SENTIMENT ANALYSIS ON SOCIAL NETWORKS USING MACHINE LEARNING AND AUDIO PROCESSING

A Thesis

by

Mihail Duşcu

Submitted to the
Graduate School of Sciences and Engineering
In Partial Fulfillment of the Requirements for
the Degree of

M.Sc. in Industrial Engineering

in the
Department of Industrial Engineering

Özyeğin University
August 2019

SENTIMENT ANALYSIS ON SOCIAL NETWORKS USING MACHINE LEARNING AND AUDIO PROCESSING

Approved by:

Assistant Professor Dilek Günneç Danış,
Advisor
Department of Industrial Engineering
Özyeğin University

Assistant Professor Erinç Albey
Department of Industrial Engineering
Özyeğin University

Assistant Professor Evren Güney
Department of Industrial Engineering
MEF University

Date Approved: 20 August 2019

To everyone who supports me on my path.

ABSTRACT

Polarity classification is one of the most fundamental problems in sentiment analysis. Our study strives to develop a new definition, extraction technique and utilization of features based on the audio data for polarity classification on Twitter messages. The background of work relies on a recent study which suggests that brain uses sound as a part of language generation and words are comprehended as they are converted into sound. Using sound is effective especially for social media messages which are likely to contain misspelled or shortened words, where the sound is similar to the actual word (e.g., thank u, b4). Our results show that one of our proposed feature set definitions demonstrate an improvement in accuracy in comparison to existing studies.

ÖZETÇE

Kutuplaşma sınıflandırması, duygu analizinin en temel problemlerinden biridir. Bizim yaptığımız inceleme, Twitter mesajlarında kutuplaşma sınıflandırması yapmak için ses verilerine dayanarak yeni bir tanım, çıkarım ve kullanım geliştirmeye çalışmaktadır. Çalışmanın arka planı son dönemde yapılan bir incelemeye dayanmaktadır: beyin, dil oluşturmak/üretmek için sesleri kullanır ve kelimeler sese dönüştükçe anlaşılır hale gelir. Özellikle imlası bozuk olabilecek veya kısaltılmış kelimeler kullanılacak olan sosyal medya mesajlarında ses kelime ile benzer ise, ses kullanmak etkili olur (thank u, b4). Vardığımız sonuçlara göre önerdiğimiz bazı özellik tanımları mevcut araştırmalara kıyasla doğruluk/kesinlik açısından ilerleme gösteriyor.

ACKNOWLEDGEMENTS

I would like to express my deep gratitude to my advisor, Dr. Dilek Günneç for guiding and supporting me during my whole graduate school life. Without her participation and assistance, the completion of this work could not be possible.

I wish to thank the committee members: Dr. Dilek Günneç, Dr. Erinc Albey, Dr. Evren Güney for the time allocated to this work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Social media is a platform where millions of people, with similar or different opinions on matters, can come across and freely share their thoughts and express their opinions about anything. As social networks become increasingly popular, companies start to seek ways to utilize available content to their benefit. Information provided by individuals over online social networks allowed them to assess their own or competitor's brand image, identify business risks, evaluate marketing campaign effectiveness or determine target audience.

These tasks are accomplished by processing the posted messages by a person or a group of people and determining their emotions. Having a giant flow of messages across a singe social media every day, it is impossible for a person to manually go over each message and determine the emotion it expresses toward a specific subject. A field of Sentiment Analysis addresses this problem and has developed a certain methodology to determine the emotion of the text automatically. This process is known as Sentiment Analysis in Natural Language Processing [1]. More specifically it is named as polarity classification when the goal is to determine whether the text expresses a positive or negative opinion. However, the accuracy of automatic classification is yet far from human in major domains.

1.1 Sentiment Analysis

Sentiment Analysis (SA) is part of Natural Language Processing (NLP) and focuses on opinion mining which identifies and extracts subjective information from a source to determine the active state such as polarity. Usually, besides identifying the polarity, Sentiment Analysis aims to determine: Subject(the subject being talked about),

Opinion Holder (the entity which expresses opinion), etc.

In general, sentiment analysis used to detect emotions. Emotion detection aims to determine the polarity of a given text such as positive, negative or neutral. This information can be used to evaluate the public emotions toward the product, company or election campaign.

With the power of opinion mining, SA found its application in different areas such as business, politics and sports. In business, Sentiment Analysis helps to answer many questions including decision making by determining the polarity of the public toward a particular subject; a business can make a decision to modify a particular product, assess the performance of the marketing campaign, determine potential customers and others. In politics, depending on the attitude of the population, the politicians may switch their election plan or put more effort to justify the need for a particular action.

1.2 Limitations of the existing methods

Sentiment analysis is a well-studied and active research area where models have been developed [2] and tested over many applications. However, polarity classification is still challenging for text obtained from online social media (such as Twitter) [3], since such text does not carry rich contextual information and is most likely to contain misspellings, shortened words.

First of all, Social Media (SM) usually have limitations on the message size and some users do not want to write long texts to express their ideas in broad details. This makes polarity classification task challenging since in such setting messages carry limited contextual information [4].

In addition, SM permit free language and messages may be written on mobile devices while doing another activity such as walking, i.e., they do not necessarily have draft versions written earlier for editing, which is one of the sources of misspells.

To address the problem of misspellings in SA, an additional normalization engine is used to derive the correct word [3] or a specialized dictionary.

Moreover, limitation on message size forces a major fraction of users to use shortcuts, for example: "u"-"you", "4ever"-"forever". The majority of this shortcuts phonetically carry the same information as original words. Therefore, the readers quickly understand what a shortcut means, unlike the machines. This problem usually solved by lookup in the specialized dictionaries that have a list of shortcuts and associated formal word or expression with it [3]. However, no one knows when a new shortcut might emerge and which structure it might have. Such dictionaries should be continuously updated manually and millions of messages have to be processed to determine whether there exists a shortcut that does not exist in the dictionary.

1.3 Sentiment Analysis using Audio Information

Recent research shows that when we read silently our brain shows neural activity that is similar to when we read aloud, suggesting that the part of understanding happens by reading the word inside our heads as we see the characters [5]. Brain processes the letters or words by generating a sound in our head and finds the association with the words we already know and then it becomes clear what a letter or word means. In other words, what we read becomes clear in our minds after it is transformed into sound. Moreover, the collection of sounds are merged and matched with our experience to find and understand the meaning of the word. This suggests that the way a human reads and understands a text may differ from the assumptions of the tools used in existing Sentiment Analysis studies. It can be further improved with audio features and without the need to keep specialized dictionaries, by implementing a model which will mimic the human brain reading process. Audio processing is relevant especially for messages that include misspellings (such as sorrry, haappy, etc.) and short forms of words (such as, thank u, gr8, b4, etc.)

1.4 Twitter Messages

This work focuses on analyzing the sentiment of messages on the Twitter social network. Twitter was chosen because it contains data from different population segments and imposes one of the most restrictive limitations on the message size. A particular message can have academic style, contain slang, shortcuts or misspellings which creates difficulties for modern solution techniques since the model should handle each writing style; as a result, may require more training data for more accurate results.

Research conducted in this study aims to define new feature sets for polarity classification on social media networks. Sentiment Analysis is a well studied area but still needs improvements as the "language" of social networks continue to change and develop. The main contribution of this work includes a new definition of feature set designed using language theories and the procedure designed to decrease computational time and complexity to derive them. The thesis is organized as follows: Chapter 2 presents related works. Chapter 3 presents the methodology of deriving and applying the proposed features. Chapter 4 defines the dataset used to evaluate the performance of the classifiers with features used in the Bag-of-words model and features proposed in this work and computational results, namely accuracy of the classifiers with given feature sets. Chapter 5 draws some conclusions and presents future extensions aiming to increase the classification accuracy and decrease the computational time of extraction proposed features.

CHAPTER II

LITERATURE REVIEW

Polarity classification is one of the most fundamental problems in sentiment analysis. It allows for identifying whether a given text is positive or negative (in comparison to identifying specific emotions or opinions). It can be completed at several levels such as sentence (the polarity of a sentence) or document-level (the polarity of a large text). Entity-based (sentiment based on a particular entity such as a company) are also available [6]. There are two main techniques in the literature for polarity classification of a text or document: lexicon-based models and machine learning models [7]. Lexicon-based models focus on text features such as words and characters and make a polarity decision based on term-counting using existing vocabularies [8]. Machine learning is a learning approach which does not differentiate features and can utilize both semantic and other features. Machine learning techniques outperform lexicon-based models in most of cases. Many recent work on machine learning relies on the following classifiers as parts of their learning approach; Naive Bayes, Logistic Regression, Support Vector Machine and Neural Networks. These models are widely used because they provide high accuracy and do not require any particular feature set (i.e., features are highly adaptable and can be adjusted based on application purposes) [9]. In this study, we take a machine learning approach and utilize all listed classifiers, as all of them have their advantages and disadvantages in handling features, which at the end affects the overall classification performance. This allows us to determine the best suitable classifier that can handle the proposed features.

There are three streams of research that are relevant to our study. First two are

related to polarity classification on Twitter messages mainly on tweet text characteristics: feature set and Data Structure Construction and Text Normalization. Third stream is related to the Language Theory. The first focus on identifying the set of features and dataset dimensions for classification. The second relate on normalizing the text so that misspellings and short (and sometimes alphanumeric) words are corrected. Recently, [10] provide a review on sentiment analysis and [11] present a polarity analysis framework for Twitter messages together with a polarity analysis overview. Third, the language theory, gives the foundation of language and the way that humans brain learned to communicate. In the next three subsections, we cover the most related literature in these areas and explain how our study differs from them.

2.1 Feature Set and Data Structure Construction

When a classifier is designed, feature selection can be seen as the critical step of the process [12]. Classification models mentioned earlier require structured input as features, i.e., the feature vector size should be fixed and have the same dimensions for each input. Therefore, to perform sentiment analysis, text should be transformed into a structured form as an input data vector where each entity in vector carries a value defined by the feature set designer [13]. This stage creates the input for the classification model.

Although feature selection is flexible, input vector construction is a challenging problem in Natural Language Processing since determining meaningful features that would help to increase model accuracy is difficult [14]. Feature sets may be determined by a trial and error approach which, as expected, do not guarantee finding of an optimal feature set for polarity classification. Forman et. al. [15] present an extensive empirical study that compares twelve feature selection methods for text classification, and there's no common opinion on which attributes a feature set should contain [16, 17, 18].

Instead of using a predefined feature set, several studies address the systematical feature set selection techniques [19, 20, 21]. They consider linguistic and statistical measures to improve classification performance by including only features that contribute the most to the accuracy [19]. Another approach is to filter out sentences or words that are either not important or that can mislead the model, such as phrases in quotations [21]. For sentiment analysis on textual data, problem domain plays an important role. For example, online reviews are great sources for sentiment analysis. Dave et. al. [22] present unique properties in this setting and propose a method based on information retrieval techniques for feature selection. Gezici et. al. [23, 24] introduce a feature set based on subjective sentence occurrence statistics, delta-tf-idf weighting of word polarities and sentence-level features using online reviews on TripAdvisor data. An important observation is that such techniques do not provide a universal feature set; each domain has its own most important features which should be considered during sentiment analysis [20].

A simple and fundamental approach to build a feature vector is by using a Bag-of-Words (BoW) model. This model utilizes all words in the training data referenced as vocabulary [25]. Despite the name of the model, it is not limited to words. Vocabulary can contain word combinations or n-grams. An n-gram simply refers to the combination n words. For example, in the sentence "An example of n-grams", "An" is an example of a unigram, "An example" is an example of a bigram and "An example of" is an example of a trigram. In general, BoW model uses a vocabulary of n-grams. Classical feature definitions for this model would be the count of n-grams observed in a given text and present in the vocabulary, or a binary definition which implies the presence of the n-gram, in the input text and the vocabulary [26]. In our work, we use a similar model. However, our vocabulary contains sound frames and the feature vector is a binary vector indicating the existence of a particular sound frame in our vocabulary or not. Having similarities between the approaches (vector

definitions), we use the Bag-of-Words model with binary features as a benchmark model for performance evaluation.

2.2 Text Normalization

Text collected from online social platforms is in unstructured data form type for sentiment analysis. We can expect misspelled words and they should be considered in their original forms as they may have a critical role in deciding the polarity of the given text. Additionally, over time, networks have developed their "own languages"; they are less formal and may contain shortened words or abbreviations, where several different short versions may represent the same word. This creates a problem as the majority of existing models rely on the language vocabulary. All short versions should be determined and only the root word should appear in the vocabulary for better classification [3]. Another common approach is to use dictionaries with the list of short or misspelled words [27]. However, such dictionaries should be continuously updated as language on social networks changes over time. Therefore, relevant data preprocessing techniques should be applied and text domain should be included in the process [28, 29] either in the feature vector generation step or inside the classification model.

Unsupervised learning can be used to overcome such challenges and normalize words. In particular, Cosine Similarity is used to determine the correct form of a misspelled word by checking the similarity of the unlabeled word and word in the vocabulary [30]. Another method to find similarity between two words is the Longest Common Sub-Sequence Ratio [31]. Similarity approach decreases the influence of the unknown words on the models because, in classical models, incorrect words are marked as unknown words and do not contribute to the classification process. It also increases the accuracy of the models regardless of the feature sets. The accuracy can further be improved by using contextual information to determine the best match

between the unknown word and the known correspondent in the vocabulary. Contextual information discards the highest similarity between words in favor of the most similar word within the context [32].

An alternative to similarity checking between words is transformation of text to an abstract form which is then transformed back to the correct version of the word. One of such abstract forms is the sound representation. It can be implemented using a Text-To-Speech (TTS) engine (transforming text to sound) and a Speech-To-Text (STT) engine (transforming sound to text) [33]. This approach demonstrates a high performance by increasing model accuracy even when the same feature set definitions are used. [33] is one of the first studies to benefit from audio to correct mistakes in text in the preprocessing stage for sentiment analysis task. They successfully transform the misspelled word into sound and convert it back to the correct text form of the word. However, this approach potentially creates two error factors made during both transformations: incorrect mapping to sound and back to text. This is especially true if TTS and STT engines do not use compatible algorithms. This suggests that the sound of a well-written text generated by TTS is not identical to the text obtained by the SST engine. In our study, the text is converted into sound and all work is done using the sound representation which reduces the error risk factor to one.

2.3 Language Theories

The language is the main form of communication for human beings. However, why we mainly use vocal to transform the information instead of gestures or any other form is not yet clear for science. Various theories are trying to explain the origin of language and speech, but most of them agree that language carries information that has associations which are transmitted to us biologically or gained by experience (learning process). The Behaviorist Theory, where one of the basic tenants implies that the primary medium of language is oral and written form is second, also share

this statement [34]. Moreover, other language theories also rely on the theory that humans learned to speak by mimicking animal voices, that was relatively easy to mimic and associated with danger and in time different voices emerged with different associations.

While we are reading, we have certain associations with shapes or sounds [35]. One of the valid interpretations is that we see a shape which is associated with a sound and as a sequence of this vision, the mind builds a full sound which is corrected by the sound patterns that we have seen in the past from experience. One of the recent studies showed that over 80% of participants reported that they hear inner voice during reading, but this cannot be measured and validated or rejected [36].

In this work, we will have an assumption that the human brain does not directly understand the characters but have associations, mainly with sound. Having a sound representation of text (using Text-to-Speech Engine), we will develop a framework that will derive features from the sound that can be used in the classification models. The main reason to move toward this direction is that shortened or misspelled words will have less changes on the sound representation, as their phonetic will be similar to the target word phonetics.

CHAPTER III

METHODOLOGY

Our feature set definitions are based on recent observations of how the human brain processes text. In particular, feature extraction from the sound representation of the text is derived using Text-To-Speech (TTS) engine. It is critical to have a proper TTS transformation so that the phonetic structure of a set of characters (word, misspellings, shorthands) is correctly transformed into the sound; meaning creates the same phonetic structure as human while reading. One of the examples could be mapping "you" and "u", they are phonetically the same and have the same sound representation. Having feature definitions, the following processes should be accomplished to train and learn the polarity of any input text.

The process of training models with a given dataset is summarized in Figure 1. First of all, we need training data for the classifiers to learn the relation between entries in the feature vector and polarity class. Here, the *Training Data* refers to the collection of the Twitter messages with corresponding polarity class label. Training data should be transformed into an abstract form called feature vector or feature set to be used in a classifier. The entries of the feature vector can take any value type depending on the definition of the features and not supposed to be of the same type; one can be the count of a word occurring in the text, other can be the presence/absence

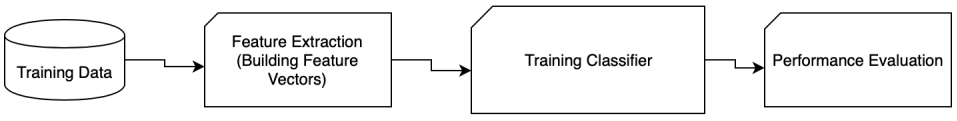


Figure 1: Polarity Classification Training Flowchart.

of a particular punctuation mark. Having feature vector for each tweet in the database and the corresponding polarity label, a classifier is trained to map the features into the correct label (polarity).

The classification of a tweet that did not participate in the training data is shown in Figure 2. Using the same feature set definition during the learning process, the feature vector must be extracted so that the classifier can process a new tweet and determine the polarity class. By feeding the feature vector into the trained classifier, the polarity of the tweet is determined.

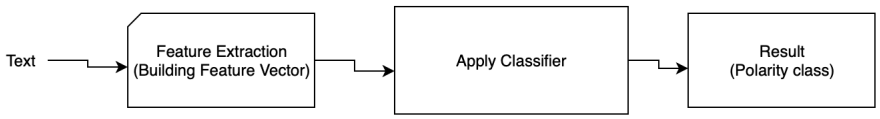


Figure 2: Polarity Classification Flowchart.

This chapter is organized as follows: Subsection 3.1 presents approaches considered in this work to convert text to speech; Subsection 3.2 describes features we have developed and evaluated in this study; Subsection 3.3 introduces the data set used in this study and describes data prepossessing and feature extraction mechanism; Subsection 3.4 provides background and a summary of the classification models used to evaluate the performance of proposed feature sets.

3.1 Text-To-Speech (TTS)

While converting text into speech (sound representation of the text), the proper transformation techniques should be applied to increase the accuracy of the models. The focus of this section is to evaluate two techniques for turning text to sound. First, a heuristic approach; mapping a letter character to its sound representation. Second, using a TTS engine that implements complex transformation processes to generate sound from a given text.

3.1.1 Array Representation

This is the heuristic approach based on the assumption that each letter is fully readable in the word (letter by letter pronunciation). Creating a sound representation of a word includes a simple search for each character in the database for its sound representation and combining all results in the same sequence in which they appear in the text. This creates the complexity of converting $O(n)$, where n is the number of characters in a given text.

A database includes all characters and numbers in the alphabet, of the given language with their corresponding sound representations as shown in Table 1. In this table column **Letter** corresponds to the character in the alphabet, **Representation** holds pronunciation of the letter in the form of amplitude data, **Phonetic** carry the phonetic form of the letter; for example, using a lookup table as shown in Table 1, the word "absolute" will be formed as follows: [a', b', s', o', l', u', t', e'] ([[eI],[bi:],..., [i:]]

Table 1: Lookup table for building text array representation (amplitude array).

Letter	Representation	Phonetic
A'-a'	[0.3, -0.4, 0.5,.....,0.9]	[eI]
B'-b'	[0.18, 0.1, -0.5,.....,-0.25]	[bi:]
...	[..., ..., ...,]	[...]
E'-e'	[0.41, -0.53, ..., 0.135, -0.24]	[i:]
...	[..., ..., ...,]	[...]
Z'-z'	[0.0, 0.23, ..., -0.23]	[...]

It can be seen that this approach does not provide high-quality transformations. Using this approach, the sound representations of "you" and "u" are not equal to each other by looking on the phonetics: $[wa\imath][o\upsilon][ju:] \neq [ju:]$.)

3.1.2 Software

Nowadays, Text-To-Speech engines provide high-quality transformations. Using well established (commercial or open-source) engine, we can obtain sound representations without adding bias to the sentiment of sound, as they do not enrich the sound with the additional intonation based on the context. In our study, the open-source eSpeak engine is used to transform data into the sound form.

The extraction of the sound amplitude performed using FFMPEG software that allows extracting amplitude of target audio, which then will be used to generate features for the classification models. The FFMPEG is an open-source multimedia framework that allows working on audio and video data[37].

3.2 Feature Set

Having the text and the tool to extract sound representations, the next step is to determine the features and the methodology to derive them for further use in classification models. Since there is no predefined methodology to follow to determine the feature sets, we will focus on two concepts for the feature extraction from sound data. First, statistical information of the sound amplitude where we will focus on histogram and transition matrix will be used. Secondly, frequent sound sequences appearing in the sound amplitudes of training data will be used.

3.2.1 Statistical Information

3.2.1.1 Histogram

The amplitude of the sound gives us complete phonetic data, and we need to derive a standardized data summary of the data that can be used as a feature set. The amplitude data is a sequence of floating-point values, and the histogram can be used to summarize the full amplitude data which will provide a fixed number of features. In the context of the histogram, features are defined to be bins for all audio samples

generated regardless of the length of the amplitude data. The original range of values an entry in the sequence can take is (-1,1) in the continuous domain; therefore, creating a bin for each unique value that cannot be fitted in the existing bins is not practical, as in the worst case it will require an infinite number of bins.

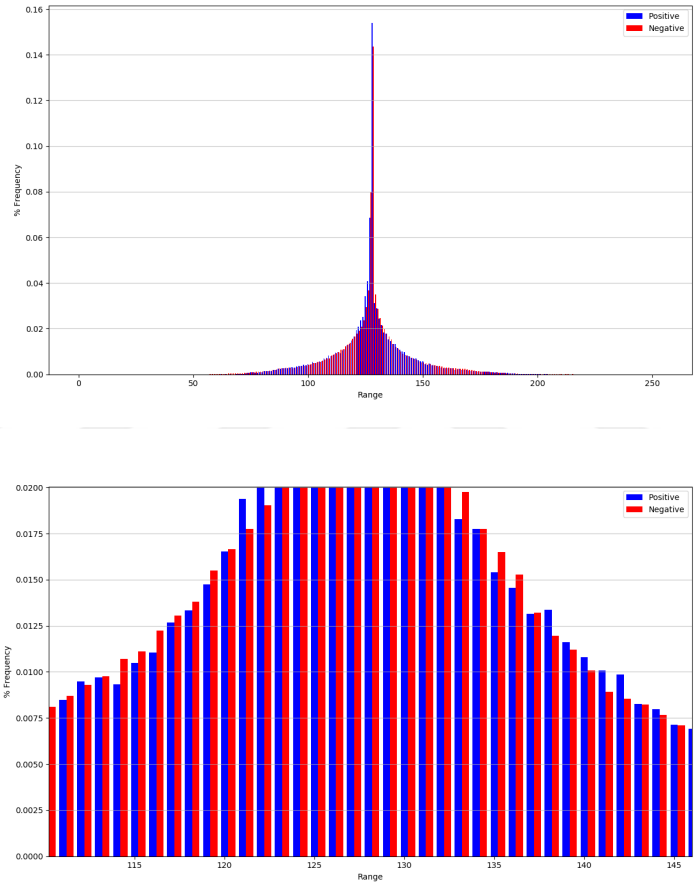


Figure 3: Histogram with domain (0, 255).

We generate a finite number of bins, where each bin covers a range of the same size in a continuous region. In our case, the range in the continuous space given by (-1, 1) will be converted to the (0, 255) range in the discrete domain. This allows splitting the data into intervals of the same size, where the length of the interval is controlled by upper and lower bounds of the discrete region.

In Figure 3, a histogram with the discrete domain range from 0 to 255 is presented

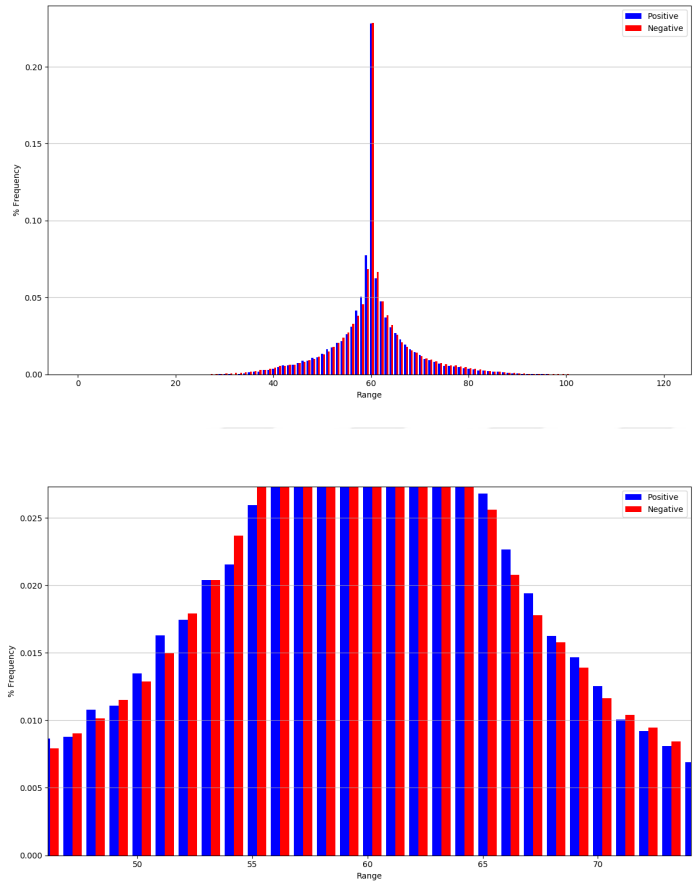


Figure 4: Histogram with domain (0, 125).

for one positive (positive polarity) and one negative (negative polarity) data samples. From the figure, we can see that the shape of the distribution is almost symmetric and we can use half of the histogram for the classification purposes. However, in Figure 3 we can take a closer look at the slice of the histogram and see that histogram is not symmetric for both positive and negative sentiment data samples; thus all bins should be used. However, by changing the bounds of the discrete domain, we can decrease the number of bins without losing the shape of the histogram and significant changes in the bar structures. Comparing Figure 3 and Figure 4 (which is histogram using domain range from 0 to 125), there is no significant change in the shape of the histograms. Moreover, we can see that the bars in both figures do not significantly

change and carry almost the same information as in the original histogram.

The empirical results show that having a histogram with domain range where upper bound is greater then 100, does not change the shape and the bins information. For the rest of the computations, we will focus on the histogram with the maximum number of bins as 126 which is equivalent to the mapping to the discrete domain between 0 and 125.

In Figure 4, we can see the difference for each bin value between the positive and negative classes. For the most of the cases, difference barely exceeds 0.02% and even decreases as we go toward the tails of the histogram, which makes calculations for the computational machines harder and the final results may differ from machine to machine. This is explained by the way the particular CPU and Operating System (OS) can manage decimal places. A solution to this problem is to multiply each bar by a large constant which helps to lower the impact of the decimal places by moving them closer to tents.

Implementation

Feature extraction for each Twitter message is accomplished using Algorithm 1. First, the message has to be converted into the audio representation from which the amplitude array is extracted and supplied to the algorithm. Along with the amplitude data, a discrete range should be defined for the algorithm; in our case, the lower bound and upper bound are set to be 0 and 125 respectively. This allows mapping of the amplitude data into discrete range by using corresponding lower and upper bounds of the raw amplitude data and newly defined discrete range. Our histogram will include a normalized count for each value that can be faced in the described domain regardless of its appearance in the remapped amplitude data. Since the histogram values of different Twitter messages might have different value counts in the histogram (one value can appear much more frequently in one tween than in another one), and the classifier might encounter a specific value of a particular bar that can appear

during the classification of a particular tweet; therefore, we have to normalize bar values so that the sum of the values adds up to 1 and are not negative .

Algorithm 1: Building Histogram Features.

```
function buildHistogramFeatures (a, lower_bound, upper_bound);
Input  : Sound representation (amplitude array) a
Output: Histogram b
c ← map(a, lower_bound, upper_bound);
histogram_index ← enumerate(lower_bound, upper_bound);
b ← empty array of size |histogram_index|;
for i ← 0 to |histogram_index| step 1 do
    b[i] ← c.count(histogram_index)/|c| ;
end
return b * 10000
```

3.2.1.2 N-Chunk Histogram

A single word may change the overall polarity of the sentence. Consider a sentence "This is not a good choice.". In such cases, a single histogram cannot reflect several changes, so the classifier will not be able to understand that this proposal carries a negative attitude. This problem can be solved by breaking the sound representation into pieces of N size and generating a separate histogram for each chunk. The result is demonstrated in Figure 5, where each histogram represents the consecutive $33.\bar{3}\%$ slice of the amplitude data. This increases the feature set by N where N is the number of chunks but will be able to detect the negative/positive sentiment at each chunk (a portion of the sound) separately since now the feature set is more sensitive to the changes by aggregating fewer data points.

Figure 5 represents the combined histogram for 3-Chunk, the chunk size value at which the model showed its best performance. From the figure, we can see for this particular instance, that the difference between positive and negative is majorly determined by the first portion of the histogram (1^{st} chunk) since values between the two classes have the highest difference at this chunk.

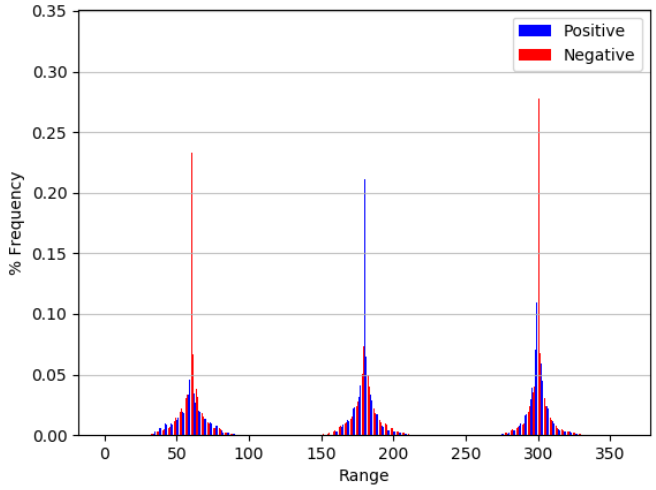


Figure 5: Histogram with 3-Chunks.

The first thing to notice is that in different sentences different histogram chunks will have different influence on the overall sentiment and there is no direct and robust relation between histograms which can create a struggle for most of the classifiers that are built on the concept of probabilistic relation; in other words, each histogram is independent.

Implementation

Feature extraction procedure of N-chuck histogram is similar to the previously defined feature extraction method for the regular histogram. Instead of focusing on complete data, we consider N-regions separately and create a histogram for each region, as shown in the Algorithm 2. In this work we use 3-chunks. Before generating a normalized histogram for each chunk we have to define the size and the boundaries of the data that will be used to generate them. As a final step, we merge these histograms into one to build a single vector which will be used by the classifiers.

3.2.1.3 Transition Matrix

The transition matrix is another way to create normalized input data for the classifier using audio amplitude data of different length size. The transition matrix is

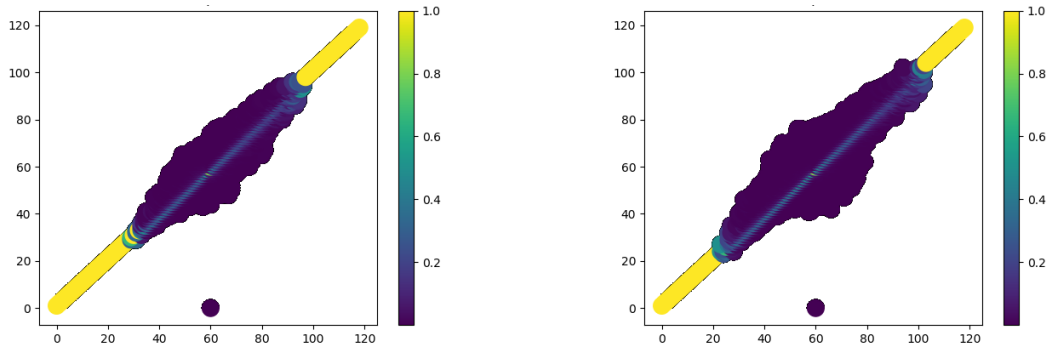
Algorithm 2: Building 3 Chunk Histogram Features.

function build_3chunk_HistogramFeatures $(a, lower\_bound, upper\_bound)$;
Input : Sound representation (amplitude array) a
Output: Histogram b
$c \leftarrow map(a, lower\_bound, upper\_bound)$;
$histogram\_index \leftarrow enumerate(lower\_bound, upper\_bound)$;
$b \leftarrow$ empty array;
$chunk1 \leftarrow c[: |c| \times 0.\bar{3}]$;
$chunk2 \leftarrow c[|c| \times 0.\bar{3} : |c| \times 0.\bar{6}]$;
$chunk3 \leftarrow c[|c| \times 0.\bar{6} :]$;
foreach *chunk* ***as*** *ch* **do**
    $d \leftarrow$ empty array of $size|histogram\_index|$;
    for $i \leftarrow 0$ **to** $|histogram\_index|$ *step* 1 **do**
        $d[i] \leftarrow ch.count(histogram\_index)/|ch|$;
    end
    $b.append(d)$;
end
return $b * 10000$

a square matrix used to describe the transitions of a Markov chain. In the context of audio amplitude, the transition matrix shows the probability of observing the upcoming amplitude value given the current value. The generalized probability matrix represents the signature of the audio amplitude.

The probability matrix is generated for each audio separately by scanning all transitions in the amplitude array and counting the number of transitions from state i to state j. In Figure 6 we have two transitions matrix for two different audio samples. Transition matrix shows the probability of a system moving from a state i to state j, which is in our case moving from amplitude i to j. From the figure, we can observe that the transition matrix generates the "signature" of the audio amplitude data. However, we cannot conclude that in general, each transition matrix will be unique, matrix only creates a summary of amplitude array and does not guarantee uniqueness.

Transition matrix can be converted to a one-dimensional array where each transition from state i to state j will be treated as a feature. Before constructing a



(a) Sample1 (b) Sample2

Figure 6: Transition Matrix representation of two audio samples.

transition matrix of audio amplitude, we have to map each value in the array to a discrete space as in case of histogram described in the previous section. Assuming the discrete space defined as $1, 2, \ldots, N$, the transition matrix will have dimensions of NxN and the one-dimensional array, which is an input vector, has dimension $1xN^2$.

Implementation

Algorithm 3 is used to generate features based on the transition matrix. For the transition matrix, it is required to define a state space. In this case, the values encountered in the mapped amplitude array from continuous space to discrete are used as states. To calculate the transition matrix of a given data, we have to count the transitions from one state to another; as row values have to add up to one it is necessary to divide the count by the total number of transitions from a given state. To build a single feature vector, transition matrix is transformed into a single array, by appending each row values in the same order into an empty array.

3.2.2 Frequent Sequences

Similar to existing Sentiment Analysis tools where words or sequence of words are used as features [25], the n-grams are used in the Bag-of-words model to derive the binary features which indicate the presence of the n-gram in the given text and vocabulary [38]. In the case of text, the features can be easily generated using tokenization [39].

Algorithm 3: Building Transition Matrix based Features.

```
function buildTransitionMatrixFeatures (a, lower_bound, upper_bound);
  Input  : Sound representation (amplitude array) a
  Output: Transition Matrix (single vector representation) b
c ← map(a, lower_bound, upper_bound);
matrix_index ← enumerate(lower_bound, upper_bound);
transitions_matrix ← zeros(matrix_index,matrix_index );
b ← empty array;
foreach i ∈ matrix_index do
    foreach j ∈ matrix_index do
        transitions_matrix[i][j] ← c.count_transitions(from i to
          j)/(c.count_transitions(from i to any )) ;
    end
end
for i ← 0 to |matrix_index| step 1 do
    b.append(transitions_matrix[i]) ;
end
return b ∗ 10000
```

Depending on the problem, a vocabulary can represent a collection of a single word (uni-gram) or two and more consecutive words (n-grams). The time complexity of generating n-grams $O(m)$, where m is the length of the pattern (using Knuth Morris Pratt algorithm).

Since text is an unstructured data type, it is not known where and when a particular word (token) can be. Therefore, if we want to have a feature set (in a Bag-Of-Words model) that will cover all possible token orderings that might be faced by the model, the feature set should cover all possible words in the language. Processing of such vector is computationally expensive. To overcome this complexity, the only frequency appearing tokens can be included. Following the Zipf's Law, if a word is frequent in one document, it is most probable that it will be frequent in others. If we look at the frequency distribution of words in any book, it will be similar to overall words frequency distribution across all books and even language [40].

In the case of audio amplitude, we can use frequent sequences (frames) as features. However, determining these frequent sequences set is another challenging task.

Firstly, we need to determine all possible n-sized sequences. Secondly, all obtained sequences have to be matched and compared to count the number of appearances of each sequence; hence, the frequent sequences. Since we don't know the boundaries or size of the frames which will be frequent, we have to consider all possible sequence lengths (i.e. from 1 to size of the array) which can be implemented using Sliding Window Algorithm. It starts from the initial element of the array and moves by one element. The procedure of obtaining all sequences include the repetitive call of Sliding Window Algorithm with different window sizes. General application of the Sliding Window Algorithm (SWA) is to find a maximum, minimum or sum of k consecutive elements in the array. The algorithm gets the maximum complexity value, $O(n)$, when the window size is equal to 1 where n is the length of the array. In our setting, the algorithm will be run for window sizes S $\in \{1, 2, ..., |A|\}$, where A is the given sound information array. Thus, the complexity of the complete algorithm is $O(n^2)$, where $n = |A|$.

Table 2: All sequences generated for array [0, 1, 2, 3].

Window Size 1

Iteration	Array Values			
1	0	1	2	3
2	0	1	2	3
3	0	1	2	3
4	0	1	2	3

Window Size 3

Iteration	Array Values			
1	0	1	2	3
2	0	1	2	3

Window Size 2

Iteration	Array Values			
1	0	1	2	3
2	0	1	2	3
3	0	1	2	3

Window Size 4

Iteration	Array Values			
1	0	1	2	3

Consider the following example: Assume that we're given an array of length 4: [0, 1, 2, 3]. The minimum and maximum sequence lengths possible are one and four, respectively. To obtain all possible sequences we need to run the SWA with the

Table 3: Number of sequences generated form array of size n with a given window size.

Window Size	# of Sequences
1	n
2	n-1
3	n-2
...	...
n	n-(n-1)

following set of window sizes [1,2,3,4]. The obtained sequences are presented in Table 2. We can see that as we increase the window size by one we obtain one less sequence. The number of sequences generated by SWA for all possible window sizes is shown in Table 3 where n is the length of array. It is directly dependent on the size of the array and can be obtained by Equation (1).

$$n + (n-1) + (n-2) + ... + (n-(n-1)) = n^2 - \sum_{i=1}^{n-1} i \tag{1}$$

When a tweet is converted into audio, the size of the corresponding array is approximately 70,000. Figure 7 shows that computing sequences of an array of size, for example, between 45,000 and 50,000 requires around 95 seconds. Assuming there are about 50,000 tweets in our dataset, it would take around 55 days only to compute all possible frames. This time does not include the time for comparing and determining the frequent subsets to decrease feature set space.

To determine set frequencies we first tried using Sequential Pattern Mining (SPM). SPM is a data mining approach specialized for analyzing sequential data and generally applied to find frequent sets in the database. One of the most well known algorithms that are applied in this field is Apriori Algorithm [41], which has both time and space complexity $O(2^{|D|})$, where $|D|$ is the number of sets. However, a more advanced filtering technique was required to decrease the number of frequent items since the most frequent frames can be subsequences of other frequent sets as some frames might be "too frequent". It would require additional computational power to determine

them. That means, a frame can appear in many other audio files, and will not play a role in the sentiment.

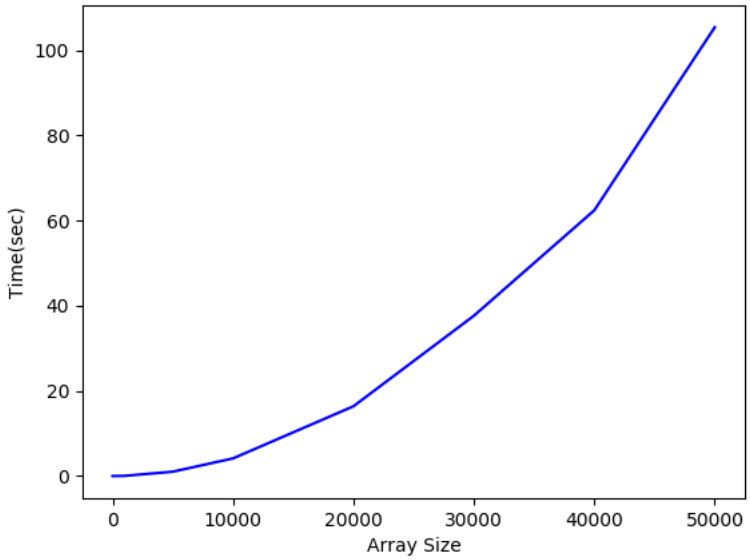


Figure 7: Array size vs. computational time in seconds.

The overall computational time required can be decreased if we can determine frequent audio frame boundaries or decrease the audio amplitude array representation size by mapping it into an abstract form. This abstract form is actually already given in the form of the original text. Since Twitter messages are restricted to 280 characters, from Figure 7 we can see that there is a significant drop in the computational time and it will take almost 0.1 seconds to compute for one tweet. Therefore, instead of the SPM approach, we determine frequent words or n-grams first and then transform the obtained vocabulary into audio frame sets. We should still note that using text will not help to handle misspelled or shortened words. We will give details about it in the next part.

We do not have an efficient algorithm to derive frequent audio frames. Using text will not help to handle misspelling and shorthand; our final vocabulary may also contain a shorthand version and the original word when they are frequent. Reducing such cases would allow to include more n-grams into a vocabulary that can increase

the prediction accuracy to the model by adding more information, which will be done using finding similarities between unknown tokens sound representation with sound vocabulary.

To decrease the computational time of learning and classification, the n-grams obtained from the text will represent their corresponding sound frames. If a particular n-gram is not directly found in the vocabulary, its sound frame will be compared to determine if there is a shortened or misspelled version. If there is a similarity above a predetermined threshold comparing sound representations, we conclude that this n-gram is present, otherwise, we do not consider this n-gram at all and decide that it should not be in the frequent set.

Misspellings and Shorthands

Misspellings and shorthands are usually ignored because the main stream of research focuses on formal texts where misspells and shorthands occur less frequently. The social media, on the other hand, imposes the conditions where user unintendedly makes mistakes during typing the text and does not correct them. Restriction on message size also forces users to use shorthands. In classical text-based sentiment analysis, some attempts that handle misspellings and shorthands have been developed. For the misspellings, the minimum edit distance is used to find the word a user wanted to use. If an unknown word to the model is faced, the minimum edit distance will be calculated concerning each word in the dictionary. It should be mentioned that in the steamed dictionary version it is harder to determine the misspellings. A general way to handle shorthands is to have a finite state machine or a separate vocabulary to make lookup and obtain the original word or expression.

In this proposed model, when an unknown word is encountered, its sound representation and the sound representations in the vocabulary will be compared using cosine similarity. Cosine similarity is a metric used to measure the similarity between two input vectors by measuring the cosine of the angle between them. In this study,

the vectors correspond to the sound representation of the text, in particular words including misspelled and shortened words. (Since two vectors have to be of the same size, zero-padding will be applied to the shortest vector.) Cosine similarity is given by Equation (2), where A and B are two vectors. The measure takes a value between -1 and 1, where -1 represents a perfect dissimilarity and 1 represents a perfect similarity.

$$similarity\,(A, B) = \frac{AB}{\|A\|\,\|B\|} \tag{2}$$

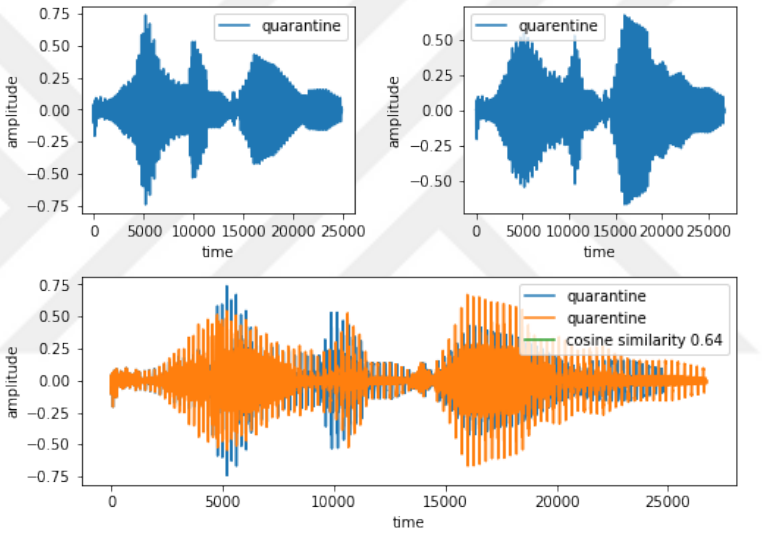


Figure 8: Cosine similarity of the word "quarantine" and its misspelled version "quarentine".

Figure 8 demonstrates the cosine similarity of the word "quarantine" to it's misspelled version "quarentine" when their sound amplitude data are compared. In the top, we have the amplitude data for both words. At the bottom, we can see the overlay of both sound representations. The cosine similarity is calculated as 0.64.

The performance of an algorithm for correctly identifying misspelled or shortened words is measured with a capture rate measure. We compute the capture rate using a list of misspelled or shortened words and their correct forms (about 350 words) taken from an online dataset [42]. Correct forms are transformed to sound representations

and their cosine similarity to the misspelled words' sound representations are measured. To increase fairness of the rate, two types of words are added: (i) words for which misspelled versions are absent, and (ii) a list of shortened words together with their target words. Using original sound forms, the capture rate of misspelled and shortened words is 44%.

Algorithm 4: Framing Algorithm 1.

```
1 Function remap (a);
  Input  : Sound representation (amplitude array) a
  Output: Modified sound representation (amplitude array) b
2 step ← 10;
3 b ← empty array;
4 for i ← 0 to |a| − step step_size step do
5 |   b.append (average (a[i:i+step]));
6 end
7 return b
```

To increase capture rate we propose implementing Algorithm 4 before using cosine similarity. With consecutive frame steps of 10, we are calculating the average of the sound signal within the obtained frame. This will make cosine similarity measure less sensitive since the values between two vectors will be less fluctuating. Implementing this algorithm increases the capture rate up to 59%.

The Algorithm 4 can further be extended to the Algorithm 5. It takes the original sound amplitude data of a Twitter message as an input. The main idea behind the algorithm is to switch all sound amplitude data to take positive values. From Figure 8, we can observe that the positive and negative amplitude data are almost symmetrical, so first, we convert all negative values in the amplitude data to positive. This, in future steps, will provide more stable average values, since negative values will not cancel positive values. Next, we compute window average values. For example, if we have an amplitude data which has 40 entries (value of array at a given index), and our window size is 10, our new amplitude will contain 4 entries where the first takes the average value of the first 10 entries in the original amplitude data, and

second entry takes the average of the second 10 (between 10 and 20) entries, etc. The resulting array of average values will represent the modified amplitude data that will be used to compute similarity. From Figure 10, we can see that the symmetry provides a clear picture on the shape to compare two words and the cosine similarity increases from 0.64 to 0.84. Mismatching is critical between times 1,500 and 2,000, which is located near the middle of the sound array, which is the same place where the misspelling occurred between the two words. In our application, the window size is determined using preliminary results. Figure 9 show the change of capture rate with respect to window size. The peak can be observed at window size 10, with a steady decrease toward window size 0. This is caused by adding some fluctuation noises. However, that noise becomes sharper as we move in the opposite direction. Having a window size set to 10, we are able to achieve a higher capture rate of 66% (in comparison to 44% as explained above).

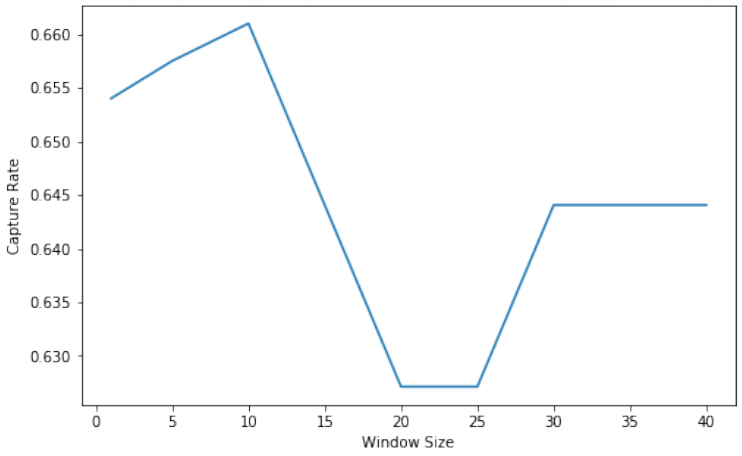


Figure 9: Capture Rate vs Window Size.

Algorithm 5: Framing Algorithm.

```
Function remap (a);
  Input  : Sound representation (amplitude array) a
  Output: Modified sound representation (amplitude array) b
for i ← 0 to |a| step 1 do
    if a[i] < 0 then
        a[i] = −a[i] ;
end
step ← 10;
b ← empty array;
for i ← 0 to |a| − step step_size step do
    b.append (average (a[i:i+step]));
end
return b
```

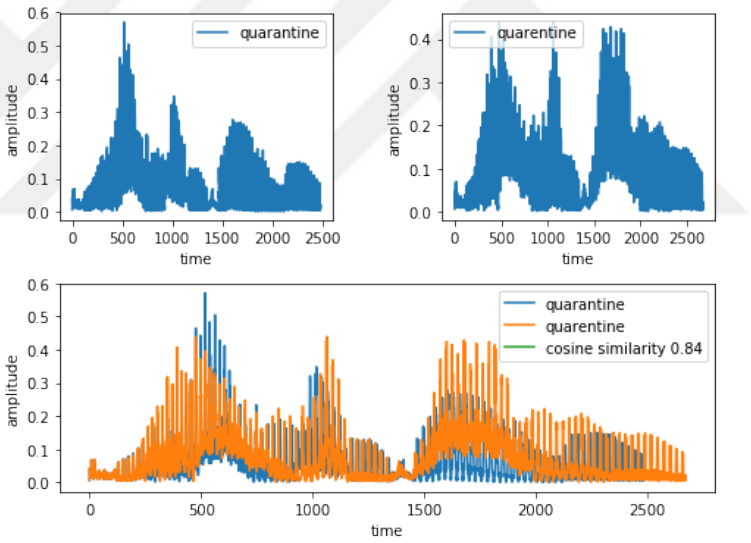


Figure 10: Cosine similarity of words "quarantine" and "quarentine" after applying Algorithm 5.

Inheriting Context Information

Inheriting contextual information could further increase the capture rate. This can be obtained if neighbors (words before and after) of the misspelled or shortened word could be included in the classification. If we can identify probabilities of two words following each other, we can utilize this in identifying the correct sound match.

For this purpose, we use Markov Chains and create a transition matrix. Markov Chain is a stochastic model that describes the sequence of possible events where the probability of observing a value in a sequence depends on the previous value in that sequence.

The probabilities can be summarized in the square matrix, called the Transition Matrix, where rows represent a preceding word in the text and columns represent the word appeared after. Given a preceding and a current word, we can see the probability (proportion of time) of observing such a transition between the words. The transition matrix can be approximated using Algorithm 6. Having a vocabulary and training data, transition matrix is estimated by counting the number of transitions in training data from one word, word A, to another, word B in the vocabulary. Before applying the algorithm, the training data should be preprocessed and stop words should be removed. Then the counted value is divided over the total number of transitions from word A. This process is repeated for all words in the vocabulary.

Using the Transition Matrix, we do not necessarily pick the most similar word as the correct form, but instead we generate a list of candidates by setting a certain threshold for cosine similarity. Any words in the vocabulary that passes this threshold is added to a list of candidates. To make the best selection from the candidate list we need the contextual information. In our case, we use the probability of seeing a word in a matched set $\bar{B}$ (composed of words whose cosine similarity is greater than 0.80) given A is the word that precedes the misspelled or shortened word. We can determine the most probable correction by using Equation (3).

$$argmax_{B \in \bar{B}}(P(B|A)) \tag{3}$$

We cannot use the previous dataset [42] to determine the transition martix, since it contains only single words. Therefore, we have modified around 200 tweets by replacing two thirds of all correct words with misspellings and shortened words in each tweet and kept the correct version for checking if matching was done correctly.

Algorithm 6: Estimating Transition Matrix.

```
Function estimateTransitionProbablities (tweet_messages, vocabulary);
  Input  : Tweet Messages tweet_messages, Vocabulary vocabulary
  Output: Transition Matrix transition_matrix
transition_matrix ← zeros[|vocabulary|][|vocabulary|];
transition_count ← zeros[|vocabulary|][|vocabulary|];
for i ← 0 to |vocabulary| step 1 do
    for j ← 0 to |vocabulary| step 1 do
        transition_count[i][j] ←
         tweet_messages.count_transition(vocabulary[i], vocabulary[j]);
    end
end
for i ← 0 to |vocabulary| step 1 do
    transition_matrix[i][j] ← transition_count[i][j]/sum(transition_count[i][:
     ]);
end
return transition_matrix
```

Algorithm 7: Cosine Similarity Matching With Transition Probabilities.

```
FunctionCosineSimilarityMatchingWithTransitionProbabilities
  (word, preceding_word, vocabulary);
  Input  : Word to match word, Preceding Word preceding_word, Vocabulary vocabulary,
           transition_matrix
  Output: Matched word in the vocabulary matched
candidates ← empty_array;
word_sound ← remap(toSound(word));
matched ← None;
max_prob ← 0;
foreach vocab_word∈ vocabulary do
    vocab_word_sound ← remap(toSound(vocab_word));
    if cosine_similarity(word_sound, vocab_word_sound) > 0.80 then
        candidates.append(vocab_word);
        if cosine_similarity(word_sound, vocab_word_sound) > max_prob then
            max_prob ← cosine_similarity(word_sound, vocab_word_sound);
             matched ← vocab_word
end
if preceding_word is None then
    return matched
matched ← None;
max_prob ← 0;
foreach candidate ∈ candidates do
    if transition_matrix[preceding][candidate] > max_prob then
        max_prob ← transition_matrix[preceding][candidate]);
        matched ← candidate;
end
return matched
```

Additional 300 tweets were randomly picked to compute the transition matrix. The matching is done using Algorithm 7. If an unknown word is encountered, cosine similarities of sound representations of the unknown word and the vocabulary are calculated. If a word in the vocabulary passes the threshold it is added to a candidate list. Given the preceding word and the candidate list, the best match is determined by looking at the highest transition probability from the preceding word to a candidate. If the preceding word does not exist, the best cosine similarity match is used. Using the transition matrix approach, we reach a capture rate of 70% (in comparison to 66% for [42] dataset).

Implementation

Algorithm 8 is used to generate features based on the sound frame concept. Before implementation, the transition matrix should be obtained using Algorithm 3 as it will be used to detect the best match among the candidates for correction. Also, all data should be preprocessed and stop words should be removed to build a solid vocabulary, which will be described in the following section. The message for which we want to build a feature set, is broken into the unigrams (tokens). Each token in the same order as appeared in the message is checked for persistence in the vocabulary. If it exists, then the corresponding entry in the vector attain binary value as True or 1. If not, the cosine similarity of the current token's sound representation and word's sound representation is checked. All words with similarities above a predefined threshold are kept and then the best is selected using the transition matrix as the most probable word. Keeping the previous unigram in memory allows us to use the transition matrix. If the first token is processed, meaning no preceding word exists, the word with the highest cosine similarity and above the threshold is chosen. If any of the candidates get similarly as 1, regardless of preceding token, it is selected as the best substitution. Having the best substitution, the corresponding entry in the feature vector is set to 1. This process continues until all tokens in the text are processed.

Algorithm 8: Building Features Based On Frequent Sound Frames Concept.

```
function buildFeaturesUsingSoundFramesConcept
  (text, vocabulary, transition_matrix);
  Input  : Text text, Array vocabulary, Matrix transition_matrix
  Output: Feature vector features
unknown ← empty_array;
previous_token ← None;
foreach token ∈ tokens do
    index=vocabulary.getIndex(token);
    if index not None then
        features[index] ← 1 ;
        previous_token ← token;
    else
        matched ← empty_array;
        token_sound ← remapV2(to_sound(token)) ;
        max_sim ← 0;
        max_index = None;
        for index ← 0 to |vocabulary| − 1 step 1 do
            word_sound ← remapV2(to_sound(vocabulary[index]));
            cos_sim = cosine_similarity(toke_sound, word_sound);
            if cos_sim == 1 then
                features[index] ← 1 ;
                max_sim ← 1;
                break;
            else if cos_sim ≥ 0.8 and previous_token not None then
                matched.append(vocabulary[i]);
            else if cos_sim ≥ 0.8 and max_sim < cos_sim) then
                max_index = index;
        end
        previous_token ← token;
        if max_indexNotNone then
            features[max_index] ← 1 ;
        if previous_token not None then
            matched_word ← None;
            max_prob ← 0;
            foreach candidate ∈ matched do
                if transition_matrix[preceding][candidate] > max_prob then
                    max_prob ← transition_matrix[preceding][candidate];
                    matched_word ← candidate;
            end
            index ← vocabulary.index(matched_word);
            features[index] ← 1;
end
return features
```

3.3 Dataset

We have used the "Sentiment140" dataset from Stanford University [43] for test and training. This dataset contains 1.6 million tweets with corresponding sentiment labels. Following six fields are present in the dataset: *Target*, which is the sentiment of the message, represented with negative as 0, neutral as 2 and positive as 4. *Ids*, the ID of the message in the Twitter database and *Date*, the date when the message was posted.

This dataset is collected automatically using Twitter API and it was not human labeled. The sentiment was decided on the "personal level" of the message, thus based on the emoticons in the tweet. For example, if a tweet contained the sign ":)" then it was labelled as a positive sentiment, and in case it contains ":(" it carried a negative sentiment. Data contains only positive and negative sentiment labels with no bias (800,000 words for positive and 800,000 words for negative sentiments).

3.3.1 Data Preprocessing

Digging into data, the first thing to notice is that data is not in traditional form. Sentiment analysis techniques were focusing on the large texts, usually paragraphs (IMDB movies review dataset is a perfect example). Twitter, on the other hand, imposes limitations on the length of the message posted by the user, which is 140 characters, excluding links and other HTML attributes that might appear in the message but not shown to the public, for example, non-breaking space is '&*nbsp*;.' So before moving to the feature generation, the data has to be cleaned from the raw data into standard text format.

HTML Decoding

Since the data provided in the dataset is in the raw web format, it contains special characters represented with specific sequences, so web compilers can understand where actual HTML code and where plain text is. Following table provides some example

of special characters and their activation that appear in the raw tweet format.

Result	Description	Entity Name
	non-breaking space	&*nbsp*;
<	less than	&*lt*;
>	greater than	&*gt*;
&	ampersand	&*amp*;
”	double quotation mark	&*quot*;
′	single quotation mark (apostrophe)	&*apos*;

Hyperlinks

Some tweets emerge in the following pattern: a reference to video/photo and comment on it. This is done by creating an external link in the background to the video/photo which appears in the raw format of the tweet as HTML link. Such hyperlinks are removed.

Hashtags

A hashtag is a form of “keyword” in social media. It helps to categorize posts in a way other users can find and follow them which proceed by the # character. In general, applications prefer to remove hashtags. In this work, we do not remove hashtags but removing the proceeding # character. Most of the hashtags refer to a place, action, characteristic or person. In some cases, the whole tweet can be written with a set of hashtags. We remove all hashtag signs and keep the actual phrases.

Emoticons

The emoticon is a representation of an expression in a different form, usually done by the various combination of the characters to generate a facial expression. Nowadays, they transformed into the more complex form, mobile devices and social media are providing their own set of emoticons that carry visual expressions, which is not clear in the raw text form. Therefore, they are removed in the prepossessing stage.

Character sequence

In some cases, users express their emotions directly in the text using a sequence of the same character, for example, "owsoooooommmmmme", that indicates a huge surprise, or shock, etc. Since for a machine this is equivalent to a new word, such sequences should be removed. In general, it is done using regular expressions in data preprocessing stage.

3.3.2 Bag Of Word and Frequent Uni-Grams

The sound frame frequency model proposed in this thesis, with the assumption of representing the frames in the form of text, brings us to the classical sentiment analysis model known as a bag of words. The idea is to determine the frequently appearing words and use them as the features. The resulting input vector for the classifier would be have binary entry, indicating whether a word did appear in the text (tweet) or not. Considering the same setting in the form of sound frames, the "word" will represent our sound frame in the vector, because the comparison of the string is less computationally expensive. In the case when text comparison did not find a match, the sound representation of the word in the dictionary (bag of words) and the word from the tweet we are trying to classify) will be used.

Having a minor difference in the application method, but the "similar" final dictionary allows us to use the same procedure to generate an initial bag of words.

A naive way to generate the feature set is to include in our dictionary all words (frames) that we have observed. However, having all observations as a feature in our input vector will increase the complexity of our model which can decrease the accuracy by creating contradictions in the data (when the same word appears equally frequent in at least two classes), or the words rarely appear in general. This can be seen in Figure 11, in the list of top frequent words across document we generally observe stop words.

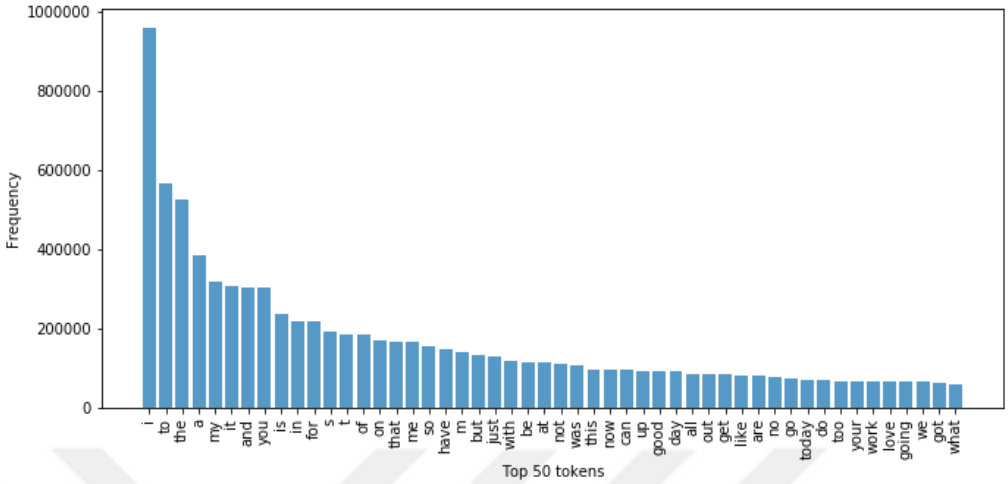


Figure 11: Top 50 words (tokens) in dataset.

Moreover, the frequency distribution is similar but not exactly same to the distribution of the words in the English documents, which is observed to follow the Zipf's distribution; i.e., the most frequent word will occur approximately twice as often as the second most frequent word, etc. Removing the stop words, we observe the more steady decrease in the frequency as the rank increases (we are ignoring the intermediate ranks and not changing the underlying distribution). In Figure 12 we can observe this effect and can observe that remaining set of words occur less often thus might have a higher impact on the model during classification.

From Figures 13 and 14 we can observe that the remaining vocabulary contains words (frames) that can more frequently appear in the positive class than in the negative class; for example, the word "love" appears more frequently in the positive class than in negative, and word "like" is approximately equally likely to be present in both classes. Moreover, we can observe that the frequency decay, except the first three most frequent words in both sets, follows a similar pattern.

Extracting the stop words allowed to decrease dictionary size, but can it be decreased even more? By looking at the frequency of words in both classes, presented in Figure 15 we can see that in general, words less frequent in both classes at the

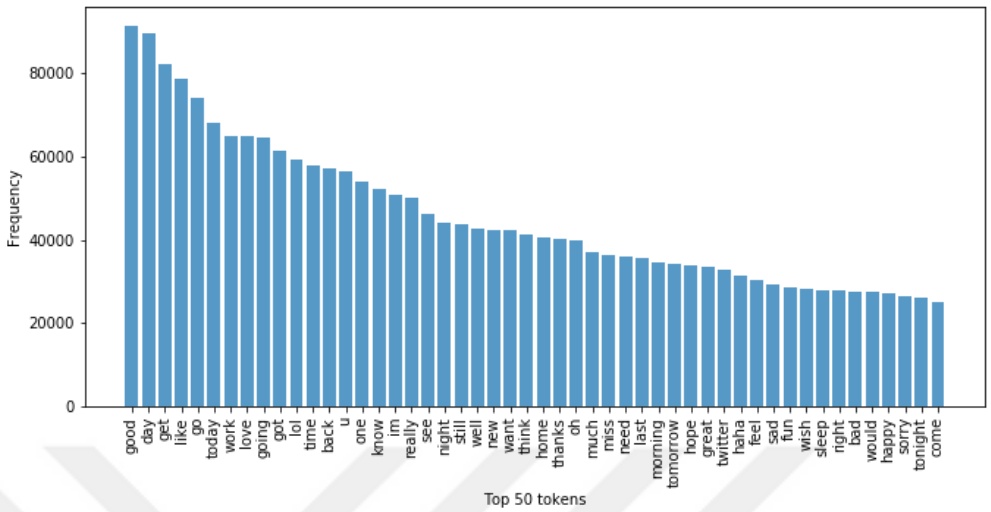


Figure 12: Top 50 words (tokens) in dataset (excluding Stop Words).

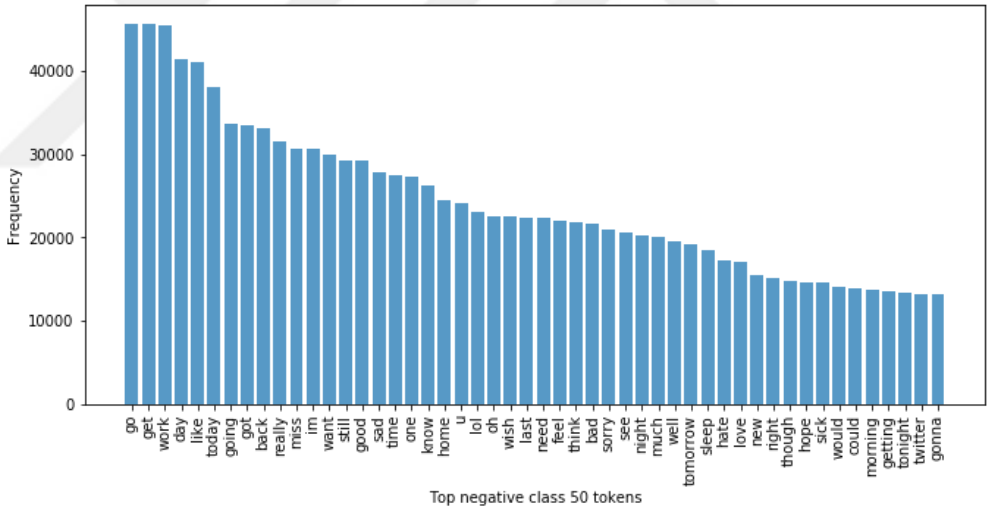


Figure 13: Top 50 words (tokens) in negative class.

same time does not carry much information individually about the sentiment of the sentence they appear in. Therefore, we would like to include the most frequent items in our dictionary and chop off the less frequent, by setting the minimum threshold requirement to the frequency. The experiments showed that the best accuracy can be achieved by setting the limiting dictionary to have a top 4,000 words.

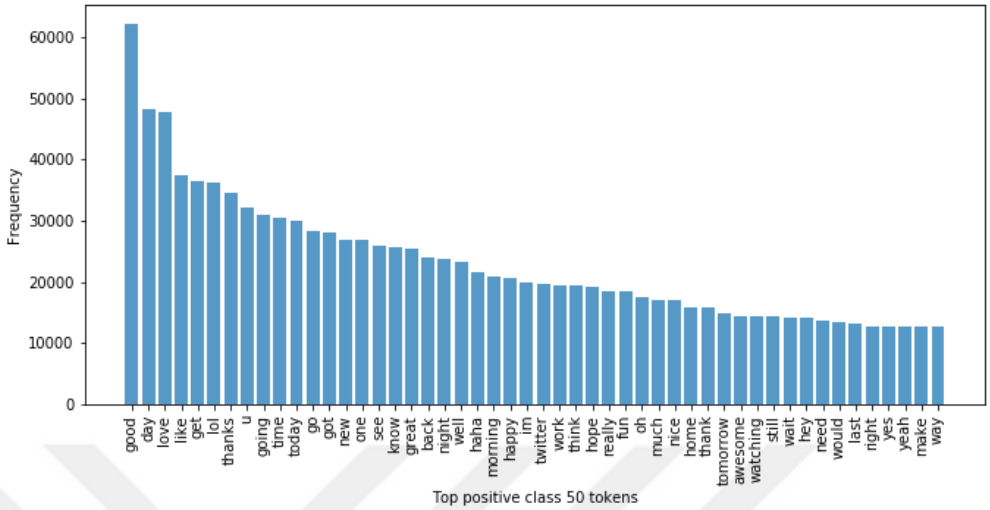


Figure 14: Top 50 words (tokens) in positive class.

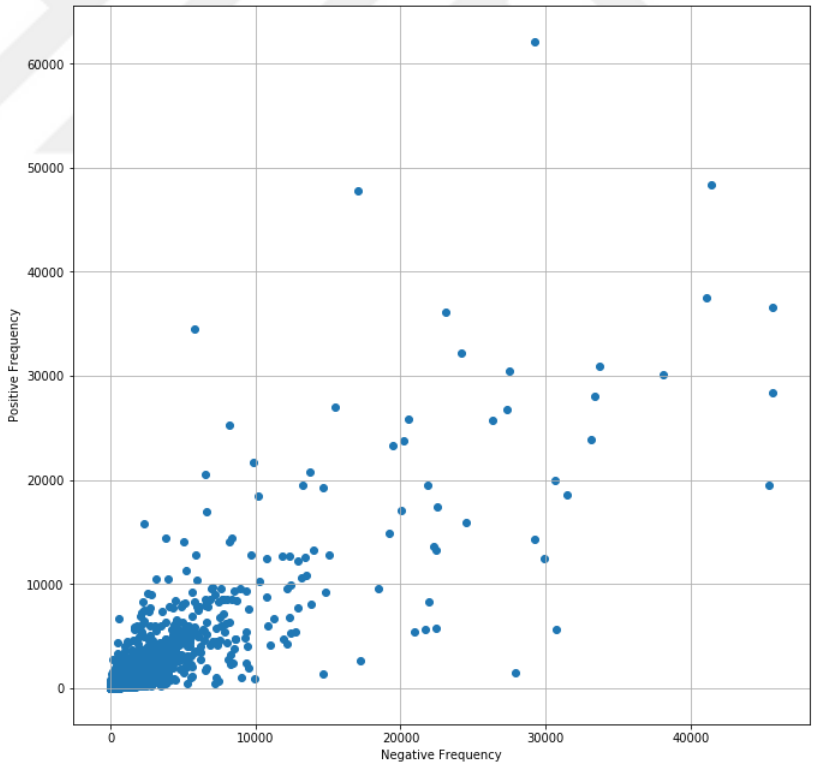


Figure 15: Word frequency in positive and negative class.

To decrease the potential number of frames, the steamed forms of words are used do define vocabulary. This allows capturing more words as frequent by removing

the different forms of the same word and accounting its frequency to the root word. Moreover, obtained vocabulary allows to compare the performance of the BoW model in the literature and the proposed model by using the same word-based vocabulary.

3.4 Classifiers

For measuring the performance of the purposed features, we use classic classifiers in the field of sentiment analysis: Naive Bayes Classifier, Logistic Regression, Support-Vector Machine and Multi-Layer Perceptron.

3.4.1 Naive Bayes Classifier

The Naive Bayes classifier is a probabilistic classifier based on the Bayes theorem that relies on the assumptions of independence between the features. It was introduced in the 1960s, based on the works of Rev. Thomas Bayes $(1702 - 61)$, and remains as a baseline method for the text categorization. Naive Bayes classifier is computationally inexpensive and does require huge training data. However, it is often outperformed by other techniques such as boosted trees, random forests, Max Entropy, Support Vector Machines, etc.

Naive Bayes is a conditional probability model: given feature set represented by a vector $\vec{x} = (x_1, ..., x_n)$ it assigns the probability of each instance to belong to a class C_j where j is set of all classes (possible outcomes). Using independence assumption and imposing a condition that we pick the most probable hypothesis, the Naive Bayes Classifier is the function that assigns a class label $\hat{y} = C_k$ for some k:

$$\hat{y} = argmax_{j \in \{1,...,K\}} P\left(C_j\right) \sum_{i=1}^{n} P(x_i|C_j) \tag{4}$$

3.4.2 Logistic Regression

Logistic Model is a statistical model using the logistic function to model binary dependent variables. Logistic Regression is the process of estimating parameters of

the logistic model. It was developed by statistician David Cox in 1958 and found application in many fields including medicine, politics, finance, etc.

The logistic function is a sigmoid function that takes input $t \in \mathbb{R}$, and maps it into 0/1 space.

$$sig(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^t} \tag{5}$$

Assuming t is defined by the linear relationship of elements in vector $\vec{x} = (x_1, ..., x_n)$, given by:

$$t(\vec{x}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n \tag{6}$$

and corresponding logistic function:

$$sig(\vec{x}) = \frac{1}{1 + e^{\beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n}} \tag{7}$$

where β is a vector of logistic regression coefficients, usually estimated using maximum likelihood estimation method.

3.4.3 Support-Vector Machine

Support-vector machine (SVM) is a supervised learning model that is capable of handling both classification and regression problems. It constructs a set of hyperplanes that maximize the margin distance between data points or set of data points, and can be used for outlier detection, classification, regression, etc. The SVM is usually selected for its capability of handling non-linear data. Using Kernel-Trick, SVM able to map non-linearly separable data to the dimension where data becomes separable.

The Support-Vector Classification (SVC) focuses on minimizing the expression of the form

$$\left[\frac{1}{n}\sum_{i=1}^{n} max(0, 1 - y_i(wx_i - b))\right] - \lambda \left\|w\right\|^2 \tag{8}$$

where $\vec{w}$ is normal vector to the hyperplane, $wx_i - b = 1$ when $y_i = 1$ (belongs to class 1), and $wx_i - b = 1$ when $y_i = -1$ (belongs to class -1)

The optimization model is given by:

$$\text{minimize } \frac{1}{n}\sum_{i=1}^{n}\zeta_i - \lambda \|w\|^2 \tag{9}$$

$$\text{s.t. } y_i(wx_i - b) \geq 1 - \zeta_i \ \forall i \tag{10}$$

$$\zeta_i \geq 0 \ \forall i \tag{11}$$

where

$$\zeta_i = max(0, 1 - y_i(wx_i - b)) \tag{12}$$

3.4.4 Multi-Layer Perceptron

Multi-Layer Perceptron is a supervised learning model that approximates a function $f(x)$ by training on a dataset. The simplest model consists of a minimum of three layers: input layer, a hidden layer, output layer. The input layer is the input to the neural network which has the same dimensions as $\vec{x}$ and takes the same value. Next layer (hidden layer) takes a value defined by a linear combination of all nodes in the previous layer. There might be more than one hidden layer depending on the data complexity (linear/non-linear). The output layer uses results from the last hidden layer and transforms them into the final output. The output range may differ depending on the activation function; thus, the output layer may contain more than one output which can be transformed to desired output externally.

For the binary classification, the sigmoid and softmax functions are usually selected. The sigmoid function is given by the logistic function, this the output is given in range (0,1) which is suitable for binary classification and require only one node in the output layer. Softmax provides the index of a node where the maximum value is obtained, thus the number of output nodes should be equal to the number of classes trying to fit. In general form the complete network is summarized by the following

equation:

$$y = \vartheta(\sum_{i=1}^{n} w_i x_i + b) = \vartheta(w^T x - b) \tag{13}$$

where w the vector of weights, x is the vector of features, b is the bias, ϑ is any activation function.

CHAPTER IV

COMPUTATIONAL RESULTS

This section introduces computational results for the models discussed in this study, namely: Classical Bag of Word, Sound Cosine Similarity, Histogram based, N-Chunk Histogram based, Transition Matrix based models and Text Cosine Similarity. Text Cosine Similarity is a BoW model which utilizes the same principle as in the frequent frequencies approach, but if a word does not exist in the vocabulary, it is matched using regular cosine similarity with a word in the vocabulary using the text form. We use it for a fair comparison in our performance evaluations. The runs were taken on the computer with processor Intel i7, 16GB of RAM, Ubuntu 16.04 OS.

Table 4: Data sizes used in computational experiments.

Number of messages	Positive	Negative	Frequent words
2,000	1,000	1,000	2,000
10,000	5,000	5,000	4,000
100,000	50,000	50,000	4,000
200,000	100,000	100,000	4,000

Numerical results are obtained using four different subset sizes as shown in Table 4. The first three columns show the number of messages, number of positive and negative messages. For example for the first row, 1,000 messages were randomly chosen from positive words and another 1,000 were randomly chosen from the negative word set. However, the random 30% of this data will be used for the validation, therefore the training might not have exactly the same distribution of the training data. The even distribution omits the bias during the training phase and removes the probability of selecting all messages from the same class. For each dataset size, 20 different random instances are taken and the average accuracy is reported. The

purpose of using different sizes of dataset is to test the model performance in term of accuracy. By taking multiple instances, we increase the possibility of encountering with a different frequent set of words, which can affect model performance. The last column represents our choices for the size of the frequent words set used within the approach determined after preliminary results; for each dataset size, models are set to run at different vocabulary sizes, and the vocabulary size where the highest accuracy is achieved is used in the computational results presented here. Moreover, the limited dataset would result in a different set of vocabulary, depending on the tweets it contains; for example, in one instance, the word "like" might be frequent and in another, it might not appear as frequent. As dataset increases, by following Zipf's Law, we expect to the frequent word set to settle down and do not change.

Table 5: Average accuracy (%) with data set size 2,000 tweets (1,000 positive, 1,000 negative).

Classifier	Bag Of Words	Text Cosine Similarity	Sound Cosine Similarity
Naive Bayes Classifier	66.20	66.40	66.56
Logistic Regression	68.12	67.80	67.70
Support-Vector Machine	54.71	56.20	52.23
Multi-Layer Perceptron	66.00	67.00	65.76

Table 6: 95% confidence intervals of the mean accuracy (%) with data set size 2,000 tweets (1,000 positive, 1,000 negative).

Classifier	Bag Of Words		Text Cosine Similarity		Sound Cosine Similarity	
	LB	UB	LB	UB	LB	UB
Naive Bayes Classifier	64.34	68.06	62.87	69.93	64.18	68.93
Logistic Regression	66.08	70.15	64.62	70.98	64.83	70.56
Support-Vector Machine	48.04	61.37	53.30	59.10	44.19	60.27
Multi-Layer Perceptron	61.59	70.40	64.49	69.51	61.40	70.17

Table 5 shows average accuracy levels of 20 dataset instances obtained by three methods: BoW, Text Cosine Similarity and Sound Cosine Similarity. We can see that all models perform at similar levels for datasets of sizes 2,000. The best performance is

achieved using Logistic Regression with the BoW model. Over the four classifiers, we see that the features proposed in this study do not necessarily improve the accuracy of the classifiers. Further, in two cases it worsens the accuracy. In theory, Sound Cosine Similarity and Text Cosine Similarity approaches should not perform worse than the BoW model, because the input vectors are the same at the beginning. However, Sound Cosine Similarity model matches the unknown words to the known words in the vocabulary, but since the capture rate is 70%, the model incorrectly matches some of the words, which in the small dataset has higher influence. The small vocabulary is more sensitive to the incorrect matching, which creates noise for the classifiers rather then strengthening the results. The same holds for the Text Cosine Similarity as it uses similar principles. (95% confidence intervals for the accuracy results in Table 5 are given in Table 6.)

Table 7: Average accuracy (%) with data set size 10,000 tweets (5,000 positive, 5,000 negative).

Classifier	Bag Of Words	Text Cosine Similarity	Sound Cosine Similarity
Naive Bayes Classifier	71.63	71.84	72.41
Logistic Regression	72.34	71.70	71.27
Support-Vector Machine	55.00	59.24	54.47
Multi-Layer Perceptron	71.71	72.16	72.45

Table 8: 95% confidence intervals of the mean accuracy (%) with data set size 10,000 tweets (5,000 positive, 5,000 negative).

Classifier	Bag Of Words		Text Cosine Similarity		Sound Cosine Similarity	
	LB	UB	LB	UB	LB	UB
Naive Bayes Classifier	70.59	72.67	71.31	72.37	71.41	73.41
Logistic Regression	71.34	73.34	71.14	72.26	70.23	72.31
Support-Vector Machine	44.44	65.56	46.26	72.22	43.19	64.81
Multi-Layer Perceptron	70.71	72.70	69.56	74.77	71.51	73.39

In Table 7, we can observe that our feature definition starts to help the classifiers, and the accuracy is higher than in the classical definition of features used in BoW

Table 9: Average accuracy (%) with data set size 100,000 tweets (50,000 positive, 50,000 negative).

Classifier	Bag Of Words	Text Cosine Similarity	Sound Cosine Similarity
Naive Bayes Classifier	75.20	76.70	78.10
Logistic Regression	ME	ME	ME
Support-Vector Machine	ME	ME	ME
Multi-Layer Perceptron	75.84	76.35	79.63

Table 10: 95% confidence intervals of the mean accuracy (%) with data set size 100,000 tweets (50,000 positive, 50,000 negative).

Classifier	Bag Of Words		Text Cosine Similarity		Sound Cosine Similarity	
	LB	UB	LB	UB	LB	UB
Naive Bayes Classifier	75.00	75.40	75.97	77.43	77.86	78.34
Logistic Regression	ME	ME	ME	ME	ME	ME
Support-Vector Machine	ME	ME	ME	ME	ME	ME
Multi-Layer Perceptron	75.57	76.11	75.63	77.08	79.43	79.83

model for three of the classifiers. Having a larger vocabulary, the mismatch during comparison of unknown word sound representation with sound vocabulary damage the features less, and the benefit starts to dominate and strengthens the results. We can see this effect in tables 9 and 11 as well, where difference in accuracy with the BoW model starts to have a larger value and becomes almost 5% in Table 11. Note that as the data size increases some of the classifiers give memory errors (represented with ME in the corresponding tables).

Figure 16 summarizes the accuracy behavior of BoW and our model as dataset size increases. As the size grows, the accuracy increases for all models. However, the rate of increase differs where the Sound Cosine Similarity model increases with a higher rate. This can be explained by the ability of the model to capture and process misspelled and shortened words using cosine similarity. The probability of observing such words increases as data set size increases and it affects the classification. This effect can also be seen in Tables 6, 8, 10 and 12 which provide 95% confidence intervals (CI) for corresponding data sizes for 20 instances each. There it can be seen that as

Table 11: Average accuracy (%) with data set size 200,000 tweets (100,000 positive, 100,000 negative).

Classifier	Bag Of Words	Text Cosine Similarity	Sound Cosine Similarity
Naive Bayes Classifier	ME	ME	ME
Logistic Regression	ME	ME	ME
Support-Vector Machine	ME	ME	ME
Multi-Layer Perceptron	80.50	82.12	85.21

Table 12: 95% confidence intervals of the mean accuracy (%) with data set size 200,000 tweets (100,000 positive, 100,000 negative).

Classifier	Bag Of Words		Text Cosine Similarity		Sound Cosine Similarity	
	LB	UB	LB	UB	LB	UB
Naive Bayes Classifier	ME	ME	ME	ME	ME	ME
Logistic Regression	ME	ME	ME	ME	ME	ME
Support-Vector Machine	ME	ME	ME	ME	ME	ME
Multi-Layer Perceptron	79.42	81.58	81.86	82.38	84.27	86.15

the data size increases, intervals get tighter. This is explained by the features derived at each instance. As we have large text data, Zipf distribution converges to the same set; i.e., the same vocabulary.

It can be observed that the SVM has the highest accuracy variation among other models. This is caused by the imbalance of the classes in the training data. As we have the same number of classes in the training data the classifier does not create a bias toward any class and aims to find a balance to maximize the accuracy.

Multi-Layer Perceptron provided the highest result, suggesting that it is more suitable for the proposed feature set. The best accuracy achieved by our feature set is 85.21% (dataset size = 200,000), in comparison to 83% on works based on the same dataset [43].

Table 13 presents additional metric data for the Multi-Layer Perceptron for all considered data sizes for both BoW and our model that utilizes sound cosine similarity. The importance of each metric depends on the application. The Precision metric is useful to evaluate the model when the classification of positives is more important.

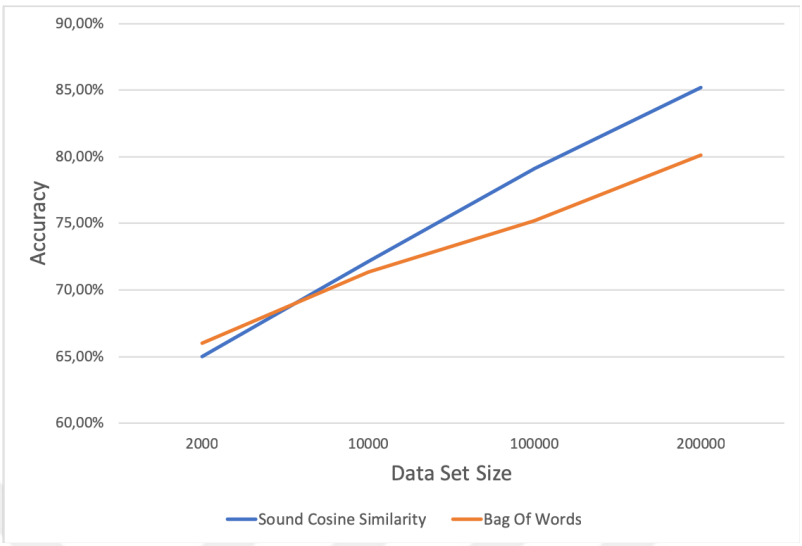


Figure 16: Average model accuracy with respect to data set size.

Table 13: Metrics Data (%) for Multi-Layer Perceptron.

	Bag Of Word				Sound Cosine Similarity			
Data Size	2,000	10,000	100,000	200,000	2,000	10,000	100,000	200,000
Accuracy	66.67	71.77	75.52	79.01	66.00	72.40	79.08	84.48
Precision	76.04	72.28	76.34	79.30	65.69	86.56	76.65	85.50
Recall	48.67	71.85	73.96	78.50	67.00	71.97	77.99	84.00
F1 Score	59.35	71.65	75.13	78.90	66.34	78.59	77.31	84.74

Further, it is a good measure to use when the cost of False Positives is high, for instance, email spam detection. In email spam detection, a false positive means that a non-spam email (actual negative) has been identified as spam (predicted spam). The email user might lose important emails if the precision is not high for the spam detection model. The Recall metric is useful when classifying positive as negative is not desirable. The F1-Score is a metric used to seek the balance between precision and recall. Moreover, F1-Score is useful when test data contain an unbalanced number of representatives of each class. By looking at F1-Score, it can be concluded that the model does not have a strong bias toward any particular class.

Features based on the statistical summary do not provide the best results. As can be seen in Table 14, the best classification accuracy at 100,000 data set size does not

Table 14: Accuracy (%) with data set size 100,000 tweets (50,000 positive, 50,000 negative) For Statistical Models.

	Histogram based	N-Chunk Histogram based	Transition Matrix based
Accuracy	55.22%	57.03%	61.41%

exceed 65%, which is obtained using other features at 2,000 data set size. One of the reasons for such pure performance is the ability of the computer to properly handle floating points which does not allow classifiers to properly determine the relation between features to derive the polarity.

From the results, we conclude that the proposed feature set (frequent frames) and its generation technique provide high accuracy results in comparison to the classical BoW model, especially for large-sized data. The Text Cosine Similarity model also performs better than the regular Bag Of Word model, but it does not beat the Cosine Similarity Model in terms of accuracy. Indeed, at the first steps of the proposed feature generation technique, the feature set within the BoW model and our feature set are identical. The difference comes when we map misspelled and shortened words using cosine similarity. The lower performance at smaller sized dataset can be explained by a higher rate of mismatch and small vocabulary size. As dataset size increases, the capture rate of misspelled and shortened words increases, which strengthens the importance of some features for the classification models. A similar effect can be seen with Text Cosine Similarity. The best accuracy achieved by our feature set is 85.21% (200,000 sentences), in contrast to 83% on works based on the same full sized dataset [43].

CHAPTER V

CONCLUSION

In this thesis, we tried to develop features for sentiment polarity classification in social media which focuses on the text with a high probability of misspelling or shorthand to occur. Proposed feature set is based on the frequent sound frequencies appearing in the sound representation of text obtained using a Text-to-speech engine. Since generating all possible sound frames is computationally expensive, without loss of generality, the written words can be used to represent frequent sound frames following the idea that if a word is frequent, then it's sound representation is also frequent. Moreover, the cosine similarity is able to match misspellings and shortened words, which are frequent in social media, on the text sound representation level.

In general, Multi-Layer Perceptron provided the highest result, which implies that it is more suitable for the proposed feature set. The largest average accuracy obtained using this model is 85.21% using a limited data set, in contrast to the 83% found in the literature.

An obvious extension of the proposed method of extracting feature set is to develop a framework that will extract fingerprints of sounds. That would allow decreasing matching complexity to $O(1)$ since fingerprint will be unique for each word (even with slight sound variations), it can be used as a table index.

Bibliography

[1] D. H. Farias and P. Rosso, "Irony, sarcasm, and sentiment analysis," in *Sentiment Analysis in Social Networks*, pp. 113–128, Elsevier, 2017.

[2] Y. Mejova, "Sentiment analysis: An overview," *University of Iowa, Computer Science Department*, 2009.

[3] U. Barman, A. Das, J. Wagner, and J. Foster, "Code mixing: A challenge for language identification in the language of social media," in *Proceedings of the first workshop on computational approaches to code switching*, pp. 13–23, 2014.

[4] J. Song, Y. He, and G. Fu, "Polarity classification of short product reviews via multiple cluster-based svm classifiers," in *Proceedings of the 29th Pacific Asia Conference on Language, Information and Computation: Posters*, pp. 267–274, 2015.

[5] L. Magrassi, G. Aromataris, A. Cabrini, V. Annovazzi-Lodi, and A. Moro, "Sound representation in higher language areas during language generation," *Proceedings of the National Academy of Sciences*, vol. 112, no. 6, pp. 1868–1873, 2015.

[6] B. Liu, "Sentiment analysis and opinion mining," *Synthesis lectures on human language technologies*, vol. 5, no. 1, pp. 1–167, 2012.

[7] C. Catal and M. Nangir, "A sentiment classification model based on multiple classifiers," *Applied Soft Computing*, vol. 50, pp. 135–141, 2017.

[8] M. Taboada, J. Brooke, M. Tofiloski, K. Voll, and M. Stede, "Lexicon-based methods for sentiment analysis," *Computational linguistics*, vol. 37, no. 2, pp. 267–307, 2011.

[9] M. M. Islam and N. Sultana, "Comparative study on machine learning algorithms for sentiment classification," *International Journal of Computer Applications*, vol. 182, 2018.

[10] J. Serrano-Guerrero, J. A. Olivas, F. P. Romero, and E. Herrera-Viedma, "Sentiment analysis: A review and comparative analysis of web services," *Information Sciences*, vol. 311, pp. 18–38, 2015.

[11] A. C. E. Lima, L. N. de Castro, and J. M. Corchado, "A polarity analysis framework for twitter messages," *Applied Mathematics and Computation*, vol. 270, pp. 756–767, 2015.

[12] J. Khairnar and M. Kinikar, "Machine learning algorithms for opinion mining and sentiment classification," *International Journal of Scientific and Research Publications*, vol. 3, no. 6, pp. 1–6, 2013.

[13] A. Agarwal, B. Xie, I. Vovsha, O. Rambow, and R. Passonneau, "Sentiment analysis of twitter data," in *Proceedings of the Workshop on Language in Social Media (LSM 2011)*, pp. 30–38, 2011.

[14] A. Abbasi, S. France, Z. Zhang, and H. Chen, "Selecting attributes for sentiment classification using feature relation networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 3, pp. 447–462, 2010.

[15] G. Forman, "An extensive empirical study of feature selection metrics for text classification," *Journal of machine learning research*, vol. 3, pp. 1289–1305, 2003.

[16] H. Jain, A. Mogadala, and V. Varma, "Sielers: Feature analysis and polarity classification of expressions from twitter and sms data," in *Second Joint Conference on Lexical and Computational Semantics (* SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013)*, vol. 2, pp. 525–529, 2013.

[17] Z. Liu, "A comparative study on linguistic feature selection in sentiment polarity classification," *arXiv preprint arXiv:1311.0833*, 2013.

[18] L. Zafar, M. T. Afzal, and U. Ahmed, "Exploiting polarity features for developing sentiment analysis tool.," in *EMSASW@ ESWC*, 2017.

[19] K. Dave, S. Lawrence, and D. M. Pennock, "Mining the peanut gallery: Opinion extraction and semantic classification of product reviews," in *Proceedings of the 12th international conference on World Wide Web*, pp. 519–528, ACM, 2003.

[20] Y. Liu, X. Huang, A. An, and X. Yu, "Arsa: a sentiment-aware model for predicting sales performance using blogs," in *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 607–614, ACM, 2007.

[21] A. Duric and F. Song, "Feature selection for sentiment analysis based on content and syntax models," *Decision support systems*, vol. 53, no. 4, pp. 704–711, 2012.

[22] K. Dave, S. Lawrence, and D. M. Pennock, "Mining the peanut gallery: Opinion extraction and semantic classification of product reviews," in *Proceedings of the 12th international conference on World Wide Web*, pp. 519–528, ACM, 2003.

[23] G. Gezici, B. Yanıkoğlu, D. Tapucu, and Y. Saygın, "New features for sentiment analysis: Do sentences matter?," CEUR Workshop Proceedings, 2012.

[24] G. Gezici, B. Yanıkoğlu, D. Tapucu, and Y. Saygın, "Sentiment analysis using domain-adaptation and sentence-based analysis," in *Advances in Social Media Analysis*, pp. 45–64, Springer, 2015.

[25] Y. Goldberg, *Neural Network Methods for Natural Language Processing*. Morgan & Claypool, 2017.

[26] J.-H. Wang, T.-W. Liu, X. Luo, and L. Wang, "An lstm approach to short text sentiment classification with word embeddings," in *Proceedings of the 30th Conference on Computational Linguistics and Speech Processing (ROCLING 2018)*, pp. 214–223, 2018.

[27] F. H. Khan, U. Qamar, and S. Bashir, "esap: A decision support framework for enhanced sentiment analysis and polarity classification," *Information Sciences*, vol. 367, pp. 862–873, 2016.

[28] S.-A. Bahrainian and A. Dengel, "Sentiment analysis and summarization of twitter data," in *2013 IEEE 16th International Conference on Computational Science and Engineering*, pp. 227–234, IEEE, 2013.

[29] A. Mittal and A. Goel, "Stock prediction using twitter sentiment analysis," *Standford University, CS229 (2011 http://cs229.stanford.edu/proj2011/GoelMittal-StockMarketPredictionUsingTwitterSentimentAnalysis.pdf)*, vol. 15, 2012.

[30] V. K. R. Sridhar, "Unsupervised text normalization using distributed representations of words and phrases," in *Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing*, pp. 8–16, 2015.

[31] C.-Y. Lin and F. J. Och, "Automatic evaluation of machine translation quality using longest common subsequence and skip-bigram statistics," in *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, p. 605, Association for Computational Linguistics, 2004.

[32] N. Desai and M. Narvekar, "Normalization of noisy text data," *Procedia Computer Science*, vol. 45, pp. 127–132, 2015.

[33] C. Kobus, F. Yvon, and G. Damnati, "Normalizing sms: are two metaphors better than one?," in *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pp. 441–448, Association for Computational Linguistics, 2008.

[34] M. Demirezen, "Behaviorist theory and language learning," *Hacettepe Üniversitesi Eğitim Fakültesi Dergisi*, vol. 3, no. 3, 1988.

[35] S. Sutherland, "When we read, we recognize words as pictures and hear them spoken aloud. [last accessed 2019-02-10]," 2015.

[36] R. P. Vilhauer, "Inner reading voices: An overlooked form of inner speech," *Psychosis*, vol. 8, no. 1, pp. 37–47, 2016.

[37] X. Lei, X. Jiang, and C. Wang, "Design and implementation of a real-time video stream analysis system based on ffmpeg," in *2013 Fourth World Congress on Software Engineering*, pp. 212–216, IEEE, 2013.

[38] G. Paltoglou and M. Thelwall, "More than bag-of-words: Sentence-based document representation for sentiment analysis," in *Proceedings of the International Conference Recent Advances in Natural Language Processing RANLP 2013*, pp. 546–552, 2013.

[39] J. Fürnkranz, "A study using n-gram features for text categorization," *Austrian Research Institute for Artifical Intelligence*, vol. 3, no. 1998, pp. 1–10, 1998.

[40] S. T. Piantadosi, "Zipf's word frequency law in natural language: A critical review and future directions," *Psychonomic bulletin & review*, vol. 21, no. 5, pp. 1112–1130, 2014.

[41] M. Hu and B. Liu, "Mining opinion features in customer reviews," in *AAAI*, vol. 4, pp. 755–760, 2004.

[42] "Commonly misspelled english words. [last accessed 2019-03-10]."

[43] A. Go, R. Bhayani, and L. Huang, "Twitter sentiment classification using distant supervision," *CS224N Project Report, Stanford*, vol. 1, no. 12, 2009.

[44] D. Bespalov, B. Bai, Y. Qi, and A. Shokoufandeh, "Sentiment classification based on supervised latent n-gram analysis," in *Proceedings of the 20th ACM international conference on Information and knowledge management*, pp. 375–382, ACM, 2011.

[45] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs up?: sentiment classification using machine learning techniques," in *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pp. 79–86, Association for Computational Linguistics, 2002.

[46] "Hootsuite. [last accessed 2019-08-10]."

[47] G. Yule, *The study of language*. Cambridge university press, 2016.

[48] B. Heredia, T. M. Khoshgoftaar, J. Prusa, and M. Crawford, "Cross-domain sentiment analysis: An empirical investigation," in *2016 IEEE 17th International Conference on Information Reuse and Integration (IRI)*, pp. 160–165, IEEE, 2016.

VITA

Mihail Duscu graduated from Orizont High School, Republic of Moldova, in 2012. He received his B.S. degree in Industrial Engineering from Özyeğin University in June 2017. In September 2017 he joined Master of Science program in Industrial Engineering and has been working under the supervision of Asst. Prof. Dilek Günneç. Participated in projects related to robotics and data mining. His research focuses on the classification of unstructured data.