Ankara Yıldırım Beyazıt University

Nazir Ahmad ZAHIRI

205101116

Ankara yıldırım Beyazıt University

nazirahmadzahiri93@gmail.com

Supervisor

Doç. Dr. SHAFQAT UR REHMAN

shafqat.rehman@gmail.com

Fake News Detection Using Machine Learning

Objective:

The major objective is to detect the fake news, which is a typical text classification problem with a consecutive further proposition. It is necessary to develop a prototype that can differentiate between “Real” news and “Fake” news.

INTRODUCTION

Fake news straightforward meaning is to integrate information that directs people to the wrong path. These Days fake news is expanding like water and people share this information with no authenticating it. This is frequently done to advance or require specific proposals and is frequently completed with diplomatic programs.

For mass media channels, the power to appeal audiences to their websites is required to produce online marketing proceeds. So, it is essential to detect fake news.

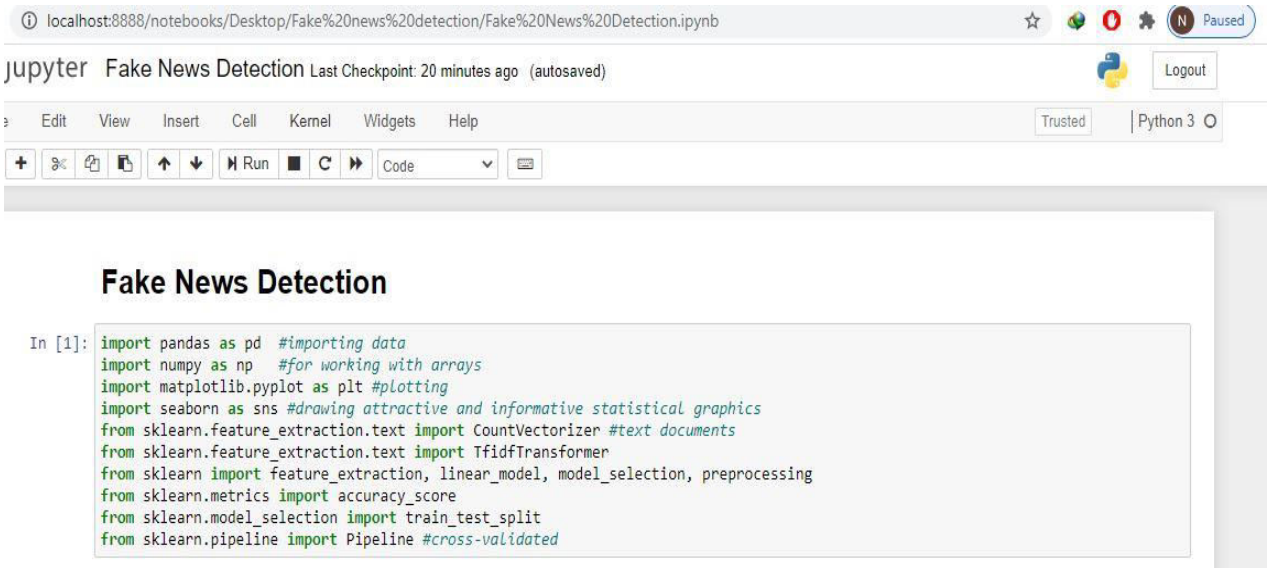
These days' fake news is creating different problems from sardonic articles to a false news and plan government misinformation in some markets. Fake news and absence of trust in the media are growing problems with huge consequences in our society. A purposely confusing story is "fake news "but lately babbling social media's conversation is altering its description. Some of them now use the term to dismiss the facts respond to their favorite viewpoints.

The significance of misinformation within electoral conversation was the subject of weighty interest, especially following the chief election. The term 'fake news' became common phrasing for the problem, especially to explain objectively wrong and deceptive articles printed mostly for the objective of making money through page views. In this project, it is necessary to create a model that can correctly predict the possibility that a given article is fake news.

Facebook has been at the attraction of much criticism following media interest. They have already executed a feature to banner fake news on the site when a user sees' it; they have also said openly they are working on to differentiate these articles in an automatic way. It is not an easy-going task. A given algorithm must be politically impartial – since fake news exists on both ends of the range – and give equal balance to valid news sources on either end of the range. In addition, the question of validity is a tricky one. However, to solve this problem, it is necessary to have an understanding on top of what Fake News is. Later, it is needed to investigate how the methods in the fields of machine learning, natural language processing helps us to detect fake news. Luckily, there are several computational methods that can be used to mark certain articles as fake based on their textual content. Majority of these techniques use fact checking websites. There are several sources provided by scientists that include lists of websites that are known as confusing and fake. However, the problem with these resources is that human skill is needed to identify articles/websites as fake. More importantly, the fact checking websites contain articles from domains such as politics and are not generalized to identify fake news articles from multiple domains such as entertainment, sports, and technology.

Dataset

Nowadays internet if full of many datasets related to any field you are looking for. There are many datasets for fake news detection on Kaggle or many other sites. I have downloaded these datasets from Kaggle. There are two dataset one for fake news and one for true news. In true news, there is 21417 news, and in fake news, there is 23481 news. Both the datasets have a label column in which 1 represent as fake news and 0 represent as true news. I have combined both datasets using pandas built in function.

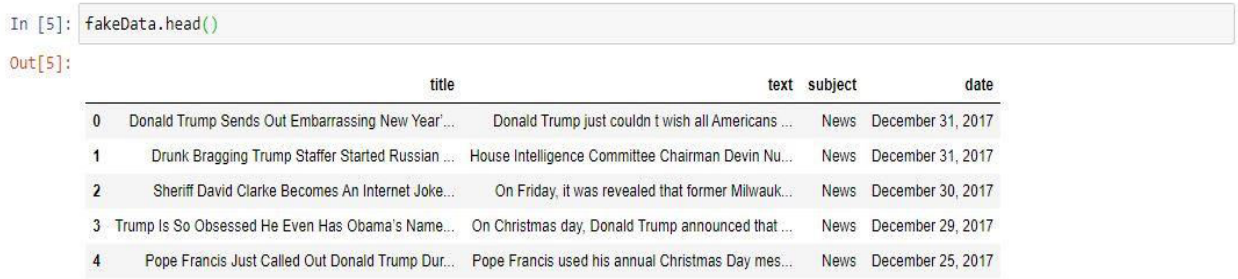


Important libraries

Reading both true and fake data with pandas read.csv function

	title	text	subject	date
0	As U.S. budget fight looms, Republicans flip t...	WASHINGTON (Reuters) - The head of a conservat...	politicsNews	December 31, 2017
1	U.S. military to accept transgender recruits o...	WASHINGTON (Reuters) - Transgender people will...	politicsNews	December 29, 2017
2	Senior U.S. Republican senator: 'Let Mr. Muell...	WASHINGTON (Reuters) - The special counsel inv...	politicsNews	December 31, 2017
3	FBI Russia probe helped by Australian diplomat...	WASHINGTON (Reuters) - Trump campaign adviser ...	politicsNews	December 30, 2017
4	Trump wants Postal Service to charge 'much mor...	SEATTLE/WASHINGTON (Reuters) - President Donal...	politicsNews	December 29, 2017

Head of true dataset



	title	text	subject	date
0	Donald Trump Sends Out Embarrassing New Year'...	Donald Trump just couldn t wish all Americans ...	News	December 31, 2017
1	Drunk Bragging Trump Staffer Started Russian ...	House Intelligence Committee Chairman Devin Nu...	News	December 31, 2017
2	Sheriff David Clarke Becomes An Internet Joke...	On Friday, it was revealed that former Milwauk...	News	December 30, 2017
3	Trump Is So Obsessed He Even Has Obama's Name...	On Christmas day, Donald Trump announced that ...	News	December 29, 2017
4	Pope Francis Just Called Out Donald Trump Dur...	Pope Francis used his annual Christmas Day mes...	News	December 25, 2017

Head of fake dataset

As we can see the dataset includes 21417 true news, and 23481 fake news. In the above Dataset there are no missing values or else we must remove that knowledge, or we must assign some value. Now dataset is balanced because both categories have approximated same no. of examples.

Cleaning and Preparing data

As we know that we cannot use text data directly because it has some unusable words and special symbols and many more things. If we used directly without cleaning, then it is very hard for the ML algorithm to detect patterns in that text and sometimes it will also generate an error. So that we must always first clean text data. In this project, first I am going to be adding flag to track real and fake and after that concatenate dataframes.

Cleaning and Preparing data

In [5]:
```
# Add flag to track real and fake
trueData['target'] = 'true'
fakeData['target'] = 'fake'
```

In [6]:
```
# Concatenate dataframes
data = pd.concat([fakeData, trueData]).reset_index(drop = True)
```

In [7]:
```
data.shape
```

Out[7]: (44898, 5)

In [8]:
```
data.head()
```

Out[8]:

	title	text	subject	date	target
0	Donald Trump Sends Out Embarrassing New Year'...	Donald Trump just couldn t wish all Americans ...	News	December 31, 2017	fake
1	Drunk Bragging Trump Staffer Started Russian ...	House Intelligence Committee Chairman Devin Nu...	News	December 31, 2017	fake
2	Sheriff David Clarke Becomes An Internet Joke...	On Friday, it was revealed that former Milwauk...	News	December 30, 2017	fake
3	Trump Is So Obsessed He Even Has Obama's Name...	On Christmas day, Donald Trump announced that ...	News	December 29, 2017	fake
4	Pope Francis Just Called Out Donald Trump Dur...	Pope Francis used his annual Christmas Day mes...	News	December 25, 2017	fake

Now I am going to Shuffle the data. By shuffling your data, you ensure that each data point creates an "independent" change on the model, without being biased by the same points before them.

In [11]:
```
# Shuffle the data
from sklearn.utils import shuffle
data = shuffle(data)
data = data.reset_index(drop=True)
```

In [12]:
```
data.shape
```

Out[12]: (44898, 5)

In [10]:
```
# Check the data
data.head()
```

Out[10]:

	title	text	subject	date	target
0	This Old Tweet From Trump Reveals The REAL Re...	Donald Trump is not the negotiator he would li...	News	August 11, 2017	fake
1	BREAKING: CLOSE FRIEND Of Bill And Hillary Cli...	The man arrested by Miami Beach police Tuesday...	left-news	Jan 19, 2017	fake
2	Boiler Room EP #77 – The Venom of Divide and Rule	Tune in to the Alternate Current Radio Network...	Middle-east	October 5, 2016	fake
3	SWEET SMELL OF REVENGE: Farmer Sprays Manure O...	GET OFF my fracking land!An irate farmer spray...	left-news	Apr 28, 2016	fake
4	[VIDEO] CNN CO-ANCHOR CHRIS CUOMO'S OBVIOUS OB...	Where s the outcry from the left about the mis...	left-news	Jun 24, 2015	fake

Removing the date (we will not use it for the analysis)

In [11]:
```
#Removing the date (we won't use it for the analysis):
data.drop(["date"],axis=1,inplace=True)
```

In [12]:
```
data.head()
```

Out[12]:

	title	text	subject	target
0	This Old Tweet From Trump Reveals The REAL Re...	Donald Trump is not the negotiator he would li...	News	fake
1	BREAKING: CLOSE FRIEND Of Bill And Hillary Cli...	The man arrested by Miami Beach police Tuesday...	left-news	fake
2	Boiler Room EP #77 – The Venom of Divide and Rule	Tune in to the Alternate Current Radio Network...	Middle-east	fake
3	SWEET SMELL OF REVENGE: Farmer Sprays Manure O...	GET OFF my fracking land!An irate farmer spray...	left-news	fake
4	[VIDEO] CNN CO-ANCHOR CHRIS CUOMO'S OBVIOUS OB...	Where s the outcry from the left about the mis...	left-news	fake

Removing the title (we will only use the text)

In [13]:
```
#Removing the title (we will only use the text):
data.drop(["title"],axis=1,inplace=True)
```

In [14]:
```
data.head()
```

Out[14]:

	text	subject	target
0	Donald Trump is not the negotiator he would li...	News	fake
1	The man arrested by Miami Beach police Tuesday...	left-news	fake
2	Tune in to the Alternate Current Radio Network...	Middle-east	fake
3	GET OFF my fracking land!An irate farmer spray...	left-news	fake
4	Where s the outcry from the left about the mis...	left-news	fake

Convert the text to lowercase and Remove punctuation

In [15]:
```
#Convert the text to lowercase:
data['text'] = data['text'].apply(lambda x: x.lower())
```

In [16]:
```
#Remove punctuation:
import string
def punctuation_removal(text):
    all_list = [char for char in text if char not in string.punctuation]
    clean_str = ''.join(all_list)
    return clean_str
data['text'] = data['text'].apply(punctuation_removal)
```

In [17]:
```
data.head()
```

Out[17]:

	text	subject	target
0	donald trump is not the negotiator he would li...	News	fake
1	the man arrested by miami beach police tuesday...	left-news	fake
2	tune in to the alternate current radio network...	Middle-east	fake
3	get off my fracking landan irate farmer spraye...	left-news	fake
4	where s the outcry from the left about the mis...	left-news	fake

Removing stopwords, stop words are words which are filtered out before or after processing of natural language data.

```
In [18]: # Removing stopwords
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
stop = stopwords.words('english')
data['text'] = data['text'].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop)]))
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\NAZIRAHMADNAZIR\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
In [19]: data.head()
```

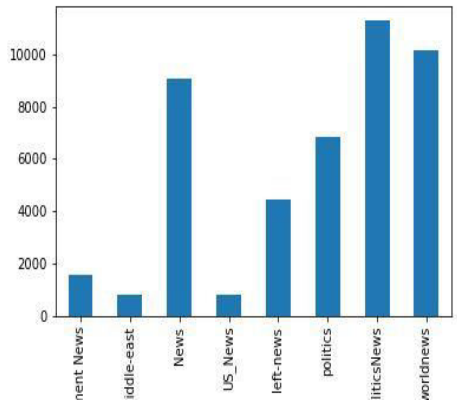
Out[19]:

	text	subject	target
0	donald trump negotiator would like everyone be...	News	fake
1	man arrested miami beach police tuesday allege...	left-news	fake
2	tune alternate current radio network acr anoth...	Middle-east	fake
3	get fracking landan irate farmer sprayed raw s...	left-news	fake
4	outcry left mistreatment relentless badgering ...	left-news	fake

Now I am going to do Basic data exploration. I will use the Pandas library for this. Pandas is the primary tool data scientists use for exploring and manipulating data. Most people abbreviate pandas in their code as pd. We do this with the command the most important part of the Pandas library is the DataFrame. A DataFrame holds the type of data you might think of as a table. First, I will check how many articles per subject?

```
In [13]: # How many articles per subject?
print(data.groupby(['subject'])['text'].count())
data.groupby(['subject'])['text'].count().plot(kind="bar")
plt.show()
```

```
subject
Government News     1570
Middle-east          778
News                9050
US_News              783
left-news           4459
politics            6841
politicsNews       11272
worldnews          10145
Name: text, dtype: int64
```

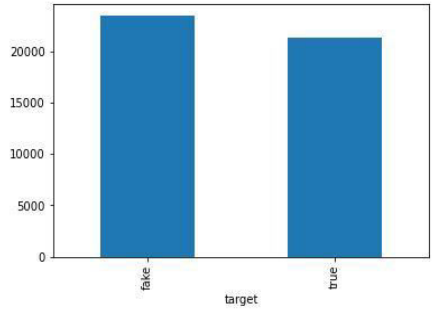


This Show How many Articles to related subjects

Now I am going to check How many fake and real articles are there

```
In [21]: # How many fake and real articles?
         print(data.groupby(['target'])['text'].count())
         data.groupby(['target'])['text'].count().plot(kind="bar")
         plt.show()
```

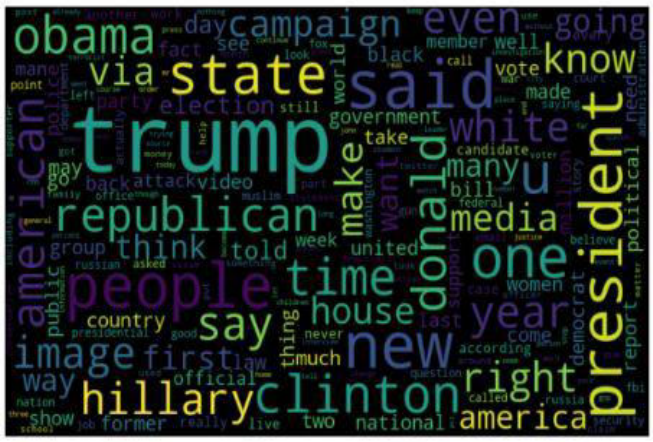
```
target
fake    23481
true    21417
Name: text, dtype: int64
```



Real and Fake articles

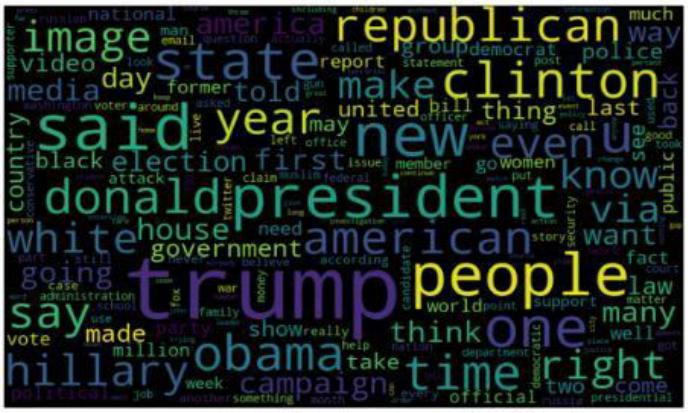
Now I am going to apply WordCloud. A Wordcloud is a graphical representation of text data. It displays a list of words, the importance of each being shown with font size or color. This format is useful for quickly perceiving the most prominent terms.

```
In [22]: # Word cloud for fake news
         from wordcloud import WordCloud

         fake_data = data[data["target"] == "fake"]
         all_words = ' '.join([text for text in fake_data.text])

         wordcloud = WordCloud(width= 800, height= 500,
                               max_font_size = 110,
                               collocations = False).generate(all_words)

         plt.figure(figsize=(10,7))
         plt.imshow(wordcloud, interpolation='bilinear')
         plt.axis("off")
         plt.show()
```



WordCloud For Fake News

```
In [23]: # Word cloud for real news
from wordcloud import WordCloud

real_data = data[data["target"] == "true"]
all_words = ' '.join([text for text in fake_data.text])

wordcloud = WordCloud(width= 800, height= 500,
                      max_font_size = 110,
                      collocations = False).generate(all_words)

plt.figure(figsize=(10,7))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```



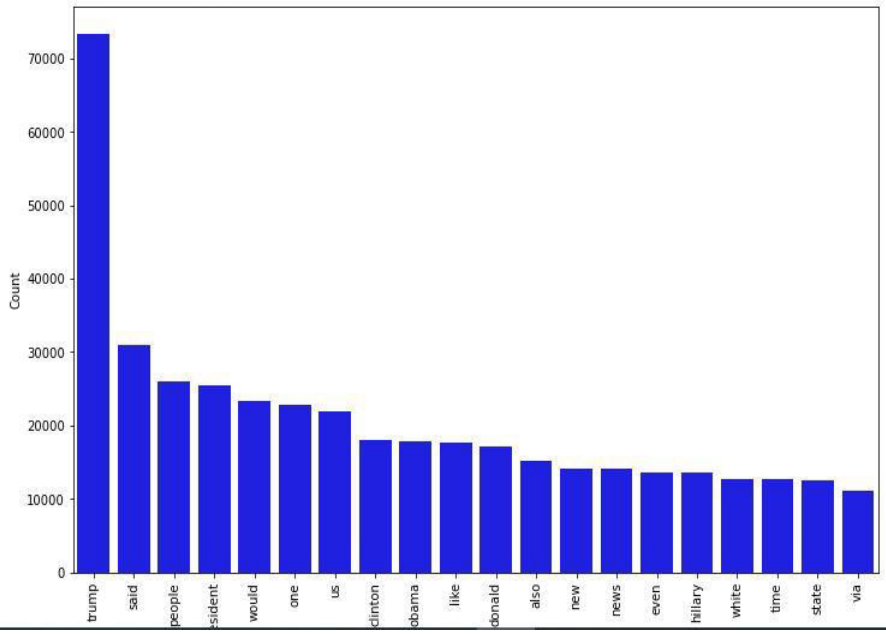
WordCloud For Real News

After all these now we will count most frequent words

```
In [24]: # Most frequent words counter
from nltk import tokenize

token_space = tokenize.WhitespaceTokenizer()

def counter(text, column_text, quantity):
    all_words = ' '.join([text for text in text[column_text]])
    token_phrase = token_space.tokenize(all_words)
    frequency = nltk.FreqDist(token_phrase)
    df_frequency = pd.DataFrame({"Word": list(frequency.keys()),
                                 "Frequency": list(frequency.values())})
    df_frequency = df_frequency.nlargest(columns = "Frequency", n = quantity)
    plt.figure(figsize=(12,8))
    ax = sns.barplot(data = df_frequency, x = "Word", y = "Frequency", color = 'blue')
    ax.set(ylabel = "Count")
    plt.xticks(rotation='vertical')
    plt.show()
```

After Counting the most frequent words Now I will show most frequent words in fake news

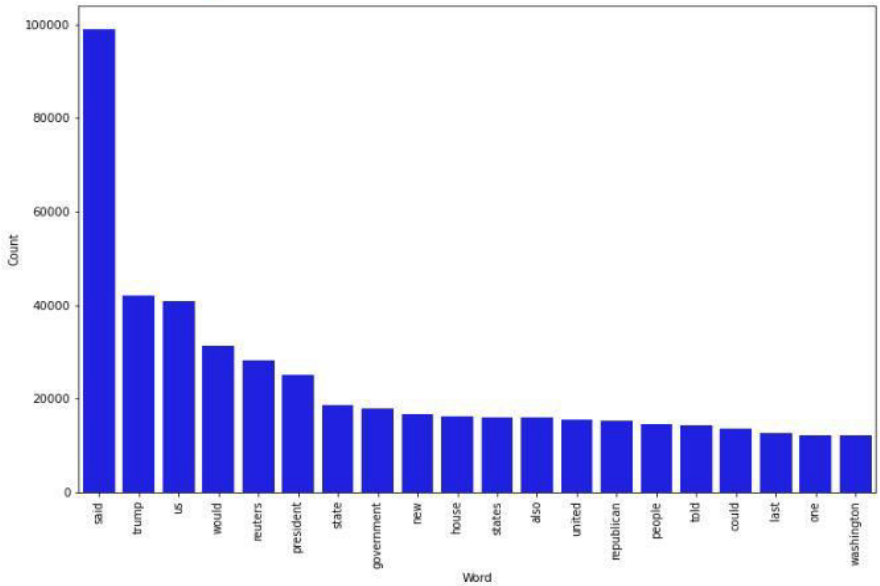
```
In [25]: # Most frequent words in fake news
         counter(data[data["target"] == "fake"], "text", 20)
```



Counting Fake words

After Counting the most frequent words and the most frequent words in fake news now I will count the most frequent words in real news

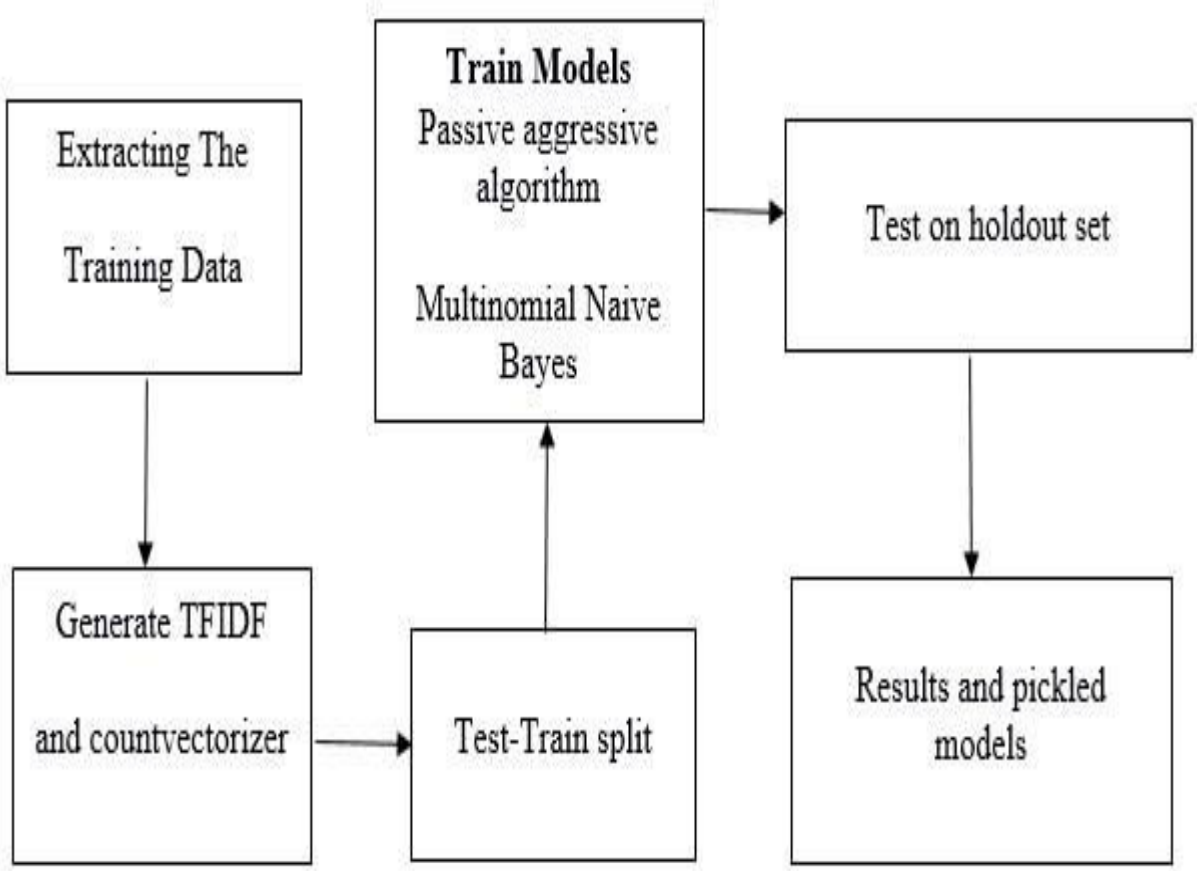
```
In [26]: # Most frequent words in real news
         counter(data[data["target"] == "true"], "text", 20)
```



Counting Real words

PROPOSED SYSTEM

The modeling process will consist of vectorizing the amount stored in the “text” column, then applying TF-IDF, and finally a classification machine learning algorithm. In the following project a model is build based on the count vectorizer or a tfidf matrix word tallies relatives to how often they are used in other articles in your dataset. Since this problem is a kind of text classification, I will be implementing different machine leaning algorithm which among them the classifier will be best because it standard for text-based processing. The actual goal is in developing a model which was the text transformation (count vectorizer vs tfidf vectorizer) and selecting which type of text to use (headlines vs full text). Now the next step is to extract the most optimal features for countvectorizer or tfidf-vectorizer, this is done by using a n-number of the most used words, and/or phrases, lower casing or not, mainly removing the stop words which are common words such as “the”, “when”, and “there” and only using those words that appear at least a given number of times in each text dataset.



Modeling Code

Modeling

```
# Function to plot the confusion matrix
from sklearn import metrics
import itertools

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

Split the data:

Splitting the data is the most essential step in machine learning. We train our model on the trainset and test our data on the testing set. We split our data in train and test using the train_test_split function from Scikit learn.

```
# Split the data
X_train,X_test,y_train,y_test = train_test_split(data['text'], data.target, test_size=0.2, random_state=42)
```

We split our 80% data for the training set and the remaining 20% data for the testing set.

Tfidf-Vectorizer: (Term Frequency * Inverse Document Frequency)

1.Term Frequency: The number of times a word appears in a document divided by the total number of words in the document. Every document has its own term frequency.

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{i,j}}$$

2. Inverse Document Frequency: The log of the number of documents divided by the number of documents that contain the word w. Inverse data frequency determines the weight of rare words across all documents in the corpus.

$$idf(w) = log(\frac{N}{df_t})$$

Finally, Tfidf vectorizer

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

This vectorizer is already predefined in Scikit Learn Library so we can import by:

```
from sklearn.feature_extraction.text import TfidfTransformer
```

After vectorizing the data, it will return the sparse matrix so that for machine learning algorithms we have to convert it into arrays. to array function will do that work for us.

Logistic regression

Logistic regression

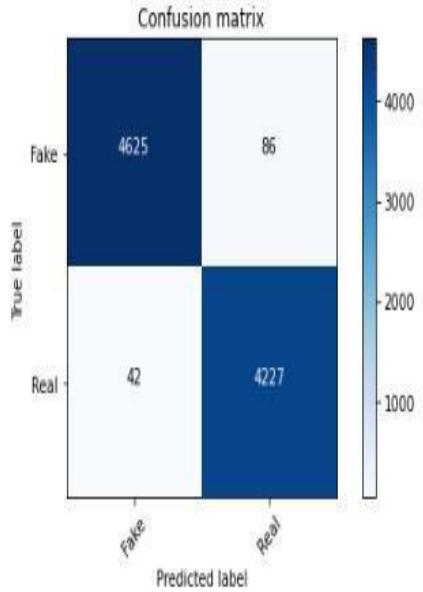
```
# Vectorizing and applying TF-IDF
from sklearn.linear_model import LogisticRegression

pipe = Pipeline([('vect', CountVectorizer()),
                 ('tfidf', TfidfTransformer()),
                 ('model', LogisticRegression())])

# Fitting the model
model = pipe.fit(X_train, y_train)

# Accuracy
prediction = model.predict(X_test)
print("accuracy: {}%".format(round(accuracy_score(y_test, prediction)*100,2)))
```

```
accuracy: 98.57%
```

I got an accuracy of 98.57%. The confusion matrix:

```
cm = metrics.confusion_matrix(y_test, prediction)
plot_confusion_matrix(cm, classes=['Fake', 'Real'])
```

```
Confusion matrix, without normalization
```



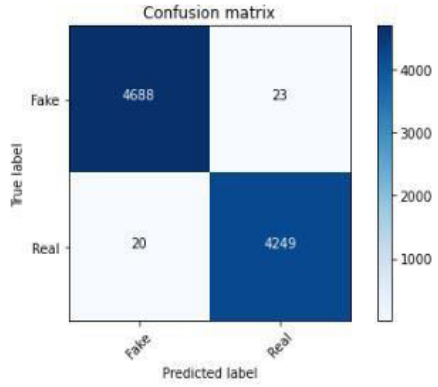
Decision Tree Classifier

Decision Tree Classifier

```
In [35]: from sklearn.tree import DecisionTreeClassifier

         # Vectorizing and applying TF-IDF
         pipe = Pipeline([('vect', CountVectorizer()),
                          ('tfidf', TfidfTransformer()),
                          ('model', DecisionTreeClassifier(criterion= 'entropy',
                                                           max_depth = 20,
                                                           splitter='best',
                                                           random_state=42))])
         # Fitting the model
         model = pipe.fit(X_train, y_train)

         # Accuracy
         prediction = model.predict(X_test)
         print("accuracy: {}%".format(round(accuracy_score(y_test, prediction)*100,2)))

         accuracy: 99.52%
```

I got an accuracy of 99.52%. The confusion matrix:

```
cm = metrics.confusion_matrix(y_test, prediction)
plot_confusion_matrix(cm, classes=['Fake', 'Real'])

Confusion matrix, without normalization
```



Random Forest Classifier

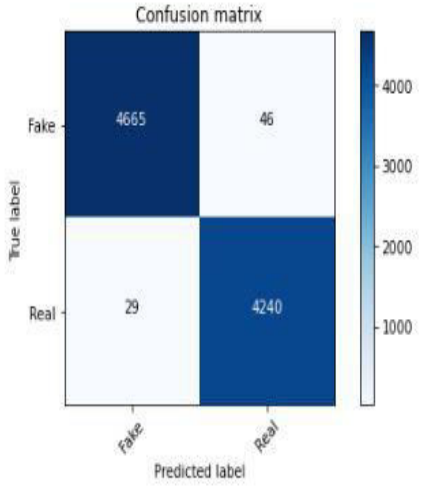
Random Forest Classifier

```
7]: from sklearn.ensemble import RandomForestClassifier

    pipe = Pipeline([('vect', CountVectorizer()),
                     ('tfidf', TfidfTransformer()),
                     ('model', RandomForestClassifier(n_estimators=50, criterion="entropy"))])

    model = pipe.fit(X_train, y_train)
    prediction = model.predict(X_test)
    print("accuracy: {}%".format(round(accuracy_score(y_test, prediction)*100,2)))

    accuracy: 99.16%
```

I got an accuracy of 99.61%. The confusion matrix:

```
cm = metrics.confusion_matrix(y_test, prediction)
plot_confusion_matrix(cm, classes=['Fake', 'Real'])
```

```
Confusion matrix, without normalization
```



Gradient Boosting Classifier

Gradient Boosting Classifier

```
In [41]: from sklearn.ensemble import GradientBoostingClassifier
```

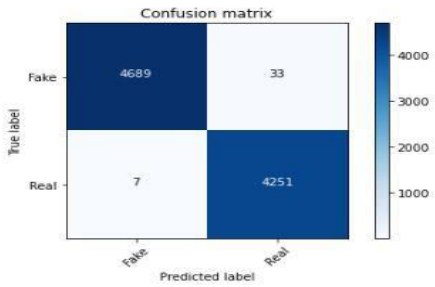
```
In [42]: pipe = Pipeline([('vect', CountVectorizer()),
                         ('tfidf', TfidfTransformer()),
                         ('model', GradientBoostingClassifier())])
# Fitting the model
model = pipe.fit(X_train, y_train)

# Accuracy
prediction = model.predict(X_test)
print("accuracy: {}%".format(round(accuracy_score(y_test, prediction)*100,2)))
```

```
accuracy: 99.55%
```

I got an accuracy of 99.55%. The confusion matrix

```
cm = metrics.confusion_matrix(y_test, prediction)
plot_confusion_matrix(cm, classes=['Fake', 'Real'])
```

```
Confusion matrix, without normalization
```



Naive Bayes MultinomialNB

Naive_Bayes MultinomialNB

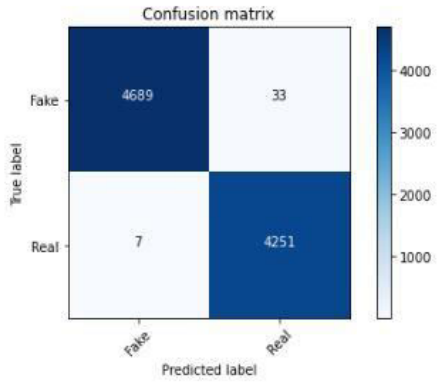
```
7]: from sklearn.naive_bayes import MultinomialNB
    pipe = Pipeline([('vect', CountVectorizer()),
                     ('tfidf', TfidfTransformer()),
                     ('model', MultinomialNB())])
    model = pipe.fit(X_train, y_train)

    # Accuracy
    prediction = model.predict(X_test)
    print("accuracy: {}%".format(round(accuracy_score(y_test, prediction)*100,2)))
```

```
accuracy: 95.6%
```

I got an accuracy of 95.6%. The confusion matrix

```
cm = metrics.confusion_matrix(y_test, prediction)
plot_confusion_matrix(cm, classes=['Fake', 'Real'])
```

```
Confusion matrix, without normalization
```



Stochastic gradient descent

Stochastic gradient descent

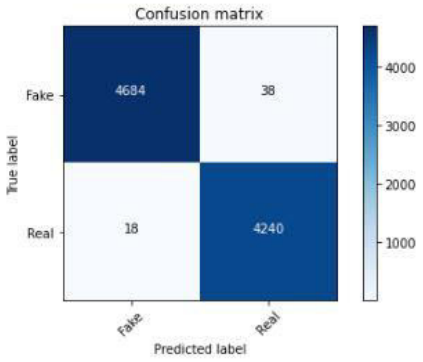
```
n [49]: from sklearn.linear_model import SGDClassifier
        pipe = Pipeline([('vect', CountVectorizer()),
                         ('tfidf', TfidfTransformer()),
                         ('model', SGDClassifier())])

        model = pipe.fit(X_train, y_train)

        # Accuracy
        prediction = model.predict(X_test)
        print("accuracy: {}%".format(round(accuracy_score(y_test, prediction)*100,2)))
```

```
accuracy: 99.38%
```

I got an accuracy of 99.38%. The confusion matrix

In [50]:
```
cm = metrics.confusion_matrix(y_test, prediction)
plot_confusion_matrix(cm, classes=['Fake', 'Real'])
```

```
Confusion matrix, without normalization
```



Fake News Detection Code

Importing Libraries

```
import pandas as pd   #importing data

import numpy as np    #for working with arrays

import matplotlib.pyplot as plt #plotting

import seaborn as sns #drawing attractive and informative statistical graphics

from sklearn.feature_extraction.text import CountVectorizer #text documents

from sklearn.feature_extraction.text import TfidfTransformer

from sklearn import feature_extraction, linear_model, model_selection, preprocessing

from sklearn.metrics import accuracy_score

from sklearn.model_selection import train_test_split

from sklearn.pipeline import Pipeline #cross-validated
```

Read Datasets

```
trueData = pd.read_csv("data/True.csv")

fakeData = pd.read_csv("data/Fake.csv")

trueData.shape

trueData.head()

fakeData.shape

fakeData.head()
```

Cleaning Data

```
# Add flag to track real and fake
trueData['target'] = 'true'
fakeData['target'] = 'fake'

# Concatenate dataframes
data = pd.concat([fakeData, trueData]).reset_index(drop = True)
data.shape
data.head()

# Shuffle the data
from sklearn.utils import shuffle
data = shuffle(data)
data = data.reset_index(drop=True)
data.shape
data.head()
```

```
#Removing the date (we won't use it for the analysis):
data.drop(["date"],axis=1,inplace=True)
data.head()

#Removing the title (we will only use the text):
data.drop(["title"],axis=1,inplace=True)
data.head()

#Convert the text to lowercase:
data['text'] = data['text'].apply(lambda x: x.lower())
data.head()

#Remove punctuation:
import string
def punctuation_removal(text):
    all_list = [char for char in text if char not in string.punctuation]
    clean_str = ''.join(all_list)
    return clean_str
data['text'] = data['text'].apply(punctuation_removal)
data.head()

# Removing stopwords
import nltk #Natural Language Toolkit
nltk.download('stopwords')
from nltk.corpus import stopwords
stop = stopwords.words('english')
data['text'] = data['text'].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop)]))
data.head()
```

Basic data exploration

```
# How many articles per subject?
print(data.groupby(['subject'])['text'].count())
data.groupby(['subject'])['text'].count().plot(kind="bar")
plt.show()

# How many fake and real articles?
print(data.groupby(['target'])['text'].count())
data.groupby(['target'])['text'].count().plot(kind="bar")
plt.show()

# Wordcloud for fake news
from wordcloud import WordCloud

fake_data = data[data["target"] == "fake"]
all_words = ' '.join([text for text in fake_data.text])

wordcloud = WordCloud(width= 800, height= 500,
```

```
                                        max_font_size = 110,
                                        collocations = False).generate(all_words)

plt.figure(figsize=(10,7))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()

# Wordcloud for real news
from wordcloud import WordCloud

real_data = data[data["target"] == "true"]
all_words = ' '.join([text for text in fake_data.text])

wordcloud = WordCloud(width= 800, height= 500,
                                        max_font_size = 110,
                                        collocations = False).generate(all_words)

plt.figure(figsize=(10,7))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()

# Most frequent words counter
from nltk import tokenize

token_space = tokenize.WhitespaceTokenizer()

def counter(text, column_text, quantity):
    all_words = ' '.join([text for text in text[column_text]])
    token_phrase = token_space.tokenize(all_words)
    frequency = nltk.FreqDist(token_phrase)
    df_frequency = pd.DataFrame({"Word": list(frequency.keys()),
                                 "Frequency": list(frequency.values())})
    df_frequency = df_frequency.nlargest(columns = "Frequency", n = quantity)
    plt.figure(figsize=(12,8))
    ax = sns.barplot(data = df_frequency, x = "Word", y = "Frequency", color = 'blue')
    ax.set(ylabel = "Count")
    plt.xticks(rotation='vertical')
    plt.show()

# Most frequent words in fake news
counter(data[data["target"] == "fake"], "text", 20)

# Most frequent words in real news
counter(data[data["target"] == "true"], "text", 20)
```

Modeling

```
# Function to plot the confusion matrix
from sklearn import metrics
import itertools

def plot_confusion_matrix(cm, classes,
                                        normalize=False,
                                        title='Confusion matrix',
                                        cmap=plt.cm.Blues):

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

Preparing the data

```
# Split the data
X_train,X_test,y_train,y_test = train_test_split(data['text'], data.target, test_size=0.2, random_state=42)
```

Logistic regression

```
# Vectorizing and applying TF-IDF
from sklearn.linear_model import LogisticRegression

pipe = Pipeline([('vect', CountVectorizer()),
                 ('tfidf', TfidfTransformer()),
                 ('model', LogisticRegression())])
```

```
# Fitting the model
model = pipe.fit(X_train, y_train)

# Accuracy
prediction = model.predict(X_test)
print("accuracy: {}%".format(round(accuracy_score(y_test, prediction)*100,2)))

cm = metrics.confusion_matrix(y_test, prediction)
plot_confusion_matrix(cm, classes=['Fake', 'Real'])
```

Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier

# Vectorizing and applying TF-IDF
pipe = Pipeline([('vect', CountVectorizer()),
                 ('tfidf', TfidfTransformer()),
                 ('model', DecisionTreeClassifier(criterion= 'entropy',
                                                  max_depth = 20,
                                                  splitter='best',
                                                  random_state=42))])
# Fitting the model
model = pipe.fit(X_train, y_train)

# Accuracy
prediction = model.predict(X_test)
print("accuracy: {}%".format(round(accuracy_score(y_test, prediction)*100,2)))

cm = metrics.confusion_matrix(y_test, prediction)
plot_confusion_matrix(cm, classes=['Fake', 'Real'])
```

Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier

pipe = Pipeline([('vect', CountVectorizer()),
                 ('tfidf', TfidfTransformer()),
                 ('model', RandomForestClassifier(n_estimators=50, criterion="entropy"))])
# Fitting the model
model = pipe.fit(X_train, y_train)
# Accuracy
prediction = model.predict(X_test)
print("accuracy: {}%".format(round(accuracy_score(y_test, prediction)*100,2)))
```

```
cm = metrics.confusion_matrix(y_test, prediction)
plot_confusion_matrix(cm, classes=['Fake', 'Real'])
```

Naive_Bayes MultinomialNB

```
from sklearn.naive_bayes import MultinomialNB
pipe = Pipeline([('vect', CountVectorizer()),
                 ('tfidf', TfidfTransformer()),
                 ('model', MultinomialNB())])
# Fitting the model
model = pipe.fit(X_train, y_train)

# Accuracy
prediction = model.predict(X_test)
print("accuracy: {}%".format(round(accuracy_score(y_test, prediction)*100,2)))

cm = metrics.confusion_matrix(y_test, prediction)
plot_confusion_matrix(cm, classes=['Fake', 'Real'])
```

Gradient Boosting Classifier

```
from sklearn.ensemble import GradientBoostingClassifier
pipe = Pipeline([('vect', CountVectorizer()),
                 ('tfidf', TfidfTransformer()),
                 ('model', GradientBoostingClassifier())])
# Fitting the model
model = pipe.fit(X_train, y_train)

# Accuracy
prediction = model.predict(X_test)
print("accuracy: {}%".format(round(accuracy_score(y_test, prediction)*100,2)))
cm = metrics.confusion_matrix(y_test, prediction)
plot_confusion_matrix(cm, classes=['Fake', 'Real'])
```