DonorsChoose-Naive Bayes

Objective: To build a Naive Bayes classifier and predict whether the project proposed ny the teacher will be approved or not.

Description:Naive Bayes is classification algorithms based on Bayes' Theorem.

Assumptions: No pair of features are dependent.

Importing packages

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

Reading the data

In [2]:

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

In [3]:

```
print("Number of data points in train data", project_data.shape)
print('-'*50)
print("Attributes :", project_data.columns.values)
project_data.head(2)
```

```
Number of data points in train data (109248, 17)
--------------------------------------------------
Attributes : ['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']
```

Out[3]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	

Handling NaN values in Teacher Prefix column of Project data : Replacing NaN value with most frequent occuring "Teacher's prefix"

In [4]:

```
a = project_data['teacher_prefix'].mode().values
```

In [5]:

```
project_data['teacher_prefix'] = project_data['teacher_prefix'].fillna(a[0])
```

In [6]:

```
#Total number of null values in each column
project_data.isnull().sum(axis = 0)
```

Out[6]:

```
Unnamed: 0                                           0
id                                                   0
teacher_id                                           0
teacher_prefix                                       0
school_state                                         0
project_submitted_datetime                           0
project_grade_category                               0
project_subject_categories                           0
project_subject_subcategories                        0
project_title                                        0
project_essay_1                                      0
project_essay_2                                      0
project_essay_3                                 105490
project_essay_4                                 105490
project_resource_summary                             0
teacher_number_of_previously_posted_projects         0
project_is_approved                                  0
dtype: int64
```

In [7]:

```
print("Number of data points in train data", resource_data.shape)
print('-'*50)
print("Attributes: ", resource_data.columns.values)
resource_data.head(2)
```

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

Out[7]:

	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 [8]:

```
# how to replace elements in list python: https://stackoverflow.com/a/2582163/4084039
cols = ['Date' if x=='project_submitted_datetime' else x for x in list(project_data.columns


#sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/4084039
project_data['Date'] = pd.to_datetime(project_data['project_submitted_datetime'])
project_data.drop('project_submitted_datetime', axis=1, inplace=True)
project_data.sort_values(by=['Date'], inplace=True)


# how to reorder columns pandas python: https://stackoverflow.com/a/13148611/4084039
project_data = project_data[cols]


project_data.head(2)
```

Out[8]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	00:
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	00:

Preprocessing of Project subject Categories

In [9]:

```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/473019

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
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", "Warmth", "
        if 'The' in j.split(): # this will split each of the catogory based on space "Math
            j=j.replace('The','') # if we have the words "The" we are going to replace it w
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())

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

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 sub categories

In [10]:

```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/473019

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

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", "Warmth", "
        if 'The' in j.split(): # this will split each of the catogory based on space "Math
            j=j.replace('The','') # if we have the words "The" we are going to replace it w
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

project_data['clean_subcategories'] = sub_cat_list
project_data.drop(['project_subject_subcategories'], axis=1, inplace=True)

# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
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]))
```

Preprocessing Project Essay

In [11]:

```
# merge two column text dataframe:
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 [12]:

```
# 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)
    return phrase

# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they'
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'l
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had',
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'u
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'd
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over',
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', '
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'v
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now',
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'do
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn'
            'won', "won't", 'wouldn', "wouldn't"]


from tqdm import tqdm
preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = sent.lower()
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split(" ") if e not in stopwords)
    preprocessed_essays.append(sent.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████████████████████
██| 109248/109248 [02:28<00:00, 735.59it/s]
```

In [13]:

```
# placing the preprocessed essay into the dataframe
project_data['clean_essays'] = preprocessed_essays
project_data.drop(['essay'], axis=1, inplace=True)
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 [14]:

```
#Printing random cleaned essay
project_data['clean_essays'].values[31]
```

Out[14]:

```
'class self contained program students autism work school 76 students reduce
d free lunch need enhance social skills research studies indicate playing mu
sical instrument improve major components social skills teacher moore oklaho
ma 76 school population reduced free lunch come homes families not able prov
ide appropriate adequate educational experience children program self contai
ned students mild severe autism teach nine students whose age range 5 12 yea
rs old 77 non verbal rest limited vocabularies might know children autism la
ck many different skills including social skills live world not make eye con
tact important component social interaction extremely difficult perceive wor
ld others perception cannot relate classmate friends even siblings teachers
try find tools teach develop important social skills example hard participat
e group activities play parts create harmony team students kids classical au
tism spectrum disorder tremendous difficulty develop social skills turn taki
ng social play games group activities etc believe design musical games activ
ities help build social skills strongly believe exploring playing musical in
struments group provide students following social benefits 1 learn part grou
p cooperation classmates 2 see hear learn small roles playing group create u
nique pleasant music whole 3 learn patiently wait turn without exhibiting in
appropriate social behaviors 4 build confidence important ingredient strengt
h social skills 5 learn classmates interests different instruments 6 learn i
nteract friends exchanging instruments musical sessions donation project hel
p students build important social skills confidence self determination found
ations developing academic skills donation project provides students dynamic
learning environment fresh motivational project gives students tools make le
arning teaching fun enthusiastic also gives opportunity go beyond academic c
oncepts'
```

Preprocessing Project Title

In [15]:

```
from tqdm import tqdm
preprocessed_titles = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['project_title'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = sent.lower()
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_titles.append(sent.lower().strip())
```

```
100%|██████████████████████████████████████████████████████████████████████████████
| 109248/109248 [00:06<00:00, 17430.25it/s]
```

In [16]:

```
# placing the preprocessed essay into the dataframe
project_data['clean_titles'] = preprocessed_titles
project_data.drop(['project_title'], axis=1, inplace=True)
```

In [17]:

```
#Printing random cleaned title
project_data['clean_titles'].values[9]
```

Out[17]:

```
'dash dot robotic duo needed'
```

Merging Price data on Project data - (Left join)

In [18]:

```
# reference : https://stackoverflow.com/questions/22407798/how-to-reset-a-dataframes-indexe
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index
price_data.head(2)
```

Out[18]:

	id	price	quantity
0	p000001	459.56	7
1	p000002	515.89	21

In [19]:

```
# join two dataframes(project_data and price_data) in python
# reference : https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.m
project_data = pd.merge(project_data, price_data, on='id', how='left')
```

Splitting the data and starifying the smplings

In [20]:

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

print(X.shape)
print(y.shape)
```

```
(109248, 15)
(109248,)
```

In [21]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_spli
#https://stackoverflow.com/questions/34842405/parameter-stratify-from-method-train-test-spl
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, shuffle=Flase)#
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify = y) # t
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify

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

```
(49041, 15) (49041,)
(24155, 15) (24155,)
(36052, 15) (36052,)
```

Preparing data for Model

One Hot Encoding of categorical data

In [22]:

```
# we use count vectorizer to convert the values into one hot encoded features

# Vectorizing "clean_categories"
from sklearn.feature_extraction.text import CountVectorizer
vectorizer_sbj = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False,
vectorizer_sbj.fit(X_train['clean_categories'].values)

X_train_categories_one_hot = vectorizer_sbj.transform(X_train['clean_categories'].values)
X_cv_categories_one_hot = vectorizer_sbj.transform(X_cv['clean_categories'].values)
X_test_categories_one_hot = vectorizer_sbj.transform(X_test['clean_categories'].values)

print("After verctorizing")
print(X_train_categories_one_hot.shape, y_train.shape)
print(X_cv_categories_one_hot.shape, y_cv.shape)
print(X_test_categories_one_hot.shape, y_test.shape)

print(vectorizer_sbj.get_feature_names())
```

```
After verctorizing
(49041, 9) (49041,)
(24155, 9) (24155,)
(36052, 9) (36052,)
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning',
'SpecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
```

In [23]:

```
# Vectorizing "clean_subcategories"
vectorizer_sub_sbj = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase
vectorizer_sub_sbj.fit(X_train['clean_subcategories'].values)

X_train_sub_categories_one_hot = vectorizer_sub_sbj.transform(X_train['clean_subcategories'
X_cv_sub_categories_one_hot = vectorizer_sub_sbj.transform(X_cv['clean_subcategories'].valu
X_test_sub_categories_one_hot = vectorizer_sub_sbj.transform(X_test['clean_subcategories'].

print("After verctorizing")
print(X_train_sub_categories_one_hot.shape, y_train.shape)
print(X_cv_sub_categories_one_hot.shape, y_cv.shape)
print(X_test_sub_categories_one_hot.shape, y_test.shape)

print(vectorizer_sub_sbj.get_feature_names())
```

```
After verctorizing
(49041, 30) (49041,)
(24155, 30) (24155,)
(36052, 30) (36052,)
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement',
'Extracurricular', 'Civics_Government', 'ForeignLanguages', 'NutritionEducat
ion', 'Warmth', 'Care_Hunger', 'SocialSciences', 'PerformingArts', 'Characte
rEducation', 'TeamSports', 'Other', 'College_CareerPrep', 'Music', 'History_
Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL', 'Gym_Fitness',
'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
```

In [24]:

```
# Vectorizing "teacher_prefix"
prefix = list(set(X_train['teacher_prefix'].values))

vectorizer_teacher = CountVectorizer(vocabulary=prefix, lowercase=False, binary=True)
vectorizer_teacher.fit(X_train['teacher_prefix'].values)

X_train_prefix_one_hot = vectorizer_teacher.transform(X_train['teacher_prefix'])
X_cv_prefix_one_hot = vectorizer_teacher.transform(X_cv['teacher_prefix'])
X_test_prefix_one_hot = vectorizer_teacher.transform(X_test['teacher_prefix'])

print("After verctorizing")
print(X_train_prefix_one_hot.shape, y_train.shape)
print(X_cv_prefix_one_hot.shape, y_cv.shape)
print(X_test_prefix_one_hot.shape, y_test.shape)

print(vectorizer_teacher.get_feature_names())
```

```
After verctorizing
(49041, 5) (49041,)
(24155, 5) (24155,)
(36052, 5) (36052,)
['Mr.', 'Teacher', 'Mrs.', 'Ms.', 'Dr.']
```

In [25]:

```
# Vectorizing "school_state"
from collections import Counter
my_counter = Counter()
for word in X_train['school_state'].values:
    my_counter.update(word.split())

state_dict = dict(my_counter)
sorted_state_dict = dict(sorted(state_dict.items(), key=lambda kv: kv[1]))

vectorizer_state = CountVectorizer(vocabulary=list(sorted_state_dict.keys()), lowercase=Fal
vectorizer_state.fit(X_train['school_state'].values)

X_train_state_one_hot = vectorizer_state.transform(X_train['school_state'].values)
X_cv_state_one_hot = vectorizer_state.transform(X_cv['school_state'].values)
X_test_state_one_hot = vectorizer_state.transform(X_test['school_state'].values)

print("After verctorizing")
print(X_train_state_one_hot.shape, y_train.shape)
print(X_cv_state_one_hot.shape, y_cv.shape)
print(X_test_state_one_hot.shape, y_test.shape)

print(vectorizer_state.get_feature_names())
```

```
After verctorizing
(49041, 51) (49041,)
(24155, 51) (24155,)
(36052, 51) (36052,)
['VT', 'WY', 'ND', 'MT', 'SD', 'NE', 'RI', 'AK', 'NH', 'DE', 'WV', 'ME', 'H
I', 'DC', 'NM', 'IA', 'KS', 'ID', 'AR', 'CO', 'OR', 'MN', 'KY', 'MS', 'NV',
'TN', 'MD', 'CT', 'UT', 'AL', 'WI', 'VA', 'AZ', 'OK', 'NJ', 'WA', 'MA', 'O
H', 'LA', 'MO', 'IN', 'PA', 'MI', 'GA', 'SC', 'IL', 'NC', 'FL', 'NY', 'TX',
'CA']
```

In [26]:

```
# Vectorizing "project_grade_category"
prefix = list(set(X_train["project_grade_category"].values))

vectorizer_grade = CountVectorizer(vocabulary=prefix, lowercase=False, binary=True)
vectorizer_grade.fit(X_train['project_grade_category'])

X_train_grade_one_hot = vectorizer_grade.transform(X_train['project_grade_category'])
X_cv_grade_one_hot = vectorizer_grade.transform(X_cv['project_grade_category'])
X_test_grade_one_hot = vectorizer_grade.transform(X_test['project_grade_category'])

print("After verctorizing")
print(X_train_grade_one_hot.shape, y_train.shape)
print(X_cv_grade_one_hot.shape, y_cv.shape)
print(X_test_grade_one_hot.shape, y_test.shape)

print(vectorizer_grade.get_feature_names())
```

```
After verctorizing
(49041, 4) (49041,)
(24155, 4) (24155,)
(36052, 4) (36052,)
['Grades 9-12', 'Grades 6-8', 'Grades 3-5', 'Grades PreK-2']
```

Normalizing Numerical Features

In [27]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Normalizer.html
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1)

number_projects_train = normalizer.transform(X_train['teacher_number_of_previously_posted_p
number_projects_cv = normalizer.transform(X_cv['teacher_number_of_previously_posted_project
number_projects_test = normalizer.transform(X_test['teacher_number_of_previously_posted_pro

print("After vectorizations")
print(number_projects_train.shape, y_train.shape)
print(number_projects_cv.shape, y_cv.shape)
print(number_projects_test.shape, y_test.shape)
```

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

In [28]:

```
number_projects_train
```

Out[28]:

```
array([[0.00045291, 0.        , 0.00120775, ..., 0.00090581, 0.00830329,
        0.00090581]])
```

In [29]:

```
number_projects_train = np.reshape(number_projects_train, (-1, 1))
number_projects_cv = np.reshape(number_projects_cv, (-1, 1))
number_projects_test = np.reshape(number_projects_test, (-1, 1))
```

In [30]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Normalizer.html
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

normalizer.fit(X_train['price'].values.reshape(1,-1))

price_train = normalizer.transform(X_train['price'].values.reshape(1,-1))
price_cv = normalizer.transform(X_cv['price'].values.reshape(1,-1))
price_test = normalizer.transform(X_test['price'].values.reshape(1,-1))

print("After vectorizations")
print(price_train.shape, y_train.shape)
print(price_cv.shape, y_cv.shape)
print(price_test.shape, y_test.shape)
```

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

In [31]:

```
price_train
```

Out[31]:

```
array([[0.00380149, 0.00340257, 0.00207176, ..., 0.00068017, 0.00175507,
        0.00373375]])
```

In [32]:

```
price_train=np.reshape(price_train, (-1, 1))
price_cv=np.reshape(price_cv, (-1, 1))
price_test=np.reshape(price_test, (-1, 1))
```

In [33]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Normalizer.html
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

normalizer.fit(X_train['quantity'].values.reshape(1,-1))

quantity_train = normalizer.transform(X_train['quantity'].values.reshape(1,-1))
quantity_cv = normalizer.transform(X_cv['quantity'].values.reshape(1,-1))
quantity_test = normalizer.transform(X_test['quantity'].values.reshape(1,-1))

print("After vectorizations")
print(quantity_train.shape, y_train.shape)
print(quantity_cv.shape, y_cv.shape)
print(quantity_test.shape, y_test.shape)
```

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

In [34]:

```
quantity_train
```

Out[34]:

```
array([[0.0001418 , 0.00269418, 0.00099259, ..., 0.01091852, 0.0001418 ,
        0.0002836 ]])
```

In [35]:

```
quantity_train =np.reshape(quantity_train, (-1, 1))
quantity_cv =np.reshape(quantity_cv, (-1, 1))
quantity_test =np.reshape(quantity_test, (-1, 1))
```

Vectorizing Text Data

BoW on Project Essay | Title

In [36]:

```
# We are considering only the words which appeared in at least 10 documents(rows or project
vectorizer_bow_essay = CountVectorizer(min_df=10, max_features = 6000)
vectorizer_bow_essay.fit(X_train['clean_essays'].values)

X_train_essay_bow = vectorizer_bow_essay.transform(X_train['clean_essays'].values)
X_cv_essay_bow = vectorizer_bow_essay.transform(X_cv['clean_essays'].values)
X_test_essay_bow = vectorizer_bow_essay.transform(X_test['clean_essays'].values)

print("After vectorizing")
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)
```

```
After vectorizing
(49041, 6000) (49041,)
(24155, 6000) (24155,)
(36052, 6000) (36052,)
```

In [37]:

```
vectorizer_bow_title = CountVectorizer(min_df=10)
vectorizer_bow_title.fit(X_train['clean_titles'].values)

X_train_titles_bow = vectorizer_bow_title.transform(X_train['clean_titles'].values)
X_cv_titles_bow = vectorizer_bow_title.transform(X_cv['clean_titles'].values)
X_test_titles_bow = vectorizer_bow_title.transform(X_test['clean_titles'].values)

print("After vectorizing")
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)
```

```
After vectorizing
(49041, 2007) (49041,)
(24155, 2007) (24155,)
(36052, 2007) (36052,)
```

TFIDF on Project Essay | Title

In [38]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer_tfidf_essay = TfidfVectorizer(min_df=10, max_features = 6000)
vectorizer_tfidf_essay.fit(X_train['clean_essays'].values)

X_train_essay_tfidf = vectorizer_tfidf_essay.transform(X_train['clean_essays'])
X_cv_essay_tfidf = vectorizer_tfidf_essay.transform(X_cv['clean_essays'])
X_test_essay_tfidf = vectorizer_tfidf_essay.transform(X_test['clean_essays'])

print("After vectorizing")
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)
```

```
After vectorizing
(49041, 6000) (49041,)
(24155, 6000) (24155,)
(36052, 6000) (36052,)
```

In [39]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer_tfidf_title = TfidfVectorizer(min_df=10)
vectorizer_tfidf_title.fit(X_train['clean_titles'].values)

X_train_title_tfidf = vectorizer_tfidf_title.transform(X_train['clean_titles'])
X_cv_title_tfidf = vectorizer_tfidf_title.transform(X_cv['clean_titles'])
X_test_title_tfidf = vectorizer_tfidf_title.transform(X_test['clean_titles'])

print("After vectorizing")
print(X_train_title_tfidf.shape, y_train.shape)
print(X_cv_title_tfidf.shape, y_cv.shape)
print(X_test_title_tfidf.shape, y_test.shape)
```

```
After vectorizing
(49041, 2007) (49041,)
(24155, 2007) (24155,)
(36052, 2007) (36052,)
```

Assignment 4: Naive Bayes

1. **Apply Multinomial NaiveBayes on these feature sets**
   - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_eassay (BOW)
   - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)
2. **The hyper paramter tuning(find best Alpha)**
   - 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
   - Consider a wide range of alpha values for hyperparameter tuning, start as low as 0.00001
   - Find the best hyper paramter using k-fold cross validation or simple cross validation data

- Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning

3. **Feature importance**

- Find the top 10 features of positive class and top 10 features of negative class for both feature sets Set 1 and Set 2 using 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

4. **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. Here on X-axis you will have alpha values, since they have a wide range, just to represent those alpha values on the graph, apply log function on those alpha values.

- 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/) with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.

(https://seaborn.pydata.org/generated/seaborn.heatmap.html)

5. **Conclusion**

- You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link (http://zetcode.com/python/prettytable/)

Set 1: categorical, numerical features + project_title(BOW) + preprocessed_essay (BOW)

In [40]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_train_bow = hstack((X_train_categories_one_hot, X_train_sub_categories_one_hot, X_train_e
X_cv_bow = hstack((X_cv_categories_one_hot, X_cv_sub_categories_one_hot, X_cv_essay_bow, qu
X_test_bow = hstack((X_test_categories_one_hot, X_test_sub_categories_one_hot, X_test_essay

print('Final matrix')
print(X_train_bow.shape, y_train.shape)
print(X_cv_bow.shape, y_cv.shape)
print(X_test_bow.shape, y_test.shape)
```

```
Final matrix
(49041, 8109) (49041,)
(24155, 8109) (24155,)
(36052, 8109) (36052,)
```

In [41]:

```
def batch_predict(clf, data):
    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000  #Iter untill 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
    y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

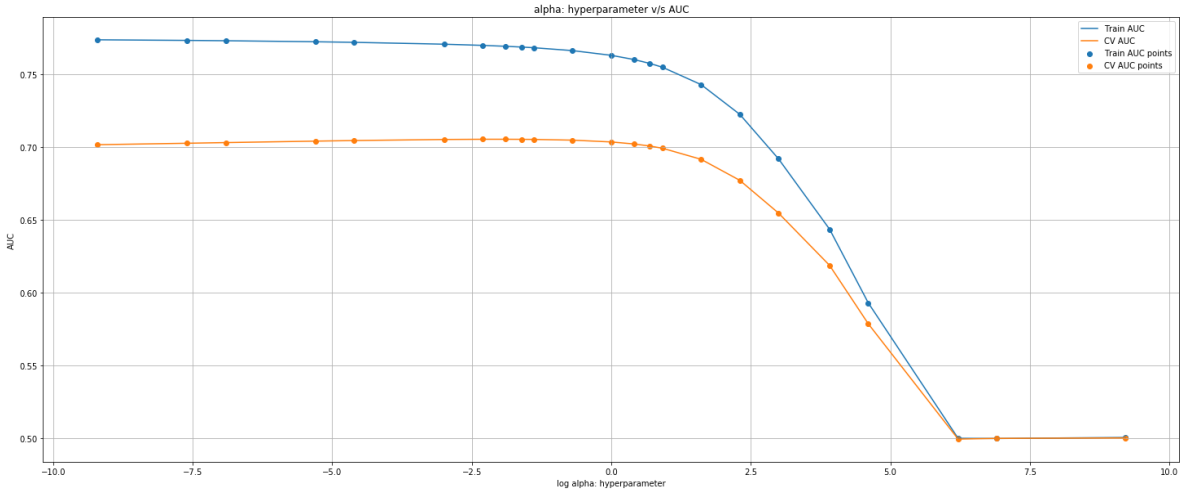
In [42]:

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

train_auc = []
cv_auc = []
log_alphas = []

alphas = [0.0001, 0.0005, 0.001,0.005, 0.01, 0.05, 0.1, 0.15, 0.2, 0.25, 0.5, 1.0, 1.5, 2.0

for i in tqdm(alphas):
    nb_bow = MultinomialNB(alpha = i, class_prior=[0.5,0.5])
    nb_bow.fit(X_train_bow, y_train)

    y_train_pred = batch_predict(nb_bow, X_train_bow )
    y_cv_pred = batch_predict(nb_bow, X_cv_bow)


    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

for a in tqdm(alphas):
    b = math.log(a)
    log_alphas.append(b)
```

```
100%|████████████████████████████████████████████████████████████████████████████████
| 23/23 [00:10<00:00,  2.00it/s]
100%|████████████████████████████████████████████████████████████████████████████████
| 23/23 [00:00<00:00, 5752.82it/s]
```

In [43]:

```
plt.figure(figsize=(25,10))
plt.plot(log_alphas, train_auc, label='Train AUC')
plt.plot(log_alphas, cv_auc, label='CV AUC')

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()
plt.show()
```



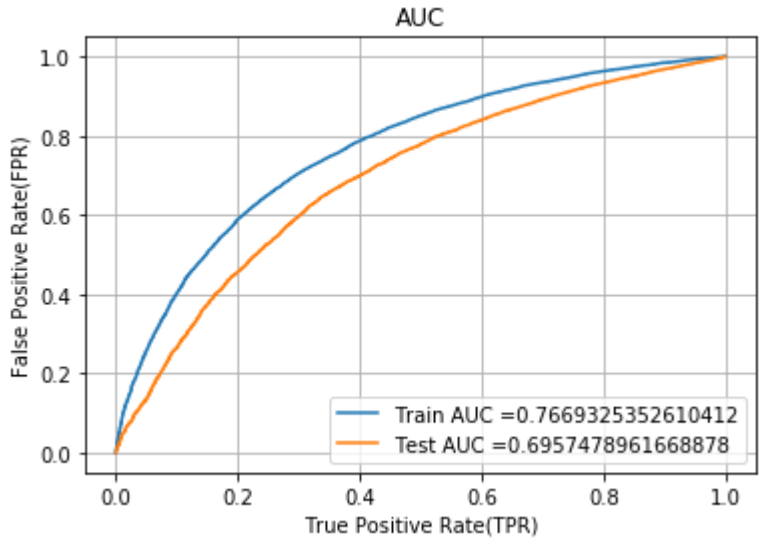
In [44]:

```
#Best Alpha is approx 0.4
best_alpha = 0.4
```

Plotting Error Plot

In [45]:

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

nb_bow = MultinomialNB(alpha = best_alpha,class_prior=[0.5, 0.5])

nb_bow.fit(X_train_bow, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the p
# not the predicted outputs

y_train_pred = batch_predict(nb_bow, X_train_bow)
y_test_pred = batch_predict(nb_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("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



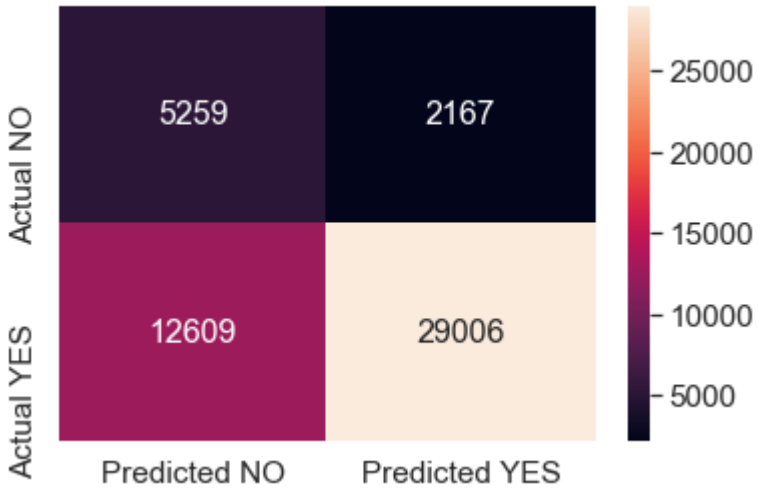
Confusion Matrix

In [46]:

```
def get_confusion_matrix(clf,X_te,y_test):
    y_pred = clf.predict(X_te)
    df_cm = pd.DataFrame(confusion_matrix(y_test, y_pred), range(2),range(2))
    df_cm.columns = ['Predicted NO','Predicted YES']
    df_cm = df_cm.rename({0: 'Actual NO', 1: 'Actual YES'})
    sns.set(font_scale=1.4)#for label size
    sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
```

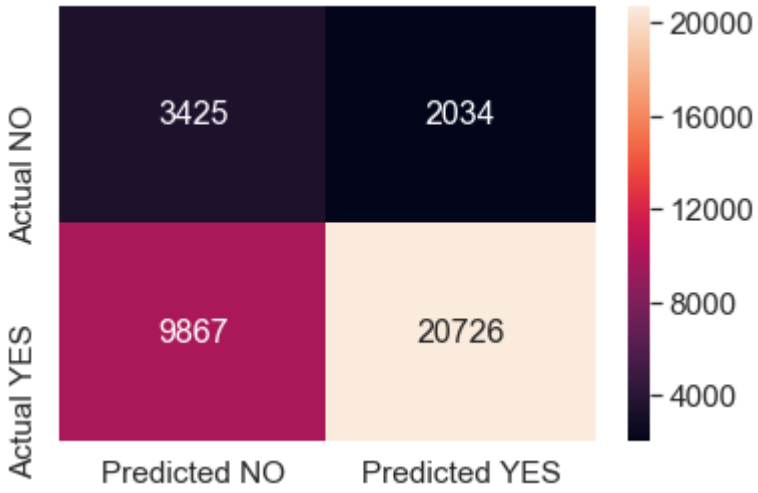
In [47]:

```
get_confusion_matrix(nb_bow,X_train_bow,y_train)
```



In [48]:

```
get_confusion_matrix(nb_bow,X_test_bow,y_test)
```



Evaluating Model performance

In [49]:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score

y_pred_new = nb_bow.predict(X_test_bow)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred_new)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred_new)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred_new)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred_new)))
```

```
Accuracy on test set: 66.989%
Precision on test set: 0.911
Recall on test set: 0.677
F1-Score on test set: 0.777
```

Set 2: categorical, numerical features + project_title(TFIDF) + preprocessed_essay (TFIDF)

In [50]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_train_tfidf = hstack((X_train_categories_one_hot, X_train_sub_categories_one_hot, X_train
X_cv_tfidf = hstack((X_cv_categories_one_hot, X_cv_sub_categories_one_hot, X_cv_essay_tfidf
X_test_tfidf = hstack((X_test_categories_one_hot, X_test_sub_categories_one_hot, X_test_ess

print('Final matrix')
print(X_train_bow.shape, y_train.shape)

print(X_cv_bow.shape, y_cv.shape)
print(X_test_bow.shape, y_test.shape)
```

```
Final matrix
(49041, 8109) (49041,)
(24155, 8109) (24155,)
(36052, 8109) (36052,)
```

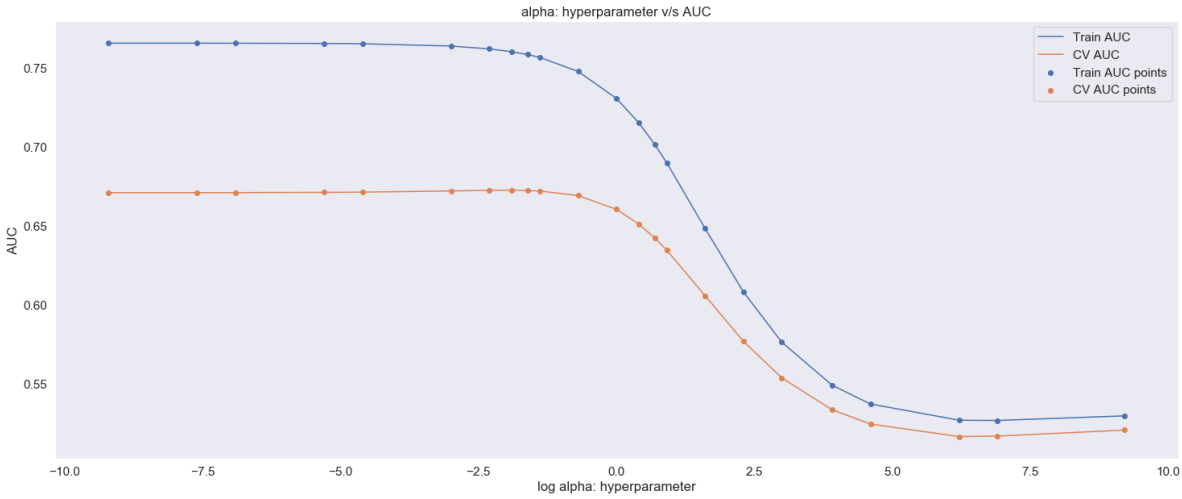
In [51]:

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

train_auc = []
cv_auc = []
log_alphas = []

alphas = [0.0001, 0.0005, 0.001,0.005, 0.01, 0.05, 0.1, 0.15, 0.2, 0.25, 0.5, 1.0, 1.5, 2.0

for i in tqdm(alphas):
    nb_tfidf = MultinomialNB(alpha = i, class_prior=[0.5, 0.5])
    nb_tfidf.fit(X_train_tfidf, y_train)

    y_train_pred = batch_predict(nb_tfidf, X_train_tfidf)
    y_cv_pred = batch_predict(nb_tfidf, X_cv_tfidf)


    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
    # not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

for a in tqdm(alphas):
    b = math.log(a)
    log_alphas.append(b)
```

```
100%|████████████████████████████████████████████████████████████████
          | 23/23 [00:10<00:00,  2.29it/s]
100%|████████████████████████████████████████████████████████████████
                  | 23/23 [00:00<?, ?it/s]
```

In [52]:

```
plt.figure(figsize=(25,10))
plt.plot(log_alphas, train_auc, label='Train AUC')
plt.plot(log_alphas, cv_auc, label='CV AUC')

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()
plt.show()
```



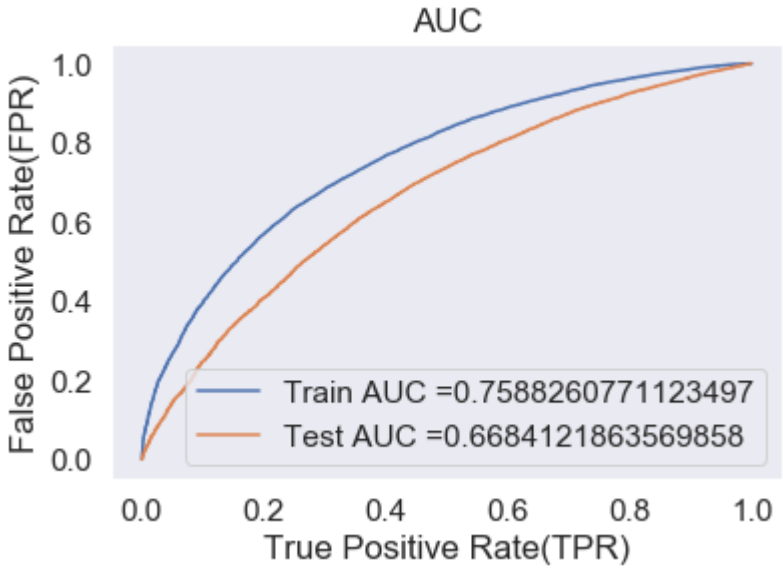
In [53]:

```
#Best Alpha is approx 0.6
best_alpha_tfidf = 0.1
```

Plotting the Error Plot

In [54]:

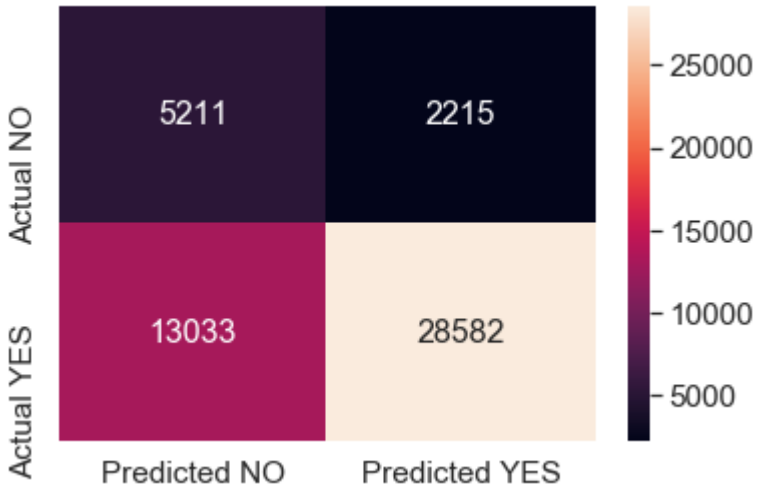
```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.
from sklearn.metrics import roc_curve, auc

nb_tfidf = MultinomialNB(alpha = best_alpha_tfidf,class_prior=[0.5, 0.5])

nb_tfidf.fit(X_train_tfidf, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the p
# not the predicted outputs

y_train_pred = batch_predict(nb_bow, X_train_tfidf)
y_test_pred = batch_predict(nb_bow, 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("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid()
plt.show()
```



Confusion Matrix

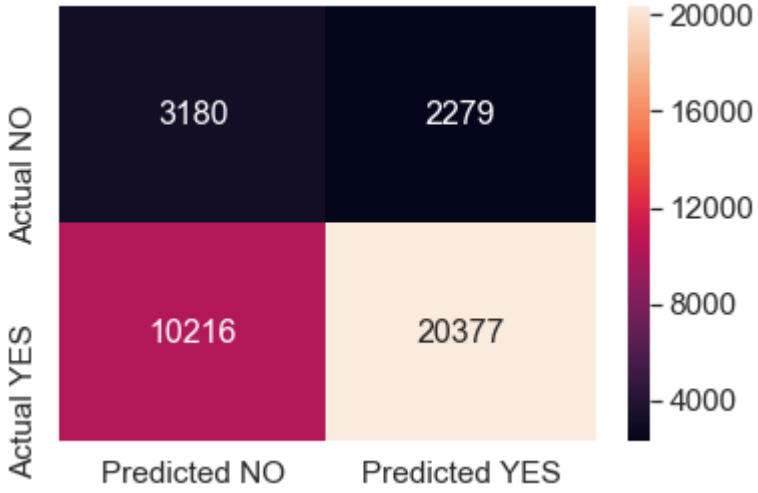
In [55]:

```
get_confusion_matrix(nb_tfidf,X_train_tfidf,y_train)
```



In [56]:

```
get_confusion_matrix(nb_tfidf,X_test_tfidf,y_test)
```



Evaluating Model Performance

In [57]:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score

y_pred_new = nb_bow.predict(X_test_tfidf)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred_new)*100))
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred_new)))
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred_new)))
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred_new)))
```

```
Accuracy on test set: 59.808%
Precision on test set: 0.907
Recall on test set: 0.587
F1-Score on test set: 0.712
```

Selecting the best 10 features for negative class on Set1

In [58]:

```
print(nb_bow)
```

```
MultinomialNB(alpha=0.4, class_prior=[0.5, 0.5], fit_prior=True)
```

In [59]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
bow_fnprob = {}
#Total number of features: 8109
for fprob in range(8109) :
    bow_fnprob[fprob] = nb_bow.feature_log_prob_[0,fprob]       #NB classifier alpha: 0.6
```

In [60]:

```
len(bow_fnprob.values()) #Features probability
```

Out[60]:

```
8109
```

In [61]:

```
#Getting features name from the vectorizers and appending them in the list
bow_features_list = []
```

In [62]:

```
#Appending features in same order as prob value of features stored
for x in vectorizer_sbj.get_feature_names():
    bow_features_list.append(x)

for x in vectorizer_sub_sbj.get_feature_names():
    bow_features_list.append(x)

for x in vectorizer_bow_essay.get_feature_names():
    bow_features_list.append(x)

bow_features_list.append("quantity")

for x in vectorizer_state.get_feature_names():
    bow_features_list.append(x)

for x in vectorizer_teacher.get_feature_names():
    bow_features_list.append(x)

for x in vectorizer_grade.get_feature_names():
    bow_features_list.append(x)

for x in vectorizer_bow_title.get_feature_names():
    bow_features_list.append(x)

bow_features_list.append("price")

bow_features_list.append("teacher_number_of_previously_posted_projects")
```

In [63]:

```
len(bow_features_list)
```

Out[63]:

```
8109
```

In [64]:

```
n_feature_prob_bow  = pd.DataFrame({'feature_name':bow_features_list, 'feature_probabilty':
```

In [65]:

```
n_feature_prob_bow.head(10)
```

Out[65]:

	feature_name	feature_probabilty
0	Warmth	-9.761815
1	Care_Hunger	-9.761815
2	History_Civics	-7.881207
3	Music_Arts	-7.229280
4	AppliedLearning	-6.970885
5	SpecialNeeds	-6.878047
6	Health_Sports	-6.932092
7	Math_Science	-5.810370
8	Literacy_Language	-5.760391
9	Economics	-10.577110

In [66]:

```
#Sorting in descending order with
n_feature_prob_bow = n_feature_prob_bow.sort_values(by = ['feature_probabilty'], ascending
```

In [67]:

```
n_feature_prob_bow.head(10)
```

Out[67]:

	feature_name	feature_probabilty
5270	students	-2.974823
4788	school	-4.061675
3210	learning	-4.374670
1038	classroom	-4.568916
3753	not	-4.729421
3206	learn	-4.731810
2675	help	-4.766582
3661	nannan	-4.947056
3413	many	-4.958137
3686	need	-5.102658

Selecting the best 10 features for positive class of SET1

In [69]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
bow_fpprob = {}
#Total number of features: 8109
for fprob in range(8109) :
    bow_fpprob[fprob] = nb_bow.feature_log_prob_[1,fprob]       #NB classifier alpha: 0.6
```

In [70]:

```
len(bow_fpprob.values()) #Features probability
```

Out[70]:

```
8109
```

In [71]:

```
p_feature_prob_bow  = pd.DataFrame({'feature_name':bow_features_list, 'feature_probabilty':
```

In [72]:

```
p_feature_prob_bow.head(10)
```

Out[72]:

	feature_name	feature_probabilty
0	Warmth	-9.309461
1	Care_Hunger	-9.309461
2	History_Civics	-7.879245
3	Music_Arts	-7.325555
4	AppliedLearning	-7.169285
5	SpecialNeeds	-7.036674
6	Health_Sports	-6.999703
7	Math_Science	-5.930801
8	Literacy_Language	-5.663317
9	Economics	-11.084488

In [73]:

```
#Sorting in descending order with
p_feature_prob_bow = p_feature_prob_bow.sort_values(by = ['feature_probabilty'], ascending
```

In [74]:

```
p_feature_prob_bow.head(10)
```

Out[74]:

	feature_name	feature_probabilty
5270	students	-2.962101
4788	school	-4.109174
3210	learning	-4.469867
1038	classroom	-4.492272
3753	not	-4.766729
3206	learn	-4.812814
2675	help	-4.840807
3413	many	-4.989915
3661	nannan	-4.998056
5966	work	-5.104864

Selecting 10 best features for "Project not Approved|Negative Class " of SET2 (TFIDF)

In [75]:

```
print(nb_tfidf)
```

```
MultinomialNB(alpha=0.1, class_prior=[0.5, 0.5], fit_prior=True)
```

In [77]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
tfidf_fnprob = {}
#Total number of features: 8109
for fprob in range(8109) :
    tfidf_fnprob[fprob] = nb_tfidf.feature_log_prob_[0,fprob]       #NB classifier alpha: 0
```

In [78]:

```
len(tfidf_fnprob.values()) #Features probability
```

Out[78]:

```
8109
```

In [79]:

```
#Getting features name from the vectorizers and appending them in the list
tfidf_features_list = []
```

In [80]:

```
#Appending features in same order as prob value of features stored
for x in vectorizer_sbj.get_feature_names():
    tfidf_features_list.append(x)

for x in vectorizer_sub_sbj.get_feature_names():
    tfidf_features_list.append(x)

for x in vectorizer_tfidf_essay.get_feature_names():
    tfidf_features_list.append(x)

tfidf_features_list.append("quantity")

for x in vectorizer_state.get_feature_names():
    tfidf_features_list.append(x)

for x in vectorizer_teacher.get_feature_names():
    tfidf_features_list.append(x)

for x in vectorizer_grade.get_feature_names():
    tfidf_features_list.append(x)

for x in vectorizer_tfidf_title.get_feature_names():
    tfidf_features_list.append(x)

tfidf_features_list.append("price")

tfidf_features_list.append("teacher_number_of_previously_posted_projects")
```

In [81]:

```
n_feature_prob_tfidf  = pd.DataFrame({'feature_name':tfidf_features_list, 'feature_probabil
```

In [82]:

```
#Sorting in descending order with
n_feature_prob_tfidf = n_feature_prob_tfidf.sort_values(by = ['feature_probabilty'], ascend
```

In [83]:

```
n_feature_prob_tfidf.head(10)
```

Out[83]:

	feature_name	feature_probabilty
8	Literacy_Language	-3.514128
7	Math_Science	-3.564112
37	Mathematics	-3.996037
38	Literacy	-4.015313
36	Literature_Writing	-4.314085
5	SpecialNeeds	-4.631980
35	SpecialNeeds	-4.631980
5270	students	-4.647143
6090	CA	-4.680912
6	Health_Sports	-4.686042

Selecting the best 10 features for "Approved class|Positive class" of SET2(TFIDF)

In [84]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
tfidf_fpprob = {}
#Total number of features: 8109
for fprob in range(8109) :
    tfidf_fpprob[fprob] = nb_tfidf.feature_log_prob_[1,fprob]       #NB classifier alpha: 0
```

In [85]:

```
len(tfidf_fpprob.values()) #Features probability
```

Out[85]:

```
8109
```

In [86]:

```
p_feature_prob_tfidf  = pd.DataFrame({'feature_name':tfidf_features_list, 'feature_probabil
```

In [87]:

```
#Sorting in descending order with
p_feature_prob_tfidf = p_feature_prob_tfidf.sort_values(by = ['feature_probabilty'], ascend
```

In [88]:

```
p_feature_prob_tfidf.head(10)
```

Out[88]:

	feature_name	feature_probabilty
8	Literacy_Language	-3.368392
7	Math_Science	-3.635880
38	Literacy	-3.796163
37	Mathematics	-4.014197
36	Literature_Writing	-4.233450
6090	CA	-4.623347
5270	students	-4.642086
6	Health_Sports	-4.704819
5	SpecialNeeds	-4.741793
35	SpecialNeeds	-4.741793

Conclusion

In [89]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vectorizer", "Hyperparameter", "Train AUC","Test AUC","F1 Score","Accurac

x.add_row(["BoW",0.4, 0.76, 0.71,0.83,"73.41%"])
x.add_row(["TFIDF",0.1, 0.75, 0.66,0.90,"83.62%"])

print(x)
```

```
+------------+----------------+-----------+----------+----------+-----------
-----------+
| Vectorizer | Hyperparameter | Train AUC | Test AUC | F1 Score | Accuracy o
n test set |
+------------+----------------+-----------+----------+----------+-----------
-----------+
|    BoW     |      0.4       |    0.76   |   0.71   |   0.83   |        73.
41%        |
|   TFIDF    |      0.1       |    0.75   |   0.66   |   0.9    |        83.
62%        |
+------------+----------------+-----------+----------+----------+-----------
-----------+
```

1. Naive Bayes is relatively much faster than KNN..
2. The TFIDF(alpha 0.1) model has performed better on test data than BoW (alpha 0.6).