In [1]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
import string
import math
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os

import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
```

1.1 Reading Data

Splitting data into Train, Cross Validation and Test

In [2]:

```
prepeocessed_data = pd.read_csv('preprocessed_data.csv')
prepeocessed_data.head(2)
```

Out[2]:

	Unnamed: 0	Unnamed: 0.1	id	teacher_id	teacher
0	0	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	
1	1	37728	p043609	3f60494c61921b3b43ab61bdde2904df	

2 rows × 21 columns

In [3]:

```
y = prepeocessed_data['project_is_approved'].values
X = prepeocessed_data.drop(['project_is_approved'], axis=1)
X.shape
```

Out[3]:

```
(109248, 20)
```

In [4]:

```
# train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, str
X_test.shape
```

Out[4]:

```
(36052, 20)
```

1.4 Encoding Categorical and Numerical features

1.4.1 encoding categorical features: clean_categories

In [5]:

```
feature_names=[]
feature_names_tfidf=[]

vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_categories'].values) # fit has to happen only o

# we use the fitted CountVectorizer to convert the text to vector
X_train_cc_ohe = vectorizer.transform(X_train['clean_categories'].values)
#X_cv_cc_ohe = vectorizer.transform(X_cv['clean_categories'].values)
X_test_cc_ohe = vectorizer.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_cc_ohe.shape, y_train.shape)
#print(X_cv_cc_ohe.shape, y_cv.shape)
print(X_test_cc_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())


feature_names.extend(vectorizer.get_feature_names())
```

```
After vectorizations
(73196, 9) (73196,)
(36052, 9) (36052,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_c
ivics', 'literacy_language', 'math_science', 'music_arts', 'sp
ecialneeds', 'warmth']
```

1.4.2 encoding categorical features: clean_subcategories

In [6]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_subcategories'].values) # fit has to happen onl

# we use the fitted CountVectorizer to convert the text to vector
X_train_csc_ohe = vectorizer.transform(X_train['clean_subcategories'].values)
#X_cv_csc_ohe = vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_csc_ohe = vectorizer.transform(X_test['clean_subcategories'].values)

print("After vectorizations")
print(X_train_csc_ohe.shape, y_train.shape)
#print(X_cv_csc_ohe.shape, y_cv.shape)
print(X_test_csc_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())

feature_names.extend(vectorizer.get_feature_names())
```

```
After vectorizations
(73196, 30) (73196,)
(36052, 30) (36052,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civi
cs_government', 'college_careerprep', 'communityservice', 'ear
lydevelopment', 'economics', 'environmentalscience', 'esl', 'e
xtracurricular', 'financialliteracy', 'foreignlanguages', 'gym
_fitness', 'health_lifescience', 'health_wellness', 'history_g
eography', 'literacy', 'literature_writing', 'mathematics', 'm
usic', 'nutritioneducation', 'other', 'parentinvolvement', 'pe
rformingarts', 'socialsciences', 'specialneeds', 'teamsports',
'visualarts', 'warmth']
```

1.4.3 encoding categorical features: school_state

In [7]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['school_state'].values) # fit has to happen only on tr

# we use the fitted CountVectorizer to convert the text to vector
X_train_state_ohe = vectorizer.transform(X_train['school_state'].values)
#X_cv_state_ohe = vectorizer.transform(X_cv['school_state'].values)
X_test_state_ohe = vectorizer.transform(X_test['school_state'].values)

print("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)
#print(X_cv_state_ohe.shape, y_cv.shape)
print(X_test_state_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())

feature_names.extend(vectorizer.get_feature_names())
```

```
After vectorizations
(73196, 51) (73196,)
(36052, 51) (36052,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl',
'ga', 'hi', 'ia', 'id', 'il', 'in', 'ks', 'ky', 'la', 'ma', 'm
d', 'me', 'mi', 'mn', 'mo', 'ms', 'mt', 'nc', 'nd', 'ne', 'n
h', 'nj', 'nm', 'nv', 'ny', 'oh', 'ok', 'or', 'pa', 'ri', 's
c', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv', 'w
y']
```

1.4.4 encoding categorical features: teacher_prefix

In [8]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['teacher_prefix'].values) # fit has to happen only on

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_ohe = vectorizer.transform(X_train['teacher_prefix'].values)
#X_cv_teacher_ohe = vectorizer.transform(X_cv['teacher_prefix'].values)
X_test_teacher_ohe = vectorizer.transform(X_test['teacher_prefix'].values)

print("After vectorizations")
print(X_train_teacher_ohe.shape, y_train.shape)
#print(X_cv_teacher_ohe.shape, y_cv.shape)
print(X_test_teacher_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())

feature_names.extend(vectorizer.get_feature_names())
```

```
After vectorizations
(73196, 5) (73196,)
(36052, 5) (36052,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
```

1.4.5 encoding categorical features: project_grade_category

In [9]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['project_grade_category'].values) # fit has to happen

# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_ohe = vectorizer.transform(X_train['project_grade_category'].va
#X_cv_grade_ohe = vectorizer.transform(X_cv['project_grade_category'].values)
X_test_grade_ohe = vectorizer.transform(X_test['project_grade_category'].valu

print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
#print(X_cv_grade_ohe.shape, y_cv.shape)
print(X_test_grade_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())

feature_names.extend(vectorizer.get_feature_names())
```

```
After vectorizations
(73196, 4) (73196,)
(36052, 4) (36052,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
```

1.4.6 encoding numerical features: price

In [10]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['price'].values.reshape(1,-1))

X_train_price_norm = normalizer.transform(X_train['price'].values.reshape(1,-
#X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(1,-1))
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(1,-1)

print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)
#print(X_cv_price_norm.shape, y_cv.shape)
print(X_test_price_norm.shape, y_test.shape)
print(X_train_price_norm)
print((X_test_price_norm))

feature_names.extend(['price'])
```

```
After vectorizations
(1, 73196) (73196,)
(1, 36052) (36052,)
[[0.00254418 0.00330246 0.00248577 ... 0.00180197 0.00020543
0.00501878]]
[[0.00180586 0.00431706 0.00514938 ... 0.00215565 0.00299181
0.00708843]]
```

1.4.7 encoding numerical features: teacher_number_of_previously_posted_projects

In [11]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values

X_train_ppp_norm = normalizer.transform(X_train['teacher_number_of_previously
#X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(1,-1))
X_test_ppp_norm = normalizer.transform(X_test['teacher_number_of_previously_p

print("After vectorizations")
print(X_train_ppp_norm.shape, y_train.shape)
#print(X_cv_price_norm.shape, y_cv.shape)
print(X_test_ppp_norm.shape, y_test.shape)


feature_names.extend(['teacher_number_of_previously_posted_projects'])
feature_names_tfidf.extend(feature_names)
```

```
After vectorizations
(1, 73196) (73196,)
(1, 36052) (36052,)
```

1.5 Vectorizing Text features

1.5.1 Vectorizing using BOW

Essay

In [12]:

```
print(X_train.shape, y_train.shape)
#print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("\n\n")


vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['essay'].values) # fit has to happen only on train dat

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_bow = vectorizer.transform(X_train['essay'].values)
#X_cv_essay_bow = vectorizer.transform(X_cv['essay'].values)
X_test_essay_bow = vectorizer.transform(X_test['essay'].values)

print("After vectorizations")
print(X_train_essay_bow.shape, y_train.shape)
#print(X_cv_essay_bow.shape, y_cv.shape)
print(X_test_essay_bow.shape, y_test.shape)


feature_names.extend(vectorizer.get_feature_names())
```

```
(73196, 20) (73196,)
(36052, 20) (36052,)



After vectorizations
(73196, 5000) (73196,)
(36052, 5000) (36052,)
```

project_title

In [13]:

```
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['project_title'].values) # fit has to happen only on t

# we use the fitted CountVectorizer to convert the text to vector
X_train_titles_bow = vectorizer.transform(X_train['project_title'].values)
#X_cv_titles_bow = vectorizer.transform(X_cv['project_title'].values)
X_test_titles_bow = vectorizer.transform(X_test['project_title'].values)

print("After vectorizations")
print(X_train_titles_bow.shape, y_train.shape)
#print(X_cv_titles_bow.shape, y_cv.shape)
print(X_test_titles_bow.shape, y_test.shape)

feature_names.extend(vectorizer.get_feature_names())
```

```
After vectorizations
(73196, 5000) (73196,)
(36052, 5000) (36052,)
```

project_resource_summary

In [14]:

```
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['project_resource_summary'].values) # fit has to happe

# we use the fitted CountVectorizer to convert the text to vector
X_train_psr_bow = vectorizer.transform(X_train['project_resource_summary'].va
#X_cv_psr_bow = vectorizer.transform(X_cv['project_resource_summary'].values)
X_test_psr_bow = vectorizer.transform(X_test['project_resource_summary'].valu

print("After vectorizations")
print(X_train_psr_bow.shape, y_train.shape)
#print(X_cv_psr_bow.shape, y_cv.shape)
print(X_test_psr_bow.shape, y_test.shape)

feature_names.extend(vectorizer.get_feature_names())
```

```
After vectorizations
(73196, 5000) (73196,)
(36052, 5000) (36052,)
```

1.5.2 Vectorizing using TFIDF

essay

In [15]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['essay'].values)

X_train_essay_tfidf = vectorizer.transform(X_train['essay'].values)
#X_cv_essay_tfidf = vectorizer.transform(X_cv['essay'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['essay'].values)

print("After vectorizations")
print(X_train_essay_tfidf.shape, y_train.shape)
#print(X_cv_essay_tfidf.shape, y_cv.shape)
print(X_test_essay_tfidf.shape, y_test.shape)

feature_names_tfidf.extend(vectorizer.get_feature_names())
```

```
After vectorizations
(73196, 5000) (73196,)
(36052, 5000) (36052,)
```

project_title

In [16]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['project_title'].values)

X_train_titles_tfidf = vectorizer.transform(X_train['project_title'].values)
#X_cv_titles_tfidf = vectorizer.transform(X_cv['project_title'].values)
X_test_titles_tfidf = vectorizer.transform(X_test['project_title'].values)

print("After vectorizations")
print(X_train_titles_tfidf.shape, y_train.shape)
#print(X_cv_titles_tfidf.shape, y_cv.shape)
print(X_test_titles_tfidf.shape, y_test.shape)

feature_names_tfidf.extend(vectorizer.get_feature_names())
```

```
After vectorizations
(73196, 5000) (73196,)
(36052, 5000) (36052,)
```

project_resource_summary

In [17]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['project_resource_summary'].values)

X_train_prs_tfidf = vectorizer.transform(X_train['project_resource_summary'].
#X_cv_prs_tfidf = vectorizer.transform(X_cv['project_resource_summary'].value
X_test_prs_tfidf = vectorizer.transform(X_test['project_resource_summary'].va

print("After vectorizations")
print(X_train_prs_tfidf.shape, y_train.shape)
#print(X_cv_prs_tfidf.shape, y_cv.shape)
print(X_test_prs_tfidf.shape, y_test.shape)

feature_names_tfidf.extend(vectorizer.get_feature_names())
```

```
After vectorizations
(73196, 5000) (73196,)
(36052, 5000) (36052,)
```

Merging all the categorical and numerical features with variations of text features

In [18]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_train_bow_matrix = hstack((X_train_cc_ohe, X_train_csc_ohe,  X_train_state_
                             X_train_grade_ohe, X_train_price_norm.reshape(-1
                             X_train_essay_bow, X_train_titles_bow, X_train_p


X_test_bow_matrix = hstack((X_test_cc_ohe, X_test_csc_ohe, X_test_state_ohe,
                            X_test_price_norm.reshape(-1,1), X_test_ppp_norm.
                            X_test_essay_bow, X_test_titles_bow,
                            X_test_psr_bow)).tocsr()

print("Final Data matrix")
print(X_train_bow_matrix.shape, y_train.shape)
#print(X_cv_bow_matrix.shape, y_cv.shape)
print(X_test_bow_matrix.shape, y_test.shape)



print(len(feature_names))
```

```
Final Data matrix
(73196, 15101) (73196,)
(36052, 15101) (36052,)
15101
```

In [19]:

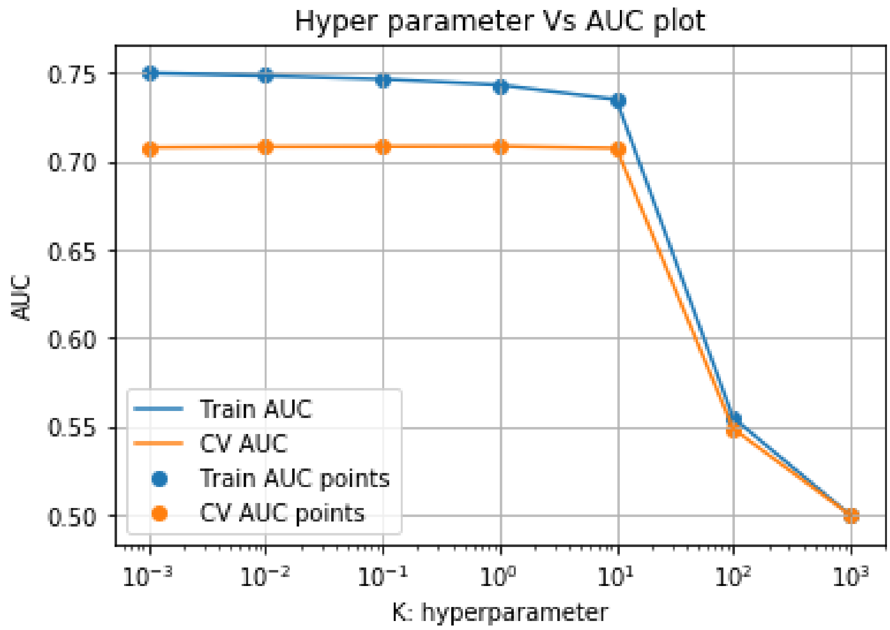
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_train_tfidf_matrix = hstack((X_train_cc_ohe, X_train_csc_ohe, X_train_state
                              X_train_price_norm.reshape(-1,1), X_train_ppp_
                             X_train_titles_tfidf, X_train_essay_tfidf, X_tra


X_test_tfidf_matrix = hstack((X_test_cc_ohe, X_test_csc_ohe, X_test_state_ohe
                            X_test_price_norm.reshape(-1,1), X_test_ppp_norm.
                              X_test_titles_tfidf, X_test_prs_tfidf)).tocsr()

print("Final Data matrix")
print(X_train_tfidf_matrix.shape, y_train.shape)
#print(X_cv_tfidf_matrix.shape, y_cv.shape)
print(X_test_tfidf_matrix.shape, y_test.shape)

print(len(feature_names_tfidf))
```

```
Final Data matrix
(73196, 15101) (73196,)
(36052, 15101) (36052,)
15101
```

Finding Best Hyper parameter using K-Fold CV on BOW representation of text features

In [20]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.G
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from scipy.stats import norm as nm
from sklearn.model_selection import RandomizedSearchCV

import matplotlib.pyplot as plt
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score

nb = MultinomialNB(class_prior = [0.5,0.5])
parameters = {'alpha':[0.001,0.01,0.1,1,10,10**2,10**3]}
clf1 = GridSearchCV(nb, parameters, cv=10, scoring='roc_auc', return_train_sc
clf1.fit(X_train_bow_matrix, y_train)

results = pd.DataFrame.from_dict(clf1.cv_results_)
results = results.sort_values(['param_alpha'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
K =  results['param_alpha']


plt.plot(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train_auc_s

plt.plot(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("K: hyperparameter")
plt.xscale('log')
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
results
```



Out[20]:

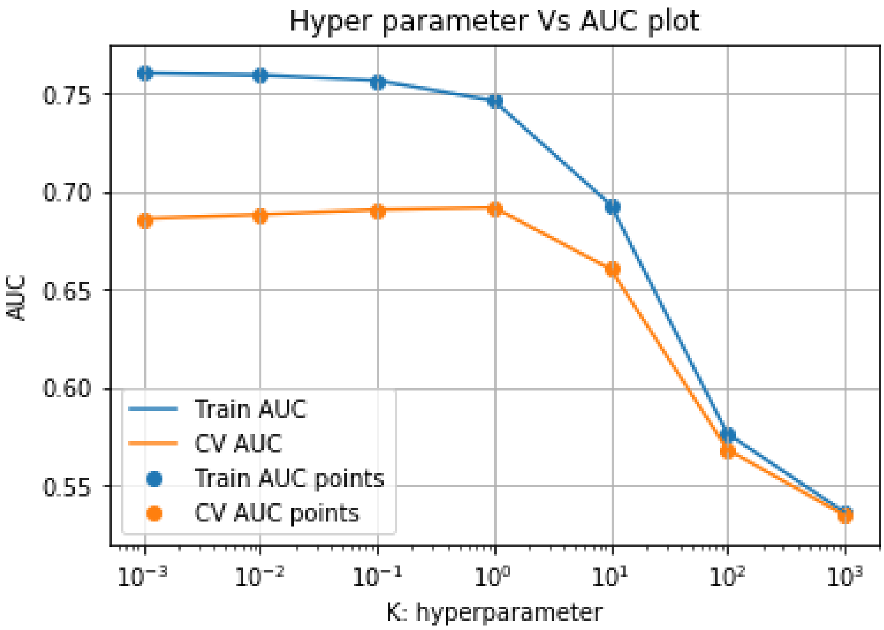
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha
0	0.229835	0.009303	0.013374	0.001419	0.001
1	0.227582	0.002220	0.013407	0.001494	0.01
2	0.229150	0.006135	0.013963	0.001355	0.1
3	0.226774	0.002843	0.013261	0.001734	1
4	0.229773	0.004796	0.013775	0.001595	10
5	0.223946	0.002794	0.013564	0.001286	100
6	0.228243	0.002483	0.012644	0.000888	1000

7 rows × 31 columns

Finding Best Hyper parameter using K-Fold CV on TFIDF representation of text features

In [21]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.G
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from scipy.stats import norm as nm
from sklearn.model_selection import RandomizedSearchCV
import matplotlib.pyplot as plt
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score

nb = MultinomialNB(class_prior = [0.5,0.5])
parameters = {'alpha':[0.001,0.01,0.1,1,10,10**2,10**3]}
clf = GridSearchCV(nb, parameters, cv=10, scoring='roc_auc', return_train_sco
clf.fit(X_train_tfidf_matrix, y_train)

results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_alpha'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
K =  results['param_alpha']


plt.plot(K, train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, train_auc - train_auc_std,train_auc + train_auc_s

plt.plot(K, cv_auc, label='CV AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
# plt.gca().fill_between(K, cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=0.2

plt.scatter(K, train_auc, label='Train AUC points')
plt.scatter(K, cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("K: hyperparameter")
plt.xscale('log')
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
results
```



Out[21]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha
0	0.232456	0.005960	0.014369	0.002146	0.001
1	0.231277	0.005977	0.013364	0.001194	0.01
2	0.229661	0.001024	0.012962	0.001412	0.1
3	0.229203	0.003328	0.013555	0.001335	1
4	0.231842	0.006450	0.013364	0.000917	10
5	0.231653	0.004571	0.013157	0.001388	100
6	0.229119	0.001947	0.013222	0.000931	1000

7 rows × 31 columns

In [22]:

```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability
    # not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your tr_loop will be 49041 - 496
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

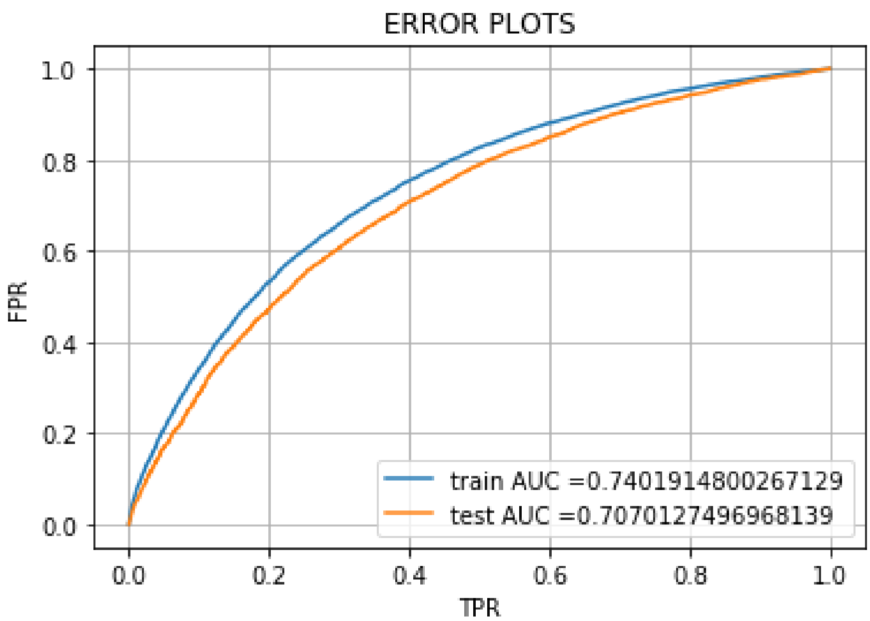
In [23]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very h
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for thresho
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

Applying NB with obtained best Alpha (Hyper parameter) on BOW

In [24]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


clf1 = MultinomialNB(alpha = 1, class_prior=[0.5,0.5])
clf1.fit(X_train_bow_matrix, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability esti
# not the predicted outputs

y_train_pred = batch_predict(clf1, X_train_bow_matrix)
y_test_pred = batch_predict(clf1, X_test_bow_matrix)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_t
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("TPR")
plt.ylabel("FPR")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

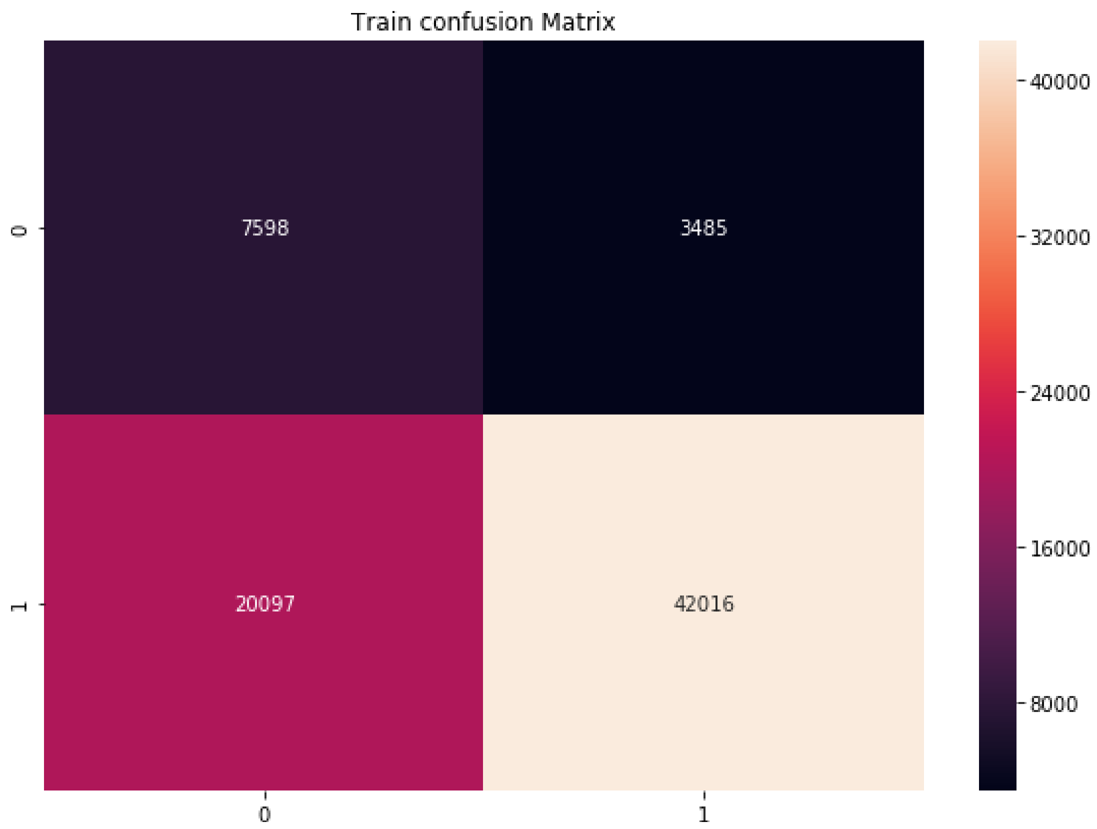


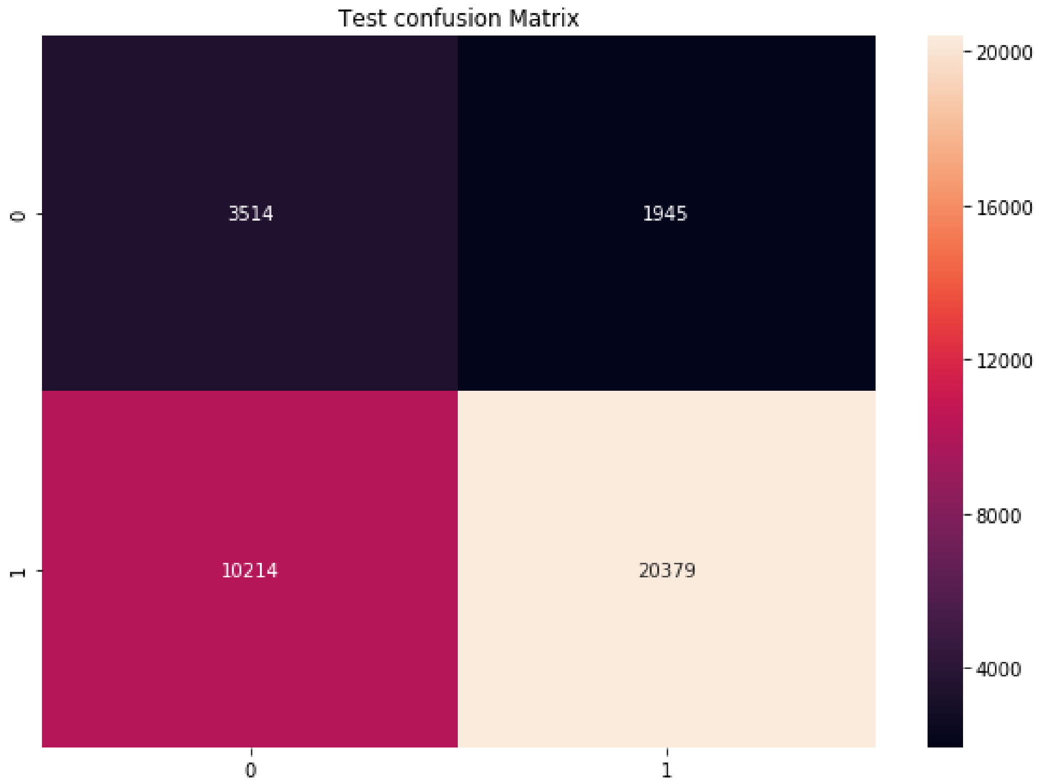
Confusion Matrix with predicted and original labels for BOW

In [25]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
train = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
test = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))

#https://stackoverflow.com/a/35572247

df_cm = pd.DataFrame(train, index = [i for i in range(2)], columns = [i for i
plt.figure(figsize = (10,7))
plt.title('Train confusion Matrix')
sns.heatmap(train, annot=True, fmt="d")
plt.show()

df_cm = pd.DataFrame(test, index = [i for i in range(2)], columns = [i for i
plt.figure(figsize = (10,7))
plt.title('Test confusion Matrix')
sns.heatmap(test, annot=True, fmt="d")
plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.46373956961336765 for thres
hold 0.448
```

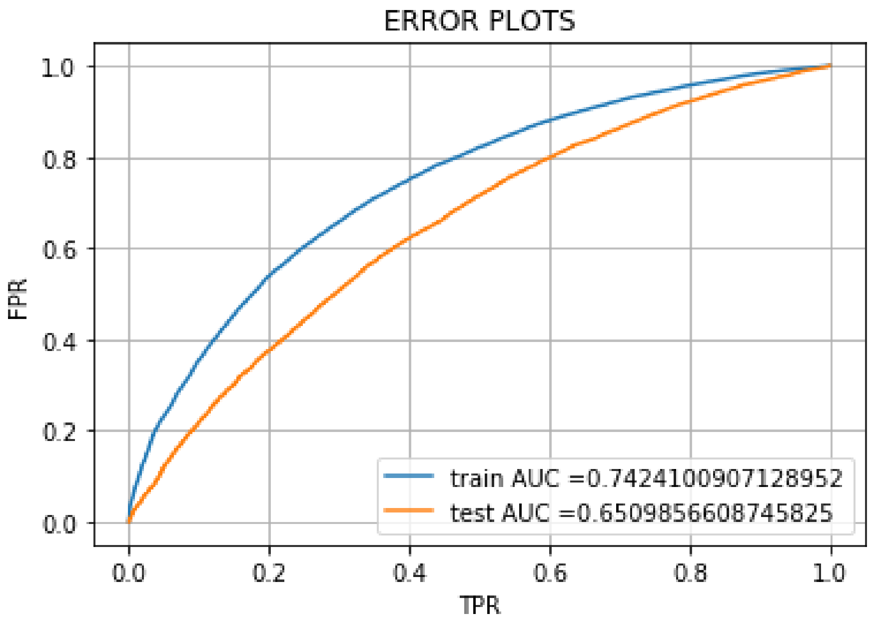




Applying NB with obtained best Alpha (Hyper parameter) on TFIDF

In [26]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


clf = MultinomialNB(alpha = 1,class_prior=[0.5,0.5])
clf.fit(X_train_tfidf_matrix, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability esti
# not the predicted outputs

y_train_pred = batch_predict(clf, X_train_tfidf_matrix)
y_test_pred = batch_predict(clf, X_test_tfidf_matrix)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_t
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr))
plt.legend()
plt.xlabel("TPR")
plt.ylabel("FPR")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

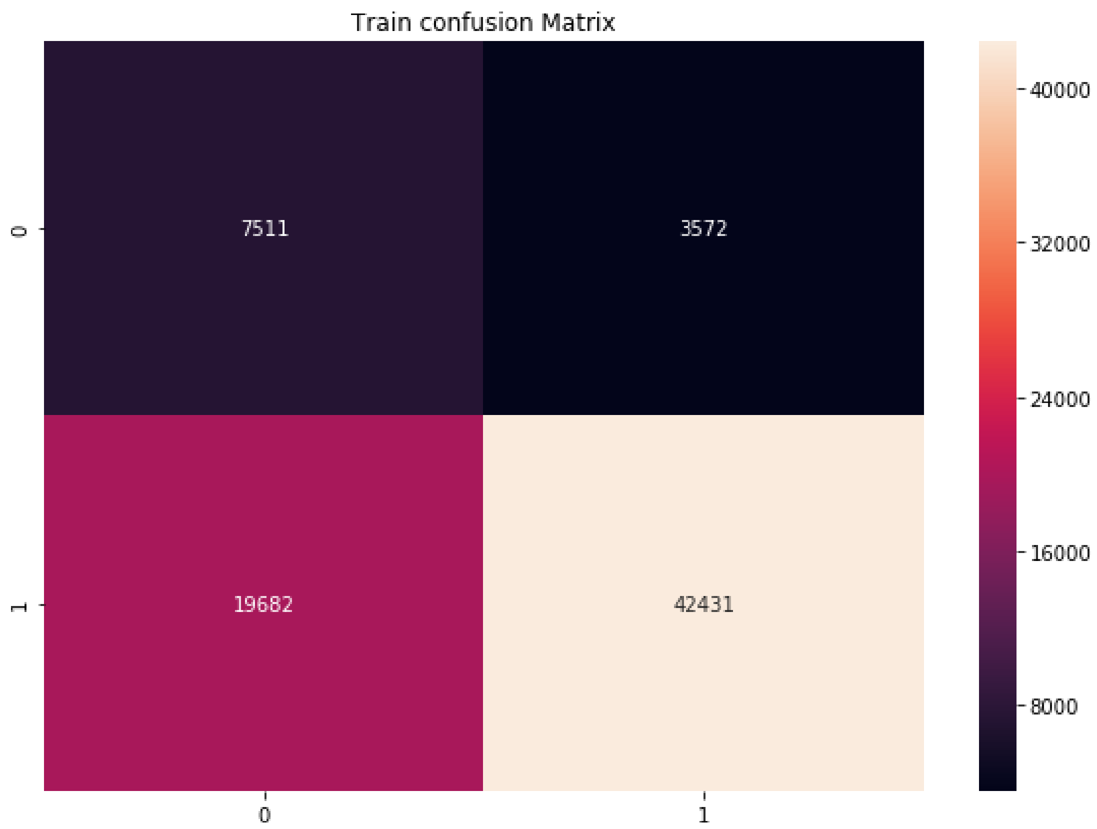


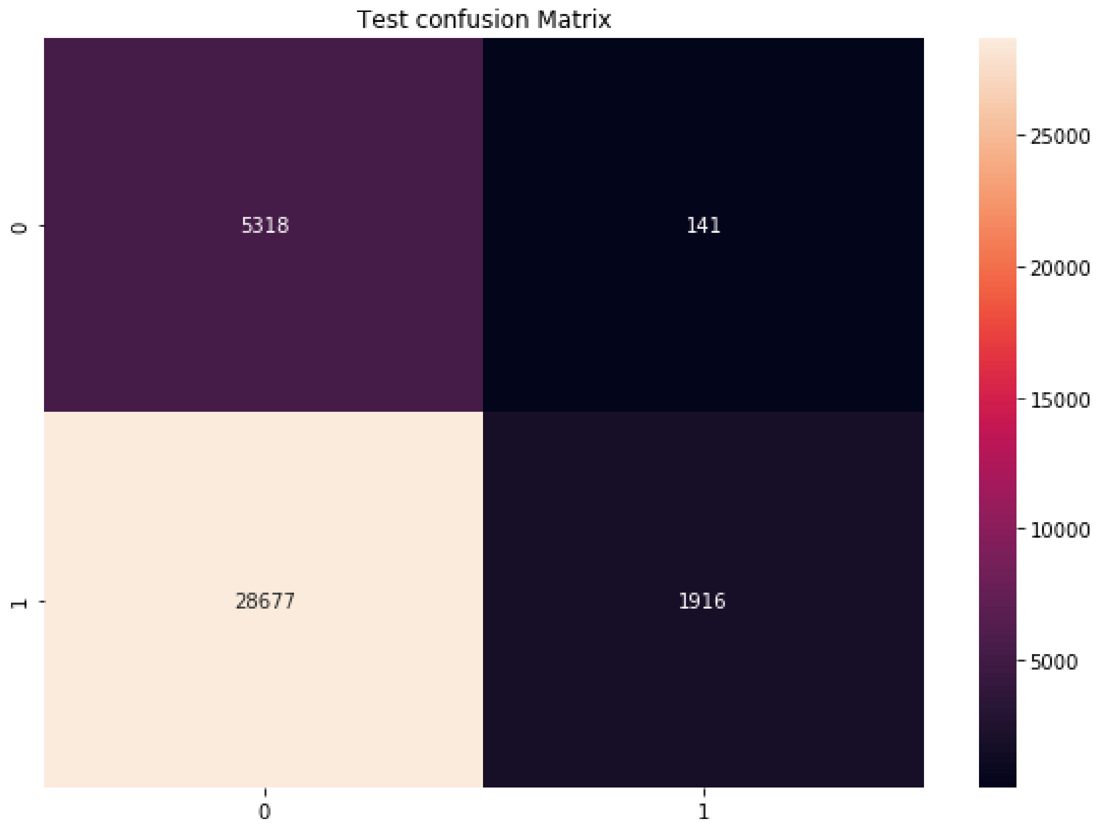
Confusion Matrix with predicted and original labels for TFIDF

In [27]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
train = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
test = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))

#https://stackoverflow.com/a/35572247

df_cm = pd.DataFrame(train, index = [i for i in range(2)], columns = [i for i
plt.figure(figsize = (10,7))
plt.title('Train confusion Matrix')
sns.heatmap(train, annot=True, fmt="d")
plt.show()

df_cm = pd.DataFrame(test, index = [i for i in range(2)], columns = [i for i
plt.figure(figsize = (10,7))
plt.title('Test confusion Matrix')
sns.heatmap(test, annot=True, fmt="d")
plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.4629575703867252 for thresh
old 0.501
```





Top 20 BOW features

In [28]:

```
max_ind_neg=np.argsort((clf1.feature_log_prob_)[0][::-1])[0:20]
top_neg=np.take(feature_names,max_ind_neg)
print('Number of features are : {}'.format(len(feature_names)))
print('\nIndices of top features to determine negetive class label are :\n{}
print('\nNames of top features to determine negetive class label are :\n{}'.f

max_ind_pos=np.argsort((clf1.feature_log_prob_)[1][::-1])[0:20]
top_pos=np.take(feature_names,max_ind_pos)
print('\nIndices of top features to determine positive class label are :\n{}
print('\nNames of top features to determine positive class label are :\n{}'.f
```

```
Number of features are : 15101

Indices of top features to determine negetive class label are
:
[8918 9726 5764 6876 7482 8232 8231 8230 8216 8212 5712 5698 5
697 5696
 5691 9376 8002 9375 9374 9368]

Names of top features to determine negetive class label are :
['space for' 'voices' 'can see' 'going green' 'language learne
rs'
 'our fingertips' 'our ears' 'our day' 'osmos' 'organizing'
 'building classroom' 'buddies' 'bronx' 'broken' 'bringing'
 'the world with' 'necessities' 'the world to' 'the world thro
ugh'
 'the win']

Indices of top features to determine positive class label are
:
[15010  7585  6364  8632  8716  9774  8856  6598  5054  8599
8280  6939
  6372  8674  6252  6889  9796  9671  6299  6282]

Names of top features to determine positive class label are :
['with hands on' 'let find' 'expanding' 'rescue' 'school suppl
ies for'
 'we like' 'signing day' 'for everything' 'would make'
 'reading writing and' 'out of' 'growing' 'exploration' 'rolli
ng'
 'electricity' 'google classroom' 'we ve' 'us be' 'engineering
in'
 'engage students']
```

Top 20 TFIDF features

In [29]:

```
max_ind_neg=np.argsort((clf.feature_log_prob_)[0][::-1])[0:20]
top_neg=np.take(feature_names_tfidf,max_ind_neg)
print('Number of features are : {}'.format(len(feature_names)))
print('\nIndices of top features to determine negetive class label are :\n{}
print('\nNames of top features to determine negetive class label are :\n{}'.f

max_ind_pos=np.argsort((clf.feature_log_prob_)[1][::-1])[0:20]
top_pos=np.take(feature_names_tfidf,max_ind_pos)
print('\nIndices of top features to determine positive class label are :\n{}
print('\nNames of top features to determine positive class label are :\n{}'.f
```

```
Number of features are : 15101

Indices of top features to determine negetive class label are
:
[10838 11641 11640 11615 11613 11611 11602 14826 11600 11593 1
1588 14831
 11517 11470 11464 11413 11412 11388 11347 11339]

Names of top features to determine negetive class label are :
['carpet' 'growing' 'grow their' 'grades' 'grade level' 'googl
e classroom'
 'glue and' 'use technology' 'gloves' 'give' 'get their wiggle
s' 'use to'
 'for reading and' 'focusing' 'focus on their' 'fill' 'file'
'extra'
 'ever' 'erasers and']

Indices of top features to determine positive class label are
:
[12585 13632 10596 10595 12182 10782 13599 11364 10299 10899 1
1359 10692
 10054 10762 14866 13141 12844 12845 13139 10597]

Names of top features to determine positive class label are :
['need binders' 'stay focused and' 'balls for' 'balls and' 'ma
cbook air'
 'build and' 'sports' 'experience in' 'and bouncy' 'charging'
'exercise'
 'bins' 'yoga for' 'bouncy bands and' 'voice' 'place' 'need va
riety of'
 'need variety of books' 'pillows' 'balls to']
```

Conclusion

In [30]:

```
# http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

table = PrettyTable()
table.field_names = ["Vectorizer", "Model", "Hyper Parameter", "AUC"]

table.add_row(['BOW', 'Naive Bayes', 1, 0.7070])
table.add_row(['TFIDF', 'Naive Bayes', 1, 0.6509])
print(table)
```

```
+------------+-------------+-----------------+--------+
| Vectorizer |    Model    | Hyper Parameter |  AUC   |
+------------+-------------+-----------------+--------+
|    BOW     | Naive Bayes |        1        | 0.707  |
|   TFIDF    | Naive Bayes |        1        | 0.6509 |
+------------+-------------+-----------------+--------+
```

Summary

- BOW vectorizer gave AUC 0.7070 with the best hyper parameter 1
- TFIDF vectorizer gave AUC 0.6509 with the best hyper parameter 1
- BOW vectorizer has better AUC compared to TFIDF vectorizer