algorithms, i.e. 3,916 s. Thus, choosing appropriate parameters for ML algorithms can improve the performance.

With respect to accuracy, the XGBoost algorithm with $GBTrees = 1500$ achieved the highest accuracy, i.e. 91.2%. It is followed by MLR, RF with $trees = 300$, RF with $trees = 100$, XGBoost with $GBTrees = 500$, DT, XGBoost with $GBTrees = 100$, MNB, KNN with $K = 3$, and KNN with $K = 2$ with accuracies of 90.8%, 89.4%, 89.1%, 89.1%, 87.5%, 84.1%, 81.4%, 61.5% and 58.1%, respectively.

The fastest algorithm is MNB which required only 0.9 s for training, validation and testing. It achieved an accuracy of 81.4% which is better than some of the other ML algorithms. In terms of total execution time, the worst algorithm is RF with $trees = 300$ which required 31,420 s and achieved an accuracy of 89.4%.

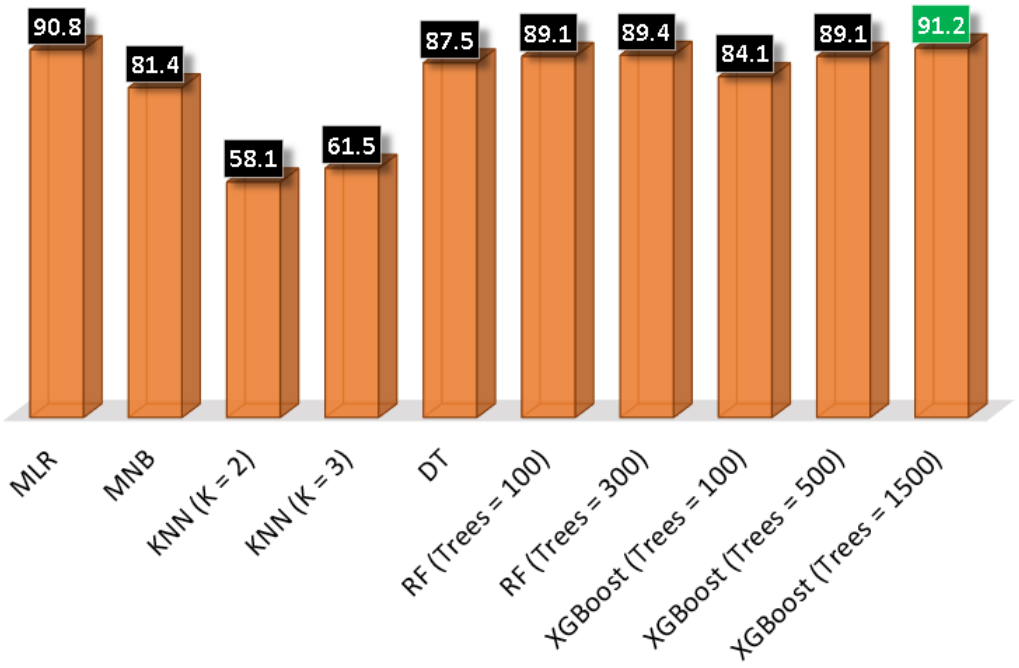


Figure 4.15: Test accuracy (%) of the ML algorithms with TF-IDF.

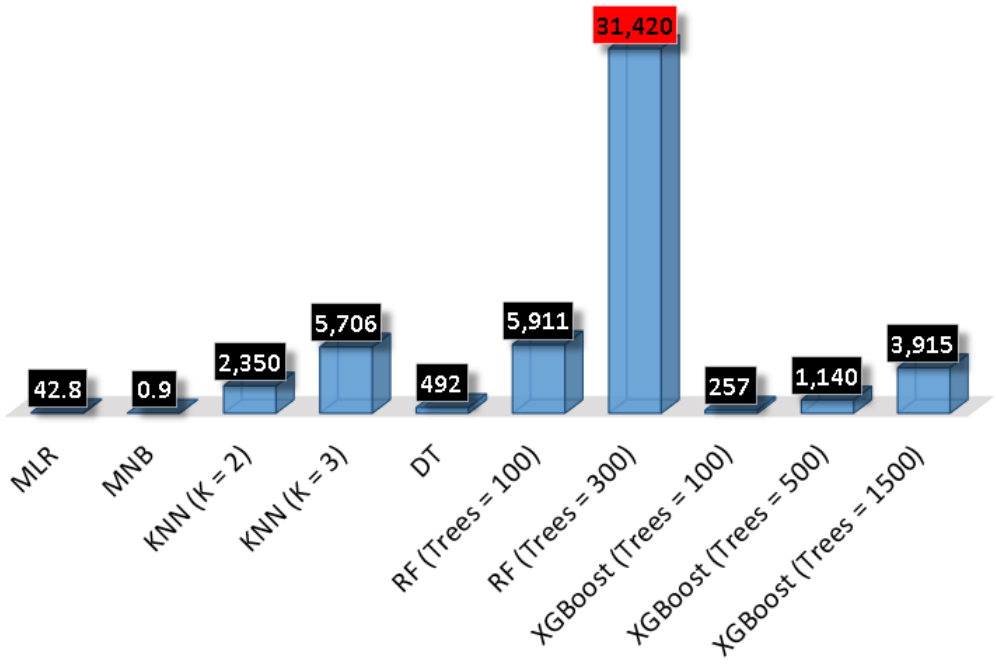


Figure 4.16: Total execution time (s) (training, validation and testing) for the ML algorithms with TF-IDF.

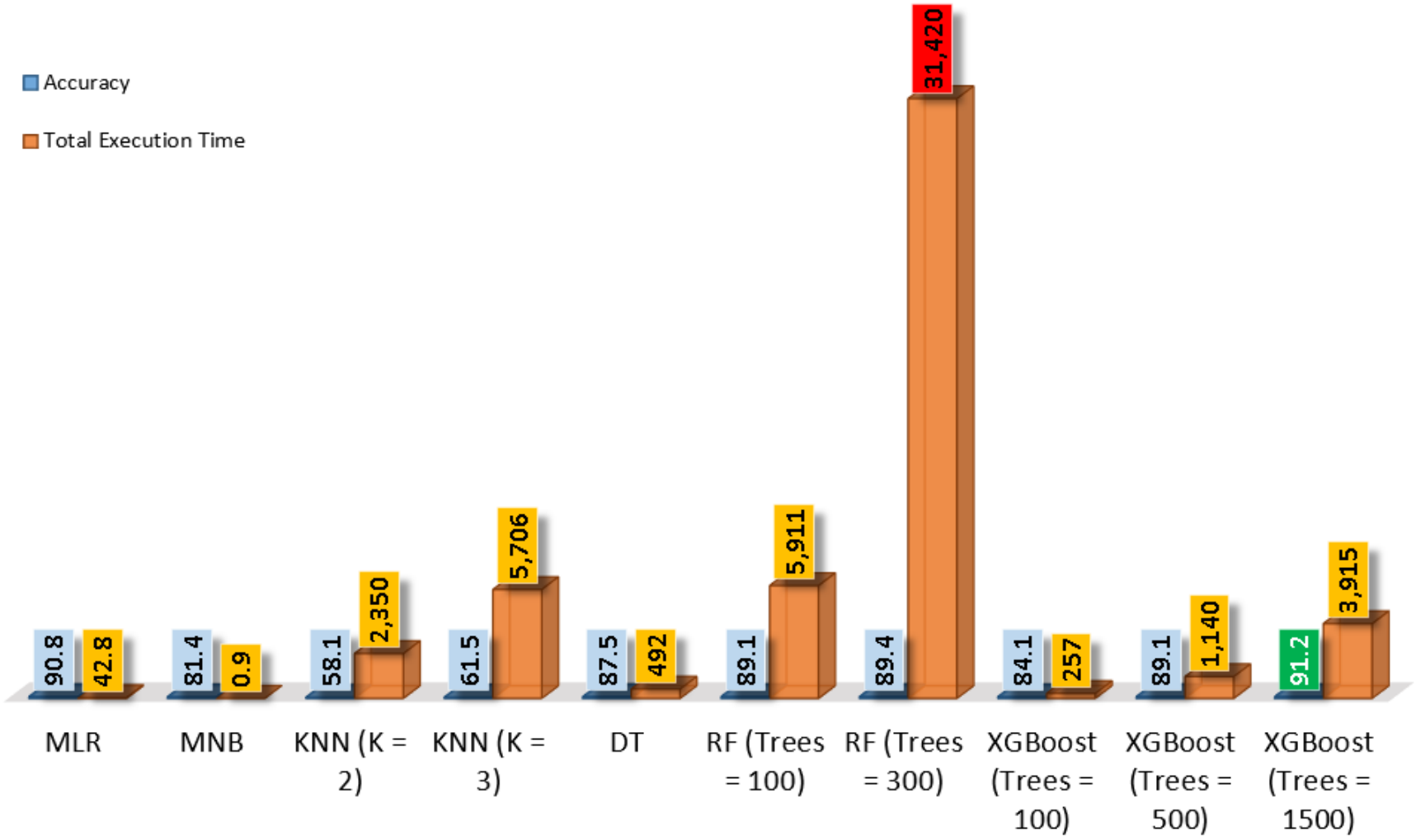
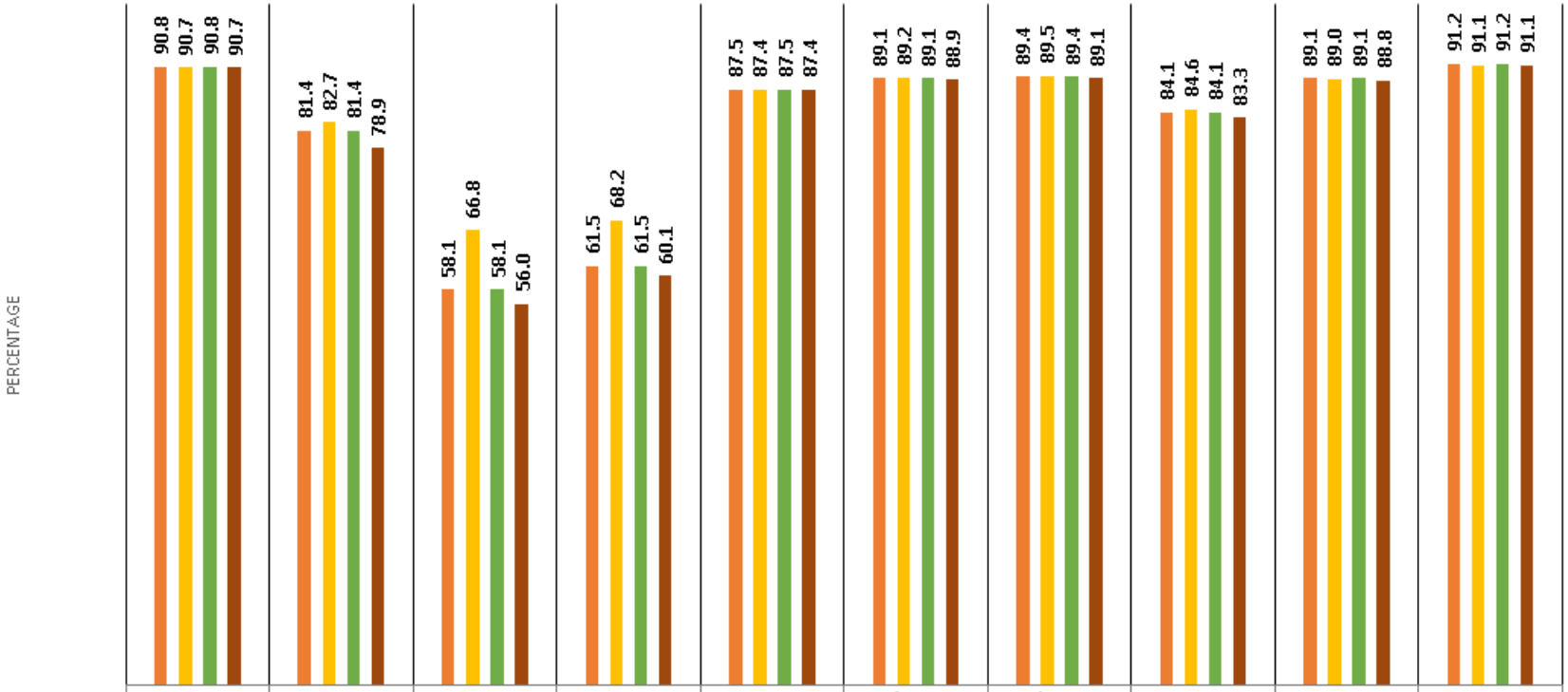


Figure 4.17: Test accuracy (%) versus total execution time (s) (training, validation and testing) for the ML algorithms with TF-IDF.



	MLR	MNB	KNN (K = 2)	KNN (K = 3)	DT	RF (Trees = 100)	RF (Trees = 300)	XGBoost (Trees = 100)	XGBoost (Trees = 500)	XGBoost (Trees = 1500)
Accuracy	90.8	81.4	58.1	61.5	87.5	89.1	89.4	84.1	89.1	91.2
Precision	90.7	82.7	66.8	68.2	87.4	89.2	89.5	84.6	89.0	91.1
Recall	90.8	81.4	58.1	61.5	87.5	89.1	89.4	84.1	89.1	91.2
f1-Score	90.7	78.9	56.0	60.1	87.4	88.9	89.1	83.3	88.8	91.1

Figure 4.18: Test accuracy, precision, recall and F1-score for the ML algorithms with TF-IDF.

4.3.2 Word2Vec

Five ML algorithms were trained, validated and tested on data encoded with W2V word embedding. Considering accuracy and total execution time, MLR achieved higher accuracy of 82.9% in just 107 s and outperformed the other ML algorithms. Three of the ML algorithms, i.e. KNN, RF and XGBoost, were tuned to achieve higher accuracy. KNN with $K = 3$ and RF with $trees = 300$ did not achieve higher accuracy than MLR and they required more time for execution. XGBoost was also tuned with 100, 500 and 1500 GBTrees. XGBoost with 100, 500 and 1500 GBTrees achieved an accuracy higher than MLR, i.e. 84.1%, 85.8% and 86.4% with execution times 2,069 s, 9,078 s and 27,645 s, respectively.

With respect to accuracy, XGBoost with $GBTrees = 1500$ achieved the highest accuracy, i.e. 86.4%. It is followed by XGBoost with $GBTrees = 500$, XGBoost with $GBTrees = 100$, RF with $trees = 300$, RF with $trees = 100$, KNN with $K = 3$, KNN with $K = 2$, DT and MNB, achieving accuracies of 85.8%, 84.1%, 81.9%, 81.1%, 77.1%, 73.8%, 68.1%, and 49.1%, respectively.

The fastest model is MNB which required only 0.92 s for training, validation and testing. It achieved an accuracy of only 49.1%. In terms of execution time, the worst algorithm is KNN with $K = 3$ which required 57,698 s (approximately 16 hr) and achieved an accuracy of 77.1%.

W2V usually performs better than the traditional TF-IDF data encoding for certain problems, specifically when capturing semantic and contextual information of text. However, in this work TF-IDF outperforms W2V because after applying text preprocessing, i.e. text normalization and tokenization, the text loses its contextual information. The number of features in the TF-IDF matrix is also higher than in the W2V matrix. Furthermore, word frequency in the entire text is considered by TF-IDF which plays a significant role in sentiment analysis, but W2V does not consider this.

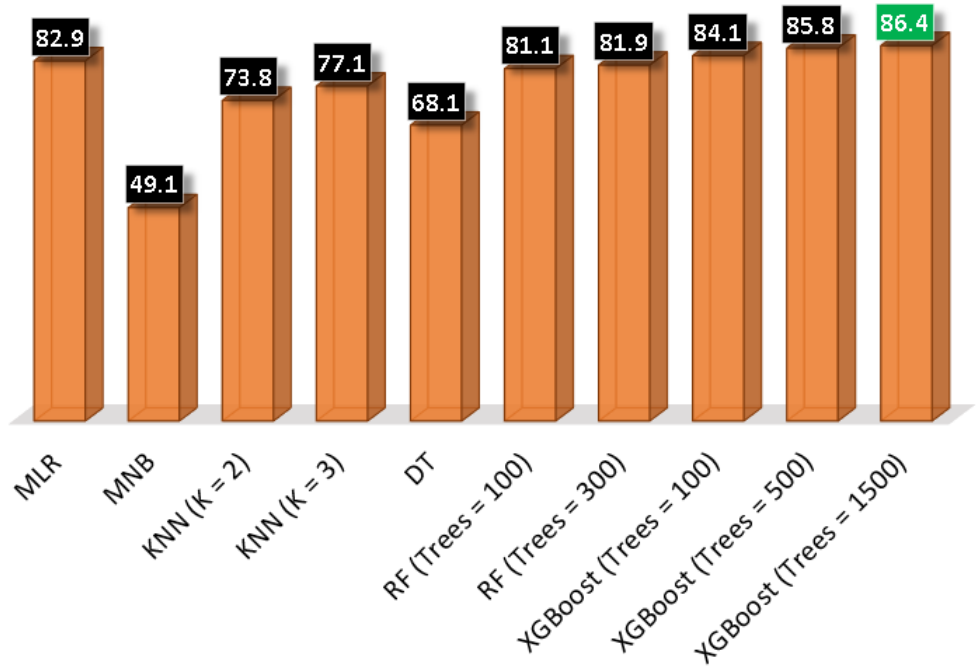


Figure 4.19: Test accuracy (%) of the ML algorithms with W2V.

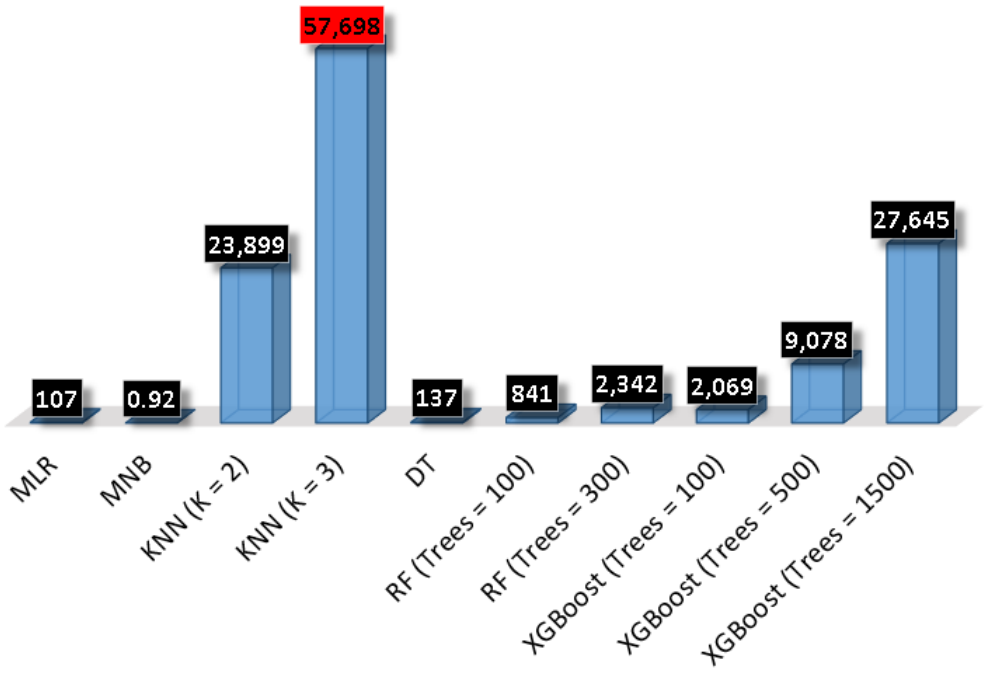


Figure 4.20: Total execution time (s) (training, validation and testing) for the ML algorithms with W2V.

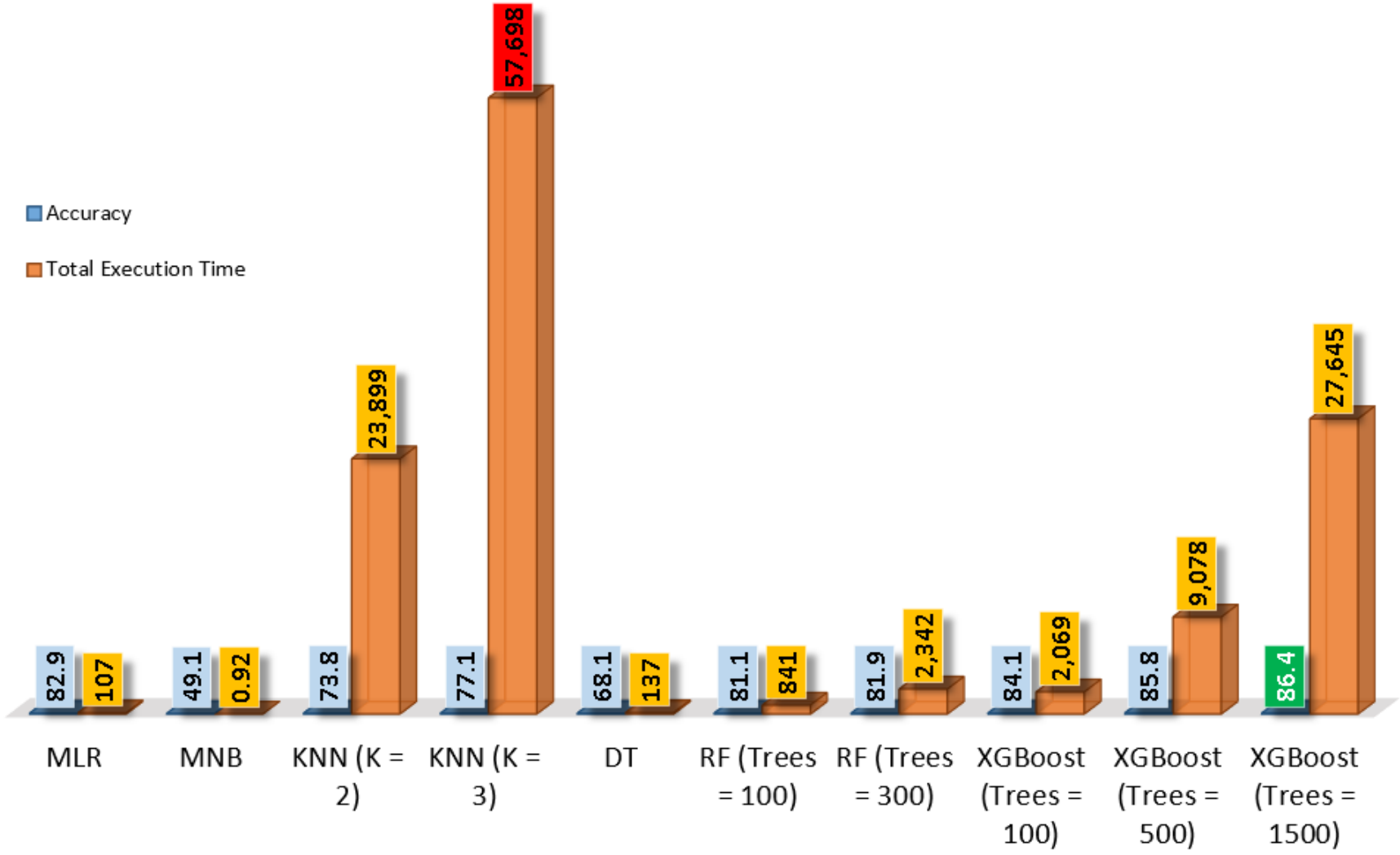
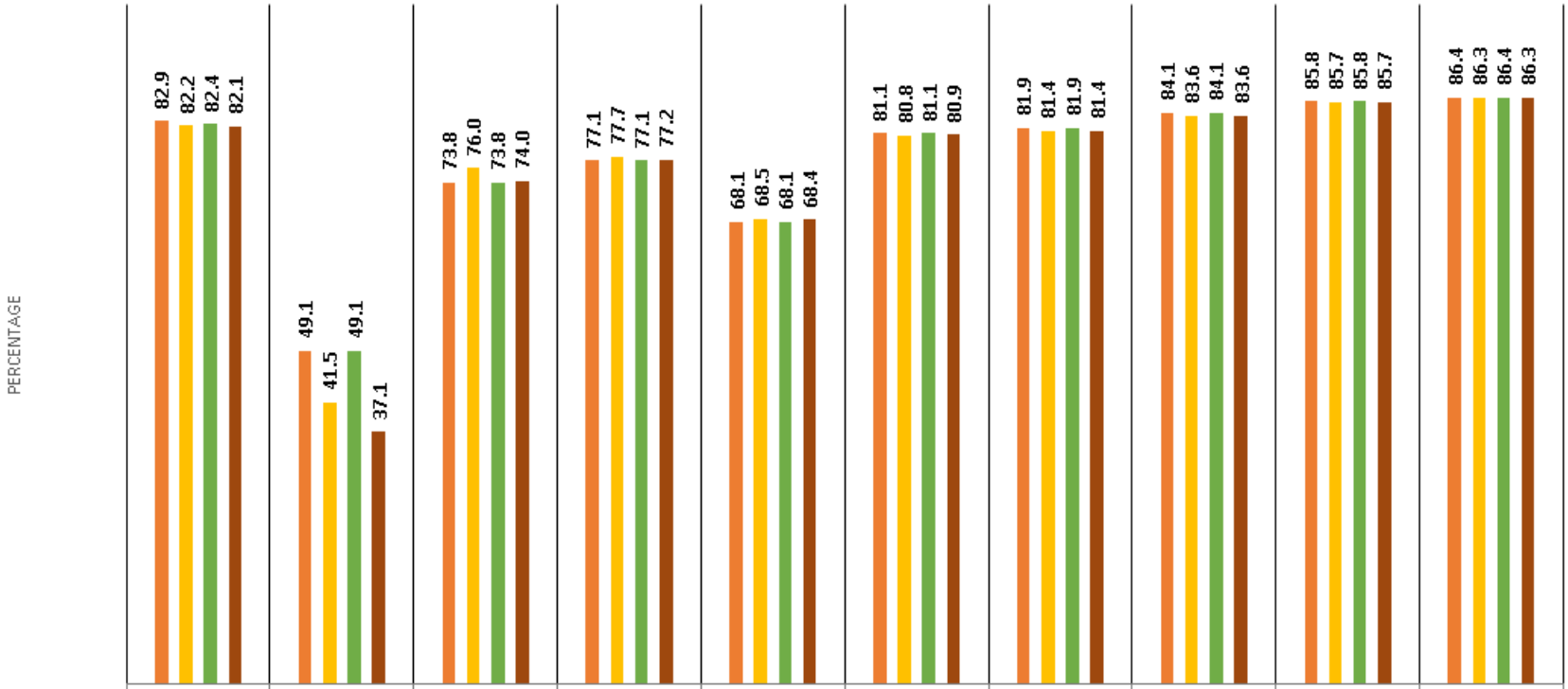


Figure 4.21: Test accuracy (%) versus total execution time (s) (training, validation and testing) for the ML algorithms with W2V.



	MLR	MNB	KNN (K = 2)	KNN (K = 3)	DT	RF (Trees = 100)	RF (Trees = 300)	XGBoost (Trees = 100)	XGBoost (Trees = 500)	XGBoost (Trees = 1500)
Accuracy	82.9	49.1	73.8	77.1	68.1	81.1	81.9	84.1	85.8	86.4
Precision	82.2	41.5	76.0	77.7	68.5	80.8	81.4	83.6	85.7	86.3
Recall	82.4	49.1	73.8	77.1	68.1	81.1	81.9	84.1	85.8	86.4
f1-Score	82.1	37.1	74.0	77.2	68.4	80.9	81.4	83.6	85.7	86.3

Figure 4.22: Test accuracy, precision, recall and F1-score for the ML algorithms with W2V.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

Stretched words along with other textual features in a tweet are not used arbitrarily, and there was a significant number of tweets containing these features, i.e. 28.5% in a randomly extracted dataset which could not be neglected as shown in Table 2.1. A very large, randomly extracted, unbiased dataset named NB-StV_Senti with 688,881 tweets containing the stretched words, smileys, and other textual features was created and labelled with the proposed rule-based sentiment analysis technique called StretchVADER. Five ML algorithms were trained, validated, and tested with TF-IDF and W2V data encoding in order to evaluate which combination performed better with the dataset.

Accuracy, precision, recall, F1-score, and execution time were considered as metrics to evaluate the performance of the ML algorithms. Some of ML algorithms were also tuned to perform better. The results obtained show that the XGBoost algorithm with $GBTrees = 1500$ and TF-IDF performed better than the other ML algorithms achieving the highest accuracy of 91.2%. The slowest ML algorithm was KNN with $K = 3$ and W2V which had a total execution time of 16 hr. Overall, the worst ML algorithm was KNN with ($K = 2$) and TF-IDF with significantly higher total execution time, i.e. 2,350 s and lower accuracy, i.e. 58.1%. The lowest accuracy of 49.1% was achieved by MNB with W2V data encoding and total execution time of 0.92 s. The worst ML algorithm in terms of accuracy, precision, recall, and F1-score was MNB with W2V, i.e. 49.1%, 41.5%, 49.1%, 37.1%, respectively. MLR with TF-IDF was

better in terms of both accuracy and total execution time. It achieved an accuracy of 90.8% with only 42.8 s total execution time. Hence, it can be concluded that proper parameter selection achieves better results. It can also be concluded that XGBoost with $GBTrees = 1500$ and TF-IDF data encoding is better for sentiment analysis of tweets containing stretched words and other textual features.

5.2 Future Work

The following can be considered in future to improve the performance. In Stretch-VADER, the generation of the root word from the condensed form of the stretched word is based on an English dictionary. The results can be improved by using a larger English dictionary. In reduction to root words, spelling correction can be used to correct words in parallel by considering their context. The proposed approach can be used with Amazon product reviews and other platforms to create a rating system using multi-class classification of customer reviews or comments. Both data encodings, i.e. TF-IDF and W2V can be combined together to make a hybrid approach. Other networks such as DL classifiers can also be employed to improve the performance.

Bibliography

[1] D. Demszky, D. Movshovitz-Attias, J. Ko, A. Cowen, G. Nemade, and S. Ravi, "GoEmotions: A Dataset of Fine-Grained Emotions," in *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pp. 4040–4054, July 2020.

[2] S. Brody and N. Diakopoulos, "Cooooooooooooooollllllllllllll!!!!!!!!!!!!!! Using Word Lengthening to Detect Sentiment in Microblogs," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, (Edinburgh, Scotland, UK), pp. 562–570, July 2011.

[3] M. Wankhade, A. C. S. Rao, and C. Kulkarni, "A Survey on Sentiment Analysis Methods, Applications, and Challenges," *Artificial Intelligence Review*, vol. 55, pp. 5731–5780, Oct 2022.

[4] A. Go, R. Bhayani, and L. Huang, "Twitter Sentiment Classification using Distant Supervision," *CS224N Project Report*, Stanford University, Stanford, CA, USA, Jan. 2009.

[5] O. Das and R. C. Balabantaray, "Sentiment Analysis of Movie Reviews using POS Tags and Term Frequencies," *International Journal of Computer Applications*, vol. 96, pp. 36–41, June 2014.

[6] M. Li, E. Ch'ng, A. Chong, and S. See, "Twitter Sentiment Analysis of the 2016 US Presidential Election using an Emoji Training Heuristic," in *Applied Informatics and Technology Innovation Conference*, (Newcastle, Australia), Nov. 2016.

[7] J. F. Sánchez-Rada and C. A. Iglesias, "Social Context in Sentiment Analysis: Formal Definition, Overview of Current Trends and Framework for Comparison," *Information Fusion*, vol. 52, pp. 344–356, May 2019.

[8] Wiktionary, "Appendix: Glossary." `https://en.wiktionary.org/w/index.php?title=Appendix:Glossary&oldid=51610328`, 07 2022. Accessed: 2022-07-10.

[9] Y. M. Kalman and D. Gergle, "Letter Repetitions in Computer-Mediated Communication: A Unique Link Between Spoken and Online Language," *Computers in Human Behavior*, vol. 34, pp. 187–193, May 2014.

[10] J. Eisenstein, "What to do about Bad Language on the Internet," in *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, (Atlanta, GA, USA), pp. 359–369, June 2013.

[11] T. Schnoebelen, "Do you Smile with your Nose? Stylistic Variation in Twitter Emoticons," *University of Pennsylvania Working Papers in Linguistics*, vol. 18, pp. 117–125, Sep. 2012.

[12] T. J. Gray, C. M. Danforth, and P. S. Dodds, "Hahahahaha, Duuuuude, Yeeessss!: A Two-Parameter Characterization of Stretchable Words and the Dynamics of Mistypings and Misspellings," *PLOS One*, vol. 15, art. e0232938, May 2020.

[13] C. Michael, "Because Internet: Understanding the New Rules of Language," *Training, Language and Culture*, vol. 3, pp. 107–111, July 2019.

[14] J. A. Simpson, E. S. Weiner, *et al.*, *The Oxford English Dictionary*. Oxford, UK: Clarendon Press, 1989.

[15] W. O'Grady, A. J. Aronoff, and M. Rees-Miller, *Contemporary Linguistics. 6th Ed.* London, UK: Longman, Aug. 2009.

[16] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs up? Sentiment Classification using Machine Learning Techniques," *arXiv preprint art. cs/0205070*, July 2002.

[17] S. Gupta, R. Singh, and V. Singla, "Emoticon and Text Sarcasm Detection in Sentiment Analysis," in *Proceedings of the International Conference on Sustainable Technologies for Computational Intelligence*, (Singapore), Jan. 2020.

[18] S. K. Bharti, B. Vachha, R. Pradhan, K. S. Babu, and S. K. Jena, "Sarcastic Sentiment Detection in Tweets Streamed in Real Time: A Big Data Approach," *Digital Communications and Networks*, vol. 2, pp. 108–121, June 2016.

[19] J. Subramanian, V. Sridharan, K. Shu, and H. Liu, "Exploiting Emojis for Sarcasm Detection," in *Proceedings of the International Conference on Social, Cultural, and Behavioral Modeling*, (Cham, Switzerland), pp. 70–80, Springer, June 2019.

[20] A. Hogenboom, D. Bal, F. Frasincar, M. Bal, F. de Jong, and U. Kaymak, "Exploiting Emoticons in Sentiment Analysis," in *Proceedings of the Annual ACM Symposium on Applied Computing*, (New York, NY, USA), pp. 703–710, Mar. 2013.

[21] Y. Huang, D. Guo, A. Kasakoff, and J. Grieve, "Understanding US Regional Linguistic Variation with Twitter Data Analysis," *Computers, Environment and Urban Systems*, vol. 59, pp. 244–255, Sep. 2016.

[22] T. J. Gray, A. J. Reagan, P. S. Dodds, and C. M. Danforth, "English Verb Regularization in Books and Tweets," *PLOS One*, vol. 13, art. e0209651, Dec. 2018.

[23] B. Gonçalves and D. Sánchez, "Crowdsourcing Dialect Characterization through Twitter," *PLOS One*, vol. 9, art. e112074, July 2014.

[24] B. Gonçalves, L. Loureiro-Porto, J. J. Ramasco, and D. Sánchez, "Mapping the Americanization of English in Space and Time," *PLOS One*, vol. 13, art. e0197741, May 2018.

[25] G. Donoso and D. Sánchez, "Dialectometric Analysis of Language Variation in Twitter," in *Proceedings of the Workshop on NLP for Similar Languages, Varieties and Dialects*, (Valencia, Spain), pp. 16–25, Apr. 2017.

[26] J. Eisenstein, B. O'Connor, N. A. Smith, and E. Xing, "A Latent Variable Model for Geographic Lexical Variation," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, (Cambridge, MA, USA), pp. 1277–1287, Oct. 2010.

[27] J. Eisenstein, B. O'Connor, N. A. Smith, and E. P. Xing, "Diffusion of Lexical Change in Social Media," *PLOS One*, vol. 9, art. e113114, Nov. 2014.

[28] J. Grieve, A. Nini, and D. Guo, "Analyzing Lexical Emergence in Modern American English Online," *English Language & Linguistics*, vol. 21, pp. 99–127, May 2016.

[29] J. Grieve, "Natural Selection in the Modern English Lexicon," in *Proceedings of the International Conference on Language Evolution*, (Birmingham, UK), pp. 153–157, Jan. 2018.

[30] S. Symeonidis, D. Effrosynidis, and A. Arampatzis, "A Comparative Evaluation of Pre-Processing Techniques and their Interactions for Twitter Sentiment Analysis," *Expert Systems with Applications*, vol. 110, pp. 298–310, Nov. 2018.

[31] S. Mohammad, S. Kiritchenko, and X. Zhu, "NRC-Canada: Building the State-of-the-art in Sentiment Analysis of Tweets," in *Proceedings of the International Workshop on Semantic Evaluation*, (Atlanta, GA, USA), pp. 321–327, June 2013.

[32] E. Fersini, E. Messina, and F. A. Pozzi, "Expressive Signals in Social Media Languages to Improve Polarity Detection," *Information Processing & Management*, vol. 52, pp. 20–35, Jan. 2016.

[33] K. Gimpel, N. Schneider, B. O'Connor, D. Das, D. Mills, J. Eisenstein, M. Heilman, D. Yogatama, J. Flanigan, and N. A. Smith, "Part-of-Speech Tagging for Twitter: Annotation, Features, and Experiments," in *Proceedings of the Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, (Portland, OR, USA), pp. 42–47, June 2011.

[34] J. Foster, Ö. Çetinoğlu, J. Wagner, J. Le Roux, J. Nivre, D. Hogan, and J. van Genabith, "From News to Comment: Resources and Benchmarks for Parsing the Language of Web 2.0," in *Proceedings of International Joint Conference on Natural Language Processing*, (Chiang Mai, Thailand), pp. 893–901, Nov. 2011.

[35] A. Ritter, S. Clark, Mausam, and O. Etzioni, "Named Entity Recognition in Tweets: An Experimental Study," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, (Edinburgh, Scotland, UK), pp. 1524–1534, July 2011.

[36] S. Fuchs, E. Savin, U. Reichel, C. Ebert, and M. Krifka, "Letter Replication as Prosodic Amplification in Social Media," in *Proceedings of the Conference on*

Phonetics and Phonology in German Speaking Areas, (Berlin, Germany), pp. 65–68, Oct. 2017.

[37] S. Fuchs, E. Savin, S. Solt, C. Ebert, and M. Krifka, "Antonym Adjective Pairs and Prosodic Iconicity: Evidence from Letter Replications in an English Blogger Corpus," *Linguistics Vanguard*, vol. 5, art. 20180017, Jan. 2019.

[38] E. Darics, "Non-Verbal Signalling in Digital Discourse: The Case of Letter Repetition," *Discourse, Context & Media*, vol. 2, pp. 141–148, 2013.

[39] I. Pak, P. L. Teh, and Y. N. Cheah, "Hidden Sentiment Behind Letter Repetition in Online Reviews," *Journal of Telecommunication, Electronic and Computer Engineering*, vol. 10, pp. 115–120, Oct. 2018.

[40] C. Hutto and E. Gilbert, "Vader: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Text," in *Proceedings of the International AAAI Conference on Web and Social Media*, (Ann Arbor, MI, USA), pp. 216–225, May 2014.

[41] B. Liu *et al.*, "Sentiment Analysis and Subjectivity," *Handbook of Natural Language Processing*, vol. 2, pp. 627–666, Jan. 2010.

[42] P. Bhatia, Y. Ji, and J. Eisenstein, "Better Document-Level Sentiment Analysis from RST Discourse Parsing," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, (Lisbon, Portugal), pp. 2212–2218, Sep. 2015.

[43] B. Yang and C. Cardie, "Context-aware Learning for Sentence-level Sentiment Analysis with Posterior Regularization," in *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, (Baltimore, MD, USA), pp. 325–335, June 2014.

[44] G. Rao, W. Huang, Z. Feng, and Q. Cong, "LSTM with Sentence Representations for Document-level Sentiment Classification," *Neurocomputing*, vol. 308, pp. 49–57, May 2018.

[45] X. Wang, X. Chen, M. Tang, T. Yang, and Z. Wang, "Aspect-level Sentiment Analysis Based on Position Features using Multilevel Interactive Bidirectional GRU and Attention Mechanism," *Discrete Dynamics in Nature and Society*, vol. 2020, pp. 1–13, July 2020.

[46] S. Rajasegarar, C. Leckie, M. Palaniswami, and J. C. Bezdek, "Quarter Sphere based Distributed Anomaly Detection in Wireless Sensor Networks," in *Proceedings of the IEEE International Conference on Communications*, (Glasgow, UK), pp. 3864–3869, July 2007.

[47] L. Tian, C. Lai, and J. Moore, "Polarity and Intensity: The Two Aspects of Sentiment Analysis," in *Proceedings of Grand Challenge and Workshop on Human Multimodal Language*, (Melbourne, Australia), pp. 40–47, July 2018.

[48] Y. Başţanlar and M. Özuysal, "Introduction to Machine Learning," *miRNomics: MicroRNA Biology and Computational Analysis*, pp. 105–128, Nov. 2013.

[49] D. Fumo, "Types of Machine Learning Algorithms You Should Know," *Towards Data Science*, vol. 15, June 2017.

[50] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer New York, NY, USA, Aug. 2009.

[51] S. Dridi, "Supervised Learning - A Systematic Literature Review," preprint, Dec. 2021.

[52] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Pearson Upper Saddle River, NJ, USA, 3rd ed., Dec. 2009.

[53] D. L. Yse, "Text Normalization for Natural Language Processing (NLP)." `https://towardsdatascience.com/text-normalization-for-natural-language-processing-nlp-70a314bfa646`, Feb. 2021. Accessed: 2023-2-18.

[54] N. Sabharwal and A. Agrawal, "Introduction to word embeddings," in *Hands-on Question Answering Systems with BERT: Applications in Neural Networks and Natural Language Processing*, pp. 47–49, Apress Berkeley, CA, Jan. 2021.

[55] G. Tripepi, K. Jager, F. Dekker, and C. Zoccali, "Linear and Logistic Regression Analysis," *Kidney International*, vol. 73, pp. 806–810, Apr. 2008.

[56] S. Raschka, "Naive Bayes and Text Classification," *arXiv preprint arXiv:1410.5329*, Oct. 2014.

[57] S. Kanj, F. Abdallah, T. Denœux, and K. Tout, "Editing Training Data for Multi-label Classification with the K-Nearest Neighbor Rule," *Pattern Analysis and Applications*, vol. 19, pp. 145–161, Feb. 2016.

[58] B. Charbuty and A. Abdulazeez, "Classification Based on Decision Tree Algorithm for Machine Learning," *Journal of Applied Science and Technology Trends*, vol. 2, pp. 20–28, Mar. 2021.

[59] F.-J. Yang, "An Extended Idea About Decision Trees," in *Proceedings of the International Conference on Computational Science and Computational Intelligence*, (Las Vegas, NV, USA), pp. 349–354, Dec. 2019.

[60] J. Liang, Z. Qin, S. Xiao, L. Ou, and X. Lin, "Efficient and Secure Decision Tree Classification for Cloud-Assisted Online Diagnosis Services," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, pp. 1632–1644, June 2019.

[61] A. Brifcani and A. Issa, "Intrusion Detection and Attack Classifier Based on Three Techniques: A Comparative Study," *Eng. & Tech. Journal*, vol. 29, pp. 368–412, Jan. 2011.