FAKE NEWS DETECTION USING NLP

TEAM MEMBER

NAME: J.JAYASREE

REG NO: 953421106019/au953421106019

PHASE 5: Project submission

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.

Abstract:

Fake news has become a pervasive problem in our digital age, with potentially far-reaching consequences for society. Detecting and mitigating the spread of misinformation is of paramount importance. This research explores the application of Natural Language Processing (NLP) techniques to identify and combat fake news.

This study employs a multifaceted approach, integrating advanced machine learning models, linguistic analysis, and deep learning techniques to discern between genuine and fabricated news articles.

The research focuses on developing a robust NLP-based system that can automatically identify fake news by analyzing textual content, linguistic patterns, and social context.

Key steps in our methodology include pre processing and cleaning of textual data, feature extraction, and the development of a classification model.

We experiment with a range of NLP tools, such as tokenization, word embeddings, and sentiment analysis, to capture linguistic nuances and contextual cues in the text.

The experimental results on various datasets demonstrate the effectiveness of the proposed NLP-based approach in fake news detection. Our system achieves high accuracy, precision, and recall, showcasing its potential for real-world applications.

In conclusion, this research presents a promising avenue for addressing the growing issue of fake news through the power of NLP.

The development of an automated system to identify and combat misinformation can contribute to a more informed and resilient society, and our findings provide valuable insights for future research and practical implementation in the fight against fake news.

Here's a list of tools and software commonly used in the process:

1. Programming Language:

- Python is the most popular language for machine learning due to its extensive libraries and frameworks. You can use libraries like NumPy, pandas, scikit-learn, and more.

2. Integrated Development Environment (IDE):

- Choose an IDE for coding and running machine learning experiments. Some popular options include Jupyter Notebook, Google Colab, or traditional IDEs like PyCharm.

3. Machine Learning Libraries:

- You'll need various machine learning libraries, including:

- scikit-learn for building and evaluating machine learning models.

- TensorFlow or PyTorch for deep learning, if needed.

- XGBoost, LightGBM, or CatBoost for gradient boosting models.

4. Data Visualization Tools: - Tools like Matplotlib, Seaborn, or Plotly are essential for data exploration and visualization.

5. Data Preprocessing Tools:

- Libraries like pandas help with data cleaning, manipulation, and preprocessing.

6. Data Collection and Storage:

- Depending on your data source, you might need web scrapingtools (e.g., BeautifulSoup or Scrapy) or databases (e.g., SQLite,PostgreSQL) for data storage.

7. Version Control:

- Version control systems like Git are valuable for tracking changes in your code and collaborating with others.

8. Notebooks and Documentation:

- Tools for documenting your work, such as Jupyter Notebooks or Markdown for creating README files and documentation.

9. Hyperparameter Tuning:

- Tools like GridSearchCV or RandomizedSearchCV from scikit-learn can help with hyperparameter tuning.

10. Web Development Tools (for Deployment):

- If you plan to create a web application for model deployment, knowledge of web development tools like Flask or Django for backend development, and HTML, CSS, and JavaScript for the front-end can be useful.

11. Cloud Services (for Scalability):

- For large-scale applications, cloud platforms like AWS, Google

Cloud, or Azure can provide scalable computing and storage resources.

12. External Data Sources (if applicable):

- Depending on your project's scope, you might require tools to access external data sources, such as APIs or data scraping tools.

13. Data Annotation and Labeling Tools (if applicable):

- For specialized projects, tools for data annotation and labeling may be necessary, such as Labelbox or Supervisely.

14. Geospatial Tools (for location-based features):

- If your dataset includes geospatial data, geospatial libraries like GeoPandas can be helpful.

Design Thinking:

Detecting fake news using Natural Language Processing (NLP) involves several steps. Here's a high-level

Overview of the process:

1.Data Collection: Gather a dataset of news articles labeled as either real or fake. You can find such Datasets online or create your own.

2.Data Preprocessing: Clean the text data by removing stop words, punctuation, and special Characters. Tokenize the text into words or phrases.

3.Feature Extraction: Convert the text data into numerical features that NLP models can work with. Common techniques include TF-IDF (Term Frequency-Inverse Document Frequency) and word Embeddings like Word2Vec or GloVe.

4.Model Selection: Choose an appropriate NLP model for fake news detection. Common choices include

5.Multinomial Naïve Baye: A simple probabilistic model.

6.Logistic Regression: A linear model that works well for text classification.

7.Recurrent Neural Networks (RNNs): Particularly LSTM or GRU cells for sequential data.

8.Convolutional Neural Networks (CNNs): For analyzing local text patterns.

9.Transformers: State-of-the-art models like BERT, GPT-3, or RoBERTa.

10.Model Training: Train the selected model on your labeled dataset. This involves feeding it the Preprocessed data and adjusting the model's parameters until it performs well.

11.Evaluation: Use evaluation metrics like accuracy, precision, recall, and F1-score to measure the Model's performance. You might also use techniques like cross-validation to ensure the model Generalizes well.

12.Post-processing: Implement post-processing techniques to improve results. This can include Threshold adjustment for classification, or ensembling multiple models.

13.Deployment: Once you have a well-performing model, deploy it in a real-world setting where it

Can analyze and classify news articles in real-time.

14.Continuous Monitoring: Fake news is dynamic, so your model should be continually monitored
And retrained to adapt to evolving disinformation.

15.User Interface: Create a user-friendly interface where users can input news articles for analysis.

Importing Libraries and Datasets:-

import pandas as pd

Import numpy as np

From sklearn.model_selection import train_test_split

From sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer

Import matplotlib.pyplot as plt

Import itertools

From sklearn import svm

From sklearn.naive_bayes import MultinomialNB

From sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

From sklearn import

metrics Import spacy

From sklearn.feature_extraction.stop_words import ENGLISH_STOP_WORDS

Import string

Import re

Import nltk

Import collections

From nltk.corpus import stopwords

```
From sklearn.feature_extraction import DictVectorizer
From sklearn.pipeline import Pipeline, FeatureUnion
From empath import Empath
From keras.preprocessing.text import Tokenizer
From keras.preprocessing.sequence import pad_sequences
Import pickle

df = pd.read_csv('Dataset/data.csv')
df.loc[df['Label']== 0, 'Label'] = 'REAL'
df.loc[df['Label']== 1, 'Label'] = 'FAKE'
df.columns
df['Label'].value_counts()
```

Out:

```
REAL    2137
FAKE    1872
Name: Label, dtype: int64
```

```
#Dropping the column URLs from the table
df.drop(['URLs'], axis = 1, inplace = True)
df.columns
```

Out:

```
Index(['Headline', 'Body', 'Label'], dtype='object')
```

```
#Selecting only fake news from all the types of news and then replacing the 'fake' by 0
df1 = pd.read_csv('Dataset/fake.csv')
df1.columns
df1['type'].value_counts()
df1 = df1.loc[df1['type']=='fake']
df1.loc[df1['type']== 'fake', 'type'] = 'FAKE'

#Selecting some columns from the table and renaming them\n",
df1 = df1[['title','text','type']]
df1.columns = ['Headline', 'Body', 'Label']
df1['Label'].value_counts()
```

Out:

```
FAKE    19
Name: Label, dtype: int64
```

```
df2= pd.read_csv('Dataset/fake_or_real_news.csv')
```

df2.columns

Out:

Index(['Unnamed: 0', 'title', 'text', 'label'],
dtype='object')

#Selecting few columns from the table and renaming the columns

df2 = df2[['title','text','label']]

df2.columns = ['Headline', 'Body', 'Label']

df2.columns

df2['Label'].value_counts()

Out:

REAL 3171

FAKE 3164

Name: Label, dtype: int64

df3 = pd.read_csv('Dataset/train.csv')

df3.columns

Out:

Index(['id', 'title', 'author', 'text', 'label'], dtype='object')

#Selecting few columns from the table and renaming the columns

df3 = df3[['title','text','label']]

df3.columns = ['Headline', 'Body', 'Label']

df3.loc[df3['Label']== 0, 'Label'] = 'REAL'

df3.loc[df3['Label']== 1, 'Label'] = 'FAKE'

df3.columns

df3['Label'].value_counts()

Out:

FAKE   10413

REAL   10387

Name: Label, dtype: int64

#Appending df1,df2,df3 to df

df = df.append(df1, ignore_index = **True**)

df = df.append(df2, ignore_index = **True**)

```
df = df.append(df3, ignore_index = True)

df = df.drop_duplicates()
# df.iloc[3647]
# print(df['Headline'][3647])
# print(len(df['Body']
[3647]))
#df = df.dropna(how='any',axis=0)
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]) df
# print(df['Headline'][3647])
# print(len(df['Body']
[3647]))
```

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 [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'])
```

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)
```

In [47]:

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

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']:
Confusion matrix
Accuracy with Gradient Boosting: 0.883
Confusion matrix
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',
'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',
'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']
```

```
In [33]:
#TF-IDF vector by taking the score for a semantic class
as its frequency.
sem = []
for i in range(df.shape[0]):
a = []
for j in range(len(semantic[0])):
for k in range(int(semantic[i][j])):
a.append(categories[j])
b = " ".join(a)
sem.append(b)
#print(len(sem))
df['Semantics'] = sem
df.to_pickle('Semantic.pkl')
```

In [34]:
```
df = pd.read_pickle('Semantic.pkl')
print(df.columns)
print(df.shape)
```

In [45]:
```
y = df.Label
y = y.astype('str')
x_train, x_test, y_train, y_test =
train_test_split(df['Semantics'], y, test_size=0.33)
x_train
```
```
Index(['Headline', 'Body', 'Label', 'headline_length',
'body_length',
'Body_pos', 'Text_pos', 'Text', 'Semantic', 'Semantics'],
dtype='object')
(27865, 10)
```

Out [45]:
```
11942 [3.0, 1.0, 0.0, 0.0, 0.0, 2.0, 0.0, 0.0, 0.0, ...
16974 [0.0, 0.0, 0.0, 2.0, 5.0, 0.0, 2.0, 1.0, 0.0, ...
27421 [10.0, 6.0, 0.0, 7.0, 5.0, 2.0, 0.0, 10.0, 0.0...
2099 [0.0, 3.0, 0.0, 0.0, 2.0, 0.0, 0.0, 0.0, 0.0, ...
456 [1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0, ...
20244 [4.0, 10.0, 0.0, 1.0, 2.0, 3.0, 1.0, 37.0, 0.0...
18196 [2.0, 2.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
4110 [6.0, 4.0, 1.0, 33.0, 3.0, 1.0, 0.0, 10.0, 0.0...
7468 [3.0, 0.0, 0.0, 0.0, 4.0, 3.0, 1.0, 0.0, 2.0, ...
16502 [0.0, 0.0, 0.0, 5.0, 2.0, 0.0, 0.0, 0.0, 0.0, ...
18146 [3.0, 4.0, 0.0, 9.0, 1.0, 2.0, 1.0, 5.0, 2.0, ...
6491 [0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, ...
111 [1.0, 1.0, 1.0, 1.0, 0.0, 3.0, 0.0, 0.0, 0.0, ...
9181 [7.0, 0.0, 0.0, 0.0, 2.0, 0.0, 0.0, 0.0, 1.0, ...
30433 [8.0, 8.0, 4.0, 3.0, 2.0, 0.0, 0.0, 2.0, 0.0, ...
14557 [6.0, 0.0, 0.0, 2.0, 5.0, 1.0, 2.0, 0.0, 2.0, ...
4720 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
3608 [2.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 3.0, 0.0, ...
13548 [1.0, 3.0, 1.0, 2.0, 10.0, 0.0, 0.0, 0.0, 2.0,...
```

```
2625 [1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, ...
18943 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
718 [0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, ...
27068 [0.0, 3.0, 9.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0, ...
19956 [0.0, 1.0, 0.0, 1.0, 2.0, 1.0, 0.0, 0.0, 0.0, ...
10544 [1.0, 2.0, 3.0, 1.0, 2.0, 2.0, 4.0, 0.0, 2.0, ...
23551 [5.0, 21.0, 2.0, 0.0, 2.0, 2.0, 2.0, 0.0, 2.0, ...
7966 [1.0, 3.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, ...
5769 [8.0, 1.0, 0.0, 4.0, 5.0, 3.0, 1.0, 0.0, 0.0, ...
2996 [1.0, 3.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, ...
18231 [1.0, 1.0, 0.0, 1.0, 1.0, 2.0, 1.0, 0.0, 0.0, ...
...
3976 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
16011 [5.0, 3.0, 1.0, 2.0, 3.0, 2.0, 1.0, 1.0, 0.0, ...
30158 [1.0, 0.0, 0.0, 4.0, 0.0, 0.0, 1.0, 0.0, 0.0, ...
24756 [1.0, 1.0, 1.0, 29.0, 4.0, 12.0, 3.0, 0.0, 0.0...
9744 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 2.0, 0.0, 0.0, ...
6742 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
29164 [0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 2.0, ...

29164 [0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 2.0, ...
22397 [1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 2.0, 0.0, 0.0, ...
23677 [0.0, 1.0, 0.0, 0.0, 3.0, 0.0, 0.0, 1.0, 1.0, ...
17697 [3.0, 5.0, 0.0, 2.0, 6.0, 1.0, 0.0, 0.0, 0.0, ...
9816 [0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, ...
25184 [0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 3.0, 0.0, 0.0, ...
10549 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
28306 [4.0, 3.0, 1.0, 11.0, 2.0, 4.0, 1.0, 5.0, 18.0...
9630 [4.0, 2.0, 0.0, 2.0, 1.0, 0.0, 1.0, 3.0, 1.0, ...
22319 [0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, ...
22338 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
3762 [3.0, 1.0, 0.0, 3.0, 2.0, 0.0, 0.0, 0.0, 1.0, ...
713 [3.0, 3.0, 0.0, 14.0, 0.0, 1.0, 0.0, 0.0, 0.0,...
19414 [0.0, 2.0, 0.0, 1.0, 4.0, 2.0, 0.0, 0.0, 0.0, ...
20956 [6.0, 2.0, 3.0, 4.0, 3.0, 0.0, 0.0, 0.0, 1.0, ...
14044 [2.0, 3.0, 0.0, 6.0, 0.0, 0.0, 0.0, 0.0, 1.0, ...
29172 [1.0, 2.0, 4.0, 1.0, 0.0, 0.0, 0.0, 2.0, 0.0, ...
29737 [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, ...
12776 [1.0, 4.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, ...
2003 [0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 2.0, ...
4296 [1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
18732 [6.0, 8.0, 2.0, 0.0, 2.0, 0.0, 0.0, 1.0, 3.0, ...
10326 [4.0, 2.0, 4.0, 6.0, 3.0, 1.0, 0.0, 1.0, 2.0, ...
3600 [4.0, 3.0, 0.0, 0.0, 0.0, 0.0, 0.0, 3.0, 0.0, ...
Name: Semantic, Length: 18669, dtype: object
```

In [ ]:

```
print(type(x_train))
print(x_train.shape)

In [ ]:
#Initialize the `tfidf_vectorizer`
tfidf2_vectorizer =
TfidfVectorizer(stop_words='english', ngram_range =
(1,1))
# Fit and transform the training data
tfidf2_train =
tfidf2_vectorizer.fit_transform(x_train.astype('str'))
# Transform the test set
tfidf2_test =
tfidf2_vectorizer.transform(x_test.astype('str'))
pickle.dump(tfidf2_train, open("tfidf2_train.pickle",
"wb"))
pickle.dump(tfidf2_test, open("tfidf2_test.pickle",
"wb"))


In [ ]:
clf = MultinomialNB()
#type(x_train.tolist())
clf.fit(x_train.tolist(), y_train)
pickle.dump(clf, open('sem_nb', 'wb'))
pred = clf.predict(x_test.tolist())
score = metrics.accuracy_score(y_test, pred)
```

```
print("Accuracy with Multinomial Naive Bayes: %0.3f"
% score)
In [ ]:
clf = RandomForestClassifier()
clf.fit(x_train.tolist(), y_train)
pickle.dump(clf, open('sem_rf', 'wb'))
pred = clf.predict(x_test.tolist())
score = metrics.accuracy_score(y_test, pred)
print("Accuracy with RandomForestClassifier: %0.3f" %
score)


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


In [ ]:

clf = GradientBoostingClassifier()
clf.fit(x_train.tolist(), y_train)
pickle.dump(clf, open('sem_gb', 'wb'))
pred = clf.predict(x_test.tolist())
score = metrics.accuracy_score(y_test, pred)
print("Accuracy with Gradient Boosting: %0.3f" %
score)

In [58]:
```

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


In [ ]:
#Combining the 3 feature vectors
import scipy.sparse as sp
# ui = sp.vstack(tfidf_train, tfidf1_train)
# yu = tfidf_train.data.tolist()
# yu.append(tfidf1_train.tolist())
# test = tfidf_test.data.tolist() + x_test.tolist()
#print(type(tfidf_train), tfidf_train.shape)
#print(type(tfidf1_train), tfidf1_train.shape)
# print(type(x_train), x_train.shape)
diff_n_rows = tfidf_train.shape[0] -
tfidf1_train.shape[0]
Xb_new = sp.vstack((tfidf1_train,
sp.csr_matrix((diff_n_rows, tfidf1_train.shape[1]))))
#where diff_n_rows is the difference of the number of
rows between Xa and Xb
c = sp.hstack((tfidf_train, Xb_new))
diff_n_rows = c.shape[0] - tfidf2_train.shape[0]
Xb_new = sp.vstack((tfidf2_train,
sp.csr_matrix((diff_n_rows, tfidf2_train.shape[1]))))
#where diff_n_rows is the difference of the number of
rows between Xa and Xb
X = sp.hstack((c, Xb_new))
```

```
X
dif_n_rows = tfidf_test.shape[0] - tfidf1_test.shape[0]
Xb_ne = sp.vstack((tfidf1_test,
sp.csr_matrix((dif_n_rows, tfidf1_test.shape[1]))))
#where diff_n_rows is the difference of the number of
rows between Xa and Xb
d = sp.hstack((tfidf_test, Xb_ne))
dif_n_rows = d.shape[0] - tfidf2_test.shape[0]
Xb_ne = sp.vstack((tfidf2_test,
sp.csr_matrix((dif_n_rows, tfidf2_test.shape[1]))))
#where diff_n_rows is the difference of the number of
rows between Xa and Xb
Y = sp.hstack((d, Xb_ne))
```

In [ ]:

```
clf = MultinomialNB()
#print(type(train), type(y_train.tolist()))
clf.fit(X, y_train)
pickle.dump(clf, open('pos_sem_nb', 'wb'))
pred = clf.predict(Y)
score = metrics.accuracy_score(y_test, pred)
print("Accuracy with Multinomial Naive Bayes: %0.3f"
% score)
```

In [ ]:

```
clf = RandomForestClassifier()
clf.fit(X, y_train)
pickle.dump(clf, open('pos_sem_rf', 'wb'))
```

```
pred = clf.predict(Y)
score = metrics.accuracy_score(y_test, pred)
print("Accuracy with RandomForestClassifier: %0.3f" %
score)
```

In [ ]:

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

In [174]:

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

In [175]:

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

In [164]:

```
#Directly loading the final dateframe by loading the
pickle file from the previously save
```

```
d pickle file
df = pd.read_pickle('Semantic.pkl')
print(df.columns)
print(df.shape)

In [165]:
y = df.Label
x_train, x_test, y_train, y_test = train_test_split(df,y,
test_size=0.33)

In [166]:
x_train_text = x_train['Text']
x_test_text = x_test['Text']
x_train_text_pos = x_train['Text_pos']
x_test_text_pos = x_test['Text_pos']
x_train_semantics = x_train['Semantics']
x_test_semantics = x_test['Semantics']

In [167]:
#Tf-idf Bigrams
#Initialize the `tfidf_vectorizer`
tfidf_vectorizer = TfidfVectorizer(stop_words='english',
ngram_range = (2,2), max_featur
es = 20000)
# Fit and transform the training data
tfidf1_train =
tfidf_vectorizer.fit_transform(x_train_text.astype('str'))
# Transform the test set
```

```
tfidf1_test =
tfidf_vectorizer.transform(x_test_text.astype('str'))
pickle.dump(tfidf1_train, open("tfidf1_train.pickle",
"wb"))
pickle.dump(tfidf1_test, open("tfidf1_test.pickle",
"wb"))
```

In [168]:

```
#POS
#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_text_pos.astype('
str'))
# Transform the test set
tfidf_test =
tfidf_vectorizer.transform(x_test_text_pos.astype('str')
)
pickle.dump(tfidf_train, open("tfidf_train.pickle",
"wb"))
pickle.dump(tfidf_test, open("tfidf_test.pickle", "wb"))
```

In [169]:

```
#Initialize the `tfidf_vectorizer`
tfidf_vectorizer = TfidfVectorizer(stop_words='english',
ngram_range = (1,1))
```

```
# Fit and transform the training data
tfidf2_train =
tfidf_vectorizer.fit_transform(x_train_semantics.astype
('str'))
# Transform the test set
tfidf2_test =
tfidf_vectorizer.transform(x_test_semantics.astype('str
'))
```

```
Index(['Headline', 'Body', 'Label', 'headline_length',
'body_length',
'Body_pos', 'Text_pos', 'Text', 'Semantic', 'Semantics'],
dtype='object')
(27865, 10)
```

```
pickle.dump(tfidf2_train, open("tfidf_train.pickle",
"wb"))
pickle.dump(tfidf2_test, open("tfidf_test.pickle",
"wb"))
```

In [170]:

```
ttf1_train = tfidf1_train
ttf1_test = tfidf1_test
ttf_train = tfidf_train
ttf_test = tfidf_test
ttf2_train = tfidf2_train
ttf2_test = tfidf2_test
```

In [218]:

```
#Giving weights to each of the 3 feature vectors
generated
big_w = 0.35
synt_w = 0.5
sem_w = 0.15
big_w *= 3
synt_w *= 3
sem_w *= 3
tfidf1_train = big_w*ttf1_train
tfidf1_test = big_w*ttf1_test
tfidf_train = synt_w*ttf_train
tfidf_test = synt_w*ttf_test
tfidf2_train = sem_w*ttf2_train
tfidf2_test = sem_w*ttf2_test
```

In [219]:

```
import scipy.sparse as sp
# ui = sp.vstack(tfidf_train, tfidf1_train)
# yu = tfidf_train.data.tolist()
# yu.append(tfidf1_train.tolist())
# test = tfidf_test.data.tolist() + x_test.tolist()
#print(type(tfidf_train), tfidf_train.shape)
#print(type(tfidf1_train), tfidf1_train.shape)
# print(type(x_train), x_train.shape)
diff_n_rows = tfidf_train.shape[0] -
tfidf1_train.shape[0]
Xb_new = sp.vstack((tfidf1_train,
sp.csr_matrix((diff_n_rows, tfidf1_train.shape[1]))))
```

```
#where diff_n_rows is the difference of the number of
rows between Xa and Xb
c = sp.hstack((tfidf_train, Xb_new))
diff_n_rows = c.shape[0] - tfidf2_train.shape[0]
Xb_new = sp.vstack((tfidf2_train,
sp.csr_matrix((diff_n_rows, tfidf2_train.shape[1]))))
#where diff_n_rows is the difference of the number of
rows between Xa and Xb
X = sp.hstack((c, Xb_new))
X
dif_n_rows = tfidf_test.shape[0] - tfidf1_test.shape[0]
Xb_ne = sp.vstack((tfidf1_test,
sp.csr_matrix((dif_n_rows, tfidf1_test.shape[1]))))
#where diff_n_rows is the difference of the number of
rows between Xa and Xb
d = sp.hstack((tfidf_test, Xb_ne))
dif_n_rows = d.shape[0] - tfidf2_test.shape[0]
Xb_ne = sp.vstack((tfidf2_test,
sp.csr_matrix((dif_n_rows, tfidf2_test.shape[1]))))
#where diff_n_rows is the difference of the number of
rows between Xa and Xb
Y = sp.hstack((d, Xb_ne))
```

In [ ]:

```
#Tf-idf Bigrams
#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_text_pos.astype('
str'))
# Transform the test set
tfidf_test = tfidf_vectorizer.transform([x_test])

In [ ]:
categories = []
a = lexicon.analyze("")
for key, value in a.items():
categories.append(key)
categories
lexicon = Empath()
semantic = []
cnt = 0
d = lexicon.analyze(x_test)
ds
em = []
for key,value in d.items() :
sem.append(value)
a = []
for j in range(len(sem)):
for k in range(int(sem[j])):
a.append(categories[j])
b = " ".join(a)
b
```

```
In [ ]:
#Initialize the `tfidf_vectorizer`
tfidf_vectorizer = TfidfVectorizer(stop_words='english',
ngram_range = (1,1))
# Fit and transform the training data
tfidf2_train =
tfidf_vectorizer.fit_transform(x_train_semantics.astype
('str'))
# Transform the test set
tfidf2_test = tfidf_vectorizer.transform([b])

In [ ]:
import scipy.sparse as sp
# ui = sp.vstack(tfidf_train, tfidf1_train)
# yu = tfidf_train.data.tolist()
# yu.append(tfidf1_train.tolist())
# test = tfidf_test.data.tolist() + x_test.tolist()
#print(type(tfidf_train), tfidf_train.shape)
#print(type(tfidf1_train), tfidf1_train.shape)
# print(type(x_train), x_train.shape)
diff_n_rows = tfidf_train.shape[0] -
tfidf1_train.shape[0]
Xb_new = sp.vstack((tfidf1_train,
sp.csr_matrix((diff_n_rows, tfidf1_train.shape[1]))))
#where diff_n_rows is the difference of the number of
rows between Xa and Xb
c = sp.hstack((tfidf_train, Xb_new))
diff_n_rows = c.shape[0] - tfidf2_train.shape[0]
```

```
Xb_new = sp.vstack((tfidf2_train,
sp.csr_matrix((diff_n_rows, tfidf2_train.shape[1]))))
#where diff_n_rows is the difference of the number of
rows between Xa and Xb
X = sp.hstack((c, Xb_new))
X
dif_n_rows = tfidf_test.shape[0] - tfidf1_test.shape[0]
Xb_ne = sp.vstack((tfidf1_test,
sp.csr_matrix((dif_n_rows, tfidf1_test.shape[1]))))
#where diff_n_rows is the difference of the number of
rows between Xa and Xb
d = sp.hstack((tfidf_test, Xb_ne))
dif_n_rows = d.shape[0] - tfidf2_test.shape[0]
Xb_ne = sp.vstack((tfidf2_test,
sp.csr_matrix((dif_n_rows, tfidf2_test.shape[1]))))
#where diff_n_rows is the difference of the number of
rows between Xa and Xb
Y = sp.hstack((d, Xb_ne))
```

In [ ]:

```
clf = MultinomialNB()
#type(x_train.tolist())
clf.fit(X, y_train)
clf.predict(Y)
```