SENTIMENT ANALYSIS USING K-NEAREST NEIGHBORS

A thesis in partial fulfilment for the degree of

Master of Computer Applications

Submitted by

Desitti Sagar

(16/CA/6005)

Under the supervision of

Dr. Suvamoy Changder

Assistant Professor

NIT Durgapur

Department of Computer Science and Engineering

National Institute of Technology Durgapur

May 2018

DECLARATION

I certify that

(a) The work contained in this report has been done by me under the guidance of my supervisor.

(b) The work has not been submitted to any other Institute for any degree or diploma.

(c) I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.

(d) Whenever I have used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references. Further, I have taken permission from the copyright owners of the sources, whenever necessary.

Date: May 2018 (Desitti Sagar)

Place: Durgapur (16/CA/6005)

Certificate Of Recommendation

This is to certify that the project report entitled “SENTIMENT ANALYSIS USING K-NN(K-NEAREST NEIGHBORS) ALGORITHM" submitted by Desitti Sagar (Roll No. 16/CA/6005) to National Institute of Technology Durgapur towards partial fulfilment of requirements for the award of degree of Master of Computer Applications in Computer Science and Engineering is a record of work carried out by him under my supervision and guidance during 2017-18.The results embodied in this thesis have not been submitted to any other University or Institute for the award of any degree or diploma. In our opinion, this thesis is of the standard required for the partial fulfilment of the requirements for the award of the degree of Master of Computer Applications.

(Counter Signed by)
Dr. Suvamoy Changder
Asst. Professor, Dept. of CSE
National Institute of Technology
Durgapur-713209, INDIA

(Counter Signed by)
Dr. Goutam Sanyal
Prof. & Head, Dept. of CSE
National Institute of Technology
Durgapur-713209, INDIA

Certificate of Approval

The foregoing thesis is hereby approved as a credible study of technological subject carried out and presented in a manner satisfactory to warrant its acceptance as a prerequisite with degree for which it has been submitted. It is to be understood that by this approval, the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn therein but approve the thesis only for the purpose for which it has been submitted.

BOARD OF THESIS EXAMINERS

1. ________________________ 2. _______________________

3. ________________________ 4. _______________________

Sentiment Analysis

K-Nearest Neighbors Approach

ABSTRACT

Millions of users share opinions on various topics using micro-blogging every day. Twitter is a very popular microblogging site where users are allowed a limit of 140 characters; this kind of restriction makes the users be concise as well as expressive at the same time. For that reason, it becomes a rich source for sentiment analysis and belief mining.

The aim of this project is to develop such a functional classifier which can correctly and automatically classify the sentiment of an unknown tweet. In our work, we propose techniques to classify the sentiment label accurately. We introduce one method: known as sentiment classification algorithm (SCA) based on k-nearest neighbors (KNN). We also evaluate their performance based on real tweets.

Table of Contents

1. Introduction

These days social networks, blogs, and other media produce a huge amount of data on the World Wide Web. This huge amount of data contains crucial opinion related information that can be used to benefit for businesses and other aspects of commercial and scientific industries. Manual tracking and extracting this useful information from this massive amount of data is almost impossible. Sentiment analysis of user posts is required to help taking business decisions. It is a process which extracts sentiments or opinions from reviews which are given by users over a subject, area or product in online.

We can categorize the sentiment into 10 types: **1) Anger 2) Enthusiasm 3) Fun 4) Happy 5) Hate 6) Love 7) Relief 8) Sadness 9) Surprise 10) Worry** that determine the general attitude of the people to a topic. Our principal goal is to correctly detect sentiment of tweets as more as possible. This project has one main part: to classify sentiment of tweets by using some feature. In KNN, we build a pair of tweets by using different features. From that pair, we measure the Euclidian distance for every tweet with its counterpart. From those distance we only consider nearest fifteen tweets label to classify that tweet.

1.1 Motivation

Being extremely interested in everything having a relation with the Machine Learning, the independent project was a great occasion to give me the time to learn and confirm my interest for this field. The fact that we can make estimations, predictions and give the ability for machines to learn by themselves is both powerful and limitless in term of application possibilities. We can use Machine Learning in Finance, Medicine, almost everywhere. That’s why I decided to conduct my project around the Machine Learning.

1.2 Objective

To investigate the different tweets of the user extracted from twitter and find the most accurate sentiment of the tweet.

2. The Project

Sentiment analysis, also refers as opinion mining, is a sub machine learning task where we want to determine which is the general sentiment of a given document. Using machine learning techniques and natural language processing we can extract the subjective information of a document and try to classify it according to its polarity such as positive, neutral or negative. It is a useful analysis since we could possibly determine the overall opinion about a selling object, or predict stock markets for a given company like, if most people think positive about it, possibly its stock markets will increase, and so on. Sentiment analysis is far from to be solved since the language is very complex (objectivity/subjectivity, negation, vocabulary, grammar, etc.) but it is also why it is very interesting to working on.

In this project I choose to try to classify tweets from Twitter into 10 different sentiments. Twitter is a microblogging website where people can share their feelings quickly and spontaneously by sending tweets limited by 140 characters. You can directly address a tweet to someone by adding the target sign “@” or participate to a topic by adding a hashtag “#” to your tweet. Because of the usage of Twitter, it is a perfect source of data to determine the current overall opinion about anything.

2.1 Data

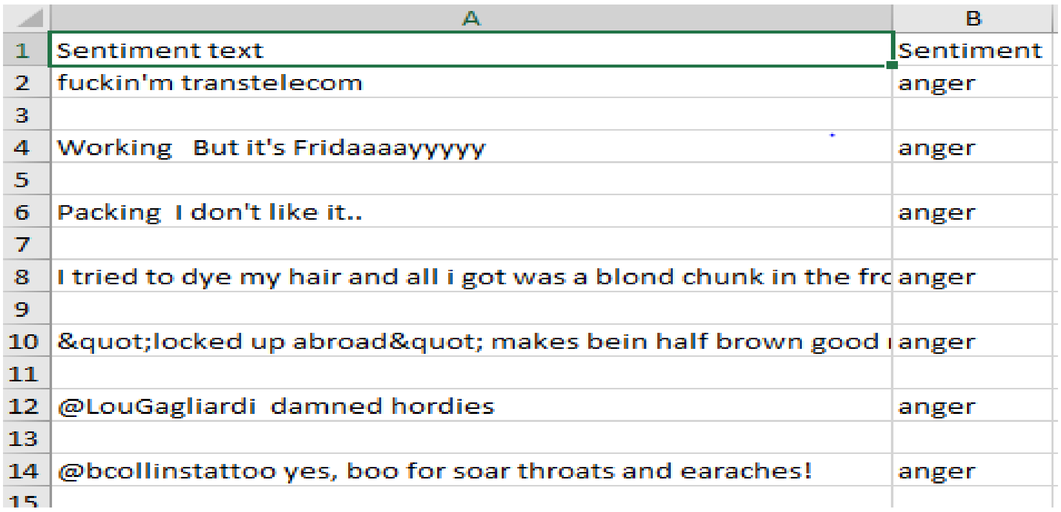
To gather the data many options are possible. In some previous paper researches, they built a program to collect automatically a corpus of tweets based on two classes, “positive” and “negative”, by querying Twitter with two type of emoticons:

- Happy emoticons, such as “:)”, “:P”, “:)” etc.
- Sad emoticons, such as “:(“, “:’(”, “=(“.

Others make their own dataset of tweets my collecting and annotating them manually which very long and fastidious.

Additionally, to find a way of getting a corpus of tweets, we need to take of having a balanced data set, meaning we should have an equal number of anger, enthusiasm, fun, happy, hate, love, relief, sadness, surprise and worry tweets, but it needs also to be large enough. Indeed, more the data we have, more we can train our classifier and more the accuracy will be.

After many researches, I found a dataset of 27532 tweets in english coming from two sources: Kaggle, T4SA Dataset, SMILE twitter emotion dataset and Sentiment140. It is composed of two columns that are SentimentText and Sentiment. We are only interested by the Sentiment column and the SentimentText columns containing the tweets in a raw format.



	A	B
1	Sentiment text	Sentiment
2	fuckin'm transtelecom	anger
3		
4	Working But it's Fridaaaayyyyy	anger
5		
6	Packing I don't like it..	anger
7		
8	I tried to dye my hair and all i got was a blond chunk in the frc	anger
9		
10	&quot;locked up abroad&quot; makes bein half brown good	anger
11		
12	@LouGagliardi damned hordies	anger
13		
14	@bcollinstattoo yes, boo for soar throats and earaches!	anger
15		

Fig 2.1: Example of twitter posts annotated with their corresponding sentiment

In the Fig 2.1 showing the first seven twitter posts we can already notice some particularities and difficulties that we are going to encounter during the pre-processing steps.

We can visualize a bit more the dataset by making a chart of how tweets do it contains.

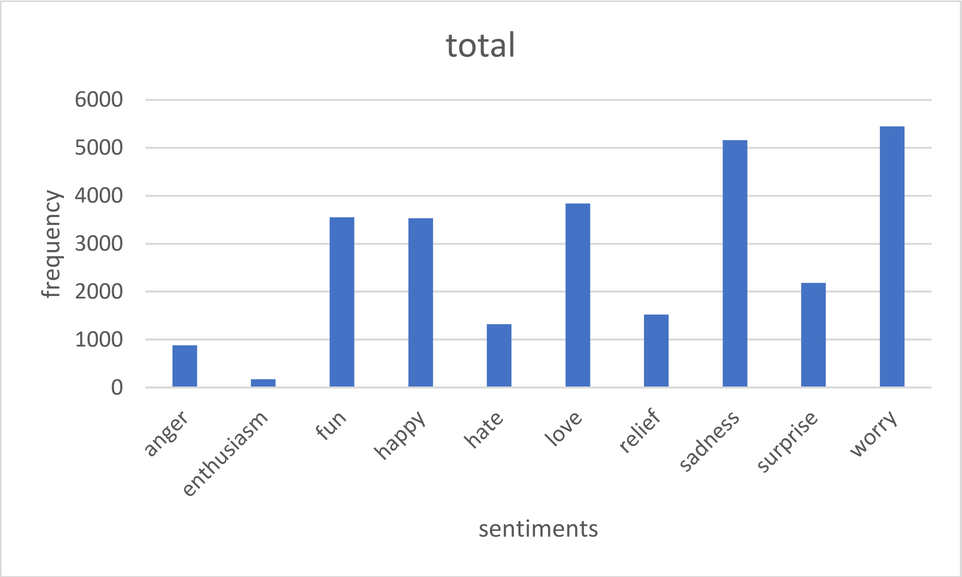


Fig 2.2: Histogram of the tweets according to their sentiments.

Finally, let's recall the Twitter terminology since we are going to have to deal with in the tweets:

- **Hashtag**: A hashtag is any word or phrase immediately preceded by the # symbol. When you click on a hashtag, you'll see other Tweets containing the same keyword or topic.
- **@username**: A username is how you're identified on Twitter and is always preceded immediately by the @ symbol. For instance, Katy Perry is @katyperry.
- **MT**: Like RT (Retweet), an abbreviation for "Modified Tweet." Placed before the Retweeted text when users manually retweet a message with modifications, for example shortening a Tweet.
- **Retweet**: RT, A Tweet that you forward to your followers is known as a Retweet. Often used to pass along news or other valuable discoveries on Twitter, Retweets always retain original attribution.
- **Emoticons**: Composed using punctuation and letters, they are used to express emotions concisely, ";) :) ...".

Now we have the corpus of tweets, we need to use other resources to make easier the pre-processing step.

2.2 Resources

In order to facilitate the pre-processing part of the data, we introduce five resources which are,

- An **emoticon dictionary** regrouping 132 of the most used emoticons in western with their sentiment, negative or positive.
- An acronym dictionary of 5465 acronyms with their translation.
- A **stop word dictionary** corresponding to words which are filtered out before or after processing of natural language data because they are not useful in our case.
- A negative contractions and auxiliaries dictionary which will be used to detect negation in a given tweet such as “don’t”, “can’t”, “cannot”, etc.

The introduction of these resources will allow to uniform tweets and remove some of their complexities with the acronym dictionary for instance because a lot of acronyms are used in tweets. The positive and negative word dictionaries could be useful to increase (or not) the accuracy score of the classifier. The emoticon dictionary has been built from Wikipedia with each emoticon annotated manually. The stop word dictionary contains 635 words such as “the”, “of”, “without”. Normally they should not be useful for classifying tweets according to their sentiment, but it is possible that they are.

Also, we use Python 2.7 (https://www.python.org/) which is a programming language widely used in data science and scikitlearn (http://scikitlearn.org/) a very complete and useful library for machine learning containing every technique, methods we need and the website is also full of tutorials well explained. With Python, the libraries, Numpy (http://www.numpy.org/) and Panda (http://pandas.pydata.org/) for manipulating data easily and intuitively are just essential.

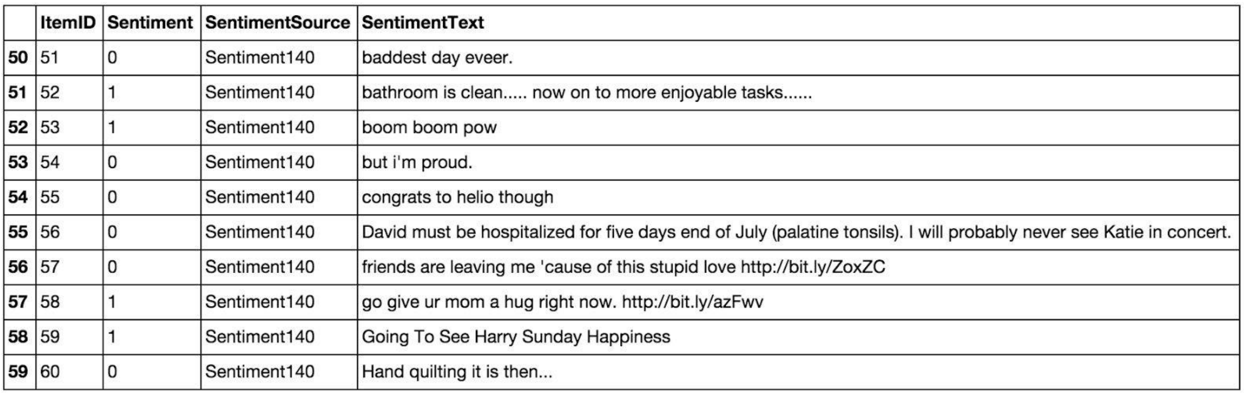
2.3 Pre-processing

Now that we have the corpus of tweets and all the resources that could be useful, we can pre-process the tweets. It is a very important since all the modifications that we are going to during this process will directly impact the classifier’s performance. The pre-processing includes cleaning, normalization, transformation, feature extraction and selection, etc. The result of pre-processing will be consistent and uniform data that are workable to maximize the classifier's performance.

All the tweets are pre-processed by passing through the following steps in the same order.

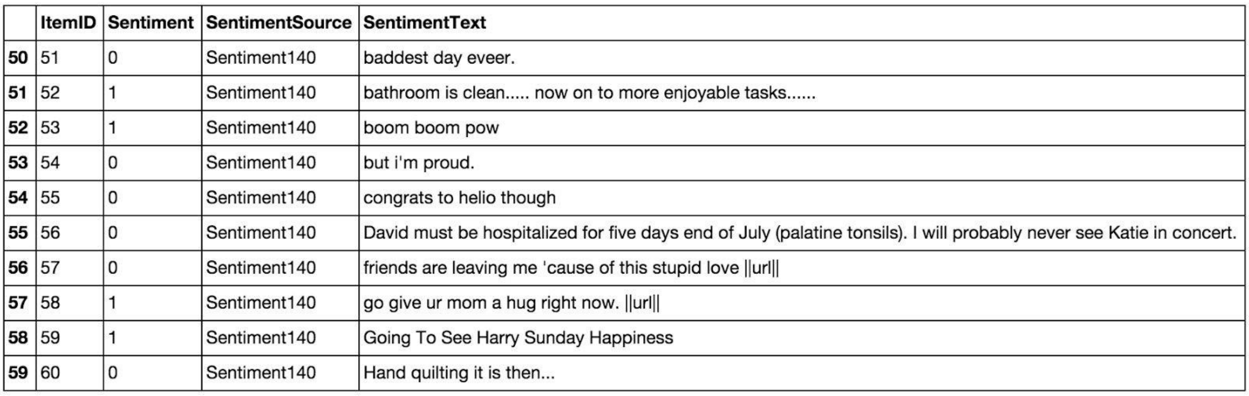
2.3.1 Urls

We replace all URLs with the tag ||**url**||. To do the replacement, we pass through each tweet and by using a regex we find out if it contains url, if yes, they are replaced by ‘||**url**||’.



	ItemID	Sentiment	SentimentSource	SentimentText
50	51	0	Sentiment140	baddest day eveer.
51	52	1	Sentiment140	bathroom is clean..... now on to more enjoyable tasks......
52	53	1	Sentiment140	boom boom pow
53	54	0	Sentiment140	but i'm proud.
54	55	0	Sentiment140	congrats to helio though
55	56	0	Sentiment140	David must be hospitalized for five days end of July (palatine tonsils). I will probably never see Katie in concert.
56	57	0	Sentiment140	friends are leaving me 'cause of this stupid love http://bit.ly/ZoxZC
57	58	1	Sentiment140	go give ur mom a hug right now. http://bit.ly/azFwv
58	59	1	Sentiment140	Going To See Harry Sunday Happiness
59	60	0	Sentiment140	Hand quilting it is then...

Fig 2.3: Tweets before processing URLs.

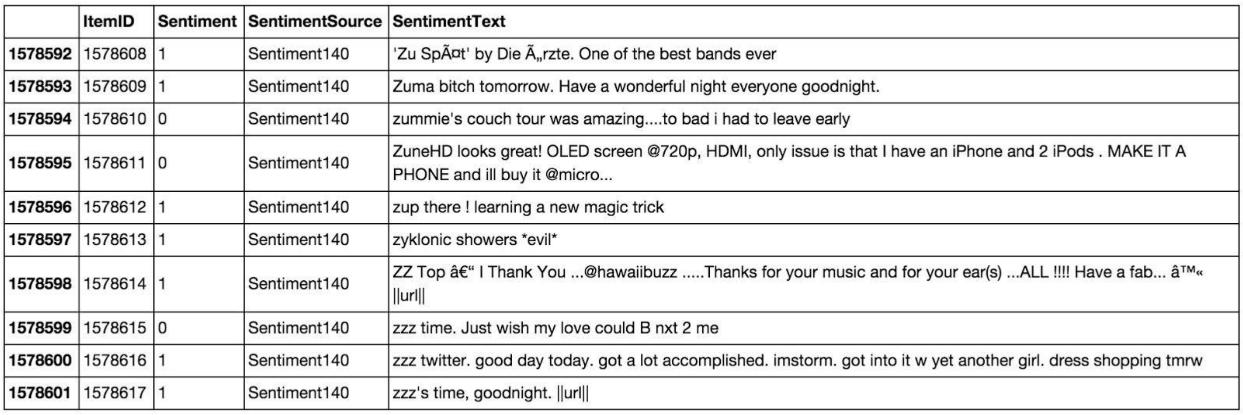


	ItemID	Sentiment	SentimentSource	SentimentText
50	51	0	Sentiment140	baddest day eveer.
51	52	1	Sentiment140	bathroom is clean..... now on to more enjoyable tasks......
52	53	1	Sentiment140	boom boom pow
53	54	0	Sentiment140	but i'm proud.
54	55	0	Sentiment140	congrats to helio though
55	56	0	Sentiment140	David must be hospitalized for five days end of July (palatine tonsils). I will probably never see Katie in concert.
56	57	0	Sentiment140	friends are leaving me 'cause of this stupid love ||url||
57	58	1	Sentiment140	go give ur mom a hug right now. ||url||
58	59	1	Sentiment140	Going To See Harry Sunday Happiness
59	60	0	Sentiment140	Hand quilting it is then...

Fig 2.4: Tweets after processing URLs.

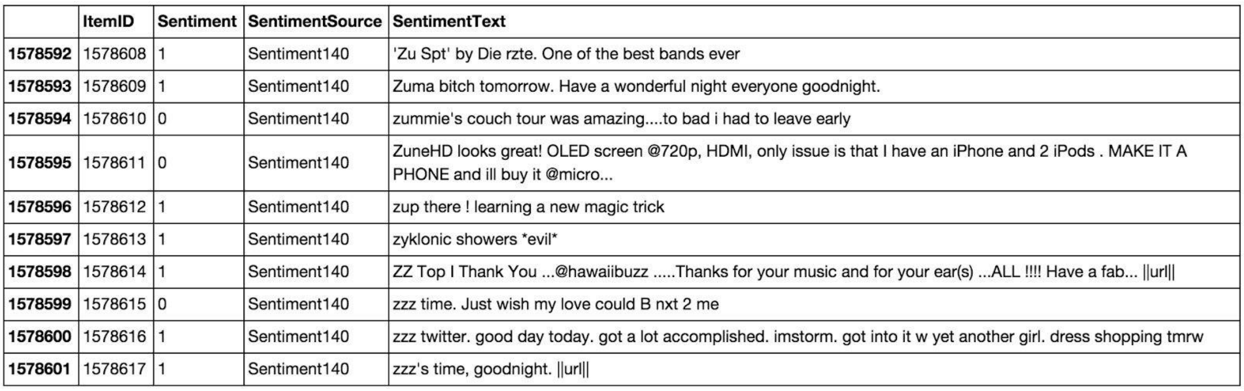
2.3.2 Unicode

For simplicity and because the ASCII table should be enough, we choose to remove any Unicode character that could be misleading for the classifier.



	ItemID	Sentiment	SentimentSource	SentimentText
1578592	1578608	1	Sentiment140	'Zu SpÃ¤t' by Die Ã„rzte. One of the best bands ever
1578593	1578609	1	Sentiment140	Zuma bitch tomorrow. Have a wonderful night everyone goodnight.
1578594	1578610	0	Sentiment140	zummie's couch tour was amazing....to bad i had to leave early
1578595	1578611	0	Sentiment140	ZuneHD looks great! OLED screen @720p, HDMI, only issue is that I have an iPhone and 2 iPods . MAKE IT A PHONE and ill buy it @micro...
1578596	1578612	1	Sentiment140	zup there ! learning a new magic trick
1578597	1578613	1	Sentiment140	zyklonic showers *evil*
1578598	1578614	1	Sentiment140	ZZ Top â€“ I Thank You ...@hawaiibuzz .....Thanks for your music and for your ear(s) ...ALL !!!! Have a fab... â™« ||url||
1578599	1578615	0	Sentiment140	zzz time. Just wish my love could B nxt 2 me
1578600	1578616	1	Sentiment140	zzz twitter. good day today. got a lot accomplished. imstorm. got into it w yet another girl. dress shopping tmrw
1578601	1578617	1	Sentiment140	zzz's time, goodnight. ||url||

Fig 2.5: Tweets before processing Unicode.

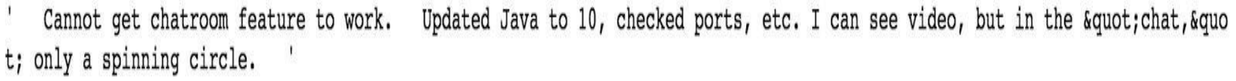


	ItemID	Sentiment	SentimentSource	SentimentText
1578592	1578608	1	Sentiment140	'Zu Spt' by Die rzte. One of the best bands ever
1578593	1578609	1	Sentiment140	Zuma bitch tomorrow. Have a wonderful night everyone goodnight.
1578594	1578610	0	Sentiment140	zummie's couch tour was amazing....to bad i had to leave early
1578595	1578611	0	Sentiment140	ZuneHD looks great! OLED screen @720p, HDMI, only issue is that I have an iPhone and 2 iPods . MAKE IT A PHONE and ill buy it @micro...
1578596	1578612	1	Sentiment140	zup there ! learning a new magic trick
1578597	1578613	1	Sentiment140	zyklonic showers *evil*
1578598	1578614	1	Sentiment140	ZZ Top I Thank You ...@hawaiibuzz .....Thanks for your music and for your ear(s) ...ALL !!!! Have a fab... ||url||
1578599	1578615	0	Sentiment140	zzz time. Just wish my love could B nxt 2 me
1578600	1578616	1	Sentiment140	zzz twitter. good day today. got a lot accomplished. imstorm. got into it w yet another girl. dress shopping tmrw
1578601	1578617	1	Sentiment140	zzz's time, goodnight. ||url||

Fig 2.6: Tweets after processing Unicode.

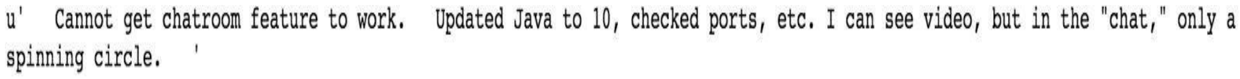
2.3.3 HTML Entities

HTML entities are characters reserved in HTML. We need to decode them in order to have characters entities to make them understandable.



```
'   Cannot get chatroom feature to work.   Updated Java to 10, checked ports, etc. I can see video, but in the &quot;chat,&quo
t; only a spinning circle.   '
```

Fig 2.7: A tweet before processing HTML entities.

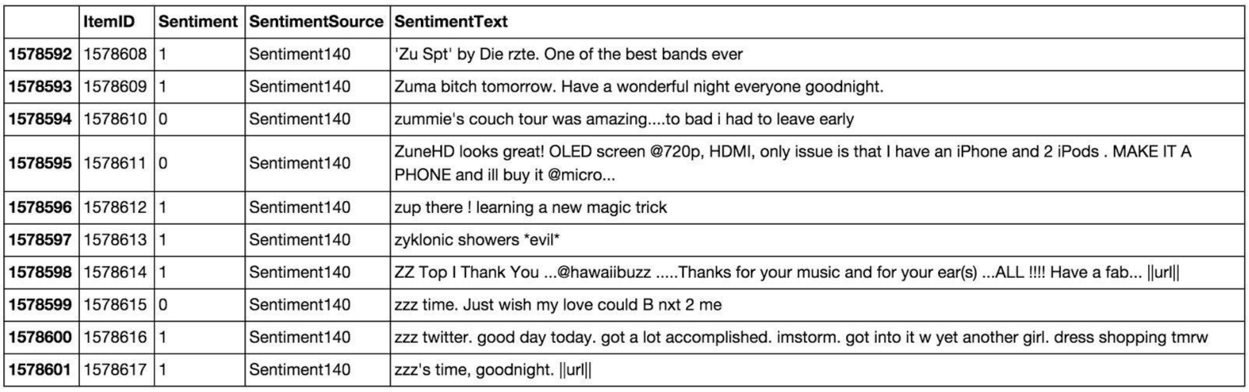


```
u'   Cannot get chatroom feature to work.   Updated Java to 10, checked ports, etc. I can see video, but in the "chat," only a
spinning circle.   '
```

Fig 2.7: A tweet after processing HTML entities.

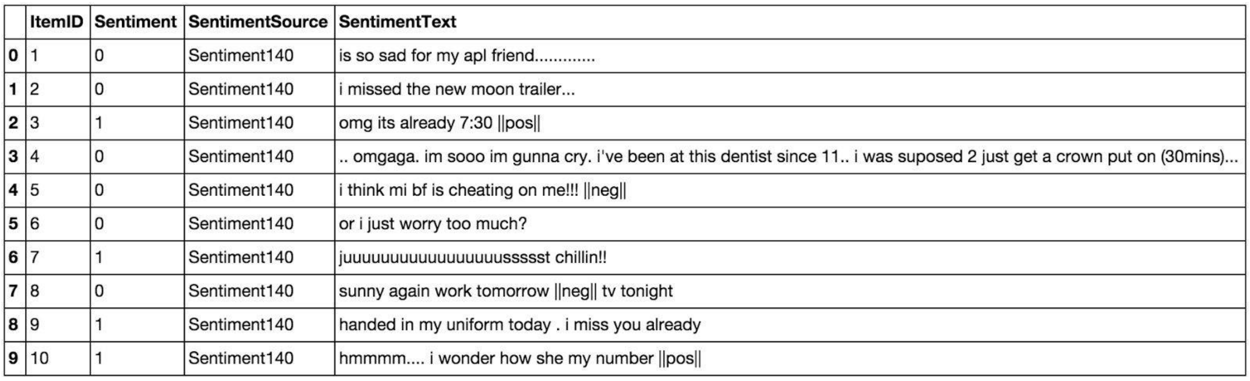
2.3.4 Case

The case is something that can appears useless but in fact it is really important for distinguish proper noun and other kind of words. Indeed: “General Motor” is the same thing that “general motor”, or “MSc” and “msc”. So reduce all letters to lowercase should be normally done wisely. In this project, for simplicity we will not take care of that since we assume that it should not impact too much the classifier’s performance.



	ItemID	Sentiment	SentimentSource	SentimentText
1578592	1578608	1	Sentiment140	'Zu Spt' by Die rzte. One of the best bands ever
1578593	1578609	1	Sentiment140	Zuma bitch tomorrow. Have a wonderful night everyone goodnight.
1578594	1578610	0	Sentiment140	zummie's couch tour was amazing....to bad i had to leave early
1578595	1578611	0	Sentiment140	ZuneHD looks great! OLED screen @720p, HDMI, only issue is that I have an iPhone and 2 iPods . MAKE IT A PHONE and ill buy it @micro...
1578596	1578612	1	Sentiment140	zup there ! learning a new magic trick
1578597	1578613	1	Sentiment140	zyklonic showers *evil*
1578598	1578614	1	Sentiment140	ZZ Top I Thank You ...@hawaiibuzz .....Thanks for your music and for your ear(s) ...ALL !!!! Have a fab... ||url||
1578599	1578615	0	Sentiment140	zzz time. Just wish my love could B nxt 2 me
1578600	1578616	1	Sentiment140	zzz twitter. good day today. got a lot accomplished. imstorm. got into it w yet another girl. dress shopping tmrw
1578601	1578617	1	Sentiment140	zzz's time, goodnight. ||url||

Fig 2.8: Tweets before processing lowercase.



	ItemID	Sentiment	SentimentSource	SentimentText
0	1	0	Sentiment140	is so sad for my apl friend.............
1	2	0	Sentiment140	i missed the new moon trailer...
2	3	1	Sentiment140	omg its already 7:30 ||pos||
3	4	0	Sentiment140	.. omgaga. im sooo im gunna cry. i've been at this dentist since 11.. i was suposed 2 just get a crown put on (30mins)...
4	5	0	Sentiment140	i think mi bf is cheating on me!!! ||neg||
5	6	0	Sentiment140	or i just worry too much?
6	7	1	Sentiment140	juuuuuuuuuuuuuuuuussssst chillin!!
7	8	0	Sentiment140	sunny again work tomorrow ||neg|| tv tonight
8	9	1	Sentiment140	handed in my uniform today . i miss you already
9	10	1	Sentiment140	hmmmm.... i wonder how she my number ||pos||

Fig 2.9: Tweets after processing lowercase.

2.4 Machine Learning

Once we have applied the different steps of the pre-processing part, we can now focus on the machine learning part. Here, we use only one algorithm i.e.; knn(K-Nearest Neighbors). K-Nearest Neighbor(KNN) is the supervised data mining pattern recognition algorithm. It classifies objects based on closed training exam-nearest neighbour algorithm. It is amongst the simplest of all machine learning algorithms. An object is classified by a majority vote of its neighbors. K is small positive integer and it is usually previously set.

Let's first introduce the K-Nearest Neighbor(KNN) model which is well known for its simplicity.

2.4.1 KNN (K-Nearest Neighbors)

KNN is a non-parametric and lazy learning algorithm. Non-parametric means there is no assumption for underlying data distribution. In other words, the model structure determined from the dataset. This will be very helpful in practice where most of the real world datasets do not follow mathematical theoretical assumptions. Lazy algorithm means it does not need any training data points for model generation. All training data used in the testing phase. This makes training faster and testing phase slower and costlier. Costly testing phase means time and memory. In the worst case, KNN needs more time to scan all data points and scanning all data points will require more memory for storing training data.

How does the KNN algorithm work?

In KNN, K is the number of nearest neighbors. The number of neighbors is the core deciding factor. K is generally an odd number if the number of classes is 2. When K=1, then the algorithm is known as the nearest neighbors algorithm. This is the simplest case. Suppose P1 is the point, for which label needs to predict. First, you find the one closest point to P1 and then the label of the nearest point assigned to P1.

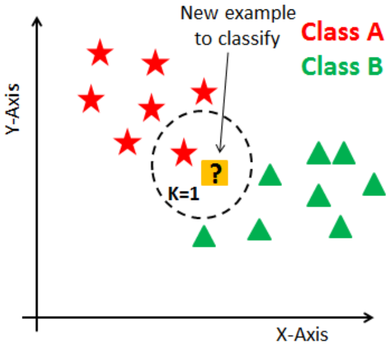


Fig 2.10: k-nn example

Suppose P1 is the point, for which label needs to predict. First, you find the k closest point to P1 and then classify points by majority vote of its k neighbors. Each object votes for their class and the class with the most votes is taken as the prediction. For finding closest similar points, you find the distance between points using distance measures such as Euclidean distance, Hamming distance, Manhattan distance and Minkowski distance. KNN has the following basic steps:

1. Calculate distance
2. Find closest neighbors
3. Vote for labels

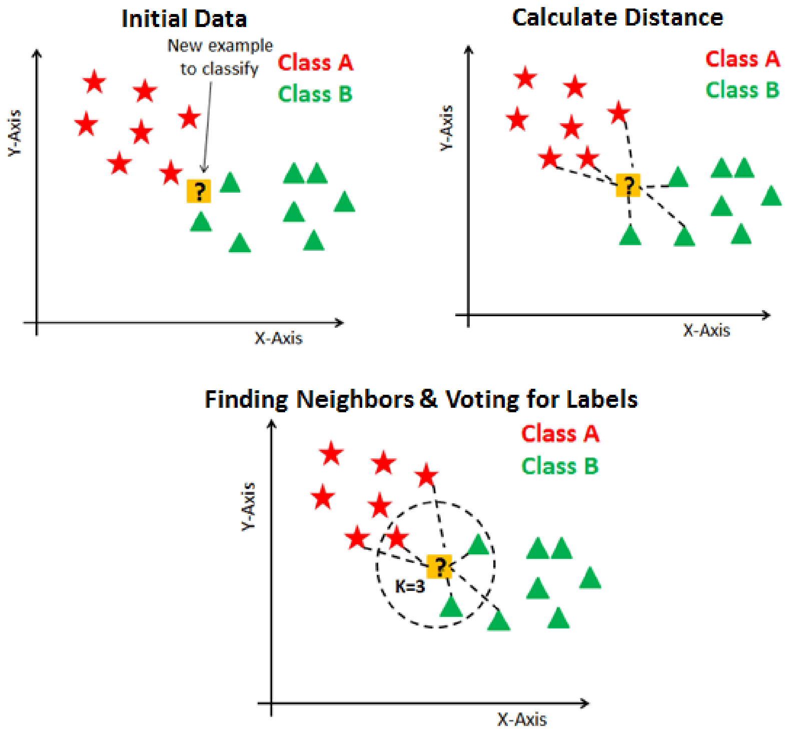


Fig 2.11: finding neighbors in k-nn

Eager Vs. Lazy Learners

Eager learners mean when given training points will construct a generalized model before performing prediction on given new points to classify. You can think of such learners as being ready, active and eager to classify unobserved data points.

Lazy Learning means there is no need for learning or training of the model and all the data points used at the time of prediction. Lazy learners wait until the last minute before classifying any data point. Lazy learner stores merely the training dataset and waits until classification needs to perform. Only when it sees the test tuple does it perform generalization to classify the tuple based on its similarity to the stored training tuples. Unlike eager learning methods, lazy learners do less work in the training phase and more work in the testing phase to make a classification. Lazy learners are also known as instance-based learners because lazy learners store the training points or instances, and all learning is based on instances.

Curse of Dimensionality

KNN performs better with a lower number of features than many features. You can say that when the number of features increases than it requires more data. Increase in dimension also leads to the problem of overfitting. To avoid overfitting, the needed data will need to grow exponentially as you increase the number of dimensions. This problem of higher dimension is known as the Curse of Dimensionality.

To deal with the problem of the curse of dimensionality, you need to perform principal component analysis before applying any machine learning algorithm, or you can also use feature selection approach. Research has shown that in large dimension Euclidean distance is not useful anymore. Therefore, you can prefer other measures such as cosine similarity, which get decidedly less affected by high dimension.

How do you decide the number of neighbors in KNN?

Now, you understand the KNN algorithm working mechanism. At this point, the question arises that How to choose the optimal number of neighbors? And what are its effects on the classifier? The number of neighbors(K) in KNN is a hyperparameter that you need choose at the time of model building. You can think of K as a controlling variable for the prediction model.

Research has shown that no optimal number of neighbors suits all kind of data sets. Each dataset has its own requirements. In the case of a small number of neighbors, the noise will have a higher influence on the result, and many neighbors make it computationally expensive. Research has also shown that a small amount of neighbors are most flexible fit which will have low bias but high variance and many neighbors will have a smoother decision boundary which means lower variance but higher bias.

Generally, Data scientists choose as an odd number if the number of classes is even. You can also check by generating the model on different values of k and check their performance. You can also try Elbow method here.

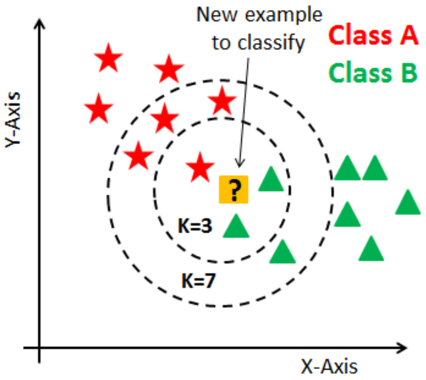
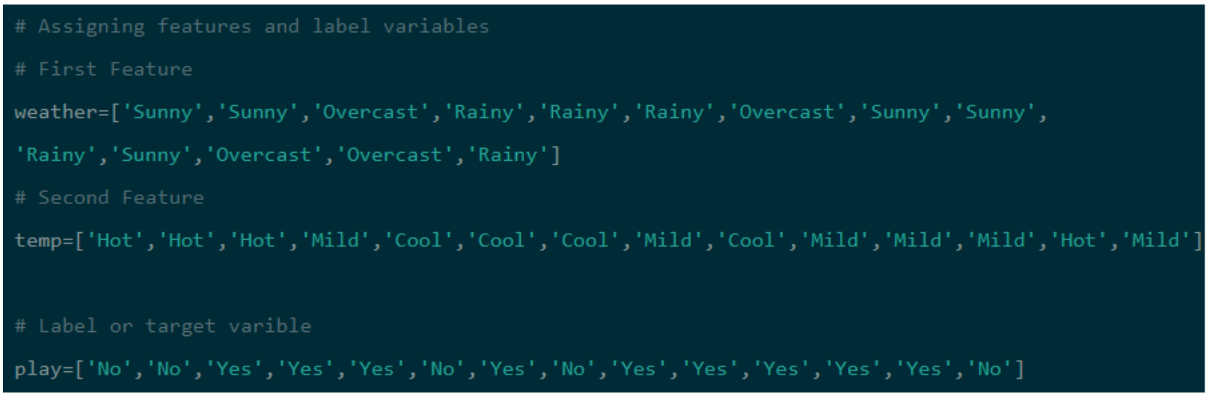


Fig 2.12: choosing k value in k-nn

KNN Classifier

Defining dataset

Let's first create your own dataset. Here you need two kinds of attributes or columns in your data: Feature and label. The reason for two type of column is "supervised nature of KNN algorithm".



```
# Assigning features and label variables
# First Feature
weather=['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy','Overcast','Sunny','Sunny',
'Rainy','Sunny','Overcast','Overcast','Rainy']
# Second Feature
temp=['Hot','Hot','Hot','Mild','Cool','Cool','Cool','Mild','Cool','Mild','Mild','Mild','Hot','Mild']

# Label or target varible
play=['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Yes','Yes','No']
```

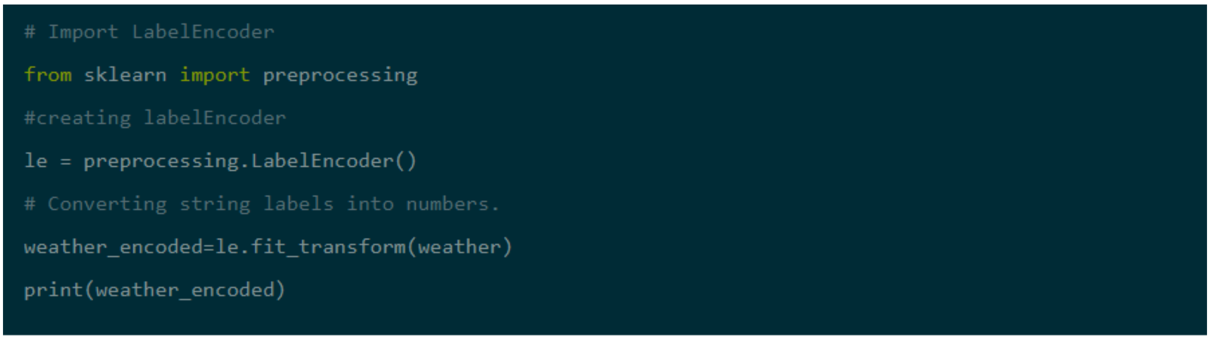
In this dataset, you have two features (weather and temperature) and one label(play).

Encoding data columns

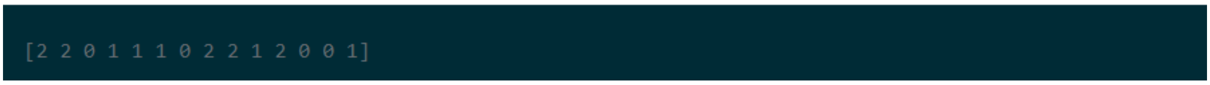
Various machine learning algorithms require numerical input data, so you need to represent categorical columns in a numerical column.

In order to encode this data, you could map each value to a number. e.g. Overcast:0, Rainy:1, and Sunny:2.

This process is known as label encoding, and sklearn conveniently will do this for you using Label Encoder.



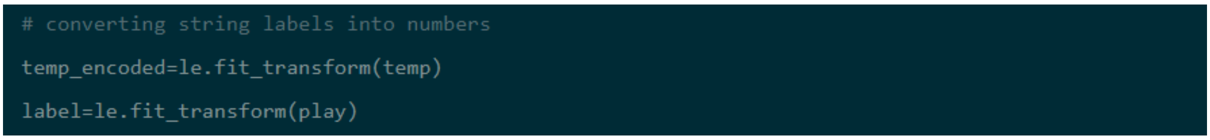
```
# Import LabelEncoder
from sklearn import preprocessing
#creating labelEncoder
le = preprocessing.LabelEncoder()
# Converting string labels into numbers.
weather_encoded=le.fit_transform(weather)
print(weather_encoded)
```



```
[2 2 0 1 1 1 0 2 2 1 2 0 0 1]
```

Here, you imported pre-processing module and created Label Encoder object. Using this LabelEncoder object, you can fit and transform "weather" column into the numeric column.

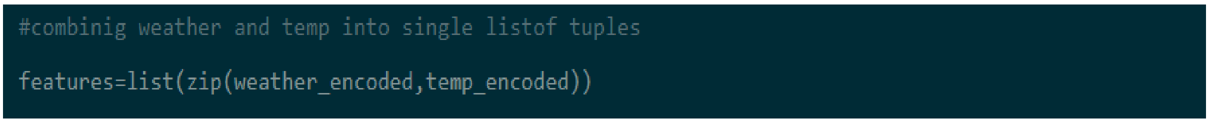
Similarly, you can encode temperature and label into numeric columns.



```
# converting string labels into numbers
temp_encoded=le.fit_transform(temp)
label=le.fit_transform(play)
```

Combining Features

Here, you will combine multiple columns or features into a single set of data using "zip" function



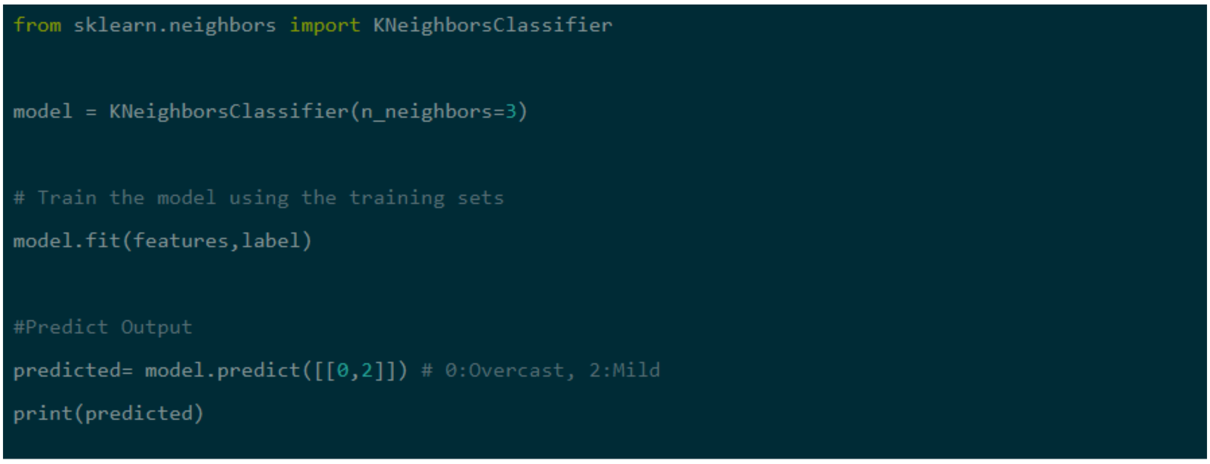
```
#combinig weather and temp into single listof tuples
features=list(zip(weather_encoded,temp_encoded))
```

Generating Model

Let's build KNN classifier model.

First, import the KNeighborsClassifier module and create KNN classifier object by passing argument number of neighbors in KNeighborsClassifier() function.

Then, fit your model on the train set using fit() and perform prediction on the test set using predict().

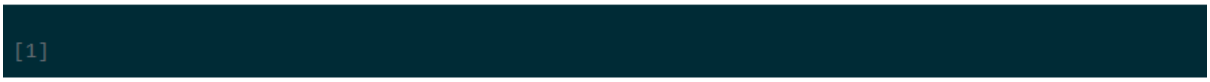


```
from sklearn.neighbors import KNeighborsClassifier

model = KNeighborsClassifier(n_neighbors=3)

# Train the model using the training sets
model.fit(features,label)

#Predict Output
predicted= model.predict([[0,2]]) # 0:Overcast, 2:Mild
print(predicted)
```



```
[1]
```

In the above example, you have given input [0,2], where 0 means Overcast weather and 2 means Mild temperature. Model predicts [1], which means play.

Pros

The training phase of K-nearest neighbor classification is much faster compared to other classification algorithms. There is no need to train a model for generalization, that is why KNN is known as the simple and instance-based learning algorithm. KNN can be useful in case of nonlinear data. It can be used with the regression problem. Output value for the object is computed by the average of k closest neighbors value.

Cons

The testing phase of K-nearest neighbor classification is slower and costlier in terms of time and memory. It requires large memory for storing the entire training dataset for prediction. KNN requires scaling of data because KNN uses the Euclidean distance between two data points to find nearest neighbors. Euclidean distance is sensitive to magnitudes. The features with high magnitudes will weigh more than features with low magnitudes. KNN also not suitable for large dimensional data.

How to improve KNN?

For better results, normalizing data on the same scale is highly recommended. Generally, the normalization range considered between 0 and 1. KNN is not suitable for the large dimensional data. In such cases, dimension needs to reduce to improve the performance. Also, handling missing values will help us in improving results.

2.4.2 Baseline

In every machine learning task, it is always good to have what we called a baseline. It often a “quick and dirty” implementation of a basic model for doing the first classification and based on its accuracy, try to improve it.

We use the knn(k-nearest neighbors) as learning algorithm representing the classic way of doing text classification.

Firstly, we divide the data set into two parts, the training set and the test set. To do this, we first shuffle the data set to get rid of any order applied to the data, then we from the set of 10 different types of sentiment tweets, we take 3/4 of tweets from each set and merge them together to make the training set. The rest is used to make the test set. Finally, the size of the training set is 20649 tweets and the test set is 6883 tweets.

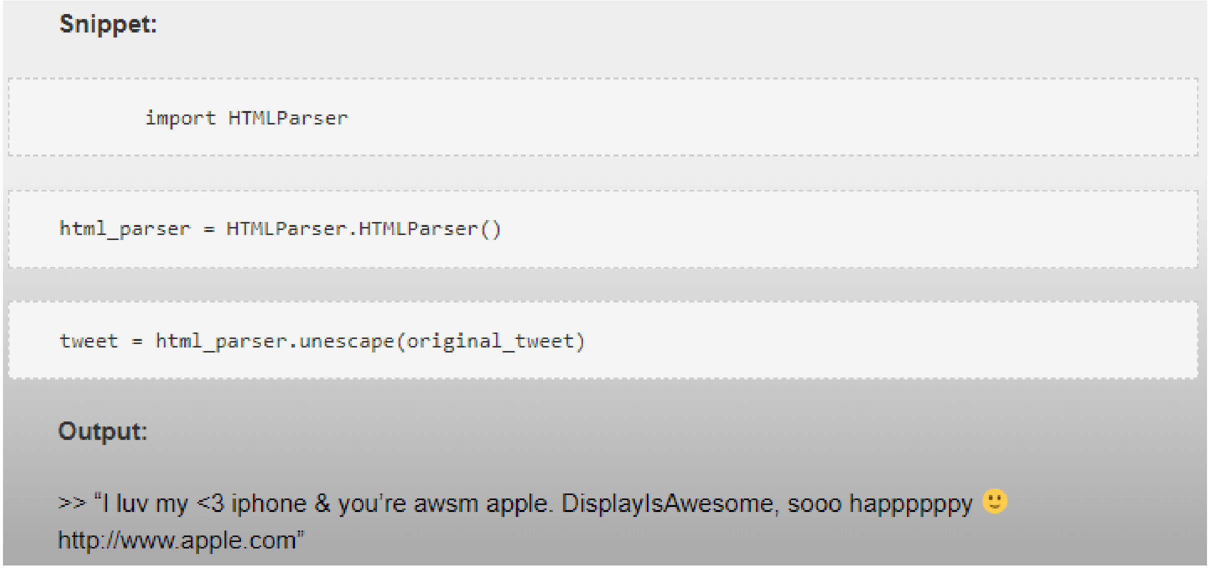
We used KNeighborsClassifier((n_neighbors = 15), inbuilt python function for knn where k = 15. We then tested our own tweets, noted the result and planned for some improvements if needed.

2.4.3 Improvements

From the baseline, the goal is to improve the accuracy of the classifier, in order to determine better. There are several ways of doing this and we present only few possible improvements (or not).

Here is what you do:

1. **Escaping HTML characters**: Data obtained from web usually contains a lot of html entities like &lt; &gt; &amp; which gets embedded in the original data. It is thus necessary to get rid of these entities. One approach is to directly remove them by the use of specific regular expressions. Another approach is to use appropriate packages and modules (for example htmlparser of Python), which can convert these entities to standard html tags. For example: &lt; is converted to “<” and &amp; is converted to “&”.



Snippet:

```
        import HTMLParser
```

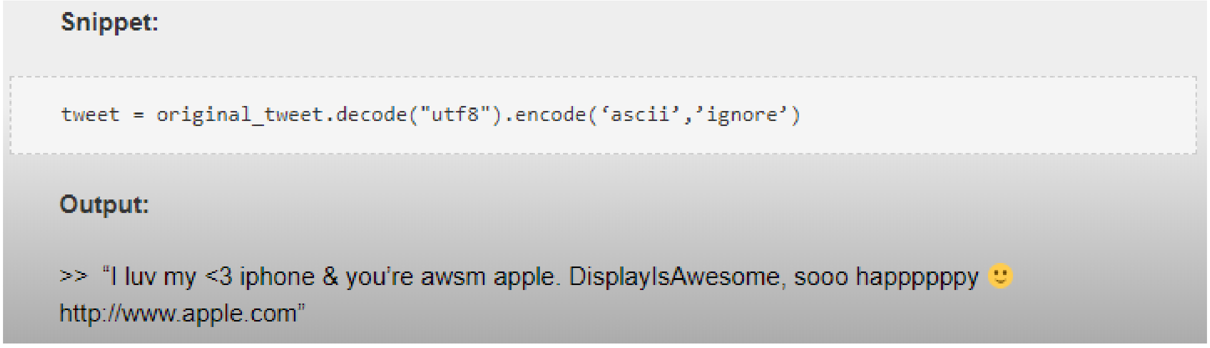
```
html_parser = HTMLParser.HTMLParser()
```

```
tweet = html_parser.unescape(original_tweet)
```

Output:

```
>> “I luv my <3 iphone & you’re awsm apple. DisplayIsAwesome, sooo happppppy 🙂
http://www.apple.com”
```

2. **Decoding data**: This is the process of transforming information from complex symbols to simple and easier to understand characters. Text data may be subject to different forms of decoding like “Latin”, “UTF8” etc. Therefore, for better analysis, it is necessary to keep the complete data in standard encoding format. UTF-8 encoding is widely accepted and is recommended to use.



Snippet:

```
tweet = original_tweet.decode("utf8").encode(‘ascii’,’ignore’)
```

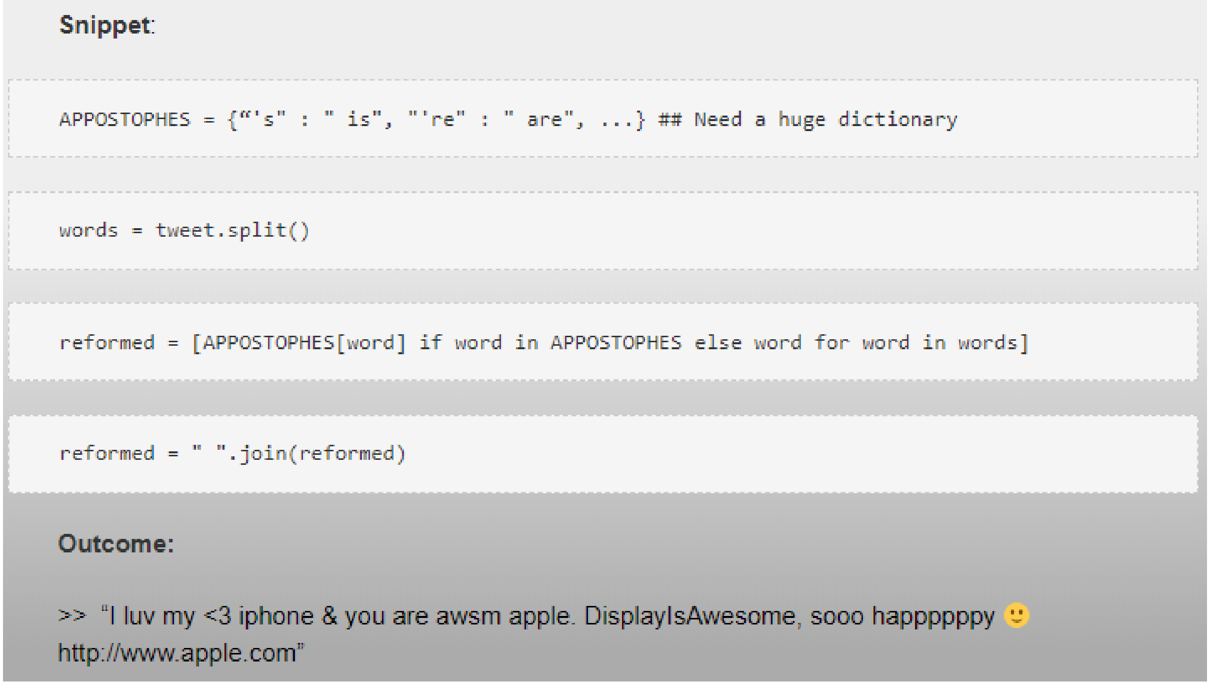
Output:

>> “I luv my <3 iphone & you’re awsm apple. DisplayIsAwesome, sooo happppppy 🙂 http://www.apple.com”

3. **Apostrophe Lookup**: To avoid any word sense disambiguation in text, it is recommended to maintain proper structure in it and to abide by the rules of context free grammar. When apostrophes are used, chances of disambiguation increase.

For example, “it’s is a contraction for it is or it has”.

All the apostrophes should be converted into standard lexicons. One can use a lookup table of all possible keys to get rid of disambiguates.



Snippet:

```
APPOSTOPHES = {“'s" : " is", "'re" : " are", ...} ## Need a huge dictionary
```

```
words = tweet.split()
```

```
reformed = [APPOSTOPHES[word] if word in APPOSTOPHES else word for word in words]
```

```
reformed = " ".join(reformed)
```

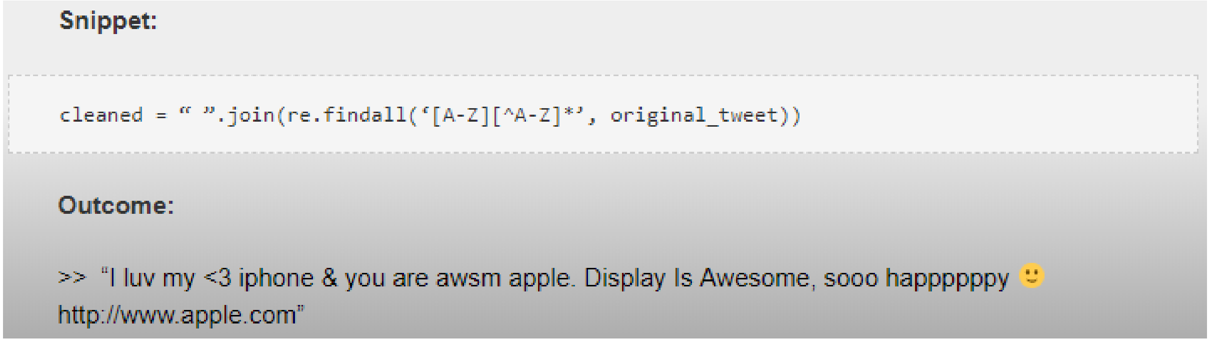
Outcome:

>> “I luv my <3 iphone & you are awsm apple. DisplayIsAwesome, sooo happppppy 🙂 http://www.apple.com”

4. **Removal of Stop-words**: When data analysis needs to be data driven at the word level, the commonly occurring words (stop-words) should be removed. One can either create a long list of stop-words or one can use predefined language specific libraries.
5. **Removal of Punctuations**: All the punctuation marks according to the priorities should be dealt with. For example: “.”, “,”,”?” are important punctuations that should be retained while others need to be removed.

6. **Removal of Expressions**: Textual data (usually speech transcripts) may contain human expressions like [laughing], [Crying], [Audience paused]. These expressions are usually non relevant to content of the speech and hence need to be removed. Simple regular expression can be useful in this case.

7. **Split Attached Words**: We humans in the social forums generate text data, which is completely informal in nature. Most of the tweets are accompanied with multiple attached words like RainyDay, PlayingInTheCold etc. These entities can be split into their normal forms using simple rules and regex.



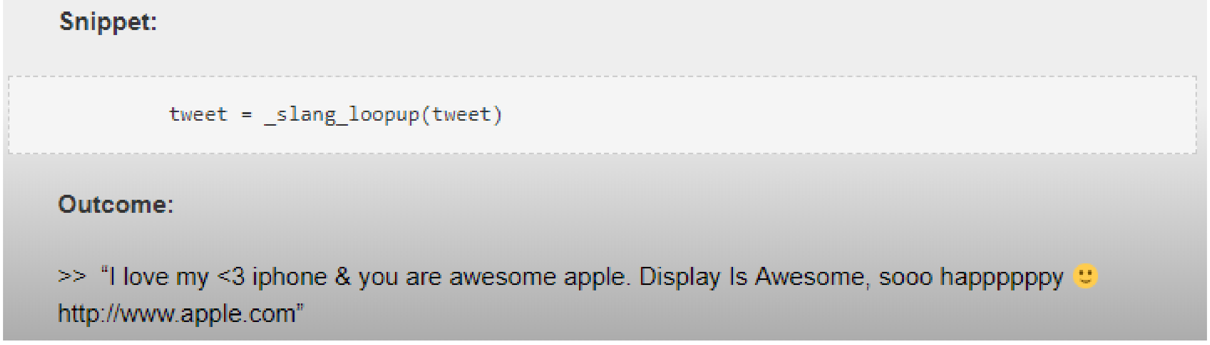
Snippet:

```
cleaned = “ ”.join(re.findall(‘[A-Z][^A-Z]*’, original_tweet))
```

Outcome:

>> “I luv my <3 iphone & you are awsm apple. Display Is Awesome, sooo happppppy 🙂 http://www.apple.com”

8. **Slangs lookup**: Again, social media comprises of a majority of slang words. These words should be transformed into standard words to make free text. The words like luv will be converted to love, Helo to Hello. The similar approach of apostrophe look up can be used to convert slangs to standard words. A number of sources are available on the web, which provides lists of all possible slangs, this would be your holy grail and you could use them as lookup dictionaries for conversion purposes.



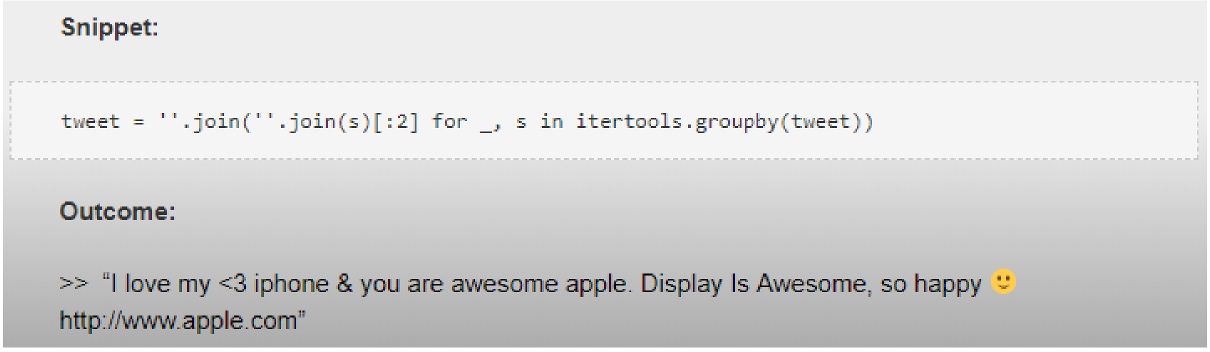
Snippet:

```
        tweet = _slang_loopup(tweet)
```

Outcome:

>> “I love my <3 iphone & you are awesome apple. Display Is Awesome, sooo happppppy 🙂 http://www.apple.com”

9. **Standardizing words**: Sometimes words are not in proper formats. For example: “I looooveee you” should be “I love you”. Simple rules and regular expressions can help solve these cases.



Snippet:

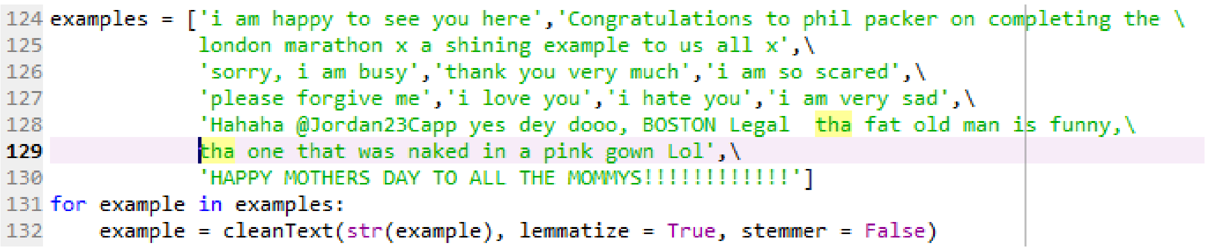
```
tweet = ''.join(''.join(s)[:2] for _, s in itertools.groupby(tweet))
```

Outcome:

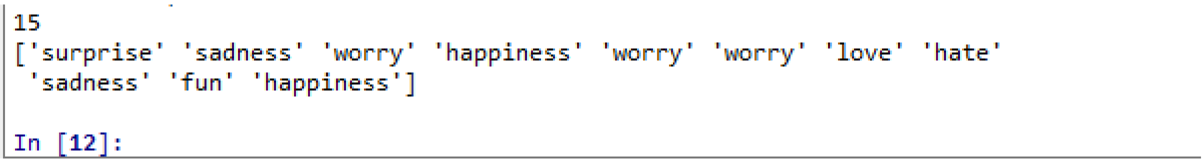
>> "I love my <3 iphone & you are awesome apple. Display Is Awesome, so happy 🙂 http://www.apple.com"

3. Result and Conclusion

We tested the knn(k-nearest neighbors) algorithm using our testing tweets which are 10 different sentiment tweets and observed the output.



```
examples = ['i am happy to see you here','Congratulations to phil packer on completing the \
            london marathon x a shining example to us all x',\
            'sorry, i am busy','thank you very much','i am so scared',\
            'please forgive me','i love you','i hate you','i am very sad',\
            'Hahaha @Jordan23Capp yes dey dooo, BOSTON Legal  tha fat old man is funny,\
            tha one that was naked in a pink gown Lol',\
            'HAPPY MOTHERS DAY TO ALL THE MOMMYS!!!!!!!!!!!!']
for example in examples:
    example = cleanText(str(example), lemmatize = True, stemmer = False)
```



```
15
['surprise' 'sadness' 'worry' 'happiness' 'worry' 'worry' 'love' 'hate'
 'sadness' 'fun' 'happiness']

In [12]:
```

In the above output, we can see that all the tweets are correctly classified. Similarly, we performed the operation with multiple datasets. Knn(k-nearest neighbors) gave 90% accuracy when tested with large datasets.

4. Source Code

source1.py

```
import csv
import re
import xlrd
import numpy
from bs4 import BeautifulSoup
from pandas import DataFrame
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
import nltk
import string
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem.wordnet import WordNetLemmatizer
from sklearn.naive_bayes import MultinomialNB
stop_words = set(stopwords.words('english'))
exclude = set(string.punctuation)
lemma = WordNetLemmatizer()
from textblob import TextBlob
from nltk.stem import SnowballStemmer
from nltk import pos_tag
from nltk.corpus import wordnet

def cleanText(msg, lemmatize, stemmer):
    if isinstance(msg, float):
        msg = str(msg)
    if isinstance(msg, numpy.int64):
        msg = str(msg)
    try:
        msg = msg.decode()
    except AttributeError:
        pass

    soup = BeautifulSoup(msg, "lxml")
    msg = soup.get_text()                   #to get text from soup
    msg = re.sub(r"[^A-Za-z]", " ", msg)    #relace non alphabetical
                                             characters with spaces
    msg = msg.lower()                         #to lowercase
    msg = " ".join(filter(lambda x:x[0]!='@', msg.split()))
    clean = [word for word in msg.split() if word not in stop_words]
    msg = " ".join(clean)
    clean = [word for word in msg.split() if len(word) >= 3]
    msg = " ".join(clean)
    msg = ''.join(ch for ch in msg if ch not in exclude)
    msg = " ".join(lemma.lemmatize(word) for word in msg.split())
    msg = re.sub(r"\d+","",msg)
    text_result = []
    tokens = word_tokenize(msg)
    snowball_stemmer = SnowballStemmer('english') #for lemmatization
    for t in tokens:
        text_result.append(snowball_stemmer.stem(t))
    return text_result

def dataFrameFromDirectory():
    rows = []
    index = []
    loc = "data1.xls"
    wb = xlrd.open_workbook(loc)
    sheet = wb.sheet_by_index(0)
    n = sheet.nrows
    j = 0
    stop_words = set(stopwords.words('english'))
    exclude = set(string.punctuation)
    lemma = WordNetLemmatizer()
    for i in range(n):
        c = sheet.cell_value(i,1)
        msg = sheet.cell_value(i,0)
        if (len(msg) > 0):
            msg = cleanText(str(msg), lemmatize = True, stemmer =
            False)
            rows.append({'message':str(msg),'class':str(c)})
            #classification})
            index.append('y')
            j += 1
    print("j ",j)
    return DataFrame(rows,index = index)

data = DataFrame({'message':{}, 'class':{}})
data = data.append(dataFrameFromDirectory())
print(data.head())
vectorizer = CountVectorizer()
counts = vectorizer.fit_transform(data['message'].values)
#classifier = MultinomialNB()
classifier = KNeighborsClassifier(n_neighbors = 15 )
#MultinomialNB()
print('15')
targets = data['class'].values
classifier.fit(counts, targets)

examples = ['i am happy to see you here', 'Congratulations to phil
            packer on completing the london marathon x a shining
            example to us all x', 'sorry, i am busy', 'thank you
            very much', 'i am so scared', 'please forgive me', 'i
            love you', 'i hate you', 'i am very sad', 'Hahaha
            @Jordan23Capp yes dey dooo, BOSTON Legal  tha fat old
            man is funny, the one that was naked in a pink gown
            Lol', 'HAPPY MOTHERS DAY TO ALL THE MOMMYS!!!!!!!!!!!!']

for example in examples:
    example = cleanText(str(example), lemmatize = True, stemmer =
             False)

example_counts = vectorizer.transform(examples)
predictions = classifier.predict(example_counts)
print(predictions)
```

source2.py

```
import csv
import re
import xlrd
import numpy
from bs4 import BeautifulSoup
from pandas import DataFrame
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
import nltk
import string
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem.wordnet import WordNetLemmatizer
from sklearn.naive_bayes import MultinomialNB
stop_words = set(stopwords.words('english'))
exclude = set(string.punctuation)
lemma = WordNetLemmatizer()
from nltk.stem import SnowballStemmer
from nltk import pos_tag
from nltk.corpus import wordnet
import itertools

contractions = {
"u": "you",
"ain't": "am not / are not",
"aren't": "are not / am not",
"can't": "cannot",
"can't've": "cannot have",
"'cause": "because",
"could've": "could have",
"couldn't": "could not",
"couldn't've": "could not have",
"didn't": "did not",
"doesn't": "does not",
"don't": "do not",
"hadn't": "had not",
"hadn't've": "had not have",
"hasn't": "has not",
"haven't": "have not",
"he'd": "he had / he would",
"he'd've": "he would have",
"he'll": "he shall / he will",
"he'll've": "he shall have / he will have",
"he's": "he has / he is",
"how'd": "how did",
"how'd'y": "how do you",
"how'll": "how will",
"how's": "how has / how is",
"i'd": "I had / I would",
"i'd've": "I would have",
"i'll": "I shall / I will",
"i'll've": "I shall have / I will have",
"i'm": "I am",
"i've": "I have",
"isn't": "is not",
"it'd": "it had / it would",
"it'd've": "it would have",
"it'll": "it shall / it will",
"it'll've": "it shall have / it will have",
"it's": "it has / it is",
"let's": "let us",
"ma'am": "madam",
"mayn't": "may not",
"might've": "might have",
"mightn't": "might not",
"mightn't've": "might not have",
"must've": "must have",
"mustn't": "must not",
"mustn't've": "must not have",
"needn't": "need not",
"needn't've": "need not have",
"o'clock": "of the clock",
"oughtn't": "ought not",
"oughtn't've": "ought not have",
"shan't": "shall not",
"sha'n't": "shall not",
"shan't've": "shall not have",
"she'd": "she had / she would",
"she'd've": "she would have",
"she'll": "she shall / she will",
"she'll've": "she shall have / she will have",
"she's": "she has / she is",
"should've": "should have",
"shouldn't": "should not",
"shouldn't've": "should not have",
"so've": "so have",
"so's": "so as / so is",
"that'd": "that would / that had",
"that'd've": "that would have",
"that's": "that has / that is",
"there'd": "there had / there would",
"there'd've": "there would have",
"there's": "there has / there is",
"they'd": "they had / they would",
"they'd've": "they would have",
"they'll": "they shall / they will",
"they'll've": "they shall have / they will have",
"they're": "they are",
"they've": "they have",
"to've": "to have",
"wasn't": "was not",
"we'd": "we had / we would",
"we'd've": "we would have",
"we'll": "we will",
"we'll've": "we will have",
"we're": "we are",
"we've": "we have",
"weren't": "were not",
"what'll": "what shall / what will",
"what'll've": "what shall have / what will have",
"what're": "what are",
"what's": "what has / what is",
"what've": "what have",
"when's": "when has / when is",
"when've": "when have",
"where'd": "where did",
"where's": "where has / where is",
"where've": "where have",
"who'll": "who shall / who will",
"who'll've": "who shall have / who will have",
"who's": "who has / who is",
"who've": "who have",
"why's": "why has / why is",
"why've": "why have",
"will've": "will have",
"won't": "will not",
"won't've": "will not have",
"would've": "would have",
"wouldn't": "would not",
"wouldn't've": "would not have",
"y'all": "you all",
"y'all'd": "you all would",
"y'all'd've": "you all would have",
"y'all're": "you all are",
"y'all've": "you all have",
"you'd": "you had / you would",
"you'd've": "you would have",
"you'll": "you shall / you will",
"you'll've": "you shall have / you will have",
"you're": "you are",
"you've": "you have"
}


def dataFrameFromDirectory():
    rows = []
    index = []
    loc = "data1.xls"
    wb = xlrd.open_workbook(loc)
    sheet = wb.sheet_by_index(0)
    n = sheet.nrows
    j = 0
    stop_words = set(stopwords.words('english'))
    exclude = set(string.punctuation)
    lemma = WordNetLemmatizer()
    for i in range(n):
        c = sheet.cell_value(i,1)
        msg = sheet.cell_value(i,0)
        msg = str(msg)
        if (len(msg) > 0):
            for word in msg.split():
                if word.lower() in contractions:
                    msg = msg.replace(word,
                    contractions[word.lower()])
            #msg = " ".join(msg)
            msg = " ".join(re.findall('[A-Z][^A-Z]*', msg))
            #msg = slang_lookup(msg)
            msg = ''.join(''.join(s)[:2] for _, s in
            itertools.groupby(msg))     # standardising words i
                                          looovvvveeeee
                                          you to i love you

            rows.append({'message':str(msg),'class':str(c)})
                                           #classification})
            index.append('y')
            j += 1
    print("j ",j)
    return DataFrame(rows,index = index)

data = DataFrame({'message':{}, 'class':{}})
data = data.append(dataFrameFromDirectory())
print(data.head())
vectorizer = CountVectorizer()
counts = vectorizer.fit_transform(data['message'].values)
#classifier = MultinomialNB()
classifier = KNeighborsClassifier(n_neighbors = 13 )
#MultinomialNB()
print('13')
targets = data['class'].values
classifier.fit(counts, targets)

examples = ['i am happy to see you here', 'Congratulations to phil
            packer on completing the london marathon x a shining
            example to us all x', 'sorry, i am busy', 'thank you
            very much', 'i am so scared', 'please forgive me', 'i
            love you', 'i hate you', 'i am very sad', 'Hahaha
            @Jordan23Capp yes dey dooo, BOSTON Legal  tha fat old
            man is funny, the one that was naked in a pink gown
            Lol', 'HAPPY MOTHERS DAY TO ALL THE MOMMYS!!!!!!!!!!!!']

example_counts = vectorizer.transform(examples)
predictions = classifier.predict(example_counts)
print(predictions)
```

5. Acknowledgement

I would like to express my deepest appreciation to all those who provided me the possibility to complete this report. A special gratitude I give to our project manager, Mr. **Suvamoy Changder**, whose contribution in stimulating suggestions and encouragement, helped me to coordinate my project especially in writing this report.

I wish to acknowledge **Dr. Goutam Sanyal, HOD of CSE**, for providing supporting environment for my work. I would also like to show gratitude and respect to other faculty members of our department for their encouragement.

Last but not the least, I would like to thank my family, my parents and my friends for their unconditional support.

6. References

- https://www.analyticsvidhya.com/blog/2014/11/text-data-cleaning-steps-python/
- https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/
- https://appliedmachinelearning.blog/2018/01/18/conventional-approach-to-text-classification-clustering-using-k-nearest-neighbor-k-means-python-implementation/
- https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn
- http://dataaspirant.com/2016/12/30/k-nearest-neighbor-implementation-scikit-learn/
- https://figshare.com/articles/smile_annotations_final_csv/3187909
- https://github.com/dabbabi-zayani/Twitter-Sentiment-Classifier/blob/master/src/knn.py
- https://towardsdatascience.com/text-classification-using-k-nearest-neighbors-46fa8a77acc5