**CHAPTER – 1**

**INTRODUCTION**

* 1. **OBJECTIVE OF THE PROJECT**

The main objective of my project is to create the corpus for sentiment analysis of Tamil Tweets. The corpus is annotated and created manually after analyzing the sentiments by using classifiers such as Naïve Bayes, Support Vector Machine, K-Means Clustering and Convolutional Neural Network.

* 1. **SCOPE OF THE PROJECT**
* The corpus is available in both English and Tamil
* The corpus is mainly categorized into positive and negative.
* The corpus contains both words and sentences.
  1. **OUTLINE OF THE PROJECT**

In the above objective mentioned, the difficult task is to annotate the corpus into daily purpose, positive and negative. In my project, the task is to create the corpus for Tamil Tweets. For the creation of corpus some supervised and unsupervised classifiers are used and the final result is manually annotated.

**CHAPTER -2**

**LITERATURE DESCRIPTION**

**Sentiment Analysis of Online Tamil Contents using Recursive Neural Network Models Approach for Tamil Language**

**By R.Padmamala and V.Prema**

This paper proposes an approach involving Recursive Neural Network models for Tamil in improving the accuracy of a sentiment analyzer tool for Tamil language. To capture the meaning of phrases and thereby detecting the sentiment of them, Naïve Bayes approach the Hidden Markov Model algorithm take into consideration frequency of occurrence of keywords. Vector space models use term-document, word-context and pair-pattern matrices for the same. None of these models can capture the meaning of long phrases, ambiguous or sarcastic phrases accurately. Sentiment detection requires richer resources than just frequency of keywords. Recursive Neural Network models can be generated for Tamil language based on which sentiment analysis can be done. The accuracy of the results obtained can be enhanced using inter sentential sentiment prediction.

**The Identification of the Emotionality of Metaphorical Expressions**

**Based on a Manually Annotated Chinese Corpus**

**by Dongyu Zhang, Hongfei Lin, Puqi Zheng, Liang Yang and Shaowu Zhang**

Metaphorical expressions are frequently used to convey emotions in human communication. However, there is limited research on the detection of emotionality in metaphorical expressions, although a number of studies have focused on sentiment analysis and metaphor detection separately. We, therefore, attempt to identify emotions in Chinese metaphorical texts. We first construct a manual corpus with an annotation scheme, which contains annotations of metaphor, and emotional categories. We then use the corpus as a train-and-test set to identify the emotions in metaphorical expressions automatically with three methods. The first method is based on a field dictionary and field conflict. The second method is based on a support vector machine. The third method is based on deep learning, and it applies the long short-term memory model to identify the emotion of metaphor. The experimental results show that the third method performs better in identifying metaphor tasks, while the first method works better for emotion classification. In this paper, we compared the strength of heuristic, stochastic and deep learning approaches, which contributes to a challenging natural language processing issue: the detection of emotionality in metaphor.

**Arasen Ti-Tweet: A Corpus for Arabic Sentiment Analysis of Saudi Tweets**

**by Nora Al-Twairesh, Hend Al-Khalifa, Abdul Malik Al-Salman and**

**Yousef Al-Ohali**

Arabic Sentiment Analysis is an active research area in these days. However, the Arabic language skill lacks sufficient language resources to enable the tasks of sentiment analysis. In this paper, we present the details of collecting and constructing a large dataset of Arabic tweets. The techniques used in cleaning and pre-processing the collected dataset are explained. A corpus of Arabic tweets annotated for sentiment analysis was extracted from this dataset. The corpus consists mainly of tweets written in Modern Standard Arabic and the Saudi dialect. The corpus was manually annotated for sentiment. The annotation process is explained in detail and the challenges during the annotation are highlighted. The corpus contains 17,573 tweets labelled with four labels for sentiment: positive, negative, neutral and mixed. Baseline experiments were conducted to provide benchmark results for future work.

**ArsentD-LEV: A Multi-Topic Corpus for Target-based**

**Sentiment Analysis in Arabic Levantine Tweets**

**by Ramy Baly, Alaa Khaddaj, Hazem Hajj, Wassim El-Hajj and Khaled Bashir Shaban**

Sentiment analysis is a highly subjective and challenging task. Its complexity further increases when applied to the Arabic language, mainly because of the large variety of dialects that are unstandardized and widely used in the Web, especially in social media. While many datasets have been released to train sentiment classifiers in Arabic, most of these datasets contain shallow annotation, only marking the sentiment of the text unit, as a word, a sentence or a document. In this paper, we present the Arabic Sentiment Twitter Dataset for the Levantine dialect (ArsenTD-LEV). Based on findings from analyzing tweets from the Levant region, we created a dataset of 4,000 tweets with the following annotations: the overall sentiment of the tweet, the target to which the sentiment was expressed, how the sentiment was expressed, and the topic of the tweet. Results confirm the importance of these annotations at improving the performance of a baseline sentiment classifier. They also confirm the gap of training in certain domain, and testing in another domain.

**CHAPTER – 3**

**SYSTEM ANALYSIS**

**3.1 PROBLEM DEFINITION:**

Although the sentiment analysis can be done in Tamil it has some problems. Many tweets are available neither in English nor in Tamil. So, the analysis could not be done with English or Tamil Vocabulary. There is no dataset available for mixed English and Tamil Tweets. In this project, a corpus is created with both English and Tamil words and classified into positive and negative.

**3.2. EXISTING SYSTEM:**

The existing corpus available for Tamil Sentiment Analysis is only with pure Tamil words of vocabulary and not with sentiment. The sentiment analysis of Tamil Tweets is also difficult. There are many corpora available only for the entertainment rather than general purpose. The creation of multilingual corpora is a challenging process.

**3.3. PROPOSED SYSTEM:**

The proposed idea for creation of corpus for Tamil language contains all the common words in day-to-day use. They are categorized into positive and negative. The corpus also contains Twitter dataset available in both Tamil, English and Tamil words written in English along with sentiments.

**CHAPTER – 4**

**SYSTEM SPECIFICATION**

**4.1. HARDWARE REQUIREMENTS:**

* CPU Type : Intel i5
* Clock Speed : 2.75 GHz
* RAM Size : 8GB
* Harddisk Capacity : 121 GB

**4.2. SOFTWARE REQUIREMENTS:**

* Operating System : Mac OS
* Language : Python
* IDE : Jupyter Notebook (Spyder)

**4.3. SOFTWARE SPECIFICATION:**

**FEATURES OF PYTHON:**

Python is a clear and powerful object-oriented programming language, comparable to Perl, Ruby, Scheme, or Java.

Python is meant to be an easily readable language. Its formatting is visually uncluttered, and it often uses English keywords where other languages use punctuation. Unlike many other languages, it does not use curly brackets to delimit blocks, and semicolons after statements are optional. It has fewer syntactic exceptions and special cases than C or Pascal.

Python uses whitespace indentation, rather than curly brackets or keywords, to delimit blocks. An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block. Thus, the program's visual structure accurately represents the program's semantic structure. This feature is also sometimes termed the off-side rule.

**Some of Python's notable features:**

* Uses an elegant syntax, making the programs you write easier to read.
* Is an easy-to-use language that makes it simple to get your program working. This makes Python ideal for prototype development and other ad-hoc programming tasks, without compromising maintainability.
* Comes with a large standard library that supports many common programming tasks such as connecting to web servers, searching text with regular expressions, reading and modifying files.
* Python's interactive mode makes it easy to test short snippets of code. There's also a bundled development environment called IDLE.
* Is easily extended by adding new modules implemented in a compiled language such as C or C++.
* Can also be embedded into an application to provide a programmable interface.
* Runs anywhere, including Mac OS X, Windows, Linux and Unix, with unofficial builds also available for Android and iOS.
* It is free software in two senses. It doesn't cost anything to download or use Python, or to include it in your application. Python can also be freely modified and re-distributed, because while the language is copyrighted it's available under an open source license.

**Some programming-language features of Python are:**

* A variety of basic data types are available: numbers (floating point, complex, and unlimited-length long integers), strings (both ASCII and Unicode), lists, and dictionaries.
* Python supports object-oriented programming with classes and multiple inheritance.
* Code can be grouped into modules and packages.
* The language supports raising and catching exceptions, resulting in cleaner error handling.
* Data types are strongly and dynamically typed. Mixing incompatible types (e.g. attempting to add a string and a number) causes an exception to be raised, so errors are caught sooner.
* Python contains advanced programming features such as generators and list comprehensions.
* Python's automatic memory management frees you from having to manually allocate and free memory in your code.

**SENTIMENT ANALYSIS IN PYTHON:**

Sentiment Analysis is the process of 'computationally' determining whether a piece of writing is positive, negative or neutral. It is also known as opinion mining, deriving the opinion or attitude of a speaker. Sentiment Analysis helps organisations to measure the Return Of Investment of their marketing campaigns and improve their customer service. Since sentiment analysis gives the organisations a sneak peek into their customer's emotions, they can be aware of any crisis that's to come well in time – and manage it accordingly. Sentiment analysis is extremely useful in social media monitoring as it allows us to gain an overview of the wider public opinion behind certain topics. The ability to extract insights from social data is a practice that is being widely adopted by organisations across the world. Sentiment Analysis is the process of determining whether a piece of writing (product/movie review, tweet, etc.) is positive, negative or neutral. It can be used to identify the customer or follower's attitude towards a brand through the use of variables such as context, tone, emotion, etc.

There has been lot of work in the field of sentiment analysis of twitter data. This survey focuses mainly on sentiment analysis of twitter data which is helpful to analyze the information in the tweets where opinions are highly unstructured, heterogeneous and are either positive or negative, or neutral in some cases.

Basic sentiment analysis algorithms use natural language processing (NLP) to classify documents as positive, neutral, or negative. Programmers and data scientists write software which feeds documents into the algorithm and stores the results in a way which is useful for clients to use and understand.

**CNN ALGORITHM:**

A convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery. CNNs use relatively little pre-processing compared to other image classification algorithms.

CNNs have also been explored for [natural language processing](https://en.wikipedia.org/wiki/Natural_language_processing). CNN models are effective for various NLP problems and achieved excellent results in [semantic parsing](https://en.wikipedia.org/wiki/Semantic_parsing), search query retrieval, sentence modeling, classification, prediction and other traditional NLP tasks.

The main adventage is their accuracy in image recognition problems.

They have some disadventages:

-High computational cost.

- If you don't have a good GPU they are quite slow to train (for complex tasks).

-They use to need a lot of training data.

**NAÏVE BAYES ALGORITHM:**

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. Naive Bayes is a kind of classifier which uses the Bayes Theorem. It predicts membership probabilities for each class such as the probability that given record or data point belongs to a particular class. The class with the highest probability is considered as the most likely class.

**Advantages of Naive Bayes:**

The Naive Bayes algorithm affords fast, highly scalable model building and scoring. It scales linearly with the number of predictors and rows. ... Naive Bayes can be used for both binary and multiclass classification problems.

**Disadvantages of Naïve Bayes:**

If we need to have one of the feature is “continuous variable” (like time). Then it’s difficult to apply Naive Bayes directly, Even though you can make “buckets” for “continuous variables” it’s not 100% correct. There is no true online variant for Naive Bayes. So it is a must to keep all of our data for retraining the model. It won’t scale when the number of classes are too high, like > 100K. Even for prediction it takes more runtime memory compared to SVM or simple logistic regression

**SVM ALGORITHM:**

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. “Support Vector Machine” (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. Support Vectors are simply the co-ordinates of individual observation. Support Vector Machine is a frontier which best segregates the two classes (hyper-plane/ line).

**SVM Advantages**

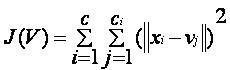
* SVM’s are very good when we have no idea on the data.
* Works well with even unstructured and semi structured data like text, Images and trees.
* The kernel trick is real strength of SVM. With an appropriate kernel function, we can solve any complex problem.
* Unlike in neural networks, SVM is not solved for local optima.
* It scales relatively well to high dimensional data.
* SVM models have generalization in practice, the risk of overfitting is less in SVM.

**SVM Disadvantages**

* Choosing a “good” kernel function is not easy.
* Long training time for large datasets.
* Difficult to understand and interpret the final model, variable weights and individual impact.
* Since the final model is not so easy to see, we can not do small calibrations to the model hence its tough to incorporate our business logic.

**K-MEANS CLUSTERING ALGORITHM:**

k-means is  one of  the simplest unsupervised  learning  algorithms  that  solve  the well  known clustering problem. The procedure follows a simple and  easy  way  to classify a given data set  through a certain number of  clusters (assume k clusters) fixed apriori. The  main  idea  is to define k centers, one for each cluster. These centers  should  be placed in a cunning  way  because of  different  location  causes different  result. So, the better  choice  is  to place them  as  much as possible  far away from each other. The  next  step is to take each point belonging  to a  given data set and associate it to the nearest center. When no point  is  pending,  the first step is completed and an early group age  is done. At this point we need to re-calculate k new centroids as barycenter of  the clusters resulting from the previous step. After we have these k new centroids, a new binding has to be done  between  the same data set points  and  the nearest new center. A loop has been generated. As a result of  this loop we  may  notice that the k centers change their location step by step until no more changes  are done or  in  other words centers do not move any more. Finally, this  algorithm  aims at  minimizing  an objective function know as squared error function given by:

[](https://sites.google.com/site/dataclusteringalgorithms/k-means-clustering-algorithm/kmeans.JPG?attredirects=0)

where,  
                           *‘||xi - vj||’* is the Euclidean distance between *xi* and *vj.*

*‘ci’* is the number of data points in *ith* cluster.

*‘c’* is the number of cluster centers.

**K-Means Advantages :**

1. If variables are huge, then  K-Means most of the times computationally faster than hierarchical clustering, if we keep k smalls.
2. K-Means produce tighter clusters than hierarchical clustering, especially if the clusters are globular.

**K-Means Disadvantages :**

1. Difficult to predict K-Value.
2. With global cluster, it didn't work well.
3. Different initial partitions can result in different final clusters.
4. It does not work well with clusters (in the original data) of Different size and Different density.

**CHAPTER – 5**

**SYSTEM DESIGN**

**5.1. SYSTEM ARCHITECTURE:**

A system architecture or systems architecture is the conceptual model that defines the structure, behavior and more views of a system. An architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structures and behaviours of the system. System architecture can comprise system components, the externally visible properties of those components, the relationships (e.g. the behavior) between them. It can provide a plan from which products can be procured, and systems developed, that will work together to implement the overall system. There have been efforts to formalize languages to describe system architecture; collectively these are called architecture description languages (ADLs).

**Various organizations define systems architecture in different ways including:**

* + An allocated arrangement of physical elements which provides the design solution for a consumer product or life-cycle process intended to satisfy the requirements of the functional architecture and the requirements baseline.
  + Architecture comprises the most important, pervasive, top-level, strategic inventions, decisions and their associated rationales about the overall structure (i.e. essential elements and their relationships) and associated characteristics and behaviour.
  + If documented, it may include information such as a detailed inventory of current hardware, software and networking capabilities.

**SYSTEM DESIGN**

Term Feature set

Pre-processing

Raw Tweets

Corpus

Classification

Training

Fig.5.1. ARCHITECTURE DESIGN

**5.2. CLASS DIAGRAM:**

A class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects. The class diagram is the main building block of object-oriented modeling. It is used for general [conceptual modeling](https://en.wikipedia.org/wiki/Conceptual_model) of the structure of the application, and for detailed modeling translating the models into [programming code](https://en.wikipedia.org/wiki/Programming_code). Class diagrams can also be used for [data modeling](https://en.wikipedia.org/wiki/Data_modeling). The classes in a class diagram represent both the main elements, interactions in the application, and the classes to be programmed. In the diagram, classes are represented with boxes that contain three compartments:

* The top compartment contains the name of the class. It is printed in bold and centered, and the first letter is capitalized.
* The middle compartment contains the attributes of the class. They are left-aligned and the first letter is lowercase.
* The bottom compartment contains the operations the class can execute. They are also left-aligned and the first letter is lowercase.

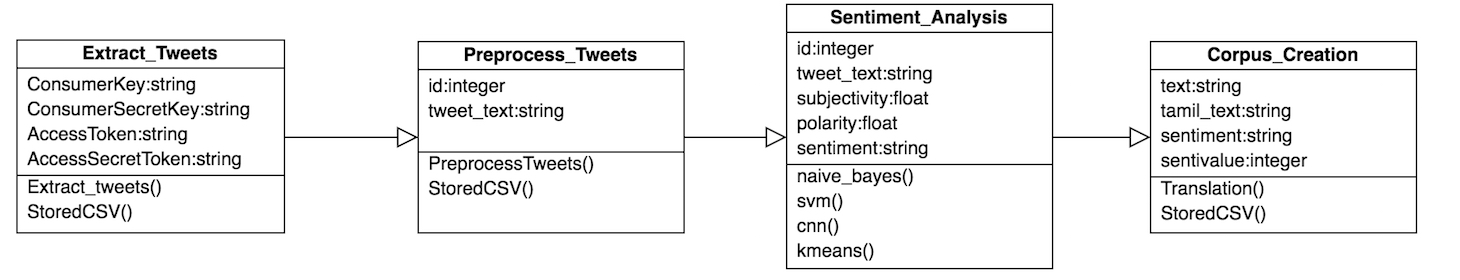


Fig.5.2. CLASS DIAGRAM

**5.3. USECASE DIAGRAM:**

A use case diagram is a dynamic or behavior diagram in UML. Use case diagrams model the functionality of a system using actors and use cases. Use cases are a set of actions, services, and functions that the system needs to perform.

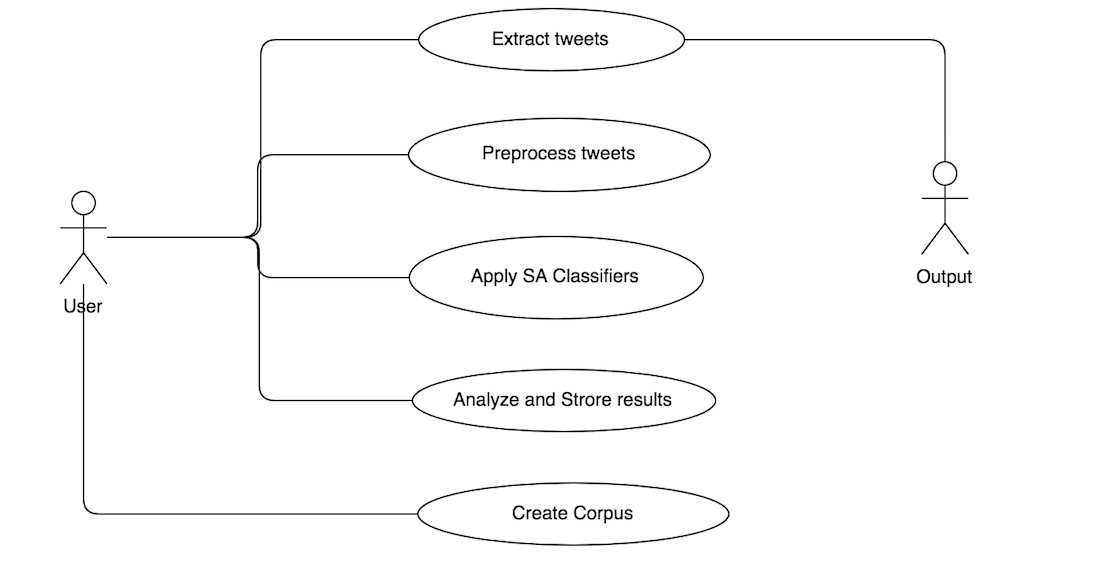
****

Fig.5.3. USECASE DIAGRAM

**5.4. FLOW DIAGRAM:**

The DFD takes an input-process-output view of a system which data objects flow into the software are transformed by processing elements and resultant data objects flow out of the software. Data objects represented by labeled arrows and transformations are represented by circles also called as bubbles. DFD is presented in a hierarchical fashion that is the first data flow model represents the system as a whole. Subsequent DFD refined the context diagram (level 0 DFD), providing increasing details with each subsequent level. The DFD enables the software engineer to develop models of the information domain and functional domain at the same time. As the DFD is refined into greater level of details the analyst performs an implicit functional decomposition of the system. At the same time the DFD refinement results in a corresponding refinement of the data as it moves through the process that embodies the applications.

A context level DFD for the system the primary external entities produce information for use by the system and consume information generated by the system. The labeled arrow represents data objects or object hierarchy.

LEVEL 0:

The Level 0 DFD shows how the system is divided into ‘subsystems’ (processes), each of which deals with one or more of the data flows to or from an external agent, and which together provide all the functionality of the system as a whole. It also identifies internal data stores that must be present in order for the system to do its job and shows the flow of data between the various parts of the system.

LEVEL 1:

The next stage is to create the Level 1 data flow diagram. This highlights the main functions carried out by the system. As a rule, to describe the system was using between two and seven functions - two being a simple system and seven being a complicated system. This enables us to keep the model manageable on screen or paper.

User

Corpus

Fig.5.4.FLOW DIAGRAM

**CHAPTER – 6**

**MODULES DESCRIPTION**

**6.1. MODULE:**

A module is a separate unit of software or hardware. Typical characteristics of modular components include portability, which allows them to be used in a variety of systems and interoperability which allows them to function with the components of other systems. The modules used in this project are

1. Twitter data set collection
2. Twitter data set preprocessing
3. Applying Naïve Bayes, SVM, CNN and K-means classifiers
4. Analysing and storing the results
5. Creation of corpus

**6.2. TWITTER DATA SET COLLECTION:**

The streaming Twitter data is collected from the Twitter using Tweepy package for sentiment analysis. For further process the collected data is stored in a .csv file with the columns id, text, polarity and subjectivity which are calculated by using text blob.

**6.3. TWITTER DATA SET PREPROCESSING:**

The collected data set stored in .csv file is taken for preprocessing and cleaning. The preprocessing includes the removal of white spaces, punctuations, hashtags, URLs and the emoticons. The preprocessed data set is stored in another .csv file for further process.

**6.4. APPLYING NAÏVE BAYES, SVM, CNN and K-MEANS CLASSIFIERS:**

The preprocessed data set is taken for classification and analysis. For this data set, the classifiers used are

1. Naïve Bayes Classifier (Multinomial Naïve Bayes)
2. SVM Classifier
3. Convolutional Neural Network (CNN)
4. K-Means Clustering Algorithm

By using these classifiers, the data is classified into three sets namely positive, negative and neutral. The preprocessed data set contains approximately 900 Tweets which is used as a training and test set. For more accuracy other training sets are also used.

**6.5. ANALYSING AND STORING THE RESULTS:**

After applying the classifiers to the data set, the accuracy is predicted for all the classifiers which may vary. For this dataset the SVM Classifier predicts high accuracy of 84% followed by CNN with 82% and Naïve Bayes Classifier with 60%. The predictions of each classifier are stored in separate files.

**6.6. CREATION OF CORPUS:**

The predicted data set is annotated into words and sentences and categorise into positive, negative and neutral. For the translation process Google translation API is used. The corpus is stored in text and .csv files.

**CHAPTER – 7**

**SYSTEM TESTING**

**7.1. TESTING:**

Testing is a series of different tests that whose primary purpose is to fully exercise the computer based system. Although each test has a different purpose, all works should verify that all system elements have been properly integrated and performed allocated function. Testing is the process of checking whether the developed system works according to the actual requirement and objectives of the system.

The philosophy behind testing is to find the errors. A good test is one that has a high probability of finding an undiscovered error. A successful test is one that uncovers the undiscovered error. Test cases are devised with this purpose in mind. A test case is a set of data that the system will process as an input.

**7.2. UNIT TESTING:**

All modules are tested individually as soon as they are completed and are checked for their correct functionality.

**7.3. INTEGRATION TESTING:**

The entire project is split into small programs; each of these single programs gives a frame as an output. These programs are tested individually; at last all these programs are combined together by creating another program where all these constructors are used. It gives a lot of program by not functioning in an integrated manner.

**7.4. VALIDATION TESTING:**

It is the process of evaluating software during the development process or at the end of the development process to determine whether it satisfies the specified business requirements. Validation testing ensures that the product actually meets the clients’ needs. It can also be defined as to demonstrate that the product fulfils its intended use when deployed on appropriate environment. It answers to the question ‘are we building the right product?’.

**CHAPTER – 8**

**CONCLUSION AND FURTHER ENHANCEMENT**

The project “A Corpus for Tamil Sentiment Analysis of Tamil Tweets” is used to create the Corpus for Tamil Tweets. For the creation of the Corpus four classifiers are used in which the two classifiers are supervised and the other two are unsupervised. The corpus is annotated manually with both sentences and words available in both English and Tamil with their respective sentiments and categories.

The project is done in Python in a simple way. The accuracy of my project is low. This accuracy can be increased by making the classifiers more trained. Here the classifiers like SVM, CNN are used. For further enhancement the unsupervised and semi-supervised classifiers such as LSTM, RNN and the combination of CNN-LSTM can be used train and predict the sentiment of the data sets.

This project is done with approximately 900 Tweets. In future, it can be increased upto millions of Tweets. The preprocessing of Tamil Tweets is more difficult. The tools available for preprocessing are also not for the bunch of files. The further enhancement is to develop the efficient preprocessing tool for the sentiment analysis of Tamil Tweets.

The metaphor recognition and the sarcasm detection is also difficult for Tamil Language as it changes for the respective situations. The further process is to develop a hybrid model to improve the accuracy and prediction of the Tamil Tweet dataset.

**CHAPTER – 9**

**APPENDICES**

**9.1. APPENDIX – 1: SOURCE CODE**

**TWEET EXTRACTION**

from textblob import TextBlob

import csv

import tweepy

import unidecode

auth1=tweepy.auth.OAuthHandler('ffhWeoODjMJoqSs2akB1ie6gV','uP63f5CZPgKqDFNPUILRc7qLuD4giwHZR03gV7QPPxGvWVVe28')

auth1.set\_access\_token('1080092593730478080LQ4jREgDtZGdKdYqyocxbYw7XUrjUG','4slpsYcz7380P9QgXRr1iZj3hFYoW8wghAk0WRYpyzJSe')

#Twitter Search

target\_num = 5000

query = "query"

dates = input('Enter starting date: ')

dateu = input('Enter ending date: ')

#Stored in CSV file

csvFile = open('/Users/sowmiyaaradha/anaconda3/envs/unsupervised/Tamil\_Corpus/Data/new\_results.csv', 'w+', newline = ' ')

csvWriter = csv.writer(csvFile)

csvWriter.writerow(["text", "polarity", "subjectivity", "date", "retweet\_count"])

counter = 0

for tweet in tweepy.cursor(api.search, q=query, lang="en", result\_type="all", since=dates, until=dateu, count=target\_num).items():

created = tweet.created\_at

text = tweet.text

text = unidecode.unidecode(text)

retwc = tweet.retweet\_count

try:

hashtag = tweet.entities[u'hashtags'][0][u'text']

except:

hashtag = "None"

username = tweet.author.name

authorid = tweet.author.id

followers = tweet.author.followers\_count

friends = tweet.auhtor.friends\_count

text\_blob = TextBlob(text)

polarity = text\_blob.polarity

subjectivity = text\_blob.subjectivity

csvWriter.writerow([text, polarity, subjectivity, created, retwc])

counter = counter + 1

if(counter == target\_num):

break

csvFile.close()

**TWITTER PREPROCESSING:**

import re

import pandas as pd

def processTweet(tweet):

tweet = tweet.lower()

tweet = re.sub('((www\.[^\s]+)|(https?://[^\s]+))','URL',tweet)

tweet = re.sub('@[^\s]+','AT\_USER',tweet)

tweet = re.sub('[\s]+', ' ', tweet)

#tweet = re.sub('r\brt\b', '',tweet)

tweet = re.sub(r'#([^\s]+)', r'\1', tweet)

tweet = tweet.strip('\'"')

#print(tweet)

return tweet

pd.set\_option('mode.chained\_assignment', None)

data=pd.read\_csv('/Users/sowmiyaaradha/anaconda3/envs/unsupervised/Tamil\_Corpus/Data/new\_results.csv')

for i in range(0,len(data)):

processedTweet = processTweet(data.iloc[:,0][i])

data.iloc[:,0][i]=processedTweet

data.to\_csv('/Users/sowmiyaaradha/anaconda3/envs/unsupervised/Tamil\_Corpus/Data/processed\_tweets.csv',index=None)

**NAÏVE BAYES CLASSIFIER:**

import re

import nltk

from nltk.sentiment.vader import SentimentIntensityAnalyzer

import pandas as pd

stopwords = nltk.corpus.stopwords.words('english')

stopwords.append('AT\_USER')

stopwords.append('URL')

def getFeatureVector(tweet):

featureVector = []

words = tweet.split()

for w in words:

w = w.strip('\'"?,.')

val = re.search(r"^[a-zA-Z][a-zA-Z0-9]\*$", w)

if(w in stopwords or val is None):

continue

else:

featureVector.append(w.lower())

return featureVector

pd.set\_option('mode.chained\_assignment', None)

data=pd.read\_csv('/Users/sowmiyaaradha/anaconda3/envs/unsupervised/Tamil\_Corpus/final\_processed.csv')

data.loc[:,'Sentiment']='neutral'

for i in range(0,len(data)):

sie=SentimentIntensityAnalyzer()

if sie.polarity\_scores(data.iloc[:,0][i])['compound'] == 0.0:

data.iloc[:,-1][i]='neutral'

elifsie.polarity\_scores(data.iloc[:,0][i])['compound'] < 0.0:

data.iloc[:,-1][i]='negative'

else:

data.iloc[:,-1][i]='positive'

data.to\_csv('final\_dataset.csv',index=None)

#Training Phase

from sklearn.feature\_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(use\_idf=True, smooth\_idf=True, sublinear\_tf=False)

t=vectorizer.fit\_transform(data.iloc[:,0].values)

from sklearn.preprocessing import LabelEncoder

le=LabelEncoder()

data.iloc[:,-1]=le.fit\_transform(data.iloc[:,-1])

#Training for Naive Bayes

from sklearn.naive\_bayes import MultinomialNB

NBclassifier=MultinomialNB()

NBclassifier.fit(t,data.iloc[:,-1])

#Testing for Naive Bayes

test\_tweets=data.iloc[:,0]

test\_vector=vectorizer.transform(test\_tweets)

arr=le.inverse\_transform(NBclassifier.predict(test\_vector))

count\_neg=0

count\_p=0

count\_neu=0

for i in arr:

if(i=='neutral'):

count\_neu+=1

if(i=='positive'):

count\_p+=1

if(i=='negative'):

count\_neg+=1

neg\_perc=count\_neg/len(data)\*100

pos\_perc=count\_p/len(data)\*100

neu\_perc=count\_neu/len(data)\*100

accuracy = (count\_neu+count\_p+count\_neg)/30

import matplotlib.pyplot as plt

labels = ['Negative','Positive','Neutral']

sizes = [count\_neg,count\_p,count\_neu]

colors=['red','green','yellow']

plt.pie(sizes, labels=labels,colors=colors)

plt.title('Analysis of tweets using Naive Bayes')

plt.savefig('/Users/sowmiyaaradha/anaconda3/envs/unsupervised/Tamil\_Corpus/Images/naivebayes.png')

plt.show()

print("Detailed Report using Naive Bayes Classifier")

print("Percentage of tweets that are Negative : "+str(neg\_perc)+'%')

print("Percentage of tweets that are Positive : "+str(pos\_perc)+'%')

print("Percentage of tweets that are Neutral : "+str(neu\_perc)+'%')

print("Accuracy of Naive Bayes Classifier:" + str(accuracy) + '%')

**SVM CLASSIFIER**

import re

import nltk

from nltk.sentiment.vader import SentimentIntensityAnalyzer

import pandas as pd

stopwords = nltk.corpus.stopwords.words('english')

stopwords.append('AT\_USER')

stopwords.append('URL')

def getFeatureVector(tweet):

featureVector = []

words = tweet.split()

for w in words:

w = w.strip('\'"?,.')

val = re.search(r"^[a-zA-Z][a-zA-Z0-9]\*$", w)

if(w in stopwords or val is None):

continue

else:

featureVector.append(w.lower())

return featureVector

pd.set\_option('mode.chained\_assignment', None)

data=pd.read\_csv('/Users/sowmiyaaradha/anaconda3/envs/unsupervised/Tamil\_Corpus/final\_processed.csv')

data.loc[:,'Sentiment']='neutral'

for i in range(0,len(data)):

sie=SentimentIntensityAnalyzer()

if sie.polarity\_scores(data.iloc[:,0][i])['compound'] == 0.0:

data.iloc[:,-1][i]='neutral'

elifsie.polarity\_scores(data.iloc[:,0][i])['compound'] < 0.0:

data.iloc[:,-1][i]='negative'

else:

data.iloc[:,-1][i]='positive'

data.to\_csv('final\_dataset.csv',index=None)

#Training Phase

from sklearn.feature\_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(use\_idf=True, smooth\_idf=True, sublinear\_tf=False)

t=vectorizer.fit\_transform(data.iloc[:,0].values)

from sklearn.preprocessing import LabelEncoder

le=LabelEncoder()

data.iloc[:,-1]=le.fit\_transform(data.iloc[:,-1])

#Training for Naive Bayes

from sklearn.naive\_bayes import MultinomialNB

NBclassifier=MultinomialNB()

NBclassifier.fit(t,data.iloc[:,-1])

#Testing for Naive Bayes

test\_tweets=data.iloc[:,0]

test\_vector=vectorizer.transform(test\_tweets)

arr=le.inverse\_transform(NBclassifier.predict(test\_vector))

count\_neg=0

count\_p=0

count\_neu=0

for i in arr:

if(i=='neutral'):

count\_neu+=1

if(i=='positive'):

count\_p+=1

if(i=='negative'):

count\_neg+=1

neg\_perc=count\_neg/len(data)\*100

pos\_perc=count\_p/len(data)\*100

neu\_perc=count\_neu/len(data)\*100

accuracy = (count\_neu+count\_p+count\_neg)/30

import matplotlib.pyplot as plt

labels = ['Negative','Positive','Neutral']

sizes = [count\_neg,count\_p,count\_neu]

colors=['red','green','yellow']

plt.pie(sizes, labels=labels,colors=colors)

plt.title('Analysis of tweets using Naive Bayes')

plt.savefig('/Users/sowmiyaaradha/anaconda3/envs/unsupervised/Tamil\_Corpus/Images/naivebayes.png')

plt.show()

print("Detailed Report using Naive Bayes Classifier")

print("Percentage of tweets that are Negative : "+str(neg\_perc)+'%')

print("Percentage of tweets that are Positive : "+str(pos\_perc)+'%')

print("Percentage of tweets that are Neutral : "+str(neu\_perc)+'%')

print("Accuracy of Naive Bayes Classifier:" + str(accuracy) + '%')

**CNN:**

import pandas

import re

import string

import tensorflow as tf

from collections import Counter

from nltk.stem.porter import PorterStemmer

from nltk.corpus import stopwords

from keras.models import Sequential

from keras.layers import Dense

from keras.callbacks import ModelCheckpoint

from keras.preprocessing.text import Tokenizer

from keras.preprocessing.sequence import pad\_sequences

from keras.layers import Flatten

from keras.layers import Embedding

from keras.layers.convolutional import Conv1D

from keras.layers.convolutional import MaxPooling1D

from sklearn.metrics import roc\_auc\_score

def clean\_text(txt):

txt = str(txt).lower()

tokens = txt.split()

isascii = lambda s: len(s) == len(s.encode())

tokens = [w for w in tokens if isascii(w)]

re\_punc = re.compile('[%s]' % re.escape(string.punctuation))

tokens = [re\_punc.sub('', w) for w in tokens]

tokens = [w for w in tokens if w.isalnum()]

re\_digt = re.compile('[%s]' % re.escape(string.digits))

tokens = [re\_digt.sub('', w) for w in tokens]

stop\_words = set(stopwords.words('english'))

tokens = [w for w in tokens if not w instop\_words]

tokens = [w for w in tokens if len(w) < 30]

tokens = [w for w in tokens if len(w) > 1]

porter = PorterStemmer()

tokens = [porter.stem(w) for w in tokens]

return tokens

def token\_to\_line(txt, vocab):

tokens = clean\_text(txt)

tokens = [w for w in tokens if w in vocab]

return ' '.join(tokens)

def process\_texts(texts, vocab):

lines = list()

for txt in texts:

line = token\_to\_line(txt, vocab)

lines.append(line)

return lines

def save\_vocab(lines, filename):

data = '\n'.join(lines)

file = open(filename, 'w')

file.write(data)

file.close()

def load\_vocab(filename):

file = open(filename, 'r')

text = file.read()

file.close()

return text

def add\_tokens\_vocab(txt, vocab):

tokens = clean\_text(txt)

vocab.update(tokens)

def build\_vocab(texts):

vocab = Counter()

for txt in texts:

add\_tokens\_vocab(txt, vocab)

save\_vocab(vocab, data\_path + "/vocab.txt")

def create\_tokenizer(lines):

tokenizer = Tokenizer()

tokenizer.fit\_on\_texts(lines)

return tokenizer

def encode\_docs(tokenizer, max\_length, docs):

encoded = tokenizer.texts\_to\_sequences(docs)

padded = pad\_sequences(encoded, maxlen = max\_length, padding='post')

return padded

def tf\_auc\_roc(y\_true, y\_pred):

value, update\_op = tf.contrib.metrics.streaming\_auc(y\_pred, y\_true)

metric\_vars = [i for i in tf.local\_variables() if 'auc\_roc' in i.name.split('/')[1]]

for v in metric\_vars:

tf.add\_to\_collection(tf.GraphKeys.GLOBAL\_VARIABLES, v)

with tf.control\_dependencies([update\_op]):

value = tf.identity(value)

return value

def define\_model(vocab\_size, max\_length):

model = Sequential()

model.add(Embedding(vocab\_size, 150, input\_length = max\_length))

model.add(Conv1D(filters = 32, kernel\_size = 8, activation='relu'))

model.add(MaxPooling1D(pool\_size=2))

model.add(Flatten())

model.add(Dense(30, activation='relu'))

model.add(Dense(1, activation='sigmoid'))

return model

data\_path = "/Users/sowmiyaaradha/anaconda3/envs/unsupervised/Tamil\_Corpus/Data"

print("Loading data sets into Memory...")

train\_df = pandas.read\_csv(data\_path + "/train.csv", quotechar='"', skipinitialspace=True, encoding='utf-8')

print("...training data dimension: " + str(train\_df.shape))

test\_df = pandas.read\_csv(data\_path + "/test\_data1.csv", quotechar='"', skipinitialspace=True, encoding='utf-8')

print("...test data (rows for prediction): " + str(test\_df.shape[0]))

print("Shuffling the training data row-wise...")

train\_df = train\_df.sample(frac=1).reset\_index(drop=True)

print("Building the neural network inputs...")

ytrain = train\_df.label

build\_vocab(train\_df.tweet)

tokens = load\_vocab(data\_path + "/vocab.txt")

texts = process\_texts(train\_df.tweet, vocab = tokens)

max\_length = max([len(t.split()) for t in texts])

tokenizer = create\_tokenizer(texts)

vocab\_size = len(tokenizer.word\_index) + 1

xtrain = encode\_docs(tokenizer, max\_length, texts)

xtest = encode\_docs(tokenizer, max\_length, process\_texts(test\_df.text, vocab = tokens))

print("Defining the neural network model...")

model = define\_model(vocab\_size, max\_length)

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=[tf\_auc\_roc])

model.summary()

filepath = data\_path + "/weights.bestmodel.hdf5"

checkpoint = ModelCheckpoint(filepath, monitor='val\_tf\_auc\_roc', verbose=1, save\_best\_only=True, mode='max')

callbacks\_list = [checkpoint]

print("Running model Build...")

class\_weight = {0 : 1000., 1: 75.}

model.fit(xtrain, ytrain, epochs = 30, validation\_split = 0.10, verbose = 0, callbacks=callbacks\_list, class\_weight = class\_weight)

print("...Model build process:COMPLETED")

model = define\_model(vocab\_size, max\_length)

model.load\_weights(data\_path + "/weights.bestmodel.hdf5")

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=[tf\_auc\_roc])

print("Running test set predictions...")

results = pandas.DataFrame(model.predict(xtest, verbose=0))

print("writing predictions to a submission file...")

test\_df["label"] = results.iloc[:,0]

test\_df["label"] = round(test\_df["label"])

test\_df = test\_df[["id", "label"]]

test\_df.to\_csv(data\_path + "/test\_predictions.csv", encoding='utf-8',index=False)

print("...Test set prediction process:COMPLETED")

**K-MEANS CLUSTERING:**

import nltk

import numpy as np

import collections

import re

import pandas as pd

import matplotlib.pyplot as plt

import sklearn as sk

import sklearn

import csv

import nbimporter

import csv\_helper

import warnings

warnings.filterwarnings("ignore")

from sklearn.metrics import silhouette\_score

from nltk.tokenize import TweetTokenizer

from nltk.corpus import stopwords

from nltk.stem import PorterStemmer

from nltk.stem.wordnet import WordNetLemmatizer

from nltk import SnowballStemmer

from csv\_helper import CSVHelper

from sklearn.cluster import DBSCAN

#required to filter noise and stem

nltk.download('stopwords')

nltk.download('wordnet')

data = CSVHelper.load\_csv("/Users/sowmiyaaradha/anaconda3/envs/unsupervised/Tamil\_Corpus/Data/kmeans\_test.csv")

#Preprocessing

def tokenize(tweets):

tknzr = TweetTokenizer()

tokenized\_tweets = []

for tweet in tweets:

tokenized\_tweets.append(tknzr.tokenize(tweet))

return tokenized\_tweets

def remove\_stopwords(tweets):

stopwords = nltk.corpus.stopwords.words('english')

stopwords.extend(("im","dont","wont"))

tweets\_nostop = []

for tweet in tweets:

tweet\_nostop = [w.lower() for w in tweet if w.lower() not in stopwords]

tweets\_nostop.append(tweet\_nostop)

return tweets\_nostop

def filter\_names(tweets):

tweets\_filtered = []

regex = re.compile("@([A-Za-z0-9\_]+)")

for tweet in tweets:

tweet\_filtered = [w for w in tweet if not regex.match(w)]

tweets\_filtered.append(tweet\_filtered)

return tweets\_filtered

def filter\_noise(tweets):

tweets\_filtered = []

regex = re.compile("[^a-zA-Z]")

for tweet in tweets:

tweet\_filtered = [re.sub("[^a-zA-Z]",'',w) for w in tweet if not regex.match(w)]

tweets\_filtered.append(tweet\_filtered)

return tweets\_filtered

def remove\_url(tweets):

tweets\_filtered = []

regex = re.compile("^https?:\/\/.\*[\r\n]\*")

for tweet in tweets:

tweet\_filtered = [w for w in tweet if not regex.match(w)]

tweets\_filtered.append(tweet\_filtered)

return tweets\_filtered

#we decided to use a lemmatizer instead of stemming

def my\_stem(tweets):

lmt = WordNetLemmatizer()

ps = PorterStemmer()

tweets\_stemmed = []

for tweet in tweets:

tweet\_stemmed = [lmt.lemmatize(w) for w in tweet]

tweets\_stemmed.append(tweet\_stemmed)

return tweets\_stemmed

all\_tweets = tokenize(data)

all\_tweets = remove\_url(all\_tweets)

all\_tweets = filter\_names(all\_tweets)

all\_tweets = filter\_noise(all\_tweets)

all\_tweets = remove\_stopwords(all\_tweets)

all\_tweets = my\_stem(all\_tweets)

#Feature Extraction

def termfreq(word, tweet):

num\_words = len(tweet)

word\_occurences = tweet.count(word)

return word\_occurences/num\_words

def count\_doc\_with\_term(word, tweets):

counter = 0

for tweet in tweets:

if word in tweet:

counter = counter + 1

return counter

#idf = amount\_of\_tweets/amount\_of\_tweets\_with\_word

def idf(word, tweets):

num\_tweets = len(tweets)

num\_tweets\_with\_word = count\_doc\_with\_term(word, tweets)

return np.log10(num\_tweets/num\_tweets\_with\_word)

def create\_vocabulary(tweets):

vocabulary = set() #creates a new set named vocabulary (set doesn't have duplicates)

for tweet in tweets: #iterates over all tweets

words = [w for w in tweet] #makes a list of words in a tweet (which our loop currently is in)

vocabulary.update(words)#update teh vocab set

return list(vocabulary)

def compute\_word\_idf(tweets, vocabulary):

word\_idf = collections.defaultdict(lambda: 0) #when I call x[k] for a nonexistent key k (such as a statement like v=x[k]), the key-value pair (k,0) will be automatically added to the dictionary, as if the statement x[k]=0 is first executed.

for w in vocabulary: #looping across each word in library to calculate the word's idf

word\_idf[w] = idf(w, tweets) #taking the wordidf dictionaries' key as the word and putting value as the idf

return word\_idf

def tf\_idf(word, tweet, tweets, word\_idf):

if tweet != []:

return termfreq(word, tweet) \* word\_idf[word]

else:

return 0

def tweet\_tfidf\_features(tweets):

vocabulary = create\_vocabulary(tweets)

word\_idf = compute\_word\_idf(tweets, vocabulary)

tweet\_features = np.empty(len(tweets), dtype=object)

for i, tweet in enumerate(tweets):

a = np.zeros(len(vocabulary))

for j, word in enumerate(vocabulary):

a[j] = tf\_idf(word, tweet, tweets,word\_idf)

tweet\_features[i] = a

return np.vstack(tweet\_features)

feature\_matrix = tweet\_tfidf\_features(all\_tweets)

feature\_matrix.shape

#K-means noise removal

def euclidean\_vectorized(A, B):

n, d = A.shape

m, d1 = B.shape

assert d == d1, 'Incompatible shape'

A\_squared = np.sum(np.square(A), axis=1, keepdims=True)

B\_squared = np.sum(np.square(B), axis=1, keepdims=True)

AB = np.matmul(A, B.T)

distances = np.sqrt(A\_squared - 2 \* AB + B\_squared.T)

return distances

# X: data matrix of size (n\_samples,n\_features)

# n\_clusters: number of clusters

# output 1: labels of X with size (n\_samples,)

# output 2: centroids of clusters

def kmeans(X,n\_clusters):

# initialize labels and prev\_labels. prev\_labels will be compared with labels to check if the stopping condition

# have been reached.

prev\_labels = np.zeros(X.shape[0])

labels = np.zeros(X.shape[0])

# init random indices

# YOUR CODE GOES HERE

indices = np.random.choice(X.shape[0], n\_clusters, replace=False)# np.random.permutation(X.shape[0])[:n\_clusters]

# assign centroids using the indices

# YOUR CODE GOES HERE

centroids = X[indices]

# the interative algorithm goes here

while (True):

#we had some numerical differences here

centroids = np.nan\_to\_num(centroids)

# calculate the distances to the centroids

distances = sk.metrics.pairwise.cosine\_distances(X,centroids)

# assign labels

labels = np.argmin(distances,axis=1)

# stopping condition

if np.array\_equal(labels, prev\_labels):

#if np.sum(labels != prev\_labels) == 0:

break

# calculate new centroids

for cluster\_indx in range(centroids.shape[0]):

members = X[labels == cluster\_indx]

centroids[cluster\_indx,:] = np.mean(members,axis=0)

centroids = np.nan\_to\_num(centroids)

#plt.plot(labels, 'ro')

#plt.show()

# keep the labels for next round's usage

prev\_labels = np.argmin(distances,axis=1)

return labels,centroids

#K-menas consensus matrix

def consensus\_matrix\_kmeans(minclusters, maxclusters, features):

consensus\_matrix = np.zeros((features.shape[0],features.shape[0]))

for i in np.arange(minclusters, maxclusters, 1):

cluster\_labels, centroids = kmeans(features, i)

#sil\_score = silhouette\_score(features, cluster\_labels)

#print("silhouette score for k=",i, 'is:',sil\_score)

for i in range(len(cluster\_labels)):

for j in range(len(cluster\_labels)):

if cluster\_labels[i] == cluster\_labels[j]:

consensus\_matrix[i,j] = (consensus\_matrix[i,j] + 1)

#print(cluster\_labels[i],cluster\_labels[j],consensus\_matrix[i,j])

plt.imshow(consensus\_matrix)

plt.title('Consensus matrix')

plt.legend()

plt.show()

return consensus\_matrix

consensus\_matrix = consensus\_matrix\_kmeans(2,15, feature\_matrix)

for i in range(len(consensus\_matrix)):

for j in range(len(consensus\_matrix)):

if consensus\_matrix[i,j] <= 2:

consensus\_matrix[i,j] = 0

plt.imshow(consensus\_matrix)

plt.title('Consensus matrix k-Means after filtering')

plt.legend()

plt.show()

#Filter noise tweets

row\_mean = consensus\_matrix.mean(1)

matrix\_mean = consensus\_matrix.mean()

#create clean tweets that removes the noise points

kmeans\_filtered\_tweets = []

for i in range(len(all\_tweets)):

if row\_mean[i] >= matrix\_mean:

kmeans\_filtered\_tweets.append(all\_tweets[i])

consensus\_DBSCAN = np.zeros((feature\_matrix.shape[0],5))

minPts = 5

epsmin = 0.91

epsmax = 1.05

for i, epsilon in enumerate(np.arange(epsmin, epsmax, 0.03)):

print(i,", eps = ",epsilon)

db = DBSCAN(eps=epsilon,min\_samples=minPts)

db.fit(feature\_matrix)

labels = db.labels\_

core\_pts = db.core\_sample\_indices\_

n\_clusters\_ = len(set(labels)) - (1 if -1 in labels else 0)

print('%d clusters found' %(n\_clusters\_))

for k in range(0,feature\_matrix.shape[0]):

if k in core\_pts:

consensus\_DBSCAN[k][i] = 1.0 #dense

else:

consensus\_DBSCAN[k][i] = 0 #border or outlier

#MAtrix to filter out noise points

consensus\_DBSCAN\_results = np.zeros((consensus\_DBSCAN.shape[0],1))

for numb in range (0,consensus\_DBSCAN.shape[0]):

zeroes = 0 #border or outlier

ones = 0

for r in range(0,consensus\_DBSCAN.shape[1]):

if consensus\_DBSCAN[numb][r] == 1:

ones = ones+1

else:

zeroes = zeroes + 1

threshold = (consensus\_DBSCAN.shape[1]\*50)/100

if zeroes > threshold:

consensus\_DBSCAN\_results[numb][0] = 0 #noise

else:

consensus\_DBSCAN\_results[numb][0] = 1 #not noise

dbscan\_filtered\_tweets = []

for i in range(0, consensus\_DBSCAN.shape[0]):

if consensus\_DBSCAN\_results[i][0] == 1:

dbscan\_filtered\_tweets.append(all\_tweets[i])

kmeans\_filtered\_tweets\_idf = tweet\_tfidf\_features(kmeans\_filtered\_tweets)

dbscan\_filtered\_tweets\_idf = tweet\_tfidf\_features(dbscan\_filtered\_tweets)

print(kmeans\_filtered\_tweets\_idf.shape)

print(dbscan\_filtered\_tweets\_idf.shape)

#K means on 2 datasets

mink = 2

maxk = 12

consensus\_kmeans = consensus\_matrix\_kmeans(mink, maxk, dbscan\_filtered\_tweets\_idf)

consensus\_dbscan = consensus\_matrix\_kmeans(mink, maxk, kmeans\_filtered\_tweets\_idf)

#Final clustering result

labels\_kmeans, centroids\_kmeans = kmeans(consensus\_kmeans, 9)

labels\_dbscan, centroids\_dbscan = kmeans(consensus\_dbscan, 9)

def topten(tweetlist, centroids):

true\_k = 9

print("Top terms per cluster:")

order\_centroids = centroids.argsort()[:, ::-1]

terms = create\_vocabulary(tweetlist)

for i in range(true\_k):

print("Cluster %d:" % i)

for ind in order\_centroids[i, :10]:

print(' %s' % terms[int(ind)])

print

topten(dbscan\_filtered\_tweets, centroids\_dbscan)

topten(kmeans\_filtered\_tweets, centroids\_kmeans)

consensus\_kmeans.shape

matrix\_for\_gephi = consensus\_matrix\_kmeans(9, 10, kmeans\_filtered\_tweets\_idf)

print(matrix\_for\_gephi)

matrix\_for\_gephi.shape

nodesforgephy = np.arange(matrix\_for\_gephi.shape[0])

np.savetxt('/Users/sowmiyaaradha/anaconda3/envs/unsupervised/Tamil\_Corpus/Data/nodes1.csv', nodesforgephy, fmt='%.2f', delimiter=',')

#Generating edges file for gephy

newlist = list()

header1 = [['Source'],['Target'],['Type'],['Weight']]

newlist.append(header1)

for ctr in range(0,matrix\_for\_gephi.shape[0]):

for j in range(0,matrix\_for\_gephi.shape[1]):

inputforrow = list()

inputforrow.append(ctr)

inputforrow.append(j)

inputforrow.append('Directed')

inputforrow.append(abs(matrix\_for\_gephi[ctr][j]))

newlist.append(inputforrow)

with open("/Users/sowmiyaaradha/anaconda3/envs/unsupervised/Tamil\_Corpus/Data/edges1.csv", "w", newline="") as f:

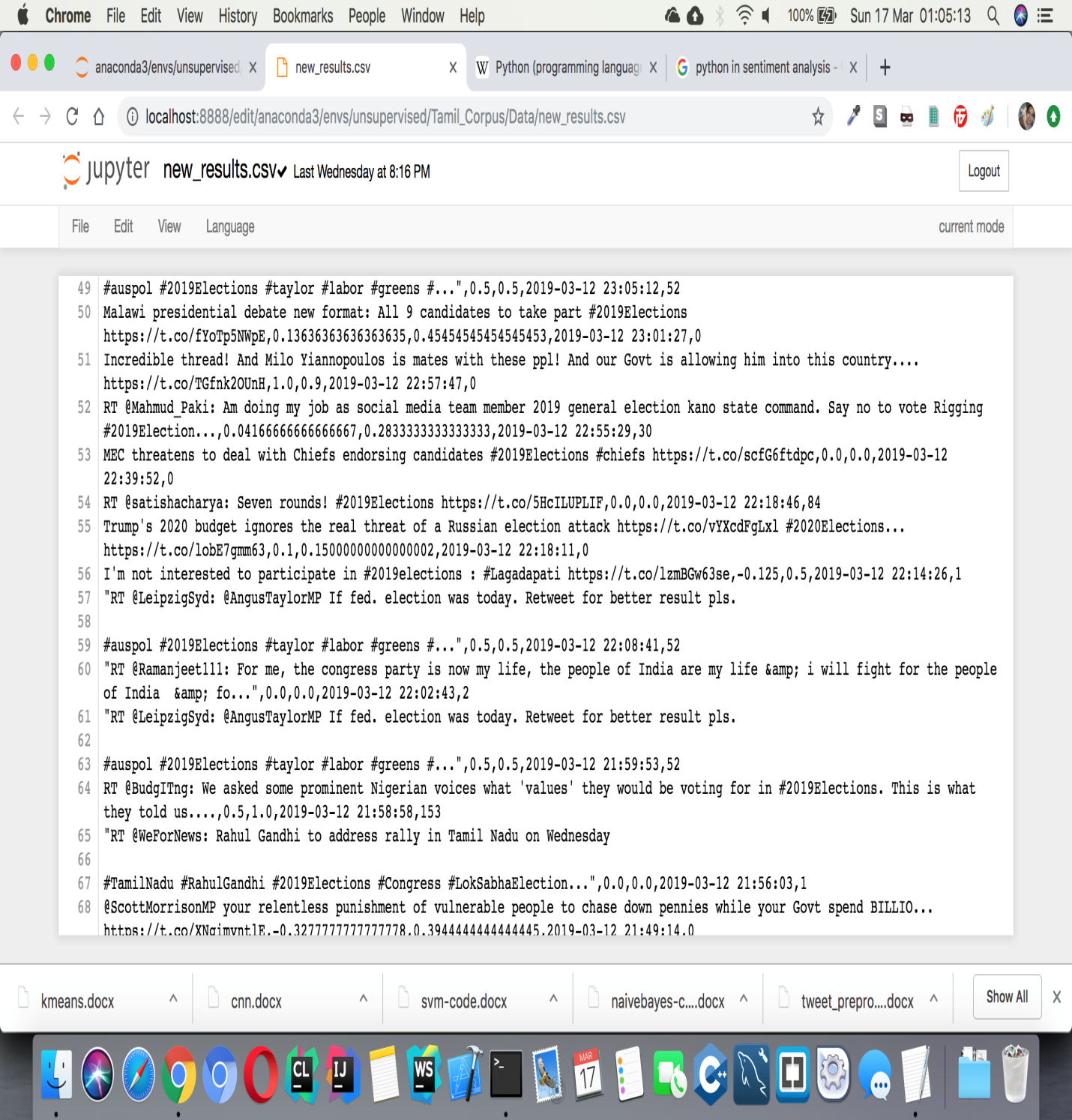
writer = csv.writer(f)

for row in newlist:

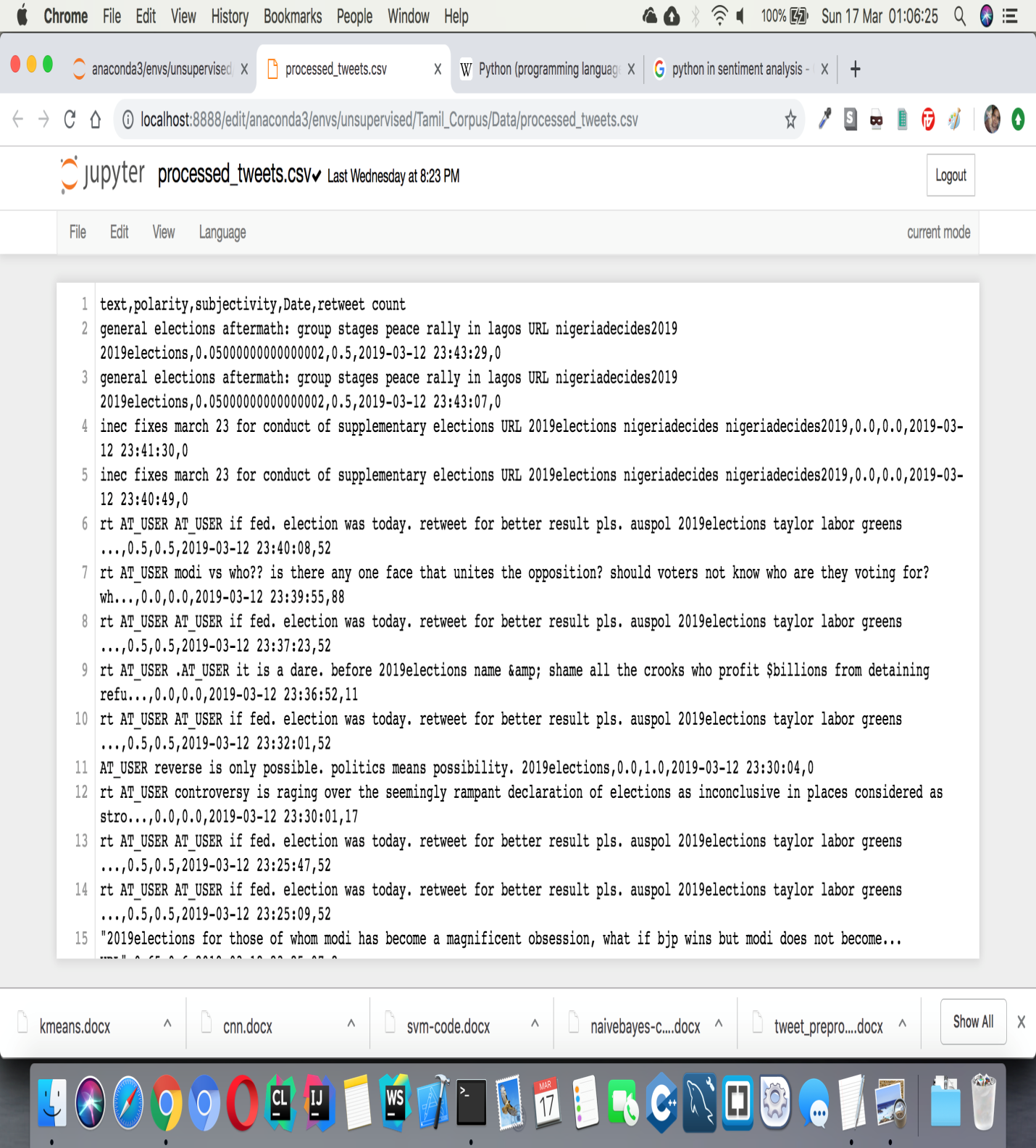
writer.writerow(row)

**9.2. APPENDIX – 2 (SCREENSHOTS)**

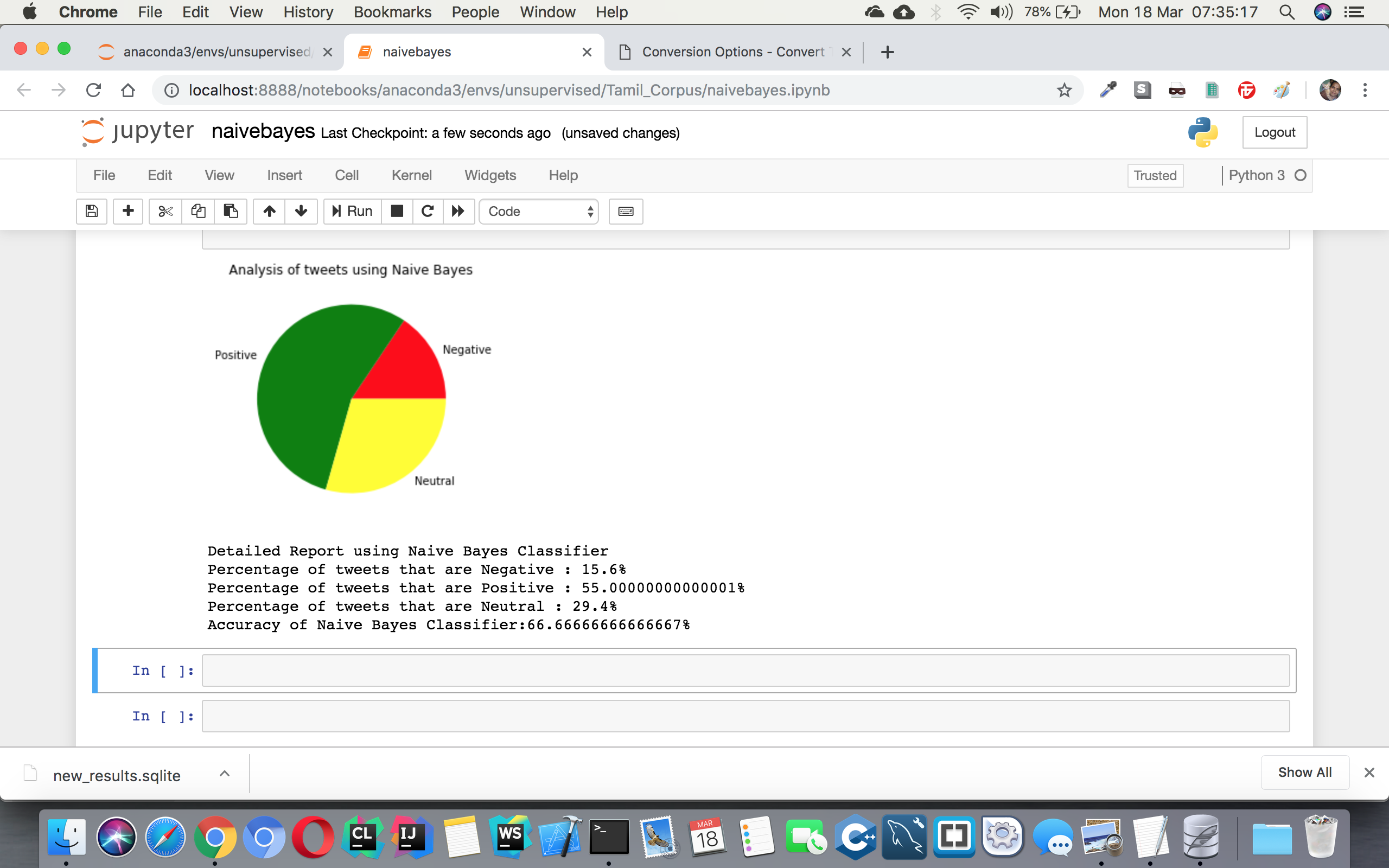
**TWEET EXTRACTION**

****

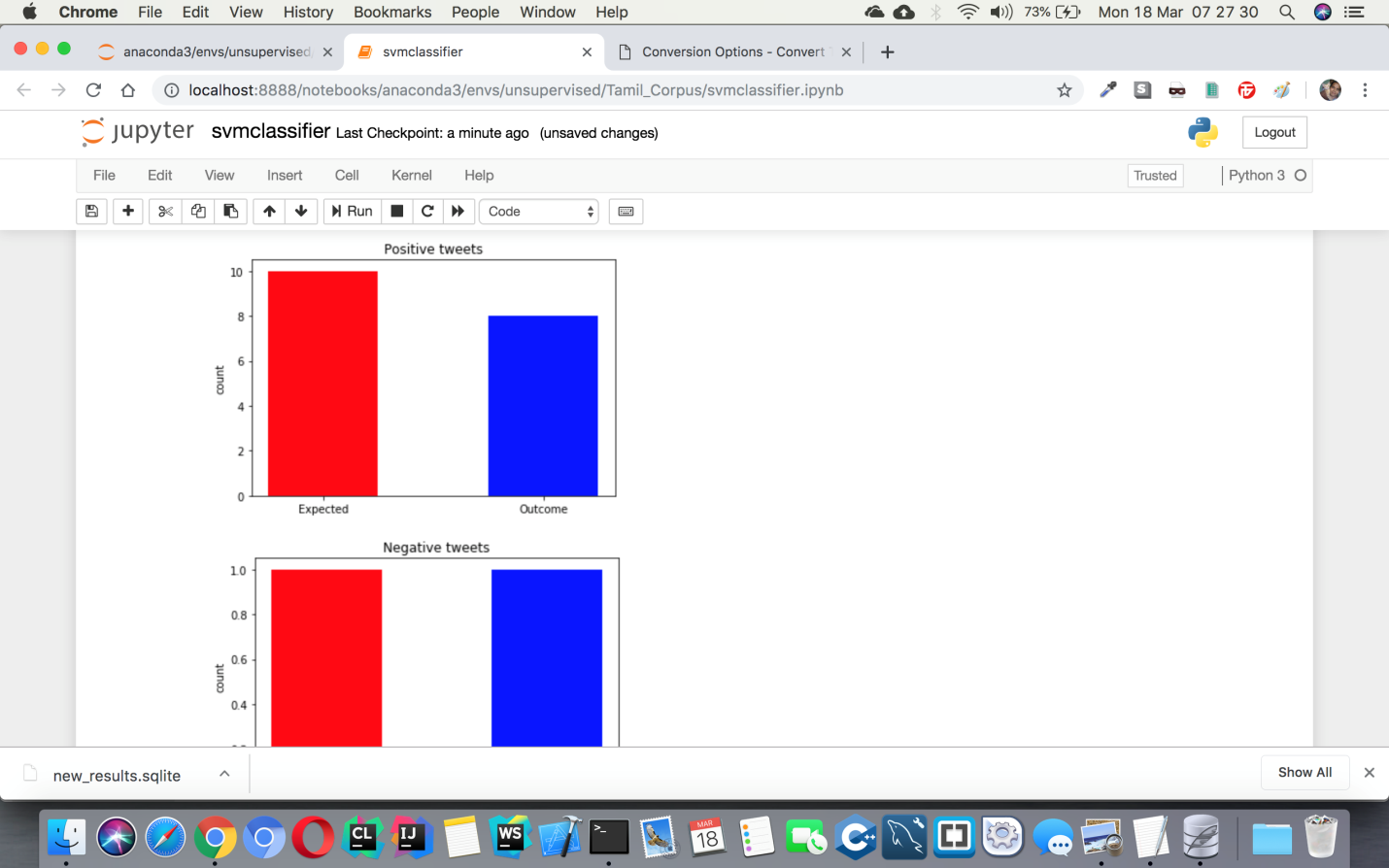
**PREPROCESSED TWEETS:**

****

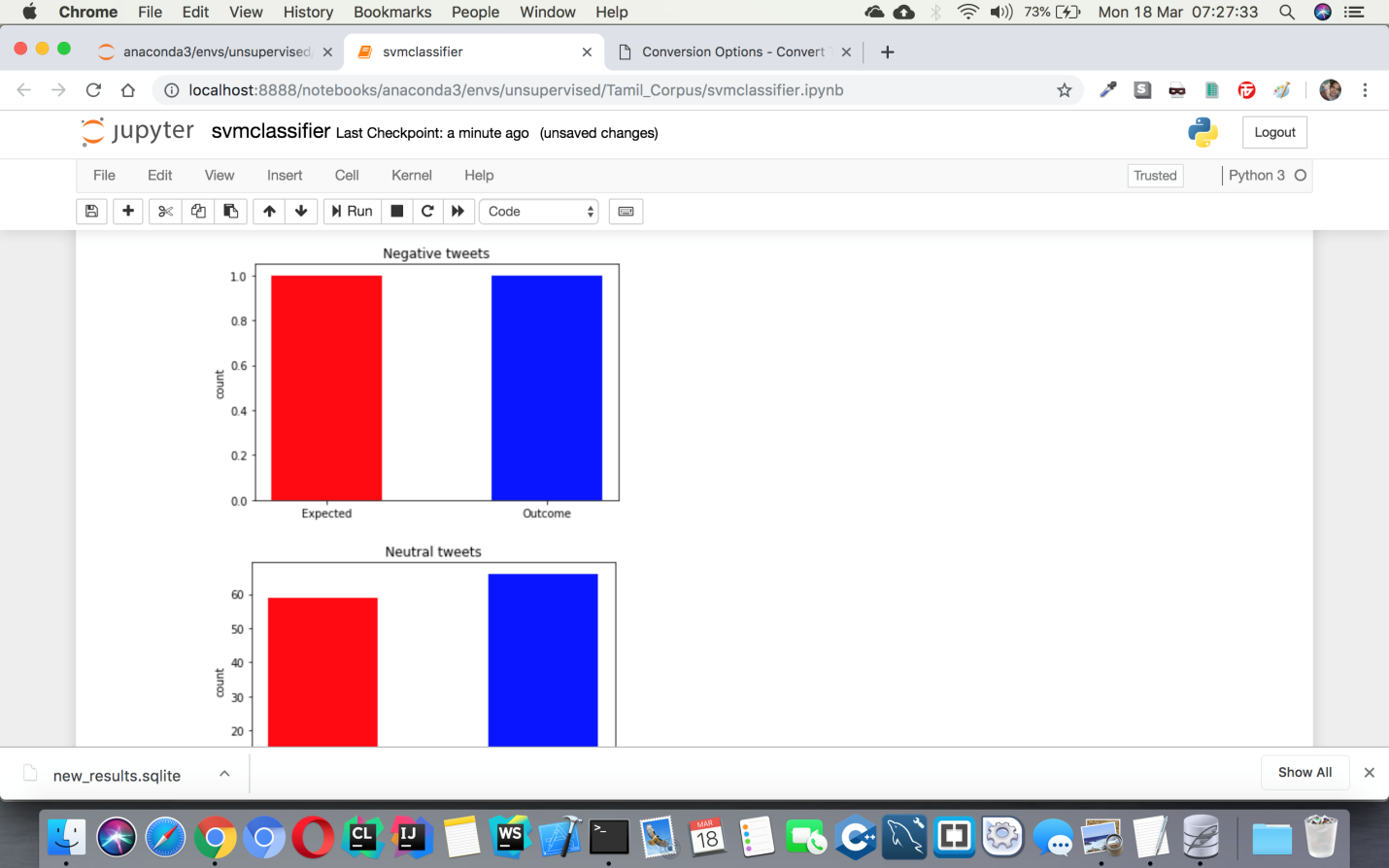
**NAÏVE BAYES ALGORITHM:**

****

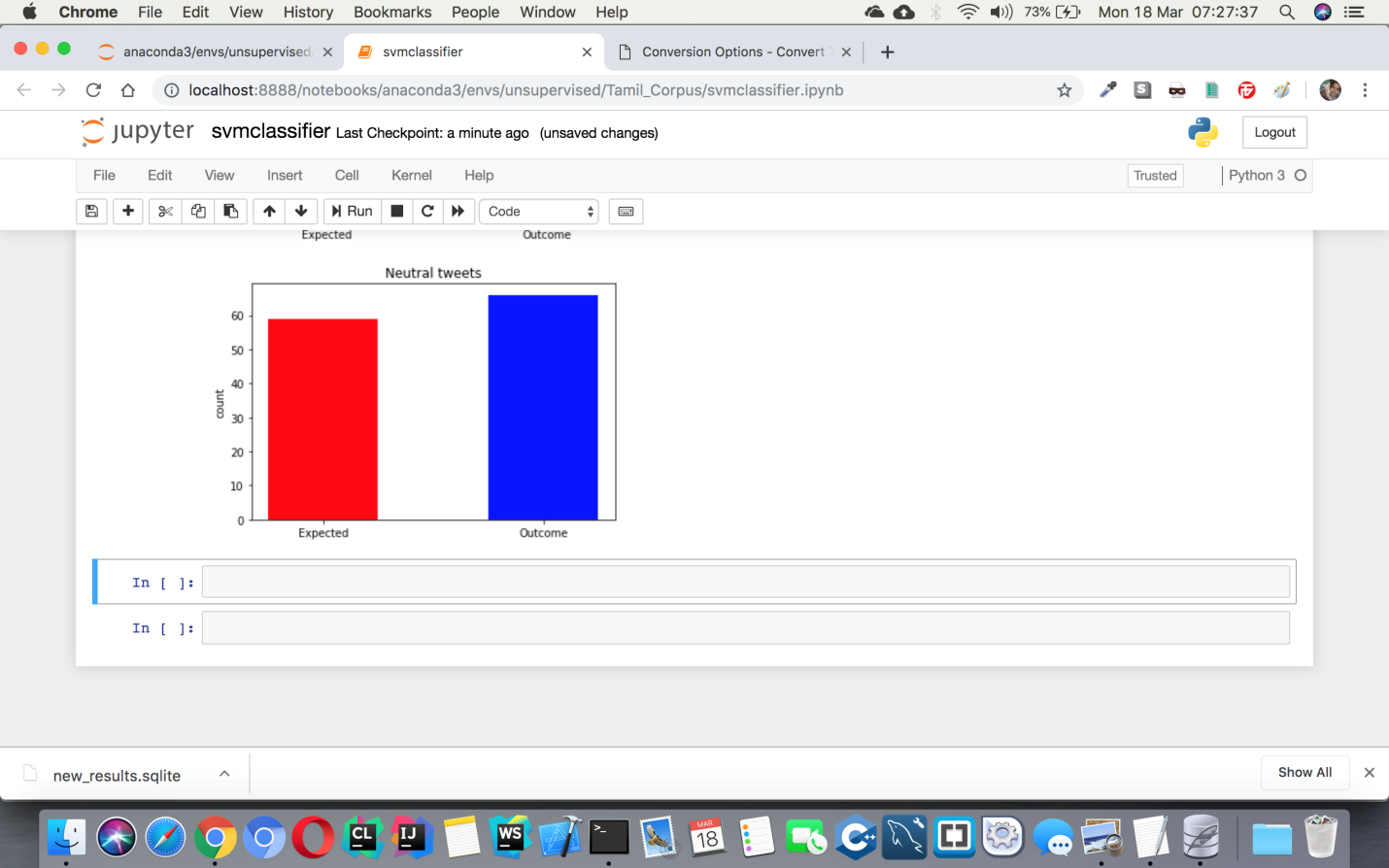
**SVM ALGORITHM: POSITIVE**

****

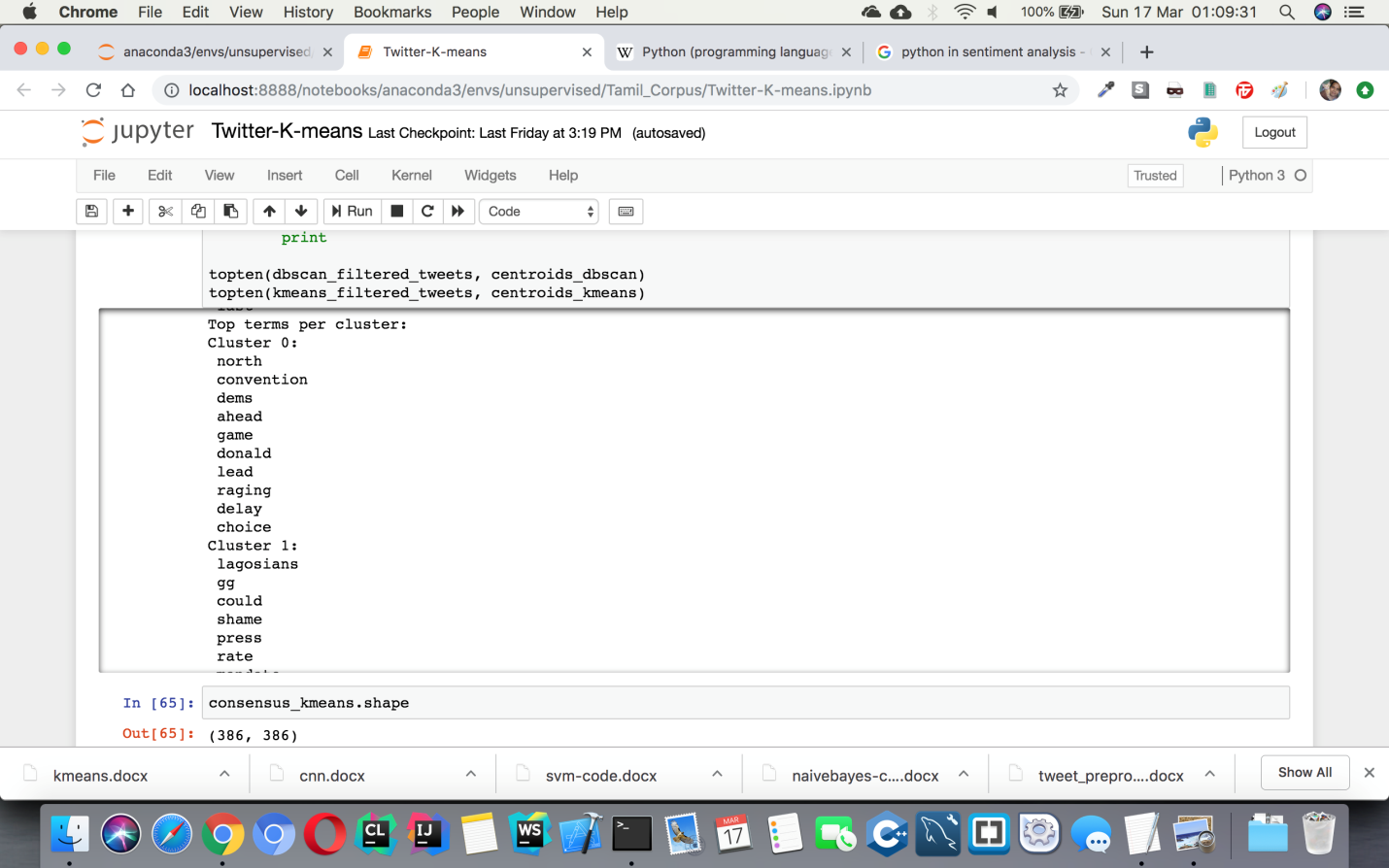
**SVM ALGORITHM: NEGATIVE**

****

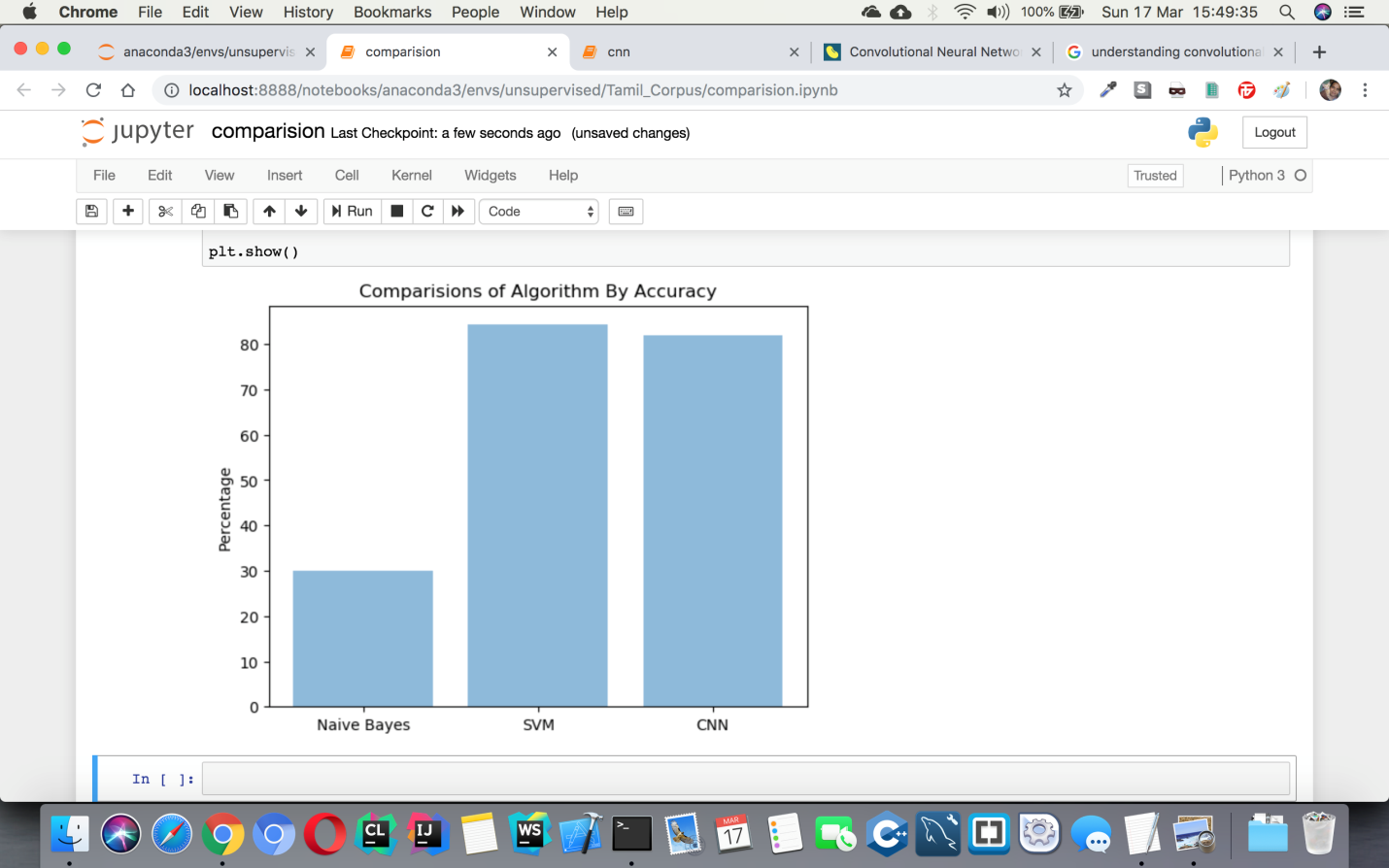
**SVM ALGORITHM: NEUTRAL**

****

**K-MEANS CLUSTERING ALGORITHM:**

****

**ALGORITHM COMPARISON:**

****

**CHAPTER - 10**

**REFERENCES**

* 1. S.J.Arunselvan, M.Anand Kumar, K.P.Soman, “Sentiment Analysis of Tamil Movie Reviews via feature frequency count”, International Journal of Applied Engineering Research, Vol.10, no.20, PP.17934-17939, 2015.
  2. Socker Richard et. al. “Recursive Deep Models for Semantic Compositionality over a Sentiment Treebank”, Proceedings of the Conference on empirical methods in natural language processing(EMNLP). Vol.1631, 2013.
  3. Pang Bo, Lec Lillian, “Opinion Mining and Sentiment Analysis”, Foundations and Trends ® in Information Retrieval, PP.2.1-2, 2008.
  4. Walaa Medhat, Ahmed Massan and Moda Korashy, “Sentiment Analysis Algorithms and Applications: A Survey”, Elsevier. 2014 May, 1093-1113.
  5. Liu and L.Zhang, “A Survey of Opinion Mining and Sentiment Analysis “, In Mining Text Data, Pages 415-463, Springer 2012.