read data

In [1]:

```
import pandas as pd
import numpy as np
import os
import time
import numpy as np
import scipy.io
import sklearn.metrics
import sklearn
import random
import pandas as pd


# from scipy.spatial.distance import pdist
# import imbalanced_databases as imbd
import matplotlib.pyplot as plt
# import smote_variants as s
import scipy.io as scio
from PIL import Image
import pandas as pd
import numpy as np
import xlsxwriter
import scipy.io
import sklearn
import os, sys
import pickle
import random
import time
import cv2

from sklearn.pipeline import Pipeline
from sklearn.datasets import make_classification
from sklearn.metrics import (classification_report,
                             confusion_matrix,
                             recall_score,
                             accuracy_score,
                             make_scorer,
                             roc_auc_score)

from sklearn.model_selection import (train_test_split,
                                     cross_validate,
                                     GridSearchCV,
                                     RepeatedStratifiedKFold,
                                     cross_val_score,
                                     validation_curve)

# dealing with imbalanced dataset
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE

# models
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.ensemble import (GradientBoostingClassifier,
                              RandomForestClassifier,
                              AdaBoostClassifier,
                             VotingClassifier)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import RidgeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn import tree
from sklearn.neighbors import (KNeighborsClassifier,
                               NearestCentroid,
                               NeighborhoodComponentsAnalysis)


random.seed(2021)
```

In [2]:

```
df_raw=pd.read_csv('../output/tweets_with_topic_label.csv')
df_raw=df_raw.drop(columns=['text'])
df_raw['id']=df_raw['id'].astype('int')
df_raw.head()
```

Out[2]:

| | id | isRetweet | isDeleted | device | favorites | retweets | date | isFlagg |
|---|---|---|---|---|---|---|---|---|
| **0** | 98454970654916608 | f | f | TweetDeck | 49 | 255 | 2011-08-02 18:07:48 | |
| **1** | 1234653427789070336 | f | f | Twitter for iPhone | 73748 | 17404 | 2020-03-03 01:34:50 | |
| **2** | 1218010753434820614 | t | f | Twitter for iPhone | 0 | 7396 | 2020-01-17 03:22:47 | |
| **3** | 1304875170860015617 | f | f | Twitter for iPhone | 80527 | 23502 | 2020-09-12 20:10:58 | |
| **4** | 1218159531554897920 | t | f | Twitter for iPhone | 0 | 9081 | 2020-01-17 13:13:59 | |

In [3]:

```
df_senti=pd.read_csv('../output/sentiment_labels.csv',index_col=None)
df_senti['id']=df_senti['id'].astype('int')
df_senti.head()
```

Out[3]:

| | Unnamed: 0 | id | text | sentiment_text | subjectivity_score | VADER |
|---|---|---|---|---|---|---|
| **0** | 0 | 98454970654916608 | Republicans and Democrats have both created ou... | republicans democrats created economic problems | 0.200000 | -1 |
| **1** | 1 | 1234653427789070336 | I was thrilled to be back in the Great city of... | thrilled_back great city charlotte north_carol... | 0.483333 | 1 |

| | Unnamed: 0 | id | text | sentiment_text | subjectivity_score | VADER |
|---|---|---|---|---|---|---|
| **2** | 2 | 1218010753434820608 | RT @CBS_Herridge: READ: Letter to surveillance... | read letter surveillance court obtained cbs ne... | 0.100000 | 0 |
| **3** | 3 | 1304875170860015616 | The Unsolicited Mail In Ballot Scam is a major... | unsolicited mail_ballot scam major threat demo... | 0.454762 | -1 |
| **4** | 4 | 1218159531554897920 | RT @MZHemingway: Very friendly telling of even... | friendly telling events comey apparent leaking... | 0.425000 | 1 |

In [4]:
```
df=df_raw.set_index('id').join(df_senti.set_index('id'),how='left')
df=df.drop(columns=['Unnamed: 0','text','sentiment_text','VADER','TextBlob','W2V
df.head()
```

Out[4]:

| id | isRetweet | isDeleted | device | favorites | retweets | date | isFlagged | Topic | subje |
|---|---|---|---|---|---|---|---|---|---|
| **1698308935** | f | f | Twitter Web Client | 939 | 519 | 2009-05-04 18:54:25 | f | 3 | |
| **1701461182** | f | f | Twitter Web Client | 259 | 34 | 2009-05-05 01:00:10 | f | 1 | |
| **1737479987** | f | f | Twitter Web Client | 37 | 15 | 2009-05-08 13:38:08 | f | 3 | |
| **1741160716** | f | f | Twitter Web Client | 29 | 11 | 2009-05-08 20:40:15 | f | 1 | |
| **1773561338** | f | f | Twitter Web Client | 1877 | 1321 | 2009-05-12 14:07:28 | f | 2 | |

In [5]:
```
df['isDeleted'].value_counts()
# imbalanced data set, consider using smote?
```

Out[5]:
```
f    55480
t     1092
Name: isDeleted, dtype: int64
```

In [6]:
```
from sklearn import preprocessing
df=df.apply(preprocessing.LabelEncoder().fit_transform)
df.head()
```

Out[6]:

| | isRetweet | isDeleted | device | favorites | retweets | date | isFlagged | Topic | subjectiv |
|---|---|---|---|---|---|---|---|---|---|

| id | isRetweet | isDeleted | device | favorites | retweets | date | isFlagged | Topic | subjectiv |
|---|---|---|---|---|---|---|---|---|---|
| **id** | | | | | | | | | |
| **1698308935** | 0 | 0 | 13 | 904 | 519 | 0 | 0 | 3 | |
| **1701461182** | 0 | 0 | 13 | 259 | 34 | 1 | 0 | 1 | |
| **1737479987** | 0 | 0 | 13 | 37 | 15 | 2 | 0 | 3 | |
| **1741160716** | 0 | 0 | 13 | 29 | 11 | 3 | 0 | 1 | |
| **1773561338** | 0 | 0 | 13 | 1645 | 1301 | 4 | 0 | 2 | |

In [7]:
```
X=df.drop(columns=['isDeleted'])
Y=df['isDeleted']
print('majority train class: %d' % np.sum(Y == 0))
print('minority train class: %d' % np.sum(Y == 1))
```

```
majority train class: 55480
minority train class: 1092
```

In [8]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test= train_test_split(X,Y,test_size=0.20,random_state=
X_train.shape, Y_train.shape, X_test.shape, Y_test.shape
print('majority train class: %d' % np.sum(Y_train == 0))
print('minority train class: %d' % np.sum(Y_train == 1))
print('majority test class: %d' % np.sum(Y_test == 0))
print('minority test class: %d' % np.sum(Y_test == 1))
#imbalanced dataset
```

```
majority train class: 44387
minority train class: 870
majority test class: 11093
minority test class: 222
```

In [9]:
```
from imblearn.over_sampling import SMOTE
# using SMOTE
smt = SMOTE()

# fit and apply the transform
X_train, Y_train = smt.fit_resample(X_train, Y_train)
X_test, Y_test = smt.fit_resample(X_test, Y_test)

print('majority train class: %d' % np.sum(Y_train == 0))
print('minority train class: %d' % np.sum(Y_train == 1))
print('majority test class: %d' % np.sum(Y_test == 0))
print('minority test class: %d' % np.sum(Y_test == 1))
```

```
majority train class: 44387
minority train class: 44387
majority test class: 11093
minority test class: 11093
```

In [10]:
```
y_train=Y_train
y_test=Y_test
```

## models

In [11]:
```
# kNN
from sklearn.neighbors import NearestCentroid
import numpy as np
from sklearn.metrics import classification_report

clf = NearestCentroid()
start_time=time.time()
clf.fit(X_train, Y_train)
NearestCentroid()
print("Training  model takes %s seconds" % round((time.time() - start_time),3))

start = time.time()
pre=clf.predict(X_test)
end = time.time()
print("Predicting test data takes %s seconds" % round((end - start),3))
print(classification_report(Y_test,pre))
```

```
Training  model takes 0.016 seconds
Predicting test data takes 0.005 seconds
              precision    recall  f1-score   support

           0       0.81      0.60      0.69     11093
           1       0.68      0.86      0.76     11093

    accuracy                           0.73     22186
   macro avg       0.75      0.73      0.73     22186
weighted avg       0.75      0.73      0.73     22186
```

In [12]:
```
# SGD with penalty=l1
from sklearn.linear_model import SGDClassifier

clf = SGDClassifier(loss="log", penalty="l1", max_iter=200, shuffle=True, class_
start_time=time.time()
clf.fit(X_train, Y_train)
print("Training  model takes %s seconds" % round((time.time() - start_time),3))

from sklearn.metrics import classification_report
start = time.time()
pre=clf.predict(X_test)
end = time.time()
print("Predicting test data takes %s seconds" % round((end - start),3))
print(classification_report(Y_test,pre))
```

```
Training  model takes 2.98 seconds
Predicting test data takes 0.003 seconds
              precision    recall  f1-score   support

           0       0.87      0.43      0.57     11093
           1       0.62      0.94      0.75     11093

    accuracy                           0.68     22186
   macro avg       0.74      0.68      0.66     22186
weighted avg       0.74      0.68      0.66     22186

/Users/aprilyang/.local/lib/python3.8/site-packages/sklearn/linear_model/_stocha
stic_gradient.py:570: ConvergenceWarning: Maximum number of iteration reached be
fore convergence. Consider increasing max_iter to improve the fit.
  warnings.warn("Maximum number of iteration reached before "
```

In [13]:
```
# SGD with penalty=l2
from sklearn.linear_model import SGDClassifier

clf = SGDClassifier(loss="log", penalty="l2", max_iter=200, shuffle=True, class_
start_time=time.time()
clf.fit(X_train, Y_train)
print("Training  model takes %s seconds" % round((time.time() - start_time),3))

from sklearn.metrics import classification_report
start = time.time()
pre=clf.predict(X_test)
end = time.time()
print("Predicting test data takes %s seconds" % round((end - start),3))
print(classification_report(Y_test,pre))
```

```
Training  model takes 2.202 seconds
Predicting test data takes 0.003 seconds
              precision    recall  f1-score   support

           0       0.49      0.97      0.65     11093
           1       0.00      0.00      0.00     11093

    accuracy                           0.48     22186
   macro avg       0.25      0.48      0.33     22186
weighted avg       0.25      0.48      0.33     22186

/Users/aprilyang/.local/lib/python3.8/site-packages/sklearn/linear_model/_stocha
stic_gradient.py:570: ConvergenceWarning: Maximum number of iteration reached be
fore convergence. Consider increasing max_iter to improve the fit.
  warnings.warn("Maximum number of iteration reached before "
```

In [14]:
```
# DecisionTrees
from sklearn import tree

clf = tree.DecisionTreeClassifier(max_depth = 30, min_samples_leaf=2, max_leaf_n
start_time=time.time()
clf = clf.fit(X_train, Y_train)
print("Training  model takes %s seconds" % round((time.time() - start_time),3))

from sklearn.metrics import classification_report
start = time.time()
pre=clf.predict(X_test)
end = time.time()
print("Predicting test data takes %s seconds" % round((end - start),3))
print(classification_report(Y_test,pre))
```

```
Training  model takes 0.159 seconds
Predicting test data takes 0.003 seconds
              precision    recall  f1-score   support

           0       0.83      0.81      0.82     11093
           1       0.82      0.83      0.83     11093

    accuracy                           0.82     22186
   macro avg       0.82      0.82      0.82     22186
weighted avg       0.82      0.82      0.82     22186
```

In [15]:
```
# RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier
```

```
clf = RandomForestClassifier(n_estimators=300, class_weight='balanced')
start_time=time.time()
clf = clf.fit(X_train, Y_train)
print("Training  model takes %s seconds" % round((time.time() - start_time),3))

from sklearn.metrics import classification_report
start = time.time()
pre=clf.predict(X_test)
end = time.time()
print("Predicting test data takes %s seconds" % round((end - start),3))
print(classification_report(Y_test,pre))
```

```
Training  model takes 39.3 seconds
Predicting test data takes 0.867 seconds
              precision    recall  f1-score   support

           0       0.72      0.98      0.83     11093
           1       0.97      0.62      0.76     11093

    accuracy                           0.80     22186
   macro avg       0.85      0.80      0.80     22186
weighted avg       0.85      0.80      0.80     22186
```

In [16]:

```
# AdaBoostClassifier
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import AdaBoostClassifier

clf = AdaBoostClassifier(n_estimators=500)
start_time=time.time()
clf = clf.fit(X_train, Y_train)
print("Training  model takes %s seconds" % round((time.time() - start_time),3))

from sklearn.metrics import classification_report
start=time.time()
pre=clf.predict(X_test)
end = time.time()
print("Predicting test data takes %s seconds" % round((end - start),3))
print(classification_report(Y_test,pre))
```

```
Training  model takes 32.648 seconds
Predicting test data takes 1.098 seconds
              precision    recall  f1-score   support

           0       0.89      0.85      0.87     11093
           1       0.86      0.89      0.87     11093

    accuracy                           0.87     22186
   macro avg       0.87      0.87      0.87     22186
weighted avg       0.87      0.87      0.87     22186
```

In [17]:

```
# GBM

# define the model
def train_model_gb(X, y):
    model_gb = GradientBoostingClassifier(n_estimators=500)
    model_gb.fit(X, y)

    return model_gb
```

```
# train
training_gbm = time.time()
model_gb = train_model_gb(X_train, Y_train)

print("Train the Gradient Boosting Model takes %s seconds" % round((time.time()

y_pred = model_gb.predict(X_test)

# print the confusion matrix
print(confusion_matrix(y_true=Y_test, y_pred=y_pred))
print(classification_report(y_true=Y_test, y_pred=y_pred))
```

```
Train the Gradient Boosting Model takes 63.676 seconds
[[10026  1067]
 [ 1292  9801]]
              precision    recall  f1-score   support

           0       0.89      0.90      0.89     11093
           1       0.90      0.88      0.89     11093

    accuracy                           0.89     22186
   macro avg       0.89      0.89      0.89     22186
weighted avg       0.89      0.89      0.89     22186
```

In [16]:

```
weighted_svm_best = SVC(
gamma = 'scale',
class_weight = {
    0: 1092.0,
    1: 55480.0
},
probability=True
)

start_time = time.time()

# fit svm model
weighted_svm_best.fit(X_train, y_train)

print("Training  model takes %s seconds" % round((time.time() - start_time),3))
print('Testing Accuracy of weighted SVM on test set: {:.3f}'
    .format(weighted_svm_best.score(X_test,y_test)))

start = time.time()

# make prediction
weighted_svm_pred = weighted_svm_best.predict(X_test)

end = time.time()

weighted_svm_predprob = weighted_svm_best.predict_proba(X_test)[:,1]

print("Predicting test data takes %s seconds" % round((end - start),3))
print('Classification error rate:', np.mean(np.array(y_test) != weighted_svm_pre
print('Classification report \n', classification_report(y_test, weighted_svm_pre

print('Confusion Matrix \n', confusion_matrix(y_test, weighted_svm_pred))
print('AUC is: {:.4f}'.format(roc_auc_score(y_test, weighted_svm_predprob)))

# callModel(eclf2, 'eclf2', X_train, X_test, y_train, y_test)
```

In [ ]: