Short text Filtering for Social Media Mining

Saurabh Kumar, Archit Muchhal

1. Introduction:

In the 21st century, with its ever-increasing vehicular population, as well as the limited road infrastructure throws out a great challenge to the daily commuter, mainly, in causing delays. Positive steps have been taken in the direction to come up with innovations for better traffic coordination like introducing mobile alerts for traffic incidents [15], social media pages for traffic updates, as well as capturing traffic through cameras and feeding it further, but proactive traffic filtering and crowd sourcing on ground are still the key challenges which need to be addressed.

An abundance of data is generated daily on Twitter in the form of tweets pertaining to congestion notifications, slow moving traffic updates, traffic images, and news ticker data. This information can be of great value to traffic administrators and commuters in proactive monitoring, decision making and effective analysis of the traffic condition. This traffic related information comprises of updates on slow moving traffic, congestion, road construction, and many other valuable data [2], which, if extracted properly can be beneficial for law enforcement, traffic administrators and commuters.

With effective and proactive traffic administration as the primary focus, we tackle these problems by introducing text filtering system. It is a system that will filter out the tweets having traffic related incidents (which can be further used as a ground for developing a traffic management system).

2. Related Work:

In terms of information extraction and data processing, microblogging websites like Twitter possess the hardest challenges, mainly, due to two reasons [20]. Firstly, the limited length (140-character maximum length) of micro-texts (called as tweets) makes the interpretation of the contained information hard. The next challenge being the structure and format of these tweets. These tweets contain Uniform Resource Locator (URLs), emoticons, abbreviations, hashtags (#), etc. which may not always contain valuable data. Moreover, these tweets, do not always follow the sentence flow as well as contain grammatical and punctuation commands, thereby making the data extraction harder.

This is a motivation for Napong, Wasawat, Wasan and Pimwadee [4] for finding the traffic related incidents using Twitter. To find out the tweets having traffic related information, to classify the traffic related tweets from other tweets, a fixed rule and dictionary based approach is used: Twitter feed must have two key words (at least) related to place and words category and Bag-of-words technique [4].

However, this is not the case with another study by Bharath, Dave, Engin, Hakan and Murat [6] where they have tried a Naïve Bayes Classifier based approach for filtering out the tweets. Moreover, to select features for the classifier algorithm, they define their own feature set called 8F [6], which consists of author information and seven other binary information.

Accuracy in a case of Napong, Wasawat, Wasan and Pimwadee is 97.75 % [4]. However, in the case of Bharath, Dave, Engin, Hakan and Murat methodology, they are not only focusing on the class of tweets [6], but also, they are trying to classify the tweets in four different categories whose accuracy vary with the classes.

Another feature selection study proposed by Zitao L., Wenchao Y., Wei C., Shuran W. and Fengyi W. of Paper [10], use Part-of-Speech (PoS) mechanism and “HowNet” to improve the clustering performance by choosing the words having a higher capacity of information. Initially, it uses PoS to filter out prepositions and functional words as they contain less information keeping only the nouns, adjectives and verbs assuming they hold extra information. Next, it uses the HowNet feature to improve the classification precision by improving the semantic similarities. As per this study, it states that, if the feature number is set between 200 to 1000, the feature selection method has a better classification result [10].

In another study by Takeshi, Makoto and Yutaka [7], earthquake events detection model is built using Twitter where they have used a combination of three feature sets, i.e. "**Statically feature, keyword feature and word context feature"**. This feature set is fed as an input to the Support Vector Machine (SVM). The cumulative accuracy is 87.50 % when all the feature sets are used. But in case of Divij and Chanh study, from Stanford, extracted feature set is done by the following method [5]:

1. Remove punctuations.
2. Remove URLs.
3. Spelling correction.
4. Remove consecutive characters.

Above feature set is used as input to different classifier algorithms like Logistic Regression with an accuracy of 82.71 %, SVM with 72.28% and Boosted decision tree with 68.7 % for positive tweets [5].

However, in our case study, we are using BOW as feature selection and Neural Network as a classifier.

3. Methodology:

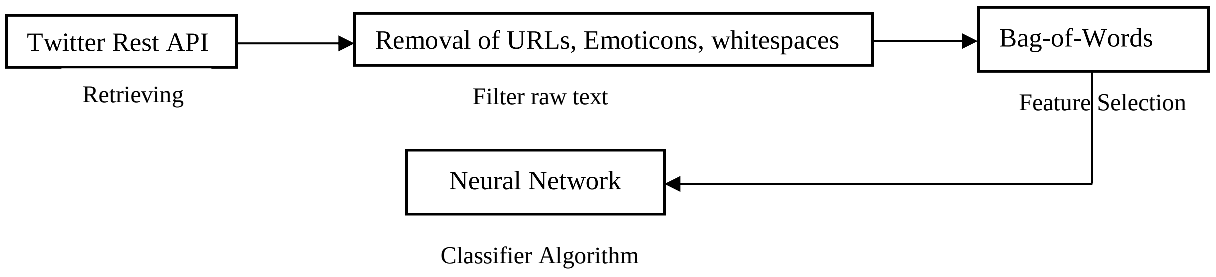

Figure 1: Pipeline

Our Twitter feed classifier pipeline is shown in figure 1.

In our methodology, there are four important parts. In the first part, we collect the tweets from Twitter Rest API and hand annotate (classify) the data, so that we can use this data for our supervised algorithm.

In the second stage, the task of noise removal from a tweet is performed. In this step, certain parts of the tweets will not provide any relevant information like URL (Universal Resource Locator), emoticons and special characters like hashtags (#), at the rate (@), hyphen, etc. which are to be removed from the tweets.

In the third section, the filtered text is converted into a vector by using Bag of Words (BOW) [4], which is basically, a feature selection technique, generating term frequency for sentences. The newly generated feature vector is fed to a supervised classifier algorithm i.e. Neural Network Algorithm, which we will train by using our 60 % of our hand-annotated dataset (3000 tweets), and for testing, the remaining 40 % of the data (2000 tweets) is going to be used as described in figure 2. The following subsections will explain our algorithm in detail.

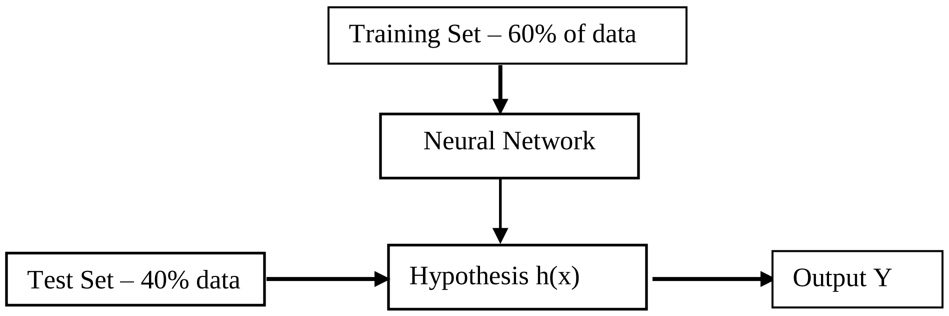

Figure 2: Training and testing data set

3.1 Retrieving the Twitter feeds from Twitter:

To get the tweets from the Twitter we are using Twitter timeline API [11], where we need to provide the specific userId as a get parameter to the API.

Here to get the as many data linked to traffic-related incidents, we are principally focusing on multiple Twitter accounts like the Chicago traffic police, a newspaper like the Daily Herald, the Chicago Tribune etc. A list of 26 Twitter accounts is considered for this purpose as shown in table 1.

```
SarahJindra, wazetrafficchi, GetRaasta, dtptraffic, TrafflineDEL, TrafflineCHN, TrafflineIndore,
TotalTrafficCHI, TfL, HighwaysSEAST, BillWest5, John_Kass, iKartikRao, shelvieana_prat,
prats_09, 94_294_tollway, DildineMedia, CrazyRicardo, TotalTrafficNYC, WazeTrafficNYC,
Traffic4NY, NYTrafficAlert, NYC_DOT, 511NYC, NYPD_Traffic, NYCityAlerts, TotalTrafficNYC
```

Table 1: List of Twitter accounts (handles) considered

3.2 Raw Text Filtering

The tweets that we get from the Twitter API include a substantial noise in it, like hashtags, URLs, and emoticons. To remove these unwanted entities from the text, we are using the regular expression (regex) [12]. In all, there are six regex which is generated [12] to remove URLs, special symbols like additional spaces, hashtags (#) and at the rate (@) as shown in table 2.

Rex Expression	**Function Performed**
“ ‘www\.[^\s]+http?://[^\s]+’,’ “	Remove URLs
“ '@[^\s]+',' “	Remove “@” sign
“ ‘[\s]+', ' “	Remove unnecessary spaces
“ ‘#([^\s]+)’,r’\l’ “	Remove “#” (hashtags)
“(.)\1{1,}”,”\1\1”	Remove repeated words
“ ‘[^A-Za-z}+’,’ “	Remove Special characters

Table 2: Regular Expressions

3.3 Feature Extraction

Feature extraction can be described as a process that converts raw data into a useful set of features called a feature vector, which can be an input to a classifier algorithm [8].

To convert a tweet into a feature vector, the Bag of Words (BOW) method is used. The BOW method generates a term frequency model, which is basically a list of numbers produced based on how many times a term is appearing in the text [4].

Findings: For extracting the features, we tested the bag of words and the Tf-Idf algorithm but we did not get any significant boost in the output of the classifier (accuracy is only 1 % more in Tf-Idf case) and that's why we decided to go with the bag of words technique (Tf-Idf is more computationally intensive as compared to Bag to words).

One fact that we found for poor performance of the system (accuracy is 60 %) is that both algorithms i.e. BOW and Tf-Idf ignore the semantics of the words, the other reason is the tweet text itself, as in case of social media, people generally, do not follow any grammar rules and tense structure properly.

So, to overcome the above problem, we figured out that out that a feature selection method that can understand the structure of text and extract the feature by itself (kind of unsupervised learning) should be used in place of above frequency-based algorithm and that is why we will use word2vec [30] in our next implementation, which uses the deep neural network for automatic feature extraction.

3.4 Classifier Algorithm

In our hand-annotated dataset, we have tweets related to politics, personal communications, product promotions, traffic related incidents and much more. To filter out the tweets linked with traffic-related incidents, we have used a Neural Network (NN) classifier [29].

Brief overview of a Neural Network

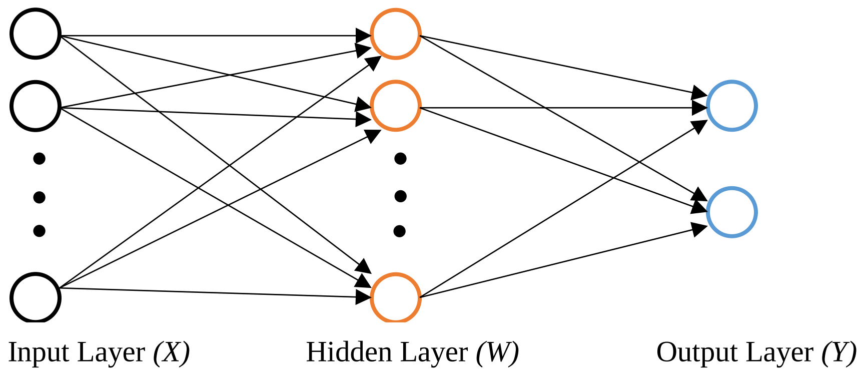

Figure 3: Feed Forward Neural Network

Input to the neural network is a vector of length 2300 from bag-of-words algorithm

Hidden Layer

$b_{ith} = \Theta_0.\alpha_0 + \Theta_1.\alpha_1 + \Theta_2.\alpha_2 + \Theta_3.\alpha_3 + \ldots\ldots \Theta_{2300}.\alpha_{2300}$

where, b_{ith} is the i^{th} value of the hidden layer neuron.

α_i is the value of the input neuron.

Θ_i is the i^{th} weight of the neuron.

Where Θ_i values are randomly initialized.

Reason for Selecting Neural Network:

We have tried two classifier algorithm Logistic Regression (LR) and Neural Network. Over LR Neural Network (NN) is chosen as:

In LR, feature selection has to me done manually while in NN the hidden layer can think of the feature selector, more the number of hidden layer, more complex feature for non-linear boundary can be determined. [28]

NN classifier blueprint:

1. Number of hidden layers - 1
2. Number of neurons in hidden layer - 8
3. Number of input neurons - 2300
4. Alpha value for gradient descent – 10^{-3}
5. Number of output neurons - 1

Reasons for above selection of parameters:

Ideally, there are no strict rules to select the parameters, but below are the points which came up during the implementation phase –

1. Hidden Layer:
   To decide the number of hidden layers, we started with one hidden layer and then gradually increased the number of layers and calculate the accuracy of the classifier. When we increase the number of hidden layer by one, the accuracy of the classifier changes by the order of 2% at first, and then 1% (approx.) on every step and that's why we select only one hidden layer (because of the limited computational power and a small improvement in accuracy).

2. Number of neurons in hidden layer:
   To calculate the number of neuron in the hidden layers, we found that (by hit and trial method), the number should always be in the range of *(a, b)*, where *a* is the number of the neuron in the output layer and *b* is the number of neuron in the input layer. We started with five random values which starts with *a* and ends at *b* and select those values based on:

A. Classifier accuracy
B. Computation time for training.

By considering point, A and B, 8 is the best option for the number of neurons in the hidden layer which will be used for the further stages.

3. Number of input neurons:
This is a dynamic selection which depends on the length of the *i*th input vector and in our case length of the *i*th vector is 2300. (This is dynamic, that can change as per our dataset and text vectoring algorithm)

4. Alpha value for gradient descent:
Alpha value determines at what rate the cost function decreases. If it is too large, then it might miss the minima and if it's too small, then its convergence rate to minima is too slow. So, to select the alpha value we start with 10^{-10}, 10^{-5}, 10^{-4}, ... 10^{0}, with fixed number of iterations for gradient descent (in our case it is 5000) and find out the 10^{-3} is suitable in our case. As with 10^{-3} cost, the function decreases continuously which is not the case with other values.

5. Number of output neurons:
Since our output is either 0 (tweet do not have traffic information) or 1(tweet have traffic information). So, output neuron is one.

4. Implementation:

As discussed in the previous sections, this study to classify the raw Twitter database, the Bag-of-Words technique will be used for the feature extraction method to convert the raw Twitter text into vector representation. The Neural Network Algorithm is used to determine if the extracted tweets contain traffic data or not.

From the implementation purposes, Python is chosen as the programing language, the version being, Python 2.7.10. The Twitter database is collected from the open source database, MySQL (Version 3.5). MySQL is an open source rational database management system (RDMS), used for storage of information in a database format. [21]

External Library Used:

1. MySQL connector for Python Version 3.5 [22]
2. Tweepy – For accessing Twitter Rest APIs Version 3.5 [23]
3. Panda – For data structure Version 0.19.0 [24]

IDE Used:

1. Spyder – A Python development environment Version 2.3.5. [25]
2. HiediSQL – A tool for database web management Version 9.3 [26]

5. Result

The most appropriate way to analyze the classifier output is the confusion matrix [31] on a set of data for which the output is known. A confusion matrix is a table structured representation that is used to outline and illustrate the performance of a classification model on the set of test data for which the true values are already know. Our test data is a set of 2290 tweets which have been classified if the given tweet contains traffic information or not. If the tweet contains traffic data, a binary high value (1) is assigned to it, while, if the tweets does not provide any traffic related information, the respective tweet is assigned a binary low value (0). The confusion matrix is generally represented in a 4-way term structured format as -

A. True Positives (TP): These are cases in which we predicted yes (have traffic information), and they do have the traffic information.
B. True Negatives (TN): We predicted no, and they don't have the traffic information.
C. False Positives (FP): We predicted yes, but they don't have the traffic information.
D. False Negatives (FN): We predicted no, but they do have the traffic information.

The confusion matrix will look like below:

Test Data (n) = 2290	Actual Positive	Actual Negative
Predicted Positive	TP	FP
Predicted Negative	FN	TN

Figure 4: Confusion Matrix Representation

And, in our case its looks like below:

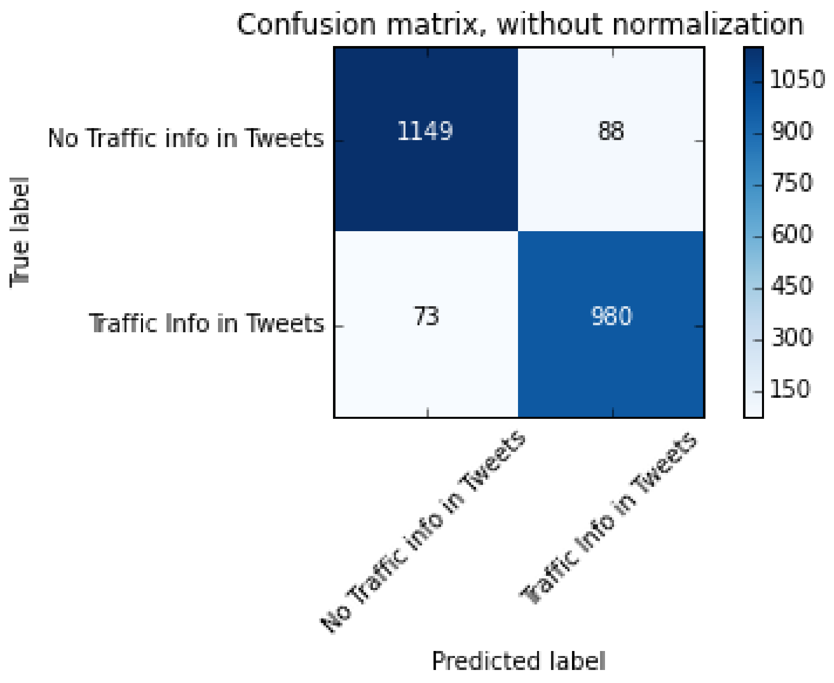

Figure 5: Confusion Matrix

There is a list of the following performance measures [31] that are computed from a confusion matrix -

- Accuracy: This is used to check the correctness of the classifier in use.
    - Formula: $\frac{TP+TN}{n}$
- Precision: This term is used to get an update on how many selected items are relevant.
    - Formula: $\frac{TP}{TP+FP}$
- Recall: This simply states how many relevant items are selected.
    - Formula: $\frac{TP}{TP+FN}$

Accuracy	**Precision**	**Recall**
$\frac{1149+980}{2290} = 92.96\ \%$	$\frac{1149}{1149+88} = 92.88\ \%$	$\frac{1149}{1149+73} = 94.06\ \%$

Figure 6: Accuracy, Precision and Recall

Note: Above result was bit different because, previously when we initialized the weights by some static numbers, accuracy was around 60 %, but right now when the weights are assigned in a range $\in [-\varepsilon, \varepsilon]$ accuracy is more then 90 %.

$$\varepsilon = \frac{\surd 6}{\sqrt{L_{in}} + \surd L_{out}}$$

where,

L_{in} = number of neurons in the previous layer

L_{out} = number of neurons in the next layer

Some of the reason for misprediction in results are as listed below:

1. Human error: Tweets were marked incorrectly, though tweets have traffic information but we marked them as non-traffic tweets.
2. Hazy information: Information in some tweets is not clear, consider tweets number 7 where it is difficult to find whether it has traffic information or not.
3. Too simple hypothesis: Our neural network has only one hidden layer having eight neurons in it. So, that might lead to high bias problem where the hypothesis misses the relation between feature and the output.

Here are the some of the tweets those were predicted wrong:

Sr. No	Tweets	Prediction	Actual
1	UPDATE I Southbound VEHICLE FIRE South of th Ave MP right lanes still blocked of	0	1
2	Missing year old girl found safe in Manhattan Suspect in custody via	1	0
3	news sites for college students Plato Socrates Aristotle	0	1
4	Have a wonderful rest of your Wednesday everyone Safe travels	0	1
5	Closed due to accident in Bolingbrook on Weber Rd between Boughton Rd and Royce Rd traffic Chicago	1	0
6	Construction on BeltSystemShoreParkway EB from Exit B Knapp Street to Exit A Flatbush Avenue North	1	0
7	Sir kindly contact area traffic Inspector Saket Circle at	1	0
8	RT You smile I smile motivationmonday MotherTeresa	1	0

Table 3: Incorrectly predicted tweets

6. Conclusion

In text mining, to convert text into a vector, it's always a good strategy to start with some simple algorithm (like simply selection the most frequent words) that rather jump on algorithms like Wor2vec, TF-IDF or even Bag of Words). Also, while applying text preprocessing (like removing the stops words (like a, all, am is, am, are), URLs etc.) it’s always a good idea to use the text with any preprocessing and then apply the preprocessing, if there is any significance change in the machine learning algorithm accuracy then only use preprocessing methods.

In order to select the algorithm, start always from the simplest algorithm (in our case we are doing the classification, so the first algorithm was the logistic regression) and gradually use the complex algorithm and note down the shortcoming of the algorithm and use the another which can overcome the problem.

Likewise, we found that the problem with the logistic regression is that for complex hypothesis we need to modify the input. As, an example consider the below equations [] (where θ_n *are the weight* and x_n are the input features) where all the feature are modified to predict complex hypothesis but how to find out those feature is totally different task and time consuming. So the main problem with logistic regression is to deciding those feature manually.

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots$$

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 (x_2)^2 + \theta_3 (x_3)^4 \ldots$$

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 \sqrt{x_2} + \cdots$$

However, in case of Neural Network, the main advantage here that we don't have to manually manipulate the input feature as above because hidden layers itself act as a complex feature selector, more the number of hidden layer more complex feature we can get. But again if the number of layer and neuron is not selected carefully then our hypotheses might be landed to a high variance problem.

Future Scope

A sophisticated application for real-time detection and diagnosis of traffic incidents, to analyze, diagnose and resolve traffic problems that are scalable to multiple cities.

Reference:

[1] "Social Media and Its Effectiveness in the Political Reform Movement in Egypt", University of Wollongong, Australia

[2] Avinash K., Miao J., Yi F., "Where Not to go? Detecting Road Hazard Using Twitter", 37th International ACM SIGIR Conference.

[3] Raymondus K., Erwin A., Steven., "Harvesting Real Time Traffic Information from Twitter Article" in Procedia Engineering

[4] Napong W., Wasawat P., Wasan P.A. and Pimwadee C., "Social Based Traffic Information Extraction and Classification", 11th International Conference on ITS Telecommunications (ITST)

[5] Divij G. and Chanh N. "Detecting RealTime Messages of Public Interest in Tweets", 2010.

[6] Bharath S., Dave F., Engin D., Hakan F., Murat D., "Short text classification in Twitter to improve information filtering", 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval

[7] Takeshi S., Makoto O., Yutaka M., "Earthquake shakes Twitter users: real-time event detection by social sensors", 9th International Conference on World Wide Web

[8] "Wikipedia - The Free Encyclopedia," *Wikipedia - The Free Encyclopedia*. [Online]. Available: https://en.wikipedia.org/wiki/feature_extraction. [Accessed: 02-Oct-2016].

[9] M. Raeesi, M. S. Mesgari and P. Mahmoudi, "Traffic Time Series Forecasting by Feedforward Neural Network: A Case Study Based on Traffic Data of Monroe", ISPRS International Conference on Geospatial Information Research, 15-17 November 2014

[10] Zitao L., Wenchao Y., Wei C., Shuran W. and Fengyi W., "Short Text Feature Selection for MicroBlog Mining", IEEE 2010, 978-1-4244-5392-4/10

[11] "Twitter," *Twitter*. [Online]. Available: https://dev.twitter.com/rest/reference/get/statuses/user_timeline. [Accessed: 02-Oct-2016].

[12] "Microsoft," *Microsoft*. [Online]. Available: https://msdn.microsoft.com/en-us/library/az24scfc(v=vs.110).aspx. [Accessed: 02-Oct-2016].

[13] "RegExr," *RegExr*. [Online]. Available: http://regexr.com/. [Accessed: 02-Oct-2016].

[14] Z. S. Harris, "Distributional Structure," vol. 10, no. 1954, pp. 146–162, Dec. 2015.

[15] "Mumbai Introduces a Modern Traffic Management System," 14-Jun-2011. [Online]. Available: http://www.worldbank.org/en/news/feature/2011/06/14/mumbai-modern-traffic-management-system. [Accessed: 02-Oct-2016].

[16] C. Y. J. Peng, K. L. LEE, and G. M. Ingersoll, “An Introduction to Logistic Regression Analysis and Reporting,” *The Journal of Educational Research*.
Indiana University - Bloomington

[17] D.W. Hosmer,S. Lemeshow. *Applied Logistic Regression, Second Edition, chapter second* equation 1.1

[18] D.W. Hosmer,S. Lemeshow. *Applied Logistic Regression, Second Edition, chapter second* equation 1.4

[19] A Ng. Stanford Machine Learning Class, cs229, pg. no 5 cs229.stanford.edu/notes/cs229-notes1

[20] K. Bontcheva, L. Dercynski, A. Funk, M. A. Greenwood, D. Maynard, N. Aswani, “TwitIE: An Open-Source Information Extraction Pipeline for Microblog Text

[21] "MySQL", *En.wikipedia.org*, 2016. [Online]. Available: https://en.wikipedia.org/wiki/MySQL. [Accessed: 14- Oct- 2016]

[22] "MySQL: Download Connector/Python", *Dev.mysql.com*, 2016. [Online]. Available: https://dev.mysql.com/downloads/connector/python/. [Accessed: 14- Oct- 2016]

[23] "Tweepy", *Tweepy.org*, 2016. [Online]. Available: http://www.tweepy.org/. [Accessed: 14- Oct- 2016]

[24] "Python Data Analysis Library — pandas: Python Data Analysis Library", *Pandas.pydata.org*, 2016. [Online]. Available: http://pandas.pydata.org/. [Accessed: 14- Oct- 2016]

[25] "spyder-ide/spyder", *GitHub*, 2016. [Online]. Available: https://github.com/spyder-ide/spyder/. [Accessed: 14- Oct- 2016]

[26] A. Becker, "HeidiSQL - MySQL, MSSQL and PostgreSQL made easy", *Heidisql.com*, 2016. [Online]. Available: http://www.heidisql.com. [Accessed: 14- Oct- 2016]

[27] "Artificial Neural Network". En.wikipedia.org. N.p., 2016. Web. 31 Oct. 2016.

[28] "Decision Boundaries for Deep Learning and other Machine Learning classifiers", *Kdnuggets.com*, 2016. [Online]. Available: http://www.kdnuggets.com/2015/06/decision-boundaries-deep-learning-machine-learning-classifiers.html. [Accessed: 02- Nov- 2016].

[29] G. Shou-Ping and L. Rong-ye, "Study of full interval feed-forward neural network", 2016.

[30] T. Mikolov, I. Sutskever, K. Chen, G. Corrado and J. Dean, "Distributed Representations of Words and Phrases and their Compositionality", 2016.

[31] "Simple Guide to Confusion Matrix Terminology". *Data School*. N.p., 2016. Web. 16 Nov. 2016.

Code Appendices:

1. **neural.py**

```
import sys
total = 5000

from fileread.TweetRead import importData
import numpy as np
from bagofwords.textToVector import bagofWords
from sklearn.utils import shuffle

X, y = importData()
X = X[1:]
y = y[1:]
X, y = shuffle(X, y, random_state=0)

#X = np.array(X)
#vectorizer = CountVectorizer(min_df=2)
#X = vectorizer.fit_transform(X)

X = np.array(bagofWords(X))

y = np.array(map(int,list(y)))
num_examples =len(X)

ratio = .6
# training data
X_train = X[1:num_examples*ratio]
y_train = y[1:num_examples*ratio]

#test data
X_test = X[num_examples*ratio:]
y_test = y[num_examples*ratio:]


barLength = 10
status = ""

nn_input_dim = len(X[0])#neural netwoek input dimension
nn_output_dim = 2# true or false
neuron_number = 6 # in a layer
# Gradient descent parameters (I picked these by hand)
alpha = 0.001 # learning rate for gradient descent
reg_lambda = 0.001 # regularization strength
num_passe = 10000
def weight_init(L1,L2):
    return np.sqrt(6)/np.sqrt(L1 + L2)

def calculate_loss(model):
    W1,  W2 = model['W1'], model['W2']
    # Forward propagation to calculate our predictions
    num_ex = len(X_train)
    z1 = X_train.dot(W1)
    a1 = np.tanh(z1)
    z2 = a1.dot(W2)
    exp_scores = np.exp(z2)
    probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)
    # Calculating the loss
    corect_logprobs = -np.log(probs[range(num_ex), y_train])
    data_loss = np.sum(corect_logprobs)
    # Add regulatization term to loss (optional)
    data_loss += reg_lambda/2 * (np.sum(np.square(W1)) +
np.sum(np.square(W2)))
    return 1./num_ex * data_loss
def status(i,model):
    progress = (float(i)/num_passe)
    block = int(round(barLength*progress))
    sys.stdout.write('\r')
    text = "[{0}] {1}% Completed.".format( "#"*block + "-"*(barLength-
block), format(progress*100,".2f"),status)
    sys.stdout.write(text)
    sys.stdout.write ("           Current Loss %.5f."
%(calculate_loss(model)))
    sys.stdout.flush()


def predict(model, x):
    W1,  W2 = model['W1'],  model['W2']
    # Forward propagation
    z1 = x.dot(W1)
    a1 = np.tanh(z1)
    z2 = a1.dot(W2)
    exp_scores = np.exp(z2)
    probs = exp_scores / np.sum(exp_scores, keepdims=True)
    return np.argmax(probs)

def build_model(nn_hdim, num_passes=num_passe, print_loss=False):
    #Initilization of weight
    #L1 number of input in the given layer
    #L2 number of input in the given layer
    L1 = nn_input_dim
    L2 = neuron_number
    esp_init = weight_init(L1,L2)
    W1 = np.random.uniform(-esp_init,esp_init,[nn_input_dim,nn_hdim])
    L1 = neuron_number
    L2 = nn_output_dim
    esp_init = weight_init(L1,L2)
    W2 = np.random.uniform(-esp_init,esp_init,[nn_hdim,
nn_output_dim])
    # This is what we return at the end
    model = {}
    # Gradient descent. For each batch...
    for i in xrange(0, num_passes):
        # Forward propagation
        z1 = X_train.dot(W1)
        a1 = np.tanh(z1)
        z2 = a1.dot(W2)
        exp_scores = np.exp(z2)
        probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)
        num_ex = len(X_train)
        # Backpropagation
        delta3 = probs
        delta3[range(num_ex), y_train] -= 1
        dW2 = (a1.T).dot(delta3)
        delta2 = delta3.dot(W2.T) * (1 - np.power(a1, 2))# diff of
tanh -- in future i will use gradient desent to calculate this
        dW1 = np.dot(X_train.T, delta2)
        # Add regularization terms (b1 and b2 don't have
regularization terms)
        dW2 =dW2 + (reg_lambda * W2)
        dW1 = dW1 + (reg_lambda * W1)
        # Gradient descent parameter update
        W1 = W1 -(alpha * dW1)
        W2 = W2 -(alpha * dW2)
        # Assign new parameters to the model
        model = { 'W1': W1, 'W2': W2}
        status(i,model)
    return model

# Build a model

    model = build_model(neuron_number, print_loss=True)
    #test = "Heavy traffic at vest avenue"
predict(model, x)
y_pred = []
#predict(model, X_test[0])
for i in X_test:
    y_pred.append(predict(model,i))

# Compute confusion matrix
cnf_matrix = confusion_matrix(y_test, y_pred)

# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='Confusion matrix, without normalization')

accuracy_score(y_test, y_pred,normalize=False)
```

2. **tweet_cleanser.py**

```
# -*- coding: utf-8 -*
##   Tweet Reprocessing - To remove wide space and repetation,special
character

##   This function is used for location extraction purpose, but not
for
##   clasification

import re

def process_tweets(tweet):
    #
    #Convert www.* or https?://* to space
    tweet = re.sub('((www\.[^\s]+)|(https?://[^\s]+))','',tweet)
    #Convert @username to space
    tweet = re.sub('@[^\s]+',' ',tweet)
    #Remove additional white spaces
    tweet = re.sub('[\s]+', ' ', tweet)
    #Replace #word with word
    tweet = re.sub('#([^\s]+)', r'\1', tweet)
    #look for 2 or more repetitions of character and replace with the
character itself
    pattern = re.compile(r"(.)\1{1,}", re.DOTALL)
    tweet = pattern.sub(r"\1\1", tweet)
    #Remove ... , -, special character
    tweet = re.sub('[^A-Za-z]+', ' ', tweet)
    #trim
    tweet = tweet.strip('\'"')

    return tweet
#end Tweet_cleanser
if __name__ == "__main__":
     data = ["this is for test","John likes to watch movies. Mary
likes movies too.","John also likes to watch football games."]
     print ""
```

3. **tweet_read.py**

```
from preProcessing.tweet_cleanser import process_tweets

#csv file read
from pandas import read_csv

import sys

def status(i,num_passe):
    barLength = 20
    status = ""
    progress = (float(i)/(num_passe-1))
    block = int(round(barLength*progress))
    sys.stdout.write('\r')
    text = "[{0}] File Read {1}% Completed.".format( "#"*block + "-
"*(barLength-block), format(progress*100,".2f"),status)
    sys.stdout.write(text)
    sys.stdout.flush()

def importData():
    tweet_data = read_csv(filepath_or_buffer
="C:/Users/Ramanuja/Desktop/data2.csv",header=None,skiprows=2,usecols
= [9,10])# since no header info
    filter_data = []
    i = 0
    for text in tweet_data[9]:
        status(i,len(tweet_data[9]))
        i = i + 1
        filter_data.append(process_tweets(text))
    y = tweet_data[10]
    #vector = bagofWords(filter_data,1)
    print "File Read Operation Completed"
    return filter_data,y
if __name__ == "__main__":
     data = ["this is for test","John likes to watch movies. Mary
likes movies too.","John also likes to watch football games."]
     X,y =  importData()
```

4. **plot.py**

```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    Credit : Sklearn - http://scikit-
learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html#sphx-
glr-auto-examples-model-selection-plot-confusion-matrix-py
    """
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```