## Data Mining Lab:2019-2023 Batch

**Semester - 6: Final Project: True vs Fake and Cleaning News Detection**

**Name: Dipesh Singla Roll: CO19322**

**Keywords: True and Fake News,Data Mining, Twitter, Google News Vector, word2vec gensim model**

## WHY FAKE NEWS IS A PROBLEM?

**Fake news is defined as misinformation, disinformation, or mal-information that spreads by word of mouth, conventional media, and, more recently, digital modes of communication such as manipulated films, memes, unconfirmed adverts, and social media disseminated rumours. Fake news on social media has become a severe concern, with the potential for it to lead to mob violence, suicides, and other negative outcomes as a result of disinformation propagated on social media.**

**Motivation:** In this ever increasing social world we see lots of news circulating may it be on social media platforms or on the internet across the globe. So there is a need to classify how can one be sure whether the news is true i.e. is genuine and can be trusted upon from our existing Machine Learning Models and Deep Learning Algorithms. We have also worked on a reasearch of how various social networking sites likeTwitter make their algorithm work to get to know the actual facts about a particular news which helped us carrying this idea forward.

**Use Case:** To visualize, classify, predict and compare various models/ prepocessing algorithms to check whether a news is true or fake based on a given dataset.

**Data Sets**

1. https://www.kaggle.com/datasets/clmentbisaillon/fake-and-real-news-dataset
2. https://www.kaggle.com/datasets/sandreds/googlenewsvectorsnegative300

## Library Imports

```
import warnings
warnings.filterwarnings('ignore')

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
import re
from wordcloud import WordCloud
```

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Embedding, LSTM, Conv1D,
MaxPool1D
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score
```

## Exploring Fake News

```
fake =
pd.read_csv("/kaggle/input/fake-and-real-news-dataset/Fake.csv")

fake.head()
```

## Counting by Subjects

```
for key,count in fake.subject.value_counts().iteritems():
    print(f"{key}:\t{count}")

#Getting Total Rows
print(f"Total Records:\t{fake.shape[0]}")

plt.figure(figsize=(8,5))
sns.countplot("subject", data=fake)
plt.show()
```

## Word Cloud

```
text = ''
for news in fake.text.values:
    text += f" {news}"
wordcloud = WordCloud(
    width = 3000,
    height = 2000,
    background_color = 'black',
    stopwords =
set(nltk.corpus.stopwords.words("english"))).generate(text)
fig = plt.figure(
    figsize = (40, 30),
    facecolor = 'k',
    edgecolor = 'k')
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
del text
```

## Exploring Real/ True News

```
real =
pd.read_csv("/kaggle/input/fake-and-real-news-dataset/True.csv")
real.head()
```

### Difference in Text

Real news seems to have source of publication which is not present in fake news set

Looking at the data:

- most of text contains reuters information such as "**WASHINGTON (Reuters)**".
- Some text are tweets from Twitter
- Few text do not contain any publication info

## Cleaning Data

Removing Reuters or Twitter Tweet information from the text

- Text can be splitted only once at " - " which is always present after mentioning source of publication, this gives us publication part and text part
- If we do not get text part, this means publication details was't given for that record
- The Twitter tweets always have same source, a long text of max 259 characters

## Creating list of index that do not have publication part

```
unknown_publishers = []
for index,row in enumerate(real.text.values):
    try:
        record = row.split(" -", maxsplit=1)
        #if no text part is present, following will give error
        record[1]
        #if len of piblication part is greater than 260
        #following will give error, ensuring no text having "-" in
between is counted
        assert(len(record[0]) < 260)
    except:
        unknown_publishers.append(index)
```

## List of indices where publisher is not mentioned

```
real.iloc[unknown_publishers].text
#true, they do not have text like "WASHINGTON (Reuters)"
```

While looking at texts that do not contain publication info such as which reuter, we noticed one thing.

**Text at index 8970 is empty**

```
real.iloc[8970]
```

## Seperating Publication info, from actual text

```
publisher = []
tmp_text = []
for index,row in enumerate(real.text.values):
    if index in unknown_publishers:
        #Add unknown of publisher not mentioned
        tmp_text.append(row)

        publisher.append("Unknown")
        continue
    record = row.split(" -", maxsplit=1)
    publisher.append(record[0])
    tmp_text.append(record[1])
```

## Replace existing text column with new text

Add seperate column for publication info

```
real["publisher"] = publisher
real["text"] = tmp_text

del publisher, tmp_text, record, unknown_publishers

real.head()
```

New column called "Publisher" has been added.

## Checking for rows with empty text like row:8970

```
[index for index,text in enumerate(real.text.values) if
str(text).strip() == '']

#dropping this record
real = real.drop(8970, axis=0)
```

## Checking for the same in fake news

```
empty_fake_index = [index for index,text in
enumerate(fake.text.values) if str(text).strip() == '']
print(f"No of empty rows: {len(empty_fake_index)}")
fake.iloc[empty_fake_index].tail()
```

**630 Rows in Fake news with empty text**

I've also observed a lot of CPATIAL-CASES in bogus news. Could keep letter cases, however we'll be utilising Google's pretrained word2vec vectors later on, which have well-formed lower case words. We will try to use lower case.

The text for these rows appears to be included in the title. Let's combine the title and text to solve these problems.

## Getting Total Rows

```
print(f"Total Records:\t{real.shape[0]}")

# Counting by Subjects
for key,count in real.subject.value_counts().iteritems():
  print(f"{key}:\t{count}")

sns.countplot(x="subject", data=real)
plt.show()
```

## WordCloud For Real News

```
text = ''
for news in real.text.values:
    text += f" {news}"
wordcloud = WordCloud(
    width = 3000,
    height = 2000,
    background_color = 'black',
    stopwords =
set(nltk.corpus.stopwords.words("english"))).generate(str(text))
fig = plt.figure(
    figsize = (40, 30),
    facecolor = 'k',
    edgecolor = 'k')
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis('off')
plt.tight_layout(pad=0)
plt.show()
del text
```

## Preprocessing Text

## Adding class Information

```
real["class"] = 1
fake["class"] = 0
```

## Combining Title and Text

```
real["text"] = real["title"] + " " + real["text"]
fake["text"] = fake["title"] + " " + fake["text"]
```

## Subject is Diffrent for real and fake => dropping

```
real = real.drop(["subject", "date","title",  "publisher"], axis=1)
fake = fake.drop(["subject", "date", "title"], axis=1)
```

## Combining both into new dataframe

```
data = real.append(fake, ignore_index=True)
del real, fake
```

Removing StopWords, Punctuations and single-character words

## Converting X to format acceptable by gensim, removing annd punctuation stopwords in the process

```
y = data["class"].values
X = []
stop_words = set(nltk.corpus.stopwords.words("english"))
tokenizer = nltk.tokenize.RegexpTokenizer(r'\w+')
for par in data["text"].values:
    tmp = []
    sentences = nltk.sent_tokenize(par)
    for sent in sentences:
        sent = sent.lower()
        tokens = tokenizer.tokenize(sent)
        filtered_words = [w.strip() for w in tokens if w not in
stop_words and len(w) > 1]
        tmp.extend(filtered_words)
    X.append(tmp)

del data
```

### Vectorization -- Word2Vec

Word2Vec is one of the most popular technique to learn word embeddings using shallow neural network. It was developed by Tomas Mikolov in 2013 at Google.

Word embedding is the most popular representation of document vocabulary. It is capable of capturing context of a word in a document, semantic and syntactic similarity, relation with other words, etc.

*Let's create and check our own Word2Vec model with **gensim***

```
import gensim
```

## Dimension of vectors we are generating

```
EMBEDDING_DIM = 100

#Creating Word Vectors by Word2Vec Method
w2v_model = gensim.models.Word2Vec(sentences=X, size=EMBEDDING_DIM,
window=5, min_count=1)
```

## Vocab Size

```
len(w2v_model.wv.vocab)

# Represented each of 122248 words by a 100dim vector.
```

### Exploring Vectors

## See a sample vector for random word, say Corona

```
w2v_model["corona"]

w2v_model.wv.most_similar("iran")

w2v_model.wv.most_similar("fbi")

w2v_model.wv.most_similar("facebook")

w2v_model.wv.most_similar("computer")
```

## Feeding US Presidents

```
w2v_model.wv.most_similar(positive=["trump","obama", "clinton"])
#First was Bush
```

**Looking at the similar words, vectors are well formed for these words :)**

These Vectors will be passed to LSTM/GRU instead of words. 1D-CNN can further be used to extract features from the vectors.

Keras has implementation called "**Embedding Layer**" which would create word embeddings(vectors). Since we did that with gensim's word2vec, we will load these vectors into embedding layer and make the layer non-trainable.

Tokenizer can represent each word by number. Since we cannot pass string words to embedding layer, thus need some way to represent each words by numbers.

## Tokenizing Text -> Repsesenting each word by a number

## Mapping of orginal word to number is preserved in word_index property of tokenizer

## Tokenized applies basic processing like changing it yo lower case, explicitely setting that as False

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(X)

X = tokenizer.texts_to_sequences(X)
```

## Checking the first 10 words of first news # every word has been represented with a number

```
X[0][:10]
```

## Mapping is preserved in dictionary -> word_index property of instance

```
word_index = tokenizer.word_index
for word, num in word_index.items():
    print(f"{word} -> {num}")
    if num == 10:
        break
```

**Notice it starts with 1**

We can pass numerical representation of words into neural network.

We can use Many-To-One (Sequence-To-Word) Model of RNN, as we have many words in news as input and one output ie Probability of being Real.

For Many-To-One model, lets use a fixed size input.

## Histogram for no of words in news shows that most news article are under 700 words.

## Keeping each news small and truncate all news to 700 while tokenizing

```
plt.hist([len(x) for x in X], bins=500)
plt.show()
```

```
# Its heavily skewed.
# Truncate these outliers

nos = np.array([len(x) for x in X])
len(nos[nos  < 700])
# Out of 48k news, 44k have less than 700 words
```

## Keep all news to 700, add padding to news with less than 700 words and truncating long ones

```
maxlen = 700

#Making all news of size maxlen defined above
X = pad_sequences(X, maxlen=maxlen)
```

#all news has 700 words (in numerical form now). If they had less words, they have been padded with 0

```
# 0 is not associated to any word, as mapping of words started from 1
# 0 will also be used later, if unknows word is encountered in test
set
len(X[0])

# Adding 1 because of reserved 0 index
# Embedding Layer creates one more vector for "UNKNOWN" words, or
padded words (0s). This Vector is filled with zeros.
# Thus our vocab size inceeases by 1
vocab_size = len(tokenizer.word_index) + 1
```

## Function to create weight matrix from word2vec gensim model

```
def get_weight_matrix(model, vocab):
    # total vocabulary size plus 0 for unknown words
    vocab_size = len(vocab) + 1
    # define weight matrix dimensions with all 0
    weight_matrix = np.zeros((vocab_size, EMBEDDING_DIM))
    # step vocab, store vectors using the Tokenizer's integer mapping
    for word, i in vocab.items():
        weight_matrix[i] = model[word]
    return weight_matrix
```

We Create a matrix of mapping between word-index and vectors. We use this as weights in embedding layer

Embedding layer accepts numecical-token of word and outputs corresponding vercor to inner layer.

It sends vector of zeros to next layer for unknown words which would be tokenized to 0.

Input length of Embedding Layer is the length of each news (700 now due to padding and truncating)

## Getting embedding vectors from word2vec and usings it as weights of non-trainable keras embedding layer

```
embedding_vectors = get_weight_matrix(w2v_model, word_index)
```

## Defining Neural Network

```
model = Sequential()
#Non-trainable embeddidng layer
model.add(Embedding(vocab_size, output_dim=EMBEDDING_DIM,
weights=[embedding_vectors], input_length=maxlen, trainable=False))
#LSTM
model.add(LSTM(units=128))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['acc'])

del embedding_vectors

model.summary()
```

#Train test split

```
X_train, X_test, y_train, y_test = train_test_split(X, y)

model.fit(X_train, y_train, validation_split=0.3, epochs=6)
```

## Prediction is in probability of news being real, so converting into classes

```
# Class 0 (Fake) if predicted prob < 0.5, else class 1 (Real)
y_pred = (model.predict(X_test) >= 0.5).astype("int")

accuracy_score(y_test, y_pred)

print(classification_report(y_test, y_pred))

del model
```

### Using Pre-Trained Word2Vec Vectors

**Requirements RAM: 12GB and HardDisk Space: 4GB**

## Invoke garbage collector to free ram

```
import gc
gc.collect()
```

```
from gensim.models.keyedvectors import KeyedVectors
```

```
# Requires RAM
word_vectors =
KeyedVectors.load_word2vec_format('../input/googlenewsvectorsnegative3
00/GoogleNews-vectors-negative300.bin', binary=True)
EMBEDDING_DIM=300
```

### Exploring these trained Vectors

```
embedding_matrix = np.zeros((vocab_size, EMBEDDING_DIM))
for word, i in word_index.items():
    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

model = Sequential()
model.add(Embedding(vocab_size, output_dim=EMBEDDING_DIM,
weights=[embedding_matrix], input_length=maxlen, trainable=False))
model.add(Conv1D(activation='relu', filters=4, kernel_size=4))
model.add(MaxPool1D())
model.add(LSTM(units=128))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['acc'])

del embedding_matrix

model.summary()

model.fit(X_train, y_train, validation_split=0.3, epochs=12)

y_pred = (model.predict(X_test) > 0.5).astype("int")

accuracy_score(y_test, y_pred)

print(classification_report(y_test, y_pred))
```

**Final Results:** Along with comparison from other top models Based on the chosen model's accuracy.

1. By selecting lbfgs as the solver and increasing the maximum number of iterations to 200, we can get an accuracy of 90.77 percent on the testing data for Logistic Regression with spaCy vectors as input features.
2. With spaCy vectors as input features, the decision tree classifier achieves an accuracy of 85.51 percent.
3. Logistic Regression with textual data as an input feature and Count Vectorizer and TFIDF Transformer in pipeline achieves 98.93% accuracy.
4. A Decision Tree Classifier using textual data as an input feature and Count Vectorizer and TFIDF Transformer in the pipeline achieves 99.63 percent accuracy.
5. Using 1000 input features, an artificial neural network constructed with keras and tensorflow backend with the Sequential Model and dense layers achieves an accuracy of 99.28 percent.
6. An Artificial Neural Network built using Keras and Tensorflow backends with the Sequential Model and dense layers achieves an accuracy of 99.21 percent on testing data with 10000 input features. Based on the F1-scores of the selected model, we can calculate the number of false positives and false negatives. Overall, we get an accuracy of 98.93%.
7. By selecting lbfgs as the solver and increasing the maximum number of iterations to 200, we can obtain a f1-score of 90.42 percent on the testing data for Logistic Regression with spaCy vectors as input features.
8. A decision tree classifier with spaCy vectors as input features achieves a f1-score of 84.49 percent.
9. Logistic Regression with textual data as an input feature and Count Vectorizer and TFIDF Transformer in the pipeline yields a f1 of 98.81%.
10. A Decision Tree Classifier using textual data as an input feature and Count Vectorizer and TFIDF Transformer in the pipeline produces a f1-score of 99.61 percent.
11. Keras with Tensorflow backend Artificial Neural Network with Sequential Model and Dense Layers supply us with a f1-score of 99.25 percent on the testing data with 1000 input features.
12. An artificial neural network built using Keras and Tensorflow backends with the Sequential Model and dense layers achieves a f1-score of 99.17 percent on testing data with 10000 input features. Also, an F1 score comparable to that of accuracy demonstrates how accurate our model is in forecasting.

values by learning whether it is real or false news via model training

**Future Work:**

1. Gathering specific news-related tweets.
2. Using the same Decision Tree Classifier and pipeline, we will forecast whether or not this is false news.
3. Another suggestion is to utilise the same for the credibility score assignment of a certain tweet, which is slightly different from this.

**Conclusions:** Using Pre-Trained Word2Vec Vectors in a pipeline with we've achieved maximum accuracy, recall and F1 score of about 99%, but there is always room for improvement. Various things, such as tweaking the hyperparameters, can be done in the future to improve the accuracy and f1 score based on another dataset. Use of tensors may enhance the overall results.