

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be “used for any purpose other than private study, scholarship, or research.” If a user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of “fair use” that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select “Pages from: first page # to: last page #” on the print dialog screen

The Van Houten library has removed some of the personal information and all signatures from the approval page and biographical sketches of theses and dissertations in order to protect the identity of NJIT graduates and faculty.

ABSTRACT

**LOOPING PREDICTIVE METHOD
TO IMPROVE ACCURACY OF A MACHINE LEARNING MODEL**

by
Subramanyam Reddy Pogili

The topic of this project is an analysis of drug-related *tweets*. The goal is to build a Machine Learning Model that can distinguish between tweets that indicate drug abuse and other tweets that also contain the name of a drug but do not describe abuse. Drugs can be illegal, such as heroin, or legal drugs with a potential of abuse, such as painkillers. However, building a good Machine Learning *Model* requires a large amount of training data. For each training tweet, a human expert has determined whether it indicates drug abuse or not. This is difficult work for humans. In this project a new “Looping Predictive Method” was developed that allows generating large training *datasets* from a small seed set of tweets by repeatedly adding machine-labeled tweets to the human-labeled tweets. With this method, an *accuracy* improvement of 15.4% was achieved from an initial set of 1,075 tweets, by expanding the training set to 29,908 tweets.

**LOOPING PREDICTIVE METHOD
TO IMPROVE ACCURACY OF A MACHINE LEARNING MODEL**

**By
Subramanyam Reddy Pogili**

**A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
In Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science**

Department of Computer Science

December 2017

Blank Page

APPROVAL PAGE
LOOPING PREDICTIVE METHOD
TO IMPROVE ACCURACY OF A MACHINE LEARNING MODEL

Subramanyam Reddy Pogili

Dr. James Geller, Thesis Advisor Professor and Associate Dean for Research of the College of Computing Sciences, NJIT	Date
--	------

Dr. Soon Ae Chun, Thesis Committee Member Professor and co-Director Information Systems and Informatics, City University of New York	Date
--	------

Dr. Hai Nhat Phan, Thesis Committee Member Professor of Computer Science, NJIT	Date
---	------

BIOGRAPHICAL SKETCH

Author: Subramanyam Reddy Pogili
Degree: Masters in Computer Science
Date: December 2017

Undergraduate and Graduate Education:

- Masters in Computer Science
New Jersey Institute of Technology, Newark, NJ, 2017
- Masters in Computer Applications
Osmania University, Hyderabad, India, 2009
- Bachelor of Science in Computer Science
Sri Venkateswara University, Tirupathi, India, 2006

Major: Computer Science

Work Experience:

Subramanyam Reddy Pogili, “worked as Software QA Engineer Intern at Externetworks Inc.” New Jersey, US, from June to August 2017.

Subramanyam Reddy Pogili, “worked as Software QA Engineer Intern at Externetworks Inc.” New Jersey, US, from September to December 2017.

Subramanyam Reddy Pogili, “worked as Lead Engineer at HCL Technologies,” Hyderabad, India, from May 2014 to Jan 2016.

Subramanyam Reddy Pogili, “worked as Engineer I R&D Support at Quark Software Inc.” Mohali, India, from July 2011 to April 2013.

Subramanyam Reddy Pogili, “worked as TS Engineer at Quark Software Inc.” Mohali, India, from May 2013 to May 2014.

Subramanyam Reddy Pogili, “worked as Senior Process Associate at Tata Consultancy Services,” Mumbai, India, from August 2009 to Jun 2011.

PERSONAL DEDICATION

The research on Looping Predictive Method is dedicated to my brother who has been great support to me for all the time during my Masters. Without his help, it would not be possible for me to travel to United States and do my Masters in computer science at New Jersey Institute of Technology. Also, my parents who asked me to do masters and sent me abroad.

ACKNOWLEDGMENT

This work is supported by Dr. Geller (Associate Dean for Research of the College of Computing Sciences) from New Jersey Institute of Technology, Dr. Chun (Professor and co-Director Information Systems and Informatics) from City University of New York, and Dr. Hai (Professor) from New Jersey Institute of Technology. I would like to extend my thanks to Sophie, who helped to create labels for the initial training dataset. This is very important to work with any Machine Learning algorithm.

TABLE OF CONTENTS

Chapter	Page
1. INTRODUCTION	1
1.1 Background.....	1
1.2 Twitter	1
2. PROCESS DATASETS.....	5
2.1 Drug Tweets	5
2.2 Machine Learning	5
2.3 Support Vector Machines	7
3. IMPLEMENTATION	9
3.1 Training Data	9
3.2 Test Data	9
3.3 Demo code showing the training and test data.....	10
3.4 Cross Validation	11
4. RESEARCH	13
4.1 Count Vector	17
4.2 Term Frequency (TF) - Inverse Document Frequency (IDF)	19
4.3 Accuracy results at every step	21
4.4 Weakness of the Model or erroneous data	24
5. FUTURE RESEARCH	26
6. CONCLUSIONS	27

TABLE OF CONTENTS (Continued)

Chapter	Page
APPENDIX A PYTHON SOURCE CODE	28
A.1 Experimental Python code	28
A.2 Python code to create a bar graph.....	48
A.3 Exporting the Model as binary.....	51
A.4 Draw Comparison Chart using Python Code	52
APPENDIX B JAVA SOURCE CODE	55
B.1 Class definition to extract tweets from Twitter.....	55
B.2 Prototype declaration for getting the drug information from database	57
B.3 Invoke Twitter API (Application Programming Interface) to extract actual tweets	58
B.4 Mapping Java Code with database to store tweets	62
B.5 Remove HTML tags from the tweets if any	64
APPENDIX C COMPARISON CHART	65
REFERENCES	66

LIST OF TABLES

Table	Page
1.1 List of tweet extraction parameters	3
4.1 Tweet count for training or to predict on each iterative mode	16
4.2 Sample Count Matrix of size 4 X 14	18
4.3 Term Frequency	19
4.4 Normalized Term Frequency	19
4.5 TF-IDF Matrix	20
4.6 Performance when using CountVectorizer	22
4.7 Erroneous Data on each Iteration	24

LIST OF FIGURES

Figure	Page
4.1 Looping Predictive Method diagram.....	15
4.2 Graph showing the improved performance with CountVectorizer.....	22
4.3 Graph showing the improved performance with TF-IDF.....	23

LIST OF DEFINITIONS

Abuse Tweet	Tweet that contains drug-abuse-related information
Accuracy	How closely the Machine Learning algorithms classify the tweets between abuse (true) or Not-abuse (false). This is a standard formula in statistics.
Compressed Row	A numeric format of a document or tweet in the Sparse Matrix representation.
Dataset	In this thesis, dataset refers to a group of structured documents or tweets saved into a single file.
Dictionary	A list of unique vocabulary terms that is extracted from the tweets containing drug names.
Estimator	Part of an algorithm computing an estimate of the performance (correctness) for the given data by following a rule and using observed data as input.
Model	Model is a (Python) object that is the output of a Machine Learning algorithm such as Support Vector Machine. The Model is then used to predict the labels of test data and other new data.
Opioid	Opioid is a drug that is used to relieve pain. It acts on the nervous system of the humans. Many opioids are used as prescription drugs.
Sparse Matrix	A group of documents or tweets is converted into number format and arranged into a matrix that is called 'Sparse Matrix.'
Text Feature	A compressed row of a document or tweet with the designated label (0 or 1 and unique tweet ID) and ready to feed into the Machine Learning algorithm.
Tweet	A message that is posted on Twitter is called a 'tweet.'

CHAPTER 1

INTRODUCTION

1.1 Background

The United States are suffering from a drug epidemic. This includes both illegal drugs and legal drugs that are over-prescribed or prescribed under false claims. More rapid increase in the use of *opioid* drugs has been reported in the United States than in any other country (Vox, 2017). Opioids that can be legally prescribed have caused many deaths in the United States. One way to better understand this problem is to look at social media. Twitter is chosen as a good source for information about the drug epidemic. Understanding the size of the problem and its distribution in the country would make it easier to plan supporting measures. For example, if there is a cluster of drug tweets tied to a certain location, the local government could consider starting a drug treatment program in the closest hospital.

The main goal of this research is to create a classification system that distinguishes between tweets that indicate drug abuse and tweets that do not indicate drug abuse. Machine Learning was used to build a Model that can differentiate between these two kinds of tweets. Among many popular Machine Learning Models, the Support Vector Machine (SVM) Model was chosen to work with (Understanding support vector machines, 2017). However, the initial results using SVM were disappointing. Therefore, our secondary goal in this thesis research was to find a new, better method using SVM to get better results in classifying drug-related tweets.

1.2 Twitter

Twitter is a web application that is accessible across the globe from the domain twitter.com (<https://twitter.com/>). Twitter is recognized across the world by its signature bird logo. It was developed by Jack Dorsey, Noah Glass, Biz Stone, and Evan Williams in the year 2006. The main idea of this application is to post messages in the network of people and interact with them. In Twitter terminology each message that is posted in the network, is referred to as a 'tweet.' To be able to post a tweet, the user must be signed up and logged into the portal. The length of a tweet is restricted to 140 characters. The

users who are not signed into the portal can still read the tweets by opening the application with the link “https://twitter.com/.” Twitter also implemented mobile applications for Android and iOS environments. They can be downloaded from Playstore [Android] and AppStore [iOS].

Twitter also exposes API services to extract the tweets from its network. API is an Application Programmable Interface service that is built by specifying the appropriate routes, protocols and rules. ‘Routes’ refer to the context path of an API through which the service is accessible. For example, searching tweets based on different criteria (words, phrases, hashtags, account names, languages, date ranges), one can use the API URL (<https://twitter.com/search-advanced?lang=en>). The context path here is **/search-advanced?lang=en**. ‘Protocols’ refer to web protocols such as HTTP, FTP, DNS, etc. ‘Rules’ define how and who can access the API. In case of Twitter, an API key provides accessibility to Twitter tweets. To get the API key, one would need to sign up for a Twitter account and submit an application for API key access.

To search and extract the drug-related tweets, one can use the Twitter Search API. This is one of the services provided by Twitter and it was used to fetch the tweets that contain the drug names “meperidine, Cocaine, Codeine, Delphine, fentanyl, Heroin, hydromorphone, LSD, Opiates, Oxycodone, PCP, Ritalin, Benzodiazepines, and opioid.” The search API allows the users to search in the tweets that are posted in the past seven days. This means one can only search the tweets in the last seven days. The Search API can only get the relevant tweets. If you want to get all the tweets, one will need to use the streaming API. The streaming API is documented at Twitter API documentation (Twitter developer documentation, 2017). Following are the steps to build a sample query to extract all relevant tweets through the API service:

- The standard API search URL (Twitter, 2017) is “https://api.twitter.com/1.1/search/tweets.json.” Https defines the secure protocol used by Twitter. ‘api.twitter.com’ is the domain, and ‘/1.1/search/tweets.json’ is the context path.
- To search for tweets that contain the drug name ‘Cocaine,’ one needs to append a key value pair to the above URL. For example, <https://api.twitter.com/1.1/search/tweets.json?q=Cocaine>

Here ‘q’ refers to query. It searches for any tweets that contain the value ‘Cocaine’.

- Once the query has been constructed, one can trigger this API with the query using a programming language such as Python, and get the relevant tweets.
- To trigger the above API, Twitter asks for authentication and authorization. This is documented at (Twitter developer documentation, 2017).

One can also append different operators along with the query word. The operators will modify the behavior. The users will need to ensure that the query operators are URL encoded, before they append them to the search API. Following is a list of operators and their usage.

Table 1.1 List of Tweet Extraction Parameters

Operator	Finds Tweets...
watching now	Containing both “watching” and “now.” This is the default operator.
“happy hour”	Containing the exact phrase “happy hour.”
love OR hate	Containing either “love” or “hate” (or both).
beer -root	Containing “beer” but not “root.”
#haiku	Containing the hashtag “haiku.”
from: interior	Sent from Twitter account “interior.”
list: NASA/astronauts-in-space-now	sent from a Twitter account in the NASA list astronauts-in-space-now.
to: NASA	A Tweet authored in reply to Twitter account “NASA.”
@NASA	Mentioning Twitter account “NASA.”

politics filter: safe	Containing “politics” with Tweets marked as potentially sensitive removed.
puppy filter: media	Containing “puppy” and an image or video.
puppy -filter: retweets	Containing “puppy,” filtering out retweets.
puppy filter: native_video	Containing “puppy” and an uploaded video, amplify video, Periscope, or Vine.
puppy filter: periscope	Containing “puppy” and a Periscope video URL.
puppy filter: vine	Containing “puppy” and a Vine.
puppy filter: images	Containing “puppy” and links identified as photos, including third parties such as Instagram.
puppy filter: twimg	Containing “puppy” and a pic.twitter.com link representing one or more photos.
hilarious filter: links	Containing “hilarious” and linking to URL.
puppy url: amazon	Containing “puppy” and a URL with the word “amazon” anywhere within it.
superhero since:2015-12-21	Containing “superhero” and sent since date “2015-12-21” (year-month-day).
puppy until:2015-12-21	Containing “puppy” and sent before the date “2015-12-21.”
Movie -scary :)	Containing “movie,” but not “scary,” and with a positive attitude.
Flight :(Containing “flight” and with a negative attitude.

CHAPTER 2

PROCESS DATASETS

2.1 Drug Tweets

People communicate with each other by posting messages in the network of Twitter Application. Here, each message is referred to as a tweet. The tweets contain a drug name such as meperidine, cocaine, codeine, benzodiazepines, opioid, etc. indicate that they are sharing some information about a drug. These tweets are distinguished into abuse and non-abuse tweets. Following are examples of both categories:

Abuse Tweet: *I was on oxycontin a while back for some severe abdominal pain and it made me so stupid I still cringe*

Non-Abuse Tweet: *Whats the name of the girl who broke your heart OxyContin...*

In order to determine whether or not a tweet should be labeled as indicating drug abuse, one first has to determine in what context it is using the drug. For example, there are many tweets where people are spamming random nonsensical words and happen to mention a certain type of drug. There are also cases of someone tweeting about something in the news that is related to drug abuse. In such cases, one labels the tweets with a 0 (non-drug abuse), since the tweet is not giving any indication that the user is taking drugs. There are cases when a tweet appears to be a computer-generated advertisement, as there are many duplicates of the same tweets that are slightly modified and are advertising a certain drug. Since these tweets are most likely generated by a computer, one marks these tweets with a 0. In other cases, a user outright says that they have taken certain drugs and is getting high. Then, one marks the tweet with a 1, indicating that this tweet was showing signs of drug abuse by the user.

2.2 Machine Learning

Machine Learning is a field in Artificial Intelligence that develops algorithms to learn from data and predict results, without the need of any programming. Since 1959, the Machine Learning has advanced from the study of pattern recognition and computational learning theory in Artificial Intelligence,

according to Arthur Samuel (Samuel, 1959). Machine Learning algorithms are segregated into supervised and unsupervised learning methods.

Unsupervised Learning

Unsupervised Machine Learning algorithms that make an assumption that a dataset contains input data, but is not associated with any labels. It is a good technique to look for hidden structures in a dataset. The most common unsupervised learning methods are Hierarchical clustering, k-Means clustering, Gaussian mixture Models (Unsupervised learning, 2017), Self-organizing maps, and Hidden Markov Models (MathWorks, 2017), (Unsupervised learning, 2017), (Zoubin Ghahramani, 2004). Cluster analysis is widely a used unsupervised learning algorithm in experimental data analysis to discover hidden patterns and groupings in datasets.

Supervised Learning

Supervised Machine Learning algorithms let machines learn from inputs and known responses. In Supervised Learning, one trains the machine by providing a list of inputs and corresponding responses. The learned knowledge is stored in the form of a Models. Later, this mode can be used to predict new results based on the past evidence. Here, the machine is learning from observations and predicting the responses for new data. Exposing the Model to more observations, the machine improves the predictive performance. Once the machine is trained with sufficient training data, it will be able to provide the target or response for a new input that is likely but not guaranteed to be correct. In supervised learning, one deals with two types of problems: classification problems and regression problems. If the response is expressed as distinct, then it is called classification problem. If the response space is continuous then it is a regression problem. Some of the supervised learning algorithms are Support Vector Machines (SVM), Naive Bayes, and Nearest Neighbor (Supervised Learning, 2017), (Statistical classification, 2017).

Following are the steps involved in Supervised Machine Learning:

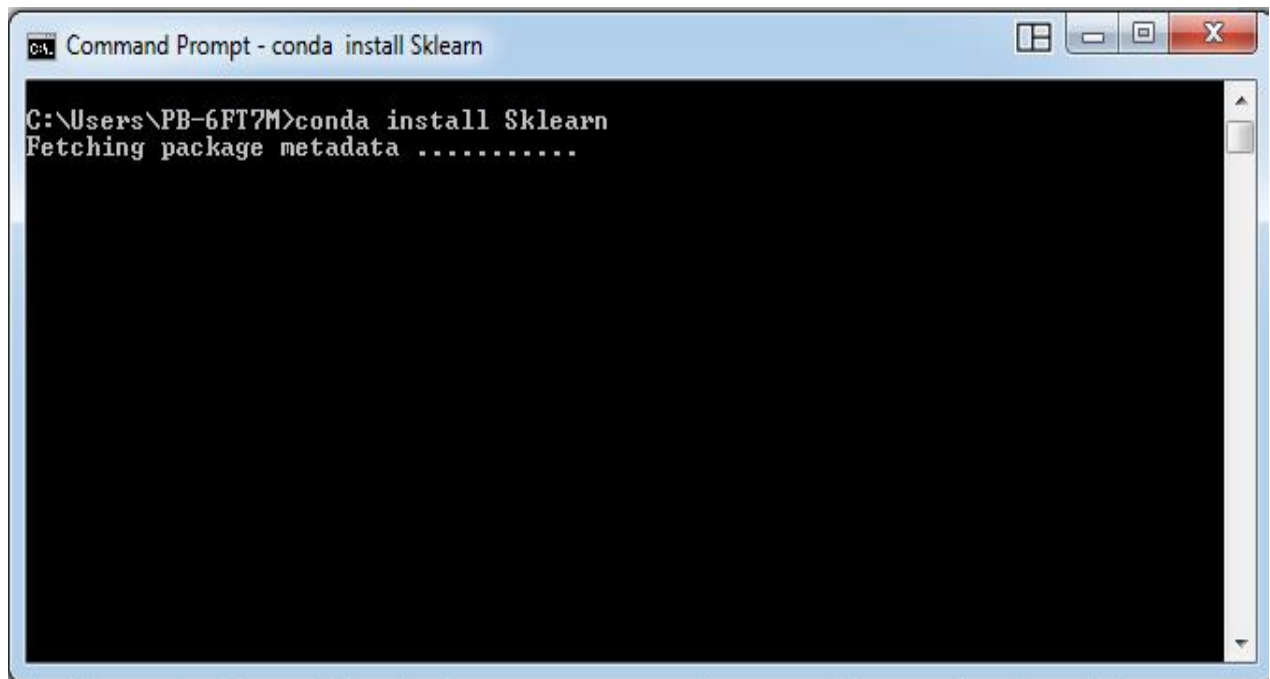
- Create human-labeled dataset of observations and results

- Choose a Machine Learning algorithm
- Build a Model by loading observations and results
- Check the accuracy of predictions by comparing the results on new observations with human results
- Export and use the Model on observations with unknown results

2.3 Support Vector Machines

Support Vector Machine (SVM) is a classification algorithm that is used for Modeling and predicting categorical variables. SVM understands only numbers. The Model can learn text features, but they have to be fed in as numbers. The SVM Model classifies the text, based on the training data. In Python, there is a library 'sklearn' which provides the classification capabilities. To use this capability, one needs to install the Sklearn package and import the library 'SVM'.

Command to install sklearn package



```
Command Prompt - conda install Sklearn
C:\Users\PB-6FT7M>conda install Sklearn
Fetching package metadata .....
```

#Command to import the library 'svm' from the sklearn package

```
from sklearn import svm
```

following is the sample prediction code in python

```
In [66]: first_Model=svm.SVC(kernel='linear').fit(train_features, y_train)
[LibSVM]
```

The SVM library contains a classification function called ‘Support Vector Classification (SVC)’, it can be used to classify the text based on the training dataset. The training dataset contains the tweets that let SVC learn Model and use it to classify the new Tweets. The SVC Model learns the features that are defined in numeric format. Thus, the features are supposed to be converted into numbers through algorithms like Word to Vector (Scikit-learn, 2017), TF (Term Frequency)/ IDF (Inverse Document Frequency) (Wikipedia, 2017).

Accuracy is defined as follows:

$$ACC = \frac{(TP + TN)}{(TP + FP + TN + FN)}$$

It is used to judge the quality of the learned Model (Sensitivity and specificity, 2017). TP stands for True Positive. A True Positive is the prediction that an observation belongs to a category C and it really belongs to C. TN stands for True Negative. True Negative is the prediction that an observation does not belong to a category C and it really does not belong to C. FP stands for False Positive. False Positive is the prediction that an observation belongs to a category C and it does not. FN stands for False Negative. False Negative is the prediction that an observation does not belong to a category C and it really belongs to C.

The accuracy would be low, when predicting the new data with the Model that is trained with less training data. Here the accuracy is calculated based on the number of tweets that are accurately labeled by a Machine Learning algorithm over the total number of machine-labeled tweets. For instance, if the Model has predicted 1000 tweets, among which 900 tweets were correctly predicted, then the accuracy is 90% $((900/1000) * 100)$.

CHAPTER 3

IMPLEMENTATION

3.1 Training Data

The initial data W fed to the Model is called training data. The Training Data is required by the Machine Learning Model to learn prediction capabilities. Humans classify the training data as either Abusive or Non-Abusive. The Machine Learning algorithms cannot process the vocabulary; therefore, the tweets are converted into a numeric format, as 1 or 0. Here, 1 represents Abuse and 0 is referred to as Non-Abuse. For instance, the following a sample that was classified as Abuse (1) by a human:

I was on oxycontin a while back for some severe abdominal pain and it made me so stupid I still cringe

Many tweets like the above constitute the training data 1,075 tweets. This data cannot be fed directly to the SVC Model. A *Sparse Matrix* representation was created by extracting the features for each tweet and storing them as *Compressed Row* format (Sparse Matrix, 2017), (Feature Extraction, 2017). The Sparse Matrix is fed into the SVC algorithm along with the human labels, which then builds the Model to classify the new tweets. As noted before, the Model only understands the Sparse Matrix that is created by extracting the features in number format. The training dataset was created by students at NJIT, under the instructions of Prof. Geller, Prof. Chun, and Prof. Phan.

3.2 Test Data

Test data is a portion of the human-labeled data that was not used for training. This data is used to test the Model that was built with the training data. It is again a group of tweets that are labeled by humans. Firstly, the features for the test data are extracted using the same *Dictionary* (see Section, 4.1). Then the Model uses these features to predict abuse or non-abuse labels. Finally, the Model output is compared with the known labels to determine how accurately the tweets were classified.

3.3 Demo Code Showing Training and Test Data

To do an experiment with Support Vector Machines (SVM), one must follow strict requirements for training and testing datasets. The main requirement in this process is that one will need to make the data available in a numeric format. The categorical data that is extracted from Twitter in the form of tweets should be converted to numeric values using any libraries or algorithms. After converting the data into numeric format, the features of the tweets were extracted, and a Sparse Matrix is created. Later, the Sparse Matrix is supplied to SVM Model to train. It was observed that the features from the human labeled tweets were used to predict the new datasets.

Check the shape of the training data in Python:

```
: HLTweets.shape  
: (1794, 2)
```

The training data contains 1794 records and '2' refers to two columns.

Check the shape of training data after extracting the features and created the Sparse Matrix:

```
train_features.shape  
(1075, 56542)
```

Here, 1075 refers to the number of features and 56542 refers to the number of unique words in the Sparse Matrix.

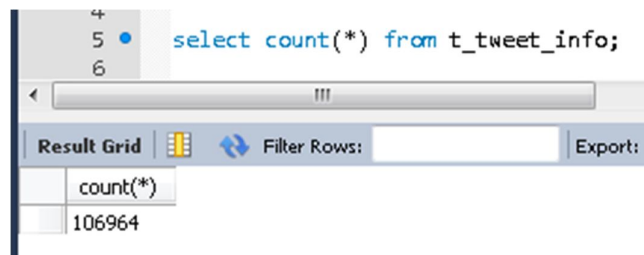
Check the shape of test data after extracting the features and created the Sparse Matrix:

```
test_features.shape  
(717, 56542)
```

Here, 717 refers to the number of features and 56542 refers to the number of unique words in the Sparse Matrix.

The total number of drug related raw tweets (not cleaned) that are store in a MySQL database is 106,964.

The number is found by the following SELECT statement on MySQL Workbench.



3.4 Cross Validation

Cross validation is one of the methods that is used to evaluate the performance of an *estimator*. It is possible that the Machine Learning Model would show perfect score if tested with part of the original training data, but would fail to correctly predict labels for previously unseen data. To avoid this problem, it is always required in Machine Learning experiments to hold out part of the labeled data and test the Model with it. I have used the Python 3.6 library 'train_test_split' to create the training and test data of different folds such as [60,40], [70, 30], [80,20], [75,25] respectively, and compare the performance.

The best performance of the Model was achieved when the Model was fed with 60% training data and tested with 40% test data.

The human-labeled dataset contains two columns: Tweets and Label.

```
HLTweets.columns.values  
array(['Tweets', 'Label'], dtype=object)
```

Following is the sample call of the Python function to split the data into training and test sets.

```
Training_input, Test_input, Training_flag, Test_flag  
= train_test_split(HLTweet_features, HLTweets['Flag'], test_size=0.25)
```

Training_input is a variable that holds the 75% of the data to train the Model.

Test_input is another variable that stores 25% of the data to test the Model.

Training_flag is a variable of array that contains all the labels for training.

Test_flag holds the labels for test.

HLTweet_features is a Sparse Matrix that is created by extracting the word features from the human labeled tweets.

HLTweets['Flag'] is a column contains all the human classification labels.

Test_size is a parameter that specifies how much test data one wants to extract from the given dataset.

This works on a scale from 0 to 1. '0' means 0 %, and 1 defines 100%. In the above example, 0.25 indicates, that the algorithm will randomly extract 25% of the data for testing.

CHAPTER 4

RESEARCH

The traditional approach to classify tweets is to train a Machine Learning Model first and use it afterwards to classify new tweets. In this research, the Machine Learning algorithm “Support Vector Machines” (SVM) was used to perform the experiments. The initial experiment was performed using 1,075 human-labeled tweets with 4 different trials where the size of training data is distributed as 60%, 75%, 80 %, 90% respectively. The SVM Model achieved between 64% to 79% accuracy in predicting the different new sets of data. The reasons for this low accuracy were as follows. As the number of tweets was low, the features extracted from the tweets were insufficient. The SVM Model was not trained with enough features, and it was not able to accurately predict the labels for test data.

The new approach ‘Looping Predictive Method’ boosted the accuracy of SVM Model by 15.4%. CountVectorizer was used to extract the features from the tweets and generate Sparse Matrix. CountVectorizer is a Python library (Scikit-learn, 2017) and it is an open source. Firstly, it creates a Dictionary with unique words from all the tweets by assigning a unique id to each word. Later, it creates unique combinations for each complete tweet and generate a Sparse Matrix that SVM Model can understand. For instance, if the vocabulary of a sample tweet *‘I feel the barbiturates in my blood’* is extracted and a Dictionary is created, it would look like in the Sparse Matrix.

I	feel	the	barbiturates	in	my	blood	
[232	3134	12	323	17	43	545]

Following is a sample screenshot showing the Dictionary vocabulary from the original experiment.

```
In [27]: dictionary32.vocabulary_
```

```
Out[27]: {'rule': 1539864,  
          'cherrytorn': 256577,  
          'thx': 1741478,  
          'rad': 1465046,  
          'director': 382466,  
          'domme': 400859,  
          'friend': 563100,  
          'https': 720949,  
          'znnq': 2093673,  
          'rg': 1498779,  
          'steer': 1657389,  
          'kobebryant': 941530,  
          'playground': 1374428,  
          'scored': 1573230,  
          'pts': 1406823,  
          'set': 1586216,  
          'world': 1954634,  
          'records': 1483913,  
          'todo': 1758460,  
          'update': 4460450}
```

Above, the left column contains terms and right column shows the unique identification number assigned to each term.

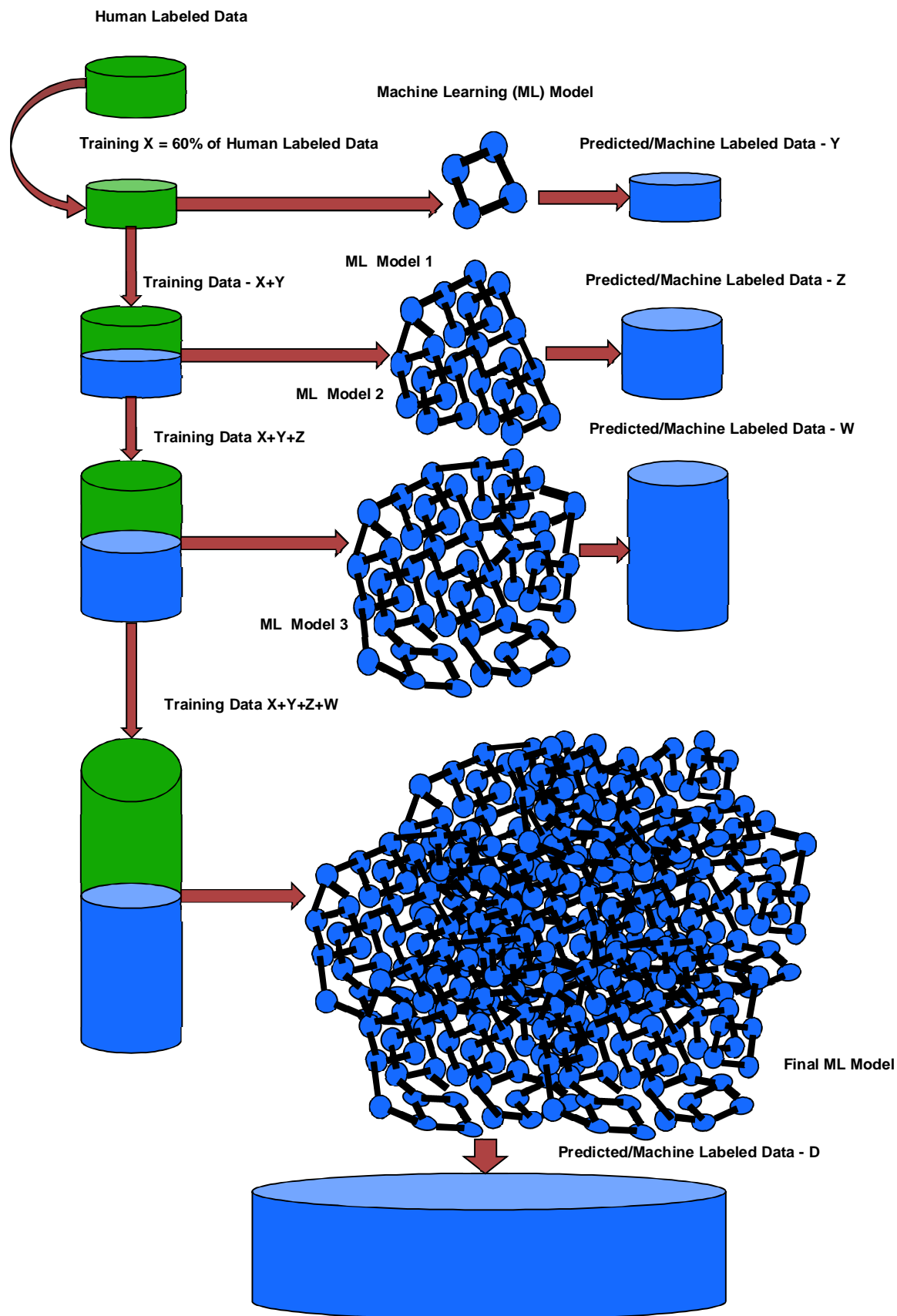


Figure 4.1 Looping predictive method diagram

In Fig. 4.1, the set T_1 of 1,076 human-labeled tweets is combined with the set T_2 of 1,076 tweets labeled by using our initial Model M_1 . This results in a new set T_2 of 2,152 tweets. Next, T_2 is fed into SVM again, building a second Model M_2 . The output T_3 of this second Model is then combined with the input data T_2 resulting in a new data set T_3 of 4,304. By this approach we are doubling the size of the output at every step. This approach continues with the sizes as follows: 1,076; 2,152; 4,304; 8,608; 17,216; 34,432; 68,864; until the count reaches 137,728 tweets. These many tweets were collected from Twitter.

Table 4.1 Tweet Count for Training or to Predict on each Iterative Mode

Training Set	Model Level	Prediction Count
1075	First Model	1075
2152	Second Model	2152
4304	Third Level	4304
8608	Fourth Level	8608
17216	Fifth Level	17216
29908	Sixth Level	29908
Training X	Final Model	Predict All Data P

As shown in the Table 4.1, the training data X was increased by combining the machine-labeled data P. Initially, 1,075 tweets were used for training and the Model predicted 1,075 tweets. At the next level, the human-labeled training data of 10,76 tweets was combined with machine-predicted labels 1,076, and therefore the data in updated training set was consisted of 2,152 labeled tweets. Using this updated training dataset, another 2,152 tweets were predicted. A new Model was built with 4,304 labeled training tweets and 4,304 newly machine-labeled tweets were generated. This process continued until the final Model was built by training 29,908 tweets and the Model labeled 29,908 new tweets.

4.1 Count Vector

Machine Learning algorithms cannot understand the raw data. Thus, the raw data cannot be directly fed to the algorithms as they cannot process the strings and they expect the data in numeric form. One needs to convert the strings into numeric format to make them compatible with Machine Learning Models. The common ways to extract the features from the strings are tokenization and counting occurrences.

In tokenization, documents are converted into tokens through white-spaces or punctuations and a unique id is assigned to each token or word. These tokens of each document are arranged into a Sparse Matrix in numeric format. Tokens are fit into columns of the matrix and document are arranged as rows. The process of extracting the features from the documents into numerical feature vectors is called 'vectorization.' The Machine Learning algorithms can process these feature vectors for Modelling and learning prediction capabilities. Scikit provided the library 'CountVectorizer' to extract features from text. Following is an example of feature extraction.

Document (D₁): *Pills do not cure pain pain*

Document (D₂): *This is unsuitable for human life*

Document (D₃): *No wonder everyone here needs OxyContin*

Document (D₄): *Pills cure pain temporarily*

Consider D is the number of documents and N is the number of unique tokens that are extracted from the corpus of tweets. These tokens later form the Dictionary and the size of the vector is given by D X N. Here, the rows in the Count Vector matrix are created from the frequency of tokens in the document.

Unique tokens: $N = | \text{'Pills', 'do', 'not', 'cure', 'pain', 'This', 'unsuitable', 'human', 'life', 'no', 'here', 'need', 'OxyContin', 'temporarily'} | = 14$

Number of documents: $D = 4$

Table 4.2 Sample Count Matrix of size 4 X 14

Documents	Unique tokens in the Dictionary													
	Pill (21)	do (22)	not (23)	cure (24)	pain (25)	this (27)	unsuitable (32)	human (33)	life (34)	no (35)	here (41)	need (28)	OxyContin (30)	temporarily (20)
D ₁	1	1	1	1	2	0	0	0	0	0	0	0	0	0
D ₂	0	0	0	0	0	1	1	1	1	0	0	0	0	0
D ₃	0	0	0	0	0	0	0	0	0	1	1	1	1	1
D ₄	1	0	0	1	1	0	0	0	0	0	0	0	0	1

In Table 4.2, the rows correspond to documents and the columns correspond to tokens with unique integer IDs in the Dictionary. Here, the document can be read as document D₄ has ‘Pill’: once, ‘cure’: once, ‘pain’: once, ‘temporarily’: once. The columns may also be understood as vectors for the corresponding words. For instance, the word vectors for the tokens ‘pain’, ‘life’, ‘human’ in the Sparse Matrix are [25,1], [34,1], [33,1] respectively.

4.2 Term Frequency (TF) - Inverse Document Frequency (IDF)

Each document in the corpus can also be treated as ‘bag’ of words. Every document is represented as a vector; however, the term weight is computed using TF-IDF. In Machine Learning TF-IDF is used to transform the raw documents to a normalized TF or TF-IDF representation.

Term Frequency (TF): Term weight of a term is the number of times the term T_i appears in the document D_j . it is denoted as F_{ij} .

Term Frequency formula:
$$TF_{ij} = \frac{F_{ij}}{\text{Max}\{F_{1j}, F_{1j}, F_{1j}, F_{1j}, \dots, F_{|v|j}\}}$$

Document (D₁): *Pills do not cure pain pain*

Document (D₂): *This is unsuitable for human life*

Document (D₄): *Pills cure pain temporarily*

Table 4.3 Term Frequency

	Pills	Do	Not	Cure	Pain	This	unsuitable	human	life	temporarily
D1	1	1	1	1	2	0	0	0	0	0
D2	0	0	0	0	0	1	1	1	1	0
D3	1	0	0	1	1	0	0	0	0	1
TF	2	1	1	2	3	1	1	1	1	1

Normalized TF = Term Frequency/max (Term Frequencies)

Table 4.4 Normalized Term Frequency

	Pills	Do	Not	Cure	Pain	This	unsuitable	human	life	temporarily
D1	½=0.5	½=0.5	½=0.5	½=0.5	2/2=1	0/2=0	0/2=0	0/2=0	0/2=0	0/2=0
D2	0/1=0	0/1=0	0/1=0	0/1=0	0/1=0	1/1=1	1/1=1	1/1=1	1/1=1	0/1=0
D3	1/1=1	0/1=0	0/1=0	1/1=1	1/1=1	0/1=0	0/1=0	0/1=0	0/1=0	1/1=1

Inverse Document Frequency (IDF): Total number of documents over the number of documents T_i appears.

IDF formula: $IDF_i = \log\left(\frac{N}{DF_i}\right)$

N: Number of documents

DF_i : Number of documents containing the term T_i

Pills	:	$\log(3/2)$	$= \log(1.5)$	$= 0.176$
Do	:	$\log(3/1)$	$= \log(3)$	$= 0.477$
Not	:	$\log(3/1)$	$= \log(3)$	$= 0.477$
Cure	:	$\log(3/2)$	$= \log(1)$	$= 0.176$
Pain	:	$\log(3/3)$	$= \log(1)$	$= 0.000$
This	:	$\log(3/1)$	$= \log(3)$	$= 0.477$
Unsuitable	:	$\log(3/1)$	$= \log(3)$	$= 0.477$
Human	:	$\log(3/1)$	$= \log(3)$	$= 0.477$
Life	:	$\log(3/1)$	$= \log(3)$	$= 0.477$
Temporarily	:	$\log(3/1)$	$= \log(3)$	$= 0.477$

TF-IDF matrix = Term Frequency * Inverse Document Frequency

Table 4.5 TF-IDF Matrix

	Pills	Do	Not	Cure	Pain	This	unsuitable	human	life	temporarily
D1	$1*0.176$	$1*0.477$	$1*0.477$	$1*0.176$	$2*0.000$	$0*0.477$	$0*0.477$	$0*0.477$	$0*0.477$	$0*0.477$
D2	$0*0.176$	$0*0.477$	$0*0.477$	$0*0.176$	$0*0.000$	$1*0.477$	$1*0.477$	$1*0.477$	$1*0.477$	$0*0.477$
D3	$1*0.176$	$0*0.477$	$0*0.477$	$1*0.176$	$1*0.000$	$0*0.477$	$0*0.477$	$0*0.477$	$0*0.477$	$1*0.477$

	Pills	Do	Not	Cure	Pain	This	unsuitable	human	life	temporarily
D1	0.176	0.477	0.477	0.176	0	0	0	0	0	0
D2	0	0	0	0	0	0.477	0.477	0.477	0.477	0
D3	0.176	0	0	0.176	0	0	0	0	0	0.477

4.3 Accuracy Results at Every Step

The data set that is labeled by humans to test the above approach contains two headers: *Tweets* and *Flag*.

Total records in the human-labeled dataset are 1794.

First Model was built by training the SVM Model with 1,076 tweets and testing with 717 tweets.

Accuracy: 79%

Second Model: training dataset

```
HLTweets.columns.values  
array(['Tweets', 'Flag'], dtype=object)  
  
HLTweets.shape  
(1794, 2)
```

The Model is built by training 60% of the data.

```
len(train)  
1075
```

Following is the first Model that's created with the above training data.

```
first_Model=svm.SVC(kernel='linear', C=1,random_state=10000000,verbose=3)  
first_Model=first_Model.fit(train_features, y_train)
```

Here, the option 'verbose' is set to 3 for printing details logging information to the screen.

Test data size:

```
len(y_test)|  
717
```

Accuracy after prediction:

```
first_Model.score(test_features,y_test)  
0.79079497907949792
```

The number of tweets to try the approach that was explained in Section 4, is below.

```
tweets.shape  
(48056, 1)
```

Result are demonstrated in Table 4.6.

Table 4.6 Performance when using CountVectorizer

Training Data Size	Accuracy (%)
1075	79
2152	88
4304	90
8608	92
17216	93.9
29908	94.4

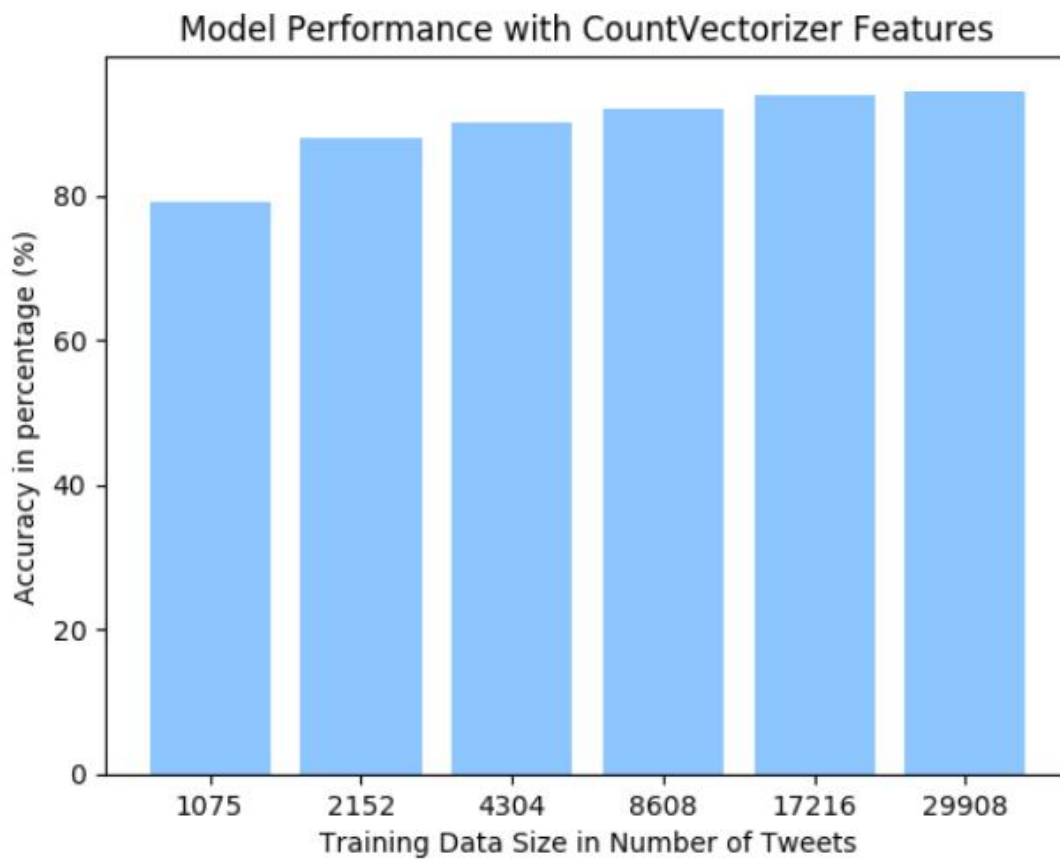


Figure 4.2 Graph showing the improved performance with CountVectorizer

The Model was tested by feeding the data in six iterations. At the first iteration, when it was tested with the training data of 1075 tweets, it showed an accuracy result of 79%. After the sixth iteration, the result was improved to 94.4%, where the size of the training data reached 29,908 tweets. The Looping Predictive Model helped to boost the performance by 15.4% through CountVectorizer.

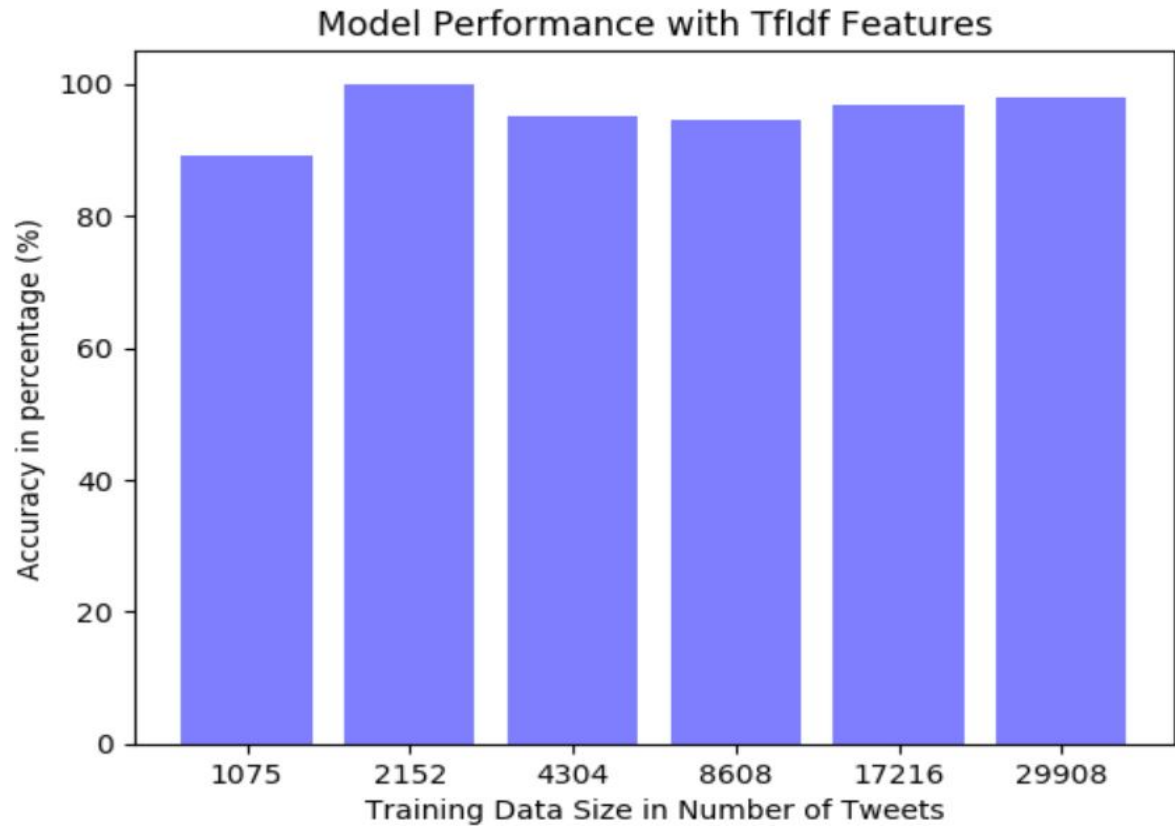


Figure 4.3 Graph showing the improved performance with TF-IDF

In Figure 4.3, it is illustrated that the accuracy of TF-IDF prediction algorithm was not stable, not showing any consistent improvement, and it showed high accuracy through all the iterations. Thus, the Looping Predictive Model was not the best fit with TF-IDF features. On the other hand, CountVectorizer showed consistent improvement on each iteration and this iterative approach was a better with CountVectorizer.

4.2 Weakness of the Method or Erroneous Data

Erroneous data is a certainty that one must deal with in any data processing through Machine Learning Models. This has a negative impact on data quality that one feeds into the Machine Learning Model and may produce additional erroneous predictive results. A thorough analysis on the factors of erroneous data and finding workarounds will help improve the data quality. The weakness of the Looping Predictive Method is that the training data used in this method on every iteration has some erroneous data, because it was automatically generated. After the first iteration, when the method used 100% accurate data that was labeled by humans, it displayed 79% accuracy. This means the prediction result had 21% percent of erroneous data. The algorithm incorrectly labeled 225 tweets from the input of size 1,075 tweets. On the second iteration, the Model labeled 258 tweets incorrectly from 2,152. After six iterations, the error rate was drastically reduced from 21% to 5.6%. This confirms that the Model used some erroneous data on each iteration. However, one can observe the error rate decreasing and accuracy improving.

Table 4.7 depicts detailed information of each iteration.

Table 4.7 Erroneous Data on each Iteration

Iterations	Training Data Size	Test Data Size	Accuracy (%)	Incorrect Labels	Error Rate (%)
I ₁	1075	717	79	225	21
I ₂	2152	7+130=847	88	258	12
I ₃	4304	1435	90	430	10
I ₄	8608	2870	92	688	8
I ₅	17216	5738	93.9	1050	6.1
I ₆	29908	11476	94.4	1674	5.6

In Table 4.7, the first iteration I_1 used 1,075 tweets as a training dataset and the Model was able to predict test data labels with 79% accuracy. Here, a 21% error rate Err_1 was observed. After the second iteration I_2 , the error rate was reduced to 12% and the Machine Learning Model was able to predict the labels for 2,152 tweets with 88% accuracy. After performing the iterations $I_1, I_2, I_3, I_4, I_5, I_6$ by doubling the training data size, the error rate dropped from 21% to 5.6%. This can be further improved.

The drawback of this research is that the Looping Predictive Method was not tested with human-labeled data in all iterations. The first iteration was tested with the human-labeled data and 79% accuracy was achieved. The second iteration was built with the training dataset that was created by combining the machine-labeled 1,076 tweets with 1,076 human-labeled tweets.

In the second iteration, the Model was tested with updated test data and the prediction capability was increased by 9% ($= 88\% - 79\%$). The revised test data contained 130 machine-labeled tweets and the 717 human-labeled tweets. However, the updated test data had 21% of incorrect machine-labeled tweets. Through all the iterations, this test data was combined with some portion of the incorrect machine-labeled data. Thus, the improvement with the performance was not good enough, because, the model was not tested with the original human-labeled data in all iterations and test data had incorrectly labeled tweets. This tradeoff was necessary due to the difficulty of getting humans to label tweets.

CHAPTER 5

FUTURE RESEARCH

In this research, six iterative Models were experimented with, with a maximum of 29,908 tweets. The Looping Predictive Method can be further extended to work with larger numbers of tweets by creating bigger training datasets, extracting more vocabulary, and adding the converted documents or tweets to the Sparse Matrix. To get a good accuracy of prediction, the Model must be tested with human-labeled data through all iterations. However, following are the challenges that may occur while working on this approach with massive datasets.

- Processing text is a very expensive task in terms of system memory
- Looping Predictive Method requires to run multiple times and it takes more time on every iteration as the number of tweets increases
- Small scale systems can be crashed if one attempts to run the approach with huge datasets. Thus, one needs to deploy and run the code on large scale hardware
- Running time with every iterative Model will be at least, doubled as the number of features fed into Model is getting doubled

To improve the accuracy, one could create a custom Dictionary with phrases that contain specific negative and positive words, and use that dictionary while extracting the features for inclusion in the Sparse Matrix. Doing more research on this approach and testing with the human-labeled dataset could decrease the error rate. Once the Model has been built with consistent performance, one must try to find ways to optimize the run time. This is very important when the Model is used in real-time applications, as end users expect fast response time.

CHAPTER 6

CONCLUSIONS

This thesis introduced a new approach in Machine Learning, where the accuracy of a Machine Learning Model that was built using a Support Vector Machine (SVM) was improved by 15.4%. The initial iteration showed 79% accuracy on testing the Model with human-labeled data. The second iteration was implemented by doubling the training data and the performance of the Model was improved by 9%, The training data was doubled through combining machine labeled data with human-labeled data. To ensure the Model performs consistently, six iterations were performed, and every iteration was executed by doubling the training data from the previous iteration. The performance of the Model was stable, and the accuracy rate was consistently improved throughout the six iterations.

The research was performed using the Machine Learning algorithm “Support Vector Machine.” Constructing the data for the SVM Model was challenging as the tweets for each iteration had to be cleaned, inserted in a Python array object and converted into numeric format. Specially, when transforming the human-labeled data into a Sparse Matrix, it took an effort to ensure that the tweet features and human-label for that tweet were correctly placed in the Sparse Matrix. Overcoming all these problems, the experiment was performed successfully, and the output was collected and analyzed. The experimental outcome demonstrated that the iterative method boosted the accuracy of SVM Model. In summary, it was proven that adding more data to the training dataset will increase the predictive capabilities of Machine Learning Model.

APPENDIX A

PYTHON SOURCE CODE

A.1 Experimental Python code

```
# import the library OS, it helps to work with operating system functionality
import os
```

```
# check the current working directory and store into 'cwd' variable/object
cwd = os.getcwd()
```

```
# print the value of 'cwd'
print(cwd)
```

C:\Users\PB-6FT7M\Subu\python

```
# set the working directory where the training and
# testing data sets are available
os.path.realpath('\\SUBU-PC\Tweets\pythonwd')
```

'C:\\SUBU-PC\\Tweets\\pythonwd'

```
# Load all the tweets into a python object from the TSV file
Tweets = pd.read_csv("C:/subu/python/workdir/Tweets.tsv", na_filter=False)
```

```
# Load all the human labeled tweets into a python object from the TSV file
HLTweets = pd.read_csv("C:/subu/python/workdir/HLDataSet.tsv",
                        na_filter=False,
                        quoting=3,
                        sep="\t")
```

```
#Print the python object and see if the data is Loaded
print(HLTweets)
```

		Tweets	Flag
0	A client with a history of abusing barbiturate...		0
1	Have frecklesYesYou suffer from Alcohol addict...		0
2	sherlock fucking holmes of Rx just uncovered t...		0

3	Have boils on your ear lobesYou probably have ...	0
4	Have you eaten todayYou probably have Bipolar....	0
5	I havent smoked a cig in about a month AKA its...	1
6	will only the three bs can kill you beerbarbit...	0
7	I feel the barbiturates in my blood	1
8	AlexWodak also Barbiturates = sleeping pills e...	0
9	Have a lack of motivationYou probably have ADD...	0
10	kathygriffin Id rather take a handful of barbi...	0
11	Ill hit you as hard as barbiturates in the rib...	0
12	Cosain02Cosain foxnation ChelseaClinton Barbit...	0
13	The Use of Benzodiazepines for Tinnitus System...	0
14	RT MadInAmerica "Study Reveals Benzodiazepines...	0
15	Psychiatric drugs are doing us more harm than ...	0
16	RubinReport See they put Benzodiazepines and s...	0
17	.realDonaldTrump Treatment 4 bipolar mania are...	0
18	Good thing benzodiazepines are cheaper than a ...	1
19	owldara112 Im thinking benzodiazepines before ...	1
20	RT IBSRemedy Chronic use of certain sedativehy...	0
21	Chronic use of certain sedativehypnotic drugs ...	0
22	Yes. Terrible. And for some benzodiazepines an...	0
23	Study Nearly 1 in 3 patients who used heroin c...	0
24	people say BenZo and i just think of benzodiaz...	0
25	mhcreek Hahahaha Thats way different than anti...	1
26	If youve seen the movie "Limitless" then you k...	1
27	I am so relieved I have a cracker box full of ...	1
28	But heres to the pain of only seeing you when ...	1
29	Have an imaginary friendYou probably have Mela...	0
...
1764	RT yeezyzus person of cocaine	0
1765	I want some fucking cocaine	1
1766	I see it in ya face I got the good stuff cocai...	1
1767	RT ARBasgall i too take my cocaine from glass ...	0
1768	RT LADEKRANE Nigerian lawSteal 5 yrsCocaine 8 ...	0
1769	That was only the beginning Read killthemessenger	0
1770	The Contras Cocaine and U.S. Covert Operations	0
1771	RT blurryaids AmazingPhil sniff cocaine on my ...	0
1772	i too take my cocaine from glass bottles	1
1773	RT yeezyzus person of cocaine	0
1774	RT Awpalot The jokes on you officer. That brea...	1
1775	RT TobyHater This year more people will use co...	0
1776	Named her cocaine.	0
1777	PsyQoKolby rainy days in your head Cocaine hyd...	0
1778	glittercocaine nice Haha i want taco soup	0
1779	i see it in your facei got the good stuff.coca...	0
1780	Lets get him in the HOF who cares about those ...	0


```

1781 RT Adweek Two men and one cocaine bear just ma... 0
1782                                         Amen 0
1783 sdchargerlover enews LOLOLOLOLOL in my mind it... 0
1784 RT BREWMYSTEW jasmine HETERO THAT COCAINE FETU... 0
1785                                         I like girls that like girls ??? 0
1786 RT Sadnightvibes 7. Yayo unreleased video unre... 0
1787 cocainemamii nooo your always good enough ur n... 0
1788                                         RT ShekeidraK Lmfao? 0
1789 RT Adweek Two men and one cocaine bear just ma... 0
1790 greece Athens Rochester woman charged outside ... 0
1791 RT setaveli Dont nobody fwm hmp or none of that ? 0
1792 How are you so thinWhips out vial of cocaine."... 0
1793 RT CeIebsInHS math teacherspoke way too fastob... 0

```

[1794 rows x 2 columns]

```

# check the titles of tweets that are loaded in the python object
HLTweets.columns.values

```

```

array(['Tweets', 'Flag'], dtype=object)

```

```

#create empty dictionary
tweet_dictionary = CountVectorizer(analyzer = "word",
                                   tokenizer = None,
                                   preprocessor = None,
                                   stop_words = 'english',
                                   max_features = 10000000)

```

```

#define a python class to clean the tweets

```

```

def process(tweet_text, preproc):

    # Remove html tags from the tweets
    tweet_text = BeautifulSoup(tweet_text,preproc).get_text()

    # Filter only the text using regular expressions
    extractText = re.sub("[^a-zA-Z?]", " ", tweet_text)

    # Transform the tweet text to lower case
    wordList = extractText.lower().split()

    # Purge the stop words
    stpWordList = set(stopwords.words("english"))
    meaningful_words = [w for w in wordList if not w in stpWordList]

```

```
# Return the processed tweet  
return( " ".join( meaningful_words ))
```

```
# Find the size of tweets in the python object 'HLTweets'  
HLTweets_len = HLTweets["Tweets"].size
```

```
# Print HLTweets_len object to see the Length  
print(HLTweets_len)
```

1794

```
# Declare an empty object or array variable  
clean_HLTweets = []
```

```
# Loop through all the tweets and  
#clean by calling the process function  
for i in range( 0, HLTweets_len ):  
    clean_HLTweets.append(  
        process( HLTweets["Tweets"][i], "lxml" ) )
```

```
# Find the Length of new dataset  
tweets_len = Tweets["Tweets"].size
```

```
# Print tweets_len object to see the Length  
print(tweets_len)
```

119433

```
# Declare an empty object or array variable  
# to process new dataset  
clean_tweets = []
```

```
# Loop through all the new tweets and  
#clean by calling the process function  
for i in range( 0, tweets_len ):  
    clean_tweets.append(  
        process( Tweets["Tweets"][i], "lxml" ) )
```



```
# Find the Length of cleaned tweets  
len(clean_tweets)
```

119433

```
# Print clean_tweets object to see the Length  
print(clean_tweets)
```

```
['even withdrawal benzodiazepines may cause tinnitus well', 'blastbeatfreakx r  
risk scary constantly antipsychotics antidepressants years benzodiazepines las  
t', 'im tryna get benzodiazepines rn ya digg', 'idea concern widespread benzod  
iazepines drs filled farmaci', 'klonopin slippery slope benzodiazepines ptsd  
mentalhealth', 'benzodiazepines antidepressants may spark longterm use via psy  
chcongress', 'risks taking benzodiazepines klonopin xanax ativan prescribed vi  
a youtube', 'safely tapering benzodiazepines info tips', 'call get best treatm  
ent options heroin cocaine alcohol barbiturates benzodiazepines amphetamine ec  
stasy lsd', 'benzodiazepines antidepressants may spark longterm use', 'essay d  
issertation help postmortem toxicology analysis benzodiazepines suspected acut  
e click help', 'safety benzodiazepines opioids severe respiratory disease nati  
onal prospective study', 'didnt leave husband dumb ass cheated benzodiazepines  
deserve get didnt leave', 'healing psychiatry community art book madinamerica  
bigpharma benzodiazepines', 'wickedwalnut papercakes benzodiazepines longer ge  
t paxil prozac celexa clonidine', 'benzodiazepines associated increased body s  
way elderly potentially lead', 'longterm use benzodiazepines elderly lead phar  
macological syndrome symptoms including', 'djpsywarrior benzodiazepines hope w  
orks well', 'use prescribed benzodiazepines associated increased rate attempt  
ed completed suicide', 'studies shown longterm use benzodiazepines benzodiazep  
ine receptor agonist', 'chrissanford kielyclan factors habit forming drugs lik
```

```
# Create a data frame  
# for the cleaned human labeled tweets  
first=pd.DataFrame(data={"Tweets":clean_HLTweets})
```

```
# Create a data frame  
# for the cleaned raw tweets  
second=pd.DataFrame(data={"Tweets":clean_tweets})
```

```
# Create an array of data frames  
# with first and second data frames  
dataFrames = [first, second]
```

```
# Extract the features from all the tweets
all_tweet_features = tweet_dictionary
                        .fit_transform(combineData['Tweets'])
```

```
# print the vocabulary from the dictionary
sorted(tweet_dictionary.vocabulary_|
```

```
['aa',
 'aaa',
 'aaaaa',
 'aaaaaa',
 'aaaaaaa',
 'aaaaaaaa',
 'aaaaaand',
 'aaaaaarms',
 'aaaaam',
 'aaaaand',
 'aaaaddictttt',
 'aaaah',
 'aaaahhh',
 'aaaahhhhh',
 'aaaand',
 'aaaanyway',
 'aaaccckkkkkkkk',
 'aaagh',
 'aaah',
 'aaahh',
```

```
# Find the unique id for the given token
# or word from the dictionary
tweet_dictionary.vocabulary_['orange']
```

63823

```
# Another way to find the unique id
tweet_dictionary.vocabulary_.get('orange')
```

63823

```
# Test the dictionary
# by transforming a sample tweet to vector
tweet_dictionary.transform
    ([ 'aa longterm use benzodiazepines '+
      'may similar effect brain alcohol also implicated' ]).toarray()

array([[1, 0, 0, ..., 0, 0, 0]], dtype=int64)
```

```
##### Loopin Predictive Model Iteration I #####
```

```
# Create train and test data sets  
# from the human Labeled data
```

```
# check the Length  
len(clean_HLTweets)
```

1794

```
# Split 60% data for training  
trn = int((len(clean_HLTweets)/100)*60)-1  
print(trn)
```

1075

```
# Create test data  
test_len=len(clean_HLTweets)-1  
train = clean_HLTweets[0:trn]  
test = clean_HLTweets[trn+1:test_len]  
print(len(test))
```

717

```
# Extract features  
# from trainig dataset into sparse matrix  
train_features = tweet_dictionary.transform(train)  
train_features
```

<1075x98860 sparse matrix of type '<class 'numpy.int64''>
with 8866 stored elements in Compressed Sparse Row format>

```
# Check shapre of training data features  
train_features.shape
```

(1075, 98860)


```
# Assign the training labels to python object
y_train = HLTweets["Flag"][0:trn]

# Check the labels are assigned properly
y_train[5:10,]
```

```
5      1
6      0
7      1
8      0
9      0
Name: Flag, dtype: object
```

```
# Extract features
# from test dataset into sparse matrix
test_features = tweet_dictionary.transform(test)
test_features
```

```
<717x98860 sparse matrix of type '<class 'numpy.int64'>'
  with 5454 stored elements in Compressed Sparse Row format>
```

```
# Check shapre of test dataset features
test_features.shape
```

```
(717, 98860)
```

```
# Assign the test labels to python object
y_test = HLTweets["Flag"][trn+1:test_len]

# Check the labels are assigned properly
y_test[5:10,]
```

```
1081    0
1082    1
1083    0
1084    1
1085    0
Name: Flag, dtype: object
```

```
# Create first machine Learning model
first_Model=svm.SVC(kernel='linear',
                    C=1,
                    random_state=10000000,
                    verbose=3).fit(train_features, y_train)
```

[LibSVM]

```
# Print the parameters used with the model
print(first_Model)
```

```
SVC(C=1, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
    max_iter=-1, probability=False, random_state=10000000, shrinking=True,
    tol=0.001, verbose=3)
```

```
# Test the model and find out the accuracy
firstModel_Accuary = first_Model.score(test_features,y_test)
print(firstModel_Accuary)
```

0.790794979079

```
##### Loopin Predictive Model Iteration II #####
```

```
# Create training dataset II for Iteration II
predict_1=clean_tweets[0:1793]
len(predict_1)
```

1793

```
# Extract features from training dataset II
predict_11 = tweet_dictionary.transform(predict_1)
predict_11
```

```
<1793x98860 sparse matrix of type '<class 'numpy.int64'>'
  with 11961 stored elements in Compressed Sparse Row format>
```

```
# Classify the labels using the first
# machine learning model (frist_Model)
predict_111 = first_Model.predict(predict_11)
```

```

# Create a data frame and
# insert the predicted labels
dataFrame_1 = pd.DataFrame(
    data={"Flag":predict_111,
          "Tweets":clean_tweets[0:1793]})
dataFrame_1 = dataFrame_1[["Tweets","Flag"]]

# Export the data frame to TSV file
dataFrame_1.to_csv( "C:/subu/python/workdir/1.tsv",
                    index=False,
                    quoting=3,
                    sep="\t")

```

```

# Combine the human Labeled data (HLTweets+1.tsv = HLML1.tsv)
# with machine Labeled data
# and train the model with it

MLHL1 = pd.read_csv("C:/subu/python/workdir/HLML1.tsv",
                    sep="\t", na_filter=False)

```

```

# Find the size of combined dataset
MLHL1_len = MLHL1["Tweets"].size
print(MLHL1_len)

```

3587

```

# Create an empty array object for MLHL1
clean_MLHL1 = []

```

```

# Process MLHL1 dataset
for i in range( 0, MLHL1_len ):
    clean_MLHL1.append(
        process( MLHL1["Tweets"][i],"lxml"))

```

```

# Check if any data is lost after cleaning
len(clean_MLHL1)

```

3587


```
# Split the data for training and test datasets  
datasetIII_len = int((len(clean_MLHL1)/100)*60)
```

```
#Create training dataset III  
trainIII=clean_MLHL1[0:datasetIII_len-1]
```

```
# Check the size of the training data  
# to ensure intended length of data is  
# split for training the model  
len(trainIII)
```

2151

```
#Create test dataset III  
testIII=clean_MLHL1[datasetIII_len:3587]
```

```
# Check the size of the test dataset  
# to ensure intended length of data is  
# split for testing the model  
len(testIII)
```

1435

```
# Extract trainIII dataset features  
trainIII_features = tweet_dictionary.transform(trainIII)  
trainIII_features
```

```
<2151x98860 sparse matrix of type '<class 'numpy.int64'>'  
  with 16979 stored elements in Compressed Sparse Row format>
```

```
# Extract testIII dataset features  
testIII_features = tweet_dictionary.transform(testIII)  
testIII_features
```

```
<1435x98860 sparse matrix of type '<class 'numpy.int64'>'  
  with 10377 stored elements in Compressed Sparse Row format>
```

```
# Create second machine Learning model
second_Model=svm.SVC(kernel='linear',
                      C=1,
                      random_state=10000000,
                      verbose=3)
                      .fit(trainIII_features,
                          MLHL1['Flag'][0:datasetIII_len-1])
```

[LibSVM]

```
# Test the second model and find out the accuracy
second_Model.score(testIII_features,
                   MLHL1['Flag'][datasetIII_len:3587])
```

0.8801393728222997

```
# Calculate the max Length of new dataset
max_len = int(len(clean_MLHL1)/2)+1+ len(clean_MLHL1)
```

```
int(len(clean_MLHL1)/2)+1
```

1794

```
# Get the tweets from int(len(clean_MLHL1)/2)+1 to max_len
predict_2=clean_tweets[int(len(clean_MLHL1)/2)+1:max_len]
```

```
# Extract the features from the dataset predict_2
predict_22 = tweet_dictionary.transform(predict_2)
```

```
# Classify the Labels using the first
# machine Learning model (second_Model)
predict_222 = second_Model.predict(predict_22)
```

```
# Create a data frame and
# insert the predicted Labels
dataFrame_2 = pd.DataFrame(
    data={"Flag":predict_222,
          "Tweets":clean_tweets[1794:1794+len(clean_MLHL1)]})
dataFrame_2 = dataFrame_2[["Tweets", "Flag"]]
```

```
# Export the data frame to TSV file
dataFrame_2.to_csv( "C:/subu/python/workdir/2.tsv",
                    index=False,
                    quoting=3,
                    sep="\t")
```

```
# Number of records predicted by second model
len(predict_2)
```

3587

```
# Combine the data of HLML1 and 2.tsv
MLHL2 = pd.read_csv("C:/subu/python/workdir/HLML2.tsv",
                    sep="\t",
                    na_filter=False)
```

```
# Find the Length of the MLHL2 dataset size
MLHL2_len = MLHL2["Tweets"].size
MLHL2_len
```

7174

```
# Create an empty array object or variable
clean_MLHL2 = []
```

```
# Process the MLHL2 dataset
for i in range( 0, MLHL2_len ):
    clean_MLHL2.append( process( MLHL2["Tweets"][i], "lxml" ))
```

```
# Check the size of the cleaned MLHL2 dataset
len(clean_MLHL2)
```

7174

```
# Separate 60% data from MLHL2 for training
trainMLHL2_len = int((len(clean_MLHL2)/100)*60)
trainMLHL2_len
```

4304


```
# Training Dataset from MLHL2
train2=clean_MLHL2[0:trainMLHL2_len-1]
len(train2)
```

4303

```
# Test Dataset from MLHL2
test2=clean_MLHL2[trainMLHL2_len:len(clean_MLHL2)]
len(test2)
```

2870

```
# Extract features from train2
train2_features = tweet_dictionary.transform(train2)
train2_features
```

<4303x98860 sparse matrix of type '<class 'numpy.int64'>'
with 32698 stored elements in Compressed Sparse Row format>

```
# Extract features from test2
test2_features = tweet_dictionary.transform(test2)
test2_features
```

<2870x98860 sparse matrix of type '<class 'numpy.int64'>'
with 23762 stored elements in Compressed Sparse Row format>

```
# Create third machine Learning model
third_Model=svm.SVC(kernel='linear',
                    C=1,
                    random_state=10000000,
                    verbose=3)
third_Model.fit(train2_features,
                MLHL2['Flag'][0:len(train2)])
```

[LibSVM]

```
# Test the third model and find out the accuracy
third_Model.score(test2_features,
                  MLHL2['Flag'][trainMLHL2_len:len(clean_MLHL2)])
```

0.9031358885017422

```
predict_3=clean_tweets[5382:12555]
```

```
predict_33 = tweet_dictionary.transform(predict_3)
```

```
predict_333 = first_Model.predict(predict_33)
```

```
len(predict_3)
```

7173

```
# Create data frame three
dataFrame_3 = pd.DataFrame(data={"Flag":predict_333,
                                "Tweets":clean_tweets[5382:12555]})
dataFrame_3 = dataFrame_3[["Tweets","Flag"]]

# Export the predicted labels
dataFrame_3.to_csv( "C:/subu/python/workdir/3.tsv",
                    index=False,
                    quoting=3,
                    sep="\t")
```

```
# Import HLML3 to a python object
MLHL3 = pd.read_csv("C:/subu/python/workdir/HLML3.tsv",
                    sep="\t",
                    na_filter=False)
```

```
# Find the size of MLHL3
MLHL3_len = MLHL3["Tweets"].size

# Declare an empty array object
clean_MLHL3 = []
```

```
# Process MLHL3 dataset
for i in range( 0, MLHL3_len ):
    clean_MLHL3.append(process( MLHL3["Tweets"][i],"lxml"))
```

```
# Create train dataset 3
train3=clean_MLHL3[0:8607]
```

```
# Create test dataset 3
test3=clean_MLHL3[8608:14346]
```



```
# Extract features for the train3 dataset
train3_features = tweet_dictionary.transform(train3)
```

```
# Extract features from the test dataset
test3_features = tweet_dictionary.transform(test3)
```

```
# Create fourth machine Learning model
fourth_Model=svm.SVC(kernel='linear',
                      C=1,
                      random_state=10000000,
                      verbose=3)
fourth_Model.fit(train3_features,
                 MLHL3['Flag'][0:8607])
```

[LibSVM]

```
# Test the fourth model and find out the accuracy
fourth_Model.score(test3_features,
                   MLHL3['Flag'][8608:14346])
```

0.91686998954339494

```
predict_4=clean_tweets[12556:26901]
```

```
predict_44 = tweet_dictionary.transform(predict_4)
```

```
predict_444 = first_Model.predict(predict_44)
```

```
# Create dataframe 4
dataFrame_4 = pd.DataFrame(data={"Flag":predict_444,
                                "Tweets":clean_tweets[12556:26901]})
dataFrame_4 = dataFrame_4[["Tweets", "Flag"]]

# Export classified labels to TSV file
dataFrame_4.to_csv( "C:/subu/python/workdir/4.tsv",
                   index=False,
                   quoting=3,
                   sep="\t")

len(predict_4)
```

14345

```
# Import the data from HLML4 dataset
MLHL4 = pd.read_csv("C:/subu/python/workdir/HLML4.tsv",
                    sep="\t",
                    na_filter=False)
```

```
# Find the length of MLHL4 dataset
MLHL4_len = MLHL4["Tweets"].size
MLHL4_len

# Create an empty array object for MLHL4
clean_MLHL4 = []
```

```
for i in range( 0, MLHL4_len ):
    clean_MLHL4.append(process(MLHL4["Tweets"][i], "lxml"))
```

```
# Create train dataset from MLHL4
train4=clean_MLHL4[0:17215]

# Create test dataset from MLHL4
test4=clean_MLHL4[17216:28692]
```

```
# Extract features from train4 dataset
train4_features = tweet_dictionary.transform(train4)

# Fetch freature from test4 dataset
test4_features = tweet_dictionary.transform(test4)
```

```
# Create fifth model
fifth_Model=svm.SVC(kernel='linear',
                    C=1,
                    random_state=10000000,
                    verbose=3).fit(train4_features,
                                   MLHL4['Flag'][0:17215])
```

[LibSVM]

```
# Test the fifth model and find out the accuracy
fifth_Model.score(test4_features, MLHL4['Flag'][17216:28692])
```

0.9395259672359707

```
predict_55 = tweet_dictionary.transform(predict_5)
```

```
predict_555 = first_Model.predict(predict_55)
```

```
# Create a data frame 5  
dataFrame_5 = pd.DataFrame(  
    data={"Flag":predict_555,  
          "Tweets":clean_tweets[26902:55594]})  
dataFrame_5 = dataFrame_5[["Tweets","Flag"]]
```

```
# Export the newly classified data to TSV file  
dataFrame_5.to_csv( "C:/subu/python/workdir/5.tsv",  
    index=False,  
    quoting=3,  
    sep="\t")
```

```
# Import the data from HLML5.tsv  
MLHL5 = pd.read_csv("C:/subu/python/workdir/HLML5.tsv",  
    sep="\t",  
    na_filter=False)
```

```
# Check the size of MLHL5  
MLHL5_len = MLHL5["Tweets"].size  
  
# Create an empty array object for MLHL5  
clean_MLHL5 = []
```

```
for i in range( 0, MLHL5_len ):  
    clean_MLHL5.append(process(MLHL5["Tweets"][i],"lxml"))
```

```
# Train dataset 5  
train5=clean_MLHL5[0:34430]  
  
# Test dataset 5  
test5=clean_MLHL5[34431:57384]
```



```
# Extract features from train5 dataset
train5_features = tweet_dictionary.transform(train5)

# Extract features from test5 dataset
test5_features = tweet_dictionary.transform(test5)
```

```
# Create sixth model
sixth_Model=svm.SVC(kernel='linear',
                    C=1,
                    random_state=10000000,
                    verbose=3)
        .fit(train5_features,
            MLHL5['Flag'][0:34430])
```

[LibSVM]

```
# Test the sixth model and find out the accuracy
sixth_Model.score(test5_features,
                  MLHL5['Flag'][34431:57384])
```

0.95094323182154838

```
predict_6=clean_tweets[55595:106964]
```

```
predict_66 = tweet_dictionary.transform(predict_6)
```

```
predict_666 = first_Model.predict(predict_66)
```

```
# Create data frame 6
dataFrame_6 = pd.DataFrame(
    data={"Flag":predict_666,
          "Tweets":clean_tweets[55595:106964]})
dataFrame_6 = dataFrame_5[["Tweets","Flag"]]
```

```
# Export the newly labeled data to tsv file
dataFrame_6.to_csv( "C:/subu/python/workdir/6.tsv",
                    index=False,
                    quoting=3,
                    sep="\t")

len(predict_6)
```

51369

```
# Import the data from HLML6.tsv file
MLHL6 = pd.read_csv("C:/subu/python/workdir/HLML6.tsv",
                    sep="\t",
                    na_filter=False)
```

```
# Find the MLHL6 dataset size
MLHL6_len = MLHL6["Tweets"].size

# Create an empty array object for MLHL6
clean_MLHL6 = []
```

```
# Create train5 dataset
train5=clean_MLHL5[0:51645]

# Create test5 dataset
test5=clean_MLHL5[51646:86076]
```

```
# Extract features from train5 dataset
train5_features = tweet_dictionary.transform(train5)

# Extract features from test5 dataset
test5_features = tweet_dictionary.transform(test5)
```

```
# Create seventh model
seventh_Model=svm.SVC(kernel='linear',
                        C=1,
                        random_state=10000000,
                        verbose=3)
                        .fit(train5_features,MLHL5['Flag'][0:51645])
```

[LibSVM]

```
# Test the seventh model and find out the accuracy
seventh_Model.score(test5_features,
                    MLHL5['Flag'][51646:86076])
```

0.96287905193447199

A.2 Python code to create a bar graph

```
# Import the Libraryies required to draw bar chart
import matplotlib.pyplot as plt; plt.rcParamsdefaults()
import numpy as np
import matplotlib.pyplot as plt
```

```
# Arrange number of tweets
# from different Iterations on X-Axis
x_tweets = ('1075', '2152', '4304', '8608', '17216', '29908')
```

```
# Number of positions where
# bar charts are supposed to be created
y_pos = np.arange(len(X_tweets))
```

```
# Assign accuracy List from
# the iterations to a python array object
Accuracy = [79,88,90,92,93.9,94.5]
```



```
# Plotting the bar chart
plt.bar(y_pos,
        Accuracy,
        align='center',
        alpha=0.5,
        color="dodgerblue")
```

<Container object of 6 artists>

```
# Mark the position on bar chart
plt.xticks(y_pos, x_tweets)
```

```
([<matplotlib.axis.XTick at 0x7176050>,
  <matplotlib.axis.XTick at 0x72a4bf0>,
  <matplotlib.axis.XTick at 0x72dc4f0>,
  <matplotlib.axis.XTick at 0x71c6770>,
  <matplotlib.axis.XTick at 0x71c6d10>,
  <matplotlib.axis.XTick at 0x71cd2d0>],
 <a list of 6 Text xticklabel objects>)
```

```
# Setting the Label for Y-Axis
plt.ylabel('Accuracy in percentage (%)')
```

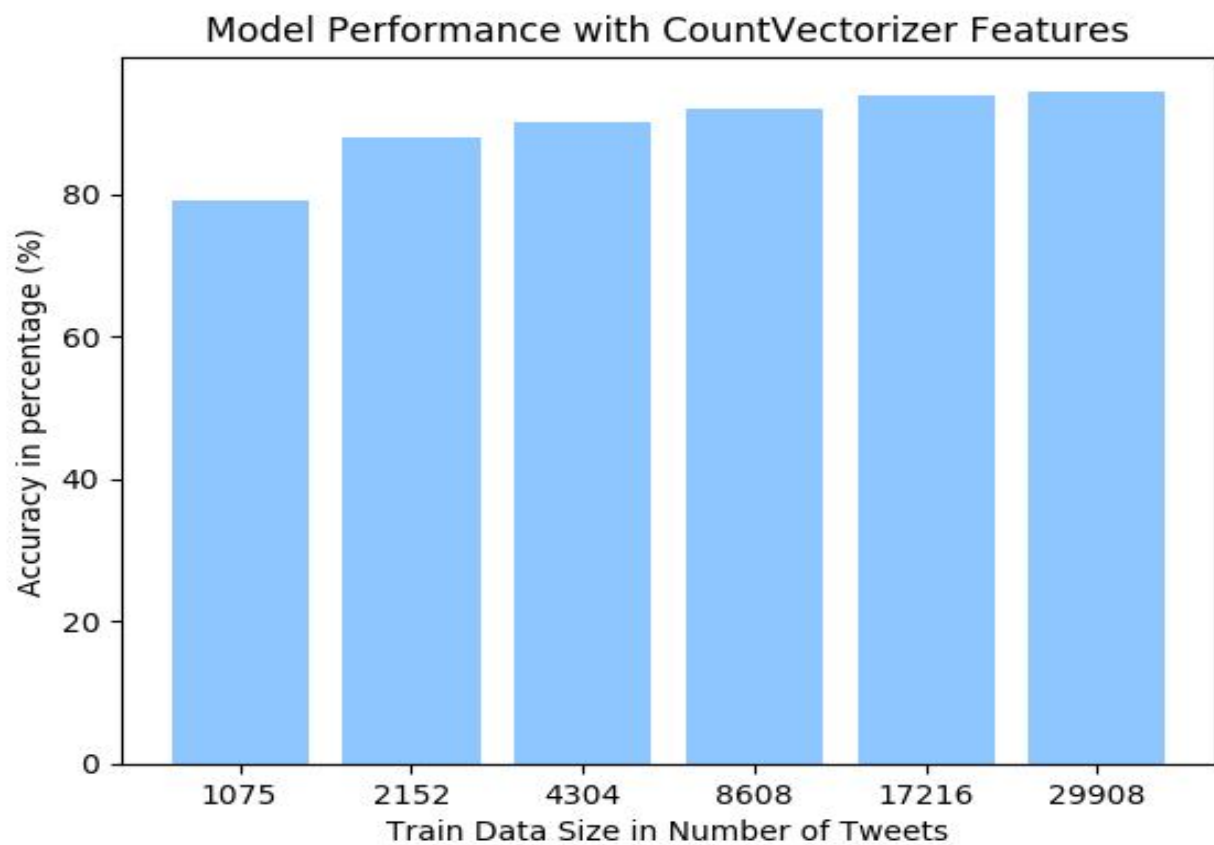
```
# Setting the Label on X-Axis
plt.xlabel('Train Data Size in Number of Tweets')
```

<matplotlib.text.Text at 0x7163450>

```
# Creating a title for the bar chart
plt.title('Model Performance with CountVectorizer Features')
```

<matplotlib.text.Text at 0x719d150>

```
# Render the bar chart that is created  
plt.show()
```



A.3 Exporting the Model as binary

```
# Import the library required to export  
# the machine learning model as binary file  
import pickle as pk
```

```
# Export the model to a specific location in binary  
# with the name 'exportedModel'  
# wb - write binary file  
pk.dump(sixth_Model,  
        open("C:/subu/Thesis/model/exportedModel", 'wb'))
```

```
# Loading the binary machine learning model  
# back to python object 'importedModel'  
# rb - read binary file  
importedModel = pk.load(  
    open("C:/subu/Thesis/model/exportedModel", 'rb'))
```

```
# Testing the imported model with test dataset  
result = importedModel.score(test_features, y_test)
```

```
# Print the accuracy of the imported model  
print(result)
```

0.927475592748

A.5 Draw comparison Chart using Python code

```
# Import the libraries required to draw bar chart
import matplotlib.pyplot as plt; plt.rcParams()
import numpy as np
import matplotlib.pyplot as plt
```

```
# Create a python and object, and assign
# the accuracy list from iterations of CountVectorizer
CountVectorizer_AccuracyPattern = (79, 88, 90, 92, 93.9, 94)

# Create a python and object, and assign
# the accuracy list from iterations of TF-IDF
TFIDF_AccuracyPattern = (89, 100, 95, 94.6, 96.7, 98)

# Create a python and object, and assign
# tweet count of each iteration
tweetCount = ('1075', '2152', '4304', '8608', '17216', '29908')

# Find size of elements
# for creating number of bars on the bar chart
bar_count = len(CountVectorizer_AccuracyPattern)
print(bar_count)
```

6

```
# Invoke subplots
fig, ax = plt.subplots()

# Create the index for bar count
index = np.arange(bar_count)

# Declare the size of the bar
bar_width= 0.25

# Declare the see through effect of the bars
opacity = 0.8
```

```

# Create bar graph for CountVectorizer
first_barChart = plt.bar(index,
                          CountVectorizer_AccuracyPattern,
                          bar_width,
                          alpha = opacity,
                          color = 'g',
                          label='CountVectorizer')

# Create bar graph for TF-IDF
first_barChart = plt.bar(index + bar_width,
                          TFIDF_AccuracyPattern,
                          bar_width,
                          alpha = opacity,
                          color = 'b',
                          label='TF-IDF')

```

```

plt.xlabel('Training Data Size (number of tweets)')
plt.ylabel('Accuracy (%)')
plt.xticks(index + bar_width + 0.25, tweetCount)

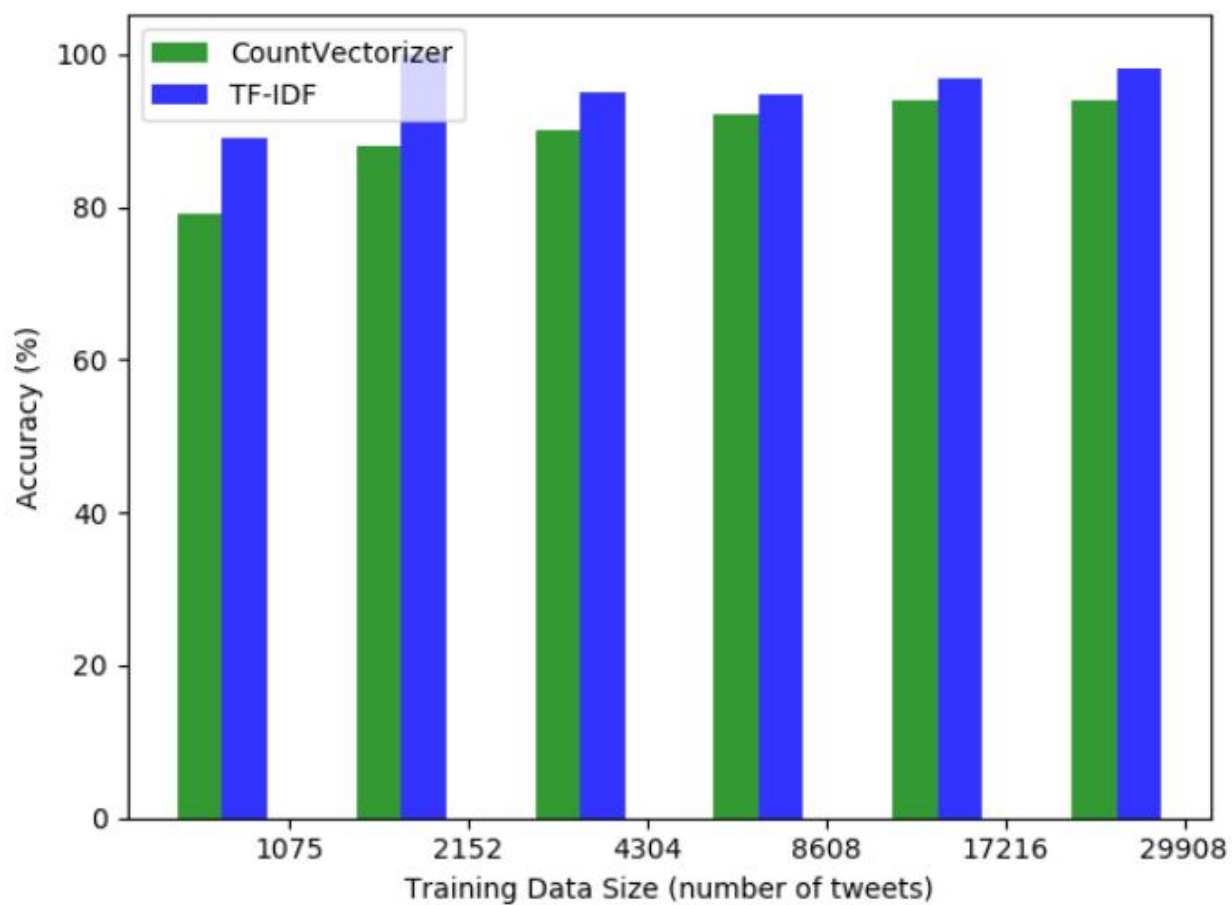
```

```

([<matplotlib.axis.XTick at 0x77296d0>,
 <matplotlib.axis.XTick at 0x7c3cdb0>,
 <matplotlib.axis.XTick at 0x7c31e30>,
 <matplotlib.axis.XTick at 0x7dc4f50>,
 <matplotlib.axis.XTick at 0x7dcb510>,
 <matplotlib.axis.XTick at 0x7dcbab0>],
 <a list of 6 Text xticklabel objects>)

```

```
# Plot the final graph  
plt.legend()  
plt.tight_layout()  
plt.show()
```



APPENDIX B

JAVA SOURCE CODE

B.1 Class definition to extract tweets from Twitter

```
1 import java.util.*;
2 import twitter4j.Status;
3
4 public class Tweet {
5
6     String Tweet;
7     String Uname;
8     Double UserLat;
9     Double UserLong;
10    Double GeoLat;
11    Double GeoLong;
12    int TId;
13    String City;
14    int RTCount;
15    Date CreatedAt;
16    int DrugId;
17
18    public Tweet() {}
19
20    public Tweet(Status s, DrugInfo d) {
21
22        setCity(s.getUser().getLocation());
23        setCreatedAt(s.getCreatedAt());
24        //setUserLat(s.getUser().getLocation());
25        if (s.getGeoLocation() != null) {
26            setGeoLat(s.getGeoLocation().getLatitude());
27            setGeoLong(s.getGeoLocation().getLongitude());
28        }
29        setRTCount(s.getRetweetCount());
30        String str = TweetExtraction.cleanString(s.getText().replaceAll("http[^\s]+", ""));
31        //str = str.replaceAll("\n", "").replaceAll("...", "");
32        setTweet(str);
33        setDrugId(d.DrugID);
34    }
35
36    public String getTweet() {
37        return Tweet;
38    }
39
40    public void setTweet(String tweet) {
41        Tweet = tweet;
42    }
43
44    public String getUname() {
45        return Uname;
46    }
47
48    public void setUname(String uname) {
49        Uname = uname;
50    }
51
52    public Double getUserLat() {
53        return UserLat;
54    }
55
```



```

56 public void setUserLat(Double userLat) {
57     UserLat = userLat;
58 }
59
60 public Double getUserLong() {
61     return UserLong;
62 }
63
64 public void setUserLong(Double userLong) {
65     UserLong = userLong;
66 }
67
68 public Double getGeoLat() {
69     return GeoLat;
70 }
71
72 public void setGeoLat(Double geoLat) {
73     GeoLat = geoLat;
74 }
75
76 public Double getGeoLong() {
77     return GeoLong;
78 }
79
80 public void setGeoLong(Double geoLong) {
81     GeoLong = geoLong;
82 }
83
84 public String getCity() {
85     return City;
86 }
87
88 public void setCity(String city) {
89     city = TweetExtraction.cleanString(city.replaceAll("http[^\s]+", ""));
90     City = city;
91 }
92
93 public int getRTCount() {
94     return RTCount;
95 }
96
97 public void setRTCount(int rTCount) {
98     RTCount = rTCount;
99 }
100
101 public Date getCreatedAt() {
102     return CreatedAt;
103 }
104
105 public void setCreatedAt(Date createdAt) {
106     CreatedAt = createdAt;
107 }
108
109 public int getDrugId() {
110     return DrugId;
111 }
112
113 public void setDrugId(int drugId) {
114     DrugId = drugId;
115 }
116
117 }

```

B.2 Prototype declaration to get the drug information from database

```
1 public class DrugInfo {  
2     public int DrugID;  
3     public String DrugInfo;  
4  
5     public DrugInfo() {}  
6  
7     public DrugInfo(int id, String info) {  
8         this.DrugID = id;  
9         this.DrugInfo = info;  
10    }  
11 }  
12
```

B.3 Invoke Twitter API (Application Programming Interface) to extract actual tweets

```
1 import java.io.*;
2 import java.util.*;
3 import twitter4j.TwitterException;
4 import twitter4j.TwitterFactory;
5 import twitter4j.Query;
6 import twitter4j.Query.ResultType;
7 import twitter4j.QueryResult;
8 import twitter4j.RateLimitStatus;
9 import twitter4j.Status;
10 import twitter4j.conf.ConfigurationBuilder;
11
12 public class TweetExtraction {
13
14     static PrintWriter pw = null;
15     static twitter4j.Twitter t = null;
16     static ArrayList < Tweet > Tweets = new ArrayList < Tweet > ();
17
18     // @SuppressWarnings("deprecation")
19     public static void main(String args[]) throws TwitterException, IOException {
20         ConfigurationBuilder cb = new ConfigurationBuilder();
21         // cb.setJSONStoreEnabled(true);
22         cb.setDebugEnabled(true).setOAuthConsumerKey(".....")
23             .setOAuthConsumerSecret(".....")
24             .setOAuthAccessToken(".....")
25             .setOAuthAccessTokenSecret(".....");
26
27         TwitterFactory tf = new TwitterFactory(cb.build());
28         t = tf.getInstance();
29         String fileRead = "List of drugs.txt";
30         // Name of the file to be read
31         String fileWrite = "CSV file 4th and 5th Nov";
32         // Name of the file to be written
33
34         FileReader fileReader = new FileReader(fileRead);
35         BufferedReader br = new BufferedReader(fileReader);
36
37         pw = new PrintWriter(new FileWriter(fileWrite + ".csv"));
38         pw.println("Drug name, Related Tweet, User name, City, Longitude");
39         pw.print("Latitude, Time of Tweet, Retweet count");
40         DBUtility db = new DBUtility();
41         ArrayList < DrugInfo > list = db.getDrugList();
42
43         String line = null;
44         try {
45             for (DrugInfo d: list) {
46                 line = d.DrugInfo;
47                 long maxID = -1;
48                 final int TWEETS_PER_QUERY = 100;
49                 final int MAX_QUERIES = 7;
50
51                 // This returns all the various rate limits
52                 // in effect for this instance with the Twitter API
53                 Map < String, RateLimitStatus > rateLimitStatus = t.getRateLimitStatus();
54
55                 // This finds the rate limit specifically for doing the search
56                 // API
57                 // call we use in this program
```



```

58 RateLimitStatus searchTweetsRateLimit
59         = rateLimitStatus.get("/search/tweets");
60
61 // See how much time is remaining
62 System.out.println("You have "
63         + searchTweetsRateLimit.getRemaining()
64         + " calls remaining out of "
65         + searchTweetsRateLimit.getLimit()
66         + ", Limit resets in "
67         + searchTweetsRateLimit.getSecondsUntilReset()
68         + " seconds\n");
69
70 Date since, until = null;
71 long DAY_IN_MS = 1000 * 60 * 60 * 24;
72 for (int queryNumber = 0; queryNumber < MAX_QUERIES; queryNumber++) {
73     // System.out.printf("\n\n!!! Starting loop %d\n\n",
74     // queryNumber);
75
76     // Do we need to delay because we've already hit our rate
77     // limits?
78     if (searchTweetsRateLimit.getRemaining() == 0)
79         // Yes
80         System.out.printf("!!! Sleeping for %d seconds due to rate limits\n",
81             searchTweetsRateLimit.getSecondsUntilReset());
82     Thread.sleep((searchTweetsRateLimit.getSecondsUntilReset() + 2) * 50);
83
84     System.out.println();
85     System.out.println("Drug name : " + line);
86     System.out.println("Related tweets - ");
87     Query searchQuery = new Query(line + " exclude:retweets");
88
89     searchQuery.setCount(TWEETS_PER_QUERY);
90     searchQuery.setLang("en");
91     searchQuery.setResultType(ResultType.mixed);
92     searchQuery.setSince("2016-11-04");
93     searchQuery.setUntil("2016-11-05");
94
95     // If maxID is -1, then this is our first call and we do
96     // not
97     // want to tell Twitter what the maximum
98     // tweet id we want to retrieve. But if it is not -1, then
99     // it
100    // represents the lowest tweet ID
101    // we've seen, so we want to start at it-1 (if we start at
102    // maxID, we would see the lowest tweet
103    // a second time.
104    if (maxID != 1) {
105        searchQuery.setMaxId(maxID - 1);
106    }
107    QueryResult qr = null;
108    try {
109        qr = t.search(searchQuery);
110    } catch (Exception e) {
111        System.out.println(e);
112        continue;
113    }
114    int totalTweets = 0;
115    for (Status string: qr.getTweets()) {
116        writeToCSV(string, line);
117        DbWriteList(string, d);

```

```

118         totalTweets++;
119         if (maxID == -1 || string.getId() < maxID) // Keep track
120             // of the
121             // lowest
122             // tweet ID
123         {
124             maxID = string.getId();
125         }
126     } // End of for Status loop
127     System.out.println("Total number of tweets = " + totalTweets);
128     if (totalTweets == 0) {
129         break;
130     }
131 } // End of for QueryNumber loop
132
133 } // End of while loop
134 br.close();
135 pw.close();
136 // fos.close();
137 } catch (Exception e) {
138     System.out.println("That didn't work well");
139     e.printStackTrace();
140 } finally {
141     br.close();
142     pw.close();
143 }
144 db.InsertTweets(Tweets);
145 } // End of try block
146
147 private static void DbWriteList(Status s, DrugInfo d) {
148     // TODO Auto-generated method stub
149     Tweets.add(new Tweet(s, d));
150 }
151
152 private static void writeToCSV(Status s, String drugName) {
153     String result = s.getText();
154     String nullString = null;
155     result = cleanString(result.replaceAll("http[^\s]+", ""));
156     result = result.replaceAll("\n", "");
157     pw.print(drugName);
158     pw.print(",");
159     pw.print(result);
160     pw.print(",");
161     pw.print(s.getUser().getName());
162     pw.print(",");
163     pw.print(s.getUser().getLocation());
164     pw.print(",");
165     pw.print(nullString);
166     pw.print(",");
167     pw.print(s.getCreatedAt());
168     pw.print(",");
169     pw.print(s.getRetweetCount());
170 }
171
172 public static String cleanString(String dirtyString) {
173     HashSet < Character > removeChars = new HashSet < Character > ();
174     for (char c: "~?&^$#@!()+-.,;<>â œ' -_*\".toCharArray()) {
175         removeChars.add(c);
176     }
177 }

```

```

182   for (Status s: t.getUserTimeline()) {
183       if (s.getRetweetedStatus().getId() == status.getId()) {
184           return true;
185       }
186   }
187   return false;
188 }
189 /*
190  * private static String calcFreq() { String[] words = new
191  * String[arr.length]; Map<String, Integer> map = new HashMap<>(); for
192  * (String w : words) { Integer n = map.get(w); n = (n == null) ? 1 : ++n;
193  * map.put(w, n); } }
194  */
195 }
196

```

B.4 Mapping Java Code with database to store tweets in database

```
1 import java.sql.*;
2 import java.text.SimpleDateFormat;
3 import java.util.*;
4
5 public class DBUtility {
6     Connection conn = null;
7
8     public Connection getConnection() throws SQLException {
9
10        // String dbURL = "jdbc:mysql://localhost:3306/";
11        String dbURL = "jdbc:mysql://localhost:3306/tweetanalysis?"
12            + "useUnicode=yes&autoReconnect=true&useSSL=false";
13        // String dbInstanceName = "tweetanalysis";
14        String dbDriver = "com.mysql.jdbc.Driver";
15        String dbUserName = "root";
16        String dbPassword = "root";
17
18        try {
19            Class.forName(dbDriver).newInstance();
20            conn = DriverManager.getConnection(dbURL, dbUserName, dbPassword);
21            // conn = DriverManager.getConnection(dbURL + dbInstanceName,
22            // dbUserName, dbPassword);
23
24        } catch (Exception e) {
25            e.printStackTrace();
26            System.out.println("Connection Failure to the database.");
27        }
28
29        return conn;
30    }
31
32    public ArrayList < DrugInfo > getDrugList() {
33        ArrayList < DrugInfo > list = new ArrayList < DrugInfo > ();
34        try {
35            conn = getConnection();
36            Statement stmt = conn.createStatement();
37            ResultSet rs = stmt.executeQuery("Select * "
38                + "from t_druginfo Where IsActive = 1");
39
40            while (rs.next()) {
41                int id = rs.getInt("Drug_ID");
42                String info = rs.getString("Drug_Name");
43                System.out.println("Drug ID = " + id);
44                System.out.println("Drug name = " + info);
45                list.add(new DrugInfo(id, info));
46            }
47        } catch (SQLException e) {
48            e.printStackTrace();
49        }
50        return list;
51    }
52
53    public void InsertTweetsBulk(ArrayList < Tweet > tweets) {
54        try {
55            conn = getConnection();
56            Statement stmt = conn.createStatement();
57            conn.createStatement().execute("SET NAMES utf8mb4");
58            String query = null;
```



```

59 query = "INSERT INTO t_tweet_info(tweet_text,Username,City,Longitude_User"
60      + ",Latitude_User,Longitude_GeoLoc,Latitude_GeoLoc,"
61      + "RT_count,Tweeted_on,DrugID) VALUES";
62 for (Tweet t: tweets) {
63
64     query += "(" + t.Tweet + "," + t.Uname + "," + t.City + "," +
65     + t.UserLong + "," + t.UserLat + "," + t.GeoLong + "," +
66     + t.GeoLat + "," + t.RTCount + "," +
67     + new SimpleDateFormat("yyyy-MM-dd").format(t.getCreatedAt())
68     + "," + t.DrugId + ")";
69
70 }
71 query = query.substring(0, query.lastIndexOf(","));
72 // System.out.println(query);
73 int rs = stmt.executeUpdate(query);
74 System.out.println(rs + " Rows executed");
75 } catch (SQLException e) {
76     e.printStackTrace();
77 }
78 }
79
80 public void InsertTweets(ArrayList < Tweet > tweets) {
81     try {
82         conn = getConnection();
83         Statement stmt = conn.createStatement();
84         conn.createStatement().execute("SET NAMES utf8mb4");
85         String query = null;
86         query = "INSERT INTO t_tweet_info(tweet_text,Username,City,Longitude_User"
87             + ",Latitude_User,Longitude_GeoLoc,Latitude_GeoLoc,RT_count"+
88             + ",Tweeted_on,DrugID) VALUES";
89         int rs = 0;
90         for (Tweet t: tweets) {
91
92             String queryValues = "(" + t.Tweet + "," + t.Uname + "," + t.City
93             + "," + t.UserLong + "," + t.UserLat + "," + t.GeoLong
94             + "," + t.GeoLat + "," + t.RTCount + "," +
95             + new SimpleDateFormat("yyyy-MM-dd").format(t.getCreatedAt())
96             + "," + t.DrugId + ")";
97             try {
98                 rs += stmt.executeUpdate(query + queryValues);
99             } catch (Exception e) {
100
101             }
102         }
103         // query = query.substring(0, query.lastIndexOf(","));
104         // System.out.println(query);
105         System.out.println(rs + " Rows executed");
106     } catch (SQLException e) {
107         e.printStackTrace();
108     }
109 }
110
111 }

```

B.5 Remove html tags from the tweets if any

```
1 import java.io.BufferedReader;
2 import java.io.FileNotFoundException;
3 import java.io.FileReader;
4 import java.io.FileWriter;
5 import java.io.PrintWriter;
6
7 public class cleanTweets {
8
9     public static void main(String[] args) throws Exception {
10         // TODO Auto-generated method stub
11         BufferedReader br = new BufferedReader(
12             new FileReader("Largest data set combined.csv"));
13         String line;
14         PrintWriter pw = new PrintWriter(new FileWriter("LargestOutput .csv"));
15
16         while ((line = br.readLine()) != null) {
17             line = line.replaceAll("[^a-zA-Z ]", "").toLowerCase();
18             System.out.println(line);
19             pw.write(line + "\n");
20         }
21     }
22 }
23
24 }
```

APPENDIX C

COMPARISON GRAPH

Figure 5.1 and 5.2 Performance comparison between when using TF-IDF and CountVectorizer.

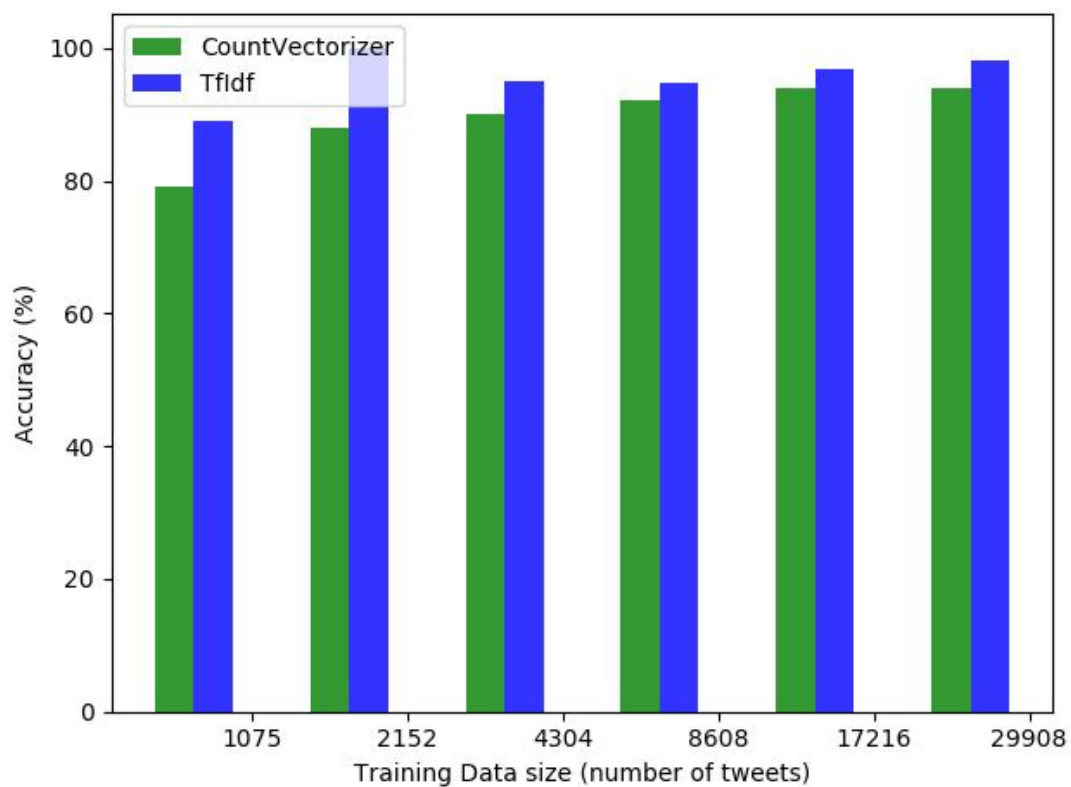


Figure A.1 Comparison Chart Depicting the performance improvement with data prediction when using the library TF-IDF and CounterVectorizer.

REFERENCES

- Twitter developer documentation (2017). Application Program Interface (API) and available methods. <https://dev.twitter.com/rest/public/search>.
- Twitter OAuth (2017). Send secure authorized requests to the Twitter API <https://dev.twitter.com/oauth>
- Machine Learning (2017). Machine Learning and different type of approaches. https://en.wikipedia.org/wiki/Machine_learning
- Machine Learning (2017). Depth introduction to Machine Learning in 25 hours. <https://www.r-bloggers.com/in-depth-introduction-to-machine-learning-in-15-hours-of-expert-videos/>
- Mathworks (2017). Elaborative information about unsupervised learning. <https://www.mathworks.com/discovery/unsupervised-learning.html>
- Supervised learning (2017). Details about Machine Learning and data mining. https://en.wikipedia.org/wiki/Supervised_learning
- Supervised learning (2017). Supervised Machine Learning and algorithms. <https://www.mathworks.com/help/stats/supervised-learning-machine-learning-workflow-and-algorithms.html>
- Wikipedia (2017). Support vector machines (SVM). https://en.wikipedia.org/wiki/Support_vector_machine
- Python (2017). Python documentation to write the Model, run, test the result. <https://docs.python.org/3/>
- A group of Machine Learning libraries ‘Numpy, Sklearn, BS4, Re, Nltk, Scipy, Numpy, Tsv, Pandas, CountVectorizer, and TF/IDF’
- Supervised learning (2017). Concept of supervised Machine Learning. https://en.wikipedia.org/wiki/Supervised_learning
- Cross Validation (2017). Evaluating estimator parameters in Machine Learning. http://scikit-learn.org/stable/modules/cross_validation.html
- Random split (2017). Splitting arrays into random train and test subsets http://scikit-learn.org/stable/modules/generated/sklearn.Model_selection.
- Understanding Support Vector Machines (2017). Practical implementation of SVM for classification <https://sadanand-singh.github.io/posts/svmpython/>
- Accuracy and precision (2017). Measure of statistical visibility, and list of formulas https://en.wikipedia.org/wiki/Accuracy_and_precision and https://en.wikipedia.org/wiki/Sensitivity_and_specificity
- Deep learning 4j organization (2017). Introduction to word2vec <https://deeplearning4j.org/word2vec.html>

Scikit-learn organization (2017). Text Feature Extraction http://scikit-learn.org/stable/modules/feature_extraction.html

Wikipedia page (2017). Term Frequency (TF) / Inverse Document Frequency (IDF), <https://en.wikipedia.org/wiki/Tf%E2%80%93idf>

Wikipedia page (2017). Unsupervised learning, https://en.wikipedia.org/wiki/Unsupervised_learning

MathWorks (2017). Unsupervised learning, <https://www.mathworks.com/discovery/unsupervised-learning.html>

Vox (2017). The opioid epidemic, <https://www.vox.com/science-and-health/2017/8/3/16079772/opioid-epidemic-drug-overdoses>

A.L. Samuel (1959). Some Studies in Machine Learning Using the Game of Checkers, <http://ieeexplore.ieee.org/document/5392560/?reload=true>

Twitter (2017). Advanced search, <https://twitter.com/search-advanced?lang=en>

Zoubin Ghahramani (2004). Unsupervised Learning, <http://mlg.eng.cam.ac.uk/zoubin/papers/ul.pdf>

Wikipedia (2017). Statistical classification, https://en.wikipedia.org/wiki/Statistical_classification

Scikit-learn (2017). CountVectorizer, http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

Wikipedia (2017). Sensitivity and specificity, https://en.wikipedia.org/wiki/Sensitivity_and_specificity

Drugfreeworld (2017). Opioids, <http://www.drugfreeworld.org/drugfacts/prescription/opioids-and-morphine-derivatives.html>

Feature Extraction (2017). Loading features form Dictionary, http://scikit-learn.org/stable/modules/feature_extraction.html

Sparse Matrix (2017). Compressed sparse row, https://en.wikipedia.org/wiki/Sparse_matrix

Wikipedia (2017). Estimator, <https://en.wikipedia.org/wiki/Estimator>