ABSTRACT

**A COMPARATIVE STUDY OF RUSSIAN TROLLS USING SEVERAL
MACHINE LEARNING MODELS ON TWITTER DATA**

by
Kannan Neten Dharan Kannan Neten Dharan

Ever since Russian trolls have been brought into light, their interference in the 2016 US Presidential elections has been monitored and studied thoroughly. These Russian trolls have fake accounts registered on several major social media sites to influence public opinions. Our work involves trying to discover patterns in these tweets and classifying them by using different machine learning approaches such as Support Vector Machines, Word2vec and neural network models, and then creating a benchmark to compare all the different models. Two machine learning models are developed for this purpose. The first one is used to classify any given specific tweet as either troll or non-troll tweet. The second model classifies specific tweets as coming from left trolls or right trolls, based on apparent extreme political orientation. Several kinds of statistical analysis on these tweets are performed based on the tweets and their classifications. Further, an analysis of the machine learning algorithms, using several performance criteria, is presented.

A COMPARATIVE STUDY OF RUSSIAN TROLLS USING SEVERAL MACHINE LEARNING MODELS ON TWITTER DATA

by
Kannan Neten Dharan Kannan Neten Dharan

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

Department of Computer Science

August 2019

.

A COMPARATIVE STUDY OF RUSSIAN TROLLS USING SEVERAL MACHINE LEARNING MODELS ON TWITTER DATA

Kannan Neten Dharan Kannan Neten Dharan

Dr. James Geller, Thesis Advisor   Date
Professor of Computer Science, NJIT

Dr. Soon Ae Chun, Co-Advisor   Date
Professor and Director of Information
Systems and Informatics Program,
College of Staten Island, NY

Dr. Senjuti Basu Roy, Assistant Professor   Date
Department of Computer Science, NJIT

BIOGRAPHICAL SKETCH

Author: Kannan Neten Dharan Kannan Neten Dharan

Degree: Master of Science

Date: August 2019

Undergraduate and Graduate Education:

- Master of Science in Computer Science,
  New Jersey Institute of Technology, Newark, NJ, 2019

- Dual Degree- Bachelor of Technology in Naval Architecture and Ocean
  Engineering with Master of Technology in Applied Mechanics,
  Indian Institute of Technology Madras, Chennai, India, 2016

Major: Computer Science

DEDICATION

I dedicate this thesis to my parents, Malathi Kannan and Kannan Manickam, and my brother, Tharani Dharan who have always supported me, no matter what decision I take. Their love, care and teaching over many years laid the foundations for the discipline and application necessary to complete this work.

ACKNOWLEDGMENT

I would like to express my gratitude to my thesis advisor, Professor James Geller and my co-advisor Professor Soon Ae Chun, who have always been there to guide me throughout the process of writing this thesis. They have always been supportive and encouraging.

I would also like to thank Ruoyu Wang, PhD Student at NJIT who has helped me with this project and also Prof. Richard Holowczak at CUNY Baruch College for sharing large datasets of tweets for analysis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF DEFINITIONS

Russian trolls   Russian trolls are state-sponsored anonymous Internet political commentators and trolls linked to the Russian government

Geospatial analysis   Geospatial analysis is the collection of data with location information, which can be used for gaining further insights

Temporal analysis   Temporal analysis is collection of data which can be analyzed to look for patterns based on time.

Neural Networks   Neural Networks are computing systems vaguely inspired by the biological neural networks that constitute animal brains. These networks can be used to replicate some of the learning behavior of the brain.

CHAPTER 1

INTRODUCTION

Twitter data has been widely used in many of text mining projects. Unlike other social media platforms, tweets are public and easy to retrieve from Twitter. Twitter has very good APIs (Application Programming Interfaces) that help users to retrieve data in a methodological way, which helps us to fetch data according to specific geographic region, tweets in a specific timeframe, etc. We can fetch data from a targeted set of users as well. That makes Twitter an excellent platform to work with.

Ever since Russian trolls have been brought into light during the 2016 US Presidential elections, the influence of trolls has been studied in Computer Science. However, there is no clear definition of what a troll is. Most authors assume that it is obvious and their meaning of the word troll can only be inferred from their treatment of the subject.

(Mojica, 2016) use of the word "troll" focuses on determining the intentions of the user, whether the user is attempting to keep their intention hidden, how the posts were interpreted by other users, and what the reactions are to specific posts.

Kumar et al. (2014) [6] use the term "trolling" when a user posts and spreads information that is deceptive, inaccurate, or outright rude. Trolling can be performed not only on social media networks but also on any editable and viewable content on the web, such as Wikipedia or YouTube. The authors develop an algorithm called TIA, Troll Identification Algorithm, in order to classify such users as malicious or benign. This study is more focused on the integrity of the network that the trolls are working on. Thus, anyone who posts information that is incorrect may be a troll, unlike in (Mojica, 2016), where the

intentions of a user are the focus, so that any user who had malicious intentions is a troll. In addition, if non-troll users make negative comments or posts, they are also considered trolls. The requirement for being classified as a troll is not only based on the users' own posts, but also on the responses.

In our work we want to leverage the data from Twitter and use it to identify Russian troll tweets. More specifically, we will be using Russian troll tweets to build a classification model that can classify any specific tweet as either being from a Russian troll or not.
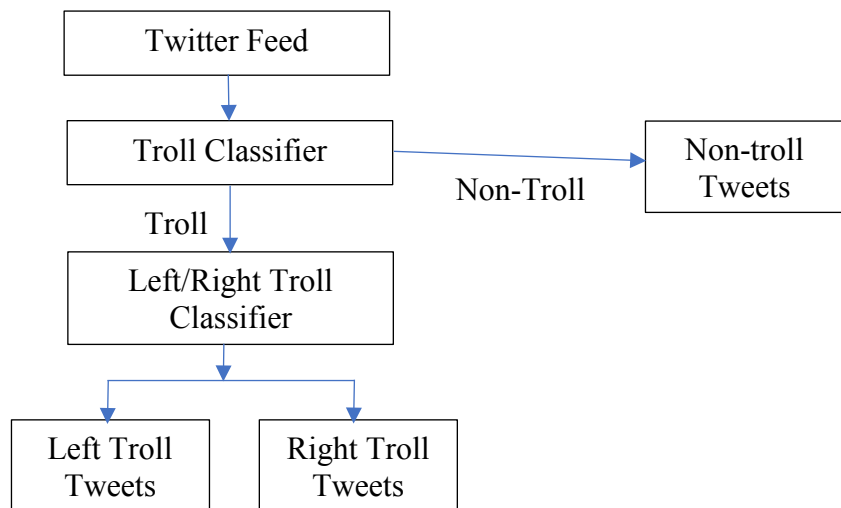
Figure 1.1 Flow diagram for troll Detector model and Left Right Troll Classifier Model.

An initial review indicated that not all Russian trolls are of the same kind. Specifically, it turned out that some of the rolls indicate a "left" political orientation, while other trolls appear to be politically at the right end of the spectrum. Therefore, after building a machine learning model that distinguishes between troll and non-troll tweets we built another model

that separates Left trolls from Right trolls. Figure 1.1 shows an overall process flow. The box marked as Troll Classifier is the result of running a machine learning model on data that was already classified by humans. Finding this model was a part of my research. This kind of machine learning algorithm is called "supervised learning." Similarly, the box marked as Left/Right Troll Classifier incorporates a machine learning model to recognize the political orientation of a troll tweet. This model was also created in this research by using supervised machine learning.

CHAPTER 2

DATASETS USED FOR BUILDING THE MODELS

We have used four different datasets in this project. We will now discuss these datasets in detail.

2.1 News Feed Dataset

For building our troll model, we used a dataset obtained from an online news portal by the name of FiveThirtyEight (https://fivethirtyeight.com/). We obtained around 3 million tweets from this web portal. Those tweets in turn were collected from accounts associated with the "Internet Research Agency." The Internet Research Agency (IRA) is a front organization for a company paid by the Russian government to sow disinformation (https://en.wikipedia.org/wiki/Internet_Research_Agency). The data is completely open source licensed and it includes 2,973,371 tweets from 2,848 Twitter handles. It includes every tweet's author, text and date; the author's follower count and the number of accounts the author followed; and an indication of whether the tweet was a retweet. Authors are in practice not real names but fabricated personas. For future reference, we will call this dataset the FiveThirtyEight Dataset

Table 2.1   Relevant Columns Extracted from the News Feed Dataset

author	content	publish_date	harvested_date	account_type

The 'author' column contains the twitter id of the user. The 'content' column contains the tweet posted by the author. The 'publish_date' is the date on which the tweet was published, whereas the 'harvested_date' is the date on which the tweets were retrieved. The column 'account_type' tells us the kind of tweet it is, that is if it's a left, right and

several other classifications. There are a lot of tweets that are in Russian as well. These tweets are filtered using the 'account_type' column. For all the tweets, which are in Russian, they have an 'account_type' as Russian.

2.2 Twitter Data from Baruch College

Three more Twitter datasets were derived, collected and maintained at Baruch College(Holowczak) (http://optionsdata.baruch.cuny.edu/data1/delivery/twitter/). The three datasets differ in size, the smallest containing 1 million tweets, the largest 20 million tweets and an intermediate dataset comprising 5 million tweets. There is no evidence that every one of those tweets is indeed a troll tweet. However, these tweets were retrieved from users (Twitter handles, really) that have been known to generate troll messages. For future reference, we will call this dataset the Baruch Dataset.

CHAPTER 3

DATA PREPROCESSING

When working on text classification, it is the first step to preprocess the data, and the preprocessing part is varied for every dataset. For this, we substitute URLs in tweets with the word URL, and also make it into one of the stop words, so that it does not interfere with our classification model or affect its performance.

We removed Twitter handles that appear to be irrelevant to the classification, and we remove the Non-ASCII characters from the tweets. For removing Non-ASCII characters, the 'Pandas' package of python was used (https://pandas.pydata.org). The code for removing Non-ASCII characters is as follows:

```
def preprocessTweets(tweet):
    tweet = re.sub("[^a-zA-Z]", str(tweet))
tweet_processed = stem(preprocessTweets(tweet))
```

It is imperative that we deal with the stop words and make sure they are removed from tweets. Stop words are words like "the, a, of, her, he, she, etc." that might be of no significance to the classification. Stop words usually slow down the process of classification. The data preprocessing is explained in more detail in Chapter 4.

CHAPTER 4

CLASSIFICATION MODELS

The implementation of the troll vs. non-troll and right troll vs. left troll classifiers was done using different models to compare the performance. For a machine learning algorithm, to derive a classifier, positive and negative instances are necessary. We have positive instances in our FiveThirtyEight data set. However, we needed to generate a negative data set of the same size. For this we fetched 3 million random tweets which are used as the negative tweets for our model. These 3 million tweets were retrieved from several Twitter feed sites (https://www.researchgate.net, http://followthehashtag.com/datasets/free-twitter-dataset-usa-200000-free-usa-tweets/) and also using the Tweepy API (https://github.com/tweepy/tweepy) to fetch real time tweets. For future reference, we will call this dataset as 'random tweets dataset.' Our Troll Detection model was then built using the 6 million troll tweets from the FiveThirtyEight and the random tweets dataset.

While the dataset contains 19 columns, only those in Table 2.1 were used in this research. For building the classification model, only the content column was used. We filtered out all the irrelevant columns that were found in our dataset. We created a new column assigning a score to each tweet. If it is a Russian troll tweet, the score is '1.' For the random tweet, the value is '0.'. There only need to be two specific columns for our classification. One is the tweet content and the other is the score of the tweet. Table 4.1 shows an example taken from the dataset with the specific columns that we used for classification.

Table 4.1 Examples of the News Feed Dataset with Sentiment Score Used for Building Our Model

Score	content
0	I'm at Apple Store, Pheasant Lane in Nashua, NH https://t.co/E5FCrUFEpL
1	Demand paper #VoteTrump #MAGA https://t.co/YywhqRJ6DR #TrumpForPresident
1	The superior man acts before he speaks, and afterwards speaks according to his action.
1	Is Eva Braun opening for Hillary Clinton? https://t.co/Asmktt8imd
0	Super excited to continue to play basketball at KCC next year with,Ä¶ https://t.co/QNWtA1bz08

4.1 Support Vector Machine Classifier

SVM (Support Vector Machines) is a supervised learning classification technique. Word2vec converts words and phrases into a vector representation. The Support Vector Machine conceptually implements the following idea: input vectors are non-linearly mapped to a very high-dimension feature space [Vapnik, 1995]. SVM has been proven effective on many text categorization tasks.

4.1.1 Hyperplanes and Support Vectors

In SVMs, we try to find a hyperplane in an N-dimensional space that can be used to classify the data points distinctly. Support Vectors are the data points that affect the position of the hyperplane. They are the data points nearest to the hyperplane.
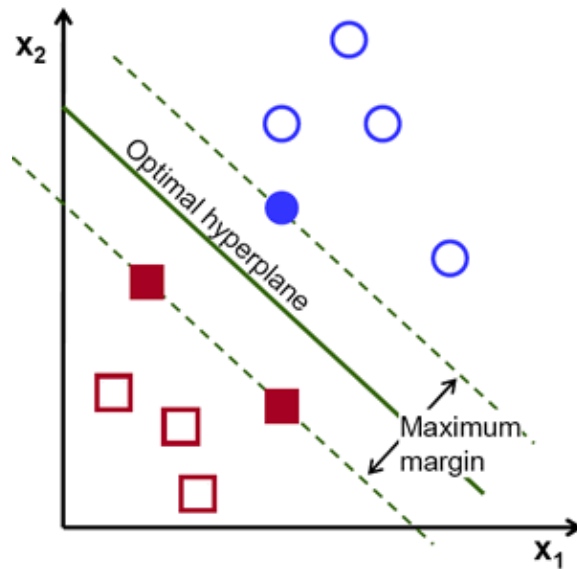
Figure 4.1 Optimal Separating Hyperplane of Support Vector Machines in 2D.

Source: Optimal Hyperplane, Introduction to Support Vector Machines
https://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html

Usually, there are several possible hyperplanes that can be used to classify a dataset into two different classes. The main objective of the SVM algorithm is to find a hyperplane with the maximum margin between the data points as shown in Figure 4.1. Finding a hyperplane with the maximum margin ensures that when this model is used to classify new data points, it will be able to classify them accurately. In Figure 4.1, the filled diamonds and the filled circles represent the Support Vectors.

4.1.2   Data Preprocessing

We apply several preprocessing steps to make tweets suitable for the SVM algorithm. We initially remove all non-letter and Non-ASCII characters and replace them by blank spaces (see Chapter 3). We also remove all URLs and twitter handles that might affect the model. We delete emoticons from our dataset, since we are not taking them into consideration for our model.

As the next step, we do stemming on the dataset. Stemming usually works by removal of either the end or the beginning of a word, by using a set of prefixes and suffixes that could be found in a word. We make use of the most common stemmer in English, the Porter Stemmer (http://snowball.tartarus.org/algorithms/porter/stemmer.html).

Word	Stem	Word	Stem
magically	magic	groveling	grovel
chewing	chew	painful	pain
unequal	unequ	daguerreotype	daguerreotyp
shoddiness	shoddi	magnitude	magnitud
headline	headlin	standing	stand
ruinously	ruinous	obstruction	obstruct
allergenic	allergen	bagpiper	bagpip
signified	signifi	disunite	disunit
truancy	truanci	tensely	tens
shiftiness	shifti		

Figure 4.2   Examples of stemming of words using Porter Stemmer.

Source: Generate and Verify Stemmed Words
https://www.wolfram.com/language/11/text-and-language-processing/generate-and-verify-stemmedwords.html?product=language

4.1.3   TF-IDF Vectorizer

To construct the model we use a tf-idf (Term Frequency—Inverse Document Frequency) vectorizer to convert the raw text data to matrix features. Term frequency (tf) gives us the frequency of a word in each document of our entire corpus. It is given by the ratio of number of times the word appears to the total number of words in one document. Inverse Document Frequency (idf) tells us how much information is provided by a word, based on whether its common or rare across all documents.

By combining tf and idf, we come up with a tf-idf score, which is computed for each word in a document in the corpus. This is used to compute the significance of words, which helps us in classifying tweets.

4.1.4   Building the SVM classifier

We build the SVM model using the FiveThirtyEight dataset, which is then stored using a Python Library by the name of 'pickle' (https://docs.python.org/3/library/pickle.html). The stored model can be called again later for classification of new data.

After the FiveThirtyEight dataset has been preprocessed, we build our model. Our model uses an SVM implementation using scikit-learn named SVC (Support vector classification). It is an implementation based on libsvm [Chih-Chung Chang, 2001].

The dataset is divided into training and testing samples for cross validation. The training set will help the algorithm learn about the dataset whereas the test set will be used for the evaluation. We use 20% of the dataset as the test data. The following code is used for splitting our data into training and testing:

```
from sklearn.cross_validation import train_test_split
def getTrainingAndTestData():
.....

    X_train, X_test, y_train, y_test = sklearn.cross_validation.
    train_test_split(X,y,test_size=0.20, random_state=42)
```

Since we are dealing with a text categorization problem, we make use of a linear SVM classifier. Here, we set the regularization Parameter, C = 0.1. The regularization

parameter is used to control the trade-off between misclassifications. The higher the value, the fewer misclassifications are allowed. In our case, since our regularization parameter is very small, misclassifications are allowed, but training is relatively faster. As this dataset is very large, it is necessary to speed up training.

We use an rbf kernel for our SVM model (Radial basis function kernel) (https://chrisalbon.com/machine_learning/support_vector_machines/svc_parameters_usin g_rbf_kernel/).

4.2 Neural Network Classifier with One Hot Encoding

In machine learning, we have two ways of representing language: vector embeddings or one-hot matrices.

4.2.1 One-Hot Encoding

One-hot matrices contain no linguistic information. They indicate what words they contain but suggest nothing about them, or their relationships to each other.

The creation of one hot matrices begins with tokenizing the sentence, that is, breaking it into words. Then we create a lookup dictionary of all the unique words, which need not have a count or an order. Figure 4.3 shows us how categorical sentences are converted into one hot encoded data.

Step 1: Retrieve all sentences from the dataset:	Step 2: Tokenizing the utterance:
Complex is better than complicated. Flat is better than nested.	['complex', 'is', 'better', 'than', 'complicated', 'flat', 'is', 'better', 'than', 'nested']

Step 3: Create lookup dictionary of all the unique words:
 	
{
  'complex': 0,
  'is': 1,
  'better': 2,
  'than': 3,
  'complicated': 4,
  'flat': 5,
  'nested': 6,
}
	Step 4: Convert individual sentences to one hot matrices
	
	[1, 0, 0, 0, 0, 0, 0], #complex
	[0, 1, 0, 0, 0, 0, 0], #is
	[0, 0, 1, 0, 0, 0, 0], #better
	[0, 0, 0, 1, 0, 0, 0], #than
	[0, 0, 0, 0, 1, 0, 0], #complicated
	
	[0, 0, 0, 0, 0, 1, 0], #flat
	[0, 1, 0, 0, 0, 0, 0], #is
	[0, 0, 1, 0, 0, 0, 0], #better
	[0, 0, 0, 1, 0, 0, 0], #than
	[0, 0, 0, 0, 0, 0, 0], #nested

Figure 4.3   Example of the process of sentence conversion into one hot matrices.

The maximum length of a sentence is set to be equal to 3000. We converted each of the tokens from a string into an array. Each array is of the length of the dictionary (3000 words), and each value in the dictionary that is not the value of the current token is represented by a 0. The value of the token is represented with a 1.

4.2.2 Data Preprocessing

For building our model, we only use two columns which are the text (content column) and the predefined classification (score column) (Table 4.1).

We initially create and save a dictionary of all the words. For this, we feed all our sentences to a tokenizer. A tokenizer breaks sentences into words, called tokens, and at the

same time removes all punctuations. These tokenized words are then saved into a dictionary. Then, we convert each sentence into an indexed array.

4.2.3 Building the Neural Network classifier with One Hot Encoding

Next, we build a sequential classifier, which is a simple neural network model, which consists of a stack of hidden layers that are executed in a specific order. We have one dense layer and two dropout layers.

Dense neural network layers are linear neural network layers that are fully connected. In general, in a dense layer, every input is connected to every output by a weight. A dense layer is usually followed by a non-linear activation function.

Dropouts are randomly used to remove data, which prevents overfitting. This helps in our case, since if we keep training the model with the same data, the accuracy will not increase, but will stay constant or it might even decrease.

Activation functions of a node are usually defined as the outputs of that node, when given any specific input or set of inputs. The output of the activation function is then used as the input for the next node. Figure 4.4 illustrates some of the most common activation functions.

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \dfrac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \dfrac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \dfrac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \dfrac{1}{1 + e^{-x}}$

Figure 4.4 Activation functions for Neural networks with their equations and derivations.

Source: Activation Function Cheatsheet, Activation Functions in Neural Networks
https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

In our first input layer, we make use of the Rectified Linear Unit (ReLu) activation function and 512 outputs come out of that layer. Our second layer, which is a hidden layer, consists of a sigmoid activation function and has 256 outputs. Our final layer, which is the output layer consists of SoftMax activation functions. The code below shows how the layers have been created in our model:

```
model = Sequential()

model.add(Dense(512, input_shape=(max_words,), activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(256, activation='sigmoid'))

model.add(Dropout(0.5))

model.add(Dense(2, activation='softmax'))
```

We make use of a categorical crossentropy loss function. This loss function is also called Softmax loss function. It measures the performance of classification model whose output is a probability value between 0 and 1. We also compute several other metrics, while building our model whose output is shown in Table 4.1 for both the models.

Table 4.1 One Hot Matrices Model Classifier Results

	Accuracy	Loss	Precision	Recall
Neural Network Troll Detector	74%	41%	76%	76%
Left or Right Troll Detector	84%	32%	88%	90%

We use smaller batch sizes of 32 sentences to train our data so that we check our model's accuracy and we are able to modify the node weights. Smaller batches make it faster and easier to train the dataset. We run 5 epochs while training, where epochs are the number of times our model goes through the entire dataset for training. We also observed that 6 epochs lead to overfitting, hence we used 5 epochs. We run our model using keras (https://keras.io) with Tensorflow (https://www.tensorflow.org/install) backend .

The code below shows how our model has been compiled and fit for training. The term "fit" is used for executing the training process by feeding the training data to the machine learning algorithm.

```
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
model.fit(train_x, train_y, batch_size=32, epochs=5,
verbose=1,validation_split=0.1,shuffle=True)
```

Once our model is constructed, it is saved in two different parts. One part contains the model's structure, and the other part consists of the model's weights. After this, the model can be used for predicting categories of tweets on a new dataset and the results will be discussed in Chapter 5.

The code below shows how the model is saved:

```
model_json = model.to_json()
with open('model.json', 'w') as json_file:
    json_file.write(model_json)
model.save_weights('model.h5')
```

4.3 CNN Neural Network Classifier with Word2vec Representation

Vector embeddings are spatial mappings of words or phrases. Relative locations of words indicate similarity and suggest semantic relationships — for instance, vector embeddings can be used to generate analogies.

4.3.1 Convolutional Neural Network (CNN)

Convolutional Neural Networks [LeCun, 1995] are a supervised machine learning algorithm which is mainly used for classification and regression. CNNs usually require very little preprocessing as compared to any other neural networks. Though CNNs are meant to be used for analyzing visual imagery, they have been shown to be effective even in Natural Language Processing (https://www.aclweb.org/anthology/D14-1181).

A CNN consists of input, output and multiple hidden layers. The inner layers which are the hidden layers generally comprise convolution layers.

4.3.2 Word2vec Models

Word2vec models are two-layered neural network models that are trained to reconstruct the linguistic contexts of the word. Word2vec takes a large amount of text and produces a vector space, consisting of several dimensions.

Word2vec models can be trained and constructed to create word embeddings for entire documents. Word2vec can group vectors of similar words together into a vector space. With enough data, Word2vec models can constrain the meaning of a word using past appearances. The output of a Word2vec neural network model is a vocabulary where each item has a vector attached to it, which can be used to query for relationships between words.

4.3.3 Data Preprocessing

For building our model, we only use two columns exactly as in our previous neural network model, which are the text (content column) and the predefined classification (score column) (Table 4.1).

The preprocessing steps are very similar to those of the neural network model. We initially create and save a dictionary of all the words. For this, we feed all our sentences to a tokenizer. A tokenizer breaks sentences into words, called tokens and at the same time removes all punctuations. These tokenized words are then saved into a dictionary. Our tokenizer also filters out all the irrelevant Non-ASCII characters.

The code below shows how the tokenization of the text is done using the keras text preprocessing function:

```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
...

tokenizer = Tokenizer(num_words=NUM_WORDS, filters='!"#$%&()*+,-./:;
<=>?@[\\]^_`{|}~\t\n\'',
                      lower=True)
tokenizer.fit_on_texts(texts)
sequences_train = tokenizer.texts_to_sequences(texts)
sequences_valid=tokenizer.texts_to_sequences(val_data.text)
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
```

4.3.4 Building the CNN classifier using existing Word2vec model

For building our classifiers, we use an existing Word2vec model, by the name of 'GoogleWord2Vec.' It is a huge model created by Google, which comprises a vocabulary of about 3 million words and phrases, which was trained on a Google news dataset of roughly 100 billion words. The length of the vectors is set to 300 features.

Our models pre-processing is exactly the same as for the previous model. In this model, instead of one hot encoding, we make use of Word2vec embeddings. Since Word2vec has around 3 million words and phrases, we cut it down to around 20,000 words, by looking for words common in both our dataset and the Word2vec model. Our embedding dimension is equivalent to the length of the vectors which is 300.

The code for the different layers of our model looks as follows:

```
inputs = Input(shape=(sequence_length,))
embedding = embedding_layer(inputs)
reshape = Reshape((sequence_length,EMBEDDING_DIM,1))(embedding)

conv_0 = Conv2D(num_filters, (filter_sizes[0], EMBEDDING_DIM),activa
tion='relu',kernel_regularizer=regularizers.l2(0.01))(reshape)
conv_1 = Conv2D(num_filters, (filter_sizes[1], EMBEDDING_DIM),activa
tion='relu',kernel_regularizer=regularizers.l2(0.01))(reshape)
conv_2 = Conv2D(num_filters, (filter_sizes[2], EMBEDDING_DIM),activa
tion='relu',kernel_regularizer=regularizers.l2(0.01))(reshape)

maxpool_0 = MaxPooling2D((sequence_length - filter_sizes[0] + 1, 1),
strides=(1,1))(conv_0)
maxpool_1 = MaxPooling2D((sequence_length - filter_sizes[1] + 1, 1),
strides=(1,1))(conv_1)
maxpool_2 = MaxPooling2D((sequence_length - filter_sizes[2] + 1, 1),
strides=(1,1))(conv_2)
```

We have a total of three convolutional layers other than the input and the output layers, and we use ReLu as the activation function for all of them. The three sequences have the same number of filters, which is equivalent to the total number of data points in the training data. The filter sizes for the three convolutional layers are 3, 4 and 5 respectively. The activation function in the final output dense layer is SoftMax, and the number of word embeddings we are using is around 20,000.

We run this model over 10 epochs, and we have a training to validation data split of 80:20. We run our model using keras with Tensorflow backend, which has its own loss

function and optimization function for computing the accuracy and loss. The code for our model to train the model and compute accuracy is as follows:

```
adam = Adam(lr=1e-3)

model.compile(loss='categorical_crossentropy',optimizer=adam,
              metrics=['acc'])
callbacks = [EarlyStopping(monitor='val_loss')]
model.fit(X_train, y_train, batch_size=1000, epochs=10, verbose=1, v
alidation_data=(X_val, y_val),
          callbacks=callbacks)  # starts training
```

4.4 CNN Neural Network Classifier with State-of-the-art NLP Model BERT

BERT (Bidirectional Encoding Representations from Transformers) applies bidirectional training of "transformers" to language modelling. Transformer is used for converting a sequence using encoder and decoder to another sequence. BERT is the first deeply bidirectional model and has a language representation that is unsupervised. It has been pre-trained using only a plain Wikipedia text corpus.

In a normal context free model, the system generates a single word embedding representation for each word in the vocabulary, whereas contextual models usually generate a representation of each word that is based on other words in the sentence. However, it does it only in one direction. BERT uses a bidirectional contextual representation that is, it uses both the previous and next context in a sentence before or after a word respectively.

4.4.1 Data Pre-processing

The pre-processing for the BERT model is very similar to that of the CNN model we used in our previous sections.

4.4.2 Building the CNN model using BERT

We make use of the BERT-base, Multilingual Cased model which has 12 layers; 768 is the size of the hidden encoder and pooling layers and in all there are 110 Million parameters. A Cased model preserves the true case (cased words) and the accent markers.

We train our model for three epochs with batch size of 32 and sequence length of 512. The learning rate is 2e-5. The code below explains how the BERT model was configured to run using keras with Tensorflow backend:

```
class BertConfig(object):
  """Configuration for `BertModel`."""
...
def __init__(self, vocab_size,hidden_size=768,num_hidden_layers=12,
             num_attention_heads=12,intermediate_size=3072,
             hidden_act="gelu",hidden_dropout_prob=0.1,
             attention_probs_dropout_prob=0.1,
             max_position_embeddings=512,type_vocab_size=16,
             initializer_range=0.02):
```

In the code above, the max_position embedding is set to 512, which is the maximum sequence length, which means that a specific tweet can have a maximum length of 512 characters. Everything larger than 512 characters is ignored. The num_attention_heads is set to 12, which is the 12-head attention mechanism. In this mechanism, since our vector dimension is 512, the vector is split into 12 chunks, each having a dimension of 512/12 = 42 (42.666…) and it applies this for each attention layer in the Transformer encoder. We make use of an Adam optimizer which is the default optimizer for BERT. Adam is an alternative to Stochastic Gradient Descent (SGD), which is used to update network weights iteratively when training with data. A learning rate is

maintained for each network weight (parameter) and separately adapted as learning unfolds.

The model is saved with all the checkpoints for future use for classification and is also evaluated. The evaluation results of the BERT models are shown in Table 5.1 in Section 5.

CHAPTER 5

RESULTS

Table 5.1 illustrates the different metrics of the different machine learning models that we created.

Table 5.1 Comparing the Accuracy, Precision and Recall of All Our Machine Learning Models

Model Type	Classifier Type	Accuracy	Precision	Recall
SVM Model	Troll Detector	84%	85%	86%

	Left or Right Troll Detector	86%	88%	91%
CNN with One Hot Encoding	Troll Detector	74%	78%	81%
	Left or Right Troll Detector	84%	85%	86%
CNN with Word2vec Representation	Troll Detector	56%	-	-
	Left or Right Troll Detector	85%	-	-
CNN with BERT model	Troll Detector	**99%**	98%	99%
	Left or Right Troll Detector	**89%**	89%	90%

In Table 5.1, we see that the CNN BERT model outperforms both the CNN models for distinguishing between left trolls and right trolls and also the SVM model, with an accuracy of 89.4%. Similarly, for the troll vs. non-troll detector, the CNN BERT model has an accuracy of 99%.

SVM models usually work better than most of the other models on smaller datasets, whereas neural network models work better on larger datasets, because these models usually try to capture the semantic relationships of words. In two of our experiments (Table 1), we used one hot encoding, which does not make use of linguistic information.

Another drawback of one-hot encoding is that it increases the dimensionality by a large amount. In our case, we used only the 3000 most commonly occurring words in the training corpus, and hence each one hot encoding representation becomes an array of 3000 words, which is very large.

From Table 5.1 it can be seen that the performance of CNN with one-hot matrix is not as good as the SVM Troll Detector model. The BERT model completely outperforms any other model in both the tasks.

The SVM models that we built were used on the Baruch 1 million tweet dataset and the following results were obtained (Table 5.2).

Table 5.2 SVM Model Classifier Results on Baruch One Million Dataset

Tweet Type	Count
Total tweets	1000000
Total Troll Tweets	730215
Left Troll Tweets	183785
Right Troll Tweets	546430

When we used the CNN neural network with One Hot Matrices representation, the results that were obtained are as shown in Table 5.3.

Table 5.3 CNN One Hot Matrices Model Classifier Results on Five Million Dataset

Tweet Type	Count
Total tweets	5000000
Total Troll Tweets	3586213
Left Troll Tweets	1032422
Right Troll Tweets	2553791

5.1 Geospatial Analysis and Temporal Analysis on Baruch Dataset

We make use of the 5 million tweets Baruch dataset in this Section. We use our troll/non-troll detector to classify the tweets into troll or non-troll. Then we use the left/right troll detector to classify them further.

After classifying the tweets, we performed a temporal analysis on this dataset to understand how the ratio of troll to non-troll tweets has changed over time. Figure 5.1 shows the temporal analysis that was performed on the tweets and how the tweet activity has changed from the year 2014 to 2018.
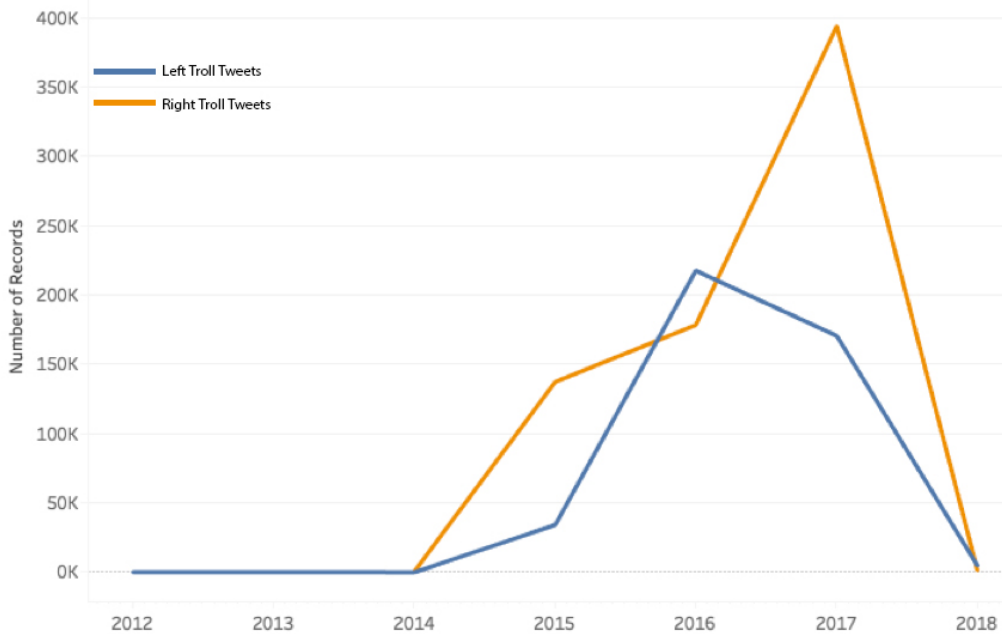
Figure 5.1 Temporal Analysis on Baruch dataset based on left and right troll tweets.

In the Baruch Dataset with 5 million tweets, only 133,801 tweets had geolocation data attached. We performed a geospatial analysis on this data to locate the left troll and right troll tweets. Table 5.4 shows the quantity of left and right troll tweets based on the geolocation data.

Table 5.4 Count of Tweets Containing Geo Location Data from 5 Million Tweets Dataset

Tweet type (from 5 Million)	Count
Total tweets with geolocation	133801
Total Troll Tweets with geolocation	83232
Right Trolls with geolocation	50154
Left Troll with geolocation	33078

Figures 5.2-5.4 show the geospatial distribution of the Left and Right troll tweets. The total ratio of right to left troll tweets is 60:40. In Japan, South Korea and Thailand, where the

right troll tweets are more prominent. In the UK, the left troll tweets are significantly higher as compared to the right troll tweets. In the United States, the amount of left/right troll tweets is at a ratio of 60:40. We also observe that the right troll tweets and left troll tweets are evenly distributed in most of the countries. Because Figure 5.2 contains overlap of tweet locations, the following Figures 5.3 and 5.4 show left and right tweets separately.
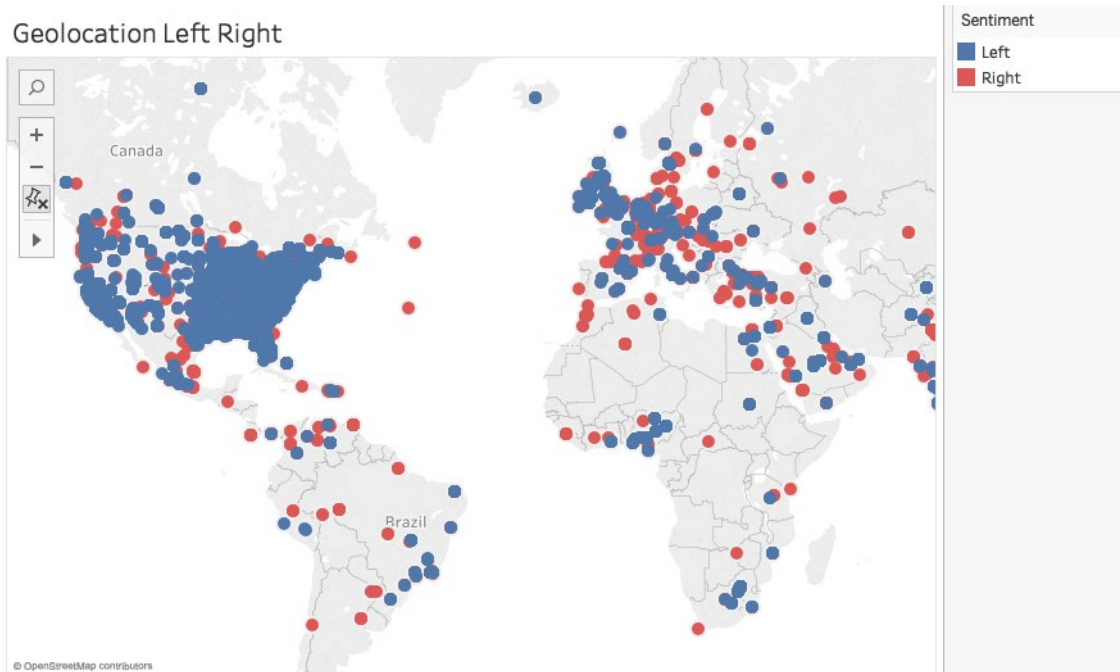
Figure 5.2 Geospatial distribution of Left/Right troll tweets (with overlaps).

Figure 5.3 Geospatial distribution of Left troll tweets.

Figure 5.4 Geospatial distribution of Right troll tweets.

5.2 Sentiment analysis on Baruch Dataset with five Million Tweets

Sentiment analysis was done on Left and Right troll tweets to understand the emotional tone of the trolls. Since trolls are using social media, we try to understand what kind of tone trolls are using to influence people's minds. The sentiment analysis model was built using the Sentiment140 dataset (https://www.kaggle.com/kazanova/sentiment140).

Figure 5.5 illustrates the sentiments, which have been classified into five different classes, namely neutral, positive, extremely positive, negative and extremely negative, based on the data.

We are trying to understand the sentiment based on left and right troll data by counting of tweets that are positive, negative or neutral. The count of left and right trolls is the same to see the pattern.

Sentiment Analysis was done on the right troll tweets which we had classified above on the news dataset to classify it into different sentiments as shown in Figure 5.6.

Sentiment_ri..	
Neutral	266,128
Positive	77,355
Negative	48,630
Extremely Positive	20,997
Extremely Negative	14,031

Figure 5.5 Count of tweets for sentiment analysis on right troll tweets.
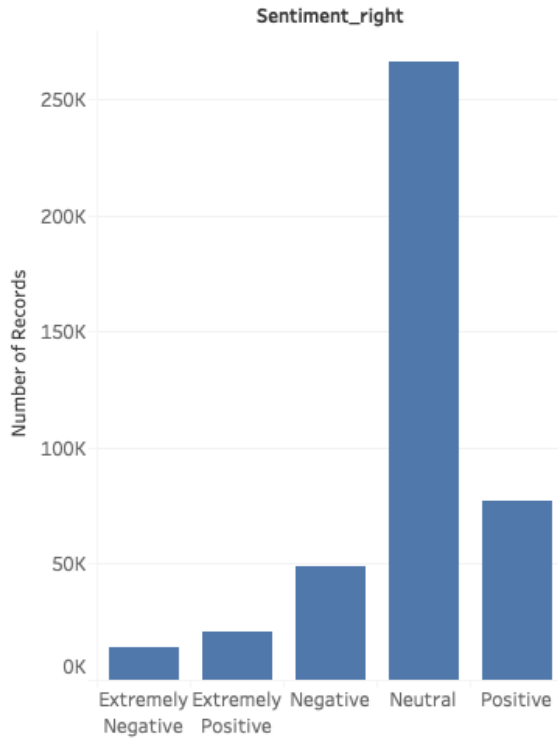
Figure 5.6 Sentiment analysis bar chart for right troll tweets.

Sentiment Analysis was done on the left troll tweets which we had classified above on the news dataset to classify it into different sentiments as shown in Figure 5.7

Sentiment_left	
Neutral	260,832
Positive	87,889
Negative	46,632
Extremely Positive	23,188
Extremely Negative	8,600

Figure 5.7 Count of tweets for sentiment analysis on left troll tweets.
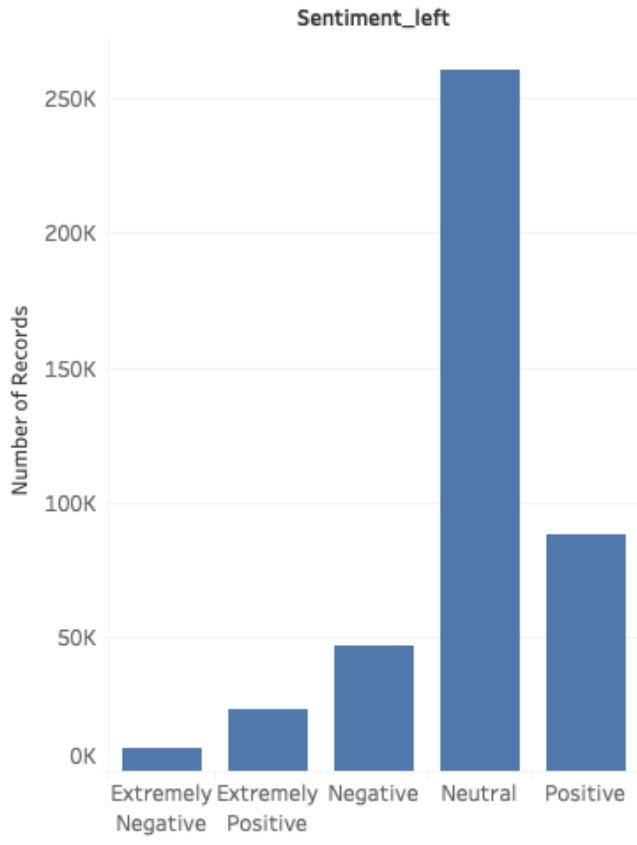
Figure 5.8 Sentiment analysis Bar Chart for Right Troll tweets.
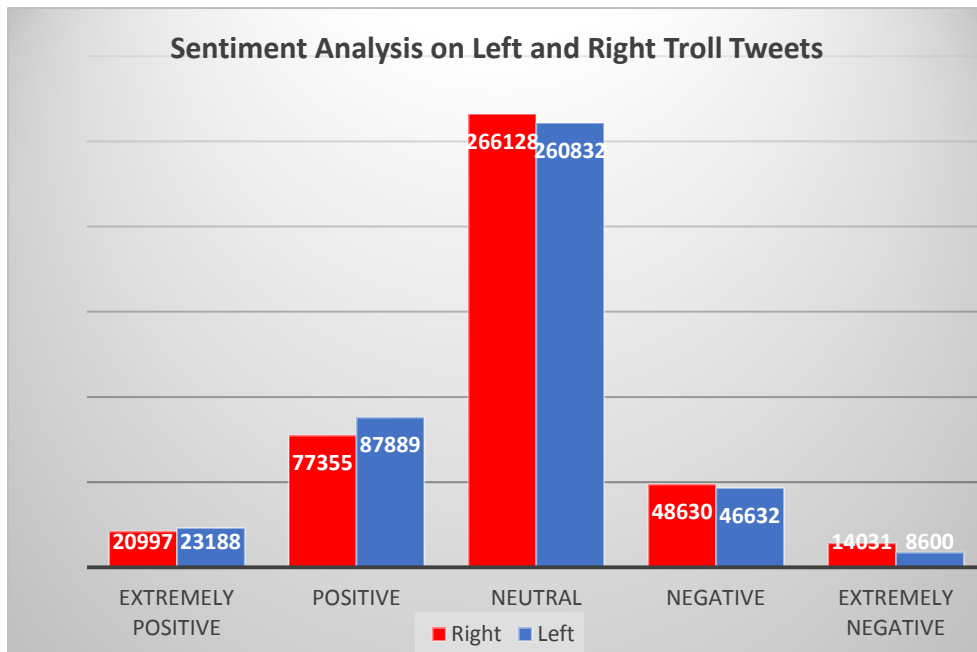
Figure 5.9 Sentiment Analysis on Left and Right Troll tweets.

Observation that we made when performing the sentiment analysis are as follows:

(i)   The ratio of positive, negative and neutral tweets does not vary much in either left or right troll tweets.

(ii) The number of negative tweets is slightly higher in right troll tweets by a count of around 7,000 tweets.

(iii) The number of positive tweets is slightly lower in right troll tweets by a count of 13,000 tweets.

CHAPTER 6

CONCLUSIONS AND FUTURE WORKS

In this thesis, we build several machine learning models to study the Russian trolls and their interference in the US Presidential elections 2016. BERT had an accuracy of 89.4% for Left/Right Troll Detector and 99% for Troll/Non-troll Detector which is higher than SVM, CNN with Word2vec and Neural Network with One-hot encoding. In all of the Baruch Datasets, the Right Troll tweets are slightly more as compared to the left troll tweets. Sentiment analysis on the Baruch tweets shows us that the positive Right troll tweets are slightly less as compared to the left troll tweets.

The main focus of our thesis was on building and comparing the models and using it for studying the geospatial, temporal and sentiment analysis. Many different analysis and experiments have been left for the future due to lack of time (creating models takes a lot of time, it can take several days to a couple of weeks).

In the future, real-time analysis of tweets can be done, that is an engine can be built to analyze tweets real-time. We need to make use of several frameworks since we needed a database to fetch the tweets real-time and then stored into a database. Then, model data will be fed to the model to process the data and classify on the data. And then the data along with the classification is sent to the database. And this database is used to show the data on a website real-time.

APPENDIX 1

CODE A1 TROLL DETECTOR USING SVM

```
/* training.py */

import csv
import os
import re
import nltk
import scipy
import sklearn.metrics
import predict
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn import svm
from sklearn.externals import joblib
from sklearn.pipeline import Pipeline
from sklearn.cross_validation import train_test_split

#Generating the Training and testing vectors
def getTrainingAndTestData():
        X = []
        y = []

        f=open('merged_3mtweets.csv','r', encoding='ISO-8859-1')
        reader = csv.reader(f)

        for row in reader:
            X.append(row[2])
            y.append(1 if (row[1]=='1') else 0)


        X_train,        X_test,        y_train,        y_test        =
sklearn.cross_validation.train_test_split(X,y,test_size=0.20,
random_state=42)
        return X_train, X_test, y_train, y_test

#Process Tweets (Stemming+Pre-processing)

def processTweets(X_train, X_test):
        X_train = [sentiment.stem(sentiment.preprocessTweets(tweet)) for
tweet in X_train]
        X_test = [sentiment.stem(sentiment.preprocessTweets(tweet)) for
tweet in X_test]
        return X_train,X_test

# SVM classifier

def classifier(X_train,y_train):
        vec  =  TfidfVectorizer(min_df=5,  max_df=0.95,  sublinear_tf  =
True,use_idf = True,ngram_range=(1, 2))
        svm_clf =svm.LinearSVC(C=0.1)
        vec_clf = Pipeline([('vectorizer', vec), ('pac', svm_clf)])
```

```
        vec_clf.fit(X_train,y_train)
        joblib.dump(vec_clf, 'svmClassifier.pkl', compress=3)
        return vec_clf

# Main function

def main():
        X_train, X_test, y_train, y_test = getTrainingAndTestData()
        X_train, X_test = processTweets(X_train, X_test)
        vec_clf = classifier(X_train,y_train)
        y_pred = vec_clf.predict(X_test)
        print(sklearn.metrics.classification_report(y_test, y_pred))

if __name__ == "__main__":
    main()


/* predict.py */


import sys
import time
import re
import nltk
import pandas as pd
from sklearn.externals import joblib

#Processing Tweets
df2 = pd.read_csv("5million_text_wID_str.csv")
print(df2['text'].head(5))
def preprocessTweets(tweet):

    #Convert www.* or https?://* to URL
    tweet = re.sub("[^a-zA-Z]",   # Search for all non-letters
                        " ",                # Replace all non-letters with
spaces
                        str(tweet))
    tweet = re.sub('((www\.[^\s]+)|(https?://[^\s]+))','URL',tweet)

    #Convert @username to __HANDLE
    tweet = re.sub('@[^\s]+','__HANDLE',tweet)

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

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

    # Repeating words like happyyyyyyyy
    rpt_regex = re.compile(r"(.)\1{1,}", re.IGNORECASE)
    tweet = rpt_regex.sub(r"\1\1", tweet)

    #Emoticons
    emoticons = \
    [
      ('__positive__',[ ':-)', ':)', '(:', '(-:', \
```

```python
                        ':-D', ':D', 'X-D', 'XD', 'xD', \
                        '<3', ':\*', ';-)', ';)', ';-D', ';D', '(;', '(-
;', ] ),\
      ('__negative__', [':-(', ':(', '(:', '(-:', ':,(,\
                        ':\'(', ':"(', ':((', ] ),\
    ]

    def replace_parenth(arr):
       return  [text.replace(')', '[)}\]]').replace('(', '[({\[]')  for
text in arr]

    def regex_join(arr):
        return '(' + '|'.join( arr ) + ')'

    emoticons_regex            =            [            (repl,
re.compile(regex_join(replace_parenth(regx))) ) \
            for (repl, regx) in emoticons ]

    for (repl, regx) in emoticons_regex :
        tweet = re.sub(regx, ' '+repl+' ', tweet)

     #Convert to lower case
    tweet = tweet.lower()

    return tweet

#Stemming of Tweets

def stem(tweet):
        stemmer = nltk.stem.PorterStemmer()
        tweet_stem = ''
        words = [word if(word[0:2]=='__') else word.lower() \
                    for word in tweet.split() \
                    if len(word) >= 3]
        words = [stemmer.stem(w) for w in words]
        tweet_stem = ' '.join(words)
        return tweet_stem


#Predict the sentiment

def predict(tweet,classifier):

    tweet_processed = stem(preprocessTweets(tweet))

    if ( ('__positive__') in (tweet_processed)):
        sentiment  = 1
         return sentiment

    elif ( ('__negative__') in (tweet_processed)):
        sentiment  = 0
         return sentiment
    else:

        X =  [tweet_processed]
        sentiment = classifier.predict(X)
        return (sentiment[0])
```

```
print('Loading the Classifier, please wait....')
classifier = joblib.load('svmClassifier.pkl')

# Main function
def fx(x):
    a = predict(x, classifier)
    return a
def main():
    # print('Loading the Classifier, please wait....')
    print('Running the classifier')
    df2['prediction'] = df2['text'].apply(fx)
    df2.to_csv("prediction_5m_wID.csv")
    # print(df)
    # for tweet in sys.stdin:
    #           print(predict(tweet, classifier))




if __name__ == "__main__":
    main()
```

CODE A2 Troll Detector using Neural Network with one-hot encoding

```
/* training.py */


import numpy as np
import json
import keras
import keras.preprocessing.text as kpt
from keras.preprocessing.text import Tokenizer
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation


# extract data from a csv
# notice the cool options to skip lines at the beginning
# and to only take data from certain columns
training = np.genfromtxt('merged3.csv', delimiter=',', skip_header=1,
      usecols=(1, 2), dtype=None, invalid_raise = False, encoding=
      'utf8')

# create our training data from the tweets
train_x = [x[1] for x in training]
# index all the sentiment labels
train_y = ([x[0] for x in training])
# only work with the 3000 most popular words found in our dataset
max_words = 3000
# create a new Tokenizer
tokenizer = Tokenizer(num_words=max_words)
# feed our tweets to the Tokenizer
```

```python
tokenizer.fit_on_texts(train_x)

# Tokenizers come with a convenient list of words and IDs
dictionary = tokenizer.word_index
# Let's save this out so we can use it later
with open('dictionary.json', 'w') as dictionary_file:
    json.dump(dictionary, dictionary_file)


def convert_text_to_index_array(text):
    # one really important thing that `text_to_word_sequence` does
    # is make all texts the same length -- in this case, the length
    # of the longest text in the set.
    return [dictionary[word] for word in
      kpt.text_to_word_sequence(text)]

allWordIndices = []
# for each tweet, change each token to its ID in the Tokenizer's
      word_index
for text in train_x:
    wordIndices = convert_text_to_index_array(text)
    allWordIndices.append(wordIndices)

# now we have a list of all tweets converted to index arrays.
# cast as an array for future usage.
allWordIndices = np.asarray(allWordIndices)

# create one-hot matrices out of the indexed tweets
train_x = tokenizer.sequences_to_matrix(allWordIndices, mode='binary')
# treat the labels as categories
train_y = keras.utils.to_categorical(train_y, 2)


model = Sequential()
model.add(Dense(512, input_shape=(max_words,), activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(256, activation='sigmoid'))
model.add(Dropout(0.5))
model.add(Dense(2, activation='softmax'))

model.compile(loss='categorical_crossentropy',
  optimizer='adam',
  metrics=['accuracy'])


model.fit(train_x, train_y,
  batch_size=32,
  epochs=5,
  verbose=1,
  validation_split=0.1,
  shuffle=True)

model_json = model.to_json()
with open('model.json', 'w') as json_file:
    json_file.write(model_json)

model.save_weights('model.h5')
```

```
/* predict.py */


import json
import numpy as np
import keras
import keras.preprocessing.text as kpt
from keras.preprocessing.text import Tokenizer
from keras.models import model_from_json

# we're still going to use a Tokenizer here, but we don't need to fit
        it
tokenizer = Tokenizer(num_words=3000)
# for human-friendly printing
labels = ['Lefttroll', 'Righttroll']

# read in our saved dictionary
with open('dictionary.json', 'r') as dictionary_file:
    dictionary = json.load(dictionary_file)

# this utility makes sure that all the words in your input
# are registered in the dictionary
# before trying to turn them into a matrix.
def convert_text_to_index_array(text):
    words = kpt.text_to_word_sequence(text)
    wordIndices = []
    for word in words:
        if word in dictionary:
            wordIndices.append(dictionary[word])
        else:
            print("'%s' not in training corpus; ignoring." %(word))
    return wordIndices


# read in your saved model structure
json_file = open('model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
# and create a model from that
model = model_from_json(loaded_model_json)
# and weight your nodes with your saved values
model.load_weights('model.h5')

input_data = np.genfromtxt('prediction_5m_troll.csv', delimiter=',',
       skip_header=1, usecols=(2, 3), dtype=None, invalid_raise = False)

text = [x[0] for x in input_data]
```

```
my_data = [None] * len(text)
# my_data[0] = 'new_rating'
print(len(text))
for i in range(0,len(text)):
    testArr = convert_text_to_index_array(text[i])
    input = tokenizer.sequences_to_matrix([testArr], mode='binary')
    pred = model.predict(input)
    if(labels[np.argmax(pred)]=='Righttroll'):
        my_data[i] = 1
    elif (labels[np.argmax(pred)]=='Lefttroll'):
        my_data[i] = 0


# print(type(my_data))

df3 = pd.read_csv('prediction_5m_troll.csv')

df4['predict_troll_leftorright'] = my_data

print(df3.head())
df3.to_csv('leftorright.csv', sep=',')
```

CODE A3 Troll Detector using CNN with Word2Vec Representation

```
import numpy as np
import pandas as pd

from keras.layers import Dense, Input, GlobalMaxPooling1D
from keras.layers import Conv1D, MaxPooling1D, Embedding
from keras.models import Model
from keras.layers import Input, Dense, Embedding, Conv2D, MaxPooling2D,
Dropout,concatenate
from keras.layers.core import Reshape, Flatten
from keras.callbacks import EarlyStopping
from keras.optimizers import Adam
from keras.models import Model
from keras import regularizers
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical


train_data=pd.read_csv('/home/k/kk534/troll_detector_word2vec/train.csv
')
test_data=pd.read_csv('/home/k/kk534/troll_detector_word2vec/test.csv')
authors=train_data.author.unique()
dic={}
for i,author in enumerate(authors):
    dic[author]=i
labels=train_data.author.apply(lambda x:dic[x])
```

```python
val_data=train_data.sample(frac=0.2,random_state=200)
train_data=train_data.drop(val_data.index)


texts=train_data.text


NUM_WORDS=20000
tokenizer = Tokenizer(num_words=NUM_WORDS,filters='!"#$%&()*+,-./:;<=>?
@[\\]^_`{|}~\t\n\'',
                      lower=True)
tokenizer.fit_on_texts(texts)
sequences_train = tokenizer.texts_to_sequences(texts)
sequences_valid=tokenizer.texts_to_sequences(val_data.text)
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))


X_train = pad_sequences(sequences_train)
X_val = pad_sequences(sequences_valid,maxlen=X_train.shape[1])
y_train = to_categorical(np.asarray(labels[train_data.index]))
y_val = to_categorical(np.asarray(labels[val_data.index]))
print('Shape of X train and X validation tensor:', X_train.shape,X_val.
shape)
print('Shape of label train and validation tensor:', y_train.shape,y_va
l.shape)


import gensim
from gensim.models import Word2Vec
from gensim.utils import simple_preprocess

from gensim.models.keyedvectors import KeyedVectors

word_vectors = KeyedVectors.load_word2vec_format('../GoogleNews-vectors
-negative300.bin', binary=True)

EMBEDDING_DIM=300
vocabulary_size=min(len(word_index)+1,NUM_WORDS)
embedding_matrix = np.zeros((vocabulary_size, EMBEDDING_DIM))
for word, i in word_index.items():
    if i>=NUM_WORDS:
        continue
    try:
        embedding_vector = word_vectors[word]
        embedding_matrix[i] = embedding_vector
    except KeyError:
        embedding_matrix[i]=np.random.normal(0,np.sqrt(0.25),EMBEDDING_
DIM)

del(word_vectors)

from keras.layers import Embedding
embedding_layer = Embedding(vocabulary_size,
                            EMBEDDING_DIM,
                            weights=[embedding_matrix],
                            trainable=True)
```

```
sequence_length = X_train.shape[1]
filter_sizes = [3,4,5]
num_filters = 100
drop = 0.5



inputs = Input(shape=(sequence_length,))
embedding = embedding_layer(inputs)
reshape = Reshape((sequence_length,EMBEDDING_DIM,1))(embedding)

conv_0 = Conv2D(num_filters, (filter_sizes[0], EMBEDDING_DIM),activatio
n='relu',kernel_regularizer=regularizers.l2(0.01))(reshape)
conv_1 = Conv2D(num_filters, (filter_sizes[1], EMBEDDING_DIM),activatio
n='relu',kernel_regularizer=regularizers.l2(0.01))(reshape)
conv_2 = Conv2D(num_filters, (filter_sizes[2], EMBEDDING_DIM),activatio
n='relu',kernel_regularizer=regularizers.l2(0.01))(reshape)

maxpool_0 = MaxPooling2D((sequence_length - filter_sizes[0] + 1, 1), st
rides=(1,1))(conv_0)
maxpool_1 = MaxPooling2D((sequence_length - filter_sizes[1] + 1, 1), st
rides=(1,1))(conv_1)
maxpool_2 = MaxPooling2D((sequence_length - filter_sizes[2] + 1, 1), st
rides=(1,1))(conv_2)

merged_tensor = concatenate([maxpool_0, maxpool_1, maxpool_2], axis=1)
flatten = Flatten()(merged_tensor)
reshape = Reshape((3*num_filters,))(flatten)
dropout = Dropout(drop)(flatten)
output = Dense(units=3, activation='softmax',kernel_regularizer=regular
izers.l2(0.01))(dropout)

# this creates a model that includes
model = Model(inputs, output)

adam = Adam(lr=1e-3)

model.compile(loss='categorical_crossentropy',
              optimizer=adam,
              metrics=['acc'])
callbacks = [EarlyStopping(monitor='val_loss')]
model.fit(X_train, y_train, batch_size=1000, epochs=10, verbose=1, vali
dation_data=(X_val, y_val),
        callbacks=callbacks)  # starts training

sequences_test=tokenizer.texts_to_sequences(test_data.text)
X_test = pad_sequences(sequences_test,maxlen=X_train.shape[1])
y_pred=model.predict(X_test)
```

CODE A4 Troll Detector using CNN with State-of-the-art NLP BERT

```
/* The BERT Code is extremely huge. Only snippets of the code
are shared here*/


class BertConfig(object):
  """Configuration for `BertModel`."""
...
def __init__(self, vocab_size,hidden_size=768,num_hidden_layers=12,
             num_attention_heads=12,intermediate_size=3072,
             hidden_act="gelu",hidden_dropout_prob=0.1,
             attention_probs_dropout_prob=0.1,
             max_position_embeddings=512,type_vocab_size=16,
             initializer_range=0.02):
```

REFERENCES

Luis Gerardo Mojica (2017). A Trolling Hierarchy in Social Media and a Conditional Random Field for Trolling Detection, *arXiv:1704.02385v1 [cs.CL] accessed on 7 Apr 2017*

Vladimir Vapnik (1995). Support-Vector Networks, *Machine Learning, 20, 273-297, accessed on (1995)*

Yoon Kim (2014). Convolutional Neural Networks for Sentence Classification, *arXiv:1408.5882v2 [cs.CL], accessed on 3 Sep 2014*

Yann LeCun (1995). Convolutional Neural Networks for Images, Speech, and Time Series, *arXiv:1408.5882v2 [cs.CL], accessed on 3 Sep 2014*

Chih-Chung Chang and Chih-Jen Lin (2001). LIBSVM: A Library for Support Vector Machines, *DOI:10.1145/1961189.1961199, accessed on 3 Mar 2001*