FAKE NEWS DETECTION USING NLP

TEAM MEMBER

NAME: J.JAYASREE

REG NO: 953421106019/au953421106019

Phase 4-Development Part 2

PROJECT: Fake News Detection

OBJECTIVE:-

The objective of fake news detection using Natural Language Processing (NLP) is to develop a system that can effectively differentiate between reliable and unreliable information in textual content.

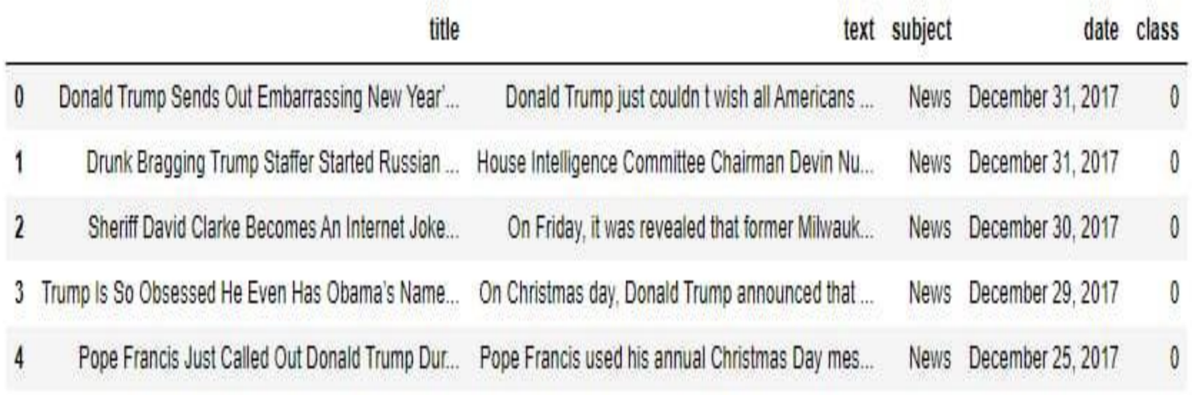
Phase 4: Development Part 2

Introduction:

In this part we will continue building our project. Continue building the fake news detection model by applying NLP techniques and training a classification model. Text Preprocessing and Feature Extraction. Model training and evaluation

Used Data Set:

https://www.kaggle.com/datasets/clmentbisaillon/fake-and-real-news-dataset



	title	text	subject	date	class
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

Python Program:-

```
#Tf-idf Bigrams
#Initialize the `tfidf_vectorizer`
tfidf_vectorizer = TfidfVectorizer(stop_words='english',
ngram_range = (2,2))
# Fit and transform the training data
tfidf1_train =
tfidf_vectorizer.fit_transform(X_train.astype('str'))
# Transform the test set
tfidf1_test =
tfidf_vectorizer.transform(X_test.astype('str'))

pickle.dump(tfidf1_train, open("tfidf1_train.pickle",
"wb"))

pickle.dump(tfidf1_test, open("tfidf1_test.pickle", "w
b"))
#Top 10 tfidf bigrams

tfidf_vectorizer.get_feature_names()[-10:]
```

Out:-

```
['中文 cooperación',

'中文 coopération',

'中文 التعاون',
'大量转体 mediamass',

'山崎 コロッケ',

'殆 ww reverbnation',

'点击查看本文中文版 despite',

'点击查看本文中文版 foreigner',

'無為 translates',

'版权所有 2012']
```

Program:-

```
   tfidf1_train
```

Out:-

```
<18669x4013463 sparse matrix of type '<class
'numpy.float64'>'
with 6804483 stored elements in Compressed Sparse
Row format>
```

Program:-

```
#Confusion Matrix
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')
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')

print("Accuracy with Multinomial Naive Bayes: %0.3f"
% score)
```

Out

```
<18669x4013463 sparse matrix of type '<class
'numpy.float64'>'
```

```
with 6804483 stored elements in Compressed Sparse
Row format>
Accuracy with Multinomial Naive Bayes: 0.829
```

Program:-

```
cm = metrics.confusion_matrix(Y_test, pred,
labels=['FAKE', 'REAL'])
plot_confusion_matrix(cm, classes=['FAKE', 'REAL'])
```

Program:-

```
clf = GradientBoostingClassifier()
clf.fit(tfidf1_train, Y_train)
pickle.dump(clf, open('tfidf_gb', 'wb'))
#model = pickle.load(open('tfidf_gb', 'rb'))
pred = clf.predict(tfidf1_test)
score = metrics.accuracy_score(Y_test, pred)
print("Accuracy with Gradient Boosting: %0.3f" %
score)
In [101]:
cm = metrics.confusion_matrix(Y_test, pred,
labels=['FAKE', 'REAL'])
plot_confusion_matrix(cm, classes=['FAKE', 'REAL'])
```

Program:-

```
clf = RandomForestClassifier()
clf.fit(tfidf1_train, Y_train)
```

```
pickle.dump(clf, open('tfidf_rf', 'wb'))
pred = clf.predict(tfidf1_test)
score = metrics.accuracy_score(Y_test, pred)
print("Accuracy with RandomForestClassifier: %0.3f" %
score)
```

Out:-

```
Accuracy with RandomForestClassifier: 0.865
```

Program:-

```
#Generating the POS tags for all the articles and adding a
new column by replacing text w
ith their POS tags
nlp = spacy.load('en_core_web_sm')
x = []
df["Text"] = df["Headline"].map(str) + df["Body"]
for text in df['Text']:
text_new = []
doc = nlp(text)
for token in doc:
text_new.append(token.pos_)
txt = ' '.join(text_new)
x.append(txt)
df['Text_pos'] = x
df.to_pickle('newdata.pkl')

df = pd.read_pickle('newdata.pkl')
cnt = 0
ind = []
for art in df['Body']:
```

```
#print(type(art))
if len(str(art)) < 10:
ind.append(cnt)
cnt+=1
df = df.drop(df.index[ind])
y = df.Label
y = y.astype('str')
x_train, x_test, y_train, y_test =
train_test_split(df['Text_pos'],y, test_size=0.33)
x_train
Accuracy with RandomForestClassifier: 0.865
Confusion matrix
```

Out:-

```
4574 PROPN PROPN PROPN VERB NOUN ADP PROPN PROPN AD...
9417 ADV PROPN VERB PROPN ADP PROPN VERB VERB PART ...
21627 NOUN PUNCT ADJ NOUN VERB NUM ADP VERB ADJ ADP ...
4755 PUNCT PUNCT PUNCT PROPN PUNCT PUNCT PUNCT NOUN...
19184 PROPN PROPN PROPN PROPN PROPN PROPN VERB VERB ...
19182 ADP PRON VERB ADJ ADP DET PROPN ADP DET PROPN ...
28908 NOUN VERB ADP PROPN PROPN NOUN PUNCT PUNCT PRO...

28908 NOUN VERB ADP PROPN PROPN NOUN PUNCT PUNCT PRO...
1589 PROPN PART PROPN PUNCT PROPN SYM PROPN PROPN P...
14069 NUM PROPN PROPN CCONJ PROPN PROPN VERB ADP PRO...
20099 PROPN NUM PUNCT PROPN PROPN PROPN NUM PUNCT AD...
1578 VERB PRON VERB ADP DET PROPN PROPN PRON VERB A...
21617 PROPN PROPN PROPN PROPN ADP PROPN PROPN PROPN ...
24919 NUM NOUN ADP PROPN PART PROPN PROPN PUNCT DET ...
17073 PROPN VERB VERB PROPN PROPN PROPN ADV ADP NOUN...
9210 PROPN VERB PRON ADV PUNCT ADV ADP PROPN ADJ AD...
26110 PROPN PROPN PROPN PROPN PROPN PUNCT NOUN VERB ...
19032 PROPN PUNCT ADJ VERB ADV ADP PROPN PROPN PUNCT...
21120 NUM PROPN PROPN PROPN VERB DET NOUN PUNCT NOUN...
28216 PROPN ADP PROPN PROPN PROPN PROPN PROPN ADP PR...
3745 PROPN PROPN VERB DET PROPN PROPN PUNCT PROPN P...
26307 ADV DET PROPN PROPN PUNCT PROPN PROPN ADV VERB...
16599 PROPN CCONJ PROPN PROPN ADP PROPN ADP PROPN PR...
16018 PROPN PART PROPN VERB VERB DET PROPN ADP PROPN...
20269 PROPN PROPN VERB NOUN NOUN VERB VERB DET ADJ A...
19783 PUNCT ADV NOUN NOUN VERB PROPN PROPN NOUN PUNC...
11410 NOUN PUNCT PROPN PROPN PROPN PROPN PROPN CCONJ...
22889 PROPN VERB ADP PROPN PROPN PROPN PRON VERB PAR...
19269 PROPN PROPN PART VERB NOUN PROPN VERB ADP ADJ ...
```

```
15738 NOUN PUNCT PROPN PROPN PROPN PROPN VERB NOUN N...
30577 PROPN PROPN ADP PROPN PROPN PROPN PUNCT VERB A...
10029 DET PROPN PROPN NOUN ADP PROPN CCONJ PROPN DET...
7194 PROPN PROPN PROPN VERB DET ADJ NOUN ADP ADJ NO...
22436 PROPN PROPN VERB PART PUNCT PROPN PROPN PROPN ...
8468 PROPN NOUN NOUN VERB ADJ NOUN ADP NUM PUNCT NO...
25138 PROPN PUNCT PROPN CCONJ ADV PUNCT VERB PUNCT P...
28411 PROPN CCONJ PROPN VERB NOUN NOUN ADV ADP DET N...
19989 PROPN PROPN PUNCT PROPN VERB VERB PROPN ADP NU...
27388 PROPN NOUN VERB PROPN PUNCT PROPN VERB PROPN N...
2201 PROPN PROPN PROPN NUM ADP PROPN PROPN PROPN AD...
19867 ADJ PROPN PUNCT PROPN ADP PROPN NUM PUNCT NUM ...
18790 PROPN PART PROPN PUNCT PROPN PART NUM PROPN PR...
18553 NOUN ADP PROPN PROPN PROPN PROPN PROPN PART PR...
25688 PROPN PART PROPN PROPN VERB PROPN ADP PROPN PR...
18711 PROPN PROPN PART VERB DET PROPN PROPN PUNCT VE...
28233 PROPN PROPN VERB PROPN PROPN PART PROPN ADP PR...
23720 PUNCT ADP NUM NOUN PRON VERB VERB PUNCT PUNCT ...
2642 PROPN PROPN PROPN PROPN ADP PROPN VERB ADP PRO...
9161 PROPN PROPN PUNCT PRON VERB PROPN PROPN PART N...
13332 PROPN NOUN PUNCT ADV ADV VERB PRON VERB DET PR...
10227 PROPN PROPN PART PROPN PROPN PROPN PROPN PROPN...
15856 PROPN PROPN NUM PUNCT PROPN VERB PROPN DET ADJ...
16990 PROPN CCONJ PROPN PROPN PROPN PROPN ADP PROPN ...
17590 ADJ NOUN ADP PROPN NUM SYM PROPN PROPN PROPN P...
15295 PROPN PUNCT PROPN PROPN PROPN PUNCT PROPN PUNC...
30775 PROPN PROPN VERB NOUN ADP NOUN ADP DET VERB NO...
17756 PROPN VERB ADJ PUNCT CCONJ PROPN VERB PRON PUN...
6451 PROPN CCONJ PROPN PROPN PART VERB PROPN PROPN ...
26272 PROPN PROPN PUNCT PROPN VERB NOUN PUNCT NOUN N...
8193 PROPN PROPN PROPN PROPN PART VERB PROPN ADP PR...
10999 ADP PROPN PUNCT DET PROPN ADP PROPN PROPN PART...
```

Program:-

```
#Initialize the `tfidf_vectorizer`
tfidf_vectorizer = TfidfVectorizer(stop_words='english',
ngram_range = (2,2))
# Fit and transform the training data
tfidf_train =
tfidf_vectorizer.fit_transform(x_train.astype('str'))
# Transform the test set
tfidf_test = tfidf_vectorizer.transform(x_test.astype('str'))
pickle.dump(tfidf_train, open("tfidf_train.pickle", "wb"))
pickle.dump(tfidf_test, open("tfidf_test.pickle", "wb"))
```

```
tfidf_vectorizer.get_feature_names()[-10:]
```

Out:-

```
['verb det',
'verb intj',
'verb noun',
'verb num',
'verb pron',
'verb propn',
'verb punct',
'verb space',
'verb sym',
'verb verb']
```

```
tfidf_train
```

Out:-

```
<18772x196 sparse matrix of type '<class 'numpy.float64'>'
with 1612837 stored elements in Compressed Sparse Row
format>
```

In [28]:

```
tfidf_vectorizer.get_feature_names()[-10:]
```

Out[28]:

```
['verb det',
 'verb intj',
 'verb noun',
 'verb num',
 'verb pron',
 'verb propn',
 'verb punct',
 'verb space',
 'verb sym',
 'verb verb']
```

In [29]:

```
tfidf_train
```

Out[29]:

```
<18772x196 sparse matrix of type '<class 'numpy.float64'>'
 with 1612837 stored elements in Compressed Sparse Row format>
```

In [30]:

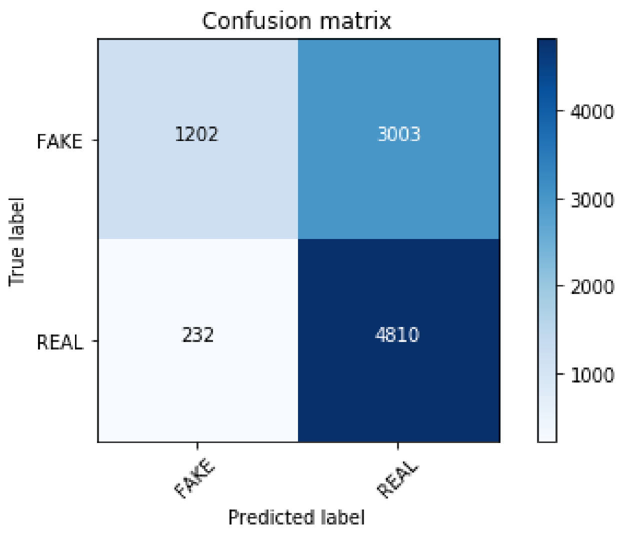
```
clf = MultinomialNB()
clf.fit(tfidf_train, y_train)
pickle.dump(clf, open('pos_nb', 'wb'))
pred = clf.predict(tfidf_test)
score = metrics.accuracy_score(y_test, pred)
print("Accuracy with Multinomial Naive Bayes:   %0.3f" % score)
```

```
Accuracy with Multinomial Naive Bayes:   0.665
```

In [43]:

```
cm = metrics.confusion_matrix(y_test, pred, labels=['FAKE', 'REAL'])
plot_confusion_matrix(cm, classes=['FAKE', 'REAL'])
```

```
Confusion matrix
```



In [44]:

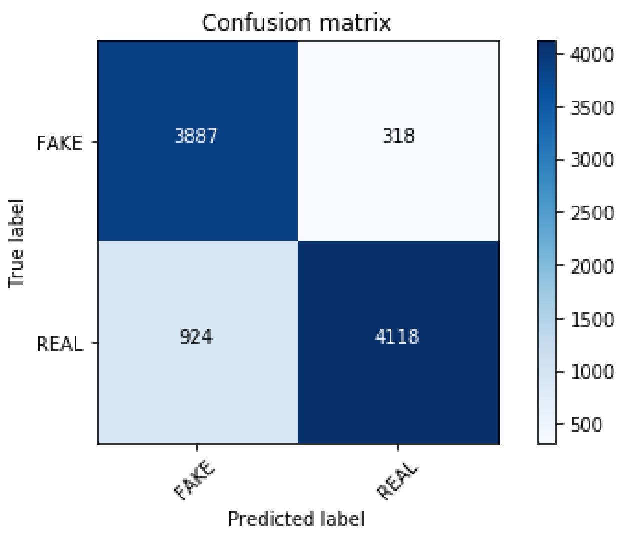
```
clf = RandomForestClassifier()
clf.fit(tfidf_train, y_train)
pickle.dump(clf, open('pos_rf', 'wb'))
pred = clf.predict(tfidf_test)
score = metrics.accuracy_score(y_test, pred)
print("Accuracy with RandomForestClassifier:   %0.3f" % score)
```

```
Accuracy with RandomForestClassifier:   0.866
```

In [45]:

```
cm = metrics.confusion_matrix(y_test, pred, labels=['FAKE', 'REAL'])
plot_confusion_matrix(cm, classes=['FAKE', 'REAL'])
```

```
Confusion matrix
```



In [46]:

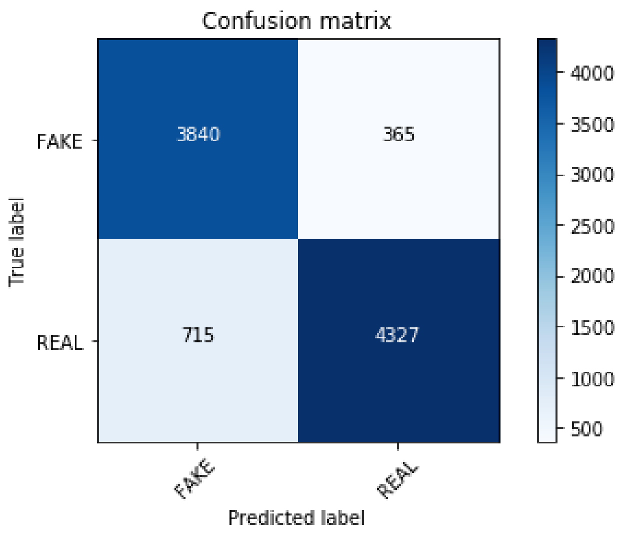
```
clf = GradientBoostingClassifier()
clf.fit(tfidf_train, y_train)
pickle.dump(clf, open('pos_gb', 'wb'))
pred = clf.predict(tfidf_test)
score = metrics.accuracy_score(y_test, pred)
print("Accuracy with Gradient Boosting:   %0.3f" % score)
```

```
Accuracy with Gradient Boosting:   0.883
```

In [47]:

```
cm = metrics.confusion_matrix(y_test, pred, labels=['FAKE', 'REAL'])
plot_confusion_matrix(cm, classes=['FAKE', 'REAL'])
```

```
Confusion matrix
```



In [31]:

```
#Getting the score of semantic categories generated by Empath of each article and generat
ing a tfidf vector of the unigrams
lexicon = Empath()
semantic = []
cnt = 0
df["Text"] = df["Headline"].map(str) + df["Body"]

for article in df['Text']:
    if article == '':
        continue
    cnt+=1
    d = lexicon.analyze(article, normalize = False)
    x = []
    for key, value in d.items():
        x.append(value)
    x = np.asarray(x)
    semantic.append(x)
df['Semantic'] = semantic
```

In [32]:

```
categories = []
a = lexicon.analyze("")
for key, value in a.items():
    categories.append(key)
categories
```

Out[32]:

```
['exercise',
 'horror',
 'hearing',
 'college',
 'science',
 'car',
 'government',
 'toy',
 'rural',
 'poor',
 'strength',
 'music',
 'weather',
 'payment',
 'disappointment',
 'dispute',
 'leader',
 'trust',
 'shame',
 'help',
 'musical',
 'appearance',
 'breaking',
 'ocean',
 'clothing',
 'farming',
 'traveling',
 'fabric',
 'social_media',
 'nervousness',
 'pride',
 'joy',
 'achievement',
 'zest',
 'writing',
 'ridicule',
 'anticipation',
 'suffering',
 'leisure',
 'driving',
 'party',
 'occupation',
 'sympathy',
 'reading',
 'power',
 'banking',
 'communication',
 'healing',
 'ancient',
 'masculine',
 'emotional',
 'affection',
 'messaging',
 'cooking',
 'terrorism',
 'swimming',
 'confusion',
 'death',
 'negative_emotion',
 'sound',
 'valuable',
 'beach',
 'law',
 'beauty',
 'anger',
 'superhero',
 'sailing',
 'restaurant',
 'family',
 'cold',
 'rage',
 'economics',
 'cleaning',
 'play',
 'exasperation',
 'exotic',
 'weapon',
 'positive_emotion',
 'ugliness',
 'royalty',
 'speaking',
 'dominant_personality',
 'politics',
 'hygiene',
 'feminine',
 'alcohol',
 'religion',
 'violence',
 'envy',
 'medical_emergency',
 'fight',
 'animal',
 'domestic_work',
 'war',
 'contentment',
 'phone',
 'shape_and_size',
 'timidity',
 'independence',
 'business',
 'torment',
 'internet',
 'heroic',
 'vacation',
 'crime',
 'gain',
 'philosophy',
 'divine',
 'giving',
 'money',
 'love',
 'home',
 'monster',
 'sexual',
 'blue_collar_job',
 'meeting',
 'dance',
 'stealing',
 'noise',
 'sadness',
 'school',
 'order',
 'fire',
'plant',
'neglect',
'vehicle',
'ship',
'smell',
'legend',
'weakness',
'worship',
'fashion',
'negotiate',
'movement',
'journalism',
'sleep',
'fear',
'celebration',
'programming',
'children',
'work',
'childish',
'medieval',
'night',
'friends',
'urban',
'dominant_heirarchical',
'body',
'surprise',
'youth',
'hipster',
'aggression',
'wealthy',
'competing',
'shopping',
'lust',
'furniture',
'warmth',
'wedding',
'art',
'optimism',
'real_estate',
'fun',
'office',
'military',
'irritability',
'water',
'cheerfulness',
'sports',
'pain',
'politeness',
'technology',
'injury',
'health',
'prison',
'anonymity',
'computer',
'disgust',
'hate',
'pet',
'eating',
'white_collar_job',
'liquid',
'kill',
'attractive',
'hiking',
'magic',
'air_travel',
'deception',
'morning',
'swearing_terms',
'tourism',
'listen',
'tool']
```