Notebook

```
%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
from wordcloud import WordCloud, STOPWORDS
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from scipy.sparse import hstack
from sklearn import metrics
from sklearn.metrics import roc_curve, roc_auc_score, auc
from nltk.stem.porter import PorterStemmer
from sklearn.preprocessing import Normalizer
from sklearn.model_selection import GridSearchCV
from wordcloud import WordCloud

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
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

# !pip install chart_studio
from chart_studio.plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter

from sklearn.model_selection import train_test_split
```

```
from google.colab import drive
```

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

```
Mounted at /content/drive
```

```
processed=pd.read_csv('/content/drive/MyDrive/DCD_processed.csv')
processed.head()
```

	school_state	teacher_prefix	project_grade_category	teacher_number_of_prev
0	ca	mrs	grades_prek_2	
1	ut	ms	grades_3_5	
2	ca	mrs	grades_prek_2	
3	ga	mrs	grades_prek_2	
4	wa	mrs	grades_3_5	

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

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, s

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

```
(49041, 8) (36052, 8) (24155, 8)
(49041,) (36052,) (24155,)
```

Essay Encoding

```
#Finding unique words in essay to decide max features in Count vectorizer

results = set()
processed['essay'].str.lower().str.split().apply(results.update)
print(len(results))
```

```
    56381
```

```
#Transforming Essay to Bag Of Words

essay_encoded=CountVectorizer(min_df=10,ngram_range=(1,4),max_features=5000)
essay_encoded.fit(X_train['essay'].values)

X_cv_essay_bow=essay_encoded.transform(X_cv['essay'].values)
X_test_essay_bow=essay_encoded.transform(X_test['essay'].values)
X_train_essay_bow=essay_encoded.transform(X_train['essay'].values)

print(X_cv.shape,X_cv_essay_bow.shape)
print(X_test.shape,X_test_essay_bow.shape)
print(X_train.shape,X_train_essay_bow.shape)
```

```
    (24155, 8) (24155, 5000)
    (36052, 8) (36052, 5000)
    (49041, 8) (49041, 5000)
```

```
#Transforming essay to TFIDF

essay_encoded_tfidf=TfidfVectorizer(min_df=10,ngram_range=(1,4),max_features=5000)
essay_encoded_tfidf.fit(X_train['essay'].values)

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

print(X_cv.shape,X_cv_essay_tfidf.shape)
print(X_test.shape,X_test_essay_tfidf.shape)
print(X_train.shape,X_train_essay_tfidf.shape)
```

```
    (24155, 8) (24155, 5000)
    (36052, 8) (36052, 5000)
    (49041, 8) (49041, 5000)
```

```
#Transforming School state

vectorizer_state = CountVectorizer()
vectorizer_state.fit(X_train['school_state'].values)
```

```
X_cv_state=vectorizer_state.transform(X_cv['school_state'].values)
X_test_state=vectorizer_state.transform(X_test['school_state'].values)
X_train_state=vectorizer_state.transform(X_train['school_state'].values)

print(X_cv['school_state'].shape,X_cv_state.shape)
print(X_test['school_state'].shape,X_test_state.shape)
print(X_train['school_state'].shape,X_train_state.shape)
```

```
(24155,) (24155, 51)
(36052,) (36052, 51)
(49041,) (49041, 51)
```

```
#Transforming Grade

vectorizer_grade = CountVectorizer()
vectorizer_grade.fit(X_train['project_grade_category'].values)

X_cv_grade=vectorizer_grade.transform(X_cv['project_grade_category'].values)
X_test_grade=vectorizer_grade.transform(X_test['project_grade_category'].values)
X_train_grade=vectorizer_grade.transform(X_train['project_grade_category'].values)

print(X_cv['project_grade_category'].shape,X_cv_grade.shape)
print(X_test['project_grade_category'].shape,X_test_grade.shape)
print(X_train['project_grade_category'].shape,X_train_grade.shape)
```

```
(24155,) (24155, 4)
(36052,) (36052, 4)
(49041,) (49041, 4)
```

```
#Transforming Teacher Prefix

vectorizer_prefix = CountVectorizer()
vectorizer_prefix.fit(X_train['teacher_prefix'].values)

X_cv_teacherPrefix=vectorizer_prefix.transform(X_cv['teacher_prefix'].values)
X_test_teacherPrefix=vectorizer_prefix.transform(X_test['teacher_prefix'].values)
X_train_teacherPrefix=vectorizer_prefix.transform(X_train['teacher_prefix'].values)

print(X_cv['teacher_prefix'].shape,X_cv_teacherPrefix.shape)
print(X_test['teacher_prefix'].shape,X_test_teacherPrefix.shape)
print(X_train['teacher_prefix'].shape,X_train_teacherPrefix.shape)
```

```
(24155,) (24155, 5)
(36052,) (36052, 5)
(49041,) (49041, 5)
```

```
#Transforming Project Categories

vectorizer_cat = CountVectorizer()
vectorizer_cat.fit(X_train['clean_categories'].values)

X_cv_category=vectorizer_cat.transform(X_cv['clean_categories'].values)
X_test_category=vectorizer_cat.transform(X_test['clean_categories'].values)
X_train_category=vectorizer_cat.transform(X_train['clean_categories'].values)
```

```
print(X_cv['clean_categories'].shape,X_cv_category.shape)
print(X_test['clean_categories'].shape,X_test_category.shape)
print(X_train['clean_categories'].shape,X_train_category.shape)
```

```
(24155,) (24155, 9)
(36052,) (36052, 9)
(49041,) (49041, 9)
```

```
#Transforming Project Subcategories

vectorizer_subcat = CountVectorizer()
vectorizer_subcat.fit(X_train['clean_subcategories'].values)

X_cv_subcategory=vectorizer_subcat.transform(X_cv['clean_subcategories'].values)
X_test_subcategory=vectorizer_subcat.transform(X_test['clean_subcategories'].values
X_train_subcategory=vectorizer_subcat.transform(X_train['clean_subcategories'].valu

print(X_cv['clean_subcategories'].shape,X_cv_subcategory.shape)
print(X_test['clean_subcategories'].shape,X_test_subcategory.shape)
print(X_train['clean_subcategories'].shape,X_train_subcategory.shape)
```

```
(24155,) (24155, 30)
(36052,) (36052, 30)
(49041,) (49041, 30)
```

```
#Normalizing price

norm = Normalizer()
norm.fit(X_train['price'].values.reshape(1,-1))

X_cv_price=norm.transform(X_cv['price'].values.reshape(-1,1))
X_test_price=norm.transform(X_test['price'].values.reshape(-1,1))
X_train_price=norm.transform(X_train['price'].values.reshape(-1,1))

print(X_cv['price'].shape,X_cv_price.shape)
print(X_test['price'].shape,X_test_price.shape)
print(X_train['price'].shape,X_train_price.shape)
```

```
(24155,) (24155, 1)
(36052,) (36052, 1)
(49041,) (49041, 1)
```

```
#Normalizing teacher's previous projects

previous_norm = Normalizer()
previous_norm.fit(X_train['price'].values.reshape(1,-1))

X_cv_previousProjects=previous_norm.transform(X_cv['teacher_number_of_previously_po
X_test_previousProjects=previous_norm.transform(X_test['teacher_number_of_previousl
X_train_previousProjects=previous_norm.transform(X_train['teacher_number_of_previou

print(X_cv['teacher_number_of_previously_posted_projects'].shape,X_cv_previousProje
print(X_test['teacher_number_of_previously_posted_projects'].shape,X_test_previousP
print(X_train['teacher_number_of_previously_posted_projects'].shape,X_train_previou
```

```
print(X_train['teacher_number_of_previously_posted_projects'].shape,X_train_previou
```

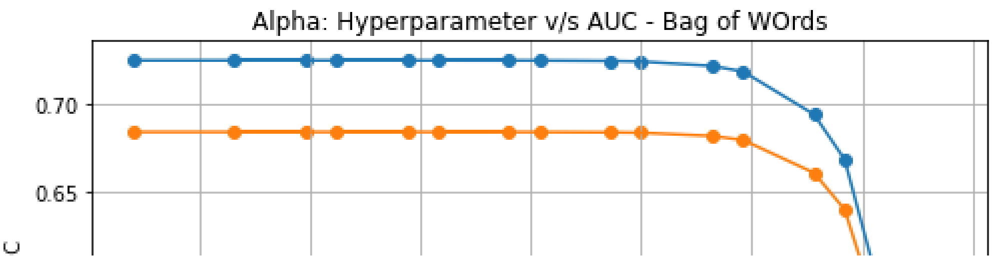
```
(24155,) (24155, 1)
(36052,) (36052, 1)
(49041,) (49041, 1)
```

```
#Stacking data for BOW

X_train_BOW=hstack((X_train_previousProjects,X_train_price,X_train_subcategory,X_tr
                    X_train_grade,X_train_state,X_train_essay_bow)).tocsr()
X_test_BOW=hstack((X_test_previousProjects,X_test_price,X_test_subcategory,X_test_c
                    X_test_grade,X_test_state,X_test_essay_bow)).tocsr()
X_cv_BOW=hstack((X_cv_previousProjects,X_cv_price,X_cv_subcategory,X_cv_category,X_
                    X_cv_grade,X_cv_state,X_cv_essay_bow)).tocsr()

print("BOW data : ")
print(X_train_BOW.shape)
print(X_test_BOW.shape)
print(X_cv_BOW.shape)

#Stacking data for tfidf

X_train_tfidf=hstack((X_train_previousProjects,X_train_price,X_train_subcategory,X_
                    X_train_grade,X_train_state,X_train_essay_tfidf)).tocsr()
X_test_tfidf=hstack((X_test_previousProjects,X_test_price,X_test_subcategory,X_test
                    X_test_grade,X_test_state,X_test_essay_tfidf)).tocsr()
X_cv_tfidf=hstack((X_cv_previousProjects,X_cv_price,X_cv_subcategory,X_cv_category,
                    X_cv_grade,X_cv_state,X_cv_essay_tfidf)).tocsr()

print("\ntfidf data : ")
print(X_train_tfidf.shape)
print(X_test_tfidf.shape)
print(X_cv_tfidf.shape)
```

```
BOW data :
(49041, 5101)
(36052, 5101)
(24155, 5101)

tfidf data :
(49041, 5101)
(36052, 5101)
(24155, 5101)
```

```
def k_fold_predict(clf, data,k):

    y_data_pred = []
    length=data.shape[0]
    fold=length//k

    tr_loop = data.shape[0] - data.shape[0]%fold
    for i in range(0, tr_loop, fold):
        y_data_pred.extend(clf.predict_proba(data[i:i+fold])[:,1])
    if data.shape[0]%fold !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])
```

```
    return y_data_pred


alphas = [0.00001,0.0005, 0.0001,0.005,0.001,0.05,0.01,0.1,0.5,1,5,10,50,100,500,10

# For BOW data
T_auc=[]
CV_auc=[]

for alfa in alphas:
   model=MultinomialNB(alpha=alfa,class_prior=[0.5,0.5])
   model.fit(X_train_BOW,y_train)

   y_train_pred_BOW=k_fold_predict(model,X_train_BOW,10)
   y_cv_pred_BOW=k_fold_predict(model,X_cv_BOW,10)

   T_auc.append(roc_auc_score(y_train,y_train_pred_BOW))
   CV_auc.append(roc_auc_score(y_cv,y_cv_pred_BOW))

print(T_auc)
print(CV_auc)

    [0.7244966322203983, 0.7244963037764997, 0.7244965772100901, 0.724493727999719
    [0.6841151365988261, 0.68411497651598, 0.6841150832378773, 0.6841131489034872,


#Taking log of alphas

log_alphas=np.log(alphas)


# Plotting ROC Curve

plt.figure(figsize=(8,4))
plt.plot(log_alphas, T_auc, label='Train AUC')
plt.plot(log_alphas, CV_auc, label='CV AUC')

plt.scatter(log_alphas, T_auc, label='Train AUC points')
plt.scatter(log_alphas, CV_auc, label='CV AUC points')

plt.legend()
plt.xlabel("Log Alpha: Hyperparameter")
plt.ylabel("AUC")
plt.title("Alpha: Hyperparameter v/s AUC - Bag of WOrds")
plt.grid()
plt.show()
```




```
MNB_BOW = MultinomialNB(class_prior=[0.5,0.5])

alphas = {'alpha':[0.00001,0.0005, 0.0001,0.005,0.001,0.05,0.01,0.1,0.5,1,5,10,50,1

GridClf = GridSearchCV(MNB_BOW, alphas, cv= 10, scoring='roc_auc',return_train_scor

GridClf.fit(X_train_BOW, y_train)

train_auc_bow = GridClf.cv_results_['mean_train_score']
train_auc_std_bow = GridClf.cv_results_['std_train_score']
cv_auc_bow = GridClf.cv_results_['mean_test_score']
cv_auc_std_bow = GridClf.cv_results_['std_test_score']
```

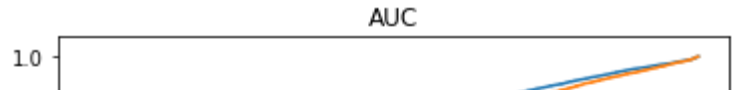
```
[CV] alpha=1e-05 ...................................................
[CV] .................................... alpha=1e-05, total=   0.1s
[CV] alpha=0.0005 ..................................................
[CV] ................................... alpha=0.0005, total=   0.1s
[CV] alpha=0.0005 ..................................................
[CV] ................................... alpha=0.0005, total=   0.1s
[CV] alpha=0.0005 ..................................................
[CV] ................................... alpha=0.0005, total=   0.1s
[CV] alpha=0.0005 ..................................................
[CV] ................................... alpha=0.0005, total=   0.1s
[CV] alpha=0.0005 ..................................................
[CV] ................................... alpha=0.0005, total=   0.1s
[CV] alpha=0.0005 ..................................................
[CV] ................................... alpha=0.0005, total=   0.1s
[CV] alpha=0.0005 ..................................................
[CV] ................................... alpha=0.0005, total=   0.1s
[CV] alpha=0.0005 ..................................................

[CV] ................................... alpha=0.0005, total=   0.1s
[CV] alpha=0.0005 ..................................................
[CV] ................................... alpha=0.0005, total=   0.1s
[CV] alpha=0.0005 ..................................................
[CV] ................................... alpha=0.0005, total=   0.1s
[CV] alpha=0.0001 ..................................................
[CV] ................................... alpha=0.0001, total=   0.1s
[CV] alpha=0.0001 ..................................................
[CV] ................................... alpha=0.0001, total=   0.1s
[CV] alpha=0.0001 ..................................................
[CV] ................................... alpha=0.0001, total=   0.1s
[CV] alpha=0.0001 ..................................................
[CV] ................................... alpha=0.0001, total=   0.1s
[CV] alpha=0.0001 ..................................................
[CV] ................................... alpha=0.0001, total=   0.1s
[CV] alpha=0.0001 ..................................................
[CV] ................................... alpha=0.0001, total=   0.1s
[CV] alpha=0.0001 ..................................................
[CV] ................................... alpha=0.0001, total=   0.1s
[CV] alpha=0.0001 ..................................................
[CV] ................................... alpha=0.0001, total=   0.1s
```

```
[CV] alpha=0.0001 ....................................................
[CV] ..................................... alpha=0.0001, total=   0.1s
[CV] alpha=0.0001 ....................................................
[CV] ..................................... alpha=0.0001, total=   0.1s
[CV] alpha=0.005 .....................................................
[CV] ...................................... alpha=0.005, total=   0.1s
[CV] alpha=0.005 .....................................................
[CV] ...................................... alpha=0.005, total=   0.1s
[CV] alpha=0.005 .....................................................
[CV] ...................................... alpha=0.005, total=   0.1s
[CV] alpha=0.005 .....................................................
[CV] ...................................... alpha=0.005, total=   0.1s
[CV] alpha=0.005 .....................................................
[CV] ...................................... alpha=0.005, total=   0.1s
[CV] alpha=0.005 .....................................................
[CV] ...................................... alpha=0.005, total=   0.1s
[CV] alpha=0.005 .....................................................
[CV] ...................................... alpha=0.005, total=   0.1s
[CV] alpha=0.005 .....................................................
[CV] ...................................... alpha=0.005, total=   0.1s
[CV] alpha=0.005 .....................................................
```

```
GridClf.best_params_
```

```
{'alpha': 1e-05}
```

```
MNB_BOW_Final = MultinomialNB(alpha = 0.00001,class_prior=[0.5,0.5])

MNB_BOW_Final.fit(X_train_BOW, y_train)

y_train_pred_bow =k_fold_predict(MNB_BOW_Final,X_train_BOW,10)
y_test_pred_bow =k_fold_predict(MNB_BOW_Final,X_test_BOW,10)

train_fpr_bow, train_tpr_bow, tr_thresholds_bow = roc_curve(y_train, y_train_pred_b
test_fpr_bow, test_tpr_bow, te_thresholds_bow = roc_curve(y_test, y_test_pred_bow)

plt.plot(train_fpr_bow, train_tpr_bow, label="Train AUC ="+str(auc(train_fpr_bow, t
plt.plot(test_fpr_bow, test_tpr_bow, label="Test AUC ="+str(auc(test_fpr_bow, test_
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.show()
```

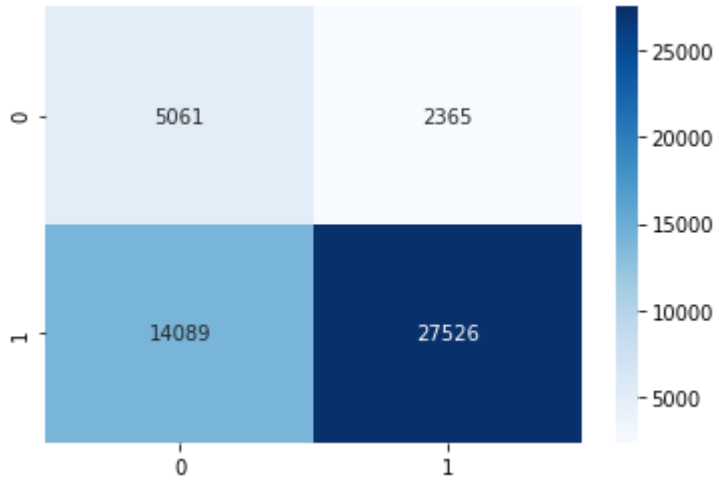



```
def best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    print("Best Threshold :", np.round(t,3))
    return t
```


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

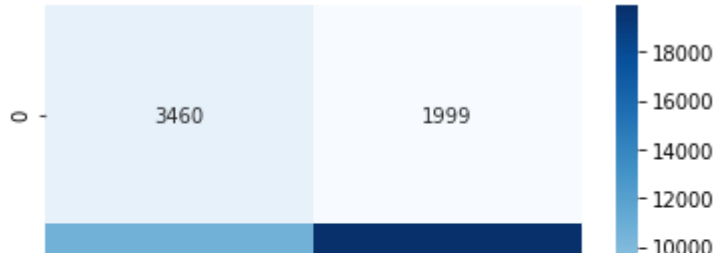
```
threshold = best_threshold(tr_thresholds_bow, train_fpr_bow, train_tpr_bow)

conf_mat_train_bow = confusion_matrix(y_train, new_prediction(y_train_pred_bow, thr
print("Confusion matrix for Train Data : ")
sns.heatmap(conf_mat_train_bow, annot=True,fmt="d",cmap='Blues')
```

```
Best Threshold : 0.525
Confusion matrix for Train Data :
<matplotlib.axes._subplots.AxesSubplot at 0x7f64b3f54090>
```




```
conf_mat_train_bow = confusion_matrix(y_test, new_prediction(y_test_pred_bow, thres
print("Confusion matrix for Train Data : ")
sns.heatmap(conf_mat_train_bow, annot=True,fmt="d",cmap='Blues')
```

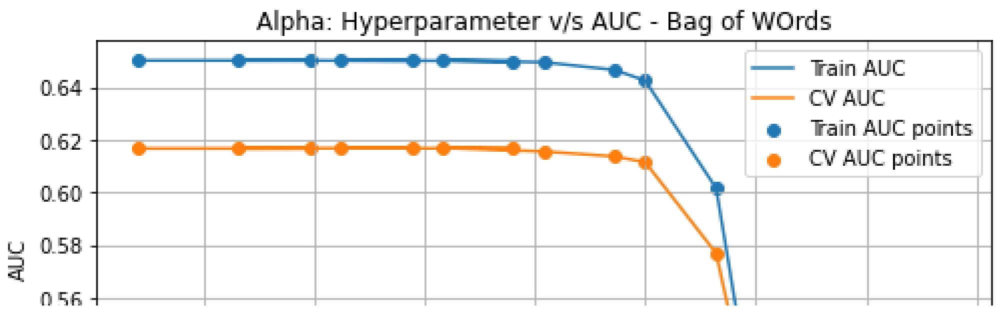
```
Confusion matrix for Train Data :
<matplotlib.axes._subplots.AxesSubplot at 0x7f64ba72f3d0>
```




```
alphas = [0.00001,0.0005, 0.0001,0.005,0.001,0.05,0.01,0.1,0.5,1,5,10,50,100,500,10

# For TFIDF data
T_auc_tfidf=[]
CV_auc_tfidf=[]

for alfa in alphas:
   model=MultinomialNB(alpha=alfa,class_prior=[0.5,0.5])
   model.fit(X_train_tfidf,y_train)

   y_train_pred_tfidf=model.predict(X_train_tfidf)
   y_cv_pred_tfidf=model.predict(X_cv_tfidf)

   T_auc_tfidf.append(roc_auc_score(y_train,y_train_pred_tfidf))
   CV_auc_tfidf.append(roc_auc_score(y_cv,y_cv_pred_tfidf))

print(T_auc_tfidf)
print(CV_auc_tfidf)
```

```
[0.6501070452057564, 0.6501070452057564, 0.6501070452057564, 0.650058985611859
[0.6166448193602486, 0.6166448193602486, 0.6166448193602486, 0.616830328698374
```

```
# Plotting ROC Curve

plt.figure(figsize=(8,4))
plt.plot(log_alphas, T_auc_tfidf, label='Train AUC')
plt.plot(log_alphas, CV_auc_tfidf, label='CV AUC')

plt.scatter(log_alphas, T_auc_tfidf, label='Train AUC points')
plt.scatter(log_alphas, CV_auc_tfidf, label='CV AUC points')

plt.legend()
plt.xlabel("Log Alpha: Hyperparameter")
plt.ylabel("AUC")
plt.title("Alpha: Hyperparameter v/s AUC - Bag of WOrds")
plt.grid()
plt.show()
```




```
MNB_tfidf = MultinomialNB(class_prior=[0.5,0.5])

alphas = {'alpha':[0.00001,0.0005, 0.0001,0.005,0.001,0.05,0.01,0.1,0.5,1,5,10,50,1

GridClf = GridSearchCV(MNB_tfidf, alphas, cv= 10, scoring='roc_auc',return_train_sc

GridClf.fit(X_train_tfidf, y_train)

train_auc_bow = GridClf.cv_results_['mean_train_score']
train_auc_std_bow = GridClf.cv_results_['std_train_score']
cv_auc_bow = GridClf.cv_results_['mean_test_score']
cv_auc_std_bow = GridClf.cv_results_['std_test_score']
```

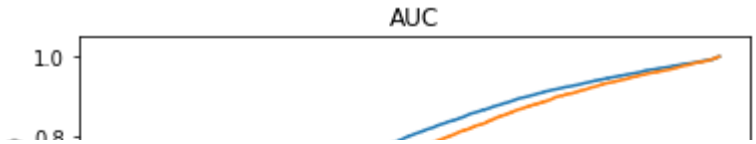
```
[CV] .................................... alpha=0.0001, total=   0.1s
[CV] alpha=0.0001 ....................................................
[CV] .................................... alpha=0.0001, total=   0.1s
[CV] alpha=0.0001 ....................................................
[CV] .................................... alpha=0.0001, total=   0.1s
[CV] alpha=0.0001 ....................................................
[CV] .................................... alpha=0.0001, total=   0.1s
[CV] alpha=0.0001 ....................................................
[CV] .................................... alpha=0.0001, total=   0.1s
[CV] alpha=0.005 .....................................................
[CV] ..................................... alpha=0.005, total=   0.1s
[CV] alpha=0.005 .....................................................
[CV] ..................................... alpha=0.005, total=   0.1s
[CV] alpha=0.005 .....................................................
[CV] ..................................... alpha=0.005, total=   0.1s
[CV] alpha=0.005 .....................................................
[CV] ..................................... alpha=0.005, total=   0.1s
[CV] alpha=0.005 .....................................................
[CV] ..................................... alpha=0.005, total=   0.1s
[CV] alpha=0.005 .....................................................
[CV] ..................................... alpha=0.005, total=   0.1s
[CV] alpha=0.005 .....................................................
[CV] ..................................... alpha=0.005, total=   0.1s
[CV] alpha=0.005 .....................................................
[CV] ..................................... alpha=0.005, total=   0.1s
[CV] alpha=0.005 .....................................................
[CV] ..................................... alpha=0.005, total=   0.1s
[CV] alpha=0.005 .....................................................
[CV] ..................................... alpha=0.005, total=   0.1s
[CV] alpha=0.001 .....................................................
[CV] ..................................... alpha=0.001, total=   0.1s
[CV] alpha=0.001 .....................................................
[CV] ..................................... alpha=0.001, total=   0.1s
[CV] alpha=0.001 .....................................................
[CV] ..................................... alpha=0.001, total=   0.1s
[CV] alpha=0.001 .....................................................
[CV] ..................................... alpha=0.001, total=   0.1s
[CV] alpha=0.001 .....................................................
```

```
[CV] ..................................... alpha=0.001, total=   0.1s
[CV] alpha=0.001 ......................................................
[CV] ..................................... alpha=0.001, total=   0.1s
[CV] alpha=0.001 ......................................................
[CV] ..................................... alpha=0.001, total=   0.1s
[CV] alpha=0.001 ......................................................
[CV] ..................................... alpha=0.001, total=   0.1s
[CV] alpha=0.001 ......................................................
[CV] ..................................... alpha=0.001, total=   0.1s
[CV] alpha=0.001 ......................................................
[CV] ..................................... alpha=0.001, total=   0.1s
[CV] alpha=0.05 .......................................................
[CV] ..................................... alpha=0.05, total=   0.1s
[CV] alpha=0.05 .......................................................
[CV] ..................................... alpha=0.05, total=   0.1s
[CV] alpha=0.05 .......................................................
[CV] ..................................... alpha=0.05, total=   0.1s
[CV] alpha=0.05 .......................................................
[CV] ..................................... alpha=0.05, total=   0.1s
[CV] alpha=0.05 .......................................................
[CV] ..................................... alpha=0.05, total=   0.1s
[CV] alpha=0.05 .......................................................
[CV] ..................................... alpha=0.05, total=   0.1s
[CV] alpha=0.05
```

```
GridClf.best_params_
```

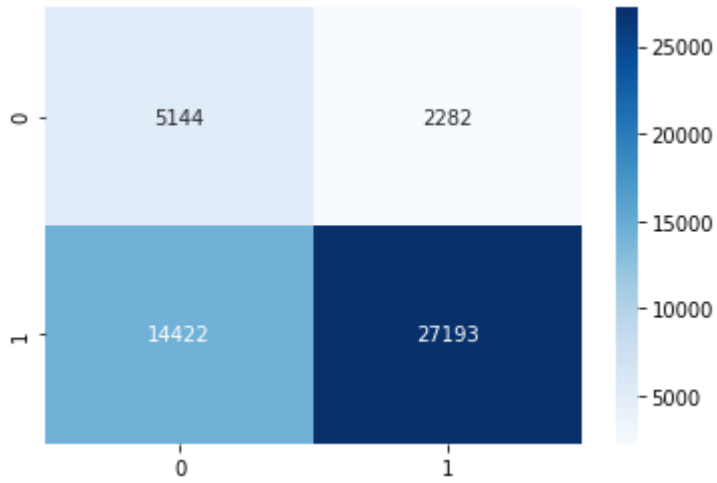
```
{'alpha': 1e-05}
```

```
MNB_tfidf_Final = MultinomialNB(alpha = 0.00001,class_prior=[0.5,0.5])

MNB_tfidf_Final.fit(X_train_tfidf, y_train)

y_train_pred_tfidf =k_fold_predict(MNB_tfidf_Final,X_train_BOW,10)
y_test_pred_tfidf =k_fold_predict(MNB_tfidf_Final,X_test_BOW,10)

train_fpr_tfidf, train_tpr_tfidf, tr_thresholds_tfidf = roc_curve(y_train, y_train_
test_fpr_tfidf, test_tpr_tfidf, te_thresholds_tfidf = roc_curve(y_test, y_test_pred

plt.plot(train_fpr_tfidf, train_tpr_tfidf, label="Train AUC ="+str(auc(train_fpr_tf
plt.plot(test_fpr_tfidf, test_tpr_tfidf, label="Test AUC ="+str(auc(test_fpr_tfidf,
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.show()
```



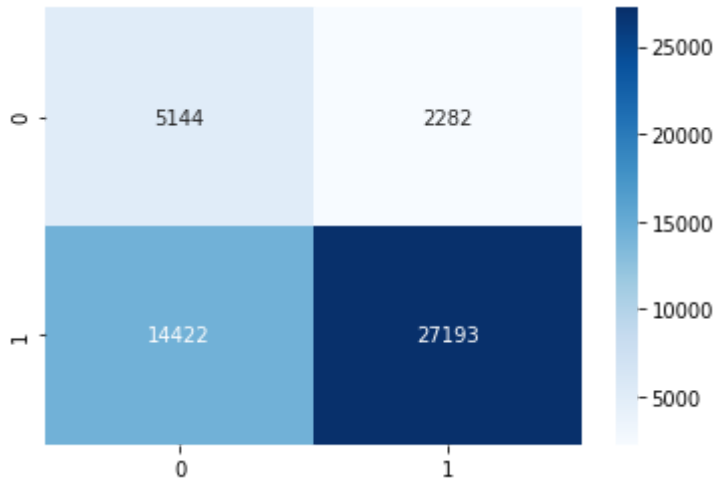

```
threshold = best_threshold(tr_thresholds_tfidf, train_fpr_tfidf, train_tpr_tfidf)

conf_mat_train_tfidf = confusion_matrix(y_train, new_prediction(y_train_pred_tfidf,
print("Confusion matrix for Train Data : ")
print(conf_mat_train_tfidf)
sns.heatmap(conf_mat_train_tfidf, annot=True,fmt="d",cmap='Blues')
```

```
Best Threshold : 0.538
Confusion matrix for Train Data :
[[ 5144  2282]
 [14422 27193]]
<matplotlib.axes._subplots.AxesSubplot at 0x7f64a4782310>
```




```
conf_mat_test_tfidf = confusion_matrix(y_train, new_prediction(y_train_pred_tfidf,
print("Confusion matrix for Train Data : ")
sns.heatmap(conf_mat_test_tfidf, annot=True,fmt="d",cmap='Blues')
```

```
Confusion matrix for Train Data :
<matplotlib.axes._subplots.AxesSubplot at 0x7f649eb76110>
```




```
f_bow=[];words=''

for f in vectorizer_subcat.get_feature_names():
  f_bow.append(f)
```

```
  f_bow.append(f)
for f in vectorizer_cat.get_feature_names():
  f_bow.append(f)
for f in vectorizer_prefix.get_feature_names():
  f_bow.append(f)
for f in vectorizer_grade.get_feature_names():
  f_bow.append(f)
for f in vectorizer_state.get_feature_names():
  f_bow.append(f)
for f in essay_encoded.get_feature_names():
  f_bow.append(f)

bow_imp = MNB_BOW_Final.feature_log_prob_[1, :].argsort()[::-1]
print(bow_imp)
for j,i in enumerate(bow_imp[:30]):
    print(f_bow[i])
    words=words+(f_bow[i].replace(" ", "_")+" ")*(30-j)
print(words)
print("\n\nPositive Probability Words BOW\n")
wordcloud = WordCloud(stopwords = STOPWORDS,
                      collocations=True).generate(words)

#plot the wordcloud object
fig = plt.figure(1, figsize=(12, 12))
plt.imshow(wordcloud, interpolation='bilInear')
plt.axis('off')
plt.show()
```

```
[4094 3679 2866 ...  100   96   41]
students able
school 100 students
my classroom
learning centers
classroom allow students
the best
they also
not able control
my students active
learn better
help become
care_hunger
many challenges
national
we also
need access
reading books
work collaboratively
use chromebooks
appliedsciences
love books
day class
able choose
come class
class full
would allow
```

```
words=''
bow_not_imp = MNB_BOW_Final.feature_log_prob_[0, :].argsort()[::-1]
print(bow_not_imp)
for i in bow_not_imp[:30]:
    print(f_bow[i])
    words=words+(f_bow[i].replace(" ", "_")+" ")*(30-j)
print(words)
print("\n\nNegative Probability Words BOW\n")
wordcloud = WordCloud(stopwords = STOPWORDS,
                      collocations=True).generate(words)

#plot the wordcloud object
fig = plt.figure(1, figsize=(12, 12))
plt.imshow(wordcloud, interpolation='bilInear')
plt.axis('off')
plt.show()
```

```
[4094 3679 2398 ...   78   96   41]
students able
school 100 students
learning centers
my classroom
classroom allow students
not able control
learn better
they also
help become
the best
my students active
care_hunger
national
many challenges
we also
need access
work collaboratively
appliedsciences
come class
love books
reading books
materials allow students
day class
able choose
skills help
use chromebooks
our classroom
class full
year my
want best
students_able school_100_students learning_centers my_classroom classroom_allo
```

Negative Probability Words BOW


```
f_tfidf=[];words=''

for f in vectorizer_subcat.get_feature_names():
  f_tfidf.append(f)
for f in vectorizer_cat.get_feature_names():
  f_tfidf.append(f)
for f in vectorizer_prefix.get_feature_names():
  f_tfidf.append(f)
for f in vectorizer_grade.get_feature_names():
  f_tfidf.append(f)
for f in vectorizer_state.get_feature_names():
  f_tfidf.append(f)
for f in essay_encoded.get_feature_names():
  f_tfidf.append(f)
```

```
tfidf_imp = MNB_tfidf_Final.feature_log_prob_[1, :].argsort()[::-1]
print(tfidf_imp)
for i in tfidf_imp[:30]:
    print(f_tfidf[i])
    words=words+(f_bow[i].replace(" ", "_")+" ")*(30-j)
print(words)
print("\n\nPositive Probability Words TFIDF\n")
wordcloud = WordCloud(stopwords = STOPWORDS,
                      collocations=True).generate(words)

#plot the wordcloud object
fig = plt.figure(1, figsize=(12, 12))
plt.imshow(wordcloud, interpolation='bilInear')
plt.axis('off')
plt.show()
```

```
[   1    0   43 ... 1906 3316   41]
care_hunger
appliedsciences
teacher
music_arts
al
specialneeds
grades_3_5
grades_9_12
mathematics
nutritioneducation
music
grades_prek_2
ct
literacy_language
```

```
words=''
tfidf_not_imp = MNB_tfidf_Final.feature_log_prob_[0, :].argsort()[::-1]
print(tfidf_not_imp)
for i in tfidf_not_imp[:30]:
    print(f_tfidf[i])
    words=words+(f_bow[i].replace(" ", "_")+" ")*(30-j)
print(words)
print("\n\nNegative Probability Words TFIDF\n")
wordcloud = WordCloud(stopwords = STOPWORDS,
                      collocations=True).generate(words)

#plot the wordcloud object
fig = plt.figure(1, figsize=(12, 12))
plt.imshow(wordcloud, interpolation='bilInear')
plt.axis('off')
plt.show()
```

```
[   1    0   43 ... 4366  692 4537]
care_hunger
appliedsciences
teacher
music_arts
al
specialneeds
grades_3_5
grades_9_12
mathematics
nutritioneducation
music
grades_prek_2
visualarts
dr
ct
literacy_language
health_sports
students able
charactereducation
ak
ms
warmth
literacy
va
appliedlearning
hi
extracurricular
ok
math_science
environmentalscience
care_hunger appliedsciences teacher music_arts al specialneeds grades_3_5 grad
```

AUC Scores

1. BOW-Vectorizer with MultinomialNB :

- Train:0.724
- Test:0.687

2. TFIDF-Vectorizer with MultinomialNB :

- Train : 0.725
- Test : 0.688

Observation

1. Vector transformation was bit lengthy procedure
2. AUC score for TFIDF was slightly more
3. More weighted words was mostly different in BOW and TFIDF

✓ 0s completed at 5:28 PM