Bachelor of Science in Computer Science & Engineering

Multi-Label Emotion Classification of Tweets using Machine Learning

by

Simon Islam

ID: 1504062

Department of Computer Science & Engineering

Chittagong University of Engineering & Technology (CUET)

Chattogram-4349, Bangladesh.

May, 2021

Multi-Label Emotion Classification of Tweets using Machine Learning

Submitted in partial fulfilment of the requirements for
Degree of Bachelor of Science
in Computer Science & Engineering

by

Simon Islam

ID: 1504062

Supervised by

Animesh Chandra Roy

Assistant Professor

Department of Computer Science & Engineering

Chittagong University of Engineering & Technology (CUET)

Chattogram-4349, Bangladesh.

The thesis titled '**Multi-Label Emotion Classification of Tweets using Machine Learning'** submitted by ID: 1504062, Session 2019-2020 has been accepted as satisfactory in fulfilment of the requirement for the degree of Bachelor of Science in Computer Science & Engineering to be awarded by the Chittagong University of Engineering & Technology (CUET).

Board of Examiners

Chairman

Animesh Chandra Roy
Assistant Professor
Department of Computer Science & Engineering
Chittagong University of Engineering & Technology (CUET)

Member (Ex-Officio)

Dr. Md. Mokammel Haque
Professor & Head
Department of Computer Science & Engineering
Chittagong University of Engineering & Technology (CUET)

Member (External)

Dr. Mohammad Shamsul Arefin
Professor
Department of Computer Science & Engineering
Chittagong University of Engineering & Technology (CUET)

Declaration of Originality

This is to certify that I am the sole author of this thesis and that neither any part of this thesis nor the whole of the thesis has been submitted for a degree to any other institution.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. I am also aware that if any infringement of anyone's copyright is found, whether intentional or otherwise, I may be subject to legal and disciplinary action determined by Dept. of CSE, CUET.

I hereby assign every rights in the copyright of this thesis work to Dept. of CSE, CUET, who shall be the owner of the copyright of this work and any reproduction or use in any form or by any means whatsoever is prohibited without the consent of Dept. of CSE, CUET.

Signature of the candidate

Date:

Acknowledgements

During my work in this thesis, I have received tremendous support and assistance. Animesh Chandra Roy, Assistant Professor, Department of Computer Science & Engineering, Chittagong University of Engineering & Technology is my honorable thesis supervisor and I am thankful for the guidance, cooperation, assistance and constructive criticism provided by him during the whole process. I want to thank him for the patience and endless support shown towards me. I am grateful for the opportunity.

My sincere thanks goes to Professor Dr. Md. Mokammel Haque, Head, Department of CSE, CUET, for providing his valuable support. I am thankful to all the teaching staff of Department of CSE, CUET, for providing me with valuable advices, encouragement, support and guidance. I would also like to express my gratitude towards all of my teachers and supporting staffs Department of CSE, CUET. I want to thank them for their cooperation and assistance which contributed to this thesis's completion.

Abstract

Twitter is one of the biggest social media network in the world. It has 330 million active monthly users. Users of Twitter use micro-blogs called tweets to express their opinions. This generates a huge amount of textual data every minute. Analyzing this data to search for emotions in it will lead us to understand the emotions currently presiding over the internet. This approach can be extended to other approaches like user managements and job managements. Emotion classification has a long history. Based on the approach, it can be divided into binary classification, multi-class classification and multi-label classification. Binary classification detects whether a emotion is present or not. Multi-class classification classify tweets into one of the many available classes. In this thesis, we proposed multi-label emotion classification of tweets using machine learning. In multi-label classification, it is possible to label a tweet with more than one emotion. Multi-label classification methods are divided into two groups. They are problem transformation methods and algorithm adaptation methods. A total of 13 state of the art multi-label classifiers were trained and evaluated on a tweet dataset containing 8501 tweets. 10 of them were problem transformation methods and 3 were algorithm adaptation methods. We found that, although all the classifiers performance are pretty close, problem adaptation method like Binary Relevance and Label Powerset performs better that other multi-label classifiers. We have also found that Random Forest Classifier works better than Support Vector Machine as base classifier in problem transformation methods for multi-label classification. We achieved Micro F-Score up to 0.91 & Subset 0/1 loss of 0.28. We also showed that, Senticnet5 can be used to improve the accuracy of the models. Considering our dataset contains tweets from various incidents, this represents a statistically significant improvement.

Keywords— Multi-Label Classification, Binary Relevance, Label Powerset, Classifier Chains, Trending, Twitter

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

We live in the age of information. Invention of internet has brought the world the closest its ever been since the dawn of mankind. Data is transferred between every corner of the world in rapid speeds unimaginable even a decade ago. Various aspects of human life has changed considerably to accommodate with this change. Messaging has moved to social media and email. Personal contacts are becoming scarce with technologies like video calling, social media etc. Social media will probably go down as one of the most important inventions of this century. Websites like Facebook, Twitter, Reddit etc. are shaping human consciousness all over the globe. It posses the power to empower people by making it easy to establish communication and form friendship quickly and easily. It can be said that the internet without social media will be a lonely place.

In social media, Twitter is one of the most prominent names. It is a micro-blogging website. Micro-blogs are usually small and compact in size. Twitter demonstrates this by capping the blogs size to 140 characters. But since 2017, they increased the size to 280 characters. The micro blogs in twitter are generally known as "tweet". The topics of tweets can range from various topics, pictures, gif or even sharing links to other websites. Twitter wants their reach to be global. For this reason, an unregistered user can see what a registered user has posted. And also two user needn't be mutually connected to follow each other. Twitter has achieved this global reach by committing to this one way follower relationship.

With the rapid advancement of technology, the world is advancing at a hectic pace. Current news and affairs are shifting constantly. Twitter keeps up with this rapid pace with their trending tab. It determines the trends by calculating

the volume of a keyword over a period of time. If the volume of a keyword gets spiked over a short time, it will get flagged as trend. The more the density of a keyword, the higher it will be on the trend board. Since it is a global platform, it is an excellent place for people to express their opinions and emotions. From average people to global leaders, many use twitter to express their opinion on current topics. In return the followers of the people also reply to the tweets with their own opinions. By extracting emotions from this active exchanges, it will be beneficial to understand the mindset of the people participating in the conversation. Multi-label emotion classification can be used to understand emotions in the tweets.

Multi label classification problem is related to multi output classification problem where a problem can have multiple labels assigned to it. It is different from multi class classification where only one label among multiple labels can be assigned to it. Due to a tweet having multiple emotion embedded into it, multi label classification is necessary over multi class classification. It is very difficult to classify complex human nature using one emotion. In this work, we aim to perform multi label emotion classification of tweets using machine learning. We hope to classify the tweets in 8 classes. They are: Joy, Sadness, Anger, Disgust, Admiration, Surprise, Interest and Fear. The overview of multi-label emotion classification is explained in this chapter. This chapter also includes difficulties, application, motivation and contribution of this thesis. At the end of this chapter, the organization of this thesis is presented.

1.2 Framework Overview

The objective of this work is to perform multi-label emotion classification of tweets using machine learning. Figure shows the outline of the framework. The classification will be performed in multiple steps. We will label the tweets, process them, extract features, convert them to matrix form and perform multi-label classification. Detailed explanation of the system will be provided in chapter 3.

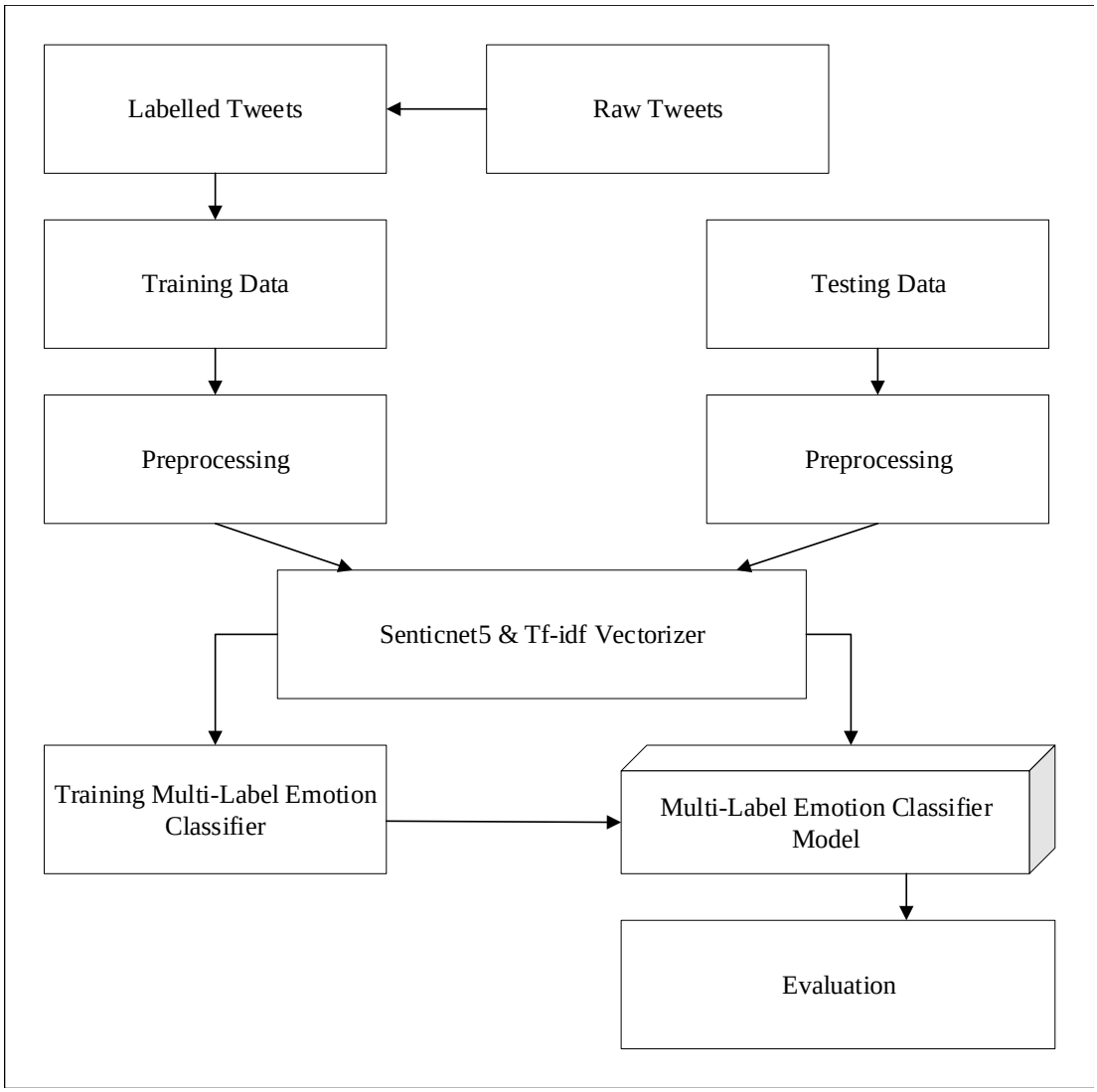

Figure 1.1: Multi-Label Emotion Classifier Framework

1.3 Difficulties

The major difficulties encountered in this problem are:

1. **Lack of datasets** Classification of tweets into multiple labels is still a new concept. Very few works have been done in multi-label emotion classification. The previous works in this field were performed with datasets containing tweets from specific incidents. Although there are many datasets with sentiment classification and multi-class classification, datasets with multi-label classification with proper emotions are really scarce. A properly labelled dataset where tweets are not related to a specific incident was

not available at the start. Thus, we began by randomly collecting tweets from various users.

2. **Bias in training data** Human beings are prone to biases. Due to the lack of raters, the dataset was not rated by multiple qualified raters. Thus, it is possible that the biases of the rater is present in this dataset. Although the labels were rechecked due minimize the amount of bias, there is still the possibility of biases being present. This may lead to errors due to biases present in the training dataset.

3. **Scope of the work** This work does not intend to focus on tweets relating to specific incidents. We want the models we train to be capable of classifying tweets of all forms. This may not provide accuracy like models trained with a specific incident in mind. Also, to encompass tweets of all types the dataset needs to be very big in size. The scope of this work is thus very big but we tried our best to manage it.

4. **Exponential complexity** In a multi-label classification problem with N labels, there are 2^N possible combinations of label orders. This exponential complexity of the problem limits various brute force approach for solving the problem. Also, for 1 correct order of labels, there could be 2^N - 1 incorrect orders. It greatly reduces the accuracy score of the models. Since the difference between correct and incorrect is very fragile, various unconventional methods are needed to evaluate the models.

1.4 Applications

The main application of this work is to classify the emotions present in tweets posted in Twitter. This can be used to get emotional state of the masses in a given time by monitoring the trends in Twitter. Other applications of this work can be:

1. Monitoring and classifying emotions in social media feeds.

2. Monitoring and classifying the emotions present in brand reviews.

3. Using in customer services to prioritize and assign jobs based on emotions.

1.5 Motivation

We live in an age of social media. It has been ingrained into our daily lives. Peeking into social media is a very good way to stay up to date with current trends. For this reason, developing models that can classify tweets can give us a good glimpse at current affairs and how people are being affected by it. Due to the variety of topics, it may not provide us with totally accurate analysis but it can give us the gist of it.

Many previous works done in the past regarding the classification of tweets has been on sentiment analysis. Some of the works also looked for a multi-class classification solution. But as we are aware, human emotion is very complex. People don't think in one emotion at a time. A human thought can be directed in multiple direction. Thus it consists of multiple emotions. For this reason, it is safe to assume that a tweet posted by a human may also contain multiple emotions. Which is why sentiment analysis or multi-class classification of tweets may not be enough to represent human psyche. We need multi-label emotion classification for this purpose.

1.6 Contribution of the thesis

A thesis or research work is done for achieving a specific set of goals. It can define a new methodology or improve upon an existing one. In this work, we tried to design multi label emotion classifiers that can classify tweets. The primary contribution of this thesis is the following:

1. Development of a tweet dataset with properly labeled emotions.
2. Training various multi label classification models with the features extracted from the dataset.
3. Using Senticnet5 to boost the accuracy of the models.

1.7 Thesis Organization

The rest of the thesis is organized as follows:

- Chapter 2 gives a brief summary of various important terms and terminology used in this work. This is followed by brief explanation and algorithms of the various machine learning algorithms used in this work. Finally, the chapter concludes with the brief summary of various works done in the field of sentiment classification, multi-class classification and multi-label classification.
- Chapter 3 describes the proposed methodology of the work.
- Chapter 4 includes the description of the dataset and evaluation of the performance of the models trained for this work.
- Chapter 5 concludes the work by giving a brief summary of the work and provide some recommendation for future improvement of the work.

1.8 Conclusion

In this chapter, an overview of multi label emotion classification of tweets is provided. We described the framework, difficulties & a summary of multi label classification models. The motivation behind our work & contributions are also noted. In the next chapter, the background and present state of the work will be provided.

Chapter 2

Literature Review

2.1 Introduction

Before delving into multi-label emotion classification, we first need to understand the evolution of the problem of sentiment classification. In sentiment classification, a text was labeled into one of two emotions. With time, this problem evolved into multi-class classification problem. In it, one text could be labeled as one of many given class. The next development in this field were multi-label emotion classification. Here, one text can contain multiple emotions.
In this chapter, we first discuss various important terms and terminology regarding multi-label emotion classification. We then discuss the various multi label classification techniques that is closely related to the proposed work. Finally, we talk about the development of multi-label emotion classifications. It is divided into three parts. They are sentiment classification, multi-class classification & multi-label classification.

2.2 Important Terms & Terminology

Some important terms and terminology related to the framework is discussed below:

- **Multi-Label classification:** It can be considered an improvement over multi-class classification. In multi-class classification, an assumption is made at the start that each sample contains only one label. Unlike multi-class classification, multi-label classification appoint all the appropriate labels for that sample. There are no limits to how many labels can be assigned

to each samples. It can be defined as mapping inputs x to binary vectors Y.

- **Emotion Classes:** According to Ekman [1], there are six basic emotions. They are anger, disgust, fear, joy, sadness, and surprise. We also added the emotions 'interest' and 'admiration' to our dataset. They help to add more expression to our dataset.
- **Tweets:** A tweet is a micro-blog that is posted on the website *Twitter*. Before 2017, a tweet could only have 140 characters. But due to it's rising popularity, Twitter extends the size of a tweet to 280 characters. It helps the user to relay their thoughts to a worldwide audience. It helps them to express themselves, their feelings and thoughts. It connects the users with other users.

2.3 Feature Extraction Methods for Multi-Label Emotion Classification

Extracting features from the tweets is an important part of this framework. This will help our models to understand the tweets better. We used *Senticnet5* [2] & tf-idf to extract features from our tweets. A brief discussion of them is given below.

2.3.1 Senticnet5

Senticnet is an initiative that began in MIT media laboratory in 2009. It has been used for emotion aware intelligent application in various fields spanning from Natural language processing to human computer interaction. It has many applications. In this work, we used it as a concept level knowledge base to enhance the training phase of our model. In *Senticnet5* [2], various word are labeled according to their primary and secondary emotions. We transformed this information suitable to our dataset and added it during training phase. Since compared to the vast amount of tweets out there, our dataset only covers a tiny

fraction. This outside help is needed to understand various words it encounters which may not be present in our dataset.

2.3.2 Tf-idf

Tf–idf stands for term frequency-inverse document frequency.It is a numerical statistics. It determines how important a word is to a document. In our case, it will determine the importance of word to a tweet. The value of Tf-idf increases the more a word appears in a document. Tf-idf is a combination of two statistics. They are term frequency and inverse document frequency. Term frequency as understood from the name, calculates the number of times a term occurs in a document.If we consider each tweet a document then term frequency, tf(t,d) is the frequency of term t in a tweet d can be written as:

$$tf(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \tag{2.1}$$

Here, $f_{t,d}$ is the raw count of term t in tweet d.

Inverse document frequency measures the amount of information provided by the word. It indicates the rarity of the word. It is logarithmically scaled. Inverse document frequency, idf(t,D) of a term t in the tweet dataset D can be written as:

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

here, N is the total number of tweets in the dataset D and $\{d \in D : t \in D\}$ is the number of tweets where term t appears.

Thus, tf-idf of a term t in a tweet d over the dataset D can be calculated as:

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

We use tf-idf to perform vectorization of our tweet dataset. It will be convert the raw tweets to a matrix of tf-idf features. As our models can't understand the meaning of written english words, performing tf-idf and converting the dataset to a matrix will help it to evaluate one term against other terms. This will help it to assign terms with emotion and improve it's classification process.

2.4 Machine Learning Methods for Multi-label Emotion Classification

Machine learning methods for multi-label classification are usually divided into two parts. They are problem transformation methods and algorithm adaptation. In problem transformation methods, multi-label classification is transformed to known solutions for sentiment classification and multi-class classification. The results are then combined to present final output. Algorithm adaption adapts classic machine learning algorithms to work for multi-label classification.

2.4.1 General Overview of the Problem

Let, T denotes the tweet dataset that will be used to perform multi label emotion classification. Now, $X = T^d$ defines the d-dimensional instance space after tf-idf vectorization. And $y = \{y_1, y_2, \ldots, y_L\}$ denote the L emotion labels present in the tweets. The purpose of a multi label emotion classifier is to perform a multi label prediction $f : X \leftrightarrow 2^y$ from the multi label training set $D = \{(x^i, y^i)|1 \leq i \leq m\}$. Here, m is the length of the training set, $x^i \in X$ is a d-dimensional feature vector & $y^i \in \{0, 1\}^L$ is a L bit binary vector where $y^i_j = 1$ indicates the emotion being present in the tweet and $y^i_j = 0$ indicating it's absence.

2.4.2 Problem Transformation Methods

Problem transformation methods transforms multi-label classification problem into sentiment classification, multi-class classification or a mix between the two using ensemble. Various problem transformation methods used in this word are described below:

2.4.2.1 Binary Relevance

Binary Relevance (BR) [3] transforms the problem of multi-label classification into a group of single label binary classification problem. It is very easy to implement and it is faster than many later methods. The number of models needed for binary relevance solution is equal to the number of models. It decomposes the

multi-label problem into independent binary learning tasks (one per label). Here, each binary learning task is performed on one emotion label present in y. To be more specific, each emotion label set, y_j, binary relevance compose a training set D_j from the original multi label training set D. Every multi label training example $(x^i.y^i)$ is converted to binary training example with relation to label y_j.

$$D_j = \{(x^i, y_j^i)|1 \leq i \leq m\} \tag{2.4}$$

After transformation, a binary classification algorithm B, i.e. $g_j = B(D_j)$ where g_j is the binary classifier for the emotion j is used to perform binary classification on training sets. All the models are trained parallelly, independent from each other. This makes is easy and simple to implement. It can be easily trained to learn from examples with missing labels. Since the tasks are independent from each others, it ignores the correlation that may be present between the labels. In a task where the labels are semi-independent of each other, binary relevance performs better than most of the other models.

An algorithm for binary relevance is shown in algorithm 1.

Algorithm 1 Algorithm for Binary Relevance [3]

Input:
D : multi-label training set
B : binary learning algorithm
x^* : test instance $(x^* \in X))$
Output
$y*$: predicted label set for x^* where $y^* \subseteq y$

1: **for** $j = 1$ to L **do**
2: Transform into binary training set D_j according to equation 2.4
3: Train binary classifier $g_j : B(D_j)$
4: $y^* : g_j(x^*)$
5: **end for**

2.4.2.2 Label Powerset

Label Powerset (LP) [4] transforms multi-label classification problem into multi-class classification. Each unique combination of labels are assigned a class. Classes are unique based on their label combination. Thus for a dataset with L labels, there are 2^L possible classes. The benefit of label powerset is that only

one model is need to be trained to distinguish between all the classes. But since the model number is exponential, it contributes to the long training time. Failure to detect a class properly will lead to a high hamming loss because of the conversion between classes and labels. But since one correct prediction means correctly predicting all the labels, label powerset has high accuracy.

We transformed our data by evaluating every y^i as a binary number and assigning the decimal representation of that binary value as the class number. This transform $y = \{y_1, y_2, \ldots, y_l\}$ to $C = \{c_1, c_2, \ldots, c_m\}$. Then we compress the class numbers from 2^L to m. This process eliminates the missing classes. Although it simplifies a multi-label problem into a multi-class problem, it also increases the runtime complexity of training the model. Also, there may not be enough data for some classes to train it properly. It will lead to training loss. And our model will only be capable of finding classes with high enough samples.

An algorithm for label powerset is shown in algorithm 2.

Algorithm 2 Algorithm for Label Powerset

Input:
D : multi-label training set
H : multi class classification algorithm
N : length of train dataset
x^* : test instance ($x^* \in X$))
Output
$y*$: predicted label set for x^* where $y^* \subseteq y$

1: **for** $i = 1$ to N **do**
2: $\quad C_i = convert_to_decimal(y_i)$
3: **end for**
4: compress $C = \{c_1, c_2, \ldots, c_m\}$ to $CC = \{1, 2, \ldots, m\}$
5: train a single multi-class classifier $G : H(CC)$
6: **return** $y* = convert_to_binary(expand(G(x^*)))$

2.4.2.3 Classifier Chains

Classifier Chains method (CC) [5] is somewhat similar to Binary Relevance. But instead of training the binary models parallelly, it trains them sequentially. And the result of previous training is used to augment the results of next training session. It maintains the efficiency of binary relevance while still maintaining the corelation between different labels. The problem arises on deciding how the chain

is formed. Since for a dataset with N labels, there are N! possible unique combinations. Thus truly determining the best chain combination actually negates the advantages of using a classifier chains. An improvement of classifier chain is **Ensemble of Classifier Chains (ECC)** [5], where several CC classifiers are trained with random combination of chains. And after training all of the models, the labels are predicted based on a user given threshold. We used majority voting for prediction. It means a tweet is labeled with an emotion if at least half of the classifier chains voted for it.

An algorithm for classifier chains is shown in algorithm 3.

Algorithm 3 Algorithm for Classifier Chains

Input:
D : multi-label training set
B : binary learning algorithm
x^* : test instance ($x^* \in X$))
Output
$y*$: predicted label set for x^* where $y^* \subseteq y$

1: **for** $j = 1$ to L **do**
2: Transform into binary training set D_j according to equation 2.4
3: Train binary classifier $g_j : B(D_j)$
4: $y^* : g_j(x^*)$
5: Append y^j to D_j {Previous classes being used to train next classes}
6: **end for**

2.4.2.4 Random k-Labelset

Random k-Labelset (RAkEL) [4] is an improvement over Label Powerset. Instead of dividing L labels into 2^{L} classes, it breaks the initial set of labels into small subset of labels. The subset are creating randomly. Then after training phase, they are ensembled to produce the final prediction. In RAkEL, k is the parameter that specifies the size of subsets. If L labels are divided into m subset of size k, the complexity will go down from 2^{L} to $m \times 2^k$ where $k < L$. In documents with large number of labels, it reduces the training time and complexity while maintaining the effectiveness of Label Powerset. It is implemented using scikit multilearn [6] package that uses the implementation of Tsoumakas *et al.* [7].

Algorithm 4 Algorithm for Random k-Labelset

Input:
D : multi-label training set
H : multi class classification algorithm
N : length of train dataset
k : length of each subset
x^* : test instance ($x^* \in X$))
Output
$y*$: predicted label set for x^* where $y^* \subseteq y$

1: $m = \lceil L/k \rceil$
2: **for** $i = 1$ to m **do**
3: $L_i = \emptyset$
4: **for** $j = 1$ to k **do**
5: y_j = randomly selected label from y
6: $L_i = L_i \cup y_j$
7: **end for**
8: train an LP classifier H_i based on D and L_i using algorithm 2
9: **end for**

2.4.3 Algorithm Adaption

Algorithm adaption methods allows the models to directly perform multi-label classification rather than transforming them to other problems. It removes the problem transformation phase and saves time in model training and gets outperformed by problem transformation methods like binary relevance. Algorithm adaption methods used in this word is discussed below:

2.4.3.1 MLkNN

MLkNN [8] which stands for Multi-label K nearest neighbors is one of the algorithm adaptation method to solve multi-label classification problem. K nearest neighbors algorithm is modified to work with a multi-label dataset. In MLkNN, during a new instance its k-nearest neighbors are identified. After identifying, maximum a posteriori (MAP) principle is utilized to determine the labels for unidentified data. MAP principle is based on the information gained from label sets of neighboring instances. It starts by finding k nearest neighbor $N(i)$ of x_i. Then the labels are identified using Bayesian conditional probability. It calculates the probability of a label being 1 or 0. Then the label with higher probabilities are defined as the labels of the sample. Let, H_1^j defines that label j is present in

the sample and H_0^j means it's absence. Now,

$$\begin{aligned} P(H_1^j) &= \frac{s + \sum_{i=1}^{N} [y_i^j \in y_i]}{s \times 2 + m} \\ P(H_0^j) &= 1 - P(H_1^j) \\ where, (1 \leq j \leq L) \end{aligned} \tag{2.5}$$

Here, N is the length of train dataset, s is a smoothing parameter. In Zhang and Zhou [8], s is set to 1 for yielding Laplace smoothing. After this, the frequency arrays c^j and $\tilde{c^j}$ are maintained with the help of $N(i)$.

$$\begin{aligned} c_j[r] &= \sum_{i=1}^{m} [y_i^j \in y_i] \cdot [\delta^j(x_i) = r] \\ \tilde{c_j}[r] &= \sum_{i=1}^{m} [y_i^j \notin y_i] \cdot [\delta^j(x_i) = r] \\ where, \delta^j(x_i) &= \sum_{(x^*, y*) \in N(x_i)} [y_j \in y^*] \; and \; (0 \leq r \leq L) \end{aligned} \tag{2.6}$$

The number of labels j in the nearest neighbors of unknown sample x^* is then calculated and maintained in C^j. Finally, each label of unknown y^* is predicted using equation 2.8.

$$C^j = \sum_{(x^*, y^*) \in N(x)} [y^j \in y^*] \tag{2.7}$$

$$y^* = \{y^j | P(H^j | C^j) / P(\neg H^j | C^j) > 1, \; 1 \leq j \leq L\} \tag{2.8}$$

The algorithm for performing MLkNN is is shown in algorithm 5.

2.4.3.2 BRkNN

BRkNN [9] combines approaches from both Binary Relevance and K-Nearest neighbor. It adopts a lazy learning approach. In this method, the k nearest neighbors are needed to be found only once. This is a big improvement from binary relevance with kNN as base classifier. Instead of running kNN $|L|$ number of times where $|L|$ is the number of labels, it is run only once. Thus, it is $|L|$ times faster than traditional BR methods. Finally, the resulting nearest neighbors are evaluated using BR to find multi labels. There are two extensions to BRkNN. They are BRkNN-a and BRkNN-b.

Algorithm 5 Algorithm of MLkNN

Input:
N : length of train dataset
k : number of neighbors
x^* : test instance ($x^* \in X$))
Output
$y*$: predicted label set for x^* where $y^* \subseteq y$

1: **for** $i = 1$ to N **do**
2: Identify k nearest neighbors $N(x_i)$ for x_i
3: **end for**
4: **for** $j = 1$ to L **do**
5: Estimate $P(H^j)$ and $P(\neg H^j)$ using equation 2.5
6: Calculate frequency arrays c^j and $\hat{c^j}$ using equation 2.6
7: **end for**
8: Find k nearest neighbors $N(x^*)$ for x^*
9: **for** $j = 1$ to L **do**
10: Calculate nearest neighbors C^j using equation 2.7
11: **end for**
12: **return** y^* using equation 2.8

Both BRkNN-a & BRkNN-b are based on calculating the confidence score for each of the labels. The confidence of a label is calculated by considering the k nearest neighbors that include it. Formally, the confidence of label j can be written as:

$$cf_j = \frac{1}{k} \sum_{i=1}^{k} y_j^i \tag{2.9}$$

Now, the first extension BRkNN-a outputs an empty set if a label is not present in at least half of the k nearest neighbors. If that happens, then BRkNN outputs the labels with highest confidence. In BRkNN-b, the average size of the label set of k nearest neighbors s is calculated by taking the average of positive labels present is k neighbors. Finally, the model outputs top $[s]$ labels with highest confidence.

2.5 Related Literature Review on Multi-Label Emotion Classification

2.5.1 Sentiment Classification

Sentiment Analysis which is also known as opinion mining is a Natural Language Technique that determines if the given data asserts positive sentiment or negative sentiment. It has been extensively used to study various product reviews and micro-blogs like Twitter [10]. Since it deals with classification from negative to positive, sometimes a neutral class is also introduced. These task of analyzing the sentiment polarity of a work can be divided three approaches. There are lexicon based approaches [11]. They consider the sentiment polarity of words in a document to get the polarity of the text. Then there is machine learning based approaches [12], where one or multiple models are trained on given datasets to build classifiers to determine the polarity of text. And finally there is a hybrid method which combines aspects of both of the previous ways together.
Since tweets used in this work can have multiple emotional states, a sentiment analysis approach will not be enough. For us, it is not enough to know whether a tweet is positive or negative, we also have to know what emotions does it display.

2.5.2 Multi-class Classification

An improvement over sentiment analysis is multi-class classification. In multi-class classification, the text can be classified into one of three or more classes, thus classifying it as part of one concrete emotional class. But in a dynamic world like ours, it can be considered a drawback of multi-class classification. Lin *et al.* [13] proposed an approach to classify based on readers emotions. Liang *et al.* [14] developed a system that will recommend emoticons to readers based on the content of what they are typing. And Bouazizi and Ohtsuki [15] proposed a scalable approach to quantify tweets into different sentiment class. They used 7 different sentiment classes which consist of 3 pairs of different sentiments and 1 neutral class. All of these works assigns only one concrete sentiment to a text. Thus they neglect other sentiments that might be present in it. Also, using

opposite pairs of emotions ignores the possibility that both tweets can be present in an emotion.

2.5.3 Multi-Label Classification

The problem of one text having multiple emotions leads us to multi-label classification. In this process, one text can have more than one classes assigned to it. All of the previous works in multi-label classification can be divided into two groups. They are problem transformation methods & algorithm adaptation methods.

In problem transformation methods, the problem is transferred to a sentiment classification problem or multi-class classification problem. Methods like Binary Relevance (BR) [3] changes the problem of multi-label classification into multiple binary classification problem. And at the end, the result of all the classifiers are combined again to produce multi-label output. On the other hand, Label Powerset (LP) [4] transforms the problem into multi-class classification. Each unique combination of labels are considered as a separate class. The models are then trained and tested on that assumption. Random k-Labelsets (RAkEL) [4] builds a collection of different LP classifiers, where each classifier is trained on different random subset of labels. It is then combined to generate the output of multi-label prediction.

The advantages of problem adaptation methods are that they are easy to implement and understand. But they fail to adjust and understand the interdependencies between models. Classifier Chain (CC) [5] transform the multi-label classification problem into a chain of binary classification problem. Here, the length of chain is equal to the number of labels. Each classifier in the chain solves a binary classification problem. But their knowledge is augmented by previous models which was trained on different emotions.

Algorithm adaptation methods converts pre-existing classification algorithms to handle multi-label data. One of the prominent example of this is multi-label k-nearest neighbor algorithm (MLkNN) [8]. It modifies the k-nearest neighbor algorithm to handle multi-label data. It uses maximum a posteriori rule to make multi-label prediction. Other examples of this includes Multi-label decision tree,

Rank SVM, Predictive clustering trees. Cabrera-Diego *et al.* [16] used issue trackers comments on Stack Overflow and JIRA to perform multi-label classification of the texts. They used two multi-label classifiers to train and test their models. Liu and Chen [17] used 11 multi-label classification based approach to train and predict 2 microblog datasets collected from different incidents. They used 3 different sentiment dictionaries. They text segmentation for preprocessing. The datasets used in their work were taken from 2 related incident. So the corelation between tweets were higher.

2.6 Conclusion

In this chapter, a detailed review of multi-label emotion classification was discussed. This discussion was divided into sentiment analysis, multi-class classification & finally, into multi-label classification. This chapter shows how changing requirements in problems leads researcher to develop more and more advance methods. The next chapter contains the detailed methodology of multi-label emotion classification of tweets.

Chapter 3

Methodology

3.1 Introduction

Classifying emotions from tweets is a hard task even for humans. But when there are multiple emotions present in tweets the task become even complex. For this reason, the tweets need to be processed before using them to train our models. We should reduce the noise & emphasize more on the features that relates to the emotions. In this chapters, the processing of tweets and implementation of multi-label classification models are discussed.

3.2 Diagram/Overview of Framework

To collect and label tweets in real time, we will use Twitter API. The collected tweets will be sent to our multi-label emotion classifiers to classify it as various emotions. But before that, we need to train and develop our emotion classifier models. The first part is to collect and label tweets which will be used to train the models. After the collection we will be developing models that can perform multi-label emotion classification. The tweets that will be used to train classifiers was selected from *Sentiment140* [18] dataset. The tweets will then be labelled by hand and processed before being used to train the classifiers. The labeling power of the classifiers will be augmented using *Senticnet5* [2].

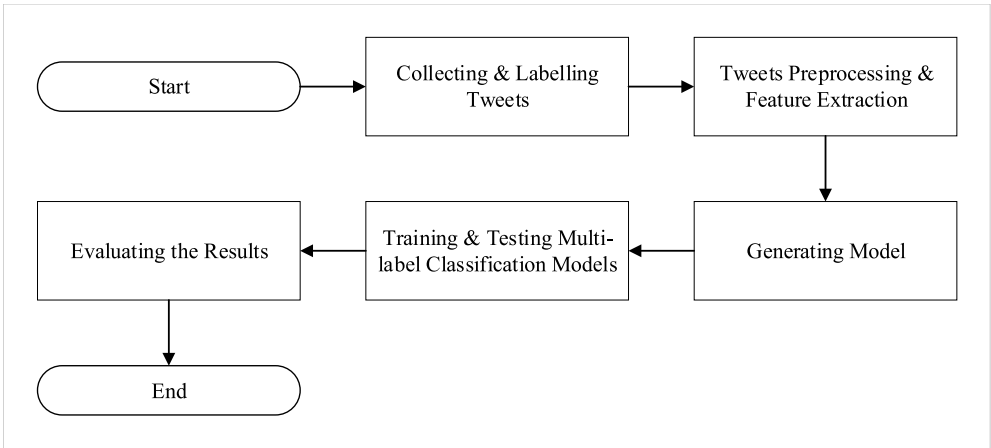

Figure 3.1: Outline of the Methodology

3.3 Detailed Explanation

3.3.1 Tweet Preprocessing

Before training our models, we needed to process our text data to be able to pass it through the training models. The purpose of this preprocessing is to keep features that are related to their labels. We also needed to discard or trim features that complicates the training phase. We started the preprocessing by removing links and pictures from the tweets. The sentence was then separated into multiple sentences. Each sentence was evaluated separately. The hashtags present in the tweet were processed by removing the hash symbol and reading the word to the sentence. If the tweet had any punctuation in it, the flags for question mark and exclamation points were turned on. We detected words that has negation (Ex. not, n't) before them. We removed stopwords from the sentence. We then tokenized the words and performed lemmatization. This will group together the infected forms of words so that we can process them as single item instead of differentiating between them. Finally, we implemented Term Frequency - Inverse Document Frequency (tf-idf) over analyzed tweet dataset. This transform raw texts in the dataset to matrix representation of tf-idf words.

The detailed explanation of the tweet processing steps are given below:

- **Raw tweet:** A tweet is a post on Twitter. It is a micro-blog that can consist up to 280 letters. It can contain mentions to other users, hashtags and hyperlinks. Example of a raw tweet: "@alamin I did not have a good

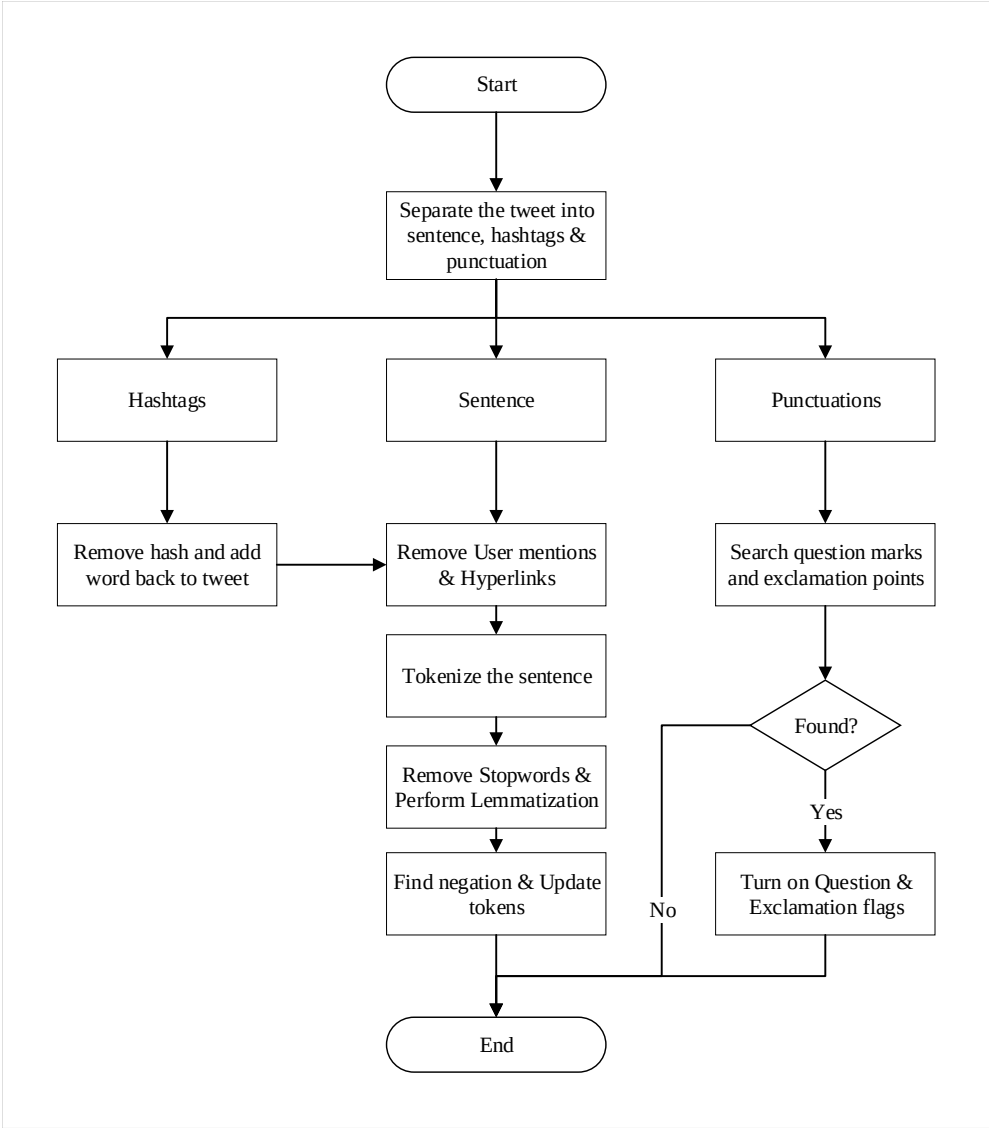

Figure 3.2: Tweet Preprocessing

day but @refat had a blast"

- **Removing User mentions:** An user mention on twiiter starts with '@' symbol. It notifies the other user that he has been mentioned in a conversation but it posses little to no importance in determining the emotion of tweets. So, we remove any user mentioned in the tweet.
- **Removing hyperlinks:** Hyperlinks or simply links refer to data not present in current location. It can take user to other parts of Twitter or even to other websites. They start with 'http' or 'www'. Their structure can be defined with the regular expression

  `https?://[A-Za-z0-9./]+`

Using this expression to detect hyperlinks in tweets will allow us to remove the hyperlinks from the tweets.

- **Processing Hashtags:** Hashtags are words or phrases preceded by a hash (#) symbol. Twitter uses them to identify content of specific topics. As they are words with hash sigh in front of it, removing the sign will will yield us the word.

- **Tokenization:** Tokenization is the process of splitting the tweet into smaller unit of words. It will help identify words effected by negation, parts of speech & words that are not related to emotions. Python's Natural Language Toolkit package will be used for this purpose.

- **Remove Stopwords:** Stopwords are words that do not add meaning to the sentence. They can be safely removed without alternating the meaning of the sentence. We remove stopwords from our tweets to reduce unnecessary clutters.

- **Lemmatization:** Lemmatization groups different instances of a word. It is done so that different appearances of a word can be processed together. It returns words back to it's root form. For example the word 'studying' becomes 'study' and the word 'studied' also becomes 'study'. Thus two separate instance of the word 'study' can be grouped together thus linking two separate sentences. Python's Natural Language Toolkit package will be used for this purpose.

- **Negation:** Negation refers to the opposite of something. Handling negation is important is emotion classification because without it the machine can train with contradictory results. For example the word 'happy' refers to the emotion of 'Joy' and 'not happy' refers to 'Sadness'. But if the machine does not understand the negation present in the sentence it will consider 'happy' as both 'Joy' and 'Sadness'. While processing the tweets, a list will keep track of all the words preceded by negation. And this list will be added to the training set during the training phase.

- **POS Tagging:** Parts of speech shows how a word is used in a sentence. But

not all the parts of speech in a sentence contribute to the emotions. Thus, removing words that do not show any emotions will reduce unnecessary noise from the data. We used Natural Language Toolkit's parts of speech tagger with detect parts of speech from the tweets and then we filters Nouns, Adjectives, Verbs and Adverbs from the lists.

- **Punctuation:** Punctuations like question marks (?) & exclamation mark (!) can convey emotions in a sentence. A question mark can indicate interest and an exclamation mark can mean surprise. Thus keeping track of these symbols in a sentence will increase the features of a sentence.

Below is a table representation of the process:

Table 3.1: Tweet Processing

Process	**Example**
Raw tweet	@alamin I did not have a good day but @refat had a blast. #Mixed #day Watch it here: https://youtu.be/dqw4w9wgxcq
Remove User mention	i did not have a good day but had a blast. #mixed #day watch it here: https://youtu.be/dqw4w9wgxcq
Remove Hyperlinks	i did not have a good day but had a blast. #mixed #day watch it here:
Process Hashtags	i did not have a good day but had a blast mixed day watch it here
Tokenization & Remove stopwords	['i', 'did', 'not', 'have', 'good', 'day', 'had', 'blast', 'mixed', 'day', 'watch', 'here']
Lemmatization	['i', 'do', 'not', 'have', 'a', 'good', 'day', 'but', 'have', 'a', 'blast', 'mixed', 'day', 'watch', 'it', 'here']
Negation Words	['have', 'a', 'good', 'day']
POS tagging	[('i', 'NN'), ('did', 'VBD'), ('not', 'RB'), ('have', 'VB'), ('a', 'DT'), ('good', 'JJ'), ('day', 'NN'), ('but', 'CC'), ('had', 'VBD'), ('a', 'DT'), ('blast', 'NN'), ('mixed', 'JJ'), ('day', 'NN'), ('watch', 'VB'), ('it', 'PRP'), ('here', 'RB')]
Post POS tagging	['i', 'did', 'not', 'have', 'good', 'day', 'had', 'blast', 'mixed', 'day', 'watch', 'here']
Final Output	i do not have good day have blast mixed day watch here

3.3.2 Senticnet5 & Tf-idf

After the completion of preprocessing steps we move on to ready the dataset for train & test the models. We use K-fold cross validation to validate our model. In this process, the dataset is divided into k sets. k-1 of them are used for training

and 1 one of them is used for evaluation. We took k=10 to perform a k-fold cross validation of the models. Before sending the model for training, we detect the tokens that are present in train set but absent in test dataset. Our reasoning behind this operation is that our model may not understand a word absent in train dataset, hence not being able to produce a valid result. This is where we will use senticnet5 to search for the word in their emotion dictionary. Aside from tracking polarity score and polarity labels of words, it also features primary and secondary mood for the words. We use this feature from the senticnet5 to add the emotions of the word to the training dataset. This process is also done with negation words. But instead of directly adding the word present in the negation set, we append the prefix 'not' before it and take the opposite of the emotions given from Senticnet5. The train dataset and the test dataset is then transformed to matrix form using tf-idf vectorizer. This final matrix form of the datasets are then sent to the models to train and evaluate them.

3.4 Implementation

3.4.1 System Requirements

3.4.1.1 Hardware Requirements

The system was executed on an ASUS K555l laptop with the following specification:

- **CPU:** Intel Core i5-5200U
- **GPU:** NVIDIA GeForce 930M (2GB DDR3)
- **RAM:** 8GB DDR3
- **HDD:** 1TB HDD, 5400 rpm
- **Display:** 15.6", HD (1366 x 768), TN
- **Battery:** 37Wh, 2-cell

3.4.1.2 Software Requirements

The software environment needed to execute the system is given below:

- Microsoft Windows 10 (OS Build 19042.928)
- python => 3.8.5
- anaconda => 2020.11
- pip => 20.2.4
- jupyter => 1.0.0
- numpy => 1.19.2
- pandas => 1.1.3
- matplotlib => 3.3.2
- nltk => 3.5
- scikit-learn => 0.23.2
- scikit-multilearn => 0.2.0

Although the system was run in an windows environment, with the proper packages present, this can be run in other operating systems too. The dependencies of the given packages are also needed to be installed for this to work properly. All the packages aren't needed at once to run the system.

3.4.2 Data Preparation

The primary data used in this system are raw tweets. They were collected randomly from the *Sentiment140* [18] dataset. This dataset contained 1.6 million tweets collected using *Twitter* API. It contained the ids, date, flag, user and text of the tweets. We randomly collected 8500 text from the dataset. The collection was done using python's random package. For this work, raw tweets were considered as inputs. Each of the tweets were processed before using them as input. A sample of raw tweets are shown in figure 3.3.

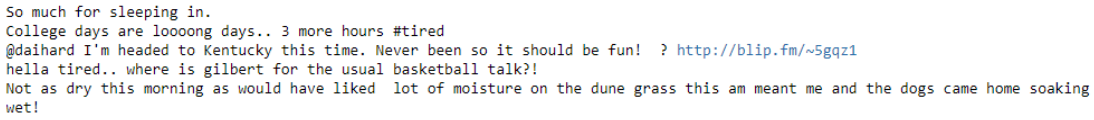

Figure 3.3: A Sample of Raw Tweets

Tweets after removing User mentions were shown in Figure 3.4. As it can be seen from the changes between third tweet from figure 3.3 to figure 3.4.

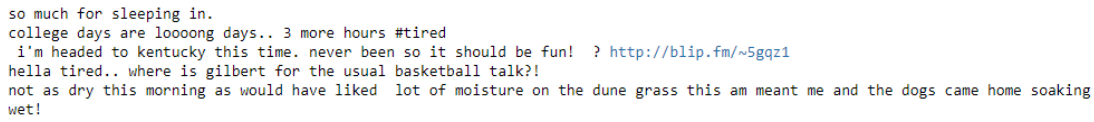

Figure 3.4: Removing User Mentions

Hyperlinks were removed in figure 3.5. The hyperlink to a website present at the end of third tweet can no longer be seen in figure 3.5.

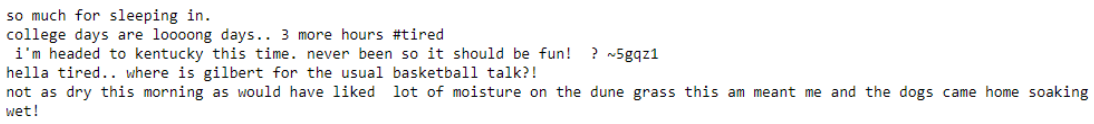

Figure 3.5: Removing Hyperlinks

Hashtags present in the tweets are processed by removing '#' from them and appending them back to the tweets(Fig. 3.6). For example, the '#tired' hashtag present in the second tweet was processed and then appended back to the tweet.

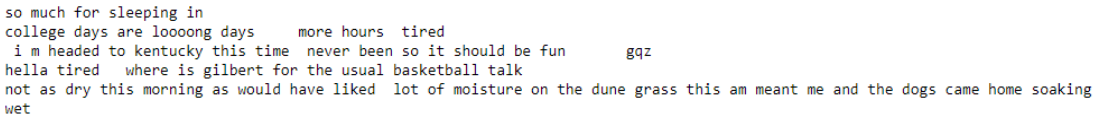

Figure 3.6: Hashtags Processing

Tweets were then tokenize and stopwords were removed from them (Fig. 3.7). In the first tweet, 'so much for sleeping in', stopwords 'for' and 'in' were removed and the words present in the tweets were converted to tokens.

```
['so', 'much', 'sleeping']
['college', 'days', 'are', 'loooong', 'days', 'more', 'hours', 'tired']
['i', 'm', 'headed', 'kentucky', 'time', 'never', 'been', 'be', 'fun', 'gqz']
['hella', 'tired', 'is', 'gilbert', 'usual', 'basketball', 'talk']
['not', 'dry', 'morning', 'as', 'have', 'liked', 'lot', 'moisture', 'dune', 'grass', 'am', 'meant', 'dogs', 'came', 'home', 'so
aking', 'wet']
```

Figure 3.7: Tokenization and Removal of Stopwords

Lemmatization of the tweets were then performed (Fig. 3.8). The token 'sleeping' present in the first tweet was transformed to sleep. This process was applied to the whole dataset.

```
['so', 'much', 'for', 'sleep', 'in']
['college', 'day', 'be', 'loooong', 'day', 'more', 'hour', 'tire']
['i', 'm', 'head', 'to', 'kentucky', 'this', 'time', 'never', 'be', 'so', 'it', 'should', 'be', 'fun', 'gqz']
['hella', 'tire', 'where', 'be', 'gilbert', 'for', 'the', 'usual', 'basketball', 'talk']
['not', 'a', 'dry', 'this', 'morning', 'a', 'would', 'have', 'like', 'lot', 'of', 'moisture', 'on', 'the', 'dune', 'grass', 'th
is', 'be', 'meant', 'me', 'and', 'the', 'dog', 'come', 'home', 'soak', 'wet']
```

Figure 3.8: Lemmatization of Tweets

Another copy of the tokens were sent to parts of speech tagger. The results are shown in figure 3.9. This was done to filter through the data. This information was used to select lemmas with proper parts of speech.

```
[('so', 'RB'), ('much', 'JJ'), ('for', 'IN'), ('sleeping', 'VBG'), ('in', 'IN')]
[('college', 'NN'), ('days', 'NNS'), ('are', 'VBP'), ('loooong', 'JJ'), ('days', 'NNS'), ('more', 'RBR'), ('hours', 'NNS'), ('t
ired', 'VBD')]
[('i', 'JJ'), ('m', 'NN'), ('headed', 'VBD'), ('to', 'TO'), ('kentucky', 'VB'), ('this', 'DT'), ('time', 'NN'), ('never', 'R
B'), ('been', 'VBN'), ('so', 'IN'), ('it', 'PRP'), ('should', 'MD'), ('be', 'VB'), ('fun', 'JJ'), ('gqz', 'NN')]
[('hella', 'NN'), ('tired', 'VBN'), ('where', 'WRB'), ('is', 'VBZ'), ('gilbert', 'JJ'), ('for', 'IN'), ('the', 'DT'), ('usual',
'JJ'), ('basketball', 'NN'), ('talk', 'NN')]
[('not', 'RB'), ('as', 'IN'), ('dry', 'JJ'), ('this', 'DT'), ('morning', 'NN'), ('as', 'RB'), ('would', 'MD'), ('have', 'VB'),
('liked', 'VBN'), ('lot', 'NN'), ('of', 'IN'), ('moisture', 'NN'), ('on', 'IN'), ('the', 'DT'), ('dune', 'NN'), ('grass', 'N
N'), ('this', 'DT'), ('am', 'VBP'), ('meant', 'VB'), ('me', 'PRP'), ('and', 'CC'), ('the', 'DT'), ('dogs', 'NNS'), ('came', 'VB
D'), ('home', 'RB'), ('soaking', 'VBG'), ('wet', 'NN')]
```

Figure 3.9: POS Tagging of Tweets

The tokens with the proper parts of speech were then filtered and are shown in figure 3.10.

```
['so', 'much', 'sleeping']
['college', 'days', 'are', 'loooong', 'days', 'more', 'hours', 'tired']
['i', 'm', 'headed', 'kentucky', 'time', 'never', 'been', 'be', 'fun', 'gqz']
['hella', 'tired', 'is', 'gilbert', 'usual', 'basketball', 'talk']
['not', 'dry', 'morning', 'as', 'have', 'liked', 'lot', 'moisture', 'dune', 'grass', 'am', 'meant', 'dogs', 'came', 'home', 'so
aking', 'wet']
```

Figure 3.10: Tokens Selected after POS Tagging

Final state of the tweets were shown in figure 3.11. This is the data that will be sent to tf-idf vectorizer to be converted into matrix representation.

```
so much sleep
college day be looooong day more hour tire
i m head kentucky time never be be fun gqz
hella tire be gilbert usual basketball talk
not dry morning a have like lot moisture dune grass be meant dog come home soak wet
```

Figure 3.11: Tweets after Preprocessing

Before training the models, the tweets were vectorized using Tf-Idf vectorizer. The sparse matrix form of the tweets are shown in figure 3.12.

	0	1	2	3	4	5	6	7	8	9	...	\
tapatio	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
hurray	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
invoked	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
nyehhh	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
married	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
matt	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
pleaseeeee	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
donnie	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
lift	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
sazza	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	

	8460	8461	8462	8463	8464	8465	8466	8467	8468	8469
tapatio	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
hurray	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
invoked	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
nyehhh	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
married	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
matt	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
pleaseeeee	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
donnie	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
lift	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
sazza	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 3.12: Training Dataset in Sparse Matrix Form

3.4.3 Training Multi-Label Emotion Classifiers

There are two approaches to solve multi-label classification problems. They are problem transformation methods and algorithm adaption methods. The problem transformation methods in this work are binary relevance, label powerset, classifier chains, random k-labelset & ensemble of classifier chains. Machine learning

adaption methods used are MLkNN, BRkNN-a & BRkNN-b. The models were trained with k-fold cross validation with k=10. The dataset was divided into ten parts. During each iteration, nine of the parts are used for training and one for testing purpose.

In binary relevance method (shown in algorithm 1), the train and test datasets are converted to eight sub-datasets. Each with the same input text but output labels are different in relation to each emotion the sub-dataset represents. Each of the sub-datasets were then trained & evaluated using the base classifiers. At the end, the results were combined to produce the multi-label classification. In label powerset method (shown in algorithm 2), each unique combination of labels were mapped to a single integer. We consider the integer as the class number of that tweet. The models were then trained and evaluated with base classifiers that supports multi-class classification. The classes derived from the test dataset were then converted back to the labels. The evaluation were then performed. In classifier chains method (shown in algorithm 3), we randomly selected the order of the chain. We used that order in the classifier chain method present scikit learn package of python. Ensemble of classifier chains used the same method. Ten random order of the classifier chain were chosen are trained parallelly. After the training phase, the results were combined with majority voting method. Random k-labelset method is implemented with the help of scikit-multilearn package. The implementation method is shown in algorithm 4. K=4 was chosen as the size of the subsets. Label powerset method was the first to complete training & evaluation. Ensemble of classifier chains took the longest to train.

Three machine learning adaption methods are used in this work.They are implemented using scikit-multilearn package. They have different methods for each of the three classifier used. Since all the methods use a variation of k nearest neighbor in one form or another, a grid search was performed to find the optimal value of k. The result of the evaluation is discussed in detail in chapter 4.

3.5 Conclusion

In this chapter, we detailed the methodology to perform multi-label emotion classification on tweets using machine learning. We described ways to process the tweets. We also explained the use of senticnet5 to boost the classification of emotions. We described the two ways to approach the problem of multi-label classification and described various algorithms to either convert it to other problems with known solutions or adapted well known machine learning algorithms to work with multi-label classification. In the next chapter, we present the experimental results of our findings.

Chapter 4

Results and Discussions

4.1 Introduction

In the previous chapter, a detailed explanation of the proposed Framework for the multi-label emotion classification using machine learning was given. In this chapter, the performance of the proposed frameworks are examined.

This framework was implemented using in an interactive python environment using Jupyter Notebook. The computer used has a Core i5 processor and 8 gigabytes of RAM. It is unfortunate that no standard dataset for multi label emotion classification exist. Most of the dataset used in the previous work are related to specific events. Hence, they won't be applicable in an environment where trends are continuously changing.

4.2 Dataset Description

The tweets used to train multi-label classifiers was selected from *Sentiment140* [18] dataset. It contains 1.6 millions tweets extracted using Twitter API. We randomly selected 8500 tweets from this. These tweets were then analyzed and properly labeled according to the emotions present in them. The emotions used are Joy, Sadness, Anger, Disgust, Admiration, Surprise, Interest & Fear. According to Ekman [1], there are six basic emotions. They are anger, disgust, fear, joy, sadness, and surprise. We added the emotions interest and admiration to our dataset because while labeling we found some tweets hard to classify with the six basic emotions.

A sample of the dataset is shown in table 4.1

Table 4.1: Sample of the dataset

Text	Emotion
So much for sleeping in.	Fear
College days are loooong days.. 3 more hours #tired	Sadness, Interest
@daihard I'm headed to Kentucky this time. Never been so it should be fun! ? http://blip.fm/ 5gqz1	Interest
hella tired.. where is gilbert for the usual basketball talk?!	Interest
Not as dry this morning as would have liked lot of moisture on the dune grass this am meant me and the dogs came home soaking wet!	Sadness, Disgust
@lil_laura_loo Really? I think we have some! I've taken Piriteeze but only works for a little while and can only take 1 a day! xo	Sadness, Fear

Multi-label datasets are not like other datasets. Hence, we need new parameters to properly describe them. They are Label Cardinality (LC) and Label Density (LD). Let, Y_i to be the number of samples for i$^{\text{th}}$ tweet in the dataset, N be the number of tweets present in the dataset and L be the number of emotions present in the dataset. We can describe LC & LD using,

$$LC = \frac{1}{N}\sum_{i=1} N|Y_i| \tag{4.1}$$

$$LD = \frac{1}{N}\sum_{i=1} N\frac{|Y_i|}{|L|} \tag{4.2}$$

Various properties of the dataset are described in table 4.2. We also generated

Table 4.2: Dataset Properties

Property	Value
Total number of tweets	8500
Label Cardinality	1.5
Label Density	0.1875
No. of tweets with 'Joy' label	2406
No. of tweets with 'Sadness' label	4126
No. of tweets with 'Anger' label	989
No. of tweets with 'Disgust' label	1380
No. of tweets with 'Admiration' label	578
No. of tweets with 'Surprise' label	624
No. of tweets with 'Interest' label	2134
No. of tweets with 'Fear' label	674
No. of tweets with zero labels	323
No. of tweets with one label	4247
No. of tweets with two labels	3142
No. of tweets with three or more labels	789

association rules from the dataset. Interesting associations and relationships were discovered during this process. Although, it is mostly used in market transactions, we used it here to show the co-relations between the emotion labels. Python

'Apyori' package which has a method for implementing apriori algorithm was used to generate the rules. Some of the generated rules are shown in table 4.3

Table 4.3: Association Rules Generated from the Dataset

Association Rules	**Support**	**Confidence**	**Lift**
Admiration → Joy	0.089	0.682	1.904
Disgust → Anger	0.087	0.394	1.252
Disgust → Sadness	0.231	0.732	1.277
Joy → Interest	0.161	0.464	1.296
Joy → Surprise	0.046	0.131	1.005
Admiration, Interest → Anger	0.0017	0.005	1.188
Admiration, Joy → Interest	0.011	0.029	1.028
Admiration, Sadness, Interest → Anger	0.0002	1	2.792

4.3 Performance Evaluation

A total of 26 experiments were conducted on our dataset in which we used different multi-label classifiers. Thirteen of them were on raw tweets and thirteen of them were on processed dataset with Senticnet5 added to improve the learning of the models for negation & missing words. Eight different multi-label classifiers were used. Five of them used problem transformation methods & the other three used algorithm adaption method. Problem transformation methods used were binary relevance, label powerset, classifier chains, random k-labelset & ensemble of classifier chains. Algorithm adaption methods used were MLkNN, BRkNN-a & BRkNN-b. For classifiers that used problem transformation methods, a base classifier was needed. For this purpose, both Random Forest Classifier [19] & Support Vector Machine [20] were used. All of the models were trained with K-fold cross validation with k=10. The results shown here are taken as the average of the results achieved from k-fold cross validation. A sample of the output after evaluation is shown in table 4.4. Here, the tweets on the left sides were evaluated and the columns on the right side with emotion labels are the output from the model. Here, '1' indicates the presence of the emotion in the tweet and '0' indicates the absence. In table 4.4, the first tweet is shown to contain the emotions sadness and anger. The presence of words like 'leave' and negation of words like 'fair' & 'give' may lead to this conclusion.

Table 4.4: Sample of the output

Text	Joy	Sadness	Anger	Disgust	Admiration	Surprise	Interest	Fear
@ncremins thats not really fair. I mean you just up and leave like that and you dont even give us a party So when you coming back or ...	0	1	1	0	0	0	0	0
@namralkeeg yep. been using Ping.fm for a long time now. yeah, it kinda works only in one direction. just hate so many fake marketers	0	0	0	1	1	0	0	0
just read that bed bugs are at all time high in hotels in the U.S. Looking forward to sleeping in my hotel tonight	0	0	0	0	0	0	1	1
@Kayker it sounds as though we all have tetchy bubbas at the mo x	1	0	0	0	0	0	0	0
@diiiannee why cant youuu	0	0	1	1	0	0	0	0
can't sleep. Too many things going on in my head. http://plurk.com/p/114wqh	0	1	0	1	0	0	0	0

We based our evaluation on 5 criteria. They are Hamming Loss (HL), Subset 0/1 Loss, Macro F1, Micro F1 and average accuracy.

- **Hamming Loss** (HL) is used to calculate the difference between test cases and predictions. It is the fraction of labels that incorrectly predicted for a sample. Its value range is between 0 and 1. The smaller value of hamming loss indicates better performance.

$$HL = \frac{1}{|N||L|} \sum_{i=1}^{|N|} \sum_{j=1}^{|L|} T_{ij} \oplus P_{ij} \tag{4.3}$$

- **Subset 0/1 Loss** counts predictions with at least one incorrect label as wrong. Thus, it can be classified as absolute accuracy of the model. Bigger value of subset 0/1 loss means better prediction capability.

$$Subset\ 0/1\ Loss = \frac{1}{|N|} \sum_{i=1}^{|N|} T_i \oplus P_i \tag{4.4}$$

- **Macro F-score** evaluates the prediction accuracy of labels. It takes into account the proportion of each label class in the data set. It informs how well the classifier performs over the dataset.

$$Macro\ F - Score = \frac{2}{|L|} \sum_{j=1}^{|L|} \frac{\sum_{i=1}^{|N|} T_{ij} \times P_{ij}}{\sum_{i=1}^{|N|} T_{ij} + P_{ij}} \tag{4.5}$$

- **Micro F-Score** calculates average label and instances prediction accuracy. If there is label imbalance, micro f-score is more preferable than macro

f-score.

$$Micro\ F-Score = \frac{2}{|L|} \frac{\sum_{j=1}^{|L|} \sum_{i=1}^{|N|} T_{ij} \times P_{ij}}{\sum_{j=1}^{|L|} \sum_{i=1}^{|N|} T_{ij} + P_{ij}} \tag{4.6}$$

- **Average Accuracy** evaluates average prediction score for each labels.

$$Average\ Accuracy = \frac{1}{|L|} \sum_{j=1}^{|L|} \frac{|N| - \sum_{i=1}^{|N|} T_{ij} \oplus P_{ij}}{|N|} \tag{4.7}$$

The evaluation of the models on the dataset without any preprocessing is shown at table 4.5 As we can see from table 4.5 that, without any preprocessing Binary

Table 4.5: Evaluation of the models using raw data

Evaluation Metric	BR		LP		CC		RAkEL		ECC		MLkNN	BRkNN	
	RFC	SVM	RFC	SVM	RFC	SVM	RFC	SVM	RFC	SVM		a	b
Hamming Loss	0.164	0.162	0.172	0.18	0.163	0.175	0.168	0.167	0.165	0.174	0.166	0.172	0.284
0/1 Subset Loss	0.192	0.197	0.272	0.253	0.23	0.261	0.24	0.228	0.227	0.247	0.207	0.183	0.063
Macro F-Score	0.892	0.893	0.865	0.84	0.888	0.866	0.878	0.875	0.885	0.86	0.888	0.883	0.790
Micro F-Score	0.905	0.906	0.897	0.893	0.904	0.895	0.9	0.901	0.903	0.896	0.902	0.901	0.822
Average Accuracy	0.835	0.837	0.827	0.819	0.836	0.824	0.831	0.832	0.834	0.825	0.833	0.827	0.715

Relevance with SVM as base classifier has the lowest hamming loss. But Label Powerset with Random Forest Classifier has the highest subset 0/1 loss rate. Evaluation of models trained with data preprocessing and Senticnet5 boosting in shown in table 4.6.

Table 4.6: Evaluation of models trained with preprocessed data

Evaluation Metric	BR		LP		CC		RAkEL		ECC		MLkNN	BRkNN	
	RFC	SVM	RFC	SVM	RFC	SVM	RFC	SVM	RFC	SVM		a	b
Hamming Loss	0.16	0.161	0.169	0.172	0.162	0.17	0.165	0.167	0.162	0.171	0.169	0.17	0.255
0/1 Subset Loss	0.223	0.213	0.277	0.278	0.263	0.281	0.249	0.262	0.26	0.265	0.202	0.169	0.094
Macro F-Score	0.895	0.894	0.877	0.867	0.888	0.875	0.884	0.875	0.885	0.869	0.887	0.891	0.821
Micro F-Score	0.907	0.906	0.899	0.897	0.904	0.898	0.903	0.901	0.904	0.897	0.901	0.902	0.843
Average Accuracy	0.839	0.838	0.83	0.827	0.837	0.829	0.835	0.832	0.837	0.828	0.83	0.829	0.744

In table 4.6, binary relevance with random forest classifier shows the lowest hamming loss. Classifier chains with SVC as base classifier shows the highest 0/1 subset loss.The Comparison between table 4.5 and table 4.5 shows noticeable improvements over all of the evaluating criteria. A comparison of hamming loss and subset 0/1 loss over preprocessed and raw data are shown in figure 4.1 & figure 4.2 respectively.

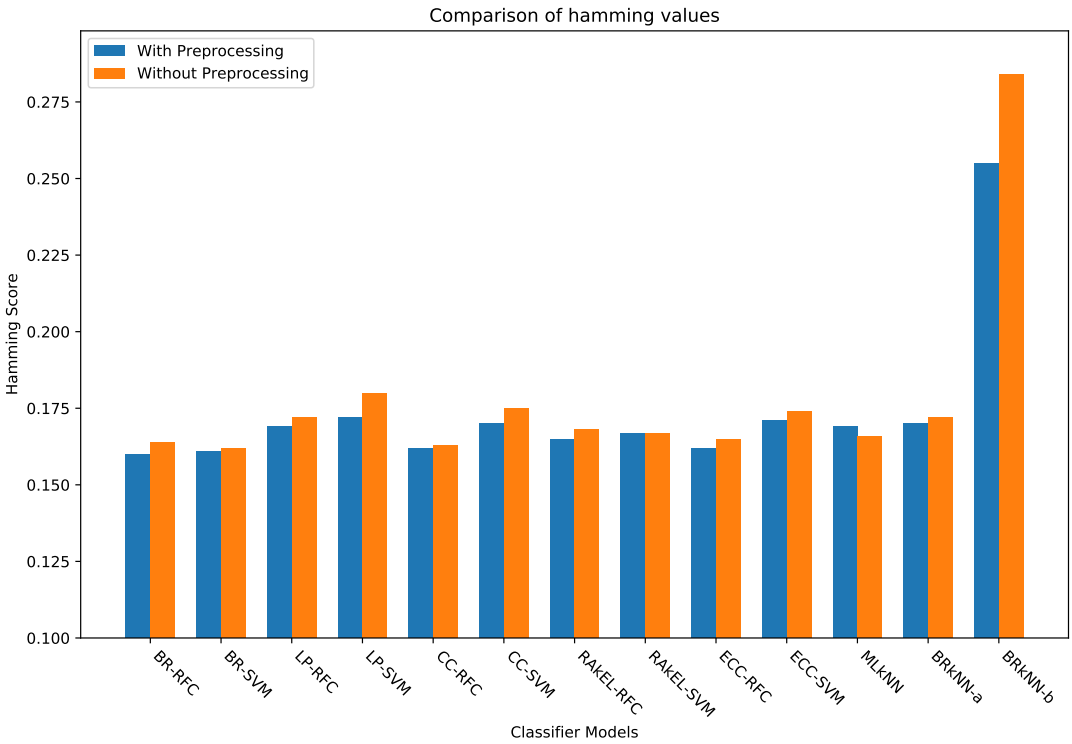

Figure 4.1: Comparison of Hamming Loss over Datasets

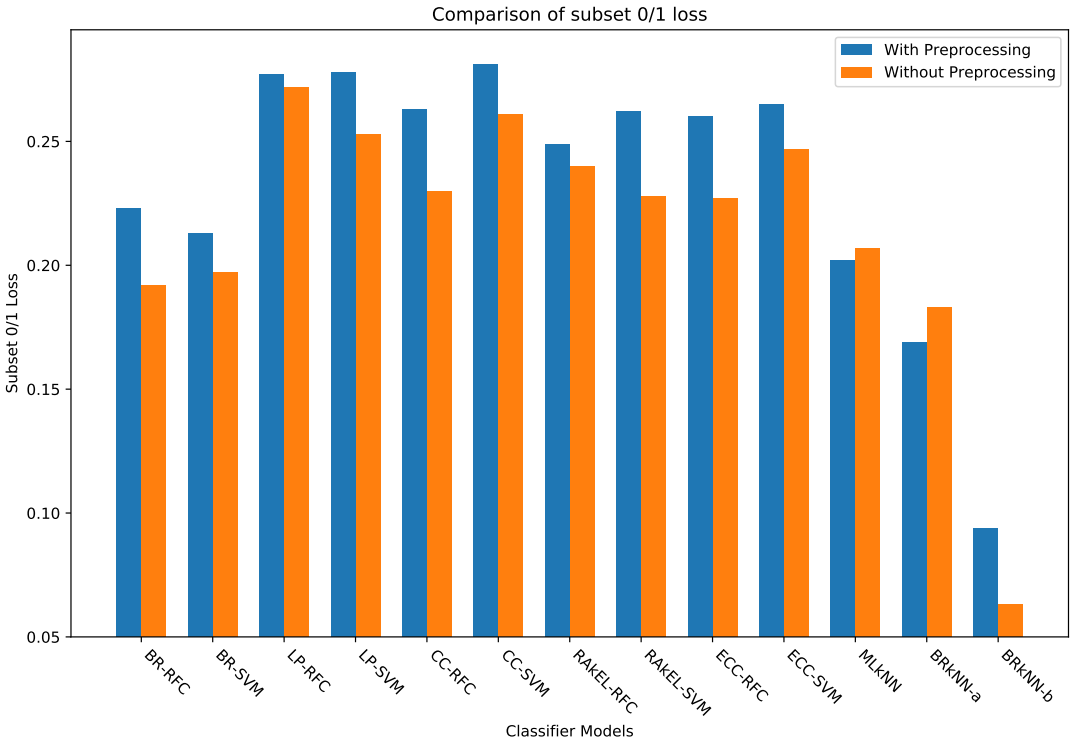

Figure 4.2: Comparison of Subset 0/1 Loss over Datasets

As shown from the figures, preprocessing the tweets & using Senticnet5 for handling unknown words and negation during training phase, leads to better hamming loss in 12 out of 13 models. The models show an average improvement of 2.25%. Same observations can be made for subset 0/1 loss. Here, the results of 10 models out of 12 improved by using preprocessing and senticnet5. The average improvement in subset 0/1 loss is 10.1%. But the unsatisfactory results in MLkNN and BRkNN shows that our Framework is finding it hard to adapt to algorithm adaption methods.

4.4 Conclusion

In this chapter, The results of classification for multi-label emotion classification of tweets using machine learning is shown. As shown by the results, out of the 12 models, the proposed framework leads to better result in 11 models. In the next chapter, the conclusion is drawn of this work.

Chapter 5

Conclusion

5.1 Conclusion

A multi-label emotion classification approach to classify tweets is proposed in this thesis. This prototype can be divided into two parts. The parts are processing of the tweet dataset and training & evaluation of multi-label classifiers. *Senticnet5* was used as booster to improve the classifiers during training phase. Various multi-label classifiers are applied. Among them, Binary Relevance (BR) with Random Forest Classifier as base classifier has shown the best result. An average hamming loss of 0.16 is pretty impressive considering that the dataset was chosen at random. It means the corelation between tweets are pretty low. Thus, even with this disadvantage, our classifiers are able to find correct emotion labels 85% of the times. In classifiers that use problem transformation method, we used both Random Forest Classifier & Support Vector Classifier as base classifiers. In all of the cases, Random Forest Classifier outperformed Support Vector Classifier.

5.2 Future Work

In future, this work can be extended in three direction. Enriching the dataset with more labelled tweets is the obvious way to go. Other ventures include lessening the dependency on *Senticnet5* to boost the classifiers strength. We can also add image and emoji analysis to along with text analysis to get a better understanding of tweet's emotions. The correlations between tweets can also be calculated using association rules generation algorithms like apriori to get dependency between various emotions. Since the dataset is labeled by one user, bias is obviously present in it. Adding more raters to label the dataset and

cross examining them will increase the validity of the dataset. We believe that combining this research with other new technologies like neural networks will vastly improve its performance and will contribute to the development of various open source emotion classifier softwares.

References

[1] P. Ekman, 'An argument for basic emotions,' *Cognition & emotion*, vol. 6, no. 3-4, pp. 169–200, 1992 (cit. on pp. 8, 32).

[2] E. Cambria, S. Poria, D. Hazarika and K. Kwok, 'Senticnet 5: Discovering conceptual primitives for sentiment analysis by means of context embeddings,' in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018 (cit. on pp. 8, 20).

[3] M. R. Boutell, J. Luo, X. Shen and C. M. Brown, 'Learning multi-label scene classification,' *Pattern recognition*, vol. 37, no. 9, pp. 1757–1771, 2004 (cit. on pp. 10, 11, 18).

[4] G. Tsoumakas, I. Katakis and I. Vlahavas, 'Random k-labelsets for multilabel classification,' *IEEE transactions on knowledge and data engineering*, vol. 23, no. 7, pp. 1079–1089, 2010 (cit. on pp. 11, 13, 18).

[5] J. Read, B. Pfahringer, G. Holmes and E. Frank, 'Classifier chains for multi-label classification,' *Machine learning*, vol. 85, no. 3, p. 333, 2011 (cit. on pp. 12, 13, 18).

[6] P. Szymański and T. Kajdanowicz, 'A scikit-based python environment for performing multi-label classification,' *arXiv preprint arXiv:1702.01460*, 2017 (cit. on p. 13).

[7] G. Tsoumakas, I. Katakis and I. Vlahavas, 'Random k-labelsets for multilabel classification,' *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 7, pp. 1079–1089, Jul. 2011, ISSN: 1041-4347. DOI: 10.1109/TKDE.2010.164 (cit. on p. 13).

[8] M.-L. Zhang and Z.-H. Zhou, 'Ml-knn: A lazy learning approach to multi-label learning,' *Pattern recognition*, vol. 40, no. 7, pp. 2038–2048, 2007 (cit. on pp. 14, 15, 18).

[9] I. V. Eleftherios Spyromitros Grigorios Tsoumakas, 'An empirical study of lazy multilabel classification algorithms,' in *Proc. 5th Hellenic Conference on Artificial Intelligence (SETN 2008)*, Syros, Greece, 2008 (cit. on p. 15).

[10] H. Tang, S. Tan and X. Cheng, 'A survey on sentiment detection of reviews,' *Expert Systems with Applications*, vol. 36, no. 7, pp. 10 760–10 773, 2009 (cit. on p. 17).

[11] 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 (cit. on p. 17).

[12] B. Pang and L. Lee, *Opinion mining and sentiment analysis. foundations and trends (r) in information retrieval, 2 (1-2), 1-135*, 2008 (cit. on p. 17).

[13] K. H.-Y. Lin, C. Yang and H.-H. Chen, 'What emotions do news articles trigger in their readers?' In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, 2007, pp. 733–734 (cit. on p. 17).

[14] W.-B. Liang, H.-C. Wang, Y.-A. Chu and C.-H. Wu, 'Emoticon recommendation in microblog using affective trajectory model,' in *Signal and Information Processing Association Annual Summit and Conference (APSIPA), 2014 Asia-Pacific*, IEEE, 2014, pp. 1–5 (cit. on p. 17).

[15] M. Bouazizi and T. Ohtsuki, 'A pattern-based approach for multi-class sentiment analysis in twitter,' *IEEE Access*, vol. 5, pp. 20 617–20 639, 2017 (cit. on p. 17).

[16] L. A. Cabrera-Diego, N. Bessis and I. Korkontzelos, 'Classifying emotions in stack overflow and jira using a multi-label approach,' *Knowledge-Based Systems*, vol. 195, p. 105 633, 2020 (cit. on p. 19).

[17] S. M. Liu and J.-H. Chen, 'A multi-label classification based approach for sentiment classification,' *Expert Systems with Applications*, vol. 42, no. 3, pp. 1083–1093, 2015 (cit. on p. 19).

[18] A. Go, R. Bhayani and L. Huang, 'Twitter sentiment classification using distant supervision,' *CS224N project report, Stanford*, vol. 1, no. 12, p. 2009, 2009 (cit. on pp. 20, 26, 32).

[19] T. K. Ho, 'Random decision forests,' in *Proceedings of 3rd international conference on document analysis and recognition*, IEEE, vol. 1, 1995, pp. 278–282 (cit. on p. 34).

[20] C. Cortes and V. Vapnik, 'Support-vector networks,' *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995 (cit. on p. 34).

Appendix A

Wordcloud Representation of the Dataset

Figure A.1: Wordcloud representation of the dataset

Detailed Training Results with processing

- Binary Relevance (RFC)

	k-fold	Hamming loss	Subset accuracy	Macro F-score	Micro F-score	Average Accuracy
0	1	0.148942	0.242068	0.901340	0.913849	0.851058
1	2	0.162941	0.231765	0.895186	0.905862	0.837059
2	3	0.159706	0.221176	0.896569	0.907227	0.840294
3	4	0.164118	0.209412	0.892092	0.904729	0.835882
4	5	0.163088	0.208235	0.893991	0.905817	0.836912
5	6	0.163382	0.221176	0.893193	0.905519	0.836618
6	7	0.156618	0.231765	0.897360	0.909013	0.843382
7	8	0.158235	0.229412	0.897280	0.908488	0.841765
8	9	0.161765	0.258824	0.893293	0.905677	0.838235
9	10	0.166176	0.183529	0.892102	0.904010	0.833824
10	average	0.160497	0.223736	0.895241	0.907019	0.839503

Figure A.2: Binary Relevance with Random Forest Classifier

- Binary Relevance (SVM)

	k-fold	Hamming loss	Subset accuracy	Macro F-score	Micro F-score	Average Accuracy
0	1	0.154818	0.217391	0.897115	0.910723	0.845182
1	2	0.164706	0.216471	0.892894	0.904972	0.835294
2	3	0.164118	0.204706	0.893081	0.904827	0.835882
3	4	0.161029	0.211765	0.894324	0.906959	0.838971
4	5	0.164559	0.197647	0.891900	0.905161	0.835441
5	6	0.162500	0.208235	0.893800	0.906348	0.837500
6	7	0.158971	0.216471	0.895615	0.907804	0.841029
7	8	0.158529	0.224706	0.897531	0.908799	0.841471
8	9	0.160147	0.237647	0.895905	0.907280	0.839853
9	10	0.161765	0.200000	0.894772	0.906732	0.838235
10	average	0.161114	0.213504	0.894694	0.906961	0.838886

Figure A.3: Binary Relevance with SVM

- Label Powerset (RFC)

	k-fold	Hamming loss	Subset accuracy	Macro F-score	Micro F-score	Average Accuracy
0	1	0.160840	0.286722	0.882620	0.904675	0.839160
1	2	0.171765	0.283529	0.876819	0.898027	0.828235
2	3	0.169853	0.262353	0.879228	0.898800	0.830147
3	4	0.171324	0.275294	0.875840	0.898048	0.828676
4	5	0.172206	0.272941	0.876624	0.897827	0.827794
5	6	0.169559	0.275294	0.876610	0.899222	0.830441
6	7	0.167794	0.264706	0.877595	0.899904	0.832206
7	8	0.171618	0.276471	0.875185	0.898194	0.828382
8	9	0.167206	0.289412	0.878161	0.900272	0.832794
9	10	0.167941	0.283529	0.877322	0.900087	0.832059
10	average	0.169010	0.277025	0.877600	0.899506	0.830990

Figure A.4: Label Powerset with Random Forest Classifier

- Label Powerset (SVM)

	k-fold	Hamming loss	Subset accuracy	Macro F-score	Micro F-score	Average Accuracy
0	1	0.164072	0.286722	0.870573	0.902641	0.835928
1	2	0.179853	0.280000	0.863995	0.893104	0.820147
2	3	0.173529	0.260000	0.868989	0.896528	0.826471
3	4	0.171029	0.290588	0.867787	0.898224	0.828971
4	5	0.171324	0.267059	0.869913	0.898386	0.828676
5	6	0.171765	0.283529	0.867631	0.897884	0.828235
6	7	0.173824	0.249412	0.866845	0.896388	0.826176
7	8	0.170147	0.297647	0.870638	0.899031	0.829853
8	9	0.174559	0.282353	0.866883	0.895941	0.825441
9	10	0.174118	0.289412	0.863343	0.896304	0.825882
10	average	0.172422	0.278672	0.867660	0.897443	0.827578

Figure A.5: Label Powerset with SVM

- Classifier Chains (RFC)

	k-fold	Hamming loss	Subset accuracy	Macro F-score	Micro F-score	Average Accuracy
0	1	0.154524	0.279671	0.892379	0.909310	0.845476
1	2	0.165294	0.271765	0.887412	0.903053	0.834706
2	3	0.164853	0.245882	0.887365	0.902784	0.835147
3	4	0.160588	0.260000	0.889627	0.905634	0.839412
4	5	0.169559	0.240000	0.883792	0.900902	0.830441
5	6	0.163824	0.262353	0.886599	0.903816	0.836176
6	7	0.157353	0.254118	0.892302	0.907487	0.842647
7	8	0.158529	0.262353	0.892006	0.907277	0.841471
8	9	0.165882	0.284706	0.885834	0.902049	0.834118
9	10	0.161029	0.275294	0.889949	0.905252	0.838971
10	average	0.162144	0.263614	0.888726	0.904757	0.837856

Figure A.6: Classifier Chain with Random Forest Classifier

- Classifier Chain (SVM)

	k-fold	Hamming loss	Subset accuracy	Macro F-score	Micro F-score	Average Accuracy
0	1	0.163631	0.283196	0.877292	0.902911	0.836369
1	2	0.175588	0.294118	0.872471	0.895629	0.824412
2	3	0.166618	0.278824	0.880221	0.900658	0.833382
3	4	0.167647	0.296471	0.877837	0.900210	0.832353
4	5	0.176912	0.256471	0.870158	0.895035	0.823088
5	6	0.171324	0.271765	0.874439	0.898173	0.828676
6	7	0.168824	0.264706	0.875728	0.899404	0.831176
7	8	0.168529	0.298824	0.876869	0.900017	0.831471
8	9	0.172794	0.282353	0.873134	0.897011	0.827206
9	10	0.176471	0.281176	0.868259	0.894921	0.823529
10	average	0.170834	0.280790	0.874641	0.898397	0.829166

Figure A.7: Classifier Chain with SVM

- RAkEL (RFC)

	k-fold	Hamming loss	Subset accuracy	Macro F-score	Micro F-score	Average Accuracy
0	1	0.155405	0.287897	0.886021	0.908160	0.844595
1	2	0.172206	0.262353	0.872848	0.898059	0.827794
2	3	0.166324	0.225882	0.887928	0.902424	0.833676
3	4	0.168235	0.252941	0.882639	0.900712	0.831765
4	5	0.170882	0.254118	0.879127	0.899027	0.829118
5	6	0.164265	0.284706	0.877595	0.902726	0.835735
6	7	0.163235	0.201176	0.893752	0.905580	0.836765
7	8	0.157941	0.234118	0.896705	0.908580	0.842059
8	9	0.164118	0.252941	0.890990	0.904272	0.835882
9	10	0.168824	0.243529	0.880583	0.900589	0.831176
10	average	0.165143	0.249966	0.884819	0.903013	0.834857

Figure A.8: RAkEL with Random Forest Classifier

- RAkEL (SVM)

	k-fold	Hamming loss	Subset accuracy	Macro F-score	Micro F-score	Average Accuracy
0	1	0.163925	0.269095	0.880072	0.903108	0.836075
1	2	0.171471	0.258824	0.875453	0.898785	0.828529
2	3	0.173824	0.252941	0.862426	0.896534	0.826176
3	4	0.159853	0.288235	0.880929	0.905041	0.840147
4	5	0.162059	0.217647	0.891207	0.905828	0.837941
5	6	0.167794	0.275294	0.869843	0.900480	0.832206
6	7	0.168529	0.261176	0.872934	0.899527	0.831471
7	8	0.167941	0.297647	0.870094	0.900332	0.832059
8	9	0.172353	0.284706	0.863619	0.897481	0.827647
9	10	0.163824	0.217647	0.891966	0.905191	0.836176
10	average	0.167157	0.262321	0.875854	0.901231	0.832843

Figure A.9: RAkEL with SVM

- MLkNN

	k-fold	Hamming loss	Subset accuracy	Macro F-score	Micro F-score	Average Accuracy
0	1	0.165834	0.200940	0.887951	0.904119	0.834166
1	2	0.174118	0.202353	0.885773	0.899371	0.825882
2	3	0.173382	0.172941	0.887108	0.899566	0.826618
3	4	0.167941	0.217647	0.885963	0.902226	0.832059
4	5	0.173235	0.190588	0.883689	0.899522	0.826765
5	6	0.172794	0.205882	0.885818	0.900060	0.827206
6	7	0.166765	0.190588	0.890545	0.903292	0.833235
7	8	0.165441	0.220000	0.891899	0.904345	0.834559
8	9	0.168235	0.216471	0.890341	0.902622	0.831765
9	10	0.167647	0.204706	0.888038	0.902880	0.832353
10	average	0.169539	0.202212	0.887712	0.901800	0.830461

Figure A.10: MLkNN

- BRkNN-a

	k-fold	Hamming loss	Subset accuracy	Macro F-score	Micro F-score	Average Accuracy
0	1	0.162750	0.170388	0.895541	0.907466	0.837250
1	2	0.171176	0.175294	0.892252	0.902578	0.828824
2	3	0.175000	0.152941	0.887939	0.899713	0.825000
3	4	0.172794	0.164706	0.890129	0.901550	0.827206
4	5	0.169412	0.168235	0.891523	0.903453	0.830588
5	6	0.175441	0.169412	0.888189	0.900192	0.824559
6	7	0.168088	0.158824	0.892204	0.903764	0.831912
7	8	0.167353	0.172941	0.893362	0.904610	0.832647
8	9	0.169412	0.207059	0.891035	0.902785	0.830588
9	10	0.171471	0.156471	0.891592	0.902361	0.828529
10	average	0.170290	0.169627	0.891377	0.902847	0.829710

Figure A.11: BRkNN-a

- BRkNN-b

	k-fold	Hamming loss	Subset accuracy	Macro F-score	Micro F-score	Average Accuracy
0	1	0.247062	0.112808	0.826217	0.849526	0.752938
1	2	0.249265	0.100000	0.827350	0.848050	0.750735
2	3	0.259853	0.098824	0.818182	0.840567	0.740147
3	4	0.265294	0.072941	0.813645	0.837770	0.734706
4	5	0.254559	0.095294	0.821357	0.844739	0.745441
5	6	0.260441	0.082353	0.819544	0.841038	0.739559
6	7	0.254559	0.104706	0.821766	0.844293	0.745441
7	8	0.259853	0.096471	0.818926	0.841283	0.740147
8	9	0.257794	0.084706	0.819419	0.842001	0.742206
9	10	0.248235	0.094118	0.825761	0.848256	0.751765
10	average	0.255692	0.094222	0.821217	0.843752	0.744308

Figure A.12: BRkNN-b

Detailed Training Results without processing

- Binary Relevance (RFC)

	k-fold	Hamming loss	Subset accuracy	Macro F-score	Micro F-score	Average Accuracy
0	1	0.160546	0.197415	0.894289	0.908051	0.839454
1	2	0.168971	0.184706	0.892039	0.903258	0.831029
2	3	0.167794	0.182353	0.891690	0.903510	0.832206
3	4	0.165588	0.196471	0.891863	0.904931	0.834412
4	5	0.163529	0.192941	0.894122	0.906460	0.836471
5	6	0.165147	0.185882	0.892652	0.905479	0.834853
6	7	0.161029	0.187059	0.895956	0.907462	0.838971
7	8	0.162206	0.215294	0.895002	0.906959	0.837794
8	9	0.166176	0.217647	0.891641	0.904221	0.833824
9	10	0.168824	0.161176	0.890521	0.903237	0.831176
10	average	0.164981	0.192094	0.892978	0.905357	0.835019

Figure A.13: Binary Relevance with Random Forest Classifier

- Binary Relevance (SVM)

	k-fold	Hamming loss	Subset accuracy	Macro F-score	Micro F-score	Average Accuracy
0	1	0.157168	0.195065	0.895133	0.909567	0.842832
1	2	0.166471	0.194118	0.892473	0.904489	0.833529
2	3	0.167500	0.182353	0.890802	0.903253	0.832500
3	4	0.161765	0.201176	0.893067	0.906732	0.838235
4	5	0.165000	0.177647	0.891501	0.905188	0.835000
5	6	0.160147	0.214118	0.895468	0.908031	0.839853
6	7	0.160735	0.188235	0.894695	0.907208	0.839265
7	8	0.159706	0.216471	0.895930	0.908168	0.840294
8	9	0.163971	0.217647	0.892012	0.905340	0.836029
9	10	0.166618	0.183529	0.891105	0.904299	0.833382
10	average	0.162908	0.197036	0.893219	0.906228	0.837092

Figure A.14: Binary Relevance with SVM

- Label Powerset (RFC)

	k-fold	Hamming loss	Subset accuracy	Macro F-score	Micro F-score	Average Accuracy
0	1	0.164806	0.289072	0.869467	0.902282	0.835194
1	2	0.175000	0.280000	0.864943	0.896197	0.825000
2	3	0.180000	0.244706	0.859789	0.892632	0.820000
3	4	0.171618	0.285882	0.865415	0.897712	0.828382
4	5	0.173529	0.268235	0.862801	0.897015	0.826471
5	6	0.172059	0.261176	0.866411	0.897656	0.827941
6	7	0.171765	0.250588	0.863762	0.897580	0.828235
7	8	0.172794	0.289412	0.865935	0.897281	0.827206
8	9	0.171176	0.280000	0.865576	0.897877	0.828824
9	10	0.167647	0.280000	0.872885	0.900332	0.832353
10	average	0.172039	0.272907	0.865698	0.897656	0.827961

Figure A.15: Label Powerset with Random Forest Classifier

- Label Powerset (SVM)

	k-fold	Hamming loss	Subset accuracy	Macro F-score	Micro F-score	Average Accuracy
0	1	0.172738	0.265570	0.843811	0.897632	0.827262
1	2	0.185735	0.261176	0.838780	0.889781	0.814265
2	3	0.181471	0.230588	0.842856	0.891887	0.818529
3	4	0.179118	0.252941	0.842765	0.893494	0.820882
4	5	0.181912	0.248235	0.837810	0.892144	0.818088
5	6	0.178235	0.249412	0.841671	0.894222	0.821765
6	7	0.177794	0.236471	0.840174	0.894179	0.822206
7	8	0.178235	0.276471	0.846452	0.894315	0.821765
8	9	0.181618	0.256471	0.839340	0.891847	0.818382
9	10	0.183971	0.256471	0.835327	0.890656	0.816029
10	average	0.180083	0.253381	0.840899	0.893016	0.819917

Figure A.16: Label Powerset with SVM

- Classifier Chains (RFC)

	k-fold	Hamming loss	Subset accuracy	Macro F-score	Micro F-score	Average Accuracy
0	1	0.158049	0.232667	0.890574	0.908348	0.841951
1	2	0.169706	0.237647	0.885050	0.901300	0.830294
2	3	0.166618	0.205882	0.887669	0.902955	0.833382
3	4	0.162500	0.237647	0.888639	0.905305	0.837500
4	5	0.164559	0.231765	0.886207	0.904383	0.835441
5	6	0.162353	0.235294	0.889336	0.905818	0.837647
6	7	0.158235	0.221176	0.892537	0.907892	0.841765
7	8	0.165147	0.248235	0.886219	0.904074	0.834853
8	9	0.161618	0.250588	0.888992	0.905722	0.838382
9	10	0.168529	0.207059	0.884879	0.902068	0.831471
10	average	0.163731	0.230796	0.888010	0.904787	0.836269

Figure A.17: Classifier Chain with Random Forest Classifier

- Classifier Chain (SVM)

	k-fold	Hamming loss	Subset accuracy	Macro F-score	Micro F-score	Average Accuracy
0	1	0.173325	0.257344	0.864641	0.897230	0.826675
1	2	0.178529	0.263529	0.865179	0.893992	0.821471
2	3	0.177206	0.243529	0.866936	0.894252	0.822794
3	4	0.176765	0.261176	0.867705	0.894801	0.823235
4	5	0.181912	0.242353	0.861950	0.891993	0.818088
5	6	0.172353	0.265882	0.869264	0.897606	0.827647
6	7	0.174412	0.245882	0.868205	0.896111	0.825588
7	8	0.169118	0.298824	0.873163	0.899633	0.830882
8	9	0.176324	0.264706	0.865742	0.894908	0.823676
9	10	0.176471	0.267059	0.866081	0.894995	0.823529
10	average	0.175641	0.261029	0.866886	0.895552	0.824359

Figure A.18: Classifier Chain with SVM

- RAkEL (RFC)

	k-fold	Hamming loss	Subset accuracy	Macro F-score	Micro F-score	Average Accuracy
0	1	0.163043	0.239718	0.884611	0.905128	0.836957
1	2	0.168676	0.234118	0.885048	0.901621	0.831324
2	3	0.169853	0.203529	0.880915	0.899974	0.830147
3	4	0.167353	0.257647	0.867379	0.900472	0.832647
4	5	0.169706	0.241176	0.878691	0.900517	0.830294
5	6	0.171471	0.250588	0.869171	0.898855	0.828529
6	7	0.170735	0.263529	0.868029	0.898345	0.829265
7	8	0.169265	0.270588	0.880496	0.900665	0.830735
8	9	0.168235	0.214118	0.889467	0.902985	0.831765
9	10	0.168824	0.228235	0.882013	0.900915	0.831176
10	average	0.168716	0.240325	0.878582	0.900948	0.831284

Figure A.19: RAkEL with Random Forest Classifier

- RAkEL (SVM)

	k-fold	Hamming loss	Subset accuracy	Macro F-score	Micro F-score	Average Accuracy
0	1	0.160546	0.229142	0.885543	0.906285	0.839454
1	2	0.170000	0.183529	0.890676	0.902677	0.830000
2	3	0.171176	0.225882	0.862494	0.897823	0.828824
3	4	0.169853	0.262353	0.861230	0.899100	0.830147
4	5	0.175735	0.243529	0.855692	0.895752	0.824265
5	6	0.161029	0.189412	0.897361	0.908191	0.838971
6	7	0.164412	0.232941	0.878753	0.902935	0.835588
7	8	0.171765	0.277647	0.860788	0.898116	0.828235
8	9	0.167353	0.201176	0.891022	0.903788	0.832647
9	10	0.167794	0.240000	0.873989	0.901101	0.832206
10	average	0.167966	0.228561	0.875755	0.901577	0.832034

Figure A.20: RAkEL with SVM

- MLkNN

	k-fold	Hamming loss	Subset accuracy	Macro F-score	Micro F-score	Average Accuracy
0	1	0.164953	0.198590	0.887878	0.904074	0.835047
1	2	0.169853	0.215294	0.886008	0.900850	0.830147
2	3	0.171176	0.201176	0.885935	0.899742	0.828824
3	4	0.164265	0.210588	0.889595	0.904194	0.835735
4	5	0.166912	0.202353	0.888327	0.903083	0.833088
5	6	0.165294	0.202353	0.891722	0.904030	0.834706
6	7	0.165441	0.204706	0.888966	0.903242	0.834559
7	8	0.165882	0.216471	0.889947	0.903491	0.834118
8	9	0.167500	0.223529	0.887789	0.902307	0.832500
9	10	0.165294	0.200000	0.888944	0.903569	0.834706
10	average	0.166657	0.207506	0.888511	0.902858	0.833343

Figure A.21: MLkNN

- BRkNN-a

	k-fold	Hamming loss	Subset accuracy	Macro F-score	Micro F-score	Average Accuracy
0	1	0.169947	0.172738	0.883056	0.902585	0.830053
1	2	0.179559	0.181176	0.878301	0.896814	0.820441
2	3	0.176471	0.163529	0.881118	0.898305	0.823529
3	4	0.171618	0.188235	0.884350	0.901378	0.828382
4	5	0.170294	0.192941	0.884805	0.902146	0.829706
5	6	0.170441	0.182353	0.886045	0.902400	0.829559
6	7	0.169265	0.177647	0.886822	0.902532	0.830735
7	8	0.171029	0.194118	0.886132	0.901898	0.828971
8	9	0.173971	0.204706	0.881849	0.899788	0.826029
9	10	0.169853	0.172941	0.887030	0.902458	0.830147
10	average	0.172245	0.183039	0.883951	0.901030	0.827755

Figure A.22: BRkNN-a

- BRkNN-b

	k-fold	Hamming loss	Subset accuracy	Macro F-score	Micro F-score	Average Accuracy
0	1	0.284224	0.065805	0.788942	0.823336	0.715776
1	2	0.277941	0.068235	0.796274	0.827018	0.722059
2	3	0.288971	0.062353	0.785688	0.819044	0.711029
3	4	0.287941	0.056471	0.787030	0.820367	0.712059
4	5	0.281765	0.062353	0.791513	0.824638	0.718235
5	6	0.289118	0.057647	0.789241	0.820161	0.710882
6	7	0.282794	0.062353	0.791424	0.823691	0.717206
7	8	0.285294	0.060000	0.791943	0.822960	0.714706
8	9	0.280294	0.076471	0.793074	0.824816	0.719706
9	10	0.283676	0.061176	0.789935	0.823011	0.716324
10	average	0.284202	0.063286	0.790507	0.822904	0.715798

Figure A.23: BRkNN-b

Appendix B

Source Codes

Preprocessing

```
import numpy as np
import pandas as pd
from senticnet5 import senticnet
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords,wordnet
from nltk.stem import WordNetLemmatizer
from nltk import pos_tag
import re

from sklearn.model_selection import train_test_split, KFold
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import classification_report,
    ↪ confusion_matrix,accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.multioutput import ClassifierChain
from skmultilearn.ensemble import RakelD

from scipy import sparse
stop_words = set(stopwords.words('english'))
wordnet_lemmatizer = WordNetLemmatizer()

df = pd.read_excel('hand8_k_random.xlsx')
print(len(df))
singleword=[]
for key,val in senticnet.items():
    if(len(key.split('_'))==1):
        singleword.append(key)
print(len(singleword))
word=[]
primary=[]
sec=[]
pola=[]
for x in singleword:
    word.append(x)
    primary.append(senticnet[x][4])
```

```
        sec.append(senticnet[x][5])
        pola.append(senticnet[x][7])
    df_emo=pd.DataFrame(list(zip(word,primary,sec,pola)),columns=["
        ↪ Word","Primary","Secondary","Polarity"])
    df_emo.sample(10)

    def remove_count_user_mentions(tweet):
    tweet_mentions_removed = re.subn(r'@[A-Za-z0-9]+','',tweet)
    tweet = tweet_mentions_removed[0]
    no_user_mentions = tweet_mentions_removed[1]
    return tweet,no_user_mentions
#%%
def remove_count_urls(tweet):
    tweet_url_removed = re.subn('https?://[A-Za-z0-9./]+','',tweet)
    tweet = tweet_url_removed[0]
    no_urls = tweet_url_removed[1]
    return tweet,no_urls
#%%
def remove_count_hashtags(tweet):
    no_hashtags = len({tag.strip("#") for tag in tweet.split() if
        ↪ tag.startswith("#")})
    tweet = re.sub("[^a-zA-Z]", "␣",tweet)
    return tweet,no_hashtags
def get_pos(word):
    tag = pos_tag([word])[0][1][0].upper()
    tag_dict = {"J": wordnet.ADJ,
                "N": wordnet.NOUN,
                "V": wordnet.VERB,
                "R": wordnet.ADV}

    return tag_dict.get(tag, wordnet.NOUN)
    need = ["J","N","V","R"]
    #need = ["V"]
    neg = ["n't","not"]
    punct = [".",",","?",";","!"]
    opposite = {}
    opposite["#joy"] = "#sadness"
    opposite[0] = 1
    opposite["#sadness"] = "#joy"
    opposite[1] = 0
    opposite["#admiration"] = "#anger"
    opposite[4] = 2
    opposite["#anger"] = "#admiration"
    opposite[2] = 4
    opposite["#surprise"] = "#fear"
    opposite[5] = 7
    opposite["#fear"] = "#surprise"
    opposite[7] = 5
```

```
opposite["#interest"] = "#disgust"
opposite[6] = 3
opposite["#disgust"] = "#interest"
opposite[3] = 6

negatives = []
def normal_algo(sen):
    NEGATION_ADVERBS = ["no", "without", "nil","not", "n't", "
        ↪ never", "none", "neith", "nor", "non"]
    NEGATION_VERBS = ["deny", "reject", "refuse", "subside", "
        ↪ retract", "non"]
    CONJUCTION_WORDS = ["for", "and", "nor", "but", "or", "yet"
        ↪ , "so"]
    sen = sen.lower()
    sen,removed_user_cnt = remove_count_user_mentions(sen)
    sen,removed_url_cnt = remove_count_urls(sen)
    sen,removed_hashtag_cnt = remove_count_hashtags(sen)
    #print(sen)
    tokens = word_tokenize(sen)
    lem = [wordnet_lemmatizer.lemmatize(t,get_pos(t)) for t in
        ↪ tokens]
    #print(lem)
    lem_lookup = {}
    for i in range(len(tokens)):
        lem_lookup[tokens[i]]=lem[i]
    mark_neg = {}
    nflag = False
    for t in lem:
        if(t[0] in punct or t in CONJUCTION_WORDS):
            nflag=False
        if(nflag==True):
            mark_neg[t]=1
            negatives.append(t)
        if(t in NEGATION_ADVERBS or t in NEGATION_VERBS):
            nflag=True
    tag1 = pos_tag(tokens)
    #print(tag1)
    tokens.clear()
    for x in tag1:
        #print(x)
        if(x[1][0] in need):
            tokens.append(x[0])
    val = {}
    #print(tokens)
    ret_str = ""
    for t in tokens:
        t=lem_lookup[t]
        ret_str+=t
```

```
            ret_str+="␣"
            """
            if(t in senticnet):
                x = senticnet[t][4]
                #print(t)
                if(t in mark_neg):
                    #print(t)
                    x=opposite[x]
                    #print(t,x)
                if(x in val):
                    val[x]+=1
                else:
                    val[x]=1
            """
        #print(mark_neg)
        return ret_str
    analysed = [normal_algo(txt) for txt in df['Text']]
    negatives = set(negatives)
    qmark = []
    exmark = []
    f=0
    for txt in df['Text']:
        f=1
        for lt in txt:
            if(lt=='?'):
                qmark.append(1)
                f=0
                break
        if(f==1):
            qmark.append(0)
        f=1
        for lt in txt:
            if(lt=='!'):
                exmark.append(1)
                f=0
                break
        if(f==1):
            exmark.append(0)
    print(len(qmark))
    print(len(exmark))

    df['Analysed'] = analysed
    df['qmark'] = qmark
    df['exmark'] = exmark
    df.head(10)
    def get_senticnet(word,em):
        em = '#'+em.lower()
        if(senticnet[word][4]==em or senticnet[word][5]==em):
```

```
            return 1
        else:
            return 0
    col_names = ['Joy','Sadness','Anger','Disgust','Admiration','
        ↪ Surprise','Interest','Fear']
    X = df[['Analysed','qmark','exmark']]
    Y = df[['Joy','Sadness','Anger','Disgust','Admiration','
        ↪ Surprise','Interest','Fear']]
    vectorizer = TfidfVectorizer()
    vectorizer.fit(df['Analysed'])
    print(len(vectorizer.vocabulary_))
    kf = KFold(n_splits = 10)
    kf.get_n_splits(X)
    print(kf)
```

Evaluation

```
def evaluation(score_list,predict_score_list):
    filter_corr = []
    exmatch = 0
    atleast1 = 0
    md1 = 0
    one_f = 0
    more_f = 0
    zero_f = 0
    sm = 0
    sdensity = 0
    hammval = 0
    test_len = len(predict_score_list[0])
    for j in range(test_len):
        cnt=0
        for i in range(8):
            hammval+=(score_list[i][j] ^ int(predict_score_list[i][
                ↪ j]))
            if(score_list[i][j]==1):
                cnt+=1
                sm+=1
        sdensity+=cnt/8
        if(cnt==0):
            zero_f+=1
        if(cnt==1):
            one_f+=1
        if(cnt>1):
            more_f+=1
        for i in range(8):
            mf = True
            if(int(predict_score_list[i][j])!=score_list[i][j]):
                mf=False
```

```
            break
    if(mf==True):
        exmatch+=1
        filter_corr.append(j)
    for i in range(8):
        if(int(predict_score_list[i][j])==score_list[i][j] and
            ↪ score_list[i][j]==1):
            atleast1+=1
            break
    mf = False
    for i in range(8):
        if(int(predict_score_list[i][j])==score_list[i][j] and
            ↪ score_list[i][j]==1):
            if(mf==True):
                md1+=1
                filter_corr.append(j)
                break
            mf=True
#print("Label Cardinality: "+ str(sm/test_len))
#print("Label Density: "+ str(sdensity/test_len))
print("Hamming␣Loss:␣"+str(hammval/(test_len*8)))
hamlos = hammval/(test_len*8)
print("Exact␣Prediction:␣"+str(exmatch/test_len))
sub_accu = exmatch/test_len
#print("At least one label predicted: "+str(atleast1/(
    ↪ test_len-zero_f)))
#print("More than one label predicted: "+str(md1/more_f))
tp_sum = 0
fp_sum = 0
fn_sum = 0
macro_preci = 0
macro_recall = 0
macro_f1 = 0
for i in range(len(score_list)):
    tmp = confusion_matrix(score_list[i],predict_score_list[i])
    tp_sum+=tmp[0][0]
    fp_sum+=tmp[0][1]
    fn_sum+=tmp[1][0]
    macro_preci_tmp=tmp[0][0]/(tmp[0][0]+tmp[0][1])
    macro_recall_tmp=tmp[0][0]/(tmp[0][0]+tmp[1][0])
    macro_f1 += ((2*macro_preci_tmp*macro_recall_tmp)/(
        ↪ macro_preci_tmp+macro_recall_tmp))
    macro_preci+=macro_preci_tmp
    macro_recall+=macro_recall_tmp
    #print(macro_f1)
micro_preci = tp_sum/(tp_sum+fp_sum)
micro_recall = tp_sum/(tp_sum+fn_sum)
```

```
    micro_f1 = (2*micro_preci*micro_recall)/(micro_preci+
        ↪ micro_recall)
    macro_preci/=8
    macro_recall/=8
    macro_f1/=8
    #print(micro_preci,micro_recall,micro_f1)
    #print(macro_preci,macro_recall,macro_f1)
    print("Macro F-Score: "+str(macro_f1))
    print("Micro F-Score: "+str(micro_f1))
    col_names = ['Joy','Sadness','Anger','Disgust','Admiration','
        ↪ Surprise','Interest','Fear']
    tmp = 0
    for i in range(len(score_list)):
        score = accuracy_score(score_list[i],predict_score_list[i])
        #print(col_names[i]+" accuracy: "+str(score))
        tmp += score
    print("Average Accuracy: " + str(tmp/8))
    avg_accu = tmp/8
    return (hamlos,sub_accu,macro_f1,micro_f1,avg_accu)
```

Binary Relevance

```
col_names = ['Joy','Sadness','Anger','Disgust','Admiration','
    ↪ Surprise','Interest','Fear']
hamm_score = []
subset_accu = []
macro_f1 = []
micro_f1 = []
avg_accu = []
cnt = 1
for train_index,test_index in kf.split(X):
    x_train,x_test = X.iloc[train_index],X.iloc[test_index]
    y_train,y_test = Y.iloc[train_index].values.tolist(),Y.iloc[
        ↪ test_index].values.tolist()
    print("k_fold validation: " + str(cnt))
    cnt+=1
    x_train_analysed = x_train['Analysed'].tolist()
    x_train_qmark = x_train['qmark'].tolist()
    x_train_exmark = x_train['exmark'].tolist()
    x_test_analysed = x_test['Analysed'].tolist()
    x_test_qmark = x_test['qmark'].tolist()
    x_test_exmark = x_test['exmark'].tolist()
    pre = {}
    for sen in x_train_analysed:
        tok = word_tokenize(sen)
        for t in tok:
            pre[t]=1
    for sen in x_test_analysed:
```

```
        tok = word_tokenize(sen)
        for t in tok:
            if(t in pre):
                continue
            else:
                if(t in senticnet):
                    x_train_analysed.append(t)
                    x_train_qmark.append(0)
                    x_train_exmark.append(0)
                    tmp_list = []
                    for cl in col_names:
                        tmp_list.append(get_senticnet(t,cl))
                    y_train.append(tmp_list)
    for word in negatives:
        if(word in senticnet):
            x_train_analysed.append("not␣"+word)
            x_train_qmark.append(0)
            x_train_exmark.append(0)
            tmp_list = []
            for cl in col_names:
                tmp_list.append(get_senticnet(word,cl))
            tmp_list2 = []
            for i in range(8):
                tmp_list2.append(0)
            for i in range(8):
                if(tmp_list[i]==1):
                    tmp_list2[opposite[i]] = 1
            y_train.append(tmp_list2)
    x_train_analysed_vec = vectorizer.transform(x_train_analysed)
    x_test_analysed_vec = vectorizer.transform(x_test_analysed)
    tmp = sparse.hstack((x_train_analysed_vec,np.array(
        ↪ x_train_qmark)[:,None]))
    x_train = sparse.hstack((tmp,np.array(x_train_exmark)[:,None]))
    y_train = np.array(y_train)
    y_test = np.array(y_test)
    tmp = sparse.hstack((x_test_analysed_vec,np.array(x_test_qmark)
        ↪ [:,None]))
    x_test = sparse.hstack((tmp,np.array(x_test_exmark)[:,None]))

    print(x_train.shape,x_test.shape)
    print(y_train.shape,y_test.shape)

    y_pred_val = []
    for i in range(8):
        print(i)
        classifier = SVC()
        tmp_y_train = y_train[:,i]
        classifier.fit(x_train,tmp_y_train)
```

```
        y_pred = classifier.predict(x_test)
        y_pred_val.append(y_pred)

    y_test_val = np.array(y_test)

    score_list = y_test_val.T.tolist()
    predict_score_list = y_pred_val

    ret = evaluation(score_list,predict_score_list)
    hamm_score.append(ret[0])
    subset_accu.append(ret[1])
    macro_f1.append(ret[2])
    micro_f1.append(ret[3])
    avg_accu.append(ret[4])
    print('\n')
print('Final Result: ')
print('Average Hamming Loss: '+str(sum(hamm_score)/len(hamm_score)
    ↪ ))
print('Average Subset Accuracy: '+str(sum(subset_accu)/len(
    ↪ subset_accu)))
print('Average Macro F-score: '+str(sum(macro_f1)/len(macro_f1)))
print('Average Micro F-score: '+str(sum(micro_f1)/len(micro_f1)))
print('Average of Average Accuracy: '+str(sum(avg_accu)/len(
    ↪ avg_accu)))
```

Label Powerset

```
def convtodec(x):
    val = 128
    ret = 0
    for y in x:
        if(y):
            ret+=val
        val=val>>1
    return ret
labelpowerset = []
for row in df.iterrows():
    tmp = row[1][1:9].tolist()
    labelpowerset.append(convtodec(tmp))
print(len(labelpowerset))

cntpowerset = {}
for val in labelpowerset:
    if(val in cntpowerset):
        cntpowerset[val]+=1
    else:
        cntpowerset[val]=1
```

```
cntpowersetlist = []
for key,val in cntpowerset.items():
    cntpowersetlist.append((val,key))
cntpowersetlist.sort(reverse=True)
print(len(cntpowersetlist))

label = {}
revlabel = {}
cnt = 0
for val in cntpowersetlist:
    label[val[1]] = cnt
    revlabel[cnt] = val[1]
    cnt+=1
powset = [label[x] for x in labelpowerset]
df['powerset'] = powset

def convert2bin(xx):
    ret = []
    for i in range(8):
        if(xx & (1<<i)):
            ret.append(1)
        else:
            ret.append(0)
    #print(xx)
    ret.reverse()
    #print(ret)
    return ret
col_names = ['Joy','Sadness','Anger','Disgust','Admiration','
    ↪ Surprise','Interest','Fear']
hamm_score = []
subset_accu = []
macro_f1 = []
micro_f1 = []
avg_accu = []
cnt = 1
for train_index,test_index in kf.split(X):
    clf = RandomForestClassifier(n_estimators=300)
    x_train,x_test = X.iloc[train_index],X.iloc[test_index]
    y_train,y_test = Y.iloc[train_index].tolist(),Y.iloc[test_index
        ↪ ].tolist()
    print("k_fold␣validation:␣" + str(cnt))
    cnt+=1
    x_train_analysed = x_train['Analysed'].tolist()
    x_train_qmark = x_train['qmark'].tolist()
    x_train_exmark = x_train['exmark'].tolist()
    x_test_analysed = x_test['Analysed'].tolist()
    x_test_qmark = x_test['qmark'].tolist()
    x_test_exmark = x_test['exmark'].tolist()
```

```
pre = {}
for sen in x_train_analysed:
    tok = word_tokenize(sen)
    for t in tok:
        pre[t]=1
for sen in x_test_analysed:
    tok = word_tokenize(sen)
    for t in tok:
        if(t in pre):
            continue
        else:
            if(t in senticnet):
                x_train_analysed.append(t)
                x_train_qmark.append(0)
                x_train_exmark.append(0)
                tmp_list = []
                for cl in col_names:
                    tmp_list.append(get_senticnet(t,cl))
                y_train.append(label[convtodec(tmp_list)])
for word in negatives:
    if(word in senticnet):
        x_train_analysed.append("not␣"+word)
        x_train_qmark.append(0)
        x_train_exmark.append(0)
        tmp_list = []
        for cl in col_names:
            tmp_list.append(get_senticnet(word,cl))
        tmp_list2 = []
        for i in range(8):
            tmp_list2.append(0)
        for i in range(8):
            if(tmp_list[i]==1):
                tmp_list2[opposite[i]] = 1
        y_train.append(label[convtodec(tmp_list2)])
x_train_analysed_vec = vectorizer.transform(x_train_analysed)
x_test_analysed_vec = vectorizer.transform(x_test_analysed)
tmp = sparse.hstack((x_train_analysed_vec,np.array(
    ↪ x_train_qmark)[:,None]))
x_train = sparse.hstack((tmp,np.array(x_train_exmark)[:,None]))
y_train = np.array(y_train)
y_test = np.array(y_test)
tmp = sparse.hstack((x_test_analysed_vec,np.array(x_test_qmark)
    ↪ [:,None]))
x_test = sparse.hstack((tmp,np.array(x_test_exmark)[:,None]))

print(x_train.shape,x_test.shape)
print(y_train.shape,y_test.shape)
```

```
    clf.fit(x_train,y_train)
    y_pred = clf.predict(x_test)
    score_list = []
    predict_score_list = []
    for i in range(len(y_test)):
        score_list.append(convert2bin(revlabel[y_test[i]]))
        predict_score_list.append(convert2bin(revlabel[y_pred[i]]))
    np_score_list = np.array(score_list)
    transpose = np_score_list.T
    score_list = transpose.tolist()

    np_predict_score_list = np.array(predict_score_list)
    transpose = np_predict_score_list.T
    predict_score_list = transpose.tolist()

    ret = evaluation(score_list,predict_score_list)
    hamm_score.append(ret[0])
    subset_accu.append(ret[1])
    macro_f1.append(ret[2])
    micro_f1.append(ret[3])
    avg_accu.append(ret[4])
    print('\n')
print('Final Result: ')
print('Average Hamming Loss: '+str(sum(hamm_score)/len(hamm_score)
    ↪ ))
print('Average Subset Accuracy: '+str(sum(subset_accu)/len(
    ↪ subset_accu)))
print('Average Macro F-score: '+str(sum(macro_f1)/len(macro_f1)))
print('Average Micro F-score: '+str(sum(micro_f1)/len(micro_f1)))
print('Average of Average Accuracy: '+str(sum(avg_accu)/len(
    ↪ avg_accu)))
```

Classifier Chain

```
    col_names = ['Joy','Sadness','Anger','Disgust','Admiration','
        ↪ Surprise','Interest','Fear']
    hamm_score = []
    subset_accu = []
    macro_f1 = []
    micro_f1 = []
    avg_accu = []
    cnt = 1
    for train_index,test_index in kf.split(X):
        x_train,x_test = X.iloc[train_index],X.iloc[test_index]
        y_train,y_test = Y.iloc[train_index].values.tolist(),Y.iloc
            ↪ [test_index].values.tolist()
        print("k_fold validation: " + str(cnt))
        cnt+=1
```

```
x_train_analysed = x_train['Analysed'].tolist()
x_train_qmark = x_train['qmark'].tolist()
x_train_exmark = x_train['exmark'].tolist()
x_test_analysed = x_test['Analysed'].tolist()
x_test_qmark = x_test['qmark'].tolist()
x_test_exmark = x_test['exmark'].tolist()
pre = {}
for sen in x_train_analysed:
    tok = word_tokenize(sen)
    for t in tok:
        pre[t]=1
for sen in x_test_analysed:
    tok = word_tokenize(sen)
    for t in tok:
        if(t in pre):
            continue
        else:
            if(t in senticnet):
                x_train_analysed.append(t)
                x_train_qmark.append(0)
                x_train_exmark.append(0)
                tmp_list = []
                for cl in col_names:
                    tmp_list.append(get_senticnet(t,cl))
                y_train.append(tmp_list)
for word in negatives:
    if(word in senticnet):
        x_train_analysed.append("not␣"+word)
        x_train_qmark.append(0)
        x_train_exmark.append(0)
        tmp_list = []
        for cl in col_names:
            tmp_list.append(get_senticnet(word,cl))
        tmp_list2 = []
        for i in range(8):
            tmp_list2.append(0)
        for i in range(8):
            if(tmp_list[i]==1):
                tmp_list2[opposite[i]] = 1
        y_train.append(tmp_list2)
x_train_analysed_vec = vectorizer.transform(
    ↪ x_train_analysed)
x_test_analysed_vec = vectorizer.transform(x_test_analysed)
tmp = sparse.hstack((x_train_analysed_vec,np.array(
    ↪ x_train_qmark)[:,None]))
x_train = sparse.hstack((tmp,np.array(x_train_exmark)[:,
    ↪ None]))
y_train = np.array(y_train)
```

```
    y_test = np.array(y_test)
    tmp = sparse.hstack((x_test_analysed_vec,np.array(
        ↪ x_test_qmark)[:,None]))
    x_test = sparse.hstack((tmp,np.array(x_test_exmark)[:,None
        ↪ ]))

    print(x_train.shape,x_test.shape)
    print(y_train.shape,y_test.shape)

    base = RandomForestClassifier()
    chain_rfc = ClassifierChain(base,order = 'random',
        ↪ random_state=0)
    chain_rfc.fit(x_train,y_train)
    y_pred = chain_rfc.predict(x_test)

    y_pred_val = y_pred
    y_test_val = np.array(y_test)

    score_list = y_test_val.T.tolist()
    predict_score_list = y_pred_val.T.tolist()

    ret = evaluation(score_list,predict_score_list)
    hamm_score.append(ret[0])
    subset_accu.append(ret[1])
    macro_f1.append(ret[2])
    micro_f1.append(ret[3])
    avg_accu.append(ret[4])
    print('\n')
print('Final Result: ')
print('Average Hamming Loss: '+str(sum(hamm_score)/len(
    ↪ hamm_score)))
print('Average Subset Accuracy: '+str(sum(subset_accu)/len(
    ↪ subset_accu)))
print('Average Macro F-score: '+str(sum(macro_f1)/len(macro_f1
    ↪ )))
print('Average Micro F-score: '+str(sum(micro_f1)/len(micro_f1
    ↪ )))
print('Average of Average Accuracy: '+str(sum(avg_accu)/len(
    ↪ avg_accu)))
```

RAkEL

```
col_names = ['Joy','Sadness','Anger','Disgust','Admiration','
    ↪ Surprise','Interest','Fear']
hamm_score = []
subset_accu = []
macro_f1 = []
micro_f1 = []
```

```
avg_accu = []
cnt = 1
for train_index,test_index in kf.split(X):
    x_train,x_test = X.iloc[train_index],X.iloc[test_index]
    y_train,y_test = Y.iloc[train_index].values.tolist(),Y.iloc
        ↪ [test_index].values.tolist()
    print("k_fold␣validation:␣" + str(cnt))
    cnt+=1
    x_train_analysed = x_train['Analysed'].tolist()
    x_train_qmark = x_train['qmark'].tolist()
    x_train_exmark = x_train['exmark'].tolist()
    x_test_analysed = x_test['Analysed'].tolist()
    x_test_qmark = x_test['qmark'].tolist()
    x_test_exmark = x_test['exmark'].tolist()
    pre = {}
    for sen in x_train_analysed:
        tok = word_tokenize(sen)
        for t in tok:
            pre[t]=1
    for sen in x_test_analysed:
        tok = word_tokenize(sen)
        for t in tok:
            if(t in pre):
                continue
            else:
                if(t in senticnet):
                    x_train_analysed.append(t)
                    x_train_qmark.append(0)
                    x_train_exmark.append(0)
                    tmp_list = []
                    for cl in col_names:
                        tmp_list.append(get_senticnet(t,cl))
                    y_train.append(tmp_list)
    for word in negatives:
        if(word in senticnet):
            x_train_analysed.append("not␣"+word)
            x_train_qmark.append(0)
            x_train_exmark.append(0)
            tmp_list = []
            for cl in col_names:
                tmp_list.append(get_senticnet(word,cl))
            tmp_list2 = []
            for i in range(8):
                tmp_list2.append(0)
            for i in range(8):
                if(tmp_list[i]==1):
                    tmp_list2[opposite[i]] = 1
            y_train.append(tmp_list2)
```

```
    x_train_analysed_vec = vectorizer.transform(
        ↪ x_train_analysed)
    x_test_analysed_vec = vectorizer.transform(x_test_analysed)
    tmp = sparse.hstack((x_train_analysed_vec,np.array(
        ↪ x_train_qmark)[:,None]))
    x_train = sparse.hstack((tmp,np.array(x_train_exmark)[:,
        ↪ None]))
    y_train = np.array(y_train)
    y_test = np.array(y_test)
    tmp = sparse.hstack((x_test_analysed_vec,np.array(
        ↪ x_test_qmark)[:,None]))
    x_test = sparse.hstack((tmp,np.array(x_test_exmark)[:,None
        ↪ ]))

    print(x_train.shape,x_test.shape)
    print(y_train.shape,y_test.shape)

    classifier = RakelD(base_classifier =
        ↪ RandomForestClassifier(),
                        base_classifier_require_dense=[False,
                            ↪ False],
                        labelset_size=4)
    classifier.fit(x_train,y_train)
    y_pred = classifier.predict(x_test)

    y_pred_val = y_pred.toarray()
    y_test_val = np.array(y_test)

    score_list = y_test_val.T.tolist()
    predict_score_list = y_pred_val.T.tolist()

    ret = evaluation(score_list,predict_score_list)
    hamm_score.append(ret[0])
    subset_accu.append(ret[1])
    macro_f1.append(ret[2])
    micro_f1.append(ret[3])
    avg_accu.append(ret[4])
    print('\n')
print('Final␣Result:␣')
print('Average␣Hamming␣Loss:␣'+str(sum(hamm_score)/len(
    ↪ hamm_score)))
print('Average␣Subset␣Accuracy:␣'+str(sum(subset_accu)/len(
    ↪ subset_accu)))
print('Average␣Macro␣F-score:␣'+str(sum(macro_f1)/len(macro_f1
    ↪ )))
print('Average␣Micro␣F-score:␣'+str(sum(micro_f1)/len(micro_f1
    ↪ )))
```

```
print('Average of Average Accuracy: '+str(sum(avg_accu)/len(
    ↪ avg_accu)))
```

Ensemble of Classifier Chains

```
base = RandomForestClassifier()

chains = [ClassifierChain(base,order = 'random',random_state=i)
    ↪  for i in range(10)]
cnt=1
y_pred_list = []
for chain in chains:
    print("Training Chain "+ str(cnt))
    cnt+=1
    chain.fit(x_train,y_train)
    y_pred = chain.predict(x_test)
    y_pred_list.append(y_pred)
y_pred = []
threshold = 5
for i in range(len(y_pred_list[0])):
    tmp = []
    for j in range(len(y_pred_list[0][0])):
        tmp.append(0)
    y_pred.append(tmp)
for k in range(len(y_pred_list)):
    for i in range(len(y_pred)):
        for j in range(len(y_pred[0])):
            y_pred[i][j]+=y_pred_list[k][i][j]
for i in range(len(y_pred)):
    for j in range(len(y_pred[0])):
        if(y_pred[i][j]>=threshold):
            y_pred[i][j]=1
        else:
            y_pred[i][j]=0
y_pred_np = np.array(y_pred)
score_list = y_test.T.tolist()
predict_score_list = y_pred_np.T.tolist()
evaluation(score_list,predict_score_list)
```

MLkNN

```
col_names = ['Joy','Sadness','Anger','Disgust','Admiration','
    ↪ Surprise','Interest','Fear']
hamm_score = []
subset_accu = []
macro_f1 = []
micro_f1 = []
```

```
avg_accu = []
cnt = 1
for train_index,test_index in kf.split(X):
    x_train,x_test = X.iloc[train_index],X.iloc[test_index]
    y_train,y_test = Y.iloc[train_index].values.tolist(),Y.iloc
        ↪ [test_index].values.tolist()
    print("k_fold validation: " + str(cnt))
    cnt+=1
    x_train_analysed = x_train['Analysed'].tolist()
    x_train_qmark = x_train['qmark'].tolist()
    x_train_exmark = x_train['exmark'].tolist()
    x_test_analysed = x_test['Analysed'].tolist()
    x_test_qmark = x_test['qmark'].tolist()
    x_test_exmark = x_test['exmark'].tolist()
    pre = {}
    for sen in x_train_analysed:
        tok = word_tokenize(sen)
        for t in tok:
            pre[t]=1
    for sen in x_test_analysed:
        tok = word_tokenize(sen)
        for t in tok:
            if(t in pre):
                continue
            else:
                if(t in senticnet):
                    x_train_analysed.append(t)
                    x_train_qmark.append(0)
                    x_train_exmark.append(0)
                    tmp_list = []
                    for cl in col_names:
                        tmp_list.append(get_senticnet(t,cl))
                    y_train.append(tmp_list)
    for word in negatives:
        if(word in senticnet):
            x_train_analysed.append("not "+word)
            x_train_qmark.append(0)
            x_train_exmark.append(0)
            tmp_list = []
            for cl in col_names:
                tmp_list.append(get_senticnet(word,cl))
            tmp_list2 = []
            for i in range(8):
                tmp_list2.append(0)
            for i in range(8):
                if(tmp_list[i]==1):
                    tmp_list2[opposite[i]] = 1
            y_train.append(tmp_list2)
```

```
        x_train_analysed_vec = vectorizer.transform(
            ↪ x_train_analysed)
        x_test_analysed_vec = vectorizer.transform(x_test_analysed)
        tmp = sparse.hstack((x_train_analysed_vec,np.array(
            ↪ x_train_qmark)[:,None]))
        x_train = sparse.hstack((tmp,np.array(x_train_exmark)[:,
            ↪ None]))
        y_train = np.array(y_train)
        y_test = np.array(y_test)
        tmp = sparse.hstack((x_test_analysed_vec,np.array(
            ↪ x_test_qmark)[:,None]))
        x_test = sparse.hstack((tmp,np.array(x_test_exmark)[:,None
            ↪ ]))

        print(x_train.shape,x_test.shape)
        print(y_train.shape,y_test.shape)

        classifier = MLkNN(k=100)
        classifier.fit(x_train,y_train)
        y_pred = classifier.predict(x_test)

        y_pred_val = y_pred.toarray()
        y_test_val = np.array(y_test)

        score_list = y_test_val.T.tolist()
        predict_score_list = y_pred_val.T.tolist()

        ret = evaluation(score_list,predict_score_list)
        hamm_score.append(ret[0])
        subset_accu.append(ret[1])
        macro_f1.append(ret[2])
        micro_f1.append(ret[3])
        avg_accu.append(ret[4])
        print('\n')
    print('Final␣Result:␣')
    print('Average␣Hamming␣Loss:␣'+str(sum(hamm_score)/len(
        ↪ hamm_score)))
    print('Average␣Subset␣Accuracy:␣'+str(sum(subset_accu)/len(
        ↪ subset_accu)))
    print('Average␣Macro␣F-score:␣'+str(sum(macro_f1)/len(macro_f1
        ↪ )))
    print('Average␣Micro␣F-score:␣'+str(sum(micro_f1)/len(micro_f1
        ↪ )))
    print('Average␣of␣Average␣Accuracy:␣'+str(sum(avg_accu)/len(
        ↪ avg_accu)))
```

BRkNN

```
col_names = ['Joy','Sadness','Anger','Disgust','Admiration','
    ↪ Surprise','Interest','Fear']
hamm_score = []
subset_accu = []
macro_f1 = []
micro_f1 = []
avg_accu = []
cnt = 1
for train_index,test_index in kf.split(X):
    x_train,x_test = X.iloc[train_index],X.iloc[test_index]
    y_train,y_test = Y.iloc[train_index].values.tolist(),Y.iloc
        ↪ [test_index].values.tolist()
    print("k_fold validation: " + str(cnt))
    cnt+=1
    x_train_analysed = x_train['Analysed'].tolist()
    x_train_qmark = x_train['qmark'].tolist()
    x_train_exmark = x_train['exmark'].tolist()
    x_test_analysed = x_test['Analysed'].tolist()
    x_test_qmark = x_test['qmark'].tolist()
    x_test_exmark = x_test['exmark'].tolist()
    pre = {}
    for sen in x_train_analysed:
        tok = word_tokenize(sen)
        for t in tok:
            pre[t]=1
    for sen in x_test_analysed:
        tok = word_tokenize(sen)
        for t in tok:
            if(t in pre):
                continue
            else:
                if(t in senticnet):
                    x_train_analysed.append(t)
                    x_train_qmark.append(0)
                    x_train_exmark.append(0)
                    tmp_list = []
                    for cl in col_names:
                        tmp_list.append(get_senticnet(t,cl))
                    y_train.append(tmp_list)
    for word in negatives:
        if(word in senticnet):
            x_train_analysed.append("not "+word)
            x_train_qmark.append(0)
            x_train_exmark.append(0)
            tmp_list = []
            for cl in col_names:
                tmp_list.append(get_senticnet(word,cl))
```

```
            tmp_list2 = []
            for i in range(8):
                tmp_list2.append(0)
            for i in range(8):
                if(tmp_list[i]==1):
                    tmp_list2[opposite[i]] = 1
            y_train.append(tmp_list2)
    x_train_analysed_vec = vectorizer.transform(
        ↪ x_train_analysed)
    x_test_analysed_vec = vectorizer.transform(x_test_analysed)
    tmp = sparse.hstack((x_train_analysed_vec,np.array(
        ↪ x_train_qmark)[:,None]))
    x_train = sparse.hstack((tmp,np.array(x_train_exmark)[:,
        ↪ None]))
    y_train = np.array(y_train)
    y_test = np.array(y_test)
    tmp = sparse.hstack((x_test_analysed_vec,np.array(
        ↪ x_test_qmark)[:,None]))
    x_test = sparse.hstack((tmp,np.array(x_test_exmark)[:,None
        ↪ ]))

    print(x_train.shape,x_test.shape)
    print(y_train.shape,y_test.shape)

    classifier = BRkNNaClassifier(k=100)
    classifier.fit(x_train,y_train)
    y_pred = classifier.predict(x_test)

    y_pred_val = y_pred.toarray()
    y_test_val = np.array(y_test)

    score_list = y_test_val.T.tolist()
    predict_score_list = y_pred_val.T.tolist()

    ret = evaluation(score_list,predict_score_list)
    hamm_score.append(ret[0])
    subset_accu.append(ret[1])
    macro_f1.append(ret[2])
    micro_f1.append(ret[3])
    avg_accu.append(ret[4])
    print('\n')
print('Final␣Result:␣')
print('Average␣Hamming␣Loss:␣'+str(sum(hamm_score)/len(
    ↪ hamm_score)))
print('Average␣Subset␣Accuracy:␣'+str(sum(subset_accu)/len(
    ↪ subset_accu)))
print('Average␣Macro␣F-score:␣'+str(sum(macro_f1)/len(macro_f1
    ↪ )))
```

```
    print('Average␣Micro␣F-score:␣'+str(sum(micro_f1)/len(micro_f1
        ↪ )))
    print('Average␣of␣Average␣Accuracy:␣'+str(sum(avg_accu)/len(
        ↪ avg_accu)))
```