FAKE NEWS DETECTION

ECE2010, CONTROL SYSTEMS

A Project Report

Submitted by

Vikram Baruah	18BEC0967
Sankalp Arora	18BEC0930
Lakshya Punjabi	18BEC0371

Under the guidance of

Prof. Bagubali A

In F1 SLOT

School of Electronics Engineering VIT University, Vellore Tamil Nadu - 632 014

November 2020

TABLE OF CONTENTS

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 a 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.

Introduction

Fake news, defined as a made-up story with an intention to deceive, has been widely cited as a contributing factor to the outcome of the 2016 United States presidential election.
While Mark Zuckerberg, Facebook's CEO, made a public statement denying that Facebook had an effect on the outcome of the election, Facebook and other online media outlets have begun to develop strategies for identifying fake news and mitigating its spread. Zuckerberg admitted identifying fake news is difficult, writing, "this is an area where I believe we must proceed very carefully though. Identifying the truth is complicated."
Fake news is increasingly becoming a menace to our society. It is typically generated for commercial interests to attract viewers and collect advertising revenue. However, people and groups with potentially malicious agendas have been known to initiate fake news in order to influence events and policies around the world. It is also believed that circulation of fake news had material impact on the outcome of the 2016 US Presidential Election.

Problem Statement

Develop a machine learning program to identify fake/unreliable news based on content acquired

Data

The datasets used for this project were drawn from Kaggle. 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, in order to train our models.
A full training dataset has the following attributes:

id: unique id for a news article

title: the title of a news article author:

author of the news article

text: the text of the article; incomplete in some cases

label: a label that marks the article as potentially unreliable
1: unreliable
0: reliable

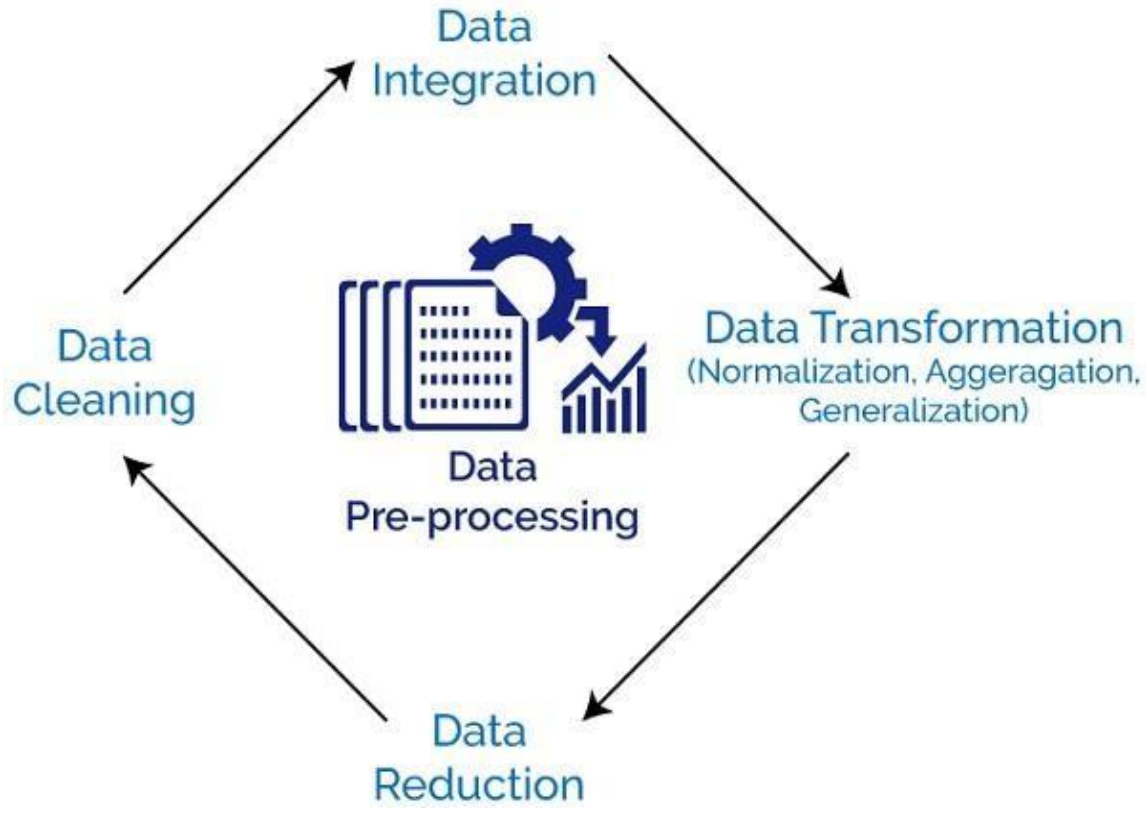
SAMPLE DATA

id	title	author	text	label
0	House Der	Darrell Luc	House	1
1	FLYNN: Hi	Daniel J. F	Ever get th	0
2	Why the T	Consortiur	Why the	1
3	15 Civilian	Jessica Pur	Videos 15	1
4	Iranian wc	Howard Pc	Print	1
5	Jackie Ma	Daniel Nu	In these tr	0
6	Life: Life C	nan	Ever	1
7	BenoÃ®t H	Alissa J. Ru	PARIS â€"	0
8	Excerpts F	nan	Donald J.	0
9	A Back-Ch	Megan Tw	A week be	0
10	Obamaâ€	Aaron Klei	Organizing	0
11	BBC Come	Chris Tom	The BBC p	0
12	Russian Re	Amando F	The	1
13	US Official	Jason Ditz	Clinton	1
14	Re: Yes, Th	AnotherAr	Yes,	
BART SIMPSONSON				
Hey	itâ€™s jus	channels	and programs fellating them da	
Itâ€™s not	I imagine	oil compa	difficult to know who to trust o	
In any soc	most people do nothing. Itâ€™s up to the minority tc			
If I read the article correctly the government is targeting conserv				
The DNC is	stupid and	but these	j@ck@sses ramp it up to 11.) Ta	
I almost pc	which wa	especially	1	

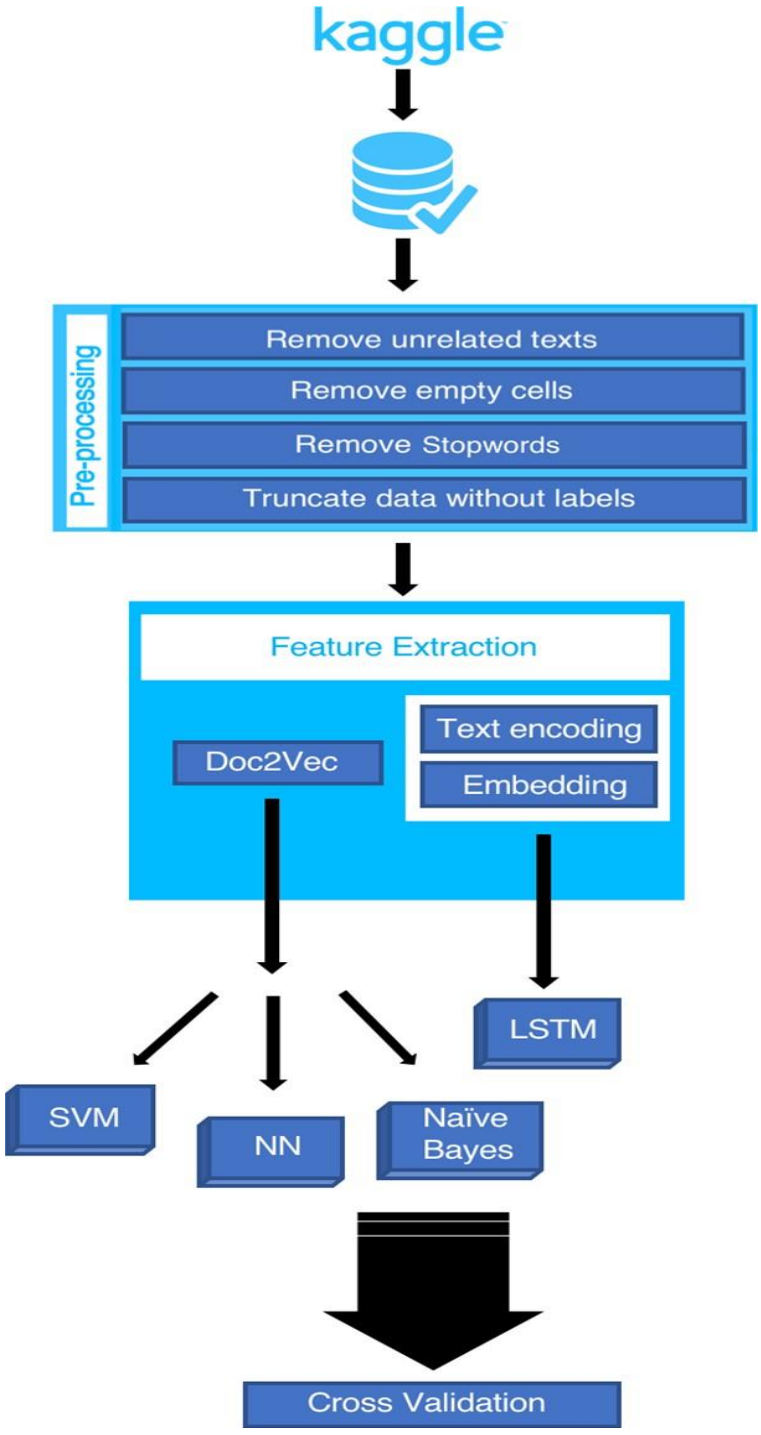
Feature Extraction and Pre-processing

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. 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 a 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. 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 application, as we are aiming to detect subtle differences between text documents.

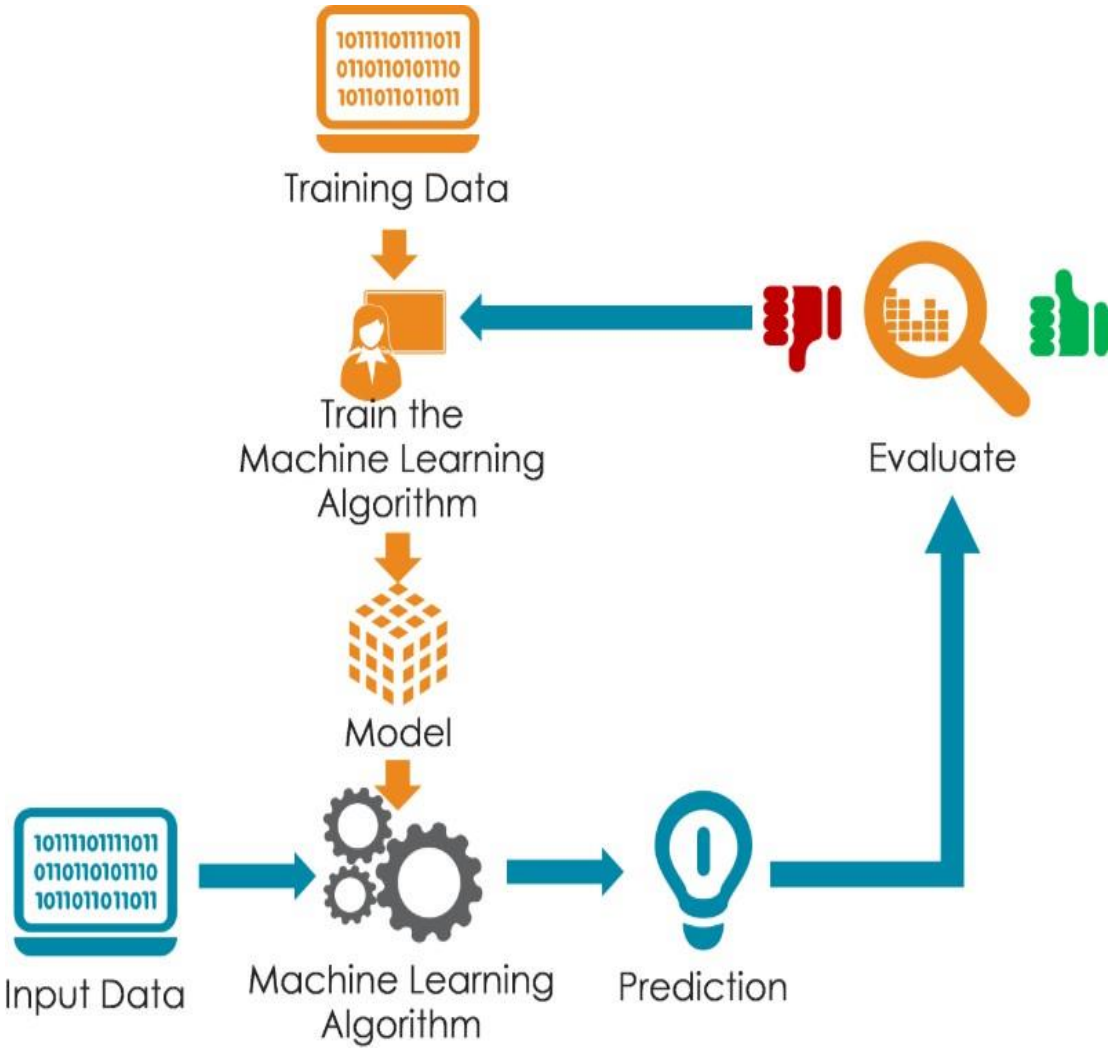


Workflow Overview



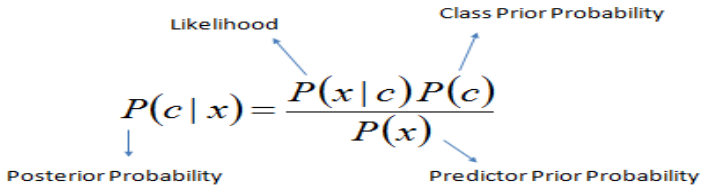
Models used:

- Naïve Bayes
- Support Vector Machine (SVM)
- Neural Network
- Long Short-term memory



Naive Bayes

In order to get a baseline accuracy rate for our data, we implemented a Naive Bayes classifier. Specifically, we used the scikit-learn implementation of Gaussian Naive Bayes. This is one of the simplest approaches to classification, in which a probabilistic approach is used, with the assumption that all features are conditionally independent given the class label. As with the other models, we used the Doc2Vec embeddings described above. The Naive Bayes Rule is based on the Bayes' theorem



$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

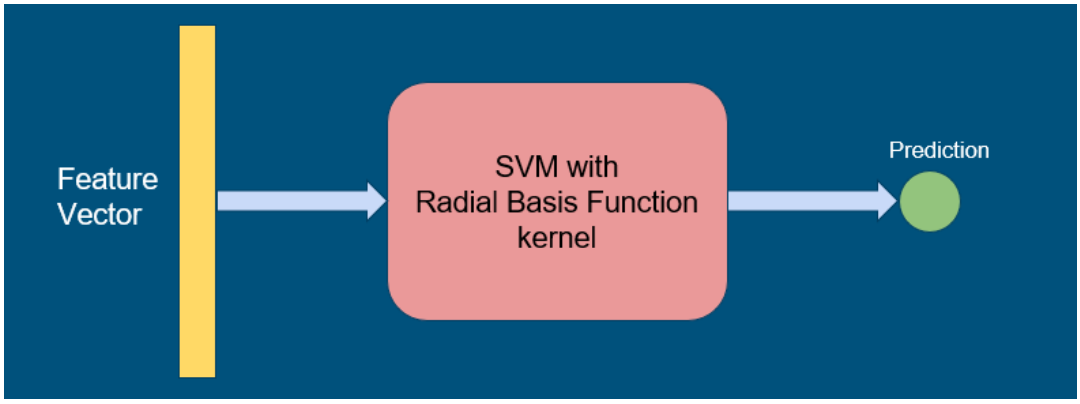
Parameter estimation for naive Bayes models uses the method of maximum likelihood. The advantage here is that it requires only a small amount of training data to estimate the parameters.

CODE:

```python
from getEmbeddings import getEmbeddings
from sklearn.naive_bayes import GaussianNB
import numpy as np
import matplotlib.pyplot as plt
import scikitplot.plotters as skplt


def plot_cmat(yte, ypred):
    '''Plotting confusion matrix'''
    skplt.plot_confusion_matrix(yte,ypred)
    plt.show()


xtr,xte,ytr,yte = getEmbeddings("datasets/train.csv")
np.save('./xtr', xtr)
np.save('./xte', xte)
np.save('./ytr', ytr)
np.save('./yte', yte)

xtr = np.load('./xtr.npy')
xte = np.load('./xte.npy')
ytr = np.load('./ytr.npy')
yte = np.load('./yte.npy')

gnb = GaussianNB()
gnb.fit(xtr,ytr)
y_pred = gnb.predict(xte)
m = yte.shape[0]
n = (yte != y_pred).sum()
print("Accuracy = " + format((m-n)/m*100, '.2f') + "%")   # 72.94%

plot_cmat(yte, y_pred)

```

Support Vector Machine

The original Support Vector Machine (SVM) was proposed by Vladimir N. Vapnik and Alexey Ya. Chervonenkis in 1963. But that model can only do linear classification so it doesn't suit for most of the practical problems. Later in 1992, Bernhard E. Boser, Isabelle M. Guyon and Vladimir N. Vapnik introduced the kernel trick which enables the SVM for non-linear classification. That makes the SVM much powerful.
The objective of the support vector machine algorithm is to find a hyperplane in an N dimensional space (N — the number of features) that distinctly classifies the data points. To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e. the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.



CODE:

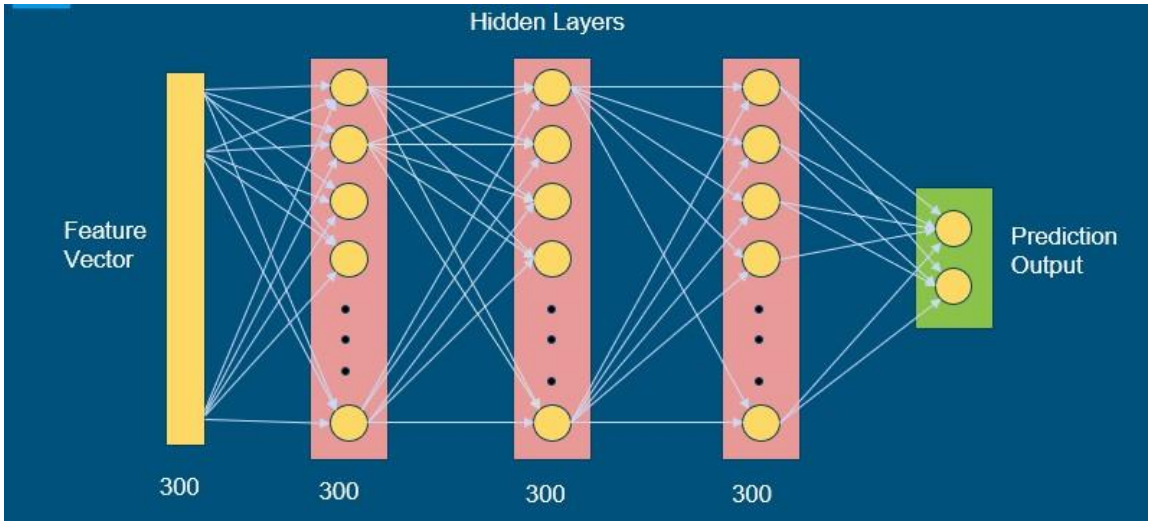
```python
from getEmbeddings import getEmbeddings
import numpy as np
from sklearn.svm import SVC
import matplotlib.pyplot as plt
import scikitplot.plotters as skplt


def plot_cmat(yte, ypred):
    '''Plotting confusion matrix'''
    skplt.plot_confusion_matrix(yte,ypred)
    plt.show()


xtr,xte,ytr,yte = getEmbeddings("datasets/train.csv")
np.save('./xtr', xtr)
np.save('./xte', xte)
np.save('./ytr', ytr)
np.save('./yte', yte)

xtr = np.load('./xtr.npy')
xte = np.load('./xte.npy')
ytr = np.load('./ytr.npy')
yte = np.load('./yte.npy')

clf = SVC()
clf.fit(xtr, ytr)
y_pred = clf.predict(xte)
m = yte.shape[0]
n = (yte != y_pred).sum()
print("Accuracy = " + format((m-n)/m*100, '.2f') + "%")   # 88.42%

plot_cmat(yte, y_pred)
```

Neural Network Using Keras

We implemented the feed-forward neural network model using Keras. Neural networks are commonly used in modern NLP applications, 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 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.
This has a fixed-size input $x \in \mathbf{R}^{\mathbf{1}\times\mathbf{300}}$

Keras is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, R, Theano, or PlaidML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible.

- Hidden Layer Structure : (256, 256, 80)
- Learning rate : 0.01
- Training Steps : 10000



CODE:

```
from getEmbeddings import getEmbeddings
import matplotlib.pyplot as plt
import numpy as np
import keras
from keras import backend as K
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense, Dropout, LSTM, Embedding, Input, RepeatVector
from keras.optimizers import SGD
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
import scikitplot.plotters as skplt


def plot_cmat(yte, ypred):
    '''Plotting confusion matrix'''
    skplt.plot_confusion_matrix(yte, ypred)
    plt.show()


xtr,xte,ytr,yte = getEmbeddings("datasets/train.csv")
np.save('./xtr', xtr)
np.save('./xte', xte)
np.save('./ytr', ytr)
np.save('./yte', yte)

xtr = np.load('./xtr.npy')
xte = np.load('./xte.npy')
ytr = np.load('./ytr.npy')
yte = np.load('./yte.npy')


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

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!")
score = model.evaluate(x_test, encoded_y_test)
print("")
print("Accuracy = " + format(score[1]*100, '.2f') + "%")   # 92.69%

probabs = model.predict_proba(x_test)
y_pred = np.argmax(probabs, axis=1)
plot_cmat(y_test, y_pred)
```

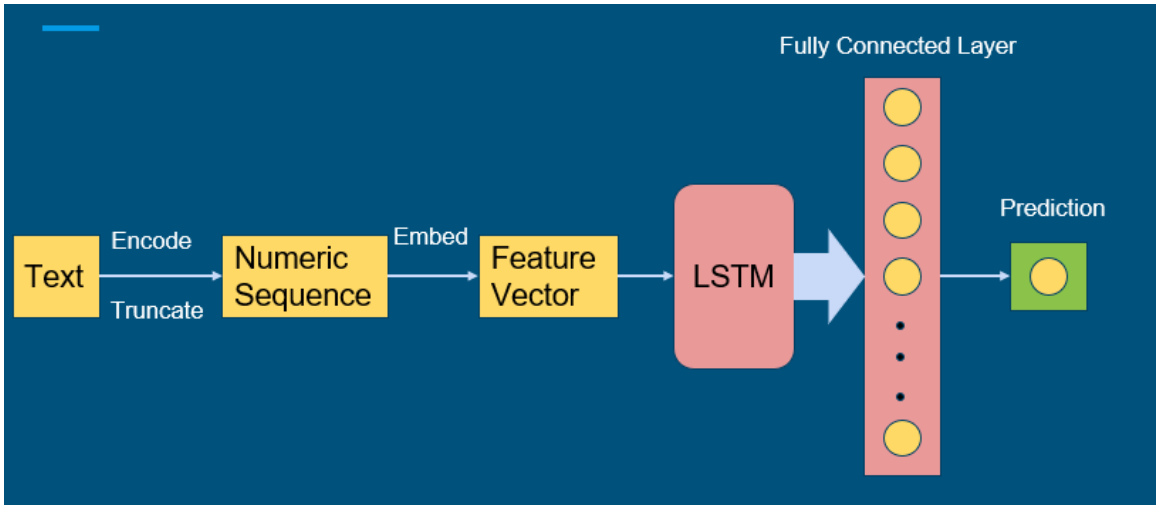
Long Short-Term Memory

The Long-Short Term Memory (LSTM) unit was proposed by Hochreiter and Schmidhuber. 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 prediction. 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 for our problem.

Since the order of the words is important for the LSTM unit, we cannot use the Doc2Vec for pre- processing 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 not letters nor numbers. Then we count the frequency of each word appeared in our training dataset to find 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, 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. 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 the fully connected neural network, it has cycle in the neuron connections. So the previous state (or memory) of the LSTM unit ct will play a role in new prediction ht.



CODE:

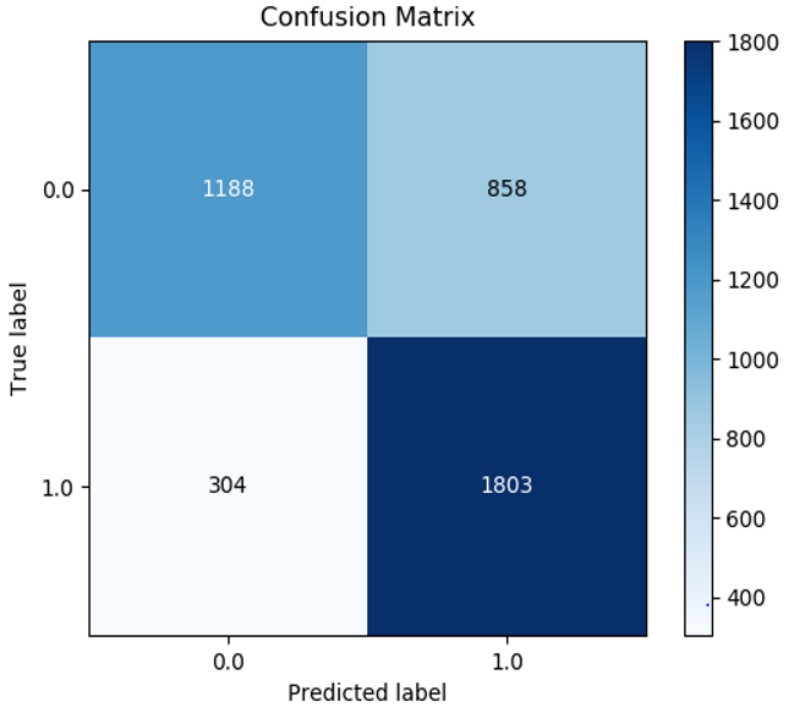
```
import numpy as np
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
from collections import Counter
import os
import getEmbeddings2
import matplotlib.pyplot as plt
import scikitplot.plotters as skplt

top_words = 5000
epoch_num = 5
batch_size = 64

def plot_cmat(yte, ypred):
    '''Plotting confusion matrix'''
    skplt.plot_confusion_matrix(yte, ypred)
    plt.show()

if not os.path.isfile('./xtr_shuffled.npy') or \
    not os.path.isfile('./xte_shuffled.npy') or \
    not os.path.isfile('./ytr_shuffled.npy') or \
    not os.path.isfile('./yte_shuffled.npy'):
    getEmbeddings2.clean_data()

xtr = np.load('./xtr_shuffled.npy')
xte = np.load('./xte_shuffled.npy')
y_train = np.load('./ytr_shuffled.npy')
y_test = np.load('./yte_shuffled.npy')

cnt = Counter()
x_train = []
```

```
for x in xtr:
    x_train.append(x.split())
    for word in x_train[-1]:
        cnt[word] += 1

# 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]

```

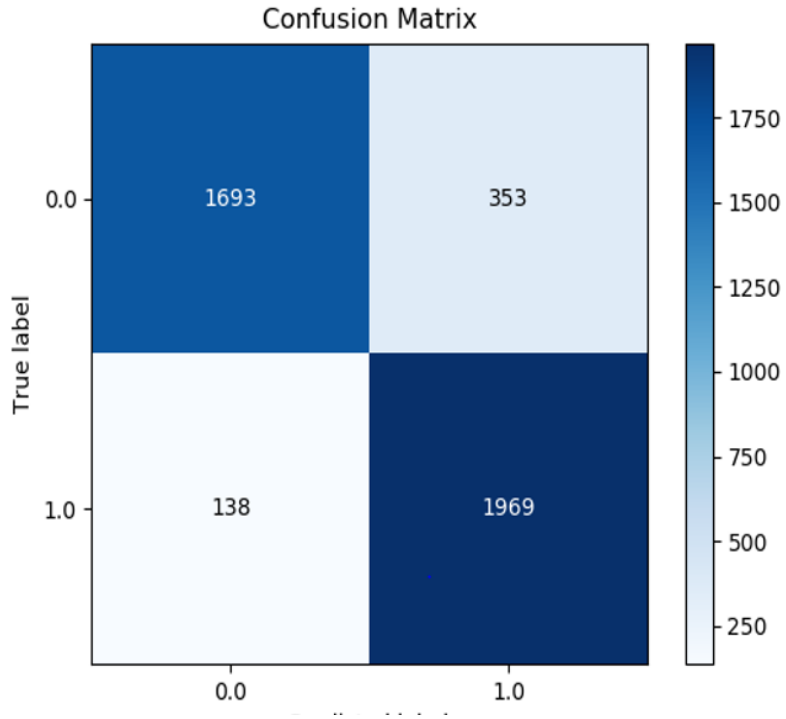
```
# Generating test data
x_test = []
for x in xte:
    x_test.append(x.split())
# Encode the sentences
for news in x_test:
    i = 0
    while i < len(news):
        if news[i] in word_bank:
            news[i] = word_bank[news[i]]
            i += 1
        else:
            del news[i]
# Truncate 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)
# Convert to numpy arrays
y_train = np.array(y_train)
y_test = np.array(y_test)

# Create the model
embedding_vecor_length = 32
model = Sequential()
model.add(Embedding(top_words+2, embedding_vecor_length, input_length=max_review_length))
model.add(LSTM(100))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epoch_num, batch_size=batch_size)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy= %.2f%%" % (scores[1]*100))
y_pred = model.predict_classes(X_test)# Draw the confusion matrix
plot_cmat(y_test, y_pred)
```

Output

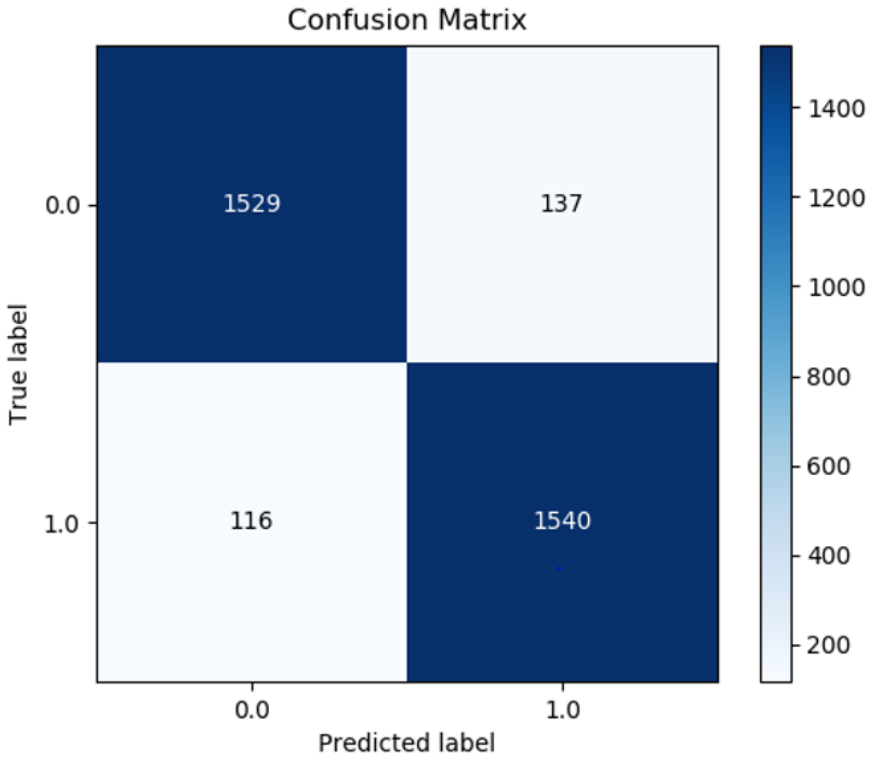
Naïve Baiyes



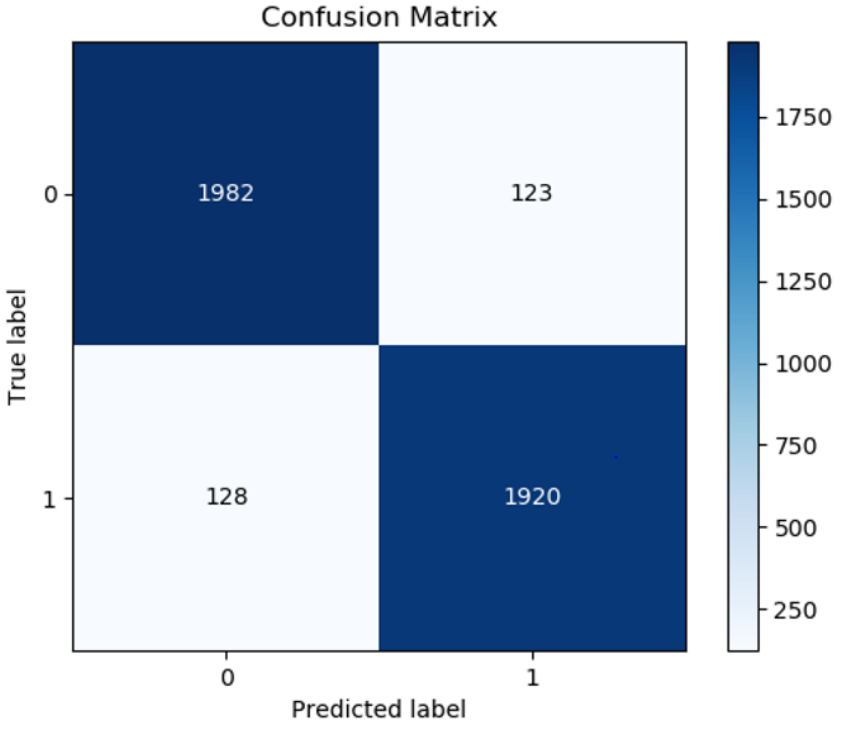
Support Vector Machine



Neural Network using Keras



Long Short-Term Memory



Comparison of Results

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

Name	precision	recall	F1	Accuracy
Naïve Bayes	0.68	0.86	0.76	72.94%
SVM	0.85	0.93	0.89	88.42%
Neural Network Using Keras	0.92	0.93	0.92	92.62%
LSTM	0.94	0.94	0.94	94.53%

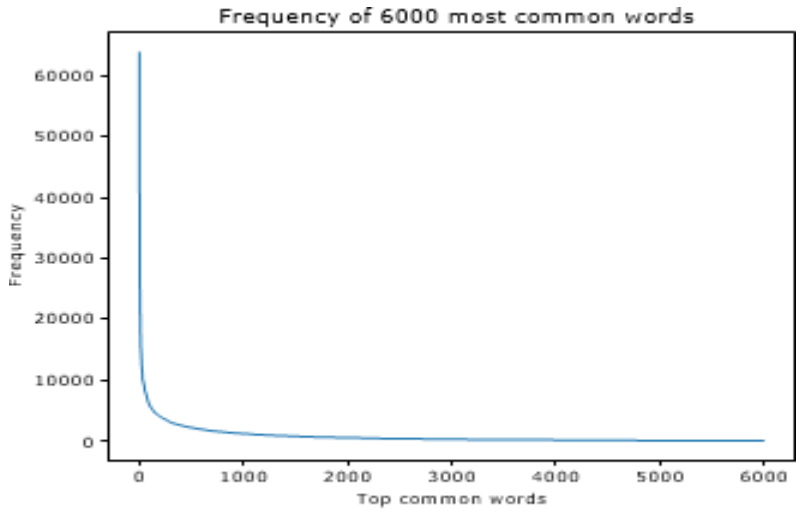


Figure 1: Frequency of Top Common Words

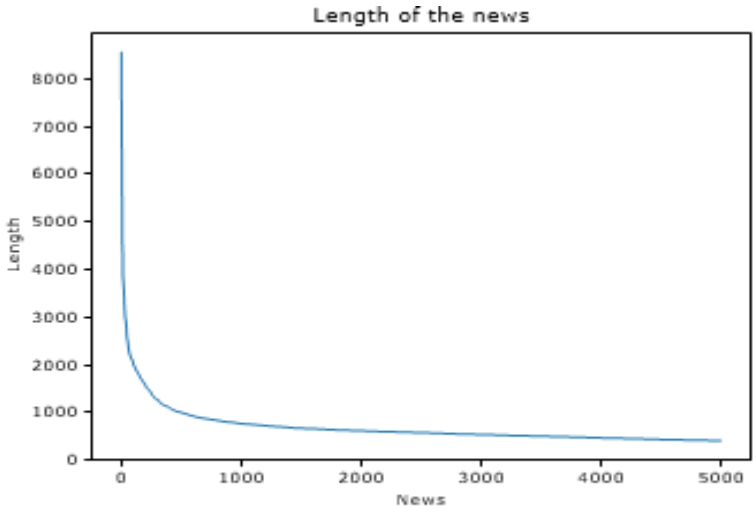


Figure 2: Length of the News

We observed that LSTM gave us the best results. We had to use a different set of embeddings for pre- processing the data to be fed to our LSTM model. It uses 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 a serialized object. 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 classification on it. On the other hand, the LSTM model preserves the order using a different pre-processing method and makes prediction based on both the words and their order. This is how it outperforms others.

Future Work

A complete, production-quality classifier will incorporate many different features beyond the vectors corresponding to the 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 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 ex- ample, using bootstrap aggregating for the Neural Network, LSTM and SVM models to get better prediction result.
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. We will have to compare the meaning of the two contents and decide whether they mean the same thing.

Acknowledgment

We would like to express our special thanks of gratitude to our teacher Prof. Bagubali A for giving us the golden opportunity of working on this wonderful project, which also helped us in doing a lot of research and we came to know and learn about so many new things and therefore we are really thankful to you.

Secondly, we would like to thank VIT for giving us the platform and resources for indulging and successfully completing our project.
And last but not the least we as a team of three members are thankful to each other for showing wonderful team spirit and finalizing this project within the limited time frame and also we would like to thank our parents for believing in us.

References

[1] Zuckerberg M.,*Facebook Post* https://www.facebook.com/zuck/posts/10103253901916271, November, 2016.

[2] Allcott, H., and Gentzkow, M., *Social Media and Fake News in the 2016 Election*, https://web. stanford.edu/œgentzkow/research/fakenews.pdf, January, 2017.

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

[4] Source Code Repository, *GitHub*, https://github.com/FakeNewsDetection/FakeBuster, April, 2018.

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

[6] Christopher, M. Bishop, *Pattern Recognition and Machine Learning*, http://users.isr.ist.utl.pt/~wurmd/Livros/school/Bishop%20%20Pattern%20Recognition%20And%20Machine%20Learning%20-%20Springer%20%202006.pdf, April, 2016.

[7] Goldberg, Y.,*A Primer on Neural Network Models for Natural Language Processing*, https://arxiv.org/pdf/1510.00726.pdf, October, 2015.

[8] Hochreiter, S., Jrgen, S., *Long short-term memory*. http://www.bioinf.jku.at/publications/older/260 4.pdf,