CERTIFICATE

This is to certify that **Deepansh Nagaria** (B.Tech CSE IIT Roorkee), **Roodram Paneri** (B.Tech CSE IIT Roorkee) and **Vishal Garg** (B.Tech CSE IIT Roorkee) did a Summer Research project on Use of Social Media data for transportation analysis to get traffic jam information during the Summer break(May 2018 - June 2018) under the guidance of **Dr. Durga Toshniwal** (Assistant Professor, Department of Computer Science, Indian Institute of Technology Roorkee).

SIGNATURE

(Dr. Durga Toshniwal)

Use of Social Media Data to Get Traffic Jam Information

Roodram Paneri, Deepansh Nagaria and Vishal Garg

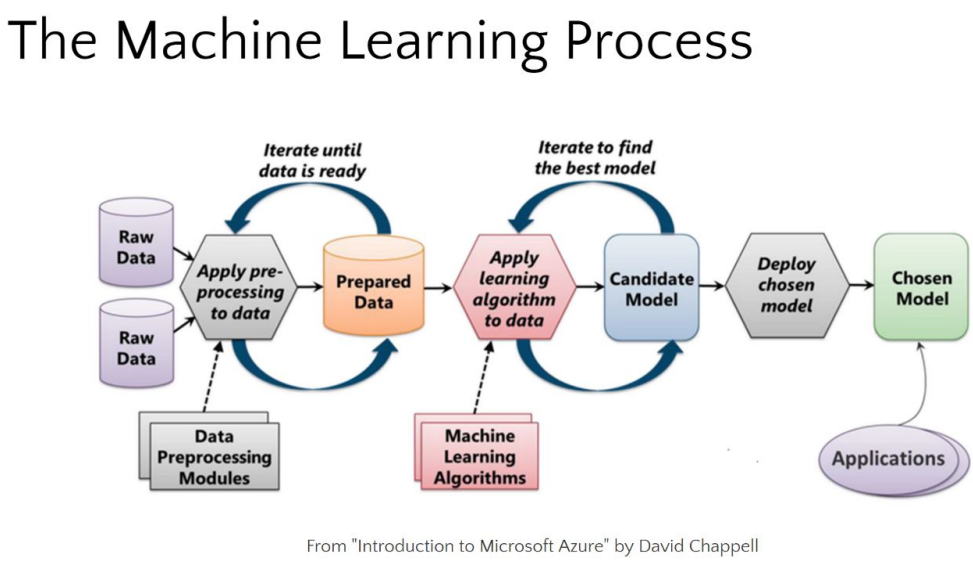
Introduction

Traffic information, such as traffic incidents and real-time traffic status, plays a fundamental role in improving the efficiency of Intelligent Transportation Systems (ITS). Conventionally, traffic information is obtained from physical sensors like GPS, cameras, or loop detectors. Recently Social media has also been regarded as the potential source to serve as social sensors to extract traffic information, since people and authoritative agencies often post transportation information online with the popularity of such platforms. As these platforms have a great number of real-time user generated contents, they have become powerful and inexpensive information sources. Nowadays use of social media is quite common and Twitter is one of the prominent social media sites currently. In 2018 more than 336 million active bloggers were recorded on Twitter monthly. The information these Tweets provide can be used for a variety of tasks such as prediction of a person's influence in an area, predicting weather, people's reviews, problems and disaster management.

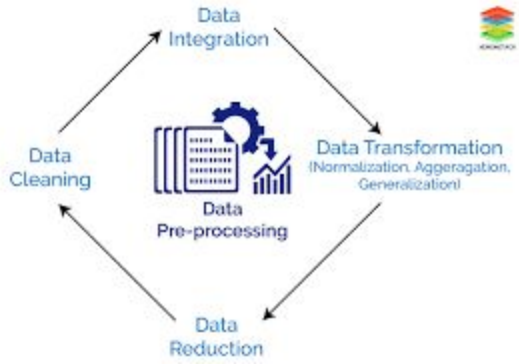
In field of transportation also there is a spike in the number of people using social media data for dealing with issues such as predicting traffic and accidents in an area. Earlier when twitter used to provide the coordinates of the blogger, work was done on reporting of accidents and issues causing traffic in that area. As now as per new terms and conditions, Twitter does not provide the coordinates of the blogger, it is quite a challenge to predict the area of the issue. Here, we have took aid of data mining, having data from twitter about transport, using India as bounding box, we filtered tweets such that each tweet we are classifying as traffic related or not has a mention of location in it. This solved the problem and made it easy for prediction of area from the blogger posted the tweet.

Hence, it is quite clear that social media data, here twitter, is an incredible source of information and this rich embedded information can help us improving traffic predictions. The Twitter information is both noisy and unstructured. An effective text mining method is necessary to extract the useful transport-related information from tweets. In this study, we employ and compare two deep learning methods: Deep Neural Network (DNN) and Long Short-Term Memory (LSTM), in training and classifying the accident-related tweets. Unlike classifiers such as logistic regression or Support Vector Machines (SVMs), deep learning does not seek direct functional relationships between the input features and the output classification results. Instead,

it is a set of machine learning algorithms that attempt to learn in multiple levels, corresponding to different levels of abstraction. The training process of DNN is divided into multiple layers, and the output result is expressed as a composition of layers, where the higher level features are the composition of lower-level features , giving the potential of modelling complex data with fewer units than a similarly performing shallow network. We have shown results with RNN (Recurrent Neural Network), DNN (with LSTM) and all basic algorithms including Logistic Regression, SVM, Naïve Bayes and Random Forest. Later we also used the latest Facebook library “FastText” for classification which provided excellent results as shown in coming sections.



Data Description and Preprocessing



- **Raw data preparation**

The dataset for the purpose of classification was prepared through the tweets collected from the twitter crawl API with a geo location filter. The tweets collection area was first limited using a bounding box for India, using geo-coordinate filtering. We were able to retrieve about 2 million tweets using the filter. Then we narrowed the time range for the month of April from 9th to 28th of the year 2018. Also the tweets specific to the purpose of transport were separated. The next step is to extract the candidate tweets that possibly describe the on-site traffic jams. Usually, these candidate tweets should contain one or more keywords such as “traffic” or “jam” or “stuck” or “congestion” that are jam-related. Hence,the final extraction of the tweets constituting the dataset was done by applying the filters of specific keywords that included “jam, traffic, congestion and stuck.” This resulted in extraction of about 2000 tweets across the country that could possibly describe traffic jams.

- **Preprocessing of tweet data**

The main aim of this study is to separate the tweets involving traffic jam from the tweets that involve related words but are not actually related to such situation. This enables us to identify the actual tweets related to jam and hence use the location data attached with tweet to pre-inform people who might get stuck in the similar position.

Hence, we classify the raw tweets obtained using the crawl into 2 parts

1. Related to or describing a jam
2. Not dealing with jam related situation.

➔ The class of tweets describing jam was labelled as ‘1’ while the other one was labelled as ‘0’. The tweets obtained as a result of crawling were then manually labelled in one or the other of the 2 classes 0 or 1.

➔ After the manual labelling, we obtained 603 tweets that were labelled as 1 and the rest as 0. So as to make the dataset to be fair, we separated 603 tweets of both types and these separated set of tweets was used for the application of various algorithms.

- **Structured database construction**

Now, the tweets in the dataset consisted of hashtags(‘#’) username mentions (‘@.....’) and even non-english words which don’t make any sense and hence are not useful for the purpose of classification.

So, every tweet string was passed through following steps to clean it:

- Convert the string to lower-case.
- Remove the punctuation marks (if any) from the string.
- Remove all the words which contain any other characters than the alphabets from the string.
- Remove all the stopwords, i.e. the words which are short enough to be ignored as well the parts the sentence that are meant for making the tweet grammatically correct instead of affecting it’s meaning.
- Each word of the string was stemmed to its base form so that the different forms of a word are considered same instead of different as the meaning that they provide to the sentence is same.

After passing through all these steps, the tweet was ready to be vectorized. Hence the set of tweet was vectorized accordingly using the TfIdf vectorizer and the vectorized data implemented into the algorithm.

Algorithms

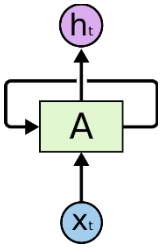
1. DNN + LSTM

Recurrent Neural Networks

Humans don't start their thinking from scratch every second. As you read this essay, you understand each word based on your understanding of previous words. You don't throw everything away and start thinking from scratch again. Your thoughts have persistence.

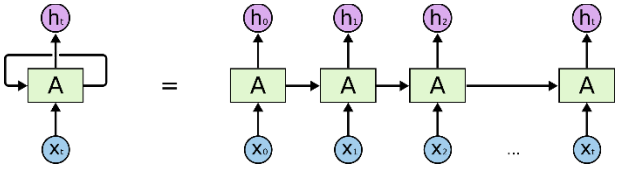
Traditional neural networks can't do this, and it seems like a major shortcoming. For example, imagine you want to classify what kind of event is happening at every point in a movie. It's unclear how a traditional neural network could use its reasoning about previous events in the film to inform later ones.

Recurrent neural networks address this issue. They are networks with loops in them, allowing information to persist.



Recurrent Neural Networks have loops.

In the above diagram, a chunk of neural network, A, looks at some input x_t and outputs a value h_t. A loop allows information to be passed from one step of the network to the next.
These loops make recurrent neural networks seem kind of mysterious. However, if you think a bit more, it turns out that they aren't all that different than a normal neural network. A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. Consider what happens if we unroll the loop:



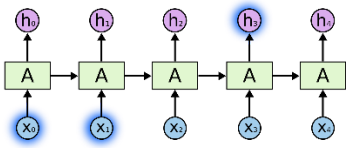
An unrolled recurrent neural network.

This chain-like nature reveals that recurrent neural networks are intimately related to sequences and lists. They're the natural architecture of neural network to use for such data.

Essential to these successes is the use of "LSTMs," a very special kind of recurrent neural network which works, for many tasks, much better than the standard version. Almost all exciting results based on recurrent neural networks are achieved with them. It's these LSTMs that this essay will explore.

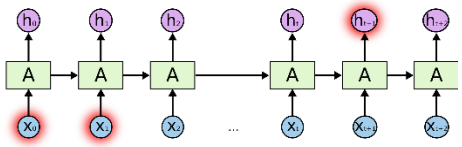
The Problem of Long-Term Dependencies

Sometimes, we only need to look at recent information to perform the present task. For example, consider a language model trying to predict the next word based on the previous ones. If we are trying to predict the last word in “the clouds are in the *sky*,” we don’t need any further context – it’s pretty obvious the next word is going to be sky. In such cases, where the gap between the relevant information and the place that it’s needed is small, RNNs can learn to use the past information.



But there are also cases where we need more context. Consider trying to predict the last word in the text “I grew up in France… I speak fluent *French*.” Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of France, from further back. It’s entirely possible for the gap between the relevant information and the point where it is needed to become very large.

Unfortunately, as that gap grows, RNNs become unable to learn to connect the information.



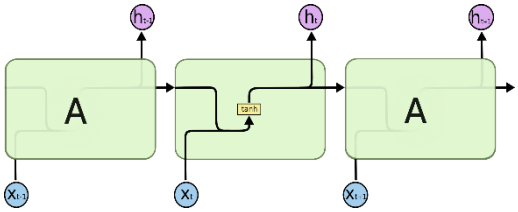
In theory, RNNs are absolutely capable of handling such “long-term dependencies.” A human could carefully pick parameters for them to solve toy problems of this form. Sadly, in practice, RNNs don’t seem to be able to learn them. Thankfully, LSTMs don’t have this problem!

LSTM Networks

Long Short Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies. They work tremendously well on a large variety of problems, and are now widely used.

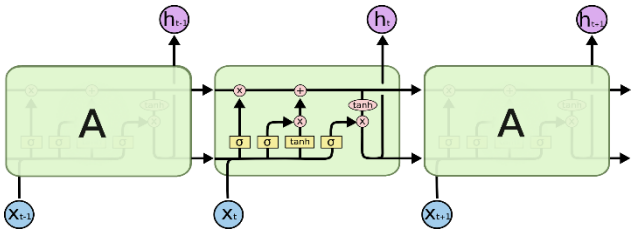
LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behaviour, not something they struggle to learn!

All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tan h layer.



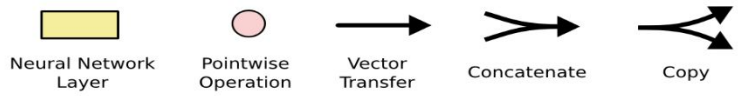
The repeating module in a standard RNN contains a single layer.

LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.



The repeating module in an LSTM contains four interacting layers.

Don't worry about the details of what's going on. We'll walk through the LSTM diagram step by step later. For now, let's just try to get comfortable with the notation we'll be using.

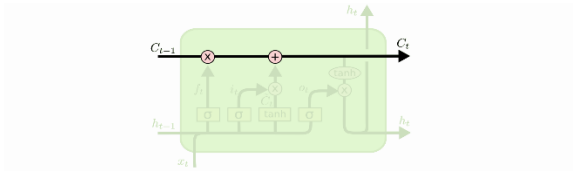


In the above diagram, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denote its content being copied and the copies going to different locations.

The Core Idea Behind LSTMs

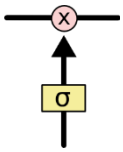
The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.

The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged.



The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.

Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.

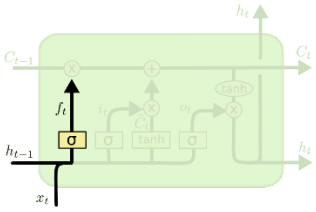


The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means “let nothing through,” while a value of one means “let everything through!”

An LSTM has three of these gates, to protect and control the cell state.

Step-by-Step LSTM Walk Through

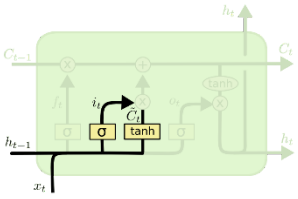
The first step in our LSTM is to decide what information we’re going to throw away from the cell state. This decision is made by a sigmoid layer called the “forget gate layer.” It looks at h_{t-1} and x_t, and outputs a number between 0 and 1 for each number in the cell state C_{t-1}. A represents “completely keep this” while a 0 represents “completely get rid of this.”
Let’s go back to our example of a language model trying to predict the next word based on all the previous ones. In such a problem, the cell state might include the gender of the present subject, so that the correct pronouns can be used. When we see a new subject, we want to forget the gender of the old subject.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

The next step is to decide what new information we’re going to store in the cell state. This has two parts. First, a sigmoid layer called the “input gate layer” decides which values we’ll update. Next, a tan h layer creates a vector of new candidate values, $C_{\sim t}$, that could be added to the state. In the next step, we’ll combine these two to create an update to the state.

In the example of our language model, we’d want to add the gender of the new subject to the cell state, to replace the old one we’re forgetting.

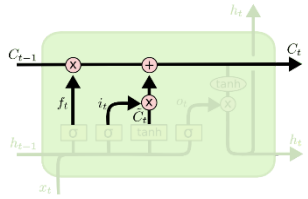


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

It’s now time to update the old cell state, Ct−1Ct−1, into the new cell state C_t. The previous steps already decided what to do, we just need to actually do it.

We multiply the old state by f_t, forgetting the things we decided to forget earlier. Then we add $i_t * C_t$. This is the new candidate values, scaled by how much we decided to update each state value.

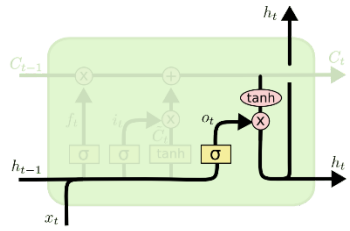
In the case of the language model, this is where we'd actually drop the information about the old subject's gender and add the new information, as we decided in the previous steps.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version. First, we run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through tan h (to push the values to be between −1−1 and 11) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.

For the language model example, since it just saw a subject, it might want to output information relevant to a verb, in case that's what is coming next. For example, it might output whether the subject is singular or plural, so that we know what form a verb should be conjugated into if that's what follows next.



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

2. Support Vector Machines (SVM) algorithm

A Support Vector Machine (SVM) is a supervised machine learning algorithm that can be employed for both classification and regression purposes. SVMs are more commonly used in classification problems and as such, this is what we will focus on in this post. SVMs are based on the idea of finding a hyperplane that best divides a dataset into two classes.

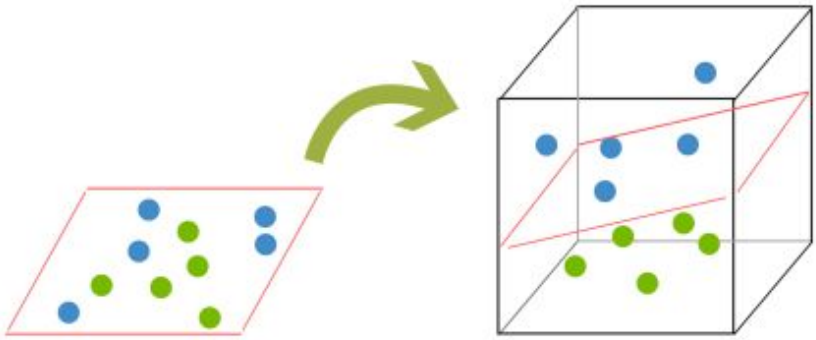
Support Vectors

Support vectors are the data points nearest to the hyperplane, the points of a data set that, if removed, would alter the position of the dividing hyperplane. Because of this, they can be considered the critical elements of a data set.

Algorithm

The goal is to choose a hyperplane with the greatest possible margin between the hyperplane and any point within the training set, giving a greater chance of new data being classified correctly.in three dimensions, our hyperplane can no longer be a line. It

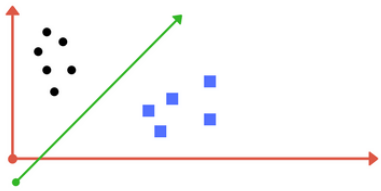
must now be a plane. The idea is that the data will continue to be mapped into higher and higher dimensions until a hyperplane can be formed to segregate it.



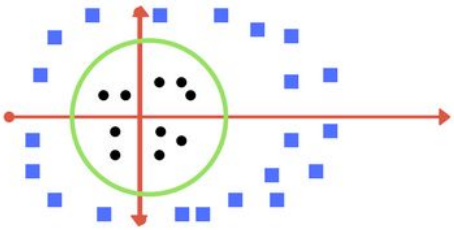
Parameters

Varying these parameters we can achieve considerable non linear classification line with more accuracy in reasonable amount of time.

1. **Kernel =** A kernel is a function in which the problem is transformed to facilitate learning of the hyperplane. It defines whether we want a linear or linear separation or polynomial or exponential hyperplane for separation of various classes.
    - **Linear kernel => *f(x) = B(0) + sum(ai * (x,xi))***



    - **Polynomial kernel => *K(x,xi) = 1 + sum(x * xi)^d***



    - **Exponential kernel => *K(x,xi) = exp(-gamma * sum((x — xi²))***

    These equations involve calculating the inner products of a new input vector (x) with all support vectors in training data.

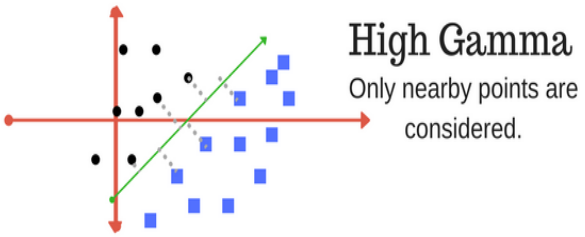
X = input

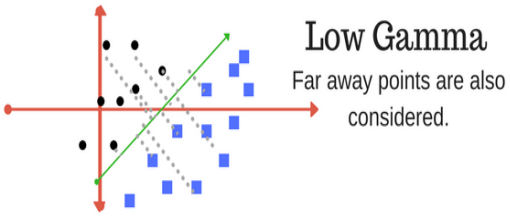
Xi = support vector

The coefficients B0 and ai (for each input) must be estimated from the training data by the learning algorithm.

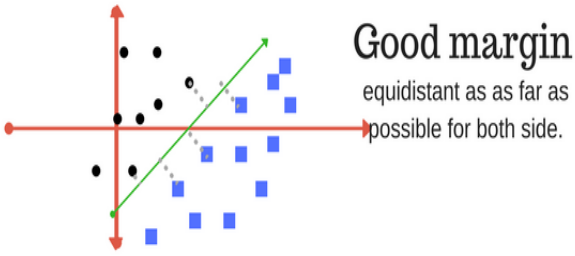
2. **Regularization =** The Regularization parameter (often termed as C parameter in python's sklearn library) tells the SVM optimization how much you want to avoid misclassifying each training example. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly and vice versa.

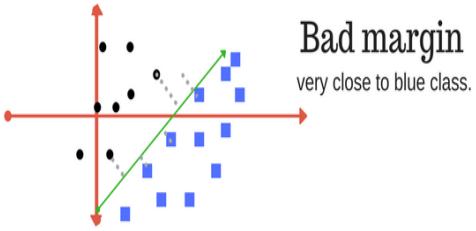
3. **Gamma =** The gamma parameter defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'.





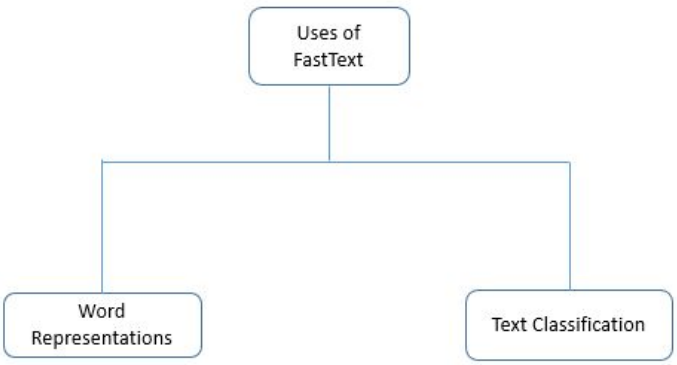
4. **Margin =** A margin is a separation of line to the closest class points.





3. FastText Algorithm/Library

FastText is a library created by the Facebook Research Team for efficient learning of word representations and sentence classification.



1. It is helpful to find the vector representation for rare words. Since rare words could still be broken into character n-grams, they could share these n-grams with the common words. For example, for a model trained on a news dataset, the medical terms eg: diseases can be the rare words.
2. It can give the vector representations for the words not present in the dictionary (OOV words) since these can also be broken down into character n-grams. word2vec and glove both fail to provide any vector representations for words not in the dictionary.
3. For example, for a word like *stupedofantabulouslyfantastic,* which might never have been in any corpus, gensim might return any two of the following solutions –
    a. zero vector
    b. a random vector with low magnitude.

But FastText can produce vectors better than random by breaking the above word in chunks and using the vectors for those chunks to create a final vector for the word. In this particular case, the final vector might be closer to the vectors of fantastic and fantabulous.

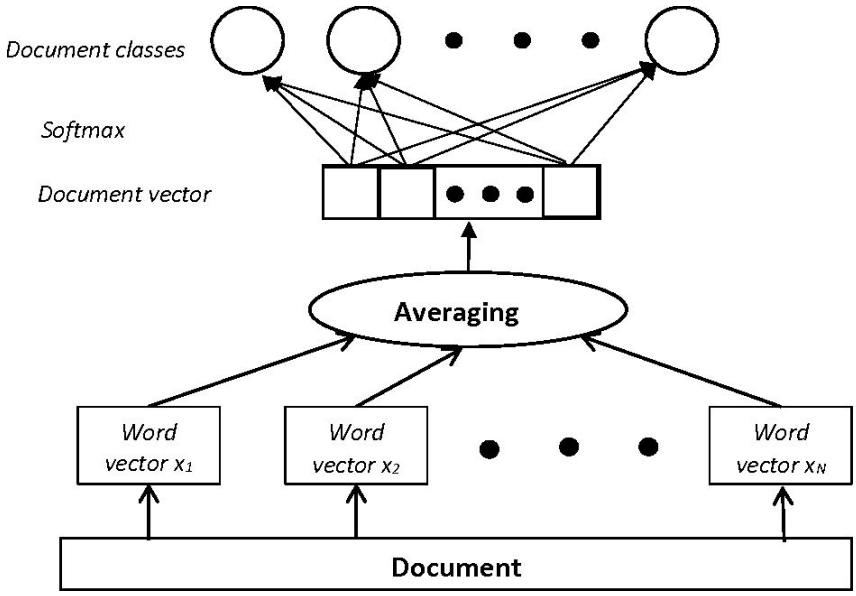
Implementation

Learning Word Representations

Words in their natural form cannot be used for any Machine Learning task in general. One way to use the words is to transform these words into some representations that capture some attributes of the word. It is analogous to describing a person as – [‘height’:5.10 ,’weight’:75, ‘colour’:’dusky’, etc.] where height, weight etc are the attributes of the person. Similarly, word representations capture some abstract attributes of words

in the manner that similar words tend to have similar word representations. There are primarily two methods used to develop word vectors – Skipgram and CBOW. Fasttext can use both these ways to develop word vectors. Then the trained vector model can be put to many uses such as, finding vectors of various words, similar words of a given word, getting analogies of various word groups and Text Classification, which is our main concern.

Text Classification



The default format of text file on which we want to train our model should be

_ _ label _ _ <X> <Text>

Where _ _label_ _ is a prefix to the class and <X> is the class assigned to the document. Also, there should not be quotes around the document and everything in one document should be on one line. If the file is not in the required format, we can use **-label** to specify labels. This argument takes care of the format of the label specified. Then, we can train the character embeddings model on the dataset using the following parameters and use it for getting accuracy, prediction, and even vectors of the various words using this model.

- **lr** : learning rate [0.1]
- **lrUpdateRate** : change the rate of updates for the learning rate [100]
- **dim** : size of word vectors [100]
- **ws** : size of the context window [5]
- **epoch** : number of epochs [5]

- **neg** : number of negatives sampled [5]
- **loss** : loss function {ns, hs, softmax} [ns]
- **thread** : number of threads [12]
- **pretrainedVectors** : pretrained word vectors for supervised learning []
- **saveOutput** : whether output params should be saved [0]

4. Naive Bayes Algorithm

In machine learning we are often interested in selecting the best hypothesis (h) given data (d).In a classification problem, our hypothesis (h) may be the class to assign for a new data instance (d).One of the easiest ways of selecting the most probable hypothesis given the data that we have that we can use as our prior knowledge about the problem. Bayes' Theorem provides a way that we can calculate the probability of a hypothesis given our prior knowledge.
Bayes' Theorem is stated as:
P(h|d) = (P(d|h) * P(h)) / P(d)

- **P(h|d)** is the probability of hypothesis h given the data d. This is called the posterior probability.

- **P(d|h)** is the probability of data d given that the hypothesis h was true.

- **P(h)** is the probability of hypothesis h being true (regardless of the data). This is called the prior probability of h.
- **P(d)** is the probability of the data (regardless of the hypothesis).

After calculating the posterior probability for a number of different hypotheses, we can select the hypothesis with the highest probability. This is the maximum probable hypothesis and may formally be called the maximum a posteriori (MAP) hypothesis.
MAP(h) = max(P(h|d))
MAP(h) = max(P(d|h) * P(h))

The P(d) is a normalizing term which allows us to calculate the probability. We can drop it when we are interested in the most probable hypothesis as it is constant and only used to normalize.
Back to classification, if we have an even number of instances in each class in our training data, then the probability of each class (e.g. P(h)) will be equal. Again, this would be a constant term in our equation and we could drop it so that we end up with:
MAP(h) = max(P(d|h))
Naive Bayes is a classification algorithm for binary (two-class) and multi-class classification problems. The technique is easiest to understand when described using binary or categorical input values.
It is called *naive Bayes* because the calculation of the probabilities for each hypothesis are simplified to make their calculation tractable. Rather than attempting to calculate the values of each attribute value P(d1, d2, d3|h), they are assumed to be conditionally independent given the target value and calculated as P(d1|h) * P(d2|H) and so on.

This is a very strong assumption that is most unlikely in real data, i.e. that the attributes do not interact. Nevertheless, the approach performs surprisingly well on data where this assumption does not hold.
Training is fast because only the probability of each class and the probability of each class given different input (x) values need to be calculated. No coefficients need to be fitted by optimization procedures.

Gaussian Naive Bayes

Naive Bayes can be extended to real-valued attributes, most commonly by assuming a Gaussian distribution.
This extension of naive Bayes is called Gaussian Naive Bayes. Other functions can be used to estimate the distribution of the data, but the Gaussian (or Normal distribution) is the easiest to work with because you only need to estimate the mean and the standard deviation from your training data.

5. Random Forest Classifier Algorithm

Random Forest Classifier is ensemble algorithm. ***Ensembled algorithms*** are those which combines more than one algorithms of same or different kind for classifying objects. For example, running prediction over Naive Bayes, SVM and Decision Tree and then taking vote for final consideration of class for test object.
Random forest classifier creates a set of decision trees from randomly selected subset of training set. It then aggregates the votes from different decision trees to decide the final class of the test object.

Each tree is grown as follows:

If the number of cases in the training set is N, sample N cases at random - but with replacement, from the original data. This sample will be the training set for growing the tree.
If there are M input variables, a number m<<M is specified such that at each node, m variables are selected at random out of the M and the best split on these m is used to split the node. The value of m is held constant during the forest growing.
Each tree is grown to the largest extent possible. There is no pruning.
In the original paper on random forests, it was shown that the forest error rate depends on two things:
The correlation between any two trees in the forest. Increasing the correlation increases the forest error rate.
The strength of each individual tree in the forest. A tree with a low error rate is a strong classifier. Increasing the strength of the individual trees decreases the forest error rate.
Reducing m reduces both the correlation and the strength. Increasing it increases both. Somewhere in between is an "optimal" range of m - usually quite wide. Using the oob error rate (see below) a value of m in the range can quickly be found. This is the only adjustable parameter to which random forests is somewhat sensitive.

The out-of-bag (oob) error estimate

In random forests, there is no need for cross-validation or a separate test set to get an unbiased estimate of the test set error. It is estimated internally, during the run, as follows:

Each tree is constructed using a different bootstrap sample from the original data. About one-third of the cases are left out of the bootstrap sample and not used in the construction of the kth tree.

Put each case left out in the construction of the kth tree down the kth tree to get a classification. In this way, a test set classification is obtained for each case in about one-third of the trees. At the end of the run, take j to be the class that got most of the votes every time case n was oob. The proportion of times that j is not equal to the true class of n averaged over all cases is the oob error estimate. This has proven to be unbiased in many tests.

Suppose training set is given as : [X1, X2, X3, X4] with corresponding labels as [L1, L2, L3, L4], random forest may create three decision trees taking input of subset for example,

1.
2. [X1, X2, X3]
3.
4. [X1, X2, X4]
5.
6. [X2, X3, X4]

So finally, it predicts based on the majority of votes from each of the decision trees made.
This works well because a single decision tree may be prone to a noise, but aggregate of many decision trees reduce the effect of noise giving more accurate results.

Alternative implementation for voting

Alternatively, the random forest can apply weight concept for considering the impact of result from any decision tree. Tree with high error rate are given low weight value and vise versa. This would increase the decision impact of trees with low error rate.

Basic parameters to Random Forest Classifier can be total number of trees to be generated and decision tree related parameters like minimum split, split criteria etc.
some of the tuning parameters in sklearn are:

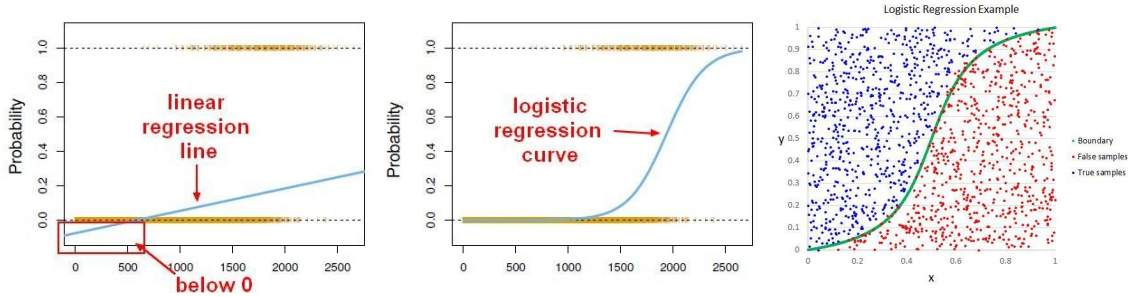
n_estimators : Number of trees in forest. a higher number of trees increases the performance and makes the predictions more stable, but it also slows down the computation.
max_features : which is the maximum number of features Random Forest is allowed to try in an individual tree.
min_samples_split : minimum number of working set size at node required to split. Default is 2.
max_depth : The maximum depth of the tree.
n_jobs : hyperparameter tells the engine how many processors it is allowed to use.
The main limitation of Random Forest is that a large number of trees can make the algorithm to slow and ineffective for real-time predictions. In general, these algorithms are fast to train, but quite slow to create predictions once they are trained. A more accurate prediction requires more trees, which results in a slower model. In most real-world applications the random forest algorithm is fast enough, but there can certainly be situations where run-time performance is important and other approaches would be preferred.
Overall, Random Forest is a (mostly) fast, simple and flexible tool, although it has its limitations.

6. Linear Regression

Logistic regression is a statistical method for analysing a dataset in which there are one or more independent variables that determine an outcome. The outcome is measured with a dichotomous variable (in which there are only two possible outcomes).

Logistic Regression is another technique borrowed by machine learning from the field of statistics. It is the go-to method for binary classification problems (problems with two class values).

Logistic regression is like linear regression in that the goal is to find the values for the coefficients that weight each input variable. Unlike linear regression, the prediction for the output is transformed using a non-linear function called the logistic function. The logistic function looks like a big S and will transform any value into the range 0 to 1. This is useful because we can apply a rule to the output of the logistic function to snap values to 0 and 1 (e.g. IF less than 0.5 then output 1) and predict a class value. Because of the way that the model is learned, the predictions made by logistic regression can also be used as the probability of a given data instance belonging to class 0 or class 1. This can be useful on problems where you need to give more rationale for a prediction. Like linear regression, logistic regression does work better when you remove attributes that are unrelated to the output variable as well as attributes that are very similar (correlated) to each other. It's a fast model to learn and effective on binary classification problems.



Methodology

1. Random Forest Classifier

After understanding the working of Random Forest Classifier and continuing from where we left during the data preparation section, let us move on to the application methodology of the Random Forest Classification Algorithm.

- ➔ After the vectorization of data the next step is to separate the data into training and testing sets which can be done by train test split available with sklearn.
- ➔ Define the model with an optional mention of the parameters described above or use the default ones.
- ➔ Fit the training set to the model.
- ➔ Predict the values corresponding to the Xdata of the training set.
- ➔ Use the accuracy_score function from sklearn to calculate accuracy using predicted and actual y values.
- ➔ Similarly get confusion_matrix function to get confusion matrix in similar way as above.
- ➔ Use roc_curve function to get true and false positive rates and calculate value of area under the curve using “auc” function.
- ➔ Use matplotlib function to plot the roc curve.

The above algorithm completely describes the process used for implementing the random forest classifier for the given data.

2.DNN with LSTM

- ➔ The data is separated into training and testing sets which can be done by train test split available in sklearn.
- ➔ After loading the dataset top n words are kept and the rest are valued 0.
- ➔ Dataset is padded to a maximum review length in words(we set max_length=150).
- ➔ Sequential Neural network is build with following 3 hidden layers :
- ➔ Embedding layer with 32 cells and input length =max_length
- ➔ LSTM layer with 128 cells and dropout=0.8.
- ➔ Dense layer with 250 cells with relu activation
- ➔ Data from Dense layer is added to a cell with sigmoid activation. (Adam optimizer can be used)
- ➔ Fit the training set to the model.
- ➔ Predict the values corresponding to the Xdata of the training set.
- ➔ Use the accuracy_score function from sklearn to calculate accuracy using predicted and actual y values.
- ➔ Similarly get confusion_matrix function to get confusion matrix in similar way as above.

- ➔ Use roc_curve function to get true and false positive rates and calculate value of area under the curve using “auc” function.
- ➔ Use matplotlib function to plot the roc curve.

Results

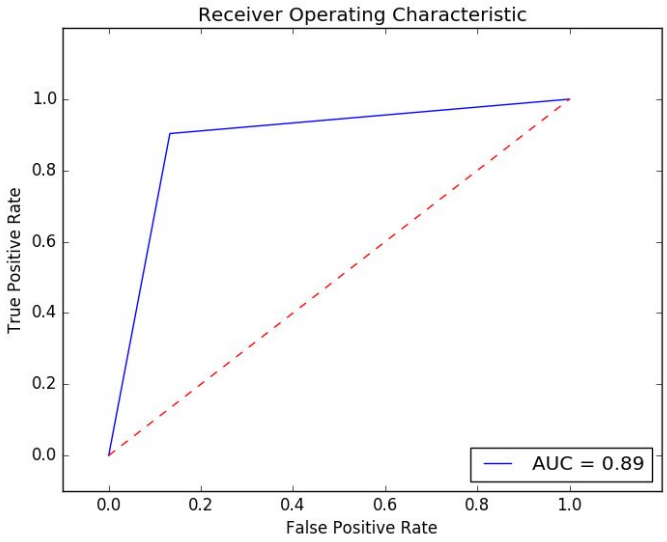
Algorithm	Accuracy	Confusion Matrix	Area under ROC Curve
Fasttext	0.8854625550660793	[[98 15] [11 103]]	0.89
Random Forest Classifier	0.8455882352941176	[[116 24] [18 114]]	0.85
DNN	0.8622589531680441	[[152 29] [21 161]]	0.85
DNN + LSTM	0.86	[[135 15] [27 125]]	0.86
SVM	0.84	[[123 27] [22 130]]	0.84
Naive-Bayes	0.81	[[104 46] [10 142]]	0.81
Logistic Regression	0.8213	[[126 24] [30 122]]	0.82

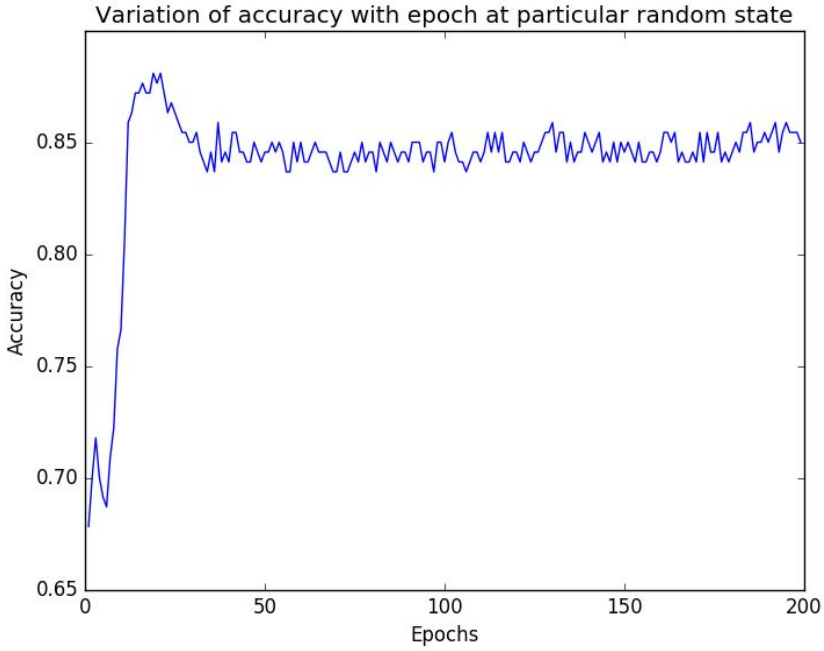
1. Fasttext

Maximum Accuracy achieved = 0.8854625550660793

Corresponding Confusion matrix = [[98 15] [11 103]]

Corresponding ROC curve is also shown



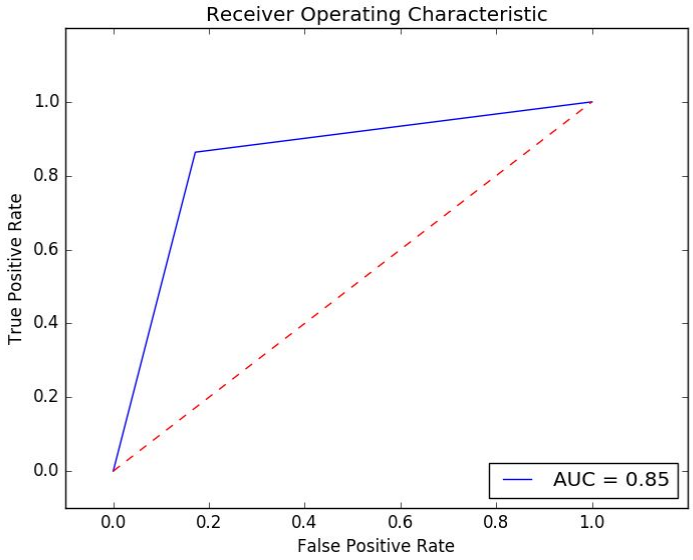


2. RandomForestClassifier

Maximum Accuracy Achieved = 0.8455882352941176

Corresponding Confusion Matrix = [[116 24] [18 114]]

ROC curve is shown in the figure

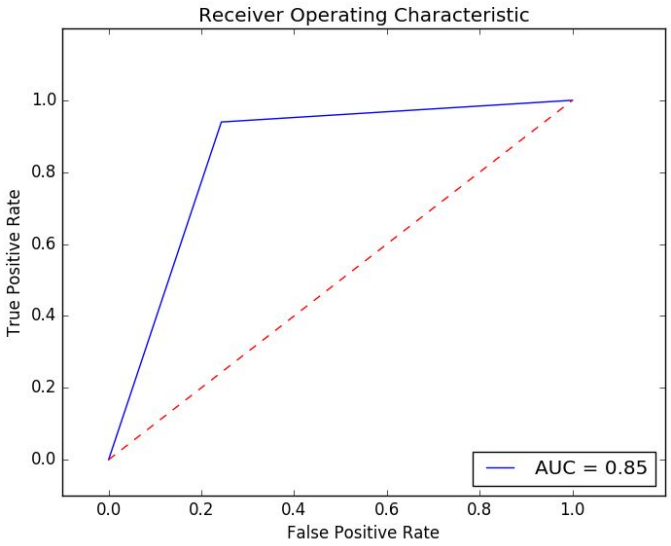


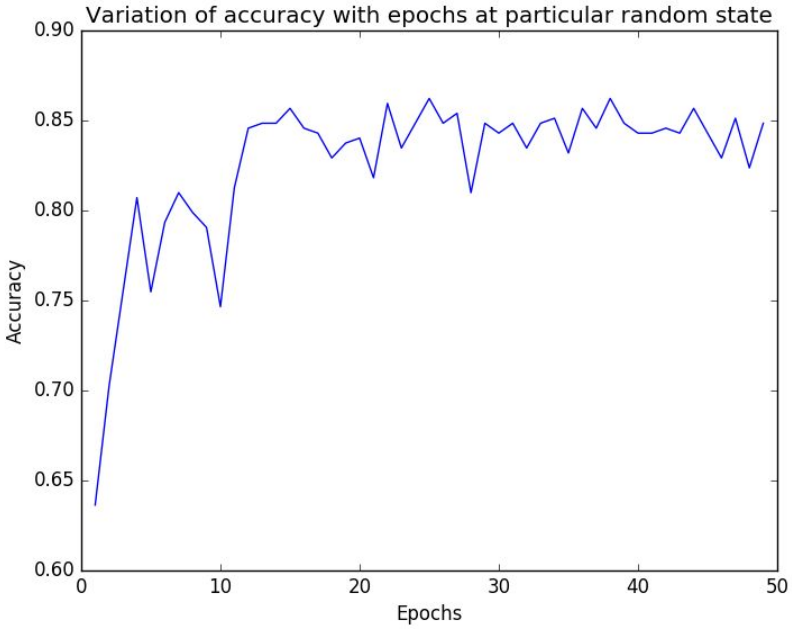
3. DNN (without LSTM)

Maximum Accuracy achieved =0.8622589531680441

Corresponding Confusion matrix = [[152 29] [21 161]]

Corresponding ROC curve is also shown



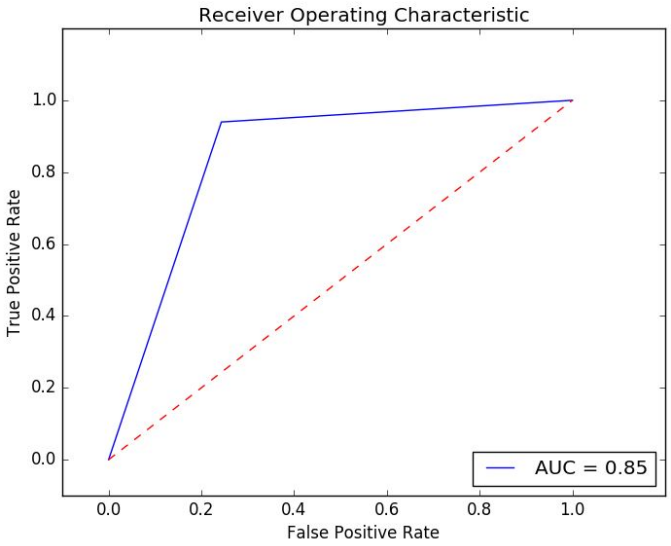


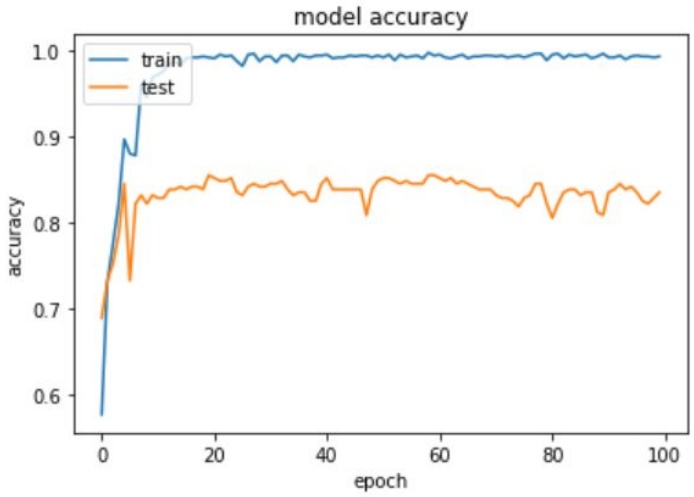
4. DNN+LSTM

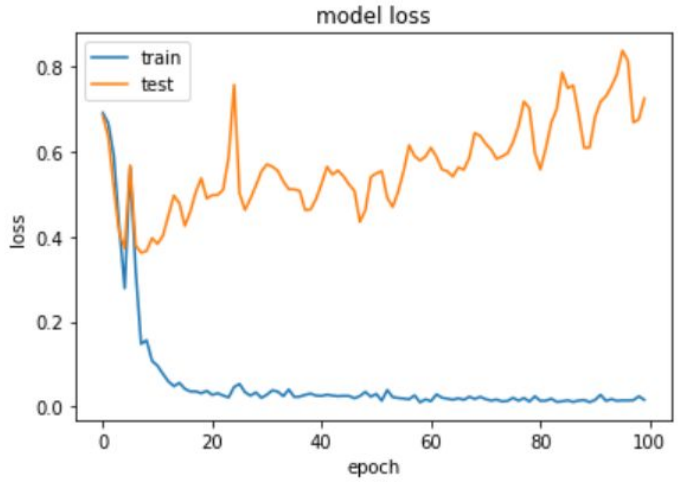
Maximum Accuracy achieved = 0.86

Corresponding Confusion matrix = [[135 15] [18 134]]

Corresponding ROC curve is also shown





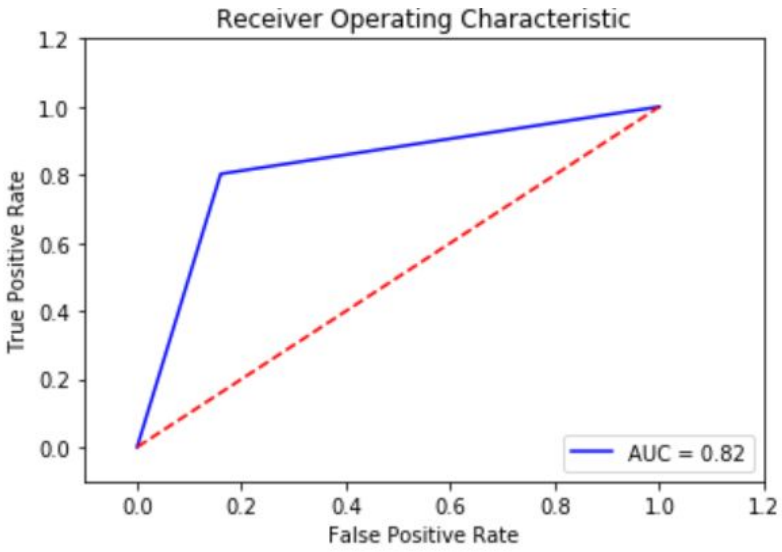


5. Logistic Regression

Maximum Accuracy achieved = 0.8213

Corresponding Confusion matrix = [[126 24] [30 122]]

Corresponding ROC curve is also shown

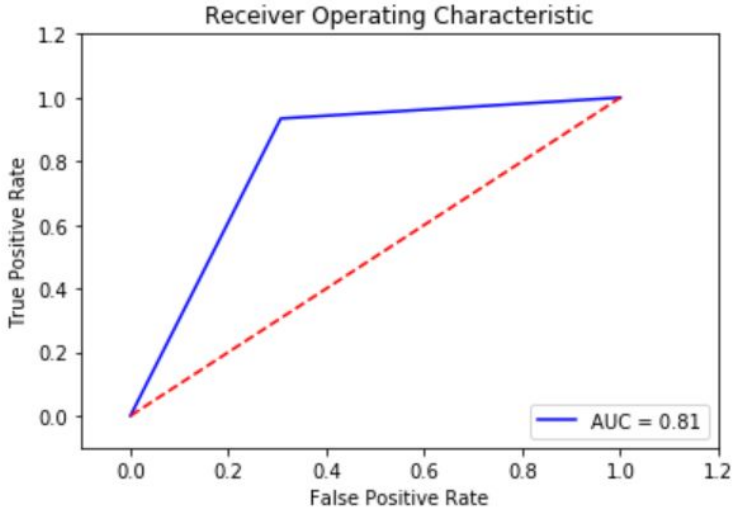


6. Naïve Bayes

Maximum Accuracy achieved = 0.81

Corresponding Confusion matrix = [[104 46] [10 142]]

Corresponding ROC curve is also shown

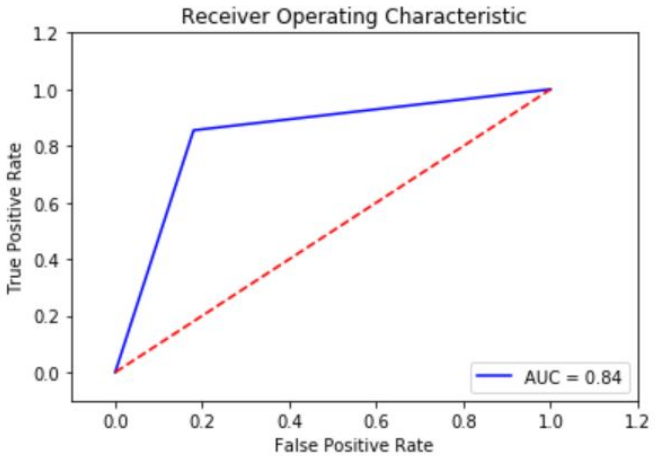


7. Support Vector Machines (SVM)

Maximum Accuracy achieved = 0.84

Corresponding Confusion matrix = [[123 27] [22 130]]

Corresponding ROC curve is also shown



Acknowledgement

We are extremely thankful to Prof. Durga Toshniwal who gave us a wonderful learning opportunity in the form of this machine/deep learning project. We are also thankful to Mr. Amit Agarwal who guided us throughout this project and without him this project could never have been accomplished to the extent it has.

Conclusion and Future Possibilities

So, by showcasing the models above along with their corresponding accuracies it is quite clear that the social media data is quite accurate in predicting traffic related information. Our work is new in the sense that no one has ever worked on Indian transport dataset as tweets in India are considered as quite noisy and they are often in local languages. We have showcased acceptable results considering the amount of noise and other factors in the dataset.The results clearly show that Deep Neural Network work better than normal classification algorithms such as SVM, Naive Bayes or Logistic Regression.Further, FastText, which works on character embedding basically provide great results and in fact the highest among all.

This study was completed using only the tweets that were in English language, were included in the dataset. There is an scope of improvement in the dataset and the tweets in

languages other than English may also be included using the Google translate tool to translate the tweets to English. Further we can use this data obtained to classify normal location related tweets as traffic related or not and hence finding traffic density in various areas which can ease planning of travelling root. Further extension in the work can be classifying the cause of congestion. The field of use of social media data in transport is quite huge and there are always scope for more ideas.

References

Francis, R.C., et al., Object tracking and management system and method using radiofrequency identification tags. 2003, Google Patents.

Abedin, B., A. Babar, and A. Abbasi. Characterization of the Use of Social Media in Natural Disasters: A Systematic Review. in Big Data and Cloud Computing (BdCloud), 2014 IEEE Fourth International Conference on. 2014. IEEE.

Cameron, M.A., et al. Emergency situation awareness from twitter for crisis management. in Proceedings of the 21st international conference companion on World Wide Web. 2012. ACM.

Gao, H., et al. Exploring temporal effects for location recommendation on location-based social networks. in Proceedings of the 7th ACM conference on Recommender systems. 2013. ACM.

Ukkusuri, S.V., et al., Use of Social Media Data to Explore Crisis Informatics. Transportation Research Record: Journal of the Transportation Research Board, 2014. 2459(1): p. 110-118.

Steur, R., Twitter as a spatio-temporal source for incident management. 2015.

Fu, K., R. Nune, and J.X. Tao. Social Media Data Analysis for Traffic Incident Detection and Management. in Transportation Research Board 94th Annual Meeting. 2015.

X. Wang, X. Zheng, Q. Zhang, T. Wang, and D. Shen, “Crowdsourcing in its: The state of the work and the networking,” IEEE Transactions on Intelligent Transportation Systems, vol. 17, no. 6, pp. 1596–1605,

2016.

S. E. Middleton, L. Middleton, and S. Modafferi, “Real-time crisis mapping of natural disasters using social media,” IEEE Intelligent Systems, vol. 29, no. 2, pp. 9–17, 2014.

E. D’Andrea, P. Ducange, B. Lazzerini, and F. Marcelloni, “Real-time detection of traffic from twitter stream analysis,” IEEE transactions on intelligent transportation systems, vol. 16, no. 4, pp. 2269–2283, 2015.

S. Zhang, J. Tang, H. Wang, and Y. Wang, “Enhancing traffic incident detection by using spatial point pattern analysis on social media,” Transportation Research Record: Journal of the Transportation Research Board, no. 2528, pp. 69–77, 2015.

M. Ni, Q. He, and J. Gao, “Using social media to predict traffic flow under special event conditions,” in Transportation Research Board 93rd Annual Meeting, no. 14-3315, 2014.

N. Kalchbrenner, E. Grefenstette, and P. Blunsom, “A convolutional neural network for modelling sentences,” arXiv preprint arXiv: 1404.2188, 2014.

C.-C. Chang and C.-J. Lin, “Libsvm: a library for support vector machines,” ACM Transactions on Intelligent Systems and Technology (TIST), vol. 2, no. 3, p. 27, 2011.

Graves, A., 2012. Supervised sequence labelling, supervised sequence labelling with recurrent neural networks. Springer, pp. 5–13.

Graves, A., Mohamed, A.-R., Hinton, G., 2013. Speech recognition with deep recurrent neural networks. In: Acoustics, speech and signal processing (ICASSP), 2013 IEEE international conference on. IEEE, pp. 6645–6649.

Cottrill, C.D., Derrible, S., 2015. Leveraging big data for the development of transport sustainability indicators. J. Urban Technol. 22 (1), 45–64

Adler, J., Horner, J., Dyer, J., Toppen, A., Burgess, L., Hatcher, G., 2015. Using Crowdsourced Data from Social Media to Enhance TMC Operations, vol. FHWAJPO-14-165. Federal Highway Administration.

Ahmed, S.A., Cook, A.R., 1980. Time series models for freeway incident detection. Transport. Eng. J. Am. Soc. Civil Eng. 106 (6), 731–745.

Mcauliffe, J.D., Blei, D.M., 2008. Supervised topic models. In: Advances in Neural Information Processing Systems, pp. 121–128.

Sethi, V., Bhandari, N., Koppelman, F.S., Schofer, J.L., 1995. Arterial incident detection using fixed detector and probe vehicle data. Transport. Res. Part C:
Emerg. Technol. 3 (2), 99–112.

S. Bhosale and S. Kokate, "Traffic Detection Using Tweets on Twitter Social Network" International Journal, Volume 4 (7).

"The Demographics of Social Media Users in 2016" , 2016. Available: http://www.thinkdigitalfirst.com/2016/01/04/the-demographics-of-socialmedia-users-in-2016/.

Carlos Gutierrez, Paulo Figuerias, Pedro Oliveira, Ruben Costa and Ricardo Jardim-Goncalves, "Twitter mining for traffic events detection," in Jul 1, 2015, pp. 371.

J. Weng and B. Lee, "Event Detection in Twitter." Icwsm, vol. 11, pp. 401-408, 2011.

A. Schulz, P. Ristoski, H. Paulheim, T. U. Darmstadt and T. Lab. I see a car crash: Real-time detection of small scale incidents in microblogs.

.