Assignment 6: Apply NB

In [82]:

```
%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 chart_studio.plotly
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
# 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

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

Importing Data

In [83]:

```
project_data = pd.read_csv('train_data.csv')
resource_data = pd.read_csv('resources.csv')
```

In [84]:

```
print("Number of data points in train data", project_data.shape)
print('-'*50)
print("The attributes of data :", project_data.columns.values)
project_data.project_is_approved.value_counts()
```

```
Number of data points in train data (109248, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'scho
ol_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

Out[84]:

```
1    92706
0    16542
Name: project_is_approved, dtype: int64
```

In [85]:

```

print("Number of data points in train data", resource_data.shape)
print(resource_data.columns.values)
resource_data.head(2)
```

```
Number of data points in train data (1541272, 4)
['id' 'description' 'quantity' 'price']
```

Out[85]:

	id	description	quantity	price
0	p233245	LC652 - Lakeshore Double-Space Mobile Drying Rack	1	149.00
1	p069063	Bouncy Bands for Desks (Blue support pipes)	3	14.95

In [86]:

```
project_data.columns
```

Out[86]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category',
       'project_subject_categories', 'project_subject_subcategories',
       'project_title', 'project_essay_1', 'project_essay_2',
       'project_essay_3', 'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved'],
      dtype='object')
```

Preprocessing of project_subject_categories

In [87]:
```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverflow.c

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-
cat_list = []
for i in catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science",
        if 'The' in j.split(): # this will split each of the catogory based on sp
            j=j.replace('The','') # if we have the words "The" we are going to r
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trail
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())
```

In [88]:
```
project_data['clean_categories'] = cat_list
project_data.drop(['project_subject_categories'], axis=1, inplace=True)
project_data
```

_datetime	project_grade_category	project_subject_subcategories	project_title	project_essay_1	projec
13:43:57	Grades PreK-2	ESL, Literacy	Educational Support for English Learners at Home	My students are English learners that are work...	\"The li language a
09:22:10	Grades 6-8	Civics & Government, Team Sports	Wanted: Projector for Hungry Learners	Our students arrive to our school eager to lea...	The projector our schoo
12:03:56	Grades 6-8	Health & Wellness, Team Sports	Soccer Equipment for AWESOME Middle School Stu...	\r\n\"True champions aren't always the ones th...	The stud campus com
21:16:17	Grades PreK-2	Literacy, Mathematics	Techie Kindergarteners	I work at a unique school filled with both ESL...	My students poverty con

In [89]:
```
from collections import Counter
my_counter = Counter()
for word in project_data['clean_categories'].values:
    my_counter.update(word.split())

cat_dict = dict(my_counter)
sorted_cat_dict = dict(sorted(cat_dict.items(), key=lambda kv: kv[1]))
```

Preprocessing of project_subject_subcategories

In [90]:
```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.co

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-

sub_cat_list = []
for i in sub_catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science",
        if 'The' in j.split(): # this will split each of the catogory based on sp
            j=j.replace('The','') # if we have the words "The" we are going to re
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trail
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())
```

In [91]:
```
project_data['clean_subcategories'] = sub_cat_list
project_data.drop(['project_subject_subcategories'], axis=1, inplace=True)
project_data.tail()
```

Out[91]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	proj
109243	38267	p048540	fadf72d6cd83ce6074f9be78a6fcd374	Mr.	MO	
109244	169142	p166281	1984d915cc8b91aa16b4d1e6e39296c6	Ms.	NJ	
109245	143653	p155633	cdbfd04aa041dc6739e9e576b1fb1478	Mrs.	NJ	
109246	164599	p206114	6d5675dbfafa1371f0e2f6f1b716fe2d	Mrs.	NY	
109247	128381	p191189	ca25d5573f2bd2660f7850a886395927	Ms.	VA	

In [92]:

```
# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4
my_counter = Counter()
for word in project_data['clean_subcategories'].values:
    my_counter.update(word.split())

sub_cat_dict = dict(my_counter)
sorted_sub_cat_dict = dict(sorted(sub_cat_dict.items(), key=lambda kv: kv[1]))
```

Grade conversion

In [93]:

```

#reference :https://github.com/robinsones/Predicting-Sucess-on-DonorsChoose/blob/

conv_grade_dict = {'Grades PreK-2' : 1.,'Grades 3-5' : 2.,'Grades 6-8':3.,'Grade
project_data['grade_level'] = project_data["project_grade_category"].apply(conv_


project_data['project_grade_category'] = [ gr_ca.replace(' ','_') for gr_ca in pr
project_data['project_grade_category'] = [ gr_ca.replace('-','_') for gr_ca in pr
```

In [94]:

```
project_data['project_grade_category'].value_counts()
```

Out[94]:

```
Grades_PreK_2    44225
Grades_3_5       37137
Grades_6_8       16923
Grades_9_12      10963
Name: project_grade_category, dtype: int64
```

Teacher prefix

In [95]:

```
project_data['teacher_prefix'] = project_data['teacher_prefix'].fillna('null')
```

In [96]:

```
def replace_cate(lst):          # Removing (.) in Mrs.
    return lst.replace('.','')


project_data['teacher_prefix']= project_data['teacher_prefix'].astype(str).apply
```

In [97]:

```
preprocessed_teacher_prefix = []

for teach_prefix in tqdm(project_data["teacher_prefix"]):
    preprocessed_teacher_prefix.append(teach_prefix.strip())
```

```
100%|██████████████████████████████████████████████████████████████████████| 10
9248/109248 [00:00<00:00, 807346.73it/s]
```

feature enginnering

In [98]:

```
project_data["essay"] = project_data["project_essay_1"].map(str) +\
                        project_data["project_essay_2"].map(str) + \
                        project_data["project_essay_3"].map(str) + \
                        project_data["project_essay_4"].map(str)
```

In [99]:

```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    phrase = re.sub('[^A-Za-z0-9]+', ' ', phrase)
    return phrase
```

In [100]:

```
# https://gist.github.com/sebleier/554280
# Removing stopwords
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', 
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itse
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'tha
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'ha
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 't
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'of
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than'
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should'v
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "d
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma'
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn
            'won', "won't", 'wouldn', "wouldn't"]
```

In [101]:
```
# https://stackoverflow.com/a/47091490/4084039
def feature_engineering(data):
    from tqdm import tqdm
    preprocessed_essays = []
    # tqdm is for printing the status bar
    for sentance in tqdm(data.values):
        pharse = decontracted(sentance)
        pharse = pharse.replace('\\r', ' ')
        pharse = pharse.replace('\\"', ' ')
        pharse = pharse.replace('\\n', ' ')
        pharse = pharse.replace('nan',' ')
        pharse = re.sub('[^A-Za-z0-9]+', ' ', pharse)
        # https://gist.github.com/sebleier/554280
        pharse = ' '.join(e for e in pharse.split() if e.lower() not in stopword
        preprocessed_essays.append(pharse.lower().strip())
    return preprocessed_essays
```

In [102]:
```
preprocessed_essays=feature_engineering(project_data['essay'])
```

```
100%|████████████████████████████████████████████████████████|
109248/109248 [02:43<00:00, 666.22it/s]
```

In [103]:
```
project_data['essay'] = preprocessed_essays
project_data.drop(['project_essay_1'], axis=1, inplace=True)
project_data.drop(['project_essay_2'], axis=1, inplace=True)
project_data.drop(['project_essay_3'], axis=1, inplace=True)
project_data.drop(['project_essay_4'], axis=1, inplace=True)
```

In [104]:
```
# Fearturing the Project_title as well
preprocessed_project_title=feature_engineering(project_data['project_title'])
```

```
100%|██████████████████████████████████████████████████████| 1
09248/109248 [00:07<00:00, 15255.92it/s]
```

In [105]:
```
project_data['featured_title']=preprocessed_project_title
project_data.drop(['project_title'], axis=1, inplace=True)
```

Merging the data

In [106]:
```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).
project_data = pd.merge(project_data, price_data, on='id', how='left')
```

In [107]:
```
project_data.drop(['project_resource_summary'], axis=1, inplace=True)
project_data.drop(['Unnamed: 0'], axis=1, inplace=True)
project_data.drop(['id'], axis=1, inplace=True)
project_data.drop(['teacher_id'], axis=1, inplace=True)
```

In [108]:

```
project_data.head()
```

Out[108]:

	teacher_prefix	school_state	project_submitted_datetime	project_grade_category	teacher_numbe
0	Mrs	IN	2016-12-05 13:43:57	Grades_PreK_2	
1	Mr	FL	2016-10-25 09:22:10	Grades_6_8	
2	Ms	AZ	2016-08-31 12:03:56	Grades_6_8	
3	Mrs	KY	2016-10-06 21:16:17	Grades_PreK_2	
4	Mrs	TX	2016-07-11 01:10:09	Grades_PreK_2	

Splitting data

In [109]:

```
#https://stackoverflow.com/questions/29763620
X=project_data.loc[:, project_data.columns != 'project_is_approved']
y=project_data['project_is_approved']
X.shape
```

Out[109]:

```
(109248, 12)
```

In [110]:

```
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,stratify = 

print(X_train.shape)
print(X_test.shape)

print(y_train.shape)
print(y_test.shape)
```

```
(73196, 12)
(36052, 12)
(73196,)
(36052,)
```

Vectorizing the data

Categorical data

In [111]:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(lowercase=False, binary=True)
vectorizer.fit(X_train['clean_categories'].values)
feature_names_bow=[]
feature_names_tfidf=[]
# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_categories=vectorizer.transform(X_train['clean_categories'].values
X_test_clean_categories=vectorizer.transform(X_test['clean_categories'].values)


print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",X_train_clean_categories.shape)
feature_names_bow.extend(vectorizer.get_feature_names())
feature_names_tfidf.extend(vectorizer.get_feature_names())
```

```
['AppliedLearning', 'Care_Hunger', 'Health_Sports', 'History_Civics', 'Literacy
_Language', 'Math_Science', 'Music_Arts', 'SpecialNeeds', 'Warmth']
Shape of matrix after one hot encodig  (73196, 9)
```

In [112]:

```
vectorizer = CountVectorizer(lowercase=False, binary=True)
vectorizer.fit(X_train['clean_subcategories'].values)

# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_sub_categories=vectorizer.transform(X_train['clean_subcategories']
X_test_clean_sub_categories=vectorizer.transform(X_test['clean_subcategories'].v


print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",X_train_clean_sub_categories.shap
feature_names_bow.extend(vectorizer.get_feature_names())
feature_names_tfidf.extend(vectorizer.get_feature_names())
```

```
['AppliedSciences', 'Care_Hunger', 'CharacterEducation', 'Civics_Government',
'College_CareerPrep', 'CommunityService', 'ESL', 'EarlyDevelopment', 'Economic
s', 'EnvironmentalScience', 'Extracurricular', 'FinancialLiteracy', 'ForeignLan
guages', 'Gym_Fitness', 'Health_LifeScience', 'Health_Wellness', 'History_Geogr
aphy', 'Literacy', 'Literature_Writing', 'Mathematics', 'Music', 'NutritionEduc
ation', 'Other', 'ParentInvolvement', 'PerformingArts', 'SocialSciences', 'Spec
ialNeeds', 'TeamSports', 'VisualArts', 'Warmth']
Shape of matrix after one hot encodig  (73196, 30)
```

In [113]:

```
vectorizer = CountVectorizer(lowercase=False, binary=True)
vectorizer.fit(X_train['school_state'].values)

# we use the fitted CountVectorizer to convert the text to vector
X_train_skl_state=vectorizer.transform(X_train['school_state'].values)
X_test_skl_state=vectorizer.transform(X_test['school_state'].values)


print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",X_train_skl_state.shape)
feature_names_bow.extend(vectorizer.get_feature_names())
feature_names_tfidf.extend(vectorizer.get_feature_names())
```

```
['AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA', 'HI', 'IA',
'ID', 'IL', 'IN', 'KS', 'KY', 'LA', 'MA', 'MD', 'ME', 'MI', 'MN', 'MO', 'MS',
'MT', 'NC', 'ND', 'NE', 'NH', 'NJ', 'NM', 'NV', 'NY', 'OH', 'OK', 'OR', 'PA',
'RI', 'SC', 'SD', 'TN', 'TX', 'UT', 'VA', 'VT', 'WA', 'WI', 'WV', 'WY']
Shape of matrix after one hot encodig  (73196, 51)
```

In [114]:

```
vectorizer = CountVectorizer(lowercase=False, binary=True)
vectorizer.fit(X_train['teacher_prefix'].values.astype("U"))

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_prefix=vectorizer.transform(X_train['teacher_prefix'].values.asty
X_test_teacher_prefix=vectorizer.transform(X_test['teacher_prefix'].values.astyp

print("Shape of matrix after one hot encodig ",X_train_teacher_prefix.shape)
feature_names_bow.extend(vectorizer.get_feature_names())
feature_names_tfidf.extend(vectorizer.get_feature_names())
```

```
Shape of matrix after one hot encodig  (73196, 6)
```

In [115]:

```
vectorizer = CountVectorizer(lowercase=False, binary=True)
vectorizer.fit(X_train['project_grade_category'].values)

# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_level=vectorizer.transform(X_train['project_grade_category'].value
X_test_grade_level=vectorizer.transform(X_test['project_grade_category'].values)


print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",X_train_grade_level.shape)
feature_names_bow.extend(vectorizer.get_feature_names())
feature_names_tfidf.extend(vectorizer.get_feature_names())
```

```
['Grades_3_5', 'Grades_6_8', 'Grades_9_12', 'Grades_PreK_2']
Shape of matrix after one hot encodig  (73196, 4)
```

Text Data

BOW

In [116]:

```
bow_essay_vectorizer = CountVectorizer(min_df=15)
bow_essay_vectorizer.fit(X_train['essay'])

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


print("Shape of matrix after one hot encodig ",X_train_essay_bow.shape)
print("Shape of matrix after one hot encodig ",X_test_essay_bow.shape)

feature_names_bow.extend(bow_essay_vectorizer.get_feature_names())
```

```
Shape of matrix after one hot encodig  (73196, 12248)
Shape of matrix after one hot encodig  (36052, 12248)
```

In [117]:

```
bow_title_vectorizer = CountVectorizer(min_df=15)
bow_title_vectorizer.fit(X_train['featured_title'])

# we use the fitted CountVectorizer to convert the text to vector
X_train_featured_title_bow=bow_title_vectorizer.transform(X_train['featured_titl
X_test_featured_title_bow=bow_title_vectorizer.transform(X_test['featured_title'


print(" after one hot encodig ",X_train_featured_title_bow.shape)
feature_names_bow.extend(bow_title_vectorizer.get_feature_names())
```

```
 after one hot encodig  (73196, 1961)
```

TFIDF vectorizer

In [118]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=15)
vectorizer.fit(X_train['essay'])

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_tfidf=vectorizer.transform(X_train['essay'].values)
X_test_essay_tfidf=vectorizer.transform(X_test['essay'].values)


print("Shape of matrix after one hot encodig ",X_train_essay_tfidf.shape)
feature_names_tfidf.extend(vectorizer.get_feature_names())
```

```
Shape of matrix after one hot encodig  (73196, 12248)
```

In [119]:

```
vectorizer = TfidfVectorizer(min_df=15)

# you can vectorize the title also
# before you vectorize the title make sure you preprocess it
vectorizer.fit(X_train['featured_title'])

# we use the fitted CountVectorizer to convert the text to vector
X_train_featured_title_tfidf=vectorizer.transform(X_train['featured_title'].valu
X_test_featured_title_tfidf=vectorizer.transform(X_test['featured_title'].values


print("Shape of matrix after one hot encodig ",X_train_featured_title_tfidf.shap
feature_names_tfidf.extend(vectorizer.get_feature_names())
```

```
Shape of matrix after one hot encodig  (73196, 1961)
```

In [120]:

```
from sklearn.preprocessing import Normalizer

def normalizer(a,b):
    normalizer = Normalizer()
    normalizer.fit(a[b].values.reshape(1,-1))
    out=normalizer.transform(a[b].values.reshape(1,-1))
    return out

X_train_price_standardized=normalizer(X_train,'price')
X_test_price_standardized=normalizer(X_test,'price')

print(X_train_price_standardized.shape, y_train.shape)
print(X_test_price_standardized.shape, y_test.shape)
```

```
(1, 73196) (73196,)
(1, 36052) (36052,)
```

In [121]:

```
X_train_price_standardized = X_train_price_standardized.reshape(-1,1)
X_test_price_standardized = X_test_price_standardized.reshape(-1,1)
```

In [122]:

```
X_train_prev_proj=normalizer(X_train,'teacher_number_of_previously_posted_projec
X_test_prev_proj=normalizer(X_test,'teacher_number_of_previously_posted_projects
```

In [123]:
```
X_train_prev_proj = X_train_prev_proj.reshape(-1,1)
X_test_prev_proj = X_test_prev_proj.reshape(-1,1)
```

stacking all the featured variable

BOW

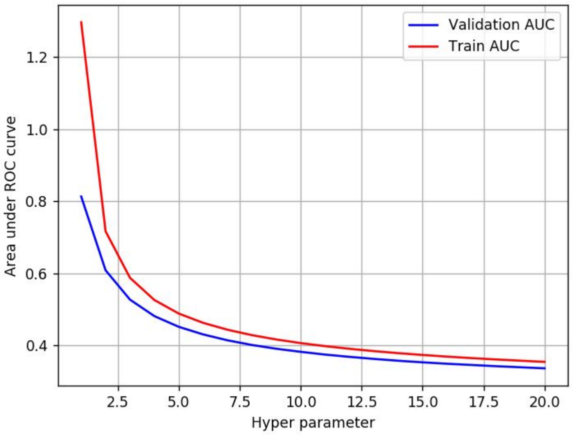
In [124]:
```
#Reference https://stackoverflow.com/a/19710648/4084039

from scipy.sparse import hstack

X_train_bow = hstack((X_train_clean_categories, X_train_clean_sub_categories,X_t
            X_train_grade_level,X_train_prev_proj,X_train_price_standardized,
            X_train_essay_bow,X_train_featured_title_tfidf
            )).tocsr()


X_test_bow = hstack((X_test_clean_categories, X_test_clean_sub_categories,X_test_
            X_test_grade_level,X_test_prev_proj,X_test_price_standardized,
            X_test_essay_bow,X_test_featured_title_bow
            )).tocsr()
```

TFIDF

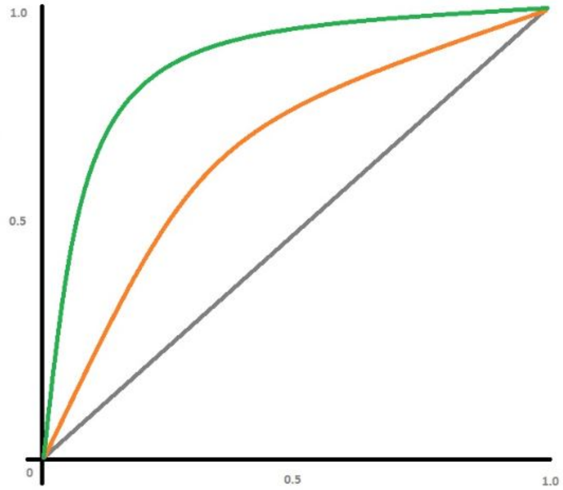
In [125]:
```
X_train_tfidf = hstack((X_train_clean_categories, X_train_clean_sub_categories,X_
            X_train_grade_level,X_train_prev_proj,X_train_price_standardized,
            X_train_essay_tfidf,X_train_featured_title_tfidf
            )).tocsr()


X_test_tfidf = hstack((X_test_clean_categories, X_test_clean_sub_categories,X_te
            X_test_grade_level,X_test_prev_proj,X_test_price_standardized,
            X_test_essay_tfidf,X_test_featured_title_bow
            )).tocsr()
```

1. **Apply Multinomial NB on these feature sets**
    - Set 1: categorical, numerical features + preprocessed_eassay (BOW)
    - Set 2: categorical, numerical features + preprocessed_eassay (TFIDF)
2. **The hyper paramter tuning(find best alpha:smoothing parameter)**
    - Find the best hyper parameter which will give the maximum AUC (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) value
    - find the best hyper paramter using k-fold cross validation(use GridsearchCV or RandomsearchCV)/simple cross validation data (write for loop to iterate over hyper parameter values)
    -
3. **Representation of results**

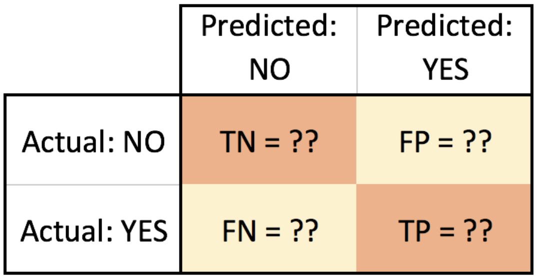
- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure



- Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.

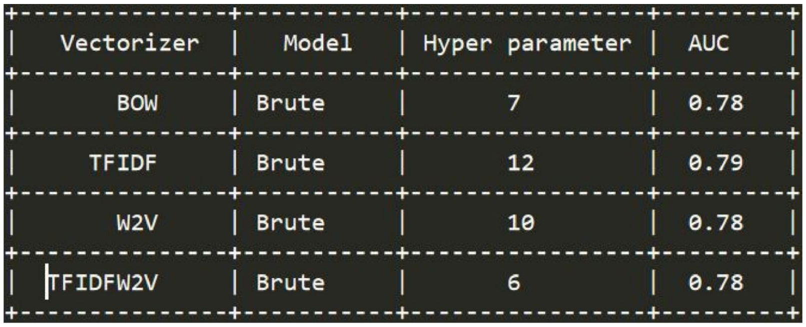


- Along with plotting ROC curve, you need to print the [confusion matrix (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/)](https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points



	Predicted: NO	Predicted: YES
Actual: NO	TN = ??	FP = ??
Actual: YES	FN = ??	TP = ??

4. fine the top 20 features from either from feature Set 1 or feature Set 2 using absolute values of \`feature_log_prob_ \` parameter of \`MultinomialNB\` (https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html) and print their corresponding feature names
5. You need to summarize the results at the end of the notebook, summarize it in the table format



```
+--------------+----------+-----------------+---------+
|  Vectorizer  |  Model   | Hyper parameter |  AUC    |
+--------------+----------+-----------------+---------+
|     BOW      | Brute    |        7        |  0.78   |
+--------------+----------+-----------------+---------+
|    TFIDF     | Brute    |       12        |  0.79   |
+--------------+----------+-----------------+---------+
|     W2V      | Brute    |       10        |  0.78   |
+--------------+----------+-----------------+---------+
|  TFIDFW2V    | Brute    |        6        |  0.78   |
+--------------+----------+-----------------+---------+
```

2. Naive Bayes

Naive Bayes on BOW

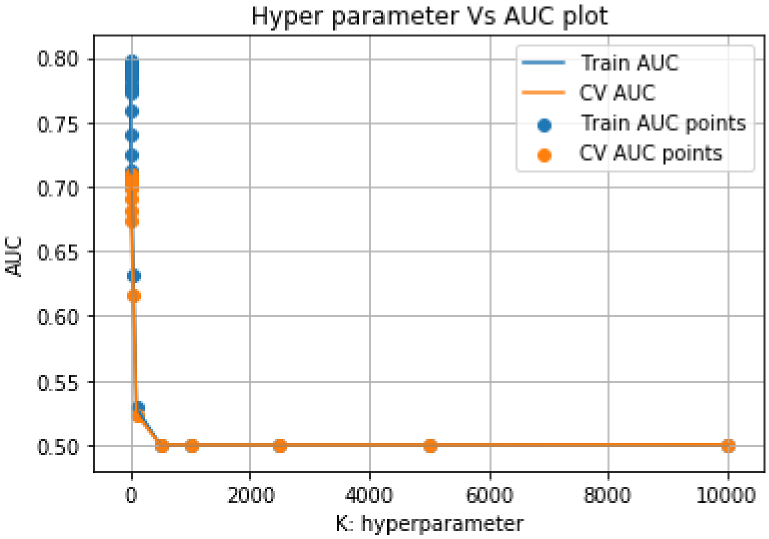
In [126]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
nb = MultinomialNB(class_prior=[0.5,0.5])

parameters = {'alpha':[0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.0

clf = GridSearchCV(nb, parameters, cv= 10, scoring='roc_auc',return_train_score=

clf.fit(X_train_bow, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std = clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std = clf.cv_results_['std_test_score']
```

In [127]:

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

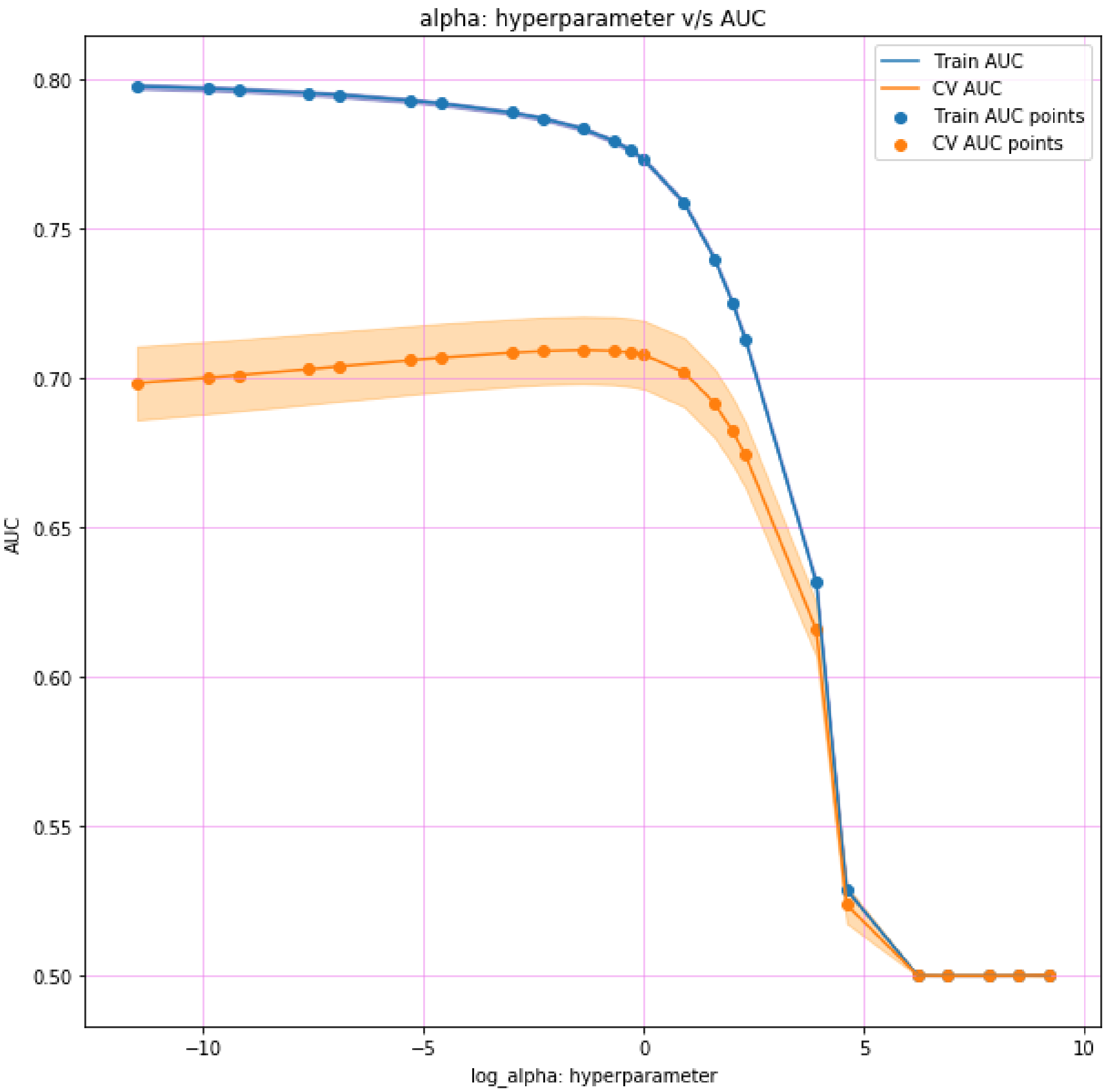
In [128]:
```
plt.plot(alphas, train_auc, label='Train AUC')
plt.plot(alphas, cv_auc, label='CV AUC')
plt.scatter(alphas, train_auc, label='Train AUC points')
plt.scatter(alphas, cv_auc, label='CV AUC points')
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
```



For better visualization

In [129]:

```
import math
log_alphas =[]

for a in alphas:
    log_a = math.log(a)
    log_alphas.append(log_a)

plt.figure(figsize=(10,10))

plt.plot(log_alphas, train_auc, label='Train AUC')
# Reference: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_alphas,train_auc - train_auc_std,train_auc + train_au

plt.plot(log_alphas, cv_auc, label='CV AUC')
# Reference: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_alphas,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=

plt.scatter(log_alphas, train_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")
plt.grid(color='violet', linestyle='-', linewidth=0.5)
plt.show()
```



In [130]:
```
print('Best score: ',clf.best_score_)
print('k value: ',clf.best_params_)
print('='*10)
print('Train AUC scores')
print(clf.cv_results_['mean_train_score'])
print('CV AUC scores')
print(clf.cv_results_['mean_test_score'])
```

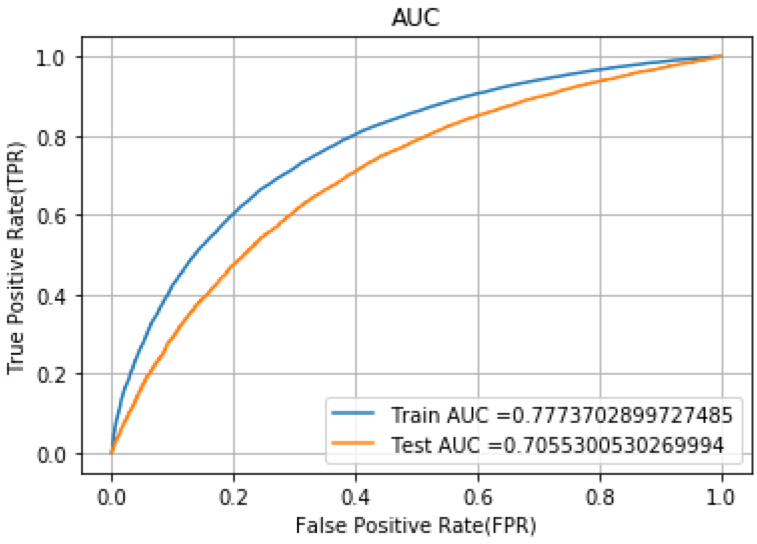
```
Best score:  0.7093242017222475
k value:  {'alpha': 0.25}
==========
Train AUC scores
[0.7975712  0.7968507  0.79645798 0.79530333 0.79468004 0.79285073
 0.79185355 0.78876725 0.78688626 0.78338901 0.77935214 0.77604757
 0.7730927  0.7585098  0.73970341 0.72493787 0.71296303 0.63167386
 0.52867894 0.50007413 0.49999195 0.49999195 0.50002901 0.50020496]
CV AUC scores
[0.69823915 0.70002421 0.70086606 0.70291634 0.70382773 0.70589896
 0.70674133 0.70842947 0.7089716  0.7093242  0.70905286 0.70847602
 0.70770665 0.70186748 0.6914289  0.68215054 0.67427347 0.61590507
 0.52368363 0.50007412 0.49999195 0.49999195 0.500029   0.50020494]
```

In [131]:

```
def pred_prob(clf, data):
    y_pred = []
    y_pred = clf.predict_proba(data)[:,1]
    return y_pred
```

In [132]:

```
from sklearn.metrics import roc_curve, auc

bow = MultinomialNB(alpha = clf.best_params_['alpha'],class_prior = [0.5,0.5])

bow.fit(X_train_bow, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimat
# not the predicted outputs

y_train_pred = pred_prob(bow,X_train_bow)
y_test_pred = pred_prob(bow,X_test_bow)

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_tpr)
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False Positive Rate(FPR)")
plt.ylabel("True Positive Rate(TPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



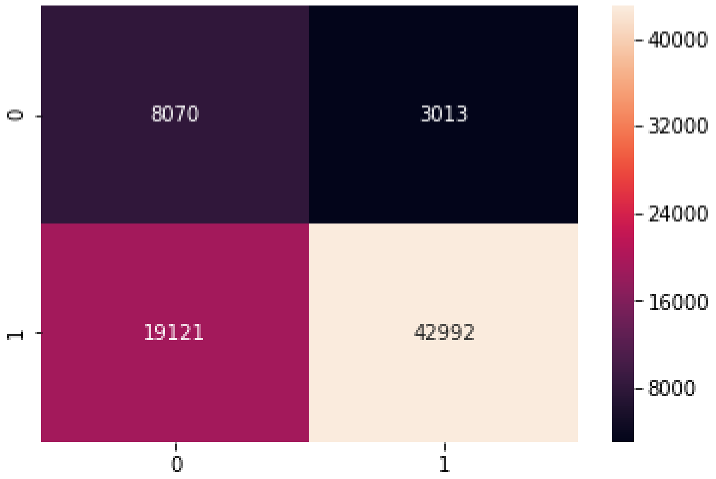
In [133]:

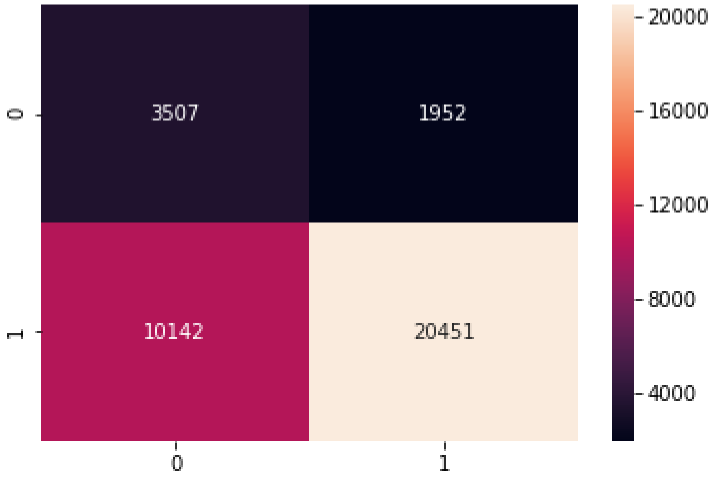
```
def find_best_threshold(threshold, fpr, tpr):
    t = threshold[np.argmax(tpr*(1-fpr))]
    print("tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    return t

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

In [134]:

```
fig = plt.figure()
ax = fig.add_subplot(111)
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
cm = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
sns.heatmap(cm, annot=True, fmt='d')

plt.show(ax)


fig = plt.figure()
ax1 = fig.add_subplot(111)
cm = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
sns.heatmap(cm, annot=True, fmt='d')

plt.show(ax1)
```

```
tpr*(1-fpr) 0.5039893331881307 for threshold 0.564
```





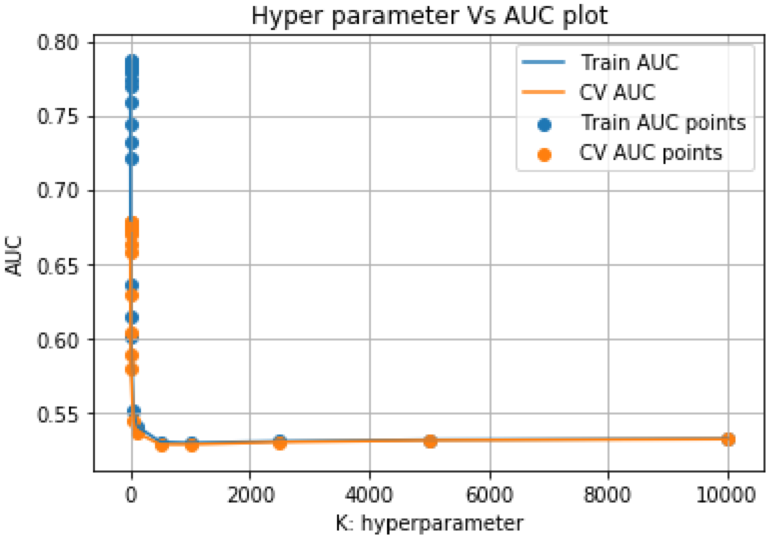
Feature importance

Naive Bayes on TFIDF

In [135]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
nb = MultinomialNB(class_prior=[0.5,0.5])

parameters = {'alpha':[0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.0

clf = GridSearchCV(nb, parameters, cv= 10, scoring='roc_auc',return_train_score=

clf.fit(X_train_tfidf, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std = clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std = clf.cv_results_['std_test_score']
```

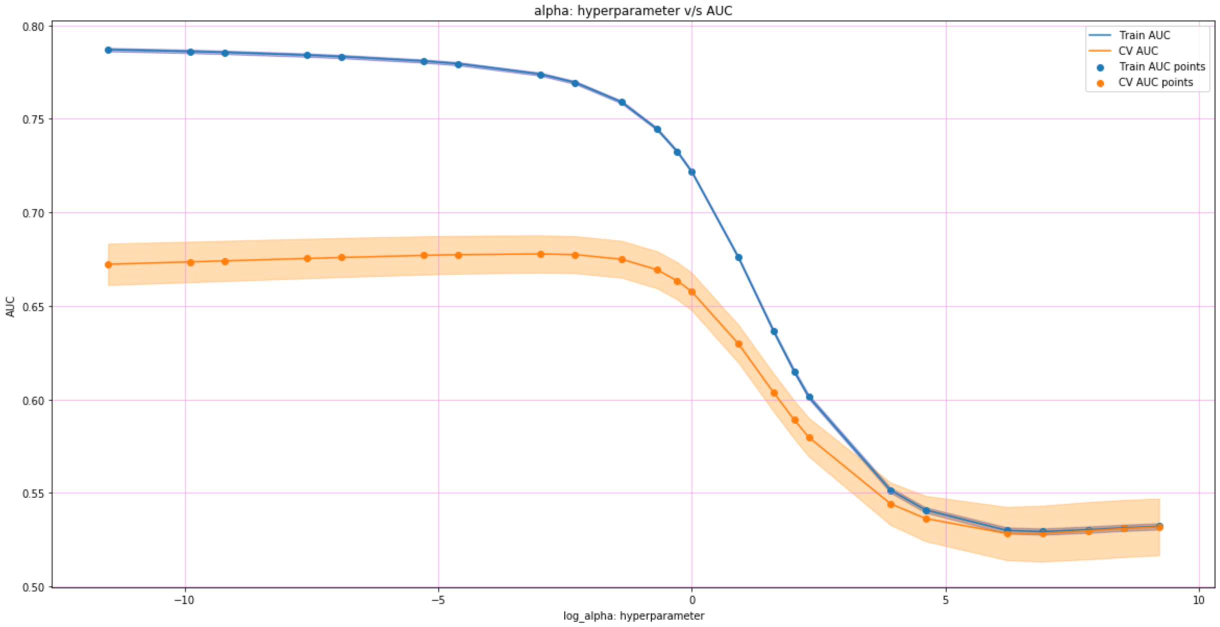
In [136]:
```
results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_alpha'])
alphas =  results['param_alpha']
```

In [137]:
```
plt.plot(alphas, train_auc, label='Train AUC')
plt.plot(alphas, cv_auc, label='CV AUC')
plt.scatter(alphas, train_auc, label='Train AUC points')
plt.scatter(alphas, cv_auc, label='CV AUC points')
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
```



For better visualization

In [138]:
```
import math
log_alphas =[]

for a in alphas:
    log_a = math.log(a)
    log_alphas.append(log_a)

plt.figure(figsize=(20,10))

plt.plot(log_alphas, train_auc, label='Train AUC')
# Reference: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_alphas,train_auc - train_auc_std,train_auc + train_auc

plt.plot(log_alphas, cv_auc, label='CV AUC')
# Reference: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(log_alphas,cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=

plt.scatter(log_alphas, train_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")
plt.grid(color='violet', linestyle='-', linewidth=0.5)
plt.show()
```



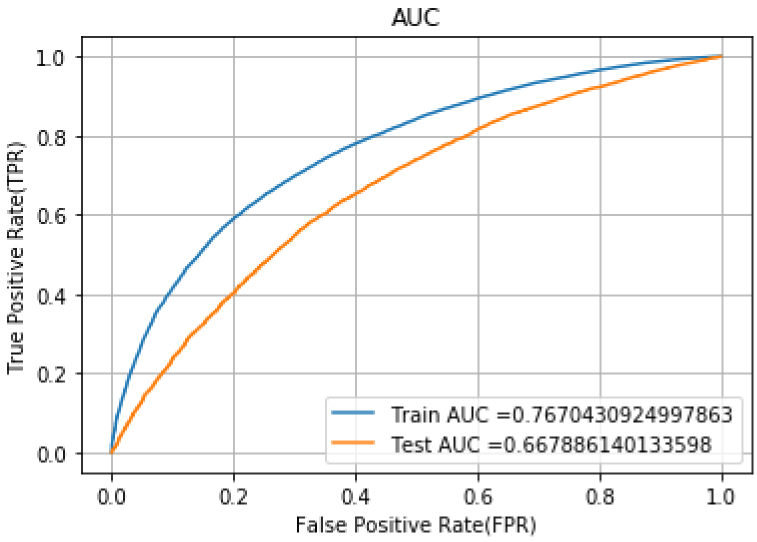
In [139]:

```
print('Best score: ',clf.best_score_)
print('alpha value with best score: ',clf.best_params_)
print('='*15)
print('Train AUC scores')
print(clf.cv_results_['mean_train_score'])
print('CV AUC scores')
print(clf.cv_results_['mean_test_score'])
```

```
Best score:  0.6777556791651996
alpha value with best score:  {'alpha': 0.05}
===============
Train AUC scores
[0.78701077 0.78606751 0.7855635  0.78409958 0.78330828 0.78089883
 0.77946373 0.77391286 0.76942697 0.75896157 0.7446989  0.7324238
 0.7215733  0.67590813 0.63647775 0.61507781 0.60143476 0.55152431
 0.54086434 0.52991115 0.52931366 0.5303797  0.53137071 0.53211499]
CV AUC scores
[0.67222484 0.67349656 0.67404804 0.67533088 0.67585867 0.67696272
 0.67734507 0.67775568 0.67738118 0.67492695 0.66935682 0.66339066
 0.65754199 0.62973419 0.60369677 0.58917663 0.5797888  0.54426486
 0.53630853 0.528248   0.52824237 0.52979121 0.53097804 0.53183034]
```

In [140]:

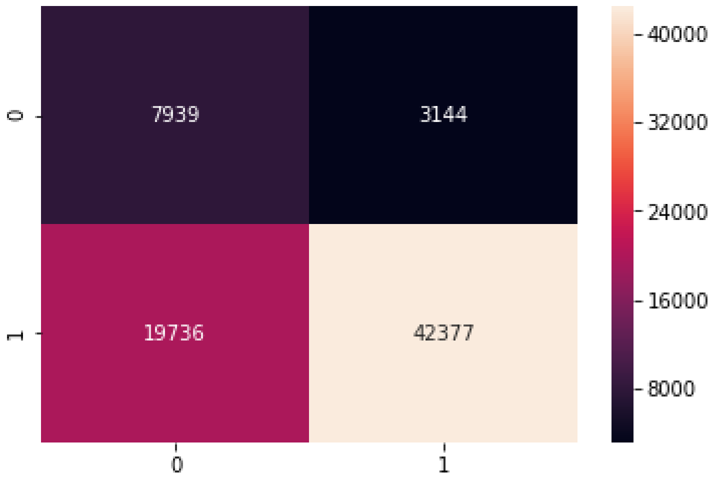
```
from sklearn.metrics import roc_curve, auc

tfidf = MultinomialNB(alpha = clf.best_params_['alpha'],class_prior = [0.5,0.5])
tfidf.fit(X_train_tfidf, y_train)

y_train_pred = pred_prob(tfidf, X_train_tfidf)
y_test_pred = pred_prob(tfidf, X_test_tfidf)

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_tpr)
plt.plot(test_fpr, test_tpr, label="Test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("False Positive Rate(FPR)")
plt.ylabel("True Positive Rate(TPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



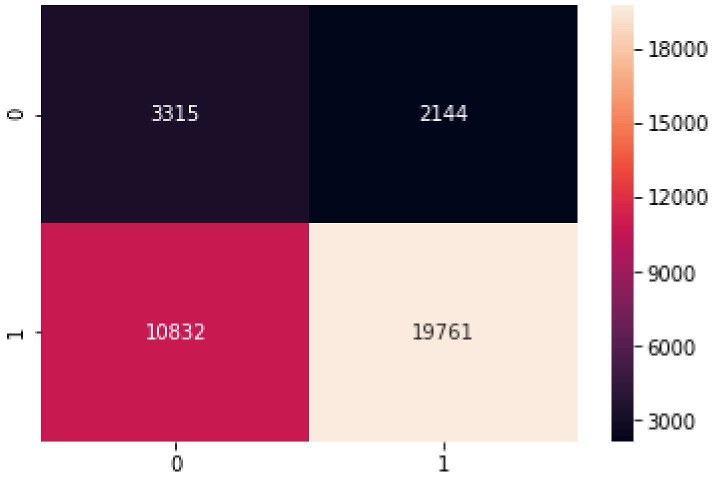
In [141]:

```
fig = plt.figure()
ax = fig.add_subplot(111)
print("Train Data confusion matrix")
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
cm = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
sns.heatmap(cm, annot=True, fmt='d')

plt.show(ax1)


print("Test data confusion matrix")
fig = plt.figure()
ax1 = fig.add_subplot(111)
cm = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
sns.heatmap(cm, annot=True, fmt='d')

plt.show(ax1)
```

```
Train Data confusion matrix
tpr*(1-fpr) 0.48871556538049316 for threshold 0.508
```



```
Test data confusion matrix
```



Feature importance

In [142]:

```
#https://stackoverflow.com/questions/50526898/how-to-get-feature-importance-in-n
neg_class_prob_sorted = bow.feature_log_prob_[0, :].argsort()
pos_class_prob_sorted = bow.feature_log_prob_[1, :].argsort()


print('Top 20 features -Positive_class')
print('='*50)
print(np.take(feature_names_tfidf, pos_class_prob_sorted[-20:]))

print('Top 20 features - negative_class')
print('='*50)
print(np.take(feature_names_tfidf, neg_class_prob_sorted[-20:]))
```

```
Top 20 features -Positive_class
==================================================
['bookshelves' 'tectonics' 'wow' 'classes' 'comers' 'abound' 'days'
 'loved' 'used' 'reads' 'neediest' 'workbook' 'mapping' 'helper' 'learner'
 'notation' 'classwork' 'learns' 'schoolers' 'studies']
Top 20 features - negative_class
==================================================
['yearbooks' 'classes' 'used' 'days' 'skip' 'reads' 'math' 'abound'
 'loved' 'comers' 'workbook' 'neediest' 'mapping' 'helper' 'learner'
 'notation' 'classwork' 'learns' 'schoolers' 'studies']
```

In [143]:

```
#https://stackoverflow.com/questions/50526898/how-to-get-feature-importance-in-n
neg_class_prob_sorted = bow.feature_log_prob_[0, :].argsort()
pos_class_prob_sorted = bow.feature_log_prob_[1, :].argsort()


print('Top 20 features -Positive_class')
print('='*50)
print(np.take(feature_names_bow, pos_class_prob_sorted[-20:]))

print('Top 20 features - negative_class')
print('='*50)
print(np.take(feature_names_bow, neg_class_prob_sorted[-20:]))
```

```
Top 20 features -Positive_class
==================================================
['bookshelves' 'tectonics' 'wow' 'classes' 'comers' 'abound' 'days'
 'loved' 'used' 'reads' 'neediest' 'workbook' 'mapping' 'helper' 'learner'
 'notation' 'classwork' 'learns' 'schoolers' 'studies']
Top 20 features - negative_class
==================================================
['yearbooks' 'classes' 'used' 'days' 'skip' 'reads' 'math' 'abound'
 'loved' 'comers' 'workbook' 'neediest' 'mapping' 'helper' 'learner'
 'notation' 'classwork' 'learns' 'schoolers' 'studies']
```

Summary

In [145]:
```
#reference :https://stackoverflow.com/questions/8356501/python-format-tabular-ou
from beautifultable import BeautifulTable
table = BeautifulTable()
table.column_headers= ["Vectorizer", "model", "Hyper parameter","AUC"]
table.append_row(["BOW", "Naive Bayes", "0.25", "0.7056"])
table.append_row(["TFIDF", "Naive Bayes", "0.05", "0.6678"])
print(table)
```

```
+------------+-------------+-----------------+-------+
| Vectorizer |    model    | Hyper parameter |  AUC  |
+------------+-------------+-----------------+-------+
|    BOW     | Naive Bayes |      0.25       | 0.706 |
+------------+-------------+-----------------+-------+
|   TFIDF    | Naive Bayes |      0.05       | 0.668 |
+------------+-------------+-----------------+-------+
```

- Naive Bayes is an eager learning classifier and it is much faster than K-NN ,besides gives better AUC than KNN(on
  comparison with KNN model.
- From the Better performance we can conclude that Navie Bayes is better for text data.
- BoW seems to performance better than TFIDF vectorizer for this dataset.