In [1]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly re
mount, call drive.mount("/content/drive", force_remount=True).
```

Import packages

In [0]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
import re
import pickle
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix
, accuracy_score
```

Dataset propocessing

1.Read the dataset

In [0]:

```
path = "drive/My Drive/Eluvio/Eluvio_DS_Challenge.csv"
df = pd.read_csv(path)
```

```
In [4]: df.head()
```

Out[4]:

	time_created	date_created	up_votes	down_votes	title	over_18	author	cate
0	1201232046	2008-01-25	3	0	Scores killed in Pakistan clashes	False	polar	world
1	1201232075	2008-01-25	2	0	Japan resumes refuelling mission	False	polar	world
2	1201232523	2008-01-25	3	0	US presses Egypt on Gaza border	False	polar	world
3	1201233290	2008-01-25	1	0	Jump-start economy: Give health care to all	False	fadi420	world
4	1201274720	2008-01-25	4	0	Council of Europe bashes EU&UN terror blacklist	False	mhermans	world

```
In [5]: len(df)
```

```
Out[5]: 509236
```

```
In [6]: print(sum(df['category'] == "worldnews"))
        print(sum(df["down_votes"] == 0))
```

```
509236
509236
```

All category is "worldnews" and all "down_votes" are 0, so dropped

```
In [0]: df = df.drop("category", axis = 1)
        df = df.drop("down_votes", axis = 1)
        df = df.drop("time_created", axis = 1)
        df = df.drop("date_created", axis = 1)
```

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

Out[8]:

	up_votes	title	over_18	author
0	3	Scores killed in Pakistan clashes	False	polar
1	2	Japan resumes refuelling mission	False	polar
2	3	US presses Egypt on Gaza border	False	polar
3	1	Jump-start economy: Give health care to all	False	fadi420
4	4	Council of Europe bashes EU&UN terror blacklist	False	mhermans

```
In [9]: len(set(df['author']))  # the number of author

Out[9]: 85838
```

2.Process the title(word vectorize)

```
In [10]: import nltk
         nltk.download('punkt')
         nltk.download('stopwords')

         [nltk_data] Downloading package punkt to /root/nltk_data...
         [nltk_data]   Package punkt is already up-to-date!
         [nltk_data] Downloading package stopwords to /root/nltk_data...
         [nltk_data]   Package stopwords is already up-to-date!

Out[10]: True
```

2.1build the corpus

```
In [0]: from nltk.stem.snowball import SnowballStemmer
        stemmer = SnowballStemmer("english")
```

In [0]:

```python
# To get the stems of words in a sentence.
def tokenize_and_stem(text):
    # first tokenize by sentence, then by word to ensure that punctuation is caught as it's own token
    tokens = [word for sent in nltk.sent_tokenize(text) for word in nltk.word_tokenize(sent)]
    filtered_tokens = []
    for token in tokens:
        if re.search('[a-zA-Z]', token):
            filtered_tokens.append(token)

    stems = [stemmer.stem(t) for t in filtered_tokens]
    return stems

# To get the words themself in a sentence.
def tokenize_only(text):
    # first tokenize by sentence, then by word to ensure that punctuation is caught as it's own token
    tokens = [word for sent in nltk.sent_tokenize(text) for word in nltk.word_tokenize(sent)]
    filtered_tokens = []
    for token in tokens:
        if re.search('[a-zA-Z]', token):
            filtered_tokens.append(token)
    return filtered_tokens
```

In [0]:

```python
#lowercase
title = df.title.str.lower()
```

In [0]:

```python
# Get full stems and tokens to build vocabulary
def tokenized_stemmed(title):
    totalvocab_stemmed = []
    totalvocab_tokenized = []
    for i in title:
        allwords_stemmed = tokenize_and_stem(i)
        totalvocab_stemmed.extend(allwords_stemmed)

        allwords_tokenized = tokenize_only(i)
        totalvocab_tokenized.extend(allwords_tokenized)
    return totalvocab_stemmed, totalvocab_tokenized
```

In [0]:

```python
totalvocab_stemmed_, totalvocab_tokenized_ = tokenized_stemmed(title)
```

In [16]:

```python
print(len(totalvocab_stemmed_))
```

```
7194561
```

In [0]:

```
# pickle.dump((totalvocab_stemmed_, totalvocab_tokenized_), open("drive/My Drive/Eluvio/stem_token_.pkl", "wb" ))
totalvocab_stemmed_, totalvocab_tokenized_ = pickle.load(open("drive/My Drive/Eluvio/stem_token_.pkl", "rb" ))
```

In [0]:

```
# Rule out repetitions of stem-token pairs
# totalvocab = zip(totalvocab_stemmed_, totalvocab_tokenized_)
# totalvocab = list(set(totalvocab))
# totalvocab_stemmed, totalvocab_tokenized = zip(*totalvocab)

# pickle.dump((totalvocab_stemmed, totalvocab_tokenized), open("drive/My Drive/Eluvio/stem_token.pkl", "wb" ))

totalvocab_stemmed, totalvocab_tokenized = pickle.load(open("drive/My Drive/Eluvio/stem_token.pkl", "rb" ))
```

In [19]:

```
print(len(totalvocab_stemmed))
```

```
115041
```

In [0]:

```
#stem-token vocabulary
# vocab_frame = pd.DataFrame({'words': totalvocab_tokenized}, index = totalvocab_stemmed)

# pickle.dump(vocab_frame, open('drive/My Drive/Eluvio/vocab_frame.pkl','wb'))

vocab_frame = pickle.load(open('drive/My Drive/Eluvio/vocab_frame.pkl','rb'))
```

In [0]:

```
# Build stopwords set. Combine two common set.
import sklearn.feature_extraction.text as text
stopwords = nltk.corpus.stopwords.words('english')
my_stop_words = text.ENGLISH_STOP_WORDS.union(stopwords)
```

2.2 Tf-idf to vectorize text

In [22]:

```
# tf-idf vectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer(min_df =10**-3 ,analyzer = 'word', max_features=len(set(totalvocab_stemmed)), stop_words=my_stop_words, tokenizer=tokenize_and_stem, ngram_range=(1,3))

tfidf_matrix = tfidf_vectorizer.fit_transform(title)

print(tfidf_matrix.shape)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/feature_extraction/
text.py:300: UserWarning: Your stop_words may be inconsistent with
your preprocessing. Tokenizing the stop words generated tokens ["'
d", "'s", 'abov', 'afterward', 'alon', 'alreadi', 'alway', 'ani',
'anoth', 'anyon', 'anyth', 'anywher', 'becam', 'becaus', 'becom',
'befor', 'besid', 'cri', 'describ', 'doe', 'dure', 'els', 'elsewhe
r', 'empti', 'everi', 'everyon', 'everyth', 'everywher', 'fifti',
'forti', 'henc', 'hereaft', 'herebi', 'howev', 'hundr', 'inde', 'm
ani', 'meanwhil', 'moreov', "n't", 'need', 'nobodi', 'noon', 'noth
', 'nowher', 'onc', 'onli', 'otherwis', 'ourselv', 'perhap', 'plea
s', 'sever', 'sha', 'sinc', 'sincer', 'sixti', 'someon', 'someth',
'sometim', 'somewher', 'themselv', 'thenc', 'thereaft', 'therebi',
'therefor', 'togeth', 'twelv', 'twenti', 'veri', 'whatev', 'whenc'
, 'whenev', 'wherea', 'whereaft', 'wherebi', 'wherev', 'whi', 'wo'
, 'yourselv'] not in stop_words.
  'stop_words.' % sorted(inconsistent))

(509236, 1814)
```

In [0]:

```
# pickle.dump(tfidf_matrix, open("drive/My Drive/Eluvio/tfidf_matrix.pkl", "wb" ))
# pickle.dump(tfidf_vectorizer, open( "drive/My Drive/Eluvio/tfidf_vectorizer.pkl", "wb" ))

tfidf_matrix = pickle.load(open("drive/My Drive/Eluvio/tfidf_matrix.pkl", "rb" ))
# tfidf_vectorizer = pickle.load(open("drive/My Drive/Eluvio/tfidf_vectorizer.pkl", "rb" ))
```

In [25]:

```
tfidf_matrix
```

Out[25]:

```
<509236x1814 sparse matrix of type '<class 'numpy.float64'>'
        with 3565328 stored elements in Compressed Sparse Row form
at>
```

Model

```
In [0]: thre = np.quantile(df['up_votes'], 0.8)
        y = [1 if i > three else 0 for i in df['up_votes']]
        y = np.array(y)
        X_train, X_test, y_train, y_test = train_test_split(tfidf_matrix, y
        , test_size = 0.2, shuffle = True, random_state = 42)
```

MultinomailNB

```
In [27]: clf = MultinomialNB()
         clf.fit(X_train, y_train)

Out[27]: MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

```
In [28]: y_predict = clf.predict(X_test)
         clf.score(X_test, y_test)

Out[28]: 0.8050624459979577
```

```
In [29]: print(classification_report(y_test, y_predict))

                       precision    recall  f1-score   support

                   0       0.81      1.00      0.89     81988
                   1       0.56      0.00      0.00     19860

            accuracy                           0.81    101848
           macro avg       0.68      0.50      0.45    101848
        weighted avg       0.76      0.81      0.72    101848
```

LogisticRegression

```
In [0]: LR = LogisticRegression(C=1.0, penalty='l1', tol=0.01)
In [31]: LR.fit(X_train, y_train)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logist
ic.py:432: FutureWarning: Default solver will be changed to 'lbfgs
' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

```
Out[31]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_inter
cept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=10
0,
                   multi_class='warn', n_jobs=None, penalty='l1',
                   random_state=None, solver='warn', tol=0.01, ver
bose=0,
                   warm_start=False)
```

```
In [32]: y_predict = LR.predict(X_test)
         LR.score(X_test, y_test)
```

```
Out[32]: 0.8061719425025529
```

```
In [33]: print(classification_report(y_test, y_predict))
```

```
              precision    recall  f1-score   support

           0       0.81      0.99      0.89     81988
           1       0.55      0.04      0.07     19860

    accuracy                           0.81    101848
   macro avg       0.68      0.51      0.48    101848
weighted avg       0.76      0.81      0.73    101848
```

GBDT

In [34]:

```
gbdt = GradientBoostingClassifier()
gbdt.fit(X_train, y_train)
```

Out[34]:

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max
_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity
_split=None,
                           min_samples_leaf=1, min_samples_split=2
,
                           min_weight_fraction_leaf=0.0, n_estimat
ors=100,
                           n_iter_no_change=None, presort='auto',
                           random_state=None, subsample=1.0, tol=0
.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

In [35]:

```
y_predict = gbdt.predict(X_test)
gbdt.score(X_test, y_test)
```

Out[35]:

```
0.8054257324640641
```

In [36]:

```
print(classification_report(y_test, y_predict))
```

```
              precision    recall  f1-score   support

           0       0.81      1.00      0.89     81988
           1       0.77      0.00      0.01     19860

    accuracy                           0.81    101848
   macro avg       0.79      0.50      0.45    101848
weighted avg       0.80      0.81      0.72    101848
```

Random Forest

In [37]:
```
rfc = RandomForestClassifier(n_jobs = -1, max_features = 'sqrt', n_
estimators = 10, oob_score = True)
rfc.fit(X_train, y_train)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:
460: UserWarning: Some inputs do not have OOB scores. This probabl
y means too few trees were used to compute any reliable oob estima
tes.
  warn("Some inputs do not have OOB scores. "
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:
465: RuntimeWarning: invalid value encountered in true_divide
  predictions[k].sum(axis=1)[:, np.newaxis])
```

Out[37]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterio
n='gini',
                       max_depth=None, max_features='sqrt', max_le
af_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_spl
it=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=
10, n_jobs=-1,
                       oob_score=True, random_state=None, verbose=
0,
                       warm_start=False)
```

In [38]:
```
y_predict = rfc.predict(X_test)
rfc.score(X_test, y_test)
```

Out[38]:
```
0.7927107061503417
```

In [39]:
```
print(classification_report(y_test, y_predict))
```

```
              precision    recall  f1-score   support

           0       0.81      0.97      0.88     81988
           1       0.30      0.05      0.08     19860

    accuracy                           0.79    101848
   macro avg       0.56      0.51      0.48    101848
weighted avg       0.71      0.79      0.73    101848
```

XGB

In [0]:
```
import xgboost as xgb
from xgboost.sklearn import XGBClassifier
```

In [0]:
```
xgb = XGBClassifier(
 learning_rate =0.1,
 n_estimators=1000,
 max_depth=5,
 min_child_weight=1,
 gamma=0,
 subsample=0.8,
 colsample_bytree=0.8,
 objective= 'binary:logistic',
 nthread=4,
 scale_pos_weight=1,
 seed=27)
```

In [42]:
```
xgb.fit(X_train, y_train)
```

Out[42]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=
1,
              colsample_bynode=1, colsample_bytree=0.8, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=5,
              min_child_weight=1, missing=None, n_estimators=1000,
n_jobs=1,
              nthread=4, objective='binary:logistic', random_state
=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=
27,
              silent=None, subsample=0.8, verbosity=1)
```

In [0]:
```
y_predict = xgb.predict(X_test)
```

In [44]:
```
xgb.score(X_test, y_test)
```

Out[44]:
```
0.8062406723745189
```

In [45]:
```
print(classification_report(y_test, y_predict))
```

```
              precision    recall  f1-score   support

           0       0.81      0.99      0.89     81988
           1       0.54      0.04      0.08     19860

    accuracy                           0.81    101848
   macro avg       0.68      0.52      0.48    101848
weighted avg       0.76      0.81      0.73    101848
```

In [0]:
```

```