Fake News Detection

Sam Earnest Alex Johnson

Introduction and Motivation

A common complaint made by many followers (including many politicians) of contemporary 21st-century U.S. politics is that of "fake" news, that a news article or that a news source itself is deliberately producing and divulging news that is knowingly counterfactual. Seeing the supposed frequency of this problem, we, the authors of this project, decided to create a machine learning algorithm that would be able to differentiate between news considered to be true against news considered to be fake. The project that follows is our attempt at the differentiation of interest, highlighted by a Random Forest classification, as well as a principal component analysis (PCA) and center-base clustering method (k-means).

To accomplish this purpose, we obtained a dataset that contains news articles classified as both true and false between the years 2015 and 2017. The dataset itself is very likely valid since all the data are merely news articles and their accompanying titles. In addition, the dataset was created by an academic source, thus likely eliminating any bias in the classification of news as true or fake. We note that this dataset is ideally suited to the demands of our project since it contains the classifications of true and false that we are trying to achieve.

We note that this work is done mostly independent of other research that has been done on the matter. We are aware of the existence of other machine learning algorithms that fulfill similar purposes to that of our own, but we looked to develop a unique method that we created. We in no way look to infringe or trespass on their intellectual property or copy their methods. We do borrow the code for the kernel function from another author.

```
import pandas as pd
import numpy as np
import random
import nltk
# nltk.download()
import re
# nltk.download('stopwords')
# nltk.download('punkt')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from sklearn.model_selection import train_test_split
from sklearn.cluster import KMeans
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestRegressor
from sklearn.decomposition import PCA
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import datasets
import matplotlib.pyplot as plt
```

Data Preparation and Cleaning

The data for this project came from kaggle.com, published by Clément Bisaillon for open source use, and is a collection of fake and true news articles from 2016 to 2018. In this section, we clean our data by removing stop words, non-english words, and limit the ammount of words to the top 500 most frequent. We remove stop words and non-English words because they will not be relevant in determining if an article contains true or fake news. We limit ourselves to the 500 most fequent words to aid in computational time and feel that 500 is a good threshold of relevance.

```
#Upload datasets and create true column
df_fake = pd.read_csv('Fake.csv')
df_true = pd.read_csv('True.csv')
df_fake['true'] = 0
df_true['true'] = 1
df = pd.concat([df_fake, df_true])
df
```

	title	text	subject	date	true
0	Donald Trump Sends Out Embarrassing New Year'...	Donald Trump just couldn t wish all Americans ...	News	December 31, 2017	0
1	Drunk Bragging Trump Staffer Started Russian ...	House Intelligence Committee Chairman Devin Nu...	News	December 31, 2017	0
2	Sheriff David Clarke Becomes An Internet Joke...	On Friday, it was revealed that former Milwauk...	News	December 30, 2017	0
3	Trump Is So Obsessed He Even Has Obama's Name...	On Christmas day, Donald Trump announced that ...	News	December 29, 2017	0
4	Pope Francis Just Called Out Donald Trump Dur...	Pope Francis used his annual Christmas Day mes...	News	December 25, 2017	0
...	...	...	...	...	...
21412	'Fully committed' NATO backs new U.S. approach...	BRUSSELS (Reuters) - NATO allies on Tuesday we...	worldnews	August 22, 2017	1
21413	LexisNexis withdrew two products from Chinese ...	LONDON (Reuters) - LexisNexis, a provider of l...	worldnews	August 22, 2017	1
21414	Minsk cultural hub becomes haven from authorities	MINSK (Reuters) - In the shadow of disused Sov...	worldnews	August 22, 2017	1
21415	Vatican upbeat on possibility of Pope Francis ...	MOSCOW (Reuters) - Vatican Secretary of State ...	worldnews	August 22, 2017	1
21416	Indonesia to buy $1.14 billion worth of Russia...	JAKARTA (Reuters) - Indonesia will buy 11 Sukh...	worldnews	August 22, 2017	1

44898 rows × 5 columns

First we need to identify all of the potential words in out data set. We throw out all stop words, numbers, and non-alphabetic charaters. The top 500 most frequent words become the columns for out dataframe. This is done using regular expressions

```
wrds = {}
stop_words = set(stopwords.words('english'))
#Set up tokenizer to only grab ASCII characters
tokenizer = nltk.RegexpTokenizer(r"\w+")

for index, row in df.iterrows():
    text = row['text'].lower()
    #Filter for stop words
    text = re.sub(r'\d+', '', text)
    word_tokens = tokenizer.tokenize(text)
    filtered_text = [w for w in word_tokens if not w in stop_words]

    #Go through each sig word in text and add frequency to dictionary of words
    for word in filtered_text:
        if '_' in word:
            continue
        if word in wrds:
            wrds[word] = wrds[word] + 1
        else:
            wrds[word] = 1

wrds = sorted(wrds, key=wrds.get, reverse=True)
```

```
#Create columns using the 500 top words a column names
master_df = pd.DataFrame(0, index= df.index, columns = wrds[:500])
master_df
```

	trump	said	president	would	u	people	one	state	also	new	reuters	clinton	obama	donald	government	house	states
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
21412	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21413	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21414	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21415	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21416	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

44898 rows × 500 columns

Now we add the frequency of each word to its associated document.

```
for index, row in df.iterrows():
    text = row['text'].lower()
    #Filter for stop words
    text = re.sub(r'\d+', '', text)
    word_tokens = tokenizer.tokenize(text)
    filtered_text = [w for w in word_tokens if not w in stop_words]

    for word in filtered_text:
        if word in wrds[:500]:
            master_df.loc[index, word] = master_df.loc[index, word] + 1
```

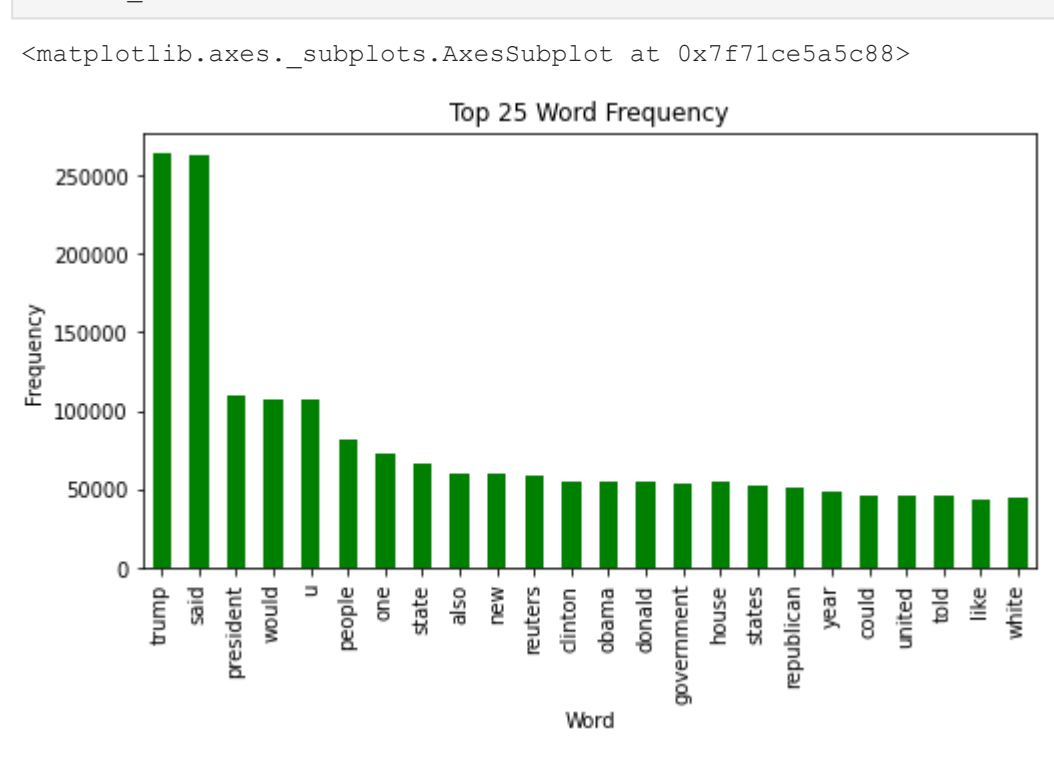
```
#Save data frame to avoid computational time
master_df = pd.read_csv('500matrix.csv', index_col=0)
#Move truth column to be the first column
truth = master_df.pop('true')
master_df.insert(0,'true',truth)

df_true = master_df[master_df['true']==1]
df_false = master_df[master_df['true']==0]
master_df
```

	true	trump	said	president	would	u	people	one	state	also	new	reuters	clinton	obama	donald	government	house
0	0	16	8	5	5	4	4	3	0	4	11	1	0	0	4	2	6
1	0	16	12	6	6	2	2	2	0	0	2	1	2	4	1	1	1
2	0	15	8	2	1	1	4	1	0	0	2	1	1	0	1	0	2
3	0	21	3	3	4	2	2	0	0	3	3	2	1	6	4	0	3
4	0	10	12	5	1	4	1	1	0	2	0	1	0	0	2	3	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
44893	1	6	10	5	3	9	1	0	1	1	3	1	0	20	1	2	2
44894	1	0	3	1	2	0	6	4	0	2	0	1	0	2	0	1	0
44895	1	0	5	1	0	0	0	1	3	0	1	1	0	0	0	1	0
44896	1	0	4	0	0	0	3	2	1	3	1	1	0	0	0	0	0
44897	1	0	5	0	0	1	0	0	2	1	1	1	0	0	0	0	0

44898 rows × 501 columns

```
master_df.sum(axis=0)[1:25].plot.bar(figsize=(8,4), title='Top 25 Word Frequency', ylabel='Frequency', xlabe
```

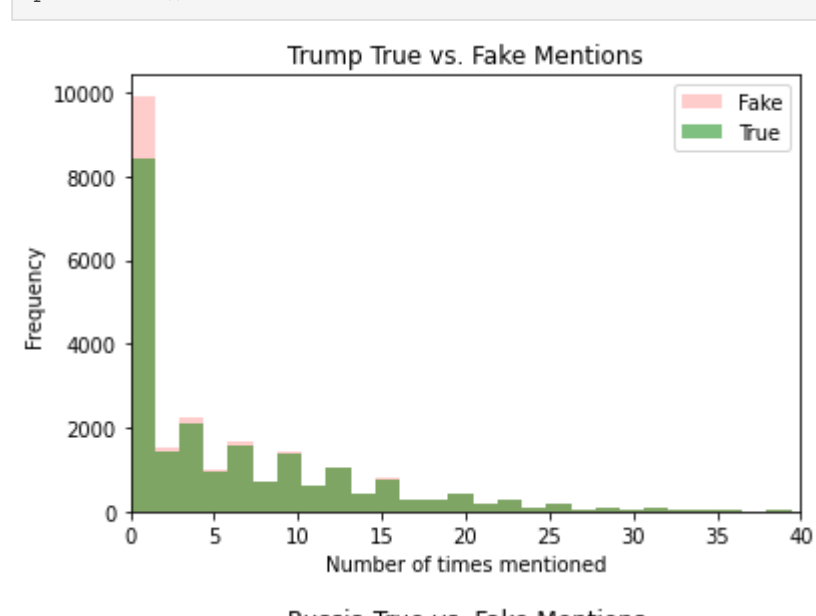
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f71ce5a5c88>
```

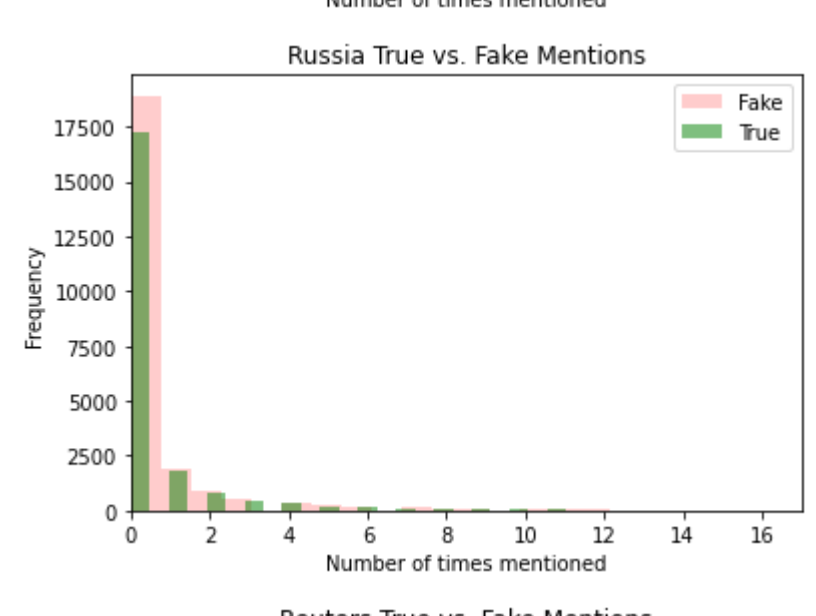


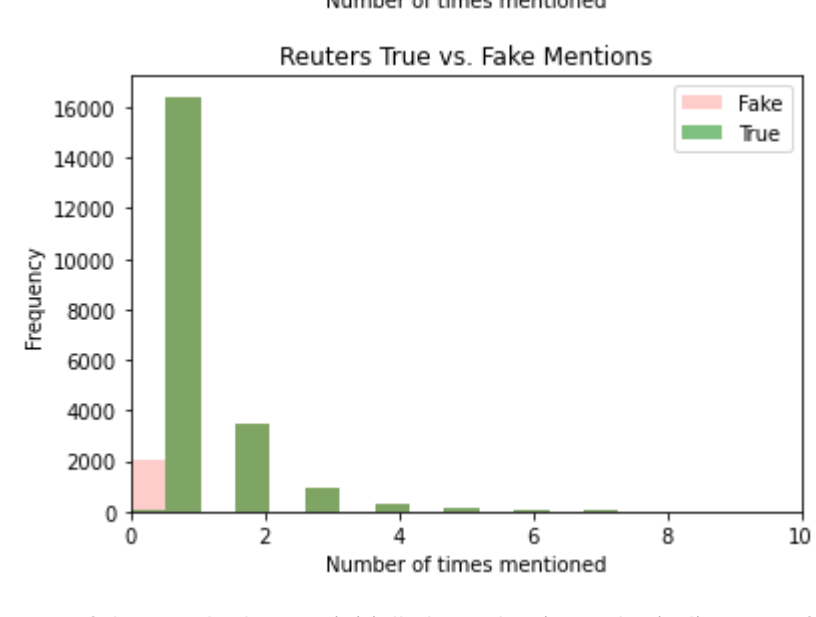
As can be seen from the bar chart the word 'trump' is by far the most frequent word in our set. Other common words include 'said', 'president', and 'would'. This being said, we note that words such as 'said' and 'would' may not be very telling of fake or true news as they are very common in the English vernacular. We note interestingly words like 'clinton', 'reuters', and 'government' also have high frequencies. In our analysis, therefore, we may purposefully choose to omit futher observation into the more ordinary words (such as 'would' or 'said'), since we hypothesize no telling findings from these words. However, there appear to be many other words of marked interest that have higher frequencies that will be the subject of further study in the analysis that follows.

Immediately, we include several histograms of a few of these words that emit higher levels of interest.

```
#Histogram 1: Trump
plt.hist(master_df[master_df["true"]==0]["trump"],alpha=0.2,bins=50,color="red",label="Fake")
plt.hist(master_df[master_df["true"]==1]["trump"],alpha=0.5,bins=50,color="green",label="True")
plt.title("Trump True vs. Fake Mentions")
plt.xlabel("Number of times mentioned")
plt.ylabel("Frequency")
plt.xlim(0,40)
plt.legend()
plt.show()

#Histogram 2: Russia
plt.hist(master_df[master_df["true"]==0]["russia"],alpha=0.2,bins=50,color="red",label="Fake")
plt.hist(master_df[master_df["true"]==1]["russia"],alpha=0.5,bins=50,color="green",label="True")
plt.title("Russia True vs. Fake Mentions")
plt.xlabel("Number of times mentioned")
plt.ylabel("Frequency")
plt.xlim(0,17)
plt.legend()
plt.show()

#Histogram 3: Obama
plt.hist(master_df[master_df["true"]==0]["reuters"],alpha=0.2,bins=50,color="red",label="Fake")
plt.hist(master_df[master_df["true"]==1]["reuters"],alpha=0.5,bins=50,color="green",label="True")
plt.title("Reuters True vs. Fake Mentions")
plt.xlabel("Number of times mentioned")
plt.ylabel("Frequency")
plt.xlim(0,10)
plt.legend()
plt.show()
```







Two of the words that we initially hypothesize to be indicators of fake news are the words "Trump" and "Russia" from the controversies that these two political figures were involved in in the years 2015-2017 (the year from which our data comes from). To check for this, we plot the overlayed histograms of fake news against true news for "Trump" and "Russia". Notice that for both "Trump" and "Russia", the frequency of number of mentions between 0 and 5 is high for both fake and true news, which seems to confirm our early interest. Moreover, in the same number of mentions category, it is evident that fake mentions for both "Trump" and "Russia" have a higher frequency than their true counterparts, showing that these words are perhaps strong indicators of fake news.

We then compare these two words to the word "Reuters" (a international news outlet based in Germany), which we hypothesize to be a control against "Trump" and "Russia". Upon first observation of the similarly overlayed histogram, it is clear that "Reuters" is indeed a strong control, since there are very few fake news mentions of this word, compared to many true news mentions (which are likely the articles put out by the the company in interest).

These observations from the visualizations are impressively telling and are consistent with our expectations on the matter. We run more tests to further solidify these expectations in what follows.

Now we format the data as y and X and split between train and test sets

```
data = master_df.to_numpy()
y = data[:,0]
X = data[:,1:]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.98)
print(X_train.shape)
```

```
(897, 500)
```

Feature Engineering (Kernel Function)

Now we define our kerneling method. We chose to use the document similarity kernel as a opposed to the cosine similarity because we're dealing with documents and this kernel will give better results.

```
def tf(x):
    return np.log(1+x)

def idf(j):
    return np.log(j.size/(1+np.count_nonzero(j)))

def phi(x, idf):
    result = np.zeros(len(idf))
    for i in range(0,len(idf)):
        result[i] = x[i]*idf[i]
    return result

def doc_sim(M):
    #Apply tf to each element in M
    M = np.vectorize(tf)(M)
    #Create the inverse document frequency array- one value for each word
    idf_v = np.zeros(M.shape[1])
    for j in range (0,M.shape[1]):
        idf_v[j] = idf(M[:,j])

    #Create kernel matrix and fill it
    K = np.zeros((M.shape[0], M.shape[0]))
    for i in range(0, K.shape[0]):
        #print(i)
        for j in range(0, K.shape[1]):
            phi_x = phi(M[i,:], idf_v)
            phi_y = phi(M[j,:], idf_v)
            K[i,j] = (np.transpose(phi_x)@phi_y)/(np.linalg.norm(phi_x)*np.linalg.norm(phi_y))

    return K
```

```
K = doc_sim(X_train)
```

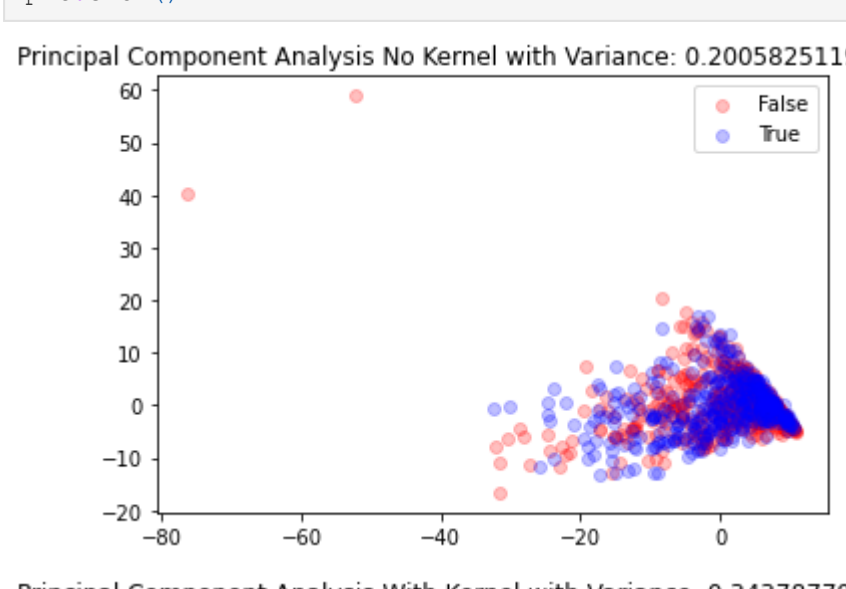
```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:35: RuntimeWarning: invalid value encountered i
n double_scalars
```

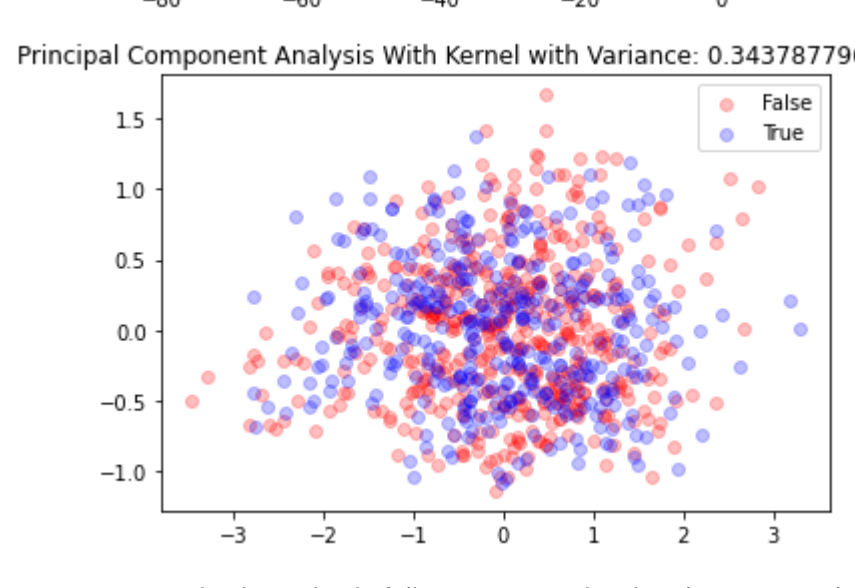
```
np.save('test5.txt', K)
```

```
k = np.load("test4.txt.npy")
l = np.argwhere(np.isnan(k))[0,1]
k[:,l] = 0.1
k[l,:] = 0.1
```

Principal Component Analysis

```
#First with no kernel
X = X_train
y = y_train
pca = PCA(n_components=5)
Xhat_1 = pca.fit_transform(X)

var = pca.explained_variance_ratio_
var = var[0]+var[1]

f = Xhat_1[y_train==0]
t = Xhat_1[y_train==1]
f_plt = plt.scatter(f[:,0]*-1, f[:,1], color = 'red', alpha=0.25)
t_plt = plt.scatter(t[:,0]*-1, t[:,1], color = 'blue',alpha=0.25)
plt.legend([f_plt, t_plt], ['False', 'True'])
plt.title("Principal Component Analysis No Kernel with Variance: " + str(var) )
plt.show()

#Now with kernel
X = k
y = y_train
pca = PCA(n_components=5)
Xhat = pca.fit_transform(X)

var = pca.explained_variance_ratio_
var = var[0]+var[1]

f = Xhat[y_train==0]
t = Xhat[y_train==1]
f_plt = plt.scatter(f[:,0]*-1, f[:,1], color = 'red', alpha=0.25)
t_plt = plt.scatter(t[:,0]*-1, t[:,1], color = 'blue',alpha=0.25)
plt.legend([f_plt, t_plt], ['False', 'True'])
plt.title("Principal Component Analysis With Kernel with Variance: " + str(var) )
plt.show()
```





As you can see both methods fail to seperate the data in any meaningful way. This being said, the PCA on the kernel data is certainly more spread out and slightly better than that on the non-kernel data, as is apparent in their variances. We now move onto a Random Forest classification to see if that method allows us to more accurately classify our data.

Random Forest Classification

Here we use the Random Forest method as a prediction tool to classify between true and fake news. Notice that we run two separate Random Forest classifications, which we do to find the "best" (or most predictive) 100 features.

```
rf = RandomForestClassifier(criterion="entropy")
rf.fit(Xhat_1,y_train)
# rf.predict(X_train,y_train)

param_grid = {"n_estimators": [100,200,300], "max_depth": [10,25,50,75], "min_samples_leaf": [10,5,2]}
rf_gs = GridSearchCV(rf, param_grid, scoring="f1", n_jobs=-1)
rf_gs.fit(Xhat_1,y_train)
print(rf_gs.best_params_, rf_gs.best_score_, sep='\n')
```

```
{'max_depth': 25, 'min_samples_leaf': 2, 'n_estimators': 100}
0.4114346911493734
```

```
result = RandomForestClassifier(oob_score=True, max_depth=25, n_estimators=100, min_samples_leaf=2).fit(X_tr
importances = result.feature_importances_

# print(importances)
```

```
importances1 = importances.argsort()[-100:][::-1]
print(importances[10])
print(importances1)
importances1_list = importances1.tolist()
```

```
0.012111894375653355
[ 10   1   2   0   3   5  36   7  39  31   6  15   4 177  24 387  20  38
  19   9  23  12  56  34  21 109  13  28  57  81 434  84  30  67  29  25
 432  85  42 194  61  45   8  73  17  50  14  16  18 118  53 159  40  26
 178 172 107 114 111 128 110 245  79 117  37 264  80 201  35 392  55 391
 116  66 129  27 268 130  22 127  74  33  89  32  62 101  76  91  44 357
 333 295 120  95 236  86  11 124 137  47]
```

After having found the most predictive 100 features, we now create a new dataframe (referred to as new_df in the code) that is solely based on these 100 features. Once this dataframe has been created, we then apply the doc_sim function once again to create the kernel matrix from the new training data. Lastly, we are be able to run the Random Forest classification again. Our analysis will follow.

```
new_df = master_df.iloc[:,importances1_list]
col_name="true"
first_col = new_df.pop(col_name)
new_df.insert(0, col_name, first_col)

data1 = new_df.to_numpy()
y = data1[:,0]
X = data1[:,1:]

rfX_train, rfX_test, rfy_train, rfy_test = train_test_split(X, y, test_size=0.98)
print(rfX_train.shape)
pca = PCA(n_components=99)
rfXhat_1 = pca.fit_transform(rfX_train)
```

```
(897, 99)
```

```
rf1 = RandomForestClassifier(criterion="entropy")
rf1.fit(rfXhat_1,rfy_train)

param_grid = {"n_estimators": [100,200,300], "max_depth": [10,25,50,75], "min_samples_leaf": [10,5,2]}
rf1_gs = GridSearchCV(rf1, param_grid, n_jobs=-1)
rf1_gs.fit(rfXhat_1,rfy_train)
print(rf1_gs.best_params_, rf1_gs.best_score_, sep='\n')
```

```
{'max_depth': 75, 'min_samples_leaf': 5, 'n_estimators': 300}
0.5418063314711359
```

```
result = rf1_gs.best_estimator_.predict(rfX_test)
CM = confusion_matrix(rfy_test,result)
print(CM)
print('\n')
CR = classification_report(rfy_test, result)
print(CR)
```

```
[[12959 10031]
 [11272  9739]]


              precision    recall  f1-score   support

           0       0.53      0.56      0.55     22990
           1       0.49      0.46      0.48     21011

    accuracy                           0.52     44001
   macro avg       0.51      0.51      0.51     44001
weighted avg       0.51      0.52      0.51     44001
```

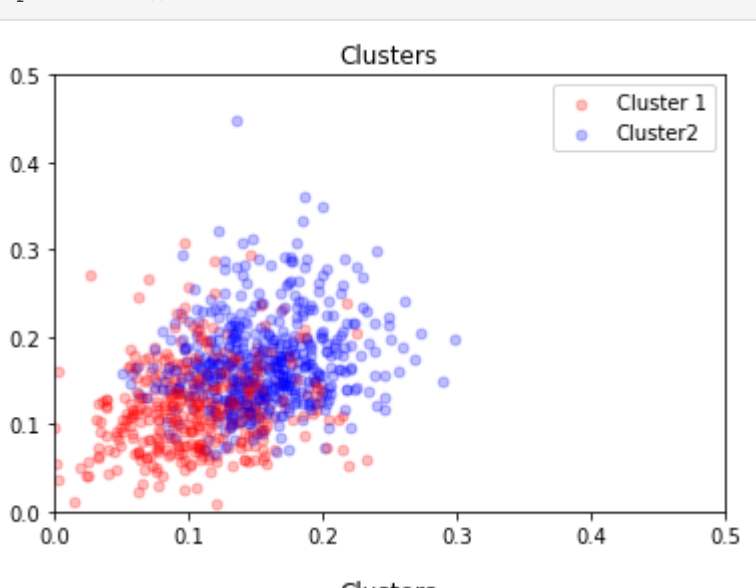
As can be seen, our Random Forest classification approach does not necessarily yield the accuracy that was initially desired. In other words, this prediction method does not classify news as true or fake with a very high accuracy, with the best score being approximately 52%.

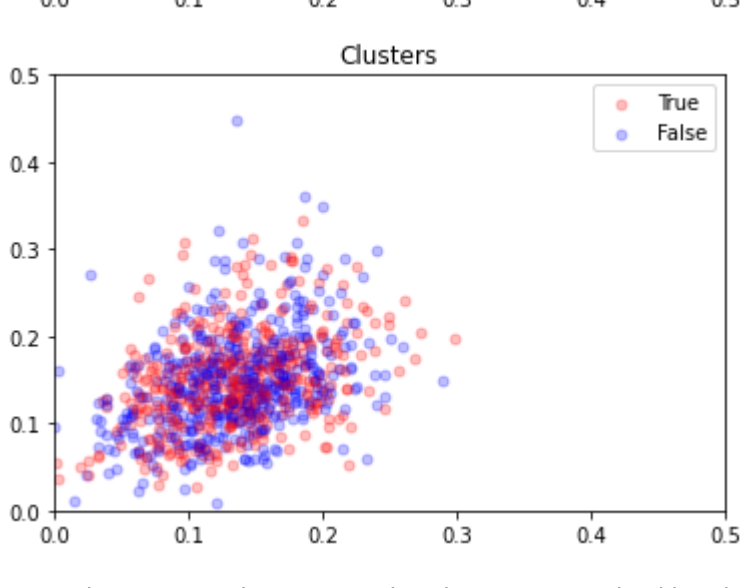
With this being said, it is entirely possible that Random Forest classification was not the ideal prediction method for a problem of this type. It may be more useful in future versions of projects of this type to run a pipeline to help us decide which prediction method and which hyperparameters would be the best to pursue. From this, we now move on to a k-means clustering to see if the method is more effective at classifying news as true or fake.

Clustering and Visualization (K-means)

```
kmeans = KMeans(n_clusters=2,init="random",algorithm="full", random_state=random.randint(0,100))
```

```
#Actual clustering
x_1 = kmeans.fit_predict(k)

fig, ax = plt.subplots()
for i in range(0, len(x_1)):
    if x_1[i] == 1:
      color = 'red'
      label = 'True'
    else:
      color = 'blue'
      label = 'False'
    ax.scatter(k[i,0], k[i,1], c = color, s = 25,alpha=0.25)
leg = ax.legend(['Cluster 1', 'Cluster2'])
leg.legendHandles[0].set_color('red')
leg.legendHandles[1].set_color('blue')
plt.title("Clusters")
plt.xlim(0,0.5)
plt.ylim(0,0.5)

plt.show()
#kmeans.cluster_centers_

x_1 = kmeans.fit_predict(k)

fig, ax = plt.subplots()
for i in range(0, len(x_1)):
    if  y_train[i] == 1:
      color = 'red'
      label = 'True'
    else:
      color = 'blue'
      label = 'False'
    ax.scatter(k[i,0], k[i,1], c = color, s = 25,alpha=0.25)
leg = ax.legend(['True', 'False'])
leg.legendHandles[0].set_color('red')
leg.legendHandles[1].set_color('blue')
plt.title("Clusters")
plt.xlim(0,0.5)
plt.ylim(0,0.5)

plt.show()
```





As can be seen on the scatter plot above, our method has led to the development of two somewhat unique clusters from our training set. After K has been formatted correctly, we then use the KMeans function to develop our clusters. We then use the scatterplot function to visualize the data as seen.

We note that the two clusters are somewhat unique and the clustering is by no means perfect. In the visualization, the blue-colored dots represent news that is being classified as fake, whereas the red-colored dots represent news that is being classified as true. Please observe that there are several locations of overlap between the two colors, indicating that the clustering method is not perfect, as specified before. However, this does provide a good representation of the classification we were seeking and is thus useful for our analysis.

We conclude that k-means clusering is, though imperfect, the best method for seperating fake news from true news. In a quick note on ethics, if this fake news detection project were to actually be used it would be important to continuously retrain the model and update it's features. News is something that can change drastically from year to year and judging fake news today based off 2016's

new cycle won't be accurate. If a false sense of trust was put in the algorithm, and it wasn't updated frequently, it could potentially prevent individuals from reading true and important information or feed them false information.