REAL-TIME ROAD TRAFFIC EVENTS DETECTION AND GEO-PARSING

A Thesis

Submitted to the Faculty

of

Purdue University

by

Saurabh Kumar

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Electrical and Computer Engineering

August 2018

Purdue University

Indianapolis, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Sarah Koskie, Chair

Department of Electrical and Computer Engineering

Dr. Brian King

Department of Electrical and Computer Engineering

Dr. Xiao Luo

Department of Computer Information Technology

Approved by:

Dr. Brian King

Head of the Graduate Program

I dedicate this thesis to my dear mother Mrs. Manju Gupta who always gave priority to education other than anything else.

ACKNOWLEDGMENTS

I would like to thank Dr. Sarah Koskie for her guidance and Dr. Brian King for his valuable suggestions. Also, I would like to thank my brother Rahul for helping me to improve upon the presentation of this thesis.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

SYMBOLS

$\exp(x)$	e^x
$\mathbf{F}$	Feature Vector
$\mathbf{F}[j]$	Feature Vector at j^{th} Layer
$\mathbf{F}^i$	Feature Vector i^{th} Component
$g(x)$	Activation Function
$h(\mathbf{w})$	Hypothesis Function
$\mathbf{I}$	Internal State Vector for RNN
$\overrightarrow{\mathbf{I}}[t]$	Internal State Vector for Forward RNN Layer at Layer T
$\overleftarrow{\mathbf{I}}[t]$	Internal State Vector for Backward RNN Layer at Layer T
$J(\mathbf{w})$	Cost Function
$\mathbf{M_i^j}$	Word Vector M_i j^{th} Component
$\mathbf{M_i}$	Vector Representation of Word W_i
$\mathbf{S}$	State Vector for NER
W_i	Word Representation
$\mathbf{w}$	Weight Vector
$\mathbf{w}^i_{[j]}$	Weight Vector i^{th} Component at j^{th} Layer
$\mathbf{w}[j]$	Weight Vector j^{th} Layer
$\overrightarrow{\mathbf{w}}$	Weight Vector for Forward RNN Layer
α	Learning Rate
β	Decay Factor
$\Theta(x)$	Sigmoid Activation Function

ABBREVIATIONS

API	Application Programming Interface
FFNN	Feed Forward Neural Network
JSON	JavaScript Object Notation
LIDAR	Light Detection and Ranging
LSTM	Long Short-Term memory
MIMO	Multiple Input Multiple Output
MISO	Multiple Input Single Output
NER	Named Entity Recognition
PASM	Publish and Subscriber Model
POS	Part of Speech
RDBMS	Relational Database Management System
RNN	Recurrent Neural Network
SVM	Support Vector Machine

GLOSSARY

Corpus	Collection of documents
Google Maps	Online Mapping Service Developed by Google
Tweet	Twitter user's post
Twitter	Online News and Social Networking Platform
Waze	Navigation Application for Smartphone

ABSTRACT

Kumar, Saurabh. M.S.E.C.E., Purdue University, August 2018. Real-Time Road Traffic Events Detection and Geo-Parsing. Major Professor: Sarah Koskie.

In the 21st century, there is an increasing number of vehicles on the road as well as a limited road infrastructure. These aspects culminate in daily challenges for the average commuter due to congestion and slow moving traffic. In the United States alone, it costs an average US driver $1200 every year in the form of fuel and time [1]. Some positive steps, including (a) introduction of the push notification system and (b) deploying more law enforcement troops, have been taken for better traffic management. However, these methods have limitations and require extensive planning [2]. Another method to deal with traffic problems is to track the congested area in a city using social media. Next, law enforcement resources can be re-routed to these areas on a real-time basis.

Given the ever-increasing number of smartphone devices, social media can be used as a source of information to track the traffic-related incidents.

Social media sites allow users to share their opinions and information. Platforms like Twitter, Facebook, and Instagram are very popular among users. These platforms enable users to share whatever they want in the form of text and images. Facebook users generate millions of posts in a minute. On these platforms, abundant data, including news, trends, events, opinions, product reviews, etc. are generated on a daily basis.

Worldwide, organizations are using social media for marketing purposes. This data can also be used to analyze the traffic-related events like congestion, construction work, slow-moving traffic etc. Thus the motivation behind this research is to use social media posts to extract information relevant to traffic, with effective and proactive

traffic administration as the primary focus. I propose an intuitive two-step process to utilize Twitter users' posts to obtain for retrieving traffic-related information on a real-time basis. It uses a text classifier to filter out the data that contains only traffic information. This is followed by a Part-Of-Speech (POS) tagger to find the geolocation information. A prototype of the proposed system is implemented using distributed microservices architecture.

1. INTRODUCTION

Traffic congestion is one of the biggest problems in our modern cities. Delays, road rage, environmental effects, and increased fuel consumption are some of its by products. To avoid these problems, governments and local law enforcement use inductive loops, cameras, and radar to monitor traffic [3]. These tools are effective but have drawbacks in term of installation and maintenance, along with high operational costs.

Large capital investments and a large workforce are required to build such infrastructure from the ground up, so leveraging the existing infrastructure for gathering traffic-related information would be more viable and cost-effective. Social media platforms can be used to serve that purpose. Every day millions of users on these platforms communicate with each other and share their opinions. With proper content filtering techniques, traffic-related incidents can be filtered out of all other events.

Twitter is a popular social media platform with millions of active users. It provides a channel between friends and co-workers to communicate using desktop or mobile applications. It offers a platform for market researchers, activists, and decision makers to access information on a real-time basis. Organizations are using it to learn about customer satisfaction levels [4]. Some researchers have even used it for the tracking seismic activity [5].

In the same way, mining this open source information can be utilized to track traffic incidents on a real-time basis. Analyzing the tweets can give us the location information without the use of hardware like LIDAR, cameras, etc.

1.1 Goal

The primary objective of this thesis is to develop an ecosystem to track traffic incidents in real time using a non-traditional source of information like social media data. It is believed that local law enforcement agencies can utilize this information for better traffic management and emergency response.

Currently, getting the real-time traffic information requires an array of sensors [6], but with the rise of social media, a massive amount of real-time traffic data is flowing through Twitter[1], Facebook, and other social media platforms that can be utilized as a substitute. These platforms are acting as a new medium where every user is a source of information.

There are applications like Google Maps[2] and Waze [3] that provide real-time traffic updates by leveraging crowd-source data, but social media channels are left out. For example, Waze provides an interface to report and geo-mark traffic-related information and Google collects data through Android phones, where every Android user acts as a data source. Google's proprietary algorithms predict the traffic congestion [7] by analyzing the number of the Android users and their speed. Although these platforms perform well, Google Maps and Waze are not utilizing other channels like social media. Therefore the primary objective of this thesis is to utilize social media platforms as a data source to monitor traffic incidents.

This thesis is divided into four major parts: data collection, text classification, location detection, and system architecture. For data collection, the Twitter platform is used as a data source. Twitter provides multiple ways to access the data using the rest API [8]. Text classification is used to filter out Tweets related to the traffic incidents and this is done by using a RNN model. Location detection means to determining the location from the text. For example, consider "Stefan is going to

[1]Twitter is an online news and social networking platform
[2]Online mapping service developed by Google
[3]Navigation application for smartphone

West Pacific Street". Here "West Pacific Street" is the location. In the last chapter, all the components are tied together to build a scalable system for real-time data processing.

2. DATA COLLECTION

At the beginning of this research, some public Twitter Id's were manually collected using Twitter's search interface. Keywords like 'traffic', 'rain', etc. (shown in Table A.2) were manually entered to the search interface to get the twitter accounts that posted tweets having these keywords. This activity was repeated multiple times for many cities. The main idea behind this exercise was to gather information about:

- Twitter accounts that frequently post traffic-related tweets.
- Number of Tweet being geo-tagged.
- Frequently occuring words in tweets.

The results of this task are:

- Identification of Twitter accounts in different cities that tweet traffic-related information.
- A vocabulary of frequently used words in the Tweets.
- Less than one percent of all the tweets contains geo-tagged data.

According to the output, a list of user accounts and keywords is compiled for the data collection task.

Twitter provides multiple ways to get the tweeted data [8] via their rest API[1]. Out of these methods, only the geo-location and the user-time-line methods are used to retrieve the data. For this thesis, only tweets that are available in the public domain have been used.

[1]Method to communicate between different components

2.1 Geo Location

In this API, the radius, longitude, and latitude of the target city are the input parameters [9]. The API returns all the Tweets within a given radius, where the center is the location specified by the geo-coordinates input to the API, and all Tweets having input keywords, within the radius.

2.2 User Time Line

In the user time-line method, the user Twitter ID is passed to the Twitter API which returns a collection of the most recent tweets and re-tweets posted by the user and the user's followers [10].

2.3 System Design

An application is created on the Linux server to download the publicly available Tweets. It requires authorization and token keys which, during an early phase of the application creation, are automatically assigned by Twitter [11]. There are multiple frameworks to access the Twitter API. One such example is Tweepy [12], which can extract the data in JSON and converts it into a Python dictionary.

To get the tweets, the system, shown in the Figure 2.1, implements both user-time-line and geolocation methods using the Tweepy framework. The system has the following two components: the data extraction server and the MySQL server.

2.4 Data Extraction Server

The data extraction server acts as middleware between Twitter and the MySQL server. It runs a Python program periodically to collect the data from Twitter. First, the Tweets' URLs are removed from the collected data. Next, Tweet Id, text and date are stored on the MySQL server.

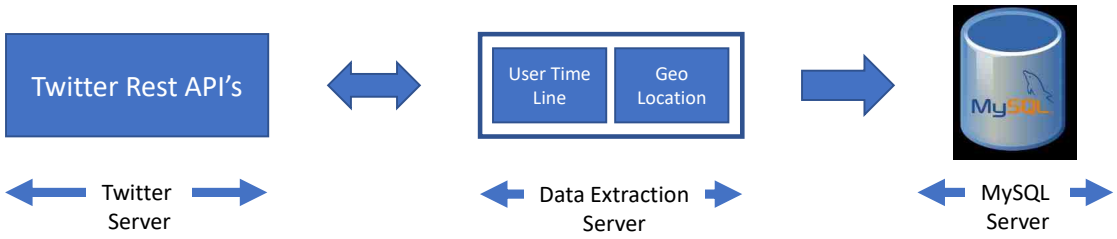

Fig. 2.1. Tweet Extraction Architecture

Approximately one hundred thousand tweets were collected in three months between October, 2016 and December, 2016.

2.5 MySQL Server

MySQL is an open source RDBMS system owned by Oracle [13]. It is used to store Tweet Id, date, and Tweet text, which are later used to train machine learning models.

3. TEXT CLASSIFICATION

3.1 Introduction

Classification is a supervised machine-learning methodology that involves assigning a label to set of input features. In machine-learning, a feature is an individual measurable property of an observed phenomena [14]. Generally, a feature is a numerical value such as the age of a person, a temperature etc. In the case of text, a sequence of letters and symbols can't be fed directly to the machine-learning algorithms [15]. Text feature-extraction algorithms convert a string into a vector.

Classification can be of a binary or a multi-class type.

- Binary classification is often used to determine whether an item is or is not in the class; but it can also be used if the data consists of two classes. Some common examples are spam detection, credit-card fraudulent-transaction detection, and gender identification.

- Examples of multi-class classification include country-of-origin detection and language detection.

3.2 Text Classification

A text classification algorithm, according to its content, assigns one of a given set of classes to an input document from a given set of classes. Document-type, song genre, book type, etc. For example, consider a text classification problem to find out whether an email is a spam or not. In this case, the classifier is a binary text classifier and the output is either "spam" or "not spam".

Text classification can be applied to solve a variety of problems such as:

- Understanding and identifying an opinion in a piece of text.

- Determining movie-review class from good, bad, or worse category.
- Spam identification.

A supervised text-classification task assigns one of a predefined set of classes. It starts by building a hypothesis function to do the classification. A hypothesis function is a mapping of input features to output classes and the classifier is trained using ground-truth data. For the learning process the whole dataset is divided into two parts; (a) test data which is used as ground-truth and (b) training data, which is used to validate the accuracy of the classifier.

3.2.1 Feature Extraction

The first step in feature extraction is to convert the text into a vector. Some popular text feature extraction methods are:

- Bag of words [16].
- Word2vec [17].

Bag of Words

The bag of words model is a vector space model of a text document. It is a frequency based vector representation where each word is represented by its number of appearances. It first builds the vocabulary using a document or set of documents and then converts a text document into a vector using word frequencies.

"Bag of words"" refers to the fact that this model ignores grammar and word order. To convert text into a vector involves the following steps (shown in Figure 3.1):

- Step One: Collect all documents.
- Step Two: Build the vocabulary - a collection of all the unique words in all documents.

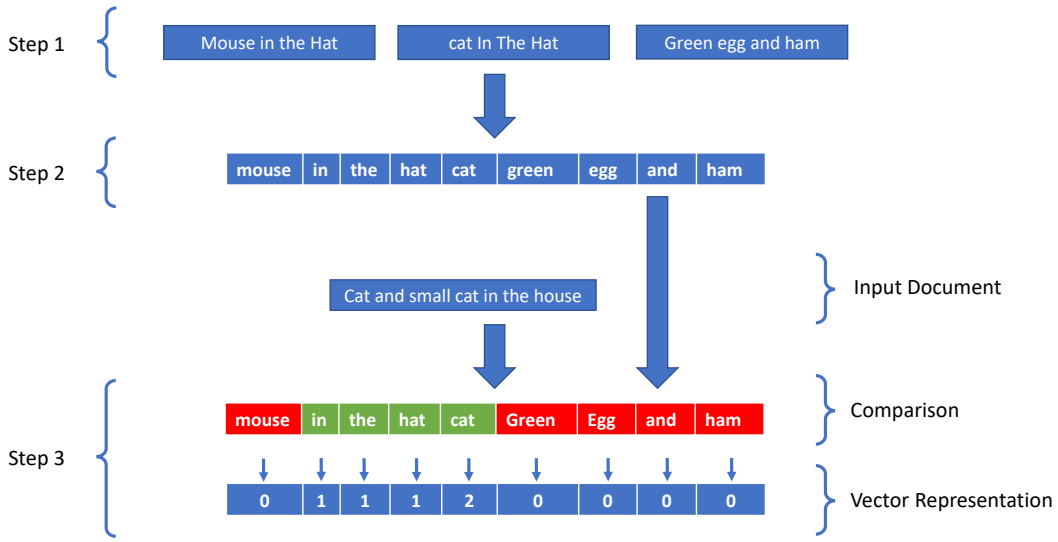

Fig. 3.1. Bag of Words Representation

- Step Three: Create the document vector by comparing the words with the vocabulary. Each word is assigned its frequency count in the vocabulary and zero otherwise.

Word2Vec

Word2vec [17] is a word-embedding model created by a team of researchers led by Tomas Mikolov at Google. Word embedding converts text into a vector. All the vectors that have the same context are placed nearby.

This model takes a large corpus as input and produces higher dimensional vectors. All the vectors have the same size. It has many advantages compared to earlier algorithms [17]. For example the word order does not influence the resultant word vectors generated by the model and the algorithm is computationally more efficient.

The Word2vec model generates word vector **M** based on its probability of occurrence based on the surrounding words, rather than its frequency. There are multiple ways by which Word2vec can generate the word vector. The skip-gram model described in next section is used because of its computational efficiency [17].

Skip Gram Model

In this model, the objective is to predict the target word based on the surrounding words within a given window size. The output vector relates the likelihood of the vocabulary words to the target words.

Consider a document having words W_1, W_2, ... W_{t-1}, W_t, W_{t+1}, ..., W_n. If the window size is one, the target word W_t will predict the surrounding words W_{t-1} and W_{t+1}, as shown in the Figure 3.2.

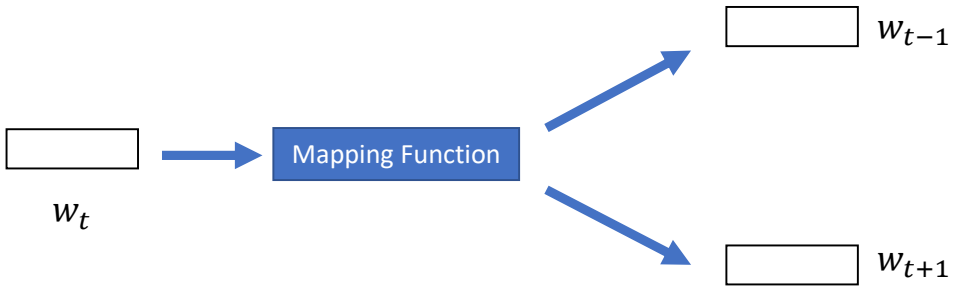

Fig. 3.2. Skip-Gram model

Word2vec Training

In the Word2vec model, the word vectors are generated randomly and stored in the embedding matrix, then the skip-gram model tries to learn the probability vector. The training goal is to find the target words given a set of surrounding words. At first,

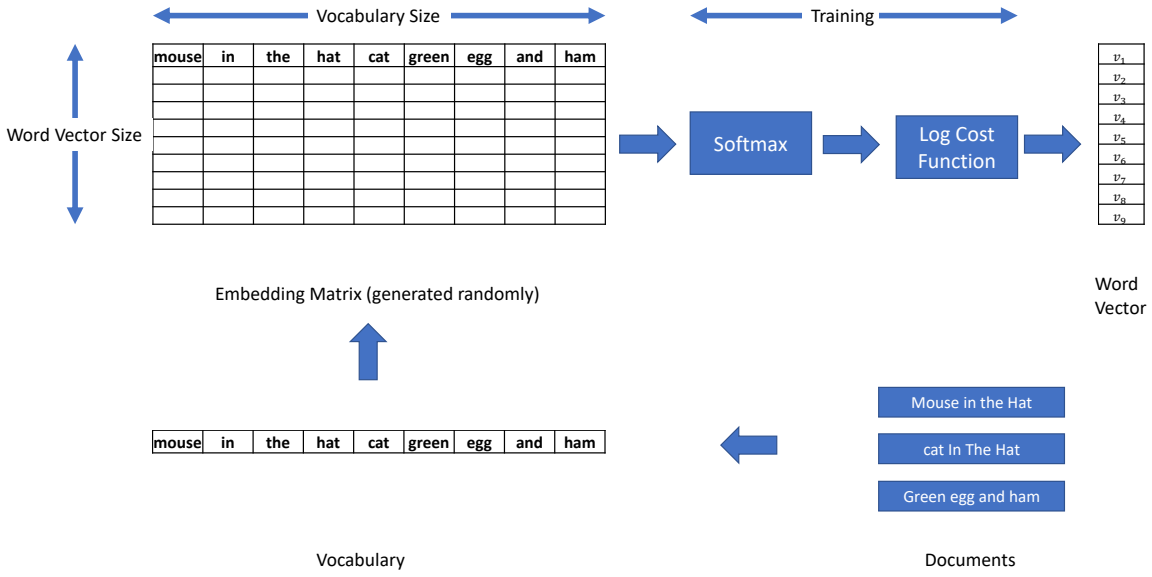

Fig. 3.3. Word2vec Training

the vocabulary is built from a set of input documents. Then the embedding matrix is generated randomly. In the training step, the embedding weights are learned using the softmax function in (3.2). The process is described in Figure 3.3.

Consider a document having words $W_1, W_2, \ldots W_n$. The skip-gram algorithm uses a gradient descent algorithm [18] to maximize the probability of the target word $p(W_{t+i}|W_t)$ given the surrounding words [17]:

$$\frac{1}{n}\sum_{t=1}^{n}\sum_{-k\leq i\leq k, k\neq 0}\log(p(W_{t+i}|W_t)) \tag{3.1}$$

where n is the vocabulary size and k is the window size. In the skip-gram model the conditional probability $p(W_{n+i}|W_n)$ is defined as:

$$p(W_{n+i}|W_n) = \frac{\exp(\tilde{\mathbf{M}}_{W_{n+i}} \cdot \mathbf{M}_{W_n})}{\sum_{i=1}^{n}\exp(\tilde{\mathbf{M}}_{W_{n+i}} \cdot \mathbf{M}_{W_n})} \tag{3.2}$$

where n is the vocabulary size, $\tilde{\mathbf{M}}_W$ is the output vector representation of the target word and $\mathbf{M}_W$ is the input representation of the target word.

3.2.2 Classification of Short Text

With the advancement of smartphones, the information content of online communication like Tweets and chat messages has increased over the time. This information can be mined from the aforementioned conservations. However, in terms of information extraction and data processing, microblogging websites like Twitter pose the hardest challenges [19]. This is due to its limited tweet length and its unstructured format. Tweets contain uniform resource locators (URLs), emoticons, abbreviations, and hashtags (#), which may not always contain valuable data. Moreover, they do not always follow normal sentence flow, grammar rules, and punctuation.

Related Work

Most of the related work focused on feature extraction. Naponget, *et. al.* [20] tried to find the tweets having traffic-related information with a fixed rule and a dictionary. In their method, a Tweet must have (at least) two keywords that must be related. In contrast Bharathet *et. al.* used a naive Bayes-classifier-based approach for filtering the Tweets. They designed a rule-based text feature-extraction method [21]. In another study by Takeshi *et. al.* [22], an earthquake event detection model was built using Twitter. They used a combination feature set including predefined keywords and word contexts. There feature set is fed as an input to the Support Vector Machine (SVM). Divij and Chanh performed feature extraction in the following steps [23]:

1. Remove punctuations and URLs.
2. Perform spelling correction.
3. Remove consecutive characters.

The resulting feature set is used as an input to different classifier algorithms. They obtained accuracy of 82.71% with Logistic Regression, 72.28% with SVM and 68.7% with boosted decision tree. One of the most significant challenges in almost all the

related work is to develop a classifier that understands the deeper meaning of the text. For example, consider the sentence "Network traffic heat's up my router". Although the given text doesn't contain any information related to road traffic, filtering out these kind of tweets with the above methods is difficult. In this work, we tried to overcome this problem by applying multiple classifiers and to identify the best one for this scenario.

3.3 Tweet Classification

Out of one hundred thousand Tweets that were collected, around eight thousand Tweets were selected randomly for use as a training/test set and were manually labelled into two classes:

- Traffic-related incidents, which contains all the traffic related tweets (those Twitter posts that contain information related to accidents, road blocks, and slow moving traffic).
- Non-traffic-incidents, which contains all tweets that are not related to traffic incidents.

The labelled dataset contains 3665 Tweets of traffic class datapoints and 4514 of non traffic class datapoints. For the data classification task, I tested (a) a logistic regression, (b) a Support Vector Machine (SVM), (c) a Feed Forward Neural Network (FDNN), and (d) a Recurrent Neural Network (RNN).

The approach that has been used with logistic regression, the SVM, and the FDNN has the following framework: The program first preprocesses the data and removes URLs, then extracts the features, and in the final stage applies classification algorithms as shown in Figure 3.4.

The three steps are discussed in more detail below:

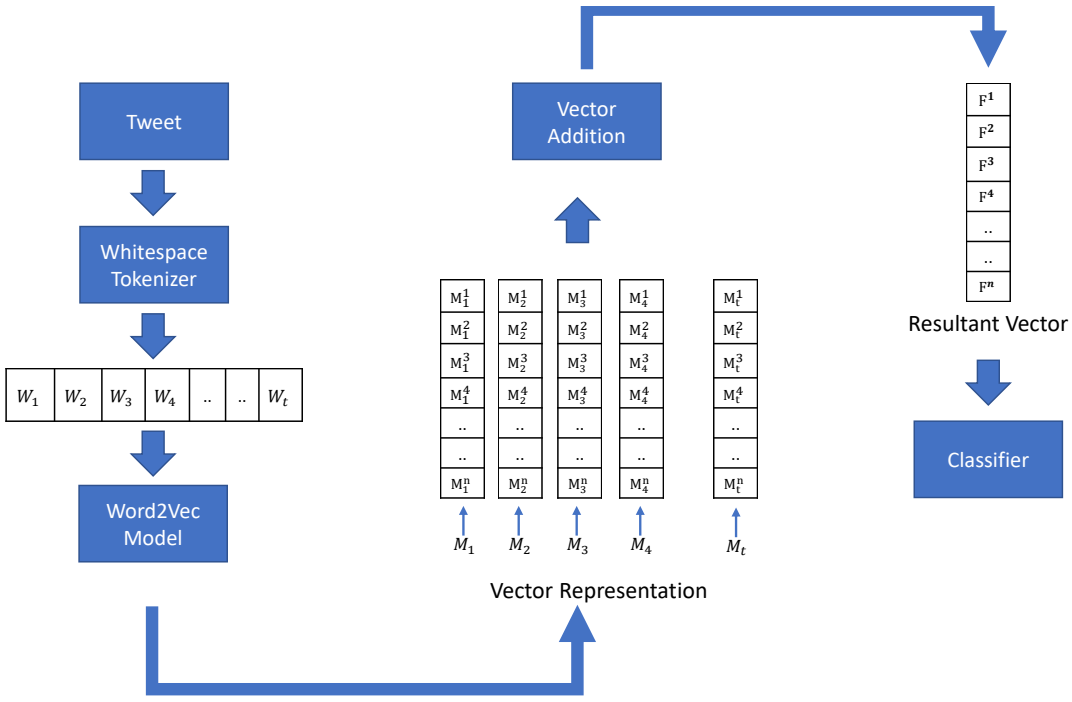

Fig. 3.4. Classification Pipeline

- Data Preprocessing: Twitter posts contains lots of formatted text, like hashtags, URLs and emoticons; but, unlike the URLs, characters like hashtags and emoticons can give a deeper understanding of the Tweets [24]. In the preprocessing step URLs are removed and each Tweet is segmented into linguistic units like emoticons, characters with hashtags, and words. This process is called tokenization. The segmentation criteria for the tokenization process is the presence of white spaces.

- Feature Extraction: Segmented words $\mathbf{W}_1, \ldots, \mathbf{W}_n$ are converted into word vectors $\mathbf{M}_1, \ldots, \mathbf{M}_n$ using the Wor2vec model. The resultant vector $\mathbf{F}$ is generated as shown in Figure 3.4 by vector addition of the individual words vector $\mathbf{M}_i$,

$$\mathbf{F}^i = \sum_{j=1}^{t} \mathbf{M}_j^i \tag{3.3}$$

and,

$$\mathbf{F} = (F^1, F^2, \ldots, F^n), \tag{3.4}$$

where t is then number of words in the tweet, and n is the size of the word vector.

- Classification: Logistic Regression, SVM and FFNN methods were applied to identify the best classifier.

3.3.1 Logistic Regression Classifier

Logistic regression [25] is a supervised machine-learning algorithm inherited from the statistical domain. It uses multiple independent input variables that determine discrete output classes. More specifically it finds the probability of the output prediction given the input, i.e. $P(\text{output}|\text{input})$. It generates the coefficients for the hypothesis function to predict the probability of the output class.

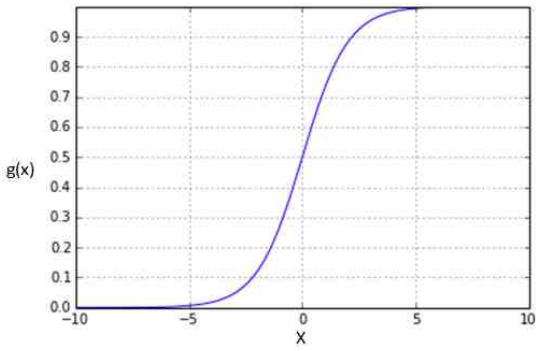

Fig. 3.5. Sigmoid Function

In logistic regression, the goal is to fit a best linear model (known as a hypothesis function) that expresses the relationship between input and output variables. The hypothesis function $h(\mathbf{w}, \mathbf{F})$, where $\mathbf{w} \equiv (\mathbf{w}^1, \mathbf{w}^2, \ldots, \mathbf{w}^n)$, is:

$$h(\mathbf{w}, \mathbf{F}) = g(\mathbf{w} \cdot \mathbf{F}) \tag{3.5}$$

and the $\mathbf{w}^i$'s are the weights associated with the input feature vector $\mathbf{F}$, of size n, and for $g(x)$, I used sigmoid activation function,

$$\Theta(x) = \frac{1}{1 + \exp(x)} \tag{3.6}$$

shown in Figure 3.5. In binary classification, the output belongs to the positive class when $h(\mathbf{w}, \mathbf{F}) \geq 0.5$; otherwise the output belongs to the negative class.

In supervised machine-learning algorithms, a cost function is defined to penalize inaccurate prediction. Because of both the lack of information (the relation between input and output is unknown) and the noise in the input features, there is always a nonzero probability of getting the same output for different input vectors. That's why our end goal here is to minimize the error in the prediction. The cost function $J(\mathbf{w}, \mathbf{F})$ is defined as:

$$J(\mathbf{w}, \mathbf{F}) = -y(\mathbf{F}) \log h(\mathbf{w}, \mathbf{F}) - (1 - y(\mathbf{F})) \log(1 - h(\mathbf{w}, \mathbf{F})) \tag{3.7}$$

Here $h(\mathbf{w}, \mathbf{F})$ is the hypothesis function defined in (3.5), and y is the actual output label associated with the input. For simplicity, $\mathbf{F}$ is removed from equation (3.7) and the resulting equation can be written as:

$$J(\mathbf{w}) = -y \log h(\mathbf{w}) - (1 - y) \log(1 - h(\mathbf{w})) \tag{3.8}$$

The weight $\mathbf{w}$ can be found by minimizing the cost function and optimization methods like gradient descent [18] are used for that purpose.

3.3.2 Support Vector Machine Classifier

The SVM is a non-probabilistic binary classifier. It first pre-processes the data and then maps it into a higher-dimensional space, whose dimension is typically much higher than that of the input. This feature mapping is done by kernels as shown in Figure 3.6. Linear classification models like Logistic Regression can classify the input data with good accuracy, but only when the data is linearly separable [26].

Thus classification algorithms need more complex feature sets to get better accuracy, and non-linear mapping can achieve this. For this purpose, any linear model can be turned into a nonlinear model using kernels.

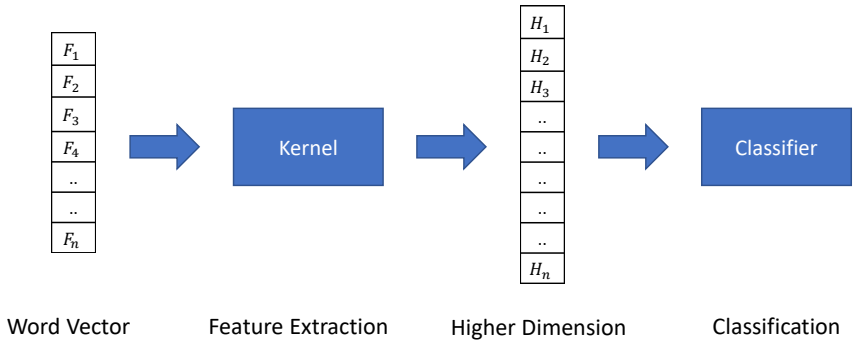

Fig. 3.6. Kernel Feature Extraction Process

The use of the kernel methods helps the hypothesis function to learn non-linear decision boundaries. Instead of learning a fixed set of parameters corresponding to the input feature space, the hypothesis function learns the weight vector $\mathbf{w}$ for the complex feature set $\mathbf{H}$.

Some commonly used kernels include polynomial kernel and radial kernel.

Polynomial Kernel

The polynomial kernel method represents the input feature vector similarity over the polynomial space. More specifically, the polynomial kernel of degree d is:

$$K(\mathbf{F}_1, \mathbf{F}_2) = (\mathbf{F}_1 \cdot \mathbf{F}_2 + c)^d \tag{3.9}$$

where $\mathbf{F}_1$ and $\mathbf{F}_2$ are the input feature vectors, d defines the degree of the polynomial and c is a real number. In our case $\mathbf{F}_1$ and $\mathbf{F}_2$ are the feature vectors from two different Tweets.

For simplicity, consider the input feature space of degree n, for which the square polynomial kernel is:

$$K(\mathbf{F}_1, \mathbf{F}_2) = \left(\sum_{t=1}^{n} \mathbf{F}_1^t \mathbf{F}_2^t + c \right)^2, \tag{3.10}$$

where $\mathbf{F}_1$ and $\mathbf{F}_2$ are arbitrary input feature vectors. Equation (3.10) can be expand using the binomial theorem to obtain:

$$\sum_{i=1}^{n} \left[\left(\mathbf{F}_1^i \mathbf{F}_2^i \right)^2 + 2c \mathbf{F}_1^i \mathbf{F}_2^i \right] + c^2 + 2 \sum_{i=2}^{n} \sum_{j=2}^{i-1} \mathbf{F}_1^i \mathbf{F}_1^j \mathbf{F}_2^i \mathbf{F}_2^j. \tag{3.11}$$

Thus the new features for the pair $\mathbf{F}_1$ and $\mathbf{F}_2$ are:

$$\begin{aligned} &\mathbf{F}_1^1, \ldots \mathbf{F}_1^n, \mathbf{F}_2^1, \ldots \mathbf{F}_2^n, \mathbf{F}_1^1 \mathbf{F}_2^1, \ldots, \mathbf{F}_1^n \mathbf{F}_2^n, \left(\mathbf{F}_1^1 \mathbf{F}_2^1 \right)^2, \ldots, \left(\mathbf{F}_1^n \mathbf{F}_2^n \right)^2, \\ &\mathbf{F}_1^3 \mathbf{F}_2^3 \mathbf{F}_1^2 \mathbf{F}_2^2, \mathbf{F}_1^4 \mathbf{F}_2^4 \mathbf{F}_1^2 \mathbf{F}_2^2, \mathbf{F}_1^4 \mathbf{F}_2^4 \mathbf{F}_1^3 \mathbf{F}_2^3, \ldots, \mathbf{F}_1^n \mathbf{F}_2^n \mathbf{F}_1^{n-1} \mathbf{F}_2^{n-1}, c^2 \end{aligned} \tag{3.12}$$

As seen in the (3.12), the dimension of the feature set is generally higher than that of the original feature vector $\mathbf{F}$.

Radial Kernel SVM

The Radial or Gaussian Kernel is defined as:

$$K(\mathbf{F}_1, \mathbf{F}_2) = \exp\left(\frac{-||\mathbf{F}_1 - \mathbf{F}_2||^2}{2\sigma^2} \right), \tag{3.13}$$

where σ is an independent parameter and

$$\|\mathbf{F}_1 - \mathbf{F}_2\| = \sqrt{\sum_{i=1}^{n} \left(\mathbf{F}_1^i - \mathbf{F}_2^i \right) \left(\mathbf{F}_1^i - \mathbf{F}_2^i \right)}. \tag{3.14}$$

SVM Training

Where $\mathbf{H}$ is the higher-dimensional feature vector obtained using the kernel method, a hyperplane that divides the dataset into two classes can be written as:

$$\mathbf{w} \cdot \mathbf{H} - b = 0, \tag{3.15}$$

where $\mathbf{w}$ is the normal vector to the hyperplane and the parameter b determines the offset from the origin. Now a loss $J_i(\mathbf{w})$ for a specific $\mathbf{H}_i$ can be defined as:

$$J_i(\mathbf{w}) = \max(0, 1 - y_i(\mathbf{w} \cdot \mathbf{H}_i - b), \tag{3.16}$$

where y_i is the actual label of i^{th} dataset and $\mathbf{w} \cdot \mathbf{H}_i - b$ is the SVM prediction. The final goal is to minimize the cost function:

$$J(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} \max(0, 1 - y_i(\mathbf{w} \cdot \mathbf{H}_i - b)) \tag{3.17}$$

using the gradient descent method.

3.3.3 Feed Forward Neural Network

The FFNN is also known as the multilayer perceptron. The inspiration for neural networks is taken from the human brain [27]. In the brain, different parts work together on the same problem to obtain the result. In the same way, an Artificial Neural Network (ANN) uses different layers working together on the same task.

The reason for the term "feed forward" is that the information flows through the hypothesis function and, with multiple intermediate calculations, the output y is determined. There is no feedback channel. While information is propagating forward. In FFNNs, the primary objective is to find a hypothesis function, $h(\mathbf{w}, \mathbf{F})$, that can map the input feature set to the output y. Data is first fed to the input layer. It then passes through the multiple hidden layers. The output layer predicts the probability of a certain class using the softmax function defined in (3.18).

Neural Network Architecture

An Artificial Neural Network (ANN) is a collection of an inter-connected units known as neurons, see Figure 3.7. Each neuron transmits signals to neurons in the next layer using a weight vector $\mathbf{w}$. An output of a neuron is a real number resulting from applying a non-linear function known as an activation function to the linear combination of the inputs.

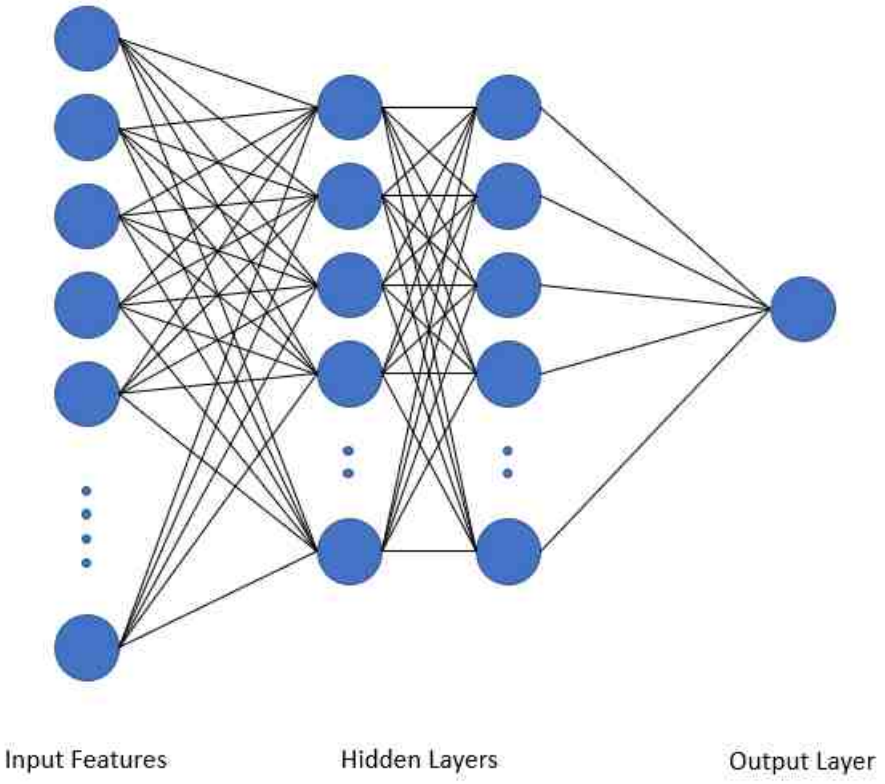

Fig. 3.7. Feed Forward Neural Network

Activation Function

The activation function controls the flow of information from a neuron. It works as a gate, which can be viewed as the conceptual representation of a neuron firing. Specifically, if the information is required, then the neuron will allow the information to pass through (using the activation function); otherwise it will not.

Generally, FFNNs have three primary sections, the input layer, the hidden layers and the output layer. These are described next:

- Input Layer: This layer acts as a bridge between the data and the neural network. It passes the data to the first hidden layer. No computation is performed here.

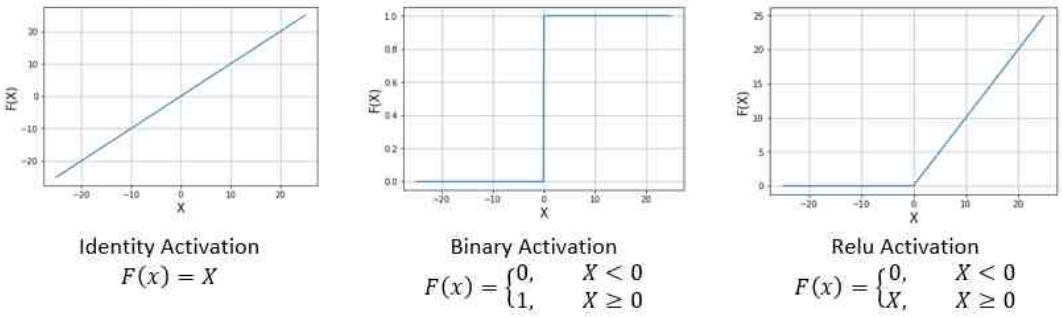

Fig. 3.8. Different activation Functions

- Hidden layers: A hidden layer consists of many neurons. It first calculates the input weight summation and then passes the weighted sum x through an activation function $g(x)$.

- Output layer: The number of neurons in the output layer depends on the number of classes in the input data. For example for binary classification the number of output neurons is two. At the last layer, each neuron's output gives the probability that the input features belongs to the specific class, calculated using the softmax function. Specifically,

$$p(y = i|\mathbf{F}) = \frac{\exp(\mathbf{F} \cdot \mathbf{w}_i)}{\sum_{j=1}^{N_c} \exp(\mathbf{F} \cdot \mathbf{w}_j)} \tag{3.18}$$

  where N_c is the number of classes and $\mathbf{w}_i$ is the weight vector assigned to the i^{th} class.

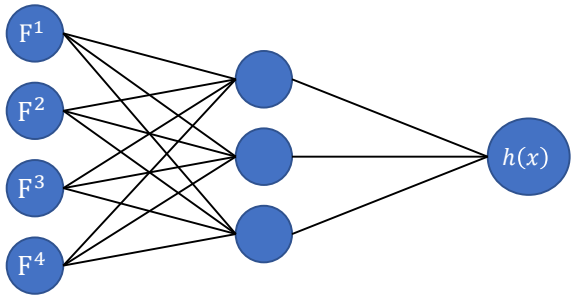

Fig. 3.9. Neural Network Training

Neural Network Training

Figure 3.9 shows a simple three-layer neural network having one input layer, one hidden layer and an output layer, interconnected by weighted links. For an arbitrary number of hidden layers, the i^{th} neuron of the j^{th} hidden layer computes the weighted sum of its inputs:

$$F^i\left[j+1\right] = g\left(\sum_{1\leq l\leq U[j]} \mathbf{F}[j]\cdot \mathbf{w}_l[j]\right), \text{where } i \in \{1,2,\ldots,N_{j+1}\}, \tag{3.19}$$

where $\mathbf{w}_i[j]$ are the hidden layer weights for i^{th} the neuron at the j^{th} hidden layer, $U[j]$ is the number of neurons at the j^{th} layer, where j =1,...,N_l, where N_l denotes the number of hidden layers, and $g(x)$ is the activation function. The k^{th} neuron in the output layer gives the probability of the original feature vector corresponding to the class k. Thus we refer to the output of this k^{th} neuron as h_k for hypothesis k. Let l_h denote the output layer, the output $h_k(\mathbf{w},\mathbf{F})$ for the k^{th} class can be written as:

$$h_k\left(\mathbf{w},\mathbf{F}\right) = g\left(\mathbf{F}\left[l_h\right]\cdot \mathbf{w}_k\left[l_h\right]\right), \tag{3.20}$$

In summary, the neural network first propagates the input through multiple layers and at the last layer the probability for different classes is calculated.

The neural network learns the weight vectors by the process of back-propagation [28]. Back-propagation adjust the weights first for the output layer and then works backward through the hidden layers. Back-propagation is a method that learns from its mistakes. In most classification tasks, the softmax function is used as an activation function at the last layer of the neural network.

The cost function $J(\mathbf{w})$, used in the neural network case, is the same as that used logistic regression, but instead of optimizing the weights corresponding to a single feature set it optimizes multiple weight vectors that is distributed across the neural network. The corresponding cost cost function is:

$$J(\mathbf{w}) = \sum_{i=1}^{N_c} -y \log(h_i(\mathbf{w}, \mathbf{F})) - (1-y) \log(1 - h_i(\mathbf{w}, \mathbf{F})) \tag{3.21}$$

where N_c is the number of classes.

The training process consists of two steps: (a) forward propagation followed by (b) backward propagation.

Forward propagation

In this step all the weights are randomly assigned, and then the neuron's output is calculated according to (3.20).

Backward Propagation

The total cost is calculated using (3.21), the gradient of the cost is back-propagated through the network, and a gradient descent method is used to update the weights, to get the minimum error.

3.3.4 Results

For the classification task, I tested different combinations of parameters including learning rate for the gradient descent, number of layers and number of neurons in the neural network, and the choice of kernels for the SVM. I also tested the algorithms including logistic regression, SVM with radial and polynomial kernels of degrees 1, 3, and 5, and neural networks with multiple hidden layers in order to find the best classifier. The results are shown in Figures 3.10, and 3.11.

Out of all the classifiers, the SVM gave the best accuracy, 96% with a Word2vec window of size three and word vector length of 300. The SVM classifier gave the best accuracy out of all the other classifiers.

Table 3.1. Examples of Incorrect Predictions: False Positive

	Tweet	Category
1	rt jbarro construction probably the more promising vehicle for this than manufacturing	False Positive
2	m northbound between j a and j vehicle fire	False Positive
3	brooklyn bushwick ave amp grand st pedestrian struck and pinned under a train fdny amp nypdspecialops on scene operating	False Positive
4	m t a n y c subways train public transit services not operating between th st and s ferry traffic	False Positive
5	update cleared	False Positive
6	cleared special event on barclayscenter	False Positive
7	how we doubled our traffic within months data included traffic sales startup	False Positive
8	propertyuklinks right on	False Positive
9	burbank crash at state and austin	False Positive
10	rt nypddetectives update	False Positive

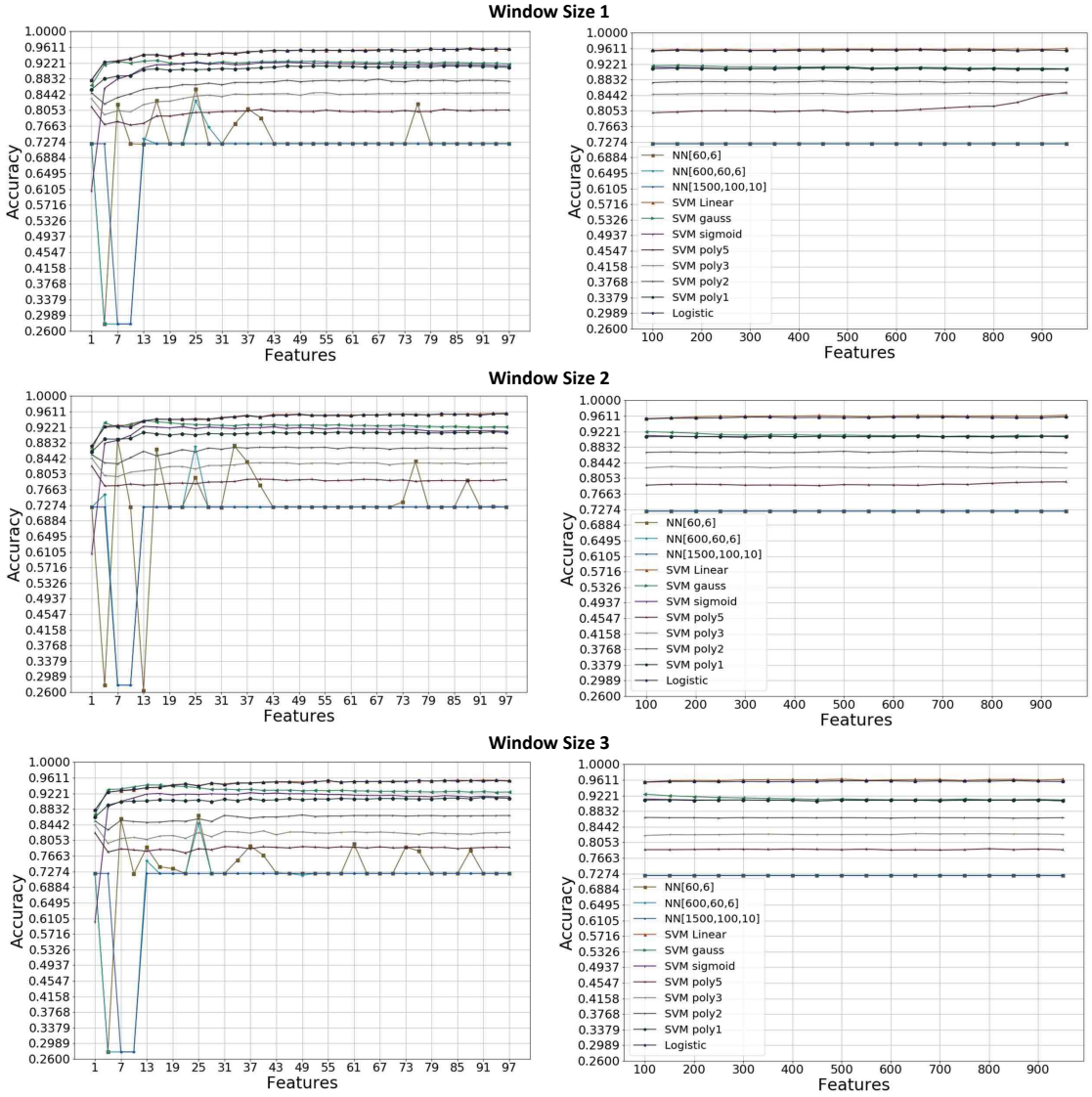

Fig. 3.10. Classifier Results With Context Window Size 1 to 3

Result Analysis

For the SVM, the incorrect predictions were divided into false positives and false negatives. A false positive means the tweet is predicted to be traffic related but actually it is not. A false negative means that the tweet is predicted to the be non-traffic related but in actuality it's traffic related.

Table 3.2. Examples of Incorrect Predictions: False Negative

	Tweet	Category
1	can t sleep so what do i do go to work early at least i can get past the weigh station and avoid some beltway traffic i suppose	False Negative
2	you d think the supermarket would be emptier in the morning not at least traffic was lighter by the time i left for work	False Negative
3	sleepy and dreading the traffic which starts to build up in front of her office	False Negative
4	worst traffic jam ever over hours and i ve only moved maybe a mile	False Negative
5	tayusa she should better learn that spent too much time here i was late today because of traffic jam	False Negative

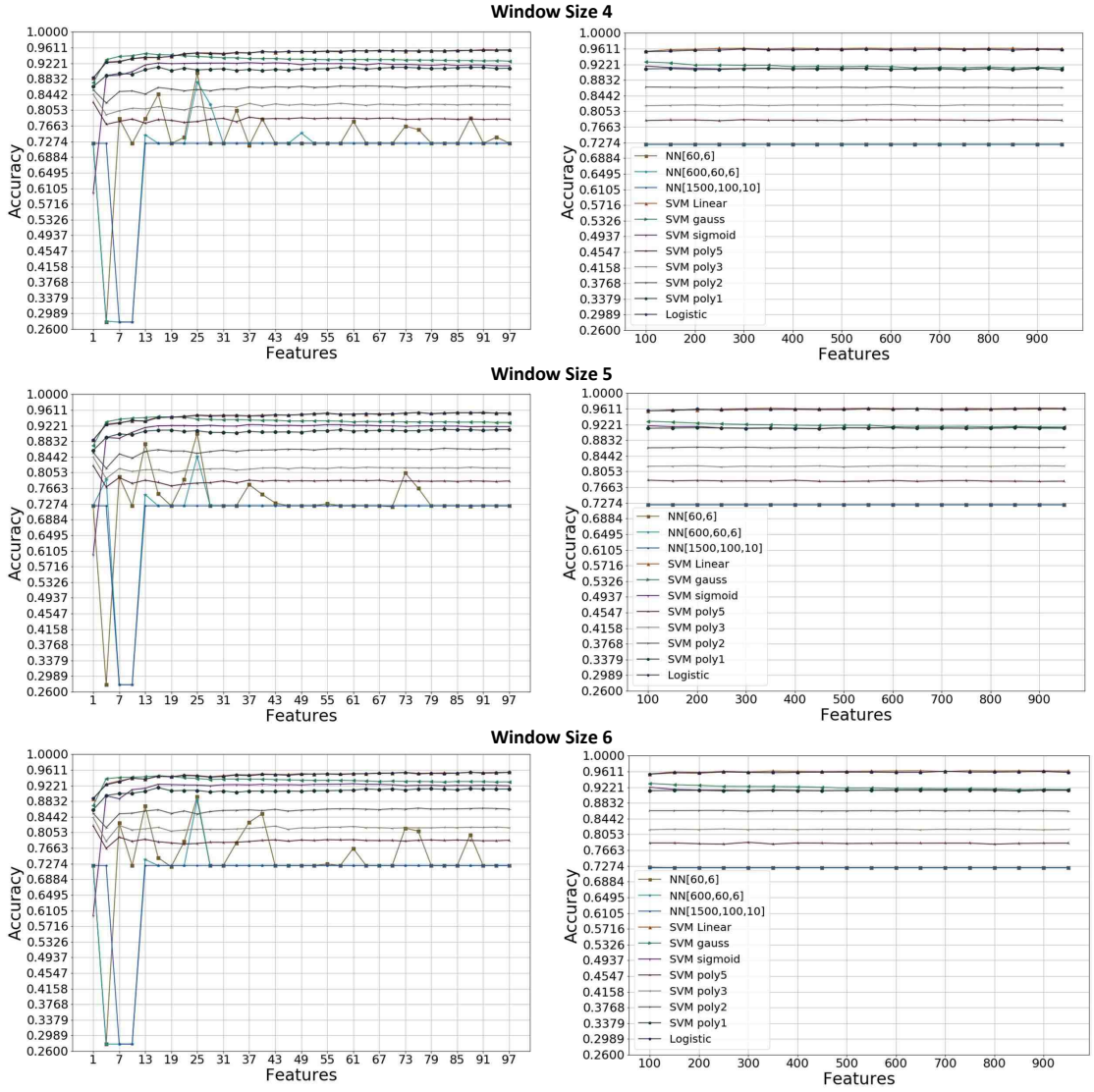

Fig. 3.11. Classifier Results With Context Window Size 4 to 6

The Tweets one, four, seven, and nine in Table 3.1 do not have any information related to road traffic but were identified as road traffic incidents. The Tweets one, three, and five in Table 3.2 contains traffic-related information but were predicted to be non-traffic related incidents.

One reason for these incorrect prediction is the method of the feature vector generation. The remaining incorrect predictions is due to the mislabeling of the Tweets. The issue in feature vector generation is, as shown in (3.3) and (3.4), that the final vector **F** is the summation of the individual word vectors shown in (3.4). Thus by using the classification pipeline shown in the Figure 3.4, word positions were not taken into consideration. More specifically, consider two sentence of seven words $W_1, W_2, \ldots W_7$ in different word orders. The resultant word vector **F** would be the same for both sentences.

The other thing that I found about the classier model is that it was sensitive to some keywords like a crash, traffic light, vehicle and traffic. If one or more of these keywords were in the tweet, then the resultant vector was assigned to the wrong side of the classification plane, causing misclassification.

To overcome this problem, an RNN is applied with a LSTM unit. The RNN model with LSTM unit can learn the word location dependencies in a sentence. In RNN, every word vector is going to the individual RNN unit and every unit transfers its learning to the next RNN unit, which is, in turn, able to recognize the much broader meaning of the tweets.

3.3.5 Recurrent Neural Network

The reason for calling the RNN recurrent, is that it performs the same task on every element in a given sequence. In an RNN, different nodes are connected by a directed graph. Unlike a feed-forward neural network, an RNN does not assume that the input feature vector components are independent. Thus the prediction of the current word in the sentence depends on the previous words. Because of the RNN's internal memory, it can remember the past input.

RNN Architecture

The RNN can have multiple architectures two of which are shown in Figure 3.12, i.e. Multiple-Input Multiple-Outputs (MIMO) and Multiple-Input Single-Output (MISO). The MISO architecture (shown in Figure 3.12 A) can be used for the classification. The intermediate computation for every sequence element is done by the

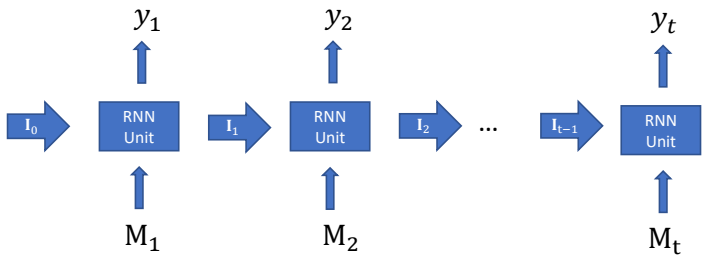

Figure A – Multiple Input Multiple Output RNN

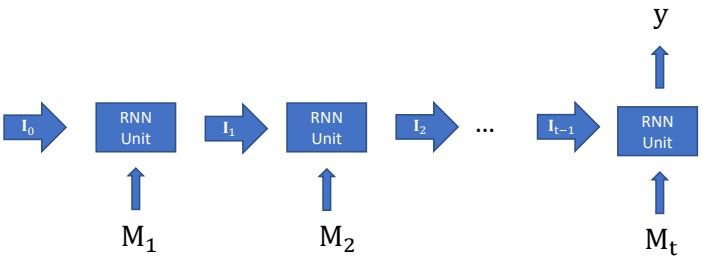

Figure B – Multiple Input Multiple Output RNN

Fig. 3.12. RNN Architectures

RNN units which also transfer their learning to the next units. Generally the internal state vector $\mathbf{I}$ is given by:

$$\mathbf{I}^i[t] = g\left(\mathbf{w}_\mathbf{I}^i \mathbf{I}^i[t-1] + \mathbf{w}_\mathbf{M}^i \mathbf{M}^i[t]\right), \tag{3.22}$$

where $g(x)$ is the activation function, $\mathbf{w_I}$ is the internal state weight vector, $\mathbf{w_M}$ is the input weight vector, and $\mathbf{M}$ the input word vector. The output y at position t is:

$$y[t] = g\left(\mathbf{w} \cdot \mathbf{I}[t-1]\right), \tag{3.23}$$

where $\mathbf{w}$ is the output weight vector. The most commonly used recurrent unit for the ordered sequence data like text is the Long Short Term Memory (LSTM) [29].

Long Short Term Memory (LSTM)

The LSTM architecture is similar to that of the RNN. The only difference is in the way it calculates the internal state. The LSTM model can capture the long-term dependency of a word in a sentence effectively [30]. Every LSTM unit (or cell) has three gates: the input gate, the forget gate, and output gate. The update gate output Γ_u, the forget gate output Γ_f, and the output gate output Γ_o are given by:

$$\Gamma_u^i\left[t\right] = \Theta\left(\mathbf{w}_u^i[t]\,\mathbf{I}^i\left[t-1\right] + \mathbf{w}_{\mathbf{M}[t]}^i\mathbf{M}^i\left[t\right]\right), \tag{3.24}$$

$$\Gamma_f^i\left[t\right] = \Theta\left(\mathbf{w}_f^i[t]\,\mathbf{I}^i\left[t-1\right] + \mathbf{w}_{\mathbf{M}[t]}^i\mathbf{M}^i\left[t\right]\right), \tag{3.25}$$

$$\Gamma_o^i\left[t\right] = \Theta\left(\mathbf{w}_o^i[t]\,\mathbf{I}^i\left[t-1\right] + \mathbf{w}_{\mathbf{M}[t]}^i\mathbf{M}^i\left[t\right]\right). \tag{3.26}$$

where Θ is the sigmoid activation function, $\mathbf{w_u}$ is the update gate's weight vector, $\mathbf{w}_f$ is the forget gate's weight vector, $\mathbf{w}_o$ is the output gate's weight vector, and $\mathbf{w}_M$ is the weight vector for the i^{th} input word vector.

These gates collectively determine the information flow to every unit. Where the internal state $\mathbf{I}$ is,

$$\mathbf{I}^i\left[t\right] = \Gamma_o^i[t]\left(\Gamma_u^i[t]\mathbf{C}^i[t] + \Gamma_f^i[t]\mathbf{I}^i[t-1]\right), \tag{3.27}$$

$\mathbf{C}$ is,

$$\mathbf{C}^i\left[t\right] = \tanh\left(\mathbf{w}_{\mathbf{C}}^i\mathbf{I}^i\left[t-1\right] + \mathbf{w}_{\mathbf{M}[t]}^i\mathbf{M}^i\left[t\right]\right), \tag{3.28}$$

and tanh is the hyperbolic tangent activation function:

$$\tanh(x) = \frac{1-\exp(-2x)}{1+\exp(2x)}. \tag{3.29}$$

Generally, a word in a sentence is related to both previous and subsequent words; however, with the RNN architecture, capturing both backward, and forward dependencies is difficult. This is where bidirectional RNNs (shown in the Figure 3.13) [31] are helpful. Bidirectional RNN architecture increases the amount of information available to the network for predictions.

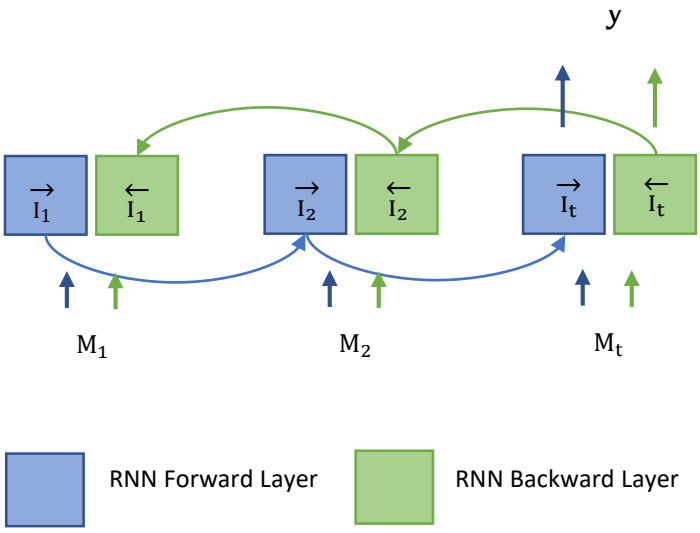

Fig. 3.13. RNN Bi-Directional Architecture

The basic idea in bidirectional RNNs is to create an additional backward layer for the internal state's propagation. More specifically, a forward-pass layer propagates the states as shown in (3.27) and (3.28) and the backward-pass layer propagates the states in the opposite direction.

All the calculations for the gates and the internal state are unchanged, and the backward-pass uses a separate set of states called the backward states and a separate sets of weights called the backward weights. The advantage of this architecture is that the output layer can make use of the information from both previous and next words. The output y at position t is calculated as:

$$y[t] = g\left(\overrightarrow{\mathbf{w}} \cdot \overrightarrow{\mathbf{I}}[t-1] + \overleftarrow{\mathbf{w}} \cdot \overleftarrow{\mathbf{I}}[t-1]\right) \tag{3.30}$$

$\overrightarrow{\mathbf{w}}$ is the forward weight vector for the forward layer, $\overrightarrow{\mathbf{I}}[t]$ is the forward internal state vector at layer t for the forward layer, $\overleftarrow{\mathbf{w}}$ is the backward weight vector for the backward layer, and $\overleftarrow{\mathbf{I}}[t]$ is the backward internal state vector at layer t for the

backward layer. The cost function is using both forward and backward layer internal state vectors and to find the forward and backward weight vectors. The optimization methods consist of the forward and backward propagation, the same as for (3.21).

Dropouts In LSTM

The dropout technique is used to address the problem of over-fitting. In the basic dropout technique, some neurons are removed randomly from the hidden layer, during the training phase. An example is shown in Figure 3.14. In the dropout-network technique, instead of removing neurons, only weights (showed with dotted line) are removed randomly from the neural network. Srivastava et al. [32] showed that the basic dropout technique [33] is ineffective for RNNs, as they lose the ability to retain the long-term dependencies. I applied a weight-dropped LSTM [34] inspired by the DropOut network [35].

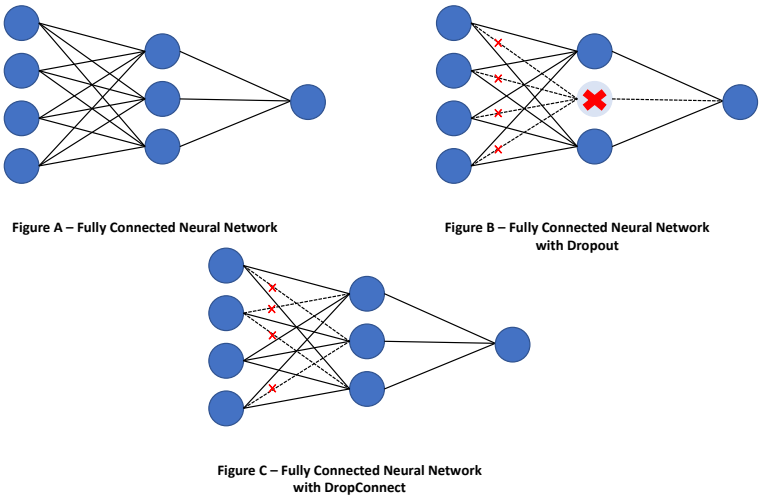

Fig. 3.14. Neural Network Dropout

Results

The effect of applying dropout techniques can be seen in the Table 3.3. For the cascaded RNN (shown in Figure 3.15-A), the cascaded RNN with neuron dropouts in FCNN (shown in Figure 3.15-B), the cascaded RNN with adder layer (shown in Figure 3.15-C), and cascaded RNN with neuron dropouts in FCNN and weight dropout in RNN layer (shown in Figure 3.15-D), the accuracy for the training data is nearly 100% while for test data it is around 91%. When 50% weight dropout in RNN layer and 60% dropout in the FCNN layer (shown in Figure 3.15-D) were applied, the accuracy for training and test data was around 92.5%.

The highest accuracy achieved on the test dataset was 92.6%, better than the previous SVM classifier because the RNN was able to capture the word dependencies. Most of the RNN's "incorrect" predictions in Table 3.5 (Tweet 1, 3, 4, 5, 6) were actually correct. They identified mislabeling of the dataset. Apart from that, the RNN's architecture was able to recognize the deeper meaning of the Tweets.

Consider the Tweets 1 and 4 in Table 3.6. These Tweets do not have information related to road traffic. The first one is talking about subway traffic and the second one is referring to a hit and run case. Though some of the other Tweets were labeled incorrectly, the classifier was able to correctly predict that they contain traffic related information. Thus, the presence of keywords such as 'accident', 'congestion' is not affecting the prediction.

I tested sensitivity to the weight regularization parameter. I found that the model was sensitive to the weight regularization parameter for both the Adam algorithm [36] and the gradient descent method. For a single layer RNN, there was an abrupt drop in the training and test accuracy for regularization parameter 0.1, 0.01 as shown in Table 3.4. That was the another motivation for using the weight dropout layer. There are two obvious ways to improve the accuracy. One way would be to correct

Table 3.3. Accuracy Table

	RNN's Architecture	Accuracy on Test Data	Accuracy on Training Data
1	Cascaded RNN	91.27	99.21
2	Cascaded RNN with Neuron Dropouts In FCNN Layer	91.53	98.64
3	Cascaded RNN with Adder Layer	91.31	99.36
4	Cascaded RNN with Neuron Dropouts in FCNN Layer and Weight Dropouts in RNN Layer	92.67	92.51

Table 3.4. RNN Regularization

	Adam Optimization		Gradient Descent	
Regularization	Test Accuracy	Training Accuracy	Test Accuracy	Training Accuracy
0	91.26	99.83	92.6	92.78
0.1	55.74	54.87	53.17	55.39
0.01	91.83	91.45	53.85	55.83
0.001	92.08	91.46	92.03	92.46
0.0001	88.78	95.84	92.52	92.57
0.00001	92.48	99.28	92.58	92.51

the labeled dataset. The other would be to increase the size of the dataset. As shown in the Table 3.3, applying a dropout technique we can improves the accuracy only to 92%. The classifier needs more data to get a better understanding of the tweets.

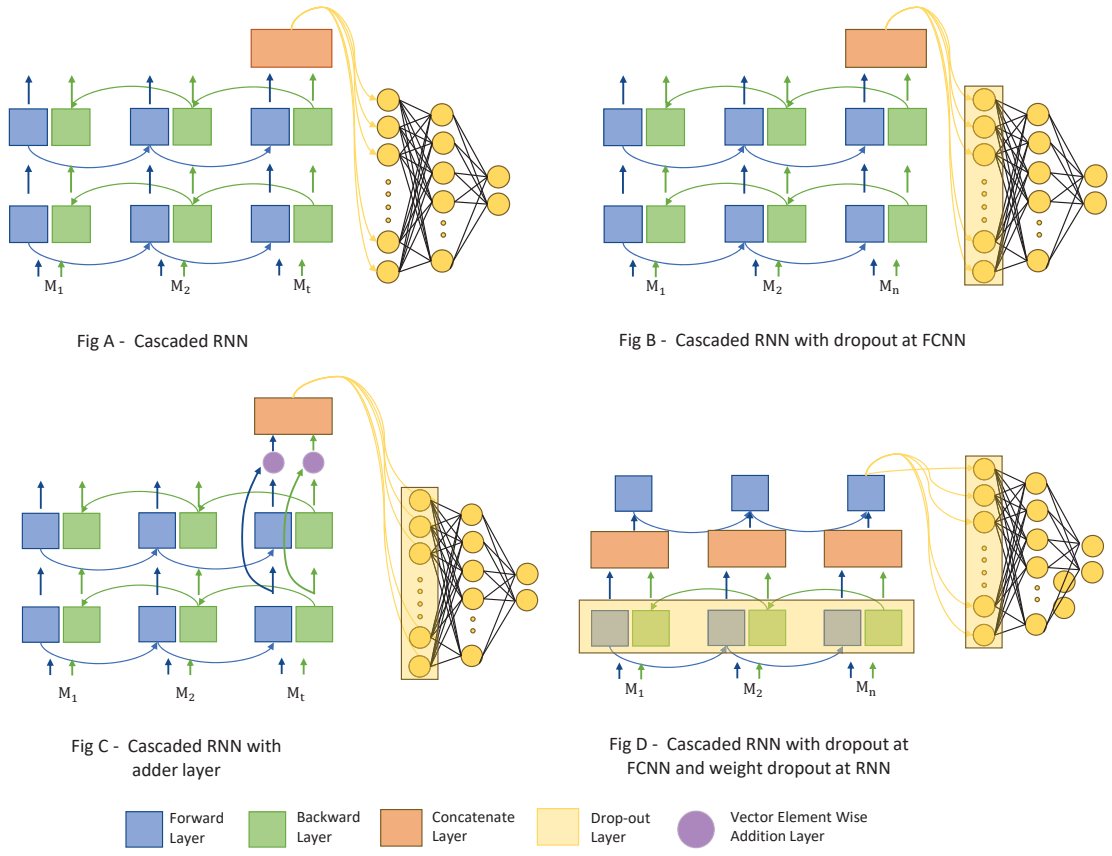

Fig. 3.15. Different RNN Architectures

Table 3.5. Examples of Incorrect Predictions: False Positives

	Tweet	Category
1	Crash at 70th St. and Damen Ave. #Chicago	False Positive
2	Update: Special Event on #5ThAvenue from 44th Street to 67th Street	False Positive
3	Disabled vehicle on the Palisades Interstate Pkwy SB north of x91/I-87 blocks right lane #nbc4ny'	False Positive
4	Cleared: Incident on #I278 EB at Queens-Kings County Line; Koskuisko Bridge'	False Positive
5	UPDATE: I-294 Southbound - STALL - South of HALSTED AV - MP 1.5 - ALL LANES OPEN'	False Positive
6	.. @AM730Traffic Don't see a stall there on our cameras right now. Must've been cleared fairly quickly! îm	False Positive
7	Closed	False Positive
8	. @wonderboy74 good news: our improvements at 24/176 should be finished in the next week - improving safety and flow at this signal. âr (2/2)'	False Positive
9	UPDATE: The left lane is now getting through. îm '	False Positive
10	Update: Construction on #M101Bus EB at 86th Street: Lexington Avenue	False Positive

Table 3.6. Examples of Incorrect Predictions: False Negatives

1	RT @NJTRANSIT_NEC:NEC, NJCL, & MidTown Direct trains are subject to 15 minute delays inbound to NY due to station congestion.	False Negative
2	Update: Incident on #Q83Bus Both directions at Liberty Avenue:168th Street	False Negative
3	A full closure of Foster Avenue over the North Branch Chicago River beginning Wed, Sep 7. for a bridge rehabilitation project.	False Negative
4	Driver surrenders in hit-and-run crash that injured 6-year-old boy via @ABC7NY. #saferidehom	False Negative
5	Road is closed	False Negative

4. LOCATION DETECTION

Location information can reveal the underlying economic and political trends for a given place. Twitter supports a geo-tagging features for tweets, which provideds finely tuned geolocation information. However, only 0.1 % of the Tweets studied contained geolocation information.

Geo-parsing of tweets, purely on their textual contents, is challenging, due to the unstructured format. People rarely follow grammar rules on social media, and that makes natural language processing techniques like Named Entity Recognition (NER) difficult.

NER is a subtask of the information retrieval task where the end goal is to categorize the text into predefined classes such as person names, locations, expressions, and organizations. A named entity may be a single word or a string of words, e.g.,"west Walnut street" is a location.

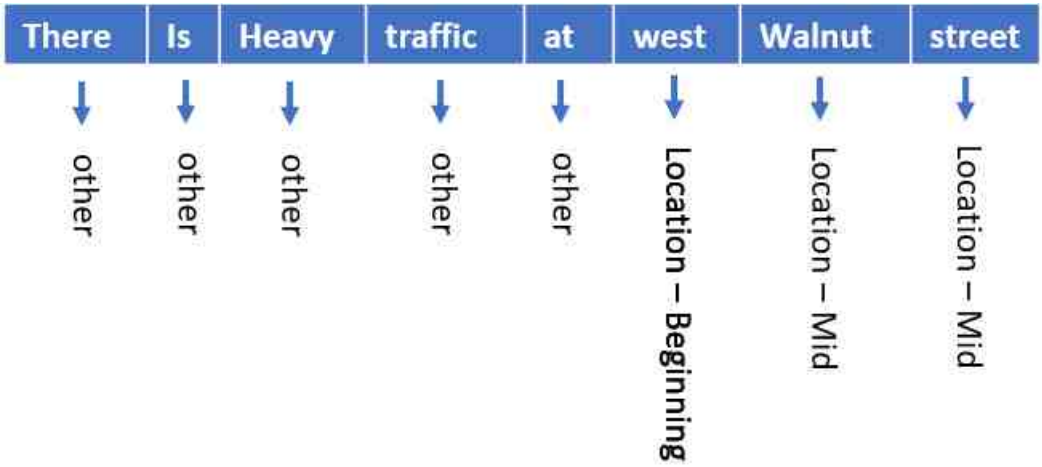

Fig. 4.1. Named Entity Recognition

NER can be used to solve a variety of real-world problems, for example producing robust search algorithms to find the documents from the text queries [37], identifying new drug hypotheses [38], and automating customer feedback analysis [39].

In the same way, determining the location from the text could be a NER problem where the entities are the locations.

4.1 Previous Work

Researchers including Cyril and Frederic [40] and Bornet and Kaplan [41] built Named Entity systems using sets of rules. However the NER task, just like the text classification task, can be solved using supervised machine-learning techniques. Manning [42] used a supervised machine-learning technique know as Conditional Random Fields (CRF). The documents used in these studies were structured, i.e. had proper grammar and punctuation.

In social media posts, people follow hardly any grammar and punctuation norms, which makes the NER detection harder. Elsafoury tried to overcome this problem by focusing on tweets that contained geo-coordinate information [43]. In another approach Cheng and Lee [44] tried to determine the user home location. Ritter, Clark, Etzioni et al. [45] tried to build a NER system specifically for Tweets. For training they augmented data from the Penn TreeBank project [46]. Gelernter and Balaji [47] used machine-learning techniques with a fixed sets of rules for basic sentence correction.

4.2 Conditional Random Field

Conditional random fields (CRFs) are a probabilistic approach, used for labelling and segmenting the sequential data. The underlying assumption in CRFs is that the next state depends on the current state and past states.

Thus the primary goal is to build a conditional distribution model

$$p\left(\mathbf{S}|\mathbf{M};\mathbf{w}\right) = p\left(\mathbf{S}_1, \mathbf{S}_2, \ldots \mathbf{S}_i|\mathbf{M}_1, \mathbf{M}_2, \ldots, \mathbf{M}_j; \mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_k\right), \tag{4.1}$$

that can map input symbols to the possible sequence states.

For a sentence, each $\mathbf{M}_r$, $r = 1, \ldots, j$ is the r^{th} input word vector and each $\mathbf{S}_a$, $a = 1, \ldots, i$, is the a^{th} state. We assume that S is a finite set. In NER, S is the set of all the named entities like persons, places, or names. Specifically,

$$p\left(\mathbf{S}|\mathbf{M};\mathbf{w}\right) = \frac{\exp\Big(\mathbf{w}\cdot\mathbf{f}(\mathbf{M},\mathbf{S})\Big)}{\sum_{a=1}^{i}\exp\Big(\mathbf{w}\cdot\mathbf{f}(\mathbf{M},\mathbf{S}_a)\Big)}, \tag{4.2}$$

where $\mathbf{f}(\mathbf{M},\mathbf{S})$ is a feature function and $\mathbf{w}$ is a weight assigned to the feature function. The feature function $\mathbf{f}(\mathbf{M},\mathbf{S})$ is defined as:

$$\mathbf{f}(\mathbf{M},\mathbf{S}) = \sum_{a=1}^{i}\mathbf{E}(\mathbf{M}, a, \mathbf{S}_{a-1}, \mathbf{S}_a). \tag{4.3}$$

Where $\mathbf{E}(\mathbf{M}, a, \mathbf{S}_{a-1}, \mathbf{S}_a)$ is the maximum entropy from a Markov model (MEMM) [48]. In Markov models the output of the current state is dependent on the previous states.

The weight vector $\mathbf{w}$ is optimized by using gradient descent with the cost,

$$\mathbf{J}(\mathbf{w}) = \arg\max_{1\le a\le i}\frac{\exp\left(\mathbf{w}\cdot\mathbf{f}\left(\mathbf{M},\mathbf{S}_a\right)\right)}{\sum_{r=1}^{i}\exp\left(\mathbf{w}\cdot\mathbf{f}\left(\mathbf{M},\mathbf{S}_r\right)\right)}. \tag{4.4}$$

4.2.1 Architecture

Bidirectional RNNs can relate word dependencies for the previous and next words in sentences and CRFs are useful for finding the current state probability with respect to the previous state. The advantage of using LSTM with a CRF is that it can utilize the previous and next words and then combined with the Twitter tag information to get a more in-depth understanding of the Twitter tokens. Hence Huang et al.

combined a bidirectional LSTM network with the CRF layer [49]. First a Tweet is vectorized (as shown in Figure 3.4), then the output word vectors are passed through a bidirectional LSTM network and the LSTM output is fed to the CRF layer as shown in Figure 4.2.

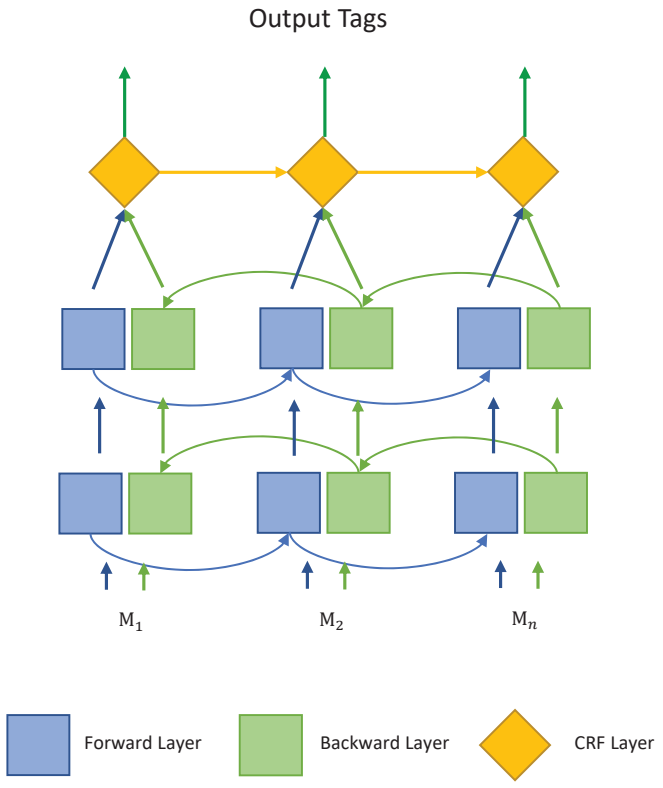

Fig. 4.2. NER Architecture for Location Detection

	B - Location			I - Location				Other- Tags		
	Correct	Wrong	Accuracy	Correct	Wrong	Accuracy	Total Accuracy	Correct	Wrong	Accuracy
Training Data	606	274	68.86	153	66	69.86	69.36	60639	168	99.72
Test Data	136	140	49.28	17	32	34.69	41.98	46047	97	99.79

Fig. 4.3. NER Result

I tested the above architecture on the Oregon State University dataset [50]. The dataset has twenty three different tags. The results obtained are shown in Figure 4.3. The above architecture was able to recognizing location tags with 41.98 % accuracy on the test data. The main reason for getting such poor accuracy is that in the training data only 0.01 % tags belong to the location category, while tags like "other", were in abundance and that's why algorithm was able to predict the "other" tag with an accuracy of 99.79% on the test dataset.

5. SYSTEM WORKFLOW AND IMPLEMENTATION

The system architecture is composed of multiple microservices running independently. A different microservice was used for each task, for example Twitter data collection, machine-learning algorithms, and database operations for the MongoDB and MySQL servers.

Microservice architecture decomposes the whole system into small and lightweight components designed for specific purposes [51]. Some of the significant benefits of this style are:

- Scalability: Since the system architecture is composed of multiple independent services, scaling a particular microservice instance is easier than if the services were intertwined. If a microservice becomes a bottleneck because of slow execution, then that particular microservice can either be run on more powerful hardware or its multiple instances can run in parallel on additional low cost hardware.

- Ease of development: Since every microservice is an independent module, it can be easily removed or upgraded without affecting other services or the whole system.

- Fault-Tolerant: Because of the decoupled nature of the message exchange system, when a specific microservice goes down, it will not bring down the whole system.

- Multi-Language Development: Because each microservice is independently implemented, so one service can be developed using Java [13] while others can be developed using Python [52] or C [53].

This modular design approach is opposite to monolithic system design where every system component is bundled together into a single rigid container and scaling is difficult to achieve. [51].

5.1 System Implementation

The distributed nature of the microservice architecture requires a message passing system for communication among microservices. This communication was implemented using the RabbitMQ [54] message broker. The RabbitMQ message broker uses the Publish And Subscriber Model (PASM) [55]

The PASM utilizes a message broker to route messages from the message exchange queue. The PASM system consists of three types of components:

- Publisher: An application that puts a message on the message exchange.
- Subscriber or Consumer: An application that read the messages from the message queue.
- Exchange: Generally routing from message exchange depends on the exchange type. Two types of exchanges are in common use: fanout exchanges and direct exchanges. In the fanout exchange method, all the messages are copied to all the queues; however, in the indirect exchange method, messages are routed using routing and binding keys.

5.2 System Architecture

The distributed microservice system has two major components as shown in Figure 5.2:

- Data Extraction and Analysis.
- Data Visualization.

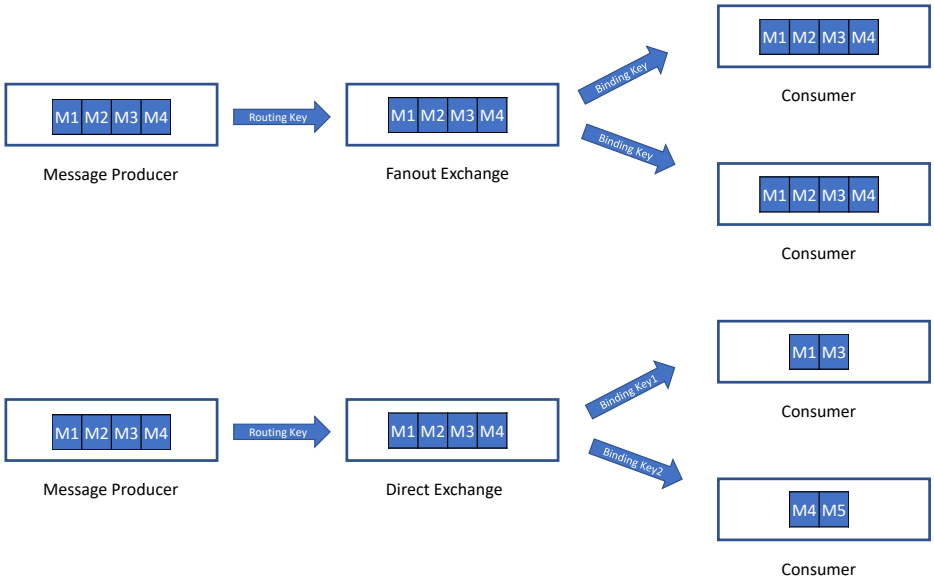

Fig. 5.1. Comparison of Fanout and Direct Message Passing Exchanges

5.3 Data Extraction and Analysis

The data extraction and analysis part of the system does all the heavy lifting for the traffic analysis application. It consists of the Twitter data extractor and the text analyser. More specifically, it consists of the following modules:

- Twitter Data Collection.
- Message Processing and Storage.

5.3.1 Twitter Data Collection

Twitter data is collected by Twitter rest APIs. In every 45 seconds, parameters like user-id, tweet-id, keywords, and city location are passed to the API to get the data (as described in the Chapter 2). Then the data is inserted into the fanout message-exchange queue with three routing keys for the MySQL, MongoDB, and the classification consumers.

5.3.2 Message Processing and Storage

Due to the large number of Twitter messages, they cannot be processed sequentially on a single machine. Using message queues, data is distributed on different servers to do different tasks and no matter how many consumers are attached to the queue, a message is only processed by one consumer.

Classification Consumers

The classification consumer, which performs the Tweet classification, runs a trained RNN model. The output of the classifier is then placed in a MySQL queue.

MySQL Consumers

The MySQL consumer, which performs the MySQL data storage, stores the output of the text classification consumer, the NER, and location triangulation to the MySQL database.

MongoDB Consumer

The MongoDB consumer stores all of the data to the MongoDB database for archival purposes.

5.4 Data Visualization

To communicate with the database of extracted data, a simple user application is built using HTML and Python. The user application interacts with the MySQL database using HTTP requests. Whenever a user requests information, the most recent tweets and their attributes, such as classifier output and location information, are displayed.

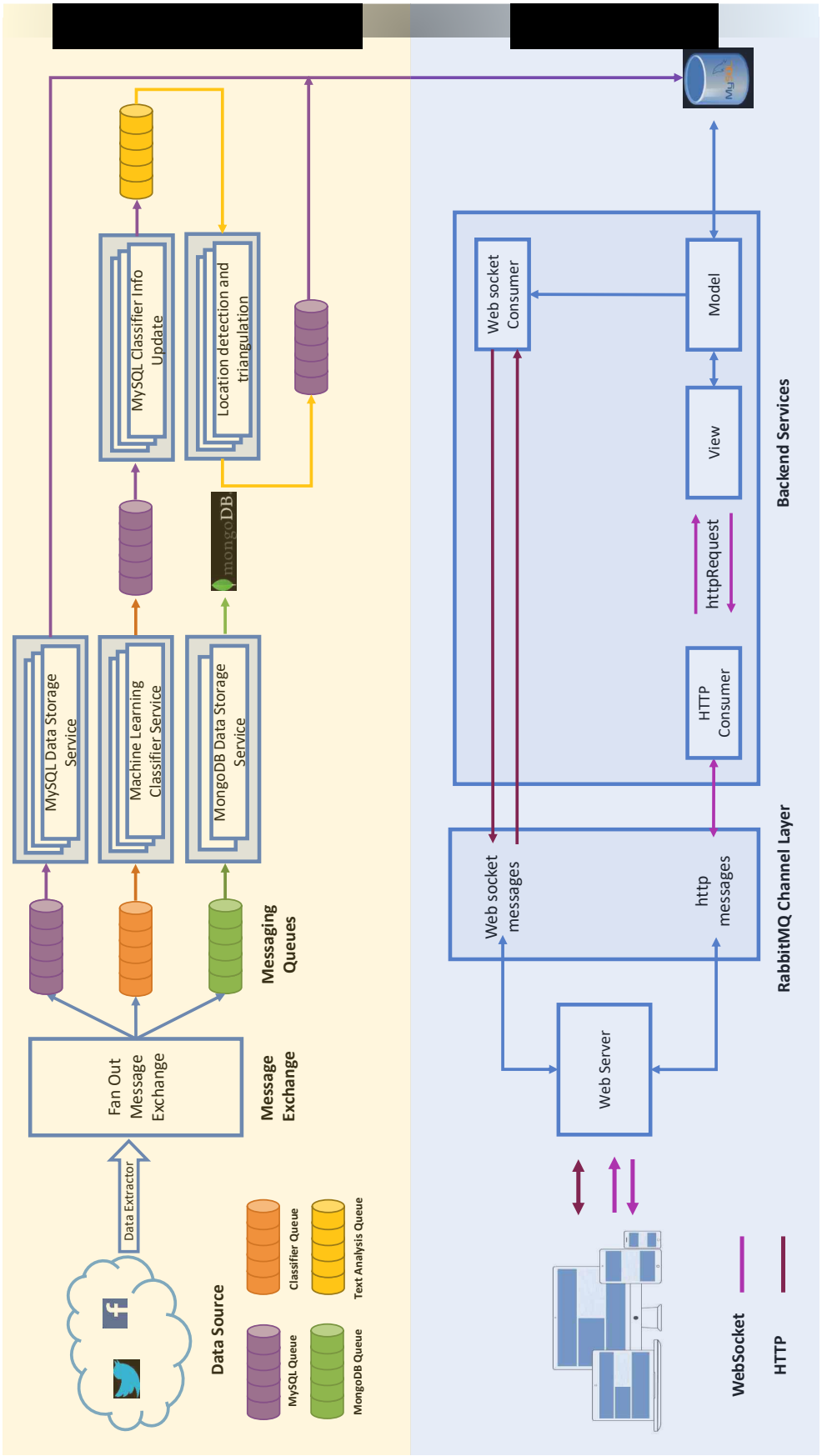

Fig. 5.2. System Architecture

6. SUMMARY

In this thesis, I constructed a novel algorithm that can track and geo-locate traffic incidents from Tweets. The main issues in building such a machine learning model are the quality and quantity of the data. With more data, the accuracy of the classification and Named Entity Recognition tasks can be increased; however more effort is required to label the training data and to optimize the network weights. Also, for the NER, if the dataset has a larger number of sentences with location tags, the current accuracy, 41% can be improved.

The best accuracy, 96% was achieved with the SVM classifier but the SVM was not able to understand the word positions. With the RNN the accuracy was only 93%, RNN was able to get the words dependencies with the surrounding words and this drop was because of the mislabeling of the dataset.

With a larger labeled dataset, we could try some more complex neural networks, which would be expected to increase the classifier and Named Entity recognition task accuracy.

REFERENCES

REFERENCES

[1] Inrix, *Traffic Problem*, 2018 (accessed June 18, 2018). [Online]. Available: http://www.businessdayonline.com/news/article/traffic-jams-cost-u-s-drivers-1200-year-study/

[2] Federal Highway Administration, *Vehicle Detection and Surveillance*, 2018 (accessed July 10, 2018). [Online]. Available: https://www.fhwa.dot.gov/policyinformation/pubs/vdstits2007/03.cfm

[3] A. Roy, N. Gale, and L. Hong, "Automated traffic surveillance using fusion of doppler radar and video information," *Mathematical and Computer Modelling*, vol. 54, no. 1-2, pp. 531–543, 2011.

[4] B. J. Jansen, M. Zhang, K. Sobel, and A. Chowdury, "Twitter power: Tweets as electronic word of mouth," *Journal of the Association for Information Science and Technology*, vol. 60, no. 11, pp. 2169–2188, 2009.

[5] N. Ambraseys, "The seismic activity of the Marmara sea region over the last 2000 years," *Bulletin of the Seismological Society of America*, vol. 92, no. 1, pp. 1–18, 2002.

[6] S. Coleri, S. Y. Cheung, and P. Varaiya, "Sensor networks for monitoring traffic," in *Allerton conference on communication, control and computing*, 2004, pp. 32–40.

[7] D. Barth, *The bright side of sitting in traffic: Crowdsourcing road congestion data*, 2018 (accessed May 20, 2018). [Online]. Available: https://googleblog.blogspot.ca/2009/08/bright-side-of-sitting-in-traffic.html

[8] Twitter, *Get Tweets*, 2018 (accessed June 18, 2018). [Online]. Available: https://developer.twitter.com/en/docs/api-reference-index.html

[9] Tweepy, *Filtering Tweets by location*, 2018 (accessed June 19, 2018). [Online]. Available: https://developer.twitter.com/en/docs/tutorials/filtering-tweets-by-location.html

[10] Twitter, *Get Tweet timelines*, 2018 (accessed May 20, 2018). [Online]. Available: https://googleblog.blogspot.ca/2009/08/bright-side-of-sitting-in-traffic

[11] D. Twitter, *Authentication*, 2018 (accessed June 18, 2018). [Online]. Available: https://developer.twitter.com/en/docs/basics/authentication/guides/access-tokens.htm

[12] Tweepy, *Twitter API access via python*, 2018 (accessed May 21, 2018). [Online]. Available: http://tweepy.readthedocs.io/en/v3.5.0/index.html

[13] Oracle, *Java*, 2018 (accessed June 7, 2018). [Online]. Available: https://java.com/en/

[14] John Paul Mueller, Luca Massaron, *Machine Learning: Creating Your Own Features In Data*, 2018 (accessed June 20, 2018). [Online]. Available: https://www.dummies.com/programming/big-data/data-science/machine-learning-creating-features-data/

[15] I. Guyon and A. Elisseeff, "An introduction to feature extraction," in *Feature extraction*. Springer, 2006, pp. 1–25.

[16] Y. Zhang, R. Jin, and Z.-H. Zhou, "Understanding bag-of-words model: a statistical framework," *International Journal of Machine Learning and Cybernetics*, vol. 1, no. 1-4, pp. 43–52, 2010.

[17] Y. Goldberg and O. Levy, "word2vec explained: Deriving Mikolov et al.'s negative-sampling word-embedding method," *arXiv preprint arXiv:1402.3722*, 2014.

[18] M. Vorontsov, G. Carhart, and J. Ricklin, "Adaptive phase-distortion correction based on parallel gradient-descent optimization," *Optics letters*, vol. 22, no. 12, pp. 907–909, 1997.

[19] K. Bontcheva, L. Derczynski, A. Funk, M. Greenwood, D. Maynard, and N. Aswani, "Twitie: An open-source information extraction pipeline for microblog text," in *Proceedings of the International Conference Recent Advances in Natural Language Processing RANLP 2013*, 2013, pp. 83–90.

[20] N. Wanichayapong, W. Pruthipunyaskul, W. Pattara-Atikom, and P. Chaovalit, "Social-based traffic information extraction and classification," in *ITS Telecommunications (ITST), 2011 11th International Conference on*. IEEE, 2011, pp. 107–112.

[21] B. Sriram, D. Fuhry, E. Demir, H. Ferhatosmanoglu, and M. Demirbas, "Short text classification in Twitter to improve information filtering," in *Proceedings of the 33rd international ACM SIGIR conference on Research and Development in information retrieval*. ACM, 2010, pp. 841–842.

[22] T. Sakaki, M. Okazaki, and Y. Matsuo, "Earthquake shakes Twitter users: real-time event detection by social sensors," in *Proceedings of the 19th international conference on World wide web*. ACM, 2010, pp. 851–860.

[23] C. N. Divij Gupta, *Detecting Real-Time Messages of Public Interest in Tweets*, 2018 (accessed May 21, 2018). [Online]. Available: snap.stanford.edu/class/cs224w-readings/mathioudakis10twitter.pdf

[24] W. Wolny, "Sentiment analysis of Twitter data using emoticons and emoji ideograms," *Studia Ekonomiczne*, vol. 296, pp. 163–171, 2016.

[25] D. R. Cox, "The regression analysis of binary sequences," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 215–242, 1958.

[26] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun, "Support vector machine learning for interdependent and structured output spaces," in *Proceedings of the 21st international conference on Machine learning*. ACM, 2004, p. 104.

[27] E. T. Rolls and A. Treves, *Neural networks and brain function*. Oxford University Press Oxford, 1998, vol. 572.

[28] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Neural networks for perception*. Elsevier, 1992, pp. 65–93.

[29] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," 1999.

[30] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An empirical exploration of recurrent network architectures," in *International Conference on Machine Learning*, 2015, pp. 2342–2350.

[31] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[32] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *arXiv preprint arXiv:1409.2329*, 2014.

[33] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[34] S. Merity, N. S. Keskar, and R. Socher, "Regularizing and optimizing lstm language models," *arXiv preprint arXiv:1708.02182*, 2017.

[35] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *International Conference on Machine Learning*, 2013, pp. 1058–1066.

[36] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[37] T. Joachims, "Optimizing search engines using clickthrough data," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2002, pp. 133–142.

[38] R. Burbidge, M. Trotter, B. Buxton, and S. Holden, "Drug design by machine learning: support vector machines for pharmaceutical data analysis," *Computers & chemistry*, vol. 26, no. 1, pp. 5–14, 2001.

[39] W. Jin, H. H. Ho, and R. K. Srihari, "Opinionminer: a novel machine learning system for web opinion mining and extraction," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2009, pp. 1195–1204.

[40] T. Eftimov, B. K. Seljak, and P. Korošec, "A rule-based named-entity recognition method for knowledge extraction of evidence-based dietary recommendations," *PloS one*, vol. 12, no. 6, p. 12(6), 2017.

[41] C. Bornet and F. Kaplan, "A simple set of rules for characters and place recognition in French novels," *Frontiers in Digital Humanities*, vol. 4, p. 6, 2017.

[42] J. R. Finkel, T. Grenager, and C. Manning, "Incorporating non-local information into information extraction systems by Gibbs sampling," in *Proceedings of the 43rd annual meeting on association for computational linguistics*. Association for Computational Linguistics, 2005, pp. 363–370.

[43] F. A. Elsafoury, *Monitoring urban traffic status using Twitter messages. Thesis submitted to the Faculty of Geo-Information Science and Earth Observation of the University of Twente, the Netherlands*, February, 2013.

[44] Z. Cheng, J. Caverlee, and K. Lee, "You are where you tweet: a content-based approach to geo-locating twitter users," in *Proceedings of the 19th ACM international conference on Information and knowledge management*. ACM, 2010, pp. 759–768.

[45] A. Ritter, S. Clark, O. Etzioni *et al.*, "Named entity recognition in tweets: an experimental study," in *Proceedings of the conference on empirical methods in natural language processing*. Association for Computational Linguistics, 2011, pp. 1524–1534.

[46] Twitter, *Get Tweet timelines*, 2018 (accessed May 20, 2018). [Online]. Available: https://cs.nyu.edu/grishman/jet/guide/PennPOS.html

[47] J. Gelernter and S. Balaji, "An algorithm for local geoparsing of microtext," *GeoInformatica*, vol. 17, no. 4, pp. 635–667, 2013.

[48] M. Collins, *Log-Linear Models, MEMMs, and CRFs*, 2018 (accessed June 7, 2018). [Online]. Available: http://www.cs.columbia.edu/ mcollins/crf.pdf

[49] Z. Huang, W. Xu, and K. Yu, "Bidirectional lstm-crf models for sequence tagging," *arXiv preprint arXiv:1508.01991*, 2015.

[50] Alan Ritter, *W-NUT data*, 2018 (accessed February 17, 2018). [Online]. Available: https://bit.ly/2LeuxK9

[51] J. L. Martin Fowler, *Microservices: a definition of this new architectural term*, 2018 (accessed June 7, 2018). [Online]. Available: https://www.martinfowler.com/articles/microservices.html

[52] P. Community, *Python*, 2018 (accessed June 7, 2018). [Online]. Available: https://www.python.org

[53] C. Community, *C*, 2018 (accessed June 7, 2018). [Online]. Available: http://www.cplusplus.com/reference/

[54] rabbitmq, *Rabbitmq Message Broker*, 2018 (accessed June 7, 2018). [Online]. Available: https://www.rabbitmq.com/

[55] E. Fidler, H.-A. Jacobsen, G. Li, and S. Mankovski, "The padres distributed publish/subscribe system." 2005, pp. 12–30.

APPENDIX

A. APPENDIX

A.1 City Names and Traffic Related Key Words

Table A.1. Traffic Related Key Words

	Key Word
1	#traffic
2	marriage
3	happy
4	rain
5	water
6	drain
7	traffic
8	road
9	block
10	road accidents
11	accidents
12	congestion
13	construction
14	frustrated
15	stuck

Table A.2. Cities' Names and Geo Locations

	City Name	Geo Coordinate
1	Chicago	41.881832, -87.627760
2	Chennai	13.083162, 80.282758
3	New York	40.714264, -73.978499
4	London	51.505234, -0.111244
5	New Delhi	28.612952, 77.211953
6	Indianapolis	39.767927, -86.158749
7	Bombay	19.110914, 72.885140
8	New Jersey	40.279865, -74.517549

A.2 Conventions

This chapter summarizes all the conventions that have been used in this Thesis

- Vector Dot Product It is represented by "$\cdot$" Ex. $\mathbf{w} \cdot \mathbf{M}$
- Vector with square bracket represent the vector at specific layer, Ex. $\mathbf{F}[\mathbf{i}]$, feature vector at layer i.
- Vector component-wise multiplication is $\mathbf{w}^i\mathbf{F}^i$