SILIGURI INSTITUTE OF TECHNOLOGY

CS 892

FAKE NEWS DETECTION

BY

CSE_PROJ_2020_05

Name of Students	Roll No.
1. Bishal Das	11900116069
2. Arup Jyoti Dutta	11900116074
3. Jyotismita Chanda	11900116053
4. Arnab Sinha	11900116076

Under the Guidance

of

Prof. Debaditya Kundu

Submitted to the Department of **Computer Science & Engineering** in partial fulfillment of the requirements for the award of the degree Bachelor of Technology in **Computer Science & Engineering.**

Year of Submission: 2020

Siliguri Institute of Technology

P.O. SUKNA, SILIGURI, DIST. DARJEELING, PIN:734009

Tel: (0353)2778002/04, Fax: (0353) 2778003

DECLARATION

This is to certify that Report entitled "**Fake News Detection**" which is submitted by us in partial fulfillment of the requirement for the award of degree B.Tech. in **Computer Science Engineering** at **Siliguri Institute of Technology** under **Maulana Abul Kalam Azad University of Technology**, West Bengal. We took the help of other materials in our dissertation which have been properly acknowledged. This report has not been submitted to any other Institute for the award of any other degree.

Date:

SN	Name of the Student	Roll No	Signature
1	Bishal Das	11900116069	
2	Arup Jyoti Dutta	11900116074	
3	Jyotismita Chanda	11900116053	
4	Arnab Sinha	11900116076	

CERTIFICATE

This is to certify that the project report entitled **"Fake News Detection"** submitted to **Department of Computer Science & Engineering of Siliguri Institute of Technology** in partial fulfilment of the requirement for the award of the degree of **Bachelor of Technology in Computer Science & Engineering** during the academic year **2019-20,** is a bonafide record of the project work carried out by them under my guidance and supervision.

Project Group Number : 05			
SN	**Name of the students**	**Registration No**	**Roll No**
1.	Bishal Das	161190110026	11900116069
2.	Arup Jyoti Dutta	161190110021	11900116074
3.	Jyotismita Chanda	161190110042	11900116053
4.	Arnab Sinha	161190110019	11900116076

---------- **----------**

Signature of Project Guide
Name of the Guide:

Signature of the HOD
Department of Computer Science & Engineering

Acknowledgement

Many thanks to **Prof. Debaditya Kundu** for guiding us throughout the course of this project with his immense knowledge.

Signature of all the group members with date

1.

2.

3.

4.

Contents

List of Tables

List of Figures

Abstract

In this project, we explore the application of Natural Language Processing techniques to identify when a news source may be producing fake news. We use a corpus of labeled real and fake new articles to build a classifier that can make decisions about information based on the content from the corpus. We use a text classification approach,using four different classification models,and analyze the results. The best performing model was the LSTM implementation.

The model focuses on identifying fake news sources, based on multiple articles originating from source.Once a source is labeled as a producer of fake news, we can predict with high confidence that any future articles from that source will also be fake news. Focusing on sources widens our article misclassification tolerance, because we then have multiple data points coming from each source.

CHAPTER 1

Introduction

1.1 Motivation

The rise of fake news during the 2016 U.S. Presidential Election highlighted not only the dangers of the effects of fake news but also the challenges presented when attempting to separate fake news from real news. Fake news may be a relatively new term but it is not necessarily a new phenomenon. Fake news has technically been around at least since the appearance and popularity of one-sided, partisan newspapers in the 19th century. However, advances in technology and the spread of news through different types of media have increased the spread of fake news today. As such, the effects of fake news have increased exponentially in the recent past and something must be done to prevent this from continuing in the future.

I have identified the three most prevalent motivations for writing fake news and chosen only one as the target for this project as a means to narrow the search in a meaningful way. The first motivation for writing fake news, which dates back to the 19th century one-sided party newspapers, is to influence public opinion. The second, which requires more recent advances in technology, is the use of fake head- lines as clickbait to raise money. The third motivation for writing fake news, which is equally prominent yet arguably less dangerous, is satirical writing. [1] [2] While all three subsets of fake news, namely, (1) clickbait, (2), influential, and (3) satire, share the common thread of being fictitious, their widespread effects are vastly different. As such, this paper will focus primarily on fake news as defined by politifact.com, “fabricated content that intentionally masquerades as news coverage of actual events.” This definition excludes satire, which is intended to be humorous and not deceptive to readers. Most satirical articles come from sources like “The Onion“, which specifically distinguish themselves as satire. Satire can already be classified, by machine learning techniques according to [3]. Therefore, our goal is to move beyond these achievements and use machine learning to classify, at least as well as humans, more difficult discrepancies between real and fake news.

The dangerous effects of fake news, as previously defined, are made clear by events such as [4] in which a man attacked a pizzeria due to a widespread fake news article. This story along with analysis from [5] provide evidence that humans are not very good at detecting fake news, possibly not better than chance . As such, the question remains whether or not machines can do a better job.

There are two methods by which machines could attempt to solve the fake news problem better than humans. The first is that machines are better at detecting and keeping track of statistics than humans, for example it is easier for a machine to detect that the majority of verbs used are “suggests” and “implies” versus, “states” and “proves.” Additionally, machines may be more efficient in surveying a knowledge base to find all relevant articles and answering based on those many different sources. Either of these methods could prove useful in detecting fake news, but we decided to focus on how a machine can solve the fake news problem using

supervised learning that extracts features of the language and content only within the source in question, without utilizing any fact checker or knowledge base. For many fake news detection techniques, a "fake" article published by a trustworthy author through a trustworthy source would not be caught. This approach would combat those "false negative" classifications of fake news. In essence, the task would be equivalent to what a human faces when reading a hard copy of a newspaper article, without internet access or outside knowledge of the subject (versus reading something online where he can simply look up relevant sources). The machine, like the human in the coffee shop, will have only access to the words in the article and must use strategies that do not rely on blacklists of authors and sources.

The current project involves utilizing machine learning and natural language processing techniques to create a model that can expose documents that are, with high probability, fake news articles. Many of the current automated approaches to this problem are centered around a "blacklist" of authors and sources that are known producers of fake news. But, what about when the author is unknown or when fake news is published through a generally reliable source? In these cases it is necessary to rely simply on the content of the news article to make a decision on whether or not it is fake. By collecting examples of both real and fake news and training a model, it should be possible to classify fake news articles with a certain degree of accuracy. The goal of this project is to find the effectiveness and limitations of language-based techniques for detection of fake news through the use of machine learning algorithms including but not limited to convolutional neural networks and recurrent neural networks. The outcome of this project should determine how much can be achieved in this task by analyzing patterns contained in the text and blind to outside information about the world.

This type of solution is not intended to be an end-to end solution for fake news classification. Like the "blacklist" approaches mentioned, there are cases in which it fails and some for which it succeeds. Instead of being an end-to-end solution, this project is intended to be one tool that could be used to aid humans who are trying to classify fake news. Alternatively, it could be one tool used in future applications that intelligently combine multiple tools to create an end-to-end solution to automating the process of fake news classification.

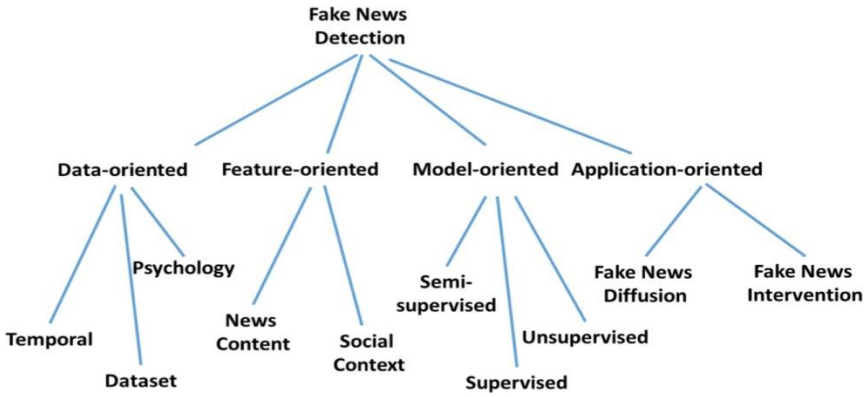


Figure 1 : Different approaches to fake news detection.

1.2 Challenges

Challenges of detecting fake news springs from the fact that it is even difficult for human beings to separate fake news from true news. Concretely,

- Language use is complex in fake news. Rashkin analyzes the difference between language use of fake news and that of true news on three kinds of fake news: satire, hoax, and propaganda. Her work reveals that a wide range of linguistic factors contribute to the formation of fake news such as subjective, intensifying, and hedging words with the intent to introduce vague, obscuring, dramatizing or sensationalizing language. Therefore, applying feature-based approaches will be labor-intensive and time-consuming.
- Fake news usually mixes true stories with false details, which are confusing to be recognized correctly. It is often that fake news maker blend true story with false details to mislead people. For example, the statement "*However, it took $19.5 million in Oregon Lottery funds for the Port of Newport to eventually land the new NOAA Marine Operations Center-Pacific*" is half-true since it combines the true number of $19.5 million and the misleading place where the money went to. In such case, it is easy to get people's attention about trusted parts without noticing the presence of fabricated ones.
- Fake news data is limited. Currently, there is only political fake news dataset published. Domains other than politics are still open to future research.

1.3 Problem Definition :

Given the news article A, the task of fake news detection is to predict whether the news article A is a fake news piece or not, i.e.,
F : A -> {0,1}
such that,
F(A) = 1, if A is a piece of fake news
0, otherwise.

where F is the prediction function we want to learn.

1.4 Methodology Used :

In order to detect whether the news is real or fake we have followed the following steps :

a. **Data Preparation** includes all the preprocessing functions needed to process all input documents and texts. First we read the dataset and then performed some preprocessing tasks like tokenizing, stemming etc. Then we perform some exploratory data analysis like data quality checks to check if our dataset contains null or missing values.

b. **Feature Extraction** contains feature extraction and selection methods. For feature extraction we have used Doc2vec . The reason behind choosing Doc2vec is that it comes with an additional benefit of word2vec following which it helps inbetter comparison. After performing feature extraction we split our dataset into two parts for training and testing purposes.

c. **Classification** is model building stage. Here we built all classifiers for predicting the fake news detection. The extracted features are fed into different classifiers. The classifiers we are going to use are Logistic Regression, Naïve-Bayes, Support Vector Machine (SVM), FeedForward Neural Network and Long-Short Term Memory (LSTM). After fitting the model, we compared the f1 score and then checked the confusion matrix, After fitting all the classifiers, the best performing model is selected as the candidate model for fake news classification.

d. **Prediction** phase is about final selection of the best performing classifier, which we saved on the disk. Then we used our prediction function to classify the fake news. It takes a news article as input from the user which we fed to our candidate classifier for final classification output, that is to show to user the probability of truth.

CHAPTER 2

Related Work

2.1 Spam Detection

The problem of detecting not-genuine sources of information through content based analysis is considered solvable at least in the domain of spam detection [6], spam detection utilizes statistical machine learning techniques to classify text (i.e. tweets [7] or emails) as spam or legitimate. These techniques involve pre-processing of the text, feature extraction (i.e. bag of words), and feature selection based on which features lead to the best performance on a test dataset. Once these features are obtained, they can be classified using Nave Bayes, Support Vector Machines, TF-IDF, or K-nearest neighbors classifiers. All of these classifiers are characteristic of supervised machine learning, meaning that they require some labeled data in order to learn the function (as seen in [8]).

$$F(m, \Theta) = \{ C_{spam}, \text{ if classified as spam} \\ C_{leg}, \text{ otherwise} \}$$

where, m is the message to be classified and is a vector of parameters and C_{spam} and C_{leg} are respectively spam and legitimate messages. The task of detecting fake news is similar and almost analogous to the task of spam detection in that both aim to separate examples of legitimate text from examples of illegitimate, ill-intended texts. The question, then, is how can we apply similar techniques to fake news detection. Instead of filtering like we do with spam, it would be beneficial to be able to flag fake news articles so that readers can be warned that what they are reading is likely to be fake news. The purpose of this project is not to decide for the reader whether or not the document is fake, but rather to alert them that they need to use extra scrutiny for some documents. Fake news detection, unlike spam detection, has many nuances that arent as easily detected by text analysis. For example, a human actually needs to apply their knowledge of a particular subject in order to decide whether or not the news is true. The “fakeness” of an article could be switched on or off simply by replacing one persons name with another persons name. Therefore, the best we can do from a content-based standpoint is to decide if it is something that requires scrutiny. The idea would be for a reader to do leg work of researching other articles on the topic to decide whether or not the article is actually fake, but a “flagging” would alert them to do so in appropriate circumstances.

2.2 Stance Detection

In December of 2016, a group of volunteers from industry and academia started a contest called the Fake News Challenge [9]. The goal of this contest was to encourage the development of tools that may help human fact checkers identify deliberate misinformation in news stories through the use of machine learning, natural language processing and artificial intelligence. The organizers decided that the first step in this overarching goal was understanding what other news organizations are saying about the topic in question. As such, they decided that stage one of their contest would be a stance detection competition.

CHAPTER 3

Data Exploration

3.1 Introduction

The lack of manually labeled fake news datasets is certainly a bottleneck for advancing computationally intensive, text-based models that cover a wide array of topics. The dataset for the fake news challenge does not suit our purpose due to the fact that it contains the ground truth regarding the relationships between texts but not whether or not those texts are actually true or false statements. For more simple and common NLP classification tasks, such as sentiment analysis, there is an abundance of labeled data from a variety of sources including Twitter, Amazon Reviews, and IMDb Reviews. Unfortunately, the same is not true for finding labeled articles of fake and real news. This presents a challenge to researchers and data scientists who want to explore the topic by implementing supervised machine learning techniques. I have researched the available datasets for sentence-level classification.

3.2 Dataset

The datasets used for this project were drawn from Kaggle[10]. The training dataset has about 16600 rows of data from various articles on the internet. We had to do quite a bit of pre-processing of the data, as is evident from our source code [11], in order to train our models.

A full training dataset has the following attributes:

1. id: unique id for a news article
2. title: the title of a news article
3. author: author of the news article
4. text: the text of the article; incomplete in some cases
5. label: a label that marks the article as potentially unreliable
    - 1: unreliable
    - 0: reliable

A snapshot of the training data is given in *Table 1*

ID	Title	Author	Text	Label
0	House Dem Aide: We Didn't Even See Comey's Letter Until Jason Chaffetz Tweeted It	Darrell Lucus	House Dem Aide: We Didn't Even See Comey's Letter Until Jason Chaffetz Tweeted It By Darrell Lucus o...	0
1	FLYNN: Hillary Clinton, Big Woman on Campus - Breitbart	Daniel J. Flynn	Ever get the feeling your life circles the roundabout rather than heads in a straight line toward th...	1
2	Why the Truth Might Get You Fired	Consortiumnews.com	Why the Truth Might Get You Fired October 29, 2016 The tension between intelligence analysts and po...	1

Table 1

3.3 General Analysis

Here we will see some analysis about our kaggle dataset to make fit for classification. Before starting the analysis, it is needed to clean up the dataset. As it is originally given in a large 94MB CSV file, the first step is to put everything in a database in order to be able to retrieve only wanted a piece of information. In order to do so, the file has been read line by line. It appears that some of the lines are badly formated, preventing them from being read correctly, in this case they are dropped without being put in the database. Also, each line that is a duplicate of a line already read is also dropped. In the next step we perform some basic pre-processing of the data. This includes removing stop words, deleting special characters and punctuation, and converting all text to lowercase. This produces a comma-separated list of words, which can be input into the Doc2Vec algorithm to produce an 300-length embedding vector for each article. All these methods are explained in the later chapter.

CHAPTER 4

Literature Review

4.1 Introduction

In this chapter, we will focus on the more traditional methods used in natural language processing such as Naïve-Bayes, Linear Regression, SVM(Support Vector Machine), FeedForward Neural Network and LSTM(Long-Short Term Memory). The first thing to do when working with text is the do words and texts embedding, indeed, in order to use machine learning algorithms on texts, a mathematical representation of these texts is required. The programming part of all the machine learning models have been provided in the appendix section.

4.2 Text to Vectors (using Doc2vec[12])

Text needs to be represented in a way that gives more meaningful information than a simple sequence of bits, which have additional drawbacks such that for a given word, the sequence of bits representing it depends on the coding. Text classification and clustering play an important role in many applications, e.g, document retrieval, web search, spam filtering. At the heart of these applications is machine learning algorithms such as logistic regression or Kmeans. These algorithms typically require the text input to be represented as a fixed-length vector. Perhaps the most common fixed-length vector representation for texts is the bag-of-words or bag-of-n-grams[13] due to its simplicity, efficiency and often surprising accuracy.

However, the bag-of-words (BOW) has many disadvantages. The word order is lost, and thus different sentences can have exactly the same representation, as long as the same words are used. Even though bag-of-n-grams considers the word order in short context, it suffers from data sparsity and high dimensionality. Bag-of-words and bag-of-n-grams have very little sense about the semantics of the words or more formally the distances between the words. This means that words “powerful,” “strong” and “Paris” are equally distant despite the fact that semantically, “powerful” should be closer to “strong” than “Paris.”

In order to deal with this problem we come up with Paragraph Vector, an unsupervised framework that learns continuous distributed vector representations for pieces of texts. The texts can be of variable-length, ranging from sentences to documents. The name Paragraph Vector is to emphasize the fact that the method can be applied to variable-length pieces of texts, anything from a phrase or sentence to a large document. Paragraph Vector is capable of constructing representations of input sequences of variable length. Unlike some of the previous approaches, it is general and applicable to texts of any length: sentences, paragraphs, and documents. It does not require task-specific tuning of the word weighting function nor does it rely on the parse trees.

Paragraph Vector: A distributed memory model :

In our Paragraph Vector framework (see Figure 2), every paragraph is mapped to a unique vector, represented by a column in matrix D and every word is also mapped to a unique vector, represented by a column in matrix W . The paragraph vector and word vectors are averaged or concate-nated to predict the next word in a context.

The paragraph token can be thought of as another word. It acts as a memory that remembers what is missing from the current context – or the topic of the paragraph. For this reason, we often call this model the Distributed Memory Model of Paragraph Vectors (PV-DM). The contexts are fixed-length and sampled from a sliding window over the paragraph. The paragraph vector is shared across all contexts generated from the same paragraph but not across paragraphs. The word vector matrix W , how-ever, is shared across paragraphs. I.e., the vector for "pow - erful" is the same for all paragraphs.

The paragraph vectors and word vectors are trained using stochastic gradient descent and the gradient is obtained via backpropagation. At every step of stochastic gradient de-scent, one can sample a fixed-length context from a random paragraph, compute the error gradient from the network in Figure 2 and use the gradient to update the parameters in our model. At prediction time, one needs to perform an inference step to compute the paragraph vector for a new paragraph. This is also obtained by gradient descent.

Suppose that there are N paragraphs in the corpus, M words in the vocabulary, and we want to learn paragraph vectors such that each paragraph is mapped to p dimen-sions and each word is mapped to q dimensions, then the model has the total of $N \times p + M \times q$ parameters (ex-cluding the softmax parameters). Even though the number of parameters can be large when N is large, the updates during training are typically sparse and thus efficient.
After being trained, the paragraph vectors can be used as features for the paragraph (e.g., in lieu of or in addition to bag-of-words). We can feed these features directly to conventional machine learning techniques such as logistic regression, support vector machines or K-means.

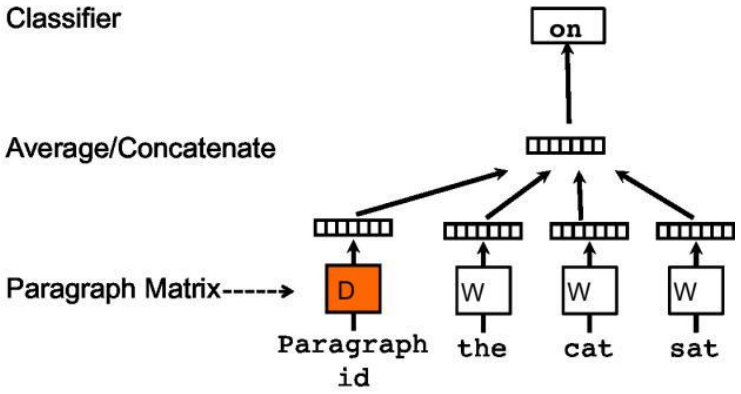


Figure 2 : A framework for learning paragraph vector.

4.3 Machine Learning Models

4.3.1 Naïve Bayes

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. There is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable. For example, a fruit may be considered to be an apple if it is red, round, and about 10 cm in diameter. A naive Bayes classifier considers each of these features to contribute independently to the probability that this fruit is an apple, regardless of any possible correlations between the color, roundness, and diameter features.

For some types of probability models, naive Bayes classifiers can be trained very efficiently in a supervised learning setting. In many practical applications, parameter estimation for naive Bayes models uses the method of maximum likelihood; in other words, one can work with the naive Bayes model without accepting Bayesian probability or using any Bayesian methods.

Despite their naive design and apparently oversimplified assumptions, naive Bayes classifiers have worked quite well in many complex real-world situations. In 2004, an analysis of the Bayesian classification problem showed that there are sound theoretical reasons for the apparently implausible efficacy of naive Bayes classifiers. Still, a comprehensive comparison with other classification algorithms in 2006 showed that Bayes classification is outperformed by other approaches, such as boosted trees or random forests. An advantage of naive Bayes is that it only requires a small number of training data to estimate the parameters necessary for classification.

Abstractly, naïve Bayes is a conditional probability model: given a problem instance to be classified, represented by a vector $x = \{x_1, \ldots, x_n\}$ representing some n features (independent variables), it assigns to this instance probabilities : $P(C_k \mid x_1, \ldots, x_n)$, for each of K possible outcomes or classes C_k.

The problem with the above formulation is that if the number of features *n* is large or if a feature can take on a large number of values, then basing such a model on probability tables is infeasible. Using Bayes' theorem, the conditional probability can be decomposed as

$$p(C_k \mid \mathbf{x}) = \frac{p(C_k)\, p(\mathbf{x} \mid C_k)}{p(\mathbf{x})}$$

In plain English, using Bayesian probability terminology, the above equation can be written as

$$\text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}$$

In practice, there is interest only in the numerator of that fraction, because the denominator does not depend on and the values of the features are given, so that the denominator is effectively constant. The numerator is equivalent to the joint probability model : $P(C_k \mid x_1, \ldots, x_n)$, which can

be rewritten as follows, using the chain rule for repeated applications of the definition of conditional probability:

$$p(x_i \mid x_{i+1}, \ldots, x_n, C_k) = p(x_i \mid C_k)$$

Thus, the joint model can be expressed as :

$$\begin{aligned} p(C_k \mid x_1, \ldots, x_n) &\propto p(C_k, x_1, \ldots, x_n) \\ &= p(C_k)\, p(x_1 \mid C_k)\, p(x_2 \mid C_k)\, p(x_3 \mid C_k)\, \cdots \\ &= p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)\,, \end{aligned}$$

where α denotes proportionality.

This means that under the above independence assumptions, the conditional distribution over the class variable C is:

$$p(C_k \mid x_1, \ldots, x_n) = \frac{1}{Z} p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)$$

Where the evidence ,

$$Z = p(\mathbf{x}) = \sum_k p(C_k)\, p(\mathbf{x} \mid C_k)$$

is a scaling factor dependent only on $x_1, \ldots, x_n$, that is, a constant if the values of the feature variables are known.

Constructing a classifier from the probability model :

The discussion so far has derived the independent feature model, that is, the naïve Bayes probability model. The naïve Bayes classifier combines this model with a decision rule. One common rule is to pick the hypothesis that is most probable; this is known as the maximum a posteriori or *MAP* decision rule. The corresponding classifier, a Bayes classifier, is the function that assigns a class label $\hat{y} = C_k$ for some k as follows :

$$\hat{y} = \underset{k \in \{1,\ldots,K\}}{\operatorname{argmax}}\ p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k).$$

A class's prior may be calculated by assuming equiprobable classes (*i.e.*, priors = 1/<number of classes>), or by calculating an estimate for the class probability from the training set (*i.e.*, <prior for a given class> = <number of samples in the class>/<total number of samples>). To estimate the parameters for a feature's distribution, one must assume a distribution or generate nonparametric models for the features from the training set.

The assumptions on distributions of features are called the "event model" of the naïve Bayes classifier. For discrete features like the ones encountered in document classification (include spam filtering), multinomial and Bernoulli distributions are popular. These assumptions lead to two distinct models, which are often confused.

We used Gaussian naïve Bayes classifier for our project which is described below.

Gaussian naïve Bayes :

When dealing with continuous data, a typical assumption is that the continuous values associated with each class are distributed according to a normal (or Gaussian) distribution. For example, suppose the training data contains a continuous attribute, x. We first segment the data by the class, and then compute the mean and variance of x in each class. Let μ_k be the mean of the values in x associated with class C_k, and let σ_k^2 be the Bessel corrected variance of the values in x associated with class C_k. Suppose we have collected some observation value v. Then, the probability distribution of v given a class C_k, $p(x = v \mid C_k)$, can be computed by plugging v into the equation for a normal distribution parameterized by μ_k and σ_k^2. That is ,

$$p(x = v \mid C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}}\, e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}}$$

4.3.2 Logistic Regression

In statistics, the logistic model (or logit model) is used to model the probability of a certain class or event existing such as pass/fail, win/lose, alive/dead or healthy/sick. This can be extended to model several classes of events such as determining whether an image contains a cat, dog, lion, etc. Each object being detected in the image would be assigned a probability between 0 and 1 and the sum adding to one.

Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression[14] (or logit regression) is estimating the parameters of a logistic model (a form of binary regression). Mathematically, a binary logistic model has a dependent variable with two possible values, such as pass/fail which is represented by an indicator variable, where the two values are labeled "0" and "1". In the logistic model, the log-odds (the logarithm of the odds) for the value labeled "1" is a linear combination of one or more independent variables ("predictors"); the independent variables can each be a binary variable (two classes, coded by an indicator variable) or a continuous variable (any real value). The corresponding probability of the value labeled "1" can vary between 0 (certainly the value "0") and 1 (certainly the value "1"), hence the labeling; the function that converts log-odds to probability is the logistic function, hence the name. The unit of measurement for the log-odds scale is called a logit, from logistic unit, hence the alternative names. Analogous models with a different sigmoid function instead of the logistic function can also be used, such as the probit model; the defining characteristic of the logistic model is that increasing one of the independent variables multiplicatively scales the odds of the given outcome at a constant rate, with each independent variable having its own parameter; for a binary dependent variable this generalizes the odds ratio.

In a binary logistic regression model, the dependent variable has two levels (categorical). Outputs with more than two values are modeled by multinomial logistic regression and, if the multiple categories are ordered, by ordinal logistic regression (for example the proportional odds ordinal logistic model[15]). The logistic regression model itself simply models probability of output in terms of input and does not perform statistical classification (it is not a classifier),

though it can be used to make a classifier, for instance by choosing a cutoff value and classifying inputs with probability greater than the cutoff as one class, below the cutoff as the other; this is a common way to make a binary classifier. The coefficients are generally not computed by a closed-form expression, unlike linear least squares; see § Model fitting. The logistic regression as a general statistical model was originally developed and popularized primarily by Joseph Berkson, beginning in Berkson (1944), where he coined "logit".

We used this Logistic Regression model present in the linear model class of scikit-learn . Logistic regression can be viewed as $\sigma(t) = 1 / (1+e^{-t})$.

4.3.3 Support Vector Machine (SVM)

In machine learning, support-vector machines (SVMs, also support-vector networks[16]) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on the side of the gap on which they fall. In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

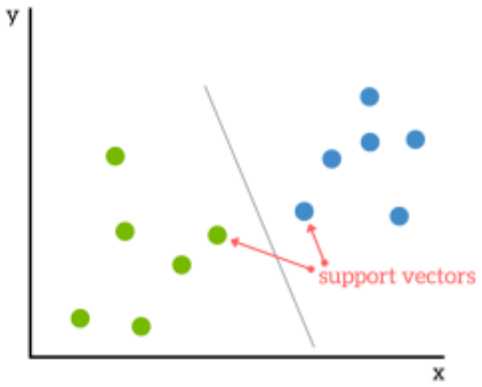


Figure 3 : Representation of two classes of clusters separated by a hyperplane

When data are unlabelled, supervised learning is not possible, and an unsupervised learning approach is required, which attempts to find natural clustering of the data to groups, and then map new data to these formed groups. The support-vector clustering[17] algorithm, created by Hava Siegelmann and Vladimir Vapnik, applies the statistics of support vectors, developed in the support vector machines algorithm, to categorize unlabeled data, and is one of the most widely used clustering algorithms in industrial applications.

More formally, a support-vector machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks like outliers detection.[18] Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (so-called functional margin), since in general the larger the margin, the lower the generalization error of the classifier.[19]

Whereas the original problem may be stated in a finite-dimensional space, it often happens that the sets to discriminate are not linearly separable in that space. For this reason, it was proposed[by whom?] that the original finite-dimensional space be mapped into a much higher-dimensional space, presumably making the separation easier in that space. To keep the computational load reasonable, the mappings used by SVM schemes are designed to ensure that dot products of pairs of input data vectors may be computed easily in terms of the variables in the original space, by defining them in terms of a kernel function k(x,y) selected to suit the problem.[20] The hyperplanes in the higher-dimensional space are defined as the set of points whose dot product with a vector in that space is constant, where such a set of vectors is an orthogonal (and thus minimal) set of vectors that defines a hyperplane. The vectors defining the hyperplanes can be chosen to be linear combinations with parameters α_i of images of feature vectors x_i that occur in the data base.

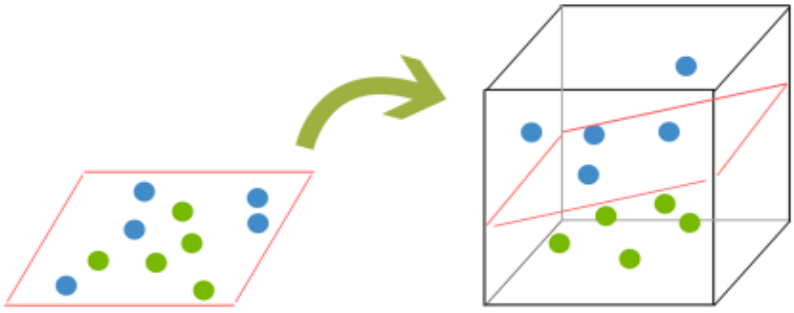


Figure 4 : Mapping of data into higher dimension to form a hyperplane.

We used Support-Vector-Machine with Radial Basis Function kernel as the next model. The reason we use this kernel is that two Doc2Vec feature vectors will be close to each other if their corresponding documents are similar, so the distance computed by the kernel function should still represent the original distance.

The Radial Basis Function is

$$K(x,x') = \exp(-\gamma \| x - x' \|^2) \text{ for } \gamma > 0$$

It correctly represents the relationship we desire and it is a common kernel for SVM.

4.3.4 FeedForward Neural Network

A feedforward neural network is an artificial neural network wherein connections between the nodes do *not* form a cycle. As such, it is different from its descendant: recurrent neural networks.

The feedforward neural network was the first and simplest type of artificial neural network devised. In this network, the information moves in only one direction, forward, from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network.

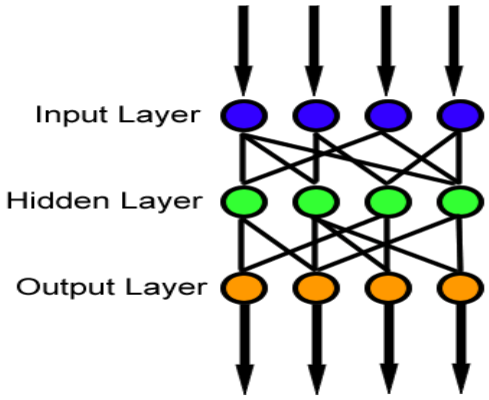


Figure 5 : Representation of feedforward neural network where information flow in one direction

We implemented feed-forward neural network model using Keras. Neural networks are commonly used in modern NLP applications [21], in contrast to older approaches which primarily focused on linear models such as SVM's and logistic regression. Our neural network implementations use three hidden layers. In the Tensorflow implementation, all layers had 300 neurons each, and in the Keras implementation we used layers of size 256, 256, and 80, interspersed with dropout layers to avoid overfitting. For our activation function, we chose the Rectified Linear Unit (ReLU), which has been found to perform well in NLP applications [21].

This has a fixed-size input $x \in R^{1\times300}$

$h_1 = \text{ReLU}\ (W_1x + b_1)$
$h_2 = \text{ReLU}\ (W_2h_1 + b_2)$
$y = \text{Logits}\ (W_3h_2 + b_3)$

4.3.5 Long-Short Term Memory (LSTM)

The Long-Short Term Memory (LSTM) unit was proposed by Hochreiter and Schmidhuber[22]. It is good at classifying serialized objects because it will selectively memorize the previous input and use that together with the current input to make predictions. The News Content(text)in our problem is inherently serialized. The order of the words carries the important information of the sentence. So the LSTM model suits our problem.

Since the order of the words is important for the LSTM unit, we cannot use the Doc2Vec for preprocessing because it will transfer the entire document into one vector and lose the order information. To prevent that, we use the word embedding instead.

We first clean the text data by removing all characters which are neither letters nor numbers. Then we count the frequency of each word appearing in our training dataset to 5000 most common words and give each one an unique integer ID. For example, the most common word will have ID 0, and the second most common one will have 1, etc. After that we replace each common word with its assigned ID and delete all uncommon words. Notice that the 5000 most common words cover the most of the text, as shown in Figure 11, so we only lose little information but transfer the string to a list of integers. Since the LSTM unit requires a fixed input vector length, we truncate the list longer than 500 numbers because more than half of the news is longer than 500 words as shown in Figure 12. Then for those lists shorter than 500 words, we pad 0's at the beginning of the list. We also delete the data with only a few words since they don't carry enough information for training. By doing this, we transfer the original text string to a fixed length integer vector while preserving the words order information. Finally we use word embedding to transfer each word ID to a 32-dimension vector. The word embedding will train each word vector based on word similarity. If two words frequently appear together in the text, they are thought to be more similar and the distance of their corresponding vectors is small.

The pre-processing transfers each news in raw text into a fixed size matrix. Then we feed the processed training data into the LSTM unit to train the model. The LSTM is still a neural network. But Different From Fully Connected Neural Network,it has a cycle in the neuron connections. So the previous state (or memory) of the LSTM unit c_t will play a role in new prediction h_t.

$$
\begin{aligned}
h_t &= o_t \,.\tanh(\, c_t \,) \\
c_t &= f_t \cdot c_{t-1} + i_t \cdot c'_t \\
c'_t &= \tanh(\, x_t W_c + h_{t-1} U_c + b_c \,) \\
o_t &= \sigma(\, x_t W_o + h_{t-1} U_o + b_o \,) \\
i_t &= \sigma(\, x_t W_i + h_{t-1} U_i + b_i \,) \\
f_t &= \sigma(\, x_t W_f + h_{t-1} U_f + b_f \,)
\end{aligned}
$$

CHAPTER 5

Evaluation and Comparison of Results

5.1 Cleaning

Pre-processing data is a normal first step before training and evaluating the data using a neural network. Machine learning algorithms are only as good as the data you are feeding them. It is crucial that data is formatted properly and meaningful features are included in order to have sufficient consistency that will result in the best possible results. As seen in [23], for computer vision machine learning algorithms, pre-processing the data involves many steps including normalizing image inputs and dimensionality reduction. The goal of these is to take away some of the unimportant distinguishing features between different images. Features like the darkness or brightness are not beneficial in the task of labeling the image. Similarly, there are portions of text that are not beneficial in the task of labeling the text as real or fake.

The task of pre-processing data is often an iterative task rather than a linear one. This was the case in this project where we used a new and not yet standardized dataset. As we found certain unmeaningful features that the neural net was learning, we learned what more we needed to pre-process from the data.

So first we removed all the special characters and the punctuations and then converted our texts to lower cases for further proceedings.

Removing Stop Words

In computing, stop words are words which are filtered out before or after processing of natural language data (text). Though "stop words" usually refers to the most common words in a language, there is no single universal list of stop words used by all natural language processing tools, and indeed not all tools even use such a list. Some tools specifically avoid removing these stop words to support phrase search.

Any group of words can be chosen as the stop words for a given purpose. For some search engines, these are some of the most common, short function words, such as the, is, at, which, and on. In this case, stop words can cause problems when searching for phrases that include them, particularly in names such as "The Who", "The The", or "Take That". Other search engines remove some of the most common words—including lexical words, such as "want"—from a query in order to improve performance.

In our project we are going to use NLTK to remove the stopwords . So after converting our texts to lower cases we are going to remove the stopwords using NLTK software toolkit.

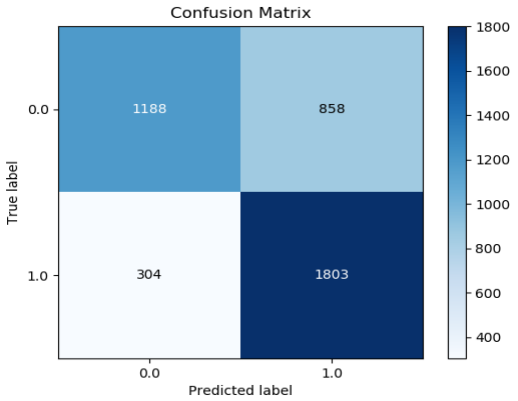
5.2 Feature Extraction

The embeddings used for the majority of our modelling are generated using the Doc2Vec model.The goal is to produce a vector representation of each article. Before applying Doc2Vec,we perform some basic pre-processing of the data. As explained above this includes removing stop words,deleting special characters and punctuation, and converting all text to lowercase. This produces a comma-separated list of words, which can be input into the Doc2Vec algorithm to produce an 300-length embedding vector for each article.

Doc2Vec is a model developed in 2014 based on the existing Word2Vec model, which generates vector representations for words [12]. Word2Vec represents documents by combining the vectors of the individual words, but in doing so it loses all word order information. Doc2Vec expands on Word2Vec by adding a ”document vector” to the output representation, which contains some information about the document as a whole, and allows the model to learn some information about word order. Preservation of word order information makes Doc2Vec useful for our project, as we are aiming to detect subtle differences between text documents. After this extraction is done the processed text is then fed into each machine learning model for classification.

5.3 Comparison of Results

We got the confusion matrix of our machine learning as follows.



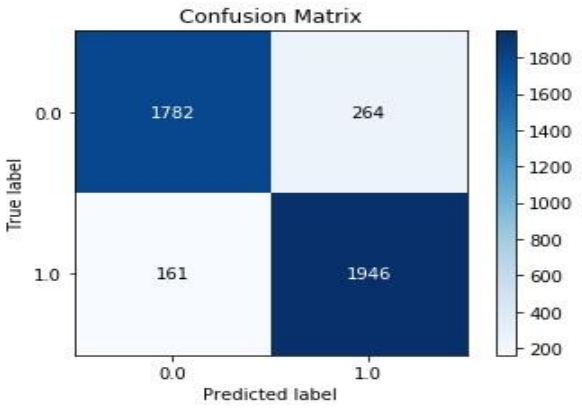


Figure 6 : Confusion matrix of naïve-Bayes. *Figure 7 : Confusion matrix of Logistic Regression*

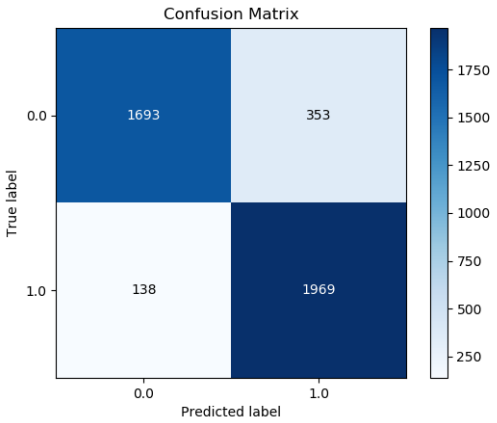


Figure 8 : Confusion matrix of SVM

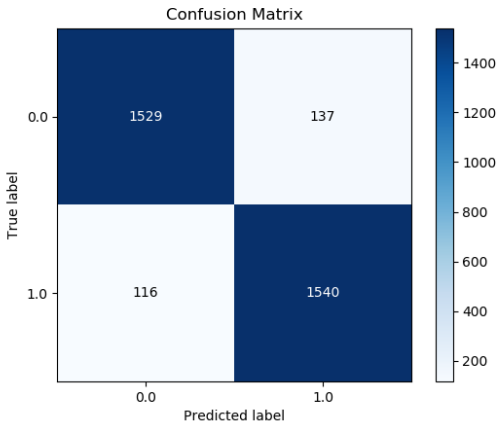


Figure 9 : Confusion matrix of Neural Network (using keras)

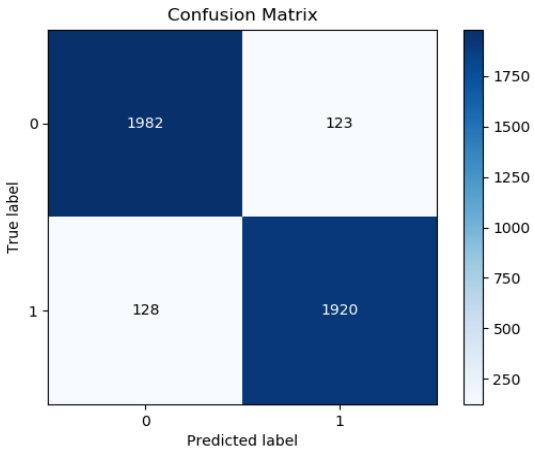


Figure 10 : Confusion matrix of LSTM (using Keras)

Classifier	**Precision**	**Recall**	**F-Measure**
Naive Bayes	0.68	0.86	0.76
SVM	0.85	0.93	0.89
Logistic Regression	0.88	0.93	0.90
Neural Network(keras)	0.92	0.93	0.92
LSTM(keras)	0.94	0.94	0.94

Table 2

We compared our models using their Confusion Matrices to calculate the Precision, Recall and the F1 scores. *Table 2* shows our results.

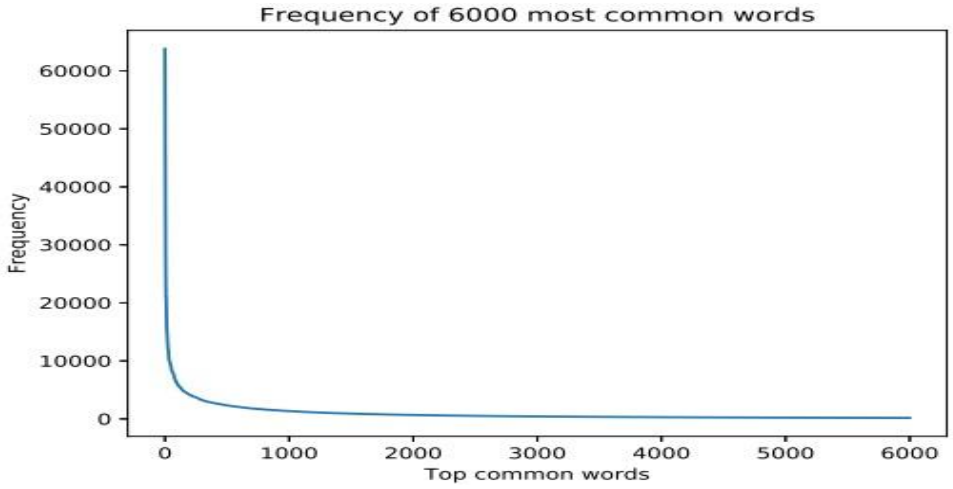


Figure 11 : Frequency of top common words

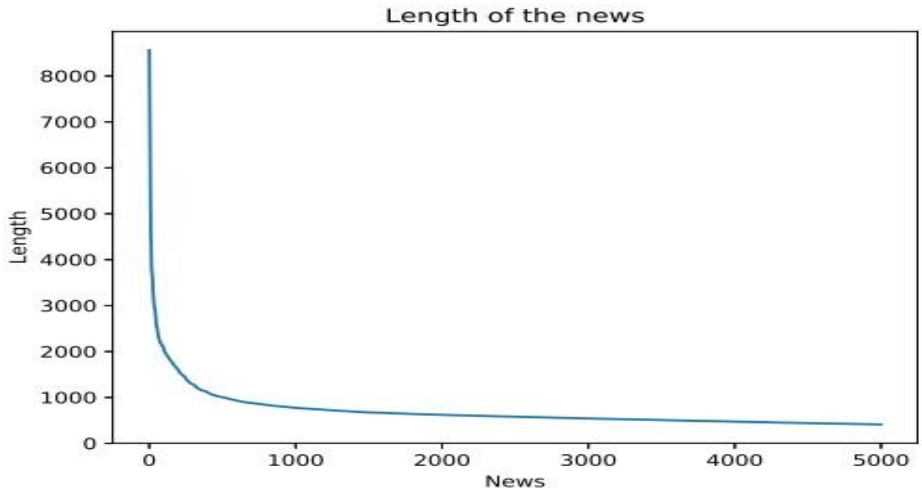


Figure 12 : Length of the news

We observed that LSTM gave us the best results.We had to use a different set of embeddings for preprocessing the data to be fed to our LSTM model. It uses an ordered set of Word2Vec representations.

The LSTM achieves the highest F1 score in comparison to all the other models, followed by the Neural Network model using Keras. One of the reasons that LSTM performs so well is because the text is inherently serialized. All the other models use the Doc2Vec to get their feature vectors and hence, they rely on the Doc2Vec to extract the order information and perform a classificationon it. On the other hand, the LSTM model preserves the order using a different preprocessing method and makes predictions based on both the words and their order. This is how it outperforms others.

CHAPTER 6

Future Work

A complete, production-quality classifier will incorporate many different features beyond the vectors corresponding to words in the text. For fake news detection, we can add as features the source of the news, including any associated URLs, the topic (e.g., science, politics, sports, etc.), publishing medium (blog , print ,social media),country or geographic region of origin, publication year, as well as linguistic features not exploited in this exercise use of capitalization, fraction of words that are proper nouns (using gazetteers), and others.

Besides, we can also aggregate the well-performed classifiers to achieve better accuracy. For example, using bootstrap aggregating for the Neural Network, LSTM and SVM models to get better prediction results.

An ambitious work would be to search the news on the Internet and compare the search results with the original news. Since the search result is usually reliable, this method should be more accurate, but also involves natural language understanding because the search results will not be exactly the same as the original news. So we will need to compare the meaning of two contents and decide whether they mean the same thing.

CHAPTER 7

Appendix

Here we will be mentioning some important programming parts related to our project.

getEmbeddings.py

Here we will mention the data pre-processing part , which consists of feature extraction part, before being used to train in models – naïve bayes, logistic regression, support vector machine and FeedForward neural network.

```
#  Get rid of the non-letter and non-number characters  and get rid of stop words.
def textClean(text):
    #Passing the regex expression
    text = re.sub(r"[^A-Za-z0-9^,!.\/'+-=]", " ", text)

    # making all the letters lowercase
    text = text.lower().split()

    # get rid of stop words
    stops = set(stopwords.words("english"))
    text = [w for w in text if not w in stops]
    text = " ".join(text)

    return (text)

#  Cleaning the  text , get rid of punctuation
def cleanup(text):
    text = textClean(text)
    text = text.translate(str.maketrans("", "", string.punctuation))
    return text

#  Utility function for the Doc2Vec model
#  It generates the  comma separated list of words in paragraph/article
#  and uses a Label for each article as Text_index
#  NOTE:index is the index in the dataset
def constructLabeledSentences(data):
    sentences = []
    for index, row in data.iteritems():
        sentences.append(LabeledSentence(utils.to_unicode(row).split(), ['Text' + '_%s' % str(index)]))
    return sentences

# Taking the dimension of vector as 300
vector_dimension = 300
```

```
#  Generate Doc2Vec training and testing data
def  Doc2Vec_model(path):
    # Reading the data source
    data = pd.read_csv(path)

    # Finding missing values in the dataset
    missing_rows = []
    for i in range(len(data)):
        if data.loc[i, 'text'] != data.loc[i, 'text']:
            missing_rows.append(i)
    data = data.drop(missing_rows).reset_index().drop(['index','id'],axis=1)

    # Calling the cleanup function for each article
    for i in range(len(data)):
        data.loc[i, 'text'] = cleanup(data.loc[i,'text'])

    # Creating the Label sentences
    x = constructLabeledSentences(data['text'])

    # Reading the labels as y variable
    y = data['label'].values

    # Creating the Doc2Vec model
    text_model = Doc2Vec(min_count=1, window=5, vector_size=vector_dimension, sample=1e-4,
                         negative=5, workers=7, epochs=10, seed=1)
    text_model.build_vocab(x)
    text_model.train(x, total_examples=text_model.corpus_count, epochs=text_model.iter)

    return x, y,text_model

#  Creating tranning and testing dataset as numpy arrays for faster processing.
def getEmbeddings(path):
    # Calling the Doc2Vec_model to get the model
    x,y,text_model = Doc2Vec_model(path)

    # Spliting the dataset for training and testing
    train_size = int(0.8 * len(x))
    test_size = len(x) - train_size

    # Creating Numpy arrays with 0 value
    text_train_arrays = np.zeros((train_size, vector_dimension))
    text_test_arrays = np.zeros((test_size, vector_dimension))
    train_labels = np.zeros(train_size)
    test_labels = np.zeros(test_size)

    # Creating Tranning numpy arrays
    for i in range(train_size):
        text_train_arrays[i] = text_model.docvecs['Text_' + str(i)]
        train_labels[i] = y[i]

    # Creating Testing numpy arrays
```

```
    j = 0
    for i in range(train_size, train_size + test_size):
        text_test_arrays[j] = text_model.docvecs['Text_' + str(i)]
        test_labels[j] = y[i]
        j = j + 1


    return text_train_arrays, text_test_arrays, train_labels, test_labels
```

getEmbeddings2.py

Here we will mention the pre-processing part required for our LSTM model. Everything is same as mentioned in getEmbeddings.py except instead we will not be using Doc2vec for preprocessing.

```
# Cleaning the data
def clean_data():
    path = 'datasets/train.csv'
    vector_dimension=300

    data = pd.read_csv(path)

    # Dropping the missing rows
    missing_rows = []
    for i in range(len(data)):
        if data.loc[i, 'text'] != data.loc[i, 'text']:
            missing_rows.append(i)
    data = data.drop(missing_rows).reset_index().drop(['index','id'],axis=1)

    # Following the cleanup operation as mentioned in getEmbeddings.py
    for i in range(len(data)):
        data.loc[i, 'text'] = cleanup(data.loc[i,'text'])

    data = data.sample(frac=1).reset_index(drop=True)

    x = data.loc[:,'text'].values
    y = data.loc[:,'label'].values

    train_size = int(0.8 * len(y))
    test_size = len(x) - train_size

    xtr = x[:train_size]
    xte = x[train_size:]
    ytr = y[:train_size]
    yte = y[train_size:]
```

LogisticRegression.py

```
# Use the built-in Logistic Regression classifier of scikit learn library

# Creating the classifier LogisticRegression() and fitting the model with xte(xtraining) and
# ytr(ytraining)
log_R = linear_model.LogisticRegression()
log_R.fit(xtr,ytr)

# Saving the models in the logistic_regression.sav file so that we can use pre trained model
model_file = 'logistic_regression_model.sav'
pickle.dump(log_R,open(model_file,'wb'))

# Predicting the y_pred values for xte(xtest)
y_pred = log_R.predict(xte)
```

NaïveBayes.py

```
# Use the built-in Naive Bayes classifier of scikit learn library

# Creating the classifier GaussianNB() and fitting the model with xte(xtraining) and
ytr(ytraining)
gnb = GaussianNB()
gnb.fit(xtr,ytr)

# Saving the models in the naive_bayes.sav file so that we can use pre trained model
model_file = 'naive_bayes_model.sav'
pickle.dump(gnb,open(model_file,'wb'))

# Predicting the y_pred values for xte(xtest)
y_pred = gnb.predict(xte)
```

SVM.py

```
# Use the built-in SVM for classification

# Creating the classifier LinearSVC() and fitting the model with xte(xtranning) and
ytr(ytranning)
clf = SVC(kernel='rbf',gamma='auto')
clf.fit(xtr, ytr)

# Saving the models in the svm_model.sav file so that we can use pre trained model
model_file = 'svm_model.sav'
pickle.dump(clf,open(model_file,'wb'))

# Predicting the y_pred values for xte(xtest)
```

```
    y_pred = clf.predict(xte)
```

FeedForward.py

```
def baseline_model():
    # Neural network with 3 hidden layers
    model = Sequential()
    model.add(Dense(256, input_dim=300, activation='relu',  kernel_initializer='normal'))
    model.add(Dropout(0.3))
    model.add(Dense(256, activation='relu', kernel_initializer='normal'))
    model.add(Dropout(0.5))
    model.add(Dense(80, activation='relu', kernel_initializer='normal'))
    model.add(Dense(2, activation="softmax", kernel_initializer='normal'))

    # gradient descent
    sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)

    # configure the learning process of the model
    model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
    return model

# Train the model
model = baseline_model()
model.summary()
x_train, x_test, y_train, y_test = train_test_split(xtr, ytr, test_size=0.2, random_state=42)
label_encoder = LabelEncoder()
label_encoder.fit(y_train)
encoded_y = np_utils.to_categorical((label_encoder.transform(y_train)))
label_encoder.fit(y_test)
encoded_y_test = np_utils.to_categorical((label_encoder.transform(y_test)))
estimator = model.fit(x_train, encoded_y, epochs=20, batch_size=64)
print("Model Trained!")
```

LSTM.py

```
# Storing most common words
most_common = cnt.most_common(top_words + 1)
word_bank = {}
id_num = 1
for word, freq in most_common:
    word_bank[word] = id_num
    id_num += 1

# Encode the sentences
for news in x_train:
    i = 0
```

```
    while i < len(news):
        if news[i] in word_bank:
            news[i] = word_bank[news[i]]
            i += 1
        else:
            del news[i]

y_train = list(y_train)
y_test = list(y_test)

# Delete the short news
i = 0
while i < len(x_train):
    if len(x_train[i]) > 10:
        i += 1
    else:
        del x_train[i]
        del y_train[i]

# Generating test data
x_test = []
for x in xte:
    x_test.append(x.split())

# Encode the sentences
for news in x_test:
    i = 0
    while i < len(news):
        if news[i] in word_bank:
            news[i] = word_bank[news[i]]
            i += 1
        else:
            del news[i]

# Truncate and pad input sequences
max_review_length = 500
X_train = sequence.pad_sequences(x_train, maxlen=max_review_length)
X_test = sequence.pad_sequences(x_test, maxlen=max_review_length)
```

References

[1] J. Soll, T. Rosenstiel, A. D. Miller, R. Sokolsky, and J. Shafer. (2016, Dec) The long and brutal history of fake news. [Online]. Available: https://www.politico.com/magazine/story/2016/12/fake-news-history-long-violent-214535

[2] C. Wardle. (2017, May) Fake news. it's complicated. [Online]. Available: https://firstdraftnews.com/fake-news-complicated/

[3] T. Ahmad, H. Akhtar, A. Chopra, and M. Waris Akhtar, "Satire detection from web documents using machine learning methods," pp. 102–105, 09 2014.

[4] C. Kang and A. Goldman. (2016, Dec) In washington pizzeria attack, fake news brought real guns. [Online]. Available: https://www.nytimes.com/2016/12/05/business/media/comet-ping-pong-pizza-shooting-fake-news-consequences.html

[5] C. Domonoske. (2016, Nov) Students have 'dismaying' inability to tell fake news from real, study finds. [Online]. Available: https://www.npr.org/sections/thetwo-way/2016/11/23/503129818/study-finds-students-have-dismaying-inability-to-tell-fake-news-from-real

[6] M. T. Banday and T. R. Jan, "Effectiveness and limitations of statistical spam filters," arXiv preprint arXiv:0910.2540, 2009.

[7] S. Sedhai and A. Sun, "Semi-supervised spam detection in twitter stream," arXiv preprint arXiv:1702.01032, 2017.

[8] A. Bhowmick and S. M. Hazarika, "Machine learning for e-mail spam filtering: Review, techniques and trends," arXiv preprint arXiv:1606.01042, 2016.

[9] Fake news challenge stage 1 (fnc-i): Stance detection. [Online]. Available: http://www.fakenewschallenge.org/

[10] Datasets, Kaggle, https://www.kaggle.com/c/fake-news/data, February, 2018.

[11] Quoc, L., Mikolov, T., Distributed Representations of Sentences and Documents, https://arxiv.org/abs/1405.4053, May, 2014.

[12] Le, Quoc; et al. (2014). "Distributed Representations of Sentences and Documents". arXiv:1405.4053

[13] Harris, Zellig (1954). "Distributional Structure".

[14] Tolles, Juliana; Meurer, William J (2016). "Logistic Regression Relating Patient Characteristics to Outcomes". JAMA.

[15] Walker, SH; Duncan, DB (1967). "Estimation of the probability of an event as a function of several independent variables". Biometrika.

[16] Cortes, Corinna; Vapnik, Vladimir N. (1995). "Support-vector networks" (PDF). Machine Learning.

[17] Ben-Hur, Asa; Horn, David; Siegelmann, Hava; Vapnik, Vladimir N. ""Support vector clustering" (2001);". Journal of Machine Learning Research. 2: 125–137.

[18] "1.4. Support Vector Machines — scikit-learn 0.20.2 documentation". Archived from the original on 2017-11-08. Retrieved 2017-11-08.

[19] Hastie, Trevor; Tibshirani, Robert; Friedman, Jerome (2008). The Elements of Statistical Learning : Data Mining, Inference, and Prediction (PDF) (Second ed.). New York: Springer. p. 134.

[20] Press, William H.; Teukolsky, Saul A.; Vetterling, William T.; Flannery, Brian P. (2007). "Section 16.5. Support Vector Machines". Numerical Recipes: The Art of Scientific Computing (3rd ed.). New York: Cambridge University Press. ISBN 978-0-521-88068-8. Archived from the original on 2011-08-11.

[21] Goldberg,Y.,APrimeron Neural Network Models for Natural Language Processing, https://arxiv. org/pdf/1510.00726.pdf, October, 2015.

[22] Hochreiter, S., Jrgen, S Long short-term memory. http://www.bioinf.jku.at/publications/older/2604.pdf, October, 1997.

[23] N. B, “Image data pre-processing for neural networks becoming human: Artificial intelligence magazine,” Sep 2017. [Online]. Available: https://becominghuman.ai/image-data-pre-processing-for-neural-networks-498289068258