Assignment 4: Apply Naive bayes on Donors Choose dataset

This exercise is to apply Naive Bayes on Donors Choose dataset and predict approval of a new project proposal.

Relevant Information : The dataset is divided into two files -

1. train.csv file which contains information regarding projects, schools and teachers who submitted the projects.
2. resources.csv which provides information about the resources required for each project.

OBJECTIVE : The goal is to predict whether or not a DonorsChoose.org project proposal submitted by a teacher will be approved, using the text of project descriptions as well as additional metadata about the project, teacher, and school using Naive Bayes classification algorithm.

In [2]:

```python
# Importing the required libraries
# Warning reference : https://stackoverflow.com/questions/41658568/chunkize-warning-while-installing-gensim

%matplotlib inline
import warnings
warnings.filterwarnings(action='ignore', category = UserWarning , module = 'gensim')

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

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

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

1. Reading the data

In [3]:

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

print("\nNumber of data points in train data", project_data.shape)
print('-'*120)
print("The attributes of data :", project_data.columns.values)
```

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

In [4]:
```
project_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 109248 entries, 0 to 109247
Data columns (total 17 columns):
Unnamed: 0                                      109248 non-null int64
id                                              109248 non-null object
teacher_id                                      109248 non-null object
teacher_prefix                                  109245 non-null object
school_state                                    109248 non-null object
project_submitted_datetime                      109248 non-null object
project_grade_category                          109248 non-null object
project_subject_categories                      109248 non-null object
project_subject_subcategories                   109248 non-null object
project_title                                   109248 non-null object
project_essay_1                                 109248 non-null object
project_essay_2                                 109248 non-null object
project_essay_3                                 3758 non-null object
project_essay_4                                 3758 non-null object
project_resource_summary                        109248 non-null object
teacher_number_of_previously_posted_projects    109248 non-null int64
project_is_approved                             109248 non-null int64
dtypes: int64(3), object(14)
memory usage: 14.2+ MB
```

NOTE:

1. We have a total of 109248 datapoints and 17 columns.
2. Now we have to sort the data according to date and time so as to have a better prediction on the future data (Test data).
3. As we can see there are null points for project_essay_3 and project_essay_4. Only 3758 points are not null.
4. In teacher_prefix there are 109245 points which means 3 points are null.

Sorting according to date

In [5]:
```
# 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]

print(cols)
```

```
['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state', 'Date', '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_approve
d']
```

In [6]:
```
# Dropping the Unnamed column

project_data.drop('Unnamed: 0', axis=1, inplace=True)
```

Adding the price and quantity column from resource_data to the project_data

In [7]:
```
# https://stackoverflow.com/questions/22407798/how-to-reset-a-dataframes-indexes-for-all-groups-in-one-step
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()

# join two dataframes in python:
project_data = pd.merge(project_data, price_data, on='id', how='left')

# Deleting the resource data variable
del resource_data
```

NOTE:

1. As we can see that the price and the quantity column has been added to the project_data
2. This is where the preprocessing will start.
3. we are going to consider

```
- school_state : categorical data
- clean_categories : categorical data
- clean_subcategories : categorical data
- project_grade_category : categorical data
- teacher_prefix : categorical data

- project_title : text data
- essay : text data

- quantity : numerical
- teacher_number_of_previously_posted_projects : numerical
- price : numerical
```

2. Preprocessing Data

project_subject_categories

In [8]:

```
catogories = list(project_data['project_subject_categories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47301924/4084039

# 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", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Science"=> "Math","&", "Sci
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math & Science"=>"Math&Science"
        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]))
```

project_subject_subcategories

In [9]:

```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflow.com/a/47301924/4084039

# 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", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Science"=> "Math","&", "Sci
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math & Science"=>"Math&Science"
        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]))
```

Text Preprocessing

Essay

In [10]:
```
# Combining all the essay
# 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 [11]:
```
project_data['essay'].describe()
```

Out[11]:
```
count                                                109248
unique                                               108986
top       Hello, I teach a great group of students who a...
freq                                                      5
Name: essay, dtype: object
```

In [12]:
```
# Making the decontracted function

# 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', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further',
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more',
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't
            'won', "won't", 'wouldn', "wouldn't"]

# Combining all the above stundents
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 = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
    preprocessed_essays.append(sent.lower().strip())
```

```
100%|██████████████████████████████████████████████████| 109248/109248 [01:23<00:00, 1313.74it/s]
```

`project_title`

In [13]:
```
project_data['project_title'].describe()
```

Out[13]:
```
count               109248
unique              100851
top       Flexible Seating
freq                   234
Name: project_title, dtype: object
```

In [14]:
```
# Project_title

# Combining all the above statemennts
from tqdm import tqdm
preprocessed_titles = []
# tqdm is for printing the status bar
for sentence in tqdm(project_data['project_title'].values):
    sent = decontracted(sentence)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    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 [00:03<00:00, 29524.86it/s]
```

Replacing the columns with new cleaned columns - Project_title and Essay

In [15]:
```
# Adding processed essay columns in place of previous essays columns and dropping the previous columns

## ESSAY

project_data['clean_essays'] = 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)
project_data.drop(['essay'], axis=1, inplace=True)
```

In [16]:
```
## Project_title

# Adding processed project_title columns in place of previous project_title column and dropping the previous column

project_data['clean_titles'] = preprocessed_titles

project_data.drop(['project_title'], axis=1, inplace=True)
```

Dropping the nan rows present in teacher_prefix

In [17]:
```
# Dropping NAN row
# https://stackoverflow.com/questions/46091924/python-how-to-drop-a-row-whose-particular-column-is-empty-nan

project_data.dropna(axis = 0, inplace = True, subset = ['teacher_prefix'])
```

In [18]:
```
project_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 109245 entries, 0 to 109247
Data columns (total 15 columns):
id                                              109245 non-null object
teacher_id                                      109245 non-null object
teacher_prefix                                  109245 non-null object
school_state                                    109245 non-null object
Date                                            109245 non-null datetime64[ns]
project_grade_category                          109245 non-null object
project_resource_summary                        109245 non-null object
teacher_number_of_previously_posted_projects    109245 non-null int64
project_is_approved                             109245 non-null int64
price                                           109245 non-null float64
quantity                                        109245 non-null int64
clean_categories                                109245 non-null object
clean_subcategories                             109245 non-null object
clean_essays                                    109245 non-null object
clean_titles                                    109245 non-null object
dtypes: datetime64[ns](1), float64(1), int64(3), object(10)
memory usage: 13.3+ MB
```

NOTE:

```
- Till now we have preprocessed the data.
- Now we have to split the data and vectorize the data for BOW, TF-IDF
```

**************************************ASSIGNMENT**************************************

3. Splitting data into Train and cross validation(or test): Stratified Sampling

In [19]:

```
# Creating label and feature data frame : Label- y, Features- X

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

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

```
(109245,)
(109245, 14)
```

In [20]:

```
## train test cross-validation split
# Referance : https://stackoverflow.com/questions/34842405/parameter-stratify-from-method-train-test-split-scikit-learn

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=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)
```

NOTE:

```
- This `stratify` parameter makes a split so that the proportion of values in the sample produced will be the same as the proportion of values provided to parameter stratify.

- For example, if variable y is a binary categorical variable with values 0 and 1 and there are 25% of zeros and 75% of ones, stratify=y will make sure that your random split has 25% of 0's and 75% of 1's.
```

In [21]:

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

```
(49039, 14) (49039,)
(24155, 14) (24155,)
(36051, 14) (36051,)
```

NOTE:

1. We will now use the train data for training our model, cv data to validate the model and perform testing on the test data

4. Make Data Model Ready: encoding numerical, categorical features

Vectorizing Categorical Features

1. clean_categories

In [22]:

```
# We use count vectorizer to convert the values into one hot encoded features
from sklearn.feature_extraction.text import CountVectorizer

vectorizer_clean_category = CountVectorizer()

# We will fit the train data only
vectorizer_clean_category.fit(X_train['clean_categories'].values)


# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_category = vectorizer_clean_category.transform(X_train['clean_categories'].values)
X_cv_clean_category = vectorizer_clean_category.transform(X_cv['clean_categories'].values)
X_test_clean_category = vectorizer_clean_category.transform(X_test['clean_categories'].values)

print("Clean categories are vectorized\n")
print(X_train_clean_category.shape, y_train.shape)
print(X_cv_clean_category.shape, y_cv.shape)
print(X_test_clean_category.shape, y_test.shape)
print(vectorizer_clean_category.get_feature_names())
```

```
Clean categories are vectorized

(49039, 9) (49039,)
(24155, 9) (24155,)
(36051, 9) (36051,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language', 'math_science', 'music_art
s', 'specialneeds', 'warmth']
```

2. clean_subcategories

In [23]:

```
vectorizer_clean_subcategories = CountVectorizer()

# We will fit the train data only
vectorizer_clean_subcategories.fit(X_train['clean_subcategories'].values)


# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_subcategories = vectorizer_clean_subcategories.transform(X_train['clean_subcategories'].values)
X_cv_clean_subcategories = vectorizer_clean_subcategories.transform(X_cv['clean_subcategories'].values)
X_test_clean_subcategories = vectorizer_clean_subcategories.transform(X_test['clean_subcategories'].values)

print("Clean_subcategories are vectorized\n")
print(X_train_clean_subcategories.shape, y_train.shape)
print(X_cv_clean_subcategories.shape, y_cv.shape)
print(X_test_clean_subcategories.shape, y_test.shape)
print(vectorizer_clean_subcategories.get_feature_names())
```

```
Clean_subcategories are vectorized

(49039, 30) (49039,)
(24155, 30) (24155,)
(36051, 30) (36051,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government', 'college_careerprep', 'communityservice',
'earlydevelopment', 'economics', 'environmentalscience', 'esl', 'extracurricular', 'financialliteracy', 'foreignlanguag
es', 'gym_fitness', 'health_lifescience', 'health_wellness', 'history_geography', 'literacy', 'literature_writing', 'ma
thematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'socialsciences', 'specialne
eds', 'teamsports', 'visualarts', 'warmth']
```

3. teacher_prefix

In [24]:

```
vectorizer_teacher_prefix = CountVectorizer()

# We will fit the train data only
vectorizer_teacher_prefix.fit(X_train['teacher_prefix'].values)

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_prefix = vectorizer_teacher_prefix.transform(X_train['teacher_prefix'].values)
X_cv_teacher_prefix = vectorizer_teacher_prefix.transform(X_cv['teacher_prefix'].values)
X_test_teacher_prefix = vectorizer_teacher_prefix.transform(X_test['teacher_prefix'].values)

print("Teacher_prefix are vectorized\n")
print(X_train_teacher_prefix.shape, y_train.shape)
print(X_cv_teacher_prefix.shape, y_cv.shape)
print(X_test_teacher_prefix.shape, y_test.shape)
print(vectorizer_teacher_prefix.get_feature_names())
```

```
Teacher_prefix are vectorized

(49039, 5) (49039,)
(24155, 5) (24155,)
(36051, 5) (36051,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
```

4. school_state

In [25]:

```
vectorizer_school_state = CountVectorizer()

# We will fit the train data only
vectorizer_school_state.fit(X_train['school_state'].values)

# we use the fitted CountVectorizer to convert the text to vector
X_train_school_state = vectorizer_school_state.transform(X_train['school_state'].values)
X_cv_school_state = vectorizer_school_state.transform(X_cv['school_state'].values)
X_test_school_state = vectorizer_school_state.transform(X_test['school_state'].values)

print("School_state are vectorized\n")
print(X_train_school_state.shape, y_train.shape)
print(X_cv_school_state.shape, y_cv.shape)
print(X_test_school_state.shape, y_test.shape)
print(vectorizer_school_state.get_feature_names())
```

```
School_state are vectorized

(49039, 51) (49039,)
(24155, 51) (24155,)
(36051, 51) (36051,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'ks', 'ky', 'la', 'm
a', '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']
```

5. project_grade_category

In [26]:

```
#This step is to intialize a vectorizer with vocab from train data

# Creating the list of grades

grades = list(set(project_data['project_grade_category'].values))

# we use count vectorizer to convert the values into one hot encoded features
# We will fit the train data only
vectorizer_project_grade_category = CountVectorizer(vocabulary = grades, lowercase=False, binary=True)
vectorizer_project_grade_category.fit(X_train['project_grade_category'].values)


# we use the fitted CountVectorizer to convert the text to vector
X_train_project_grade = vectorizer_project_grade_category.transform(X_train['project_grade_category'].values)
X_cv_project_grade = vectorizer_project_grade_category.transform(X_cv['project_grade_category'].values)
X_test_project_grade = vectorizer_project_grade_category.transform(X_test['project_grade_category'].values)

print("Project_grade_category are vectorized\n")
print(X_train_project_grade.shape, y_train.shape)
print(X_cv_project_grade.shape, y_cv.shape)
print(X_test_project_grade.shape, y_test.shape)
print(vectorizer_project_grade_category.get_feature_names())
```

```
Project_grade_category are vectorized

(49039, 4) (49039,)
(24155, 4) (24155,)
(36051, 4) (36051,)
['Grades 9-12', 'Grades 6-8', 'Grades PreK-2', 'Grades 3-5']
```

Standardizing Numerical features

1. price

In [27]:

```
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

from sklearn.preprocessing import StandardScaler

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

price_scalar = StandardScaler(with_mean = False)

# We will fit the train data only
price_scalar.fit(X_train['price'].values.reshape(-1,1)) # finding the mean and standard deviation of this data
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")


# Now standardize the data with above mean and variance.
X_train_price = price_scalar.transform(X_train['price'].values.reshape(-1,1))
X_cv_price = price_scalar.transform(X_cv['price'].values.reshape(-1,1))
X_test_price = price_scalar.transform(X_test['price'].values.reshape(-1,1))

print("Price is standardized\n")
print(X_train_price.shape, y_train.shape)
print(X_cv_price.shape, y_cv.shape)
print(X_test_price.shape, y_test.shape)
```

```
Mean : 297.78920920084016, Standard deviation : 370.20133084834544
Price is standardized

(49039, 1) (49039,)
(24155, 1) (24155,)
(36051, 1) (36051,)
```

2. teacher_number_of_previously _posted_projects

In [28]:

```
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.

# https://stackoverflow.com/questions/29086398/sklearn-turning-off-warnings
from sklearn.exceptions import DataConversionWarning
warnings.filterwarnings(action='ignore', category=DataConversionWarning)


from sklearn.preprocessing import StandardScaler
previous_post_scalar = StandardScaler(with_mean = False)


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

# We will fit the train data only
# finding the mean and standard deviation of this data
previous_post_scalar.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
print(f"Mean : {previous_post_scalar.mean_[0]}, Standard deviation : {np.sqrt(previous_post_scalar.var_[0])}")

X_train_previous_projects = previous_post_scalar.transform(X_train['teacher_number_of_previously_posted_projects'].value
X_cv_previous_projects = previous_post_scalar.transform(X_cv['teacher_number_of_previously_posted_projects'].values.resh
X_test_previous_projects = previous_post_scalar.transform(X_test['teacher_number_of_previously_posted_projects'].values.

print("Teacher_number_of_previously_posted_projects is standardized\n")
print(X_train_previous_projects.shape, y_train.shape)
print(X_cv_previous_projects.shape, y_cv.shape)
print(X_test_previous_projects.shape, y_test.shape)
```

```
Mean : 11.120659067272987, Standard deviation : 27.73421035978225
Teacher_number_of_previously_posted_projects is standardized

(49039, 1) (49039,)
(24155, 1) (24155,)
(36051, 1) (36051,)
```

3. quantity

In [29]:
```
# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.

# https://stackoverflow.com/questions/29086398/sklearn-turning-off-warnings
from sklearn.exceptions import DataConversionWarning
warnings.filterwarnings(action='ignore', category=DataConversionWarning)

from sklearn.preprocessing import StandardScaler
quantity_scalar = StandardScaler(with_mean = False)

# We will fit the train data only
# finding the mean and standard deviation of this data
quantity_scalar.fit(X_train['quantity'].values.reshape(-1,1))
print(f"Mean : {quantity_scalar.mean_[0]}, Standard deviation : {np.sqrt(quantity_scalar.var_[0])}")


# Now standardize the data with above maen and variance.
X_train_quantity = quantity_scalar.transform(X_train['quantity'].values.reshape(-1,1))
X_cv_quantity = quantity_scalar.transform(X_cv['quantity'].values.reshape(-1,1))
X_test_quantity = quantity_scalar.transform(X_test['quantity'].values.reshape(-1,1))


print("quantity is standardized")
print(X_train_quantity.shape, y_train.shape)
print(X_cv_quantity.shape, y_cv.shape)
print(X_test_quantity.shape, y_test.shape)
```

```
Mean : 16.919880095434245, Standard deviation : 26.396039021317304
quantity is standardized
(49039, 1) (49039,)
(24155, 1) (24155,)
(36051, 1) (36051,)
```

5. Make Data Model Ready: encoding eassay, and project_title

BOW

1. clean_essay

In [30]:
```
%%time

# We are considering only the words which appeared in at least 10 documents(rows or projects).
# https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

from sklearn.feature_extraction.text import CountVectorizer
vectorizer_essay_bow = CountVectorizer(min_df=10)
vectorizer_essay_bow.fit(X_train['clean_essays'].values) # fit has to happen only on train data

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

print("Essay vectorized")
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)
print("="*120)
```

```
Essay vectorized
(49039, 12054) (49039,)
(24155, 12054) (24155,)
(36051, 12054) (36051,)
=======================================================================================================================
=
Wall time: 22.4 s
```

2. clean_titles

In [31]:

```
%%time

# We are considering only the words which appeared in at least 10 documents(rows or projects).
# https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

from sklearn.feature_extraction.text import CountVectorizer
vectorizer_title_bow = CountVectorizer(min_df=10)
vectorizer_title_bow.fit(X_train['clean_titles'].values) # fit has to happen only on train data

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

print("Title vectorized")
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)
print("="*120)
```

```
Title vectorized
(49039, 2082) (49039,)
(24155, 2082) (24155,)
(36051, 2082) (36051,)
=======================================================================================================================
=
Wall time: 1.19 s
```

TF-IDF

1. clean_essay

In [30]:

```
%%time
# Vectorizing the essay column

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_selection import SelectKBest, chi2

vectorizer_tfidf_essay = TfidfVectorizer(min_df=10)

# We will fit the train data only
vectorizer_tfidf_essay.fit(X_train['clean_essays'].values)

# we use the fitted TfidfVectorizer to convert the text to vector
X_train_essay_tfidf = vectorizer_tfidf_essay.transform(X_train['clean_essays'].values)
X_cv_essay_tfidf = vectorizer_tfidf_essay.transform(X_cv['clean_essays'].values)
X_test_essay_tfidf = vectorizer_tfidf_essay.transform(X_test['clean_essays'].values)


print("Essay vectorized")
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)
print("="*120)
```

```
Essay vectorized
(49039, 12016) (49039,)
(24155, 12016) (24155,)
(36051, 12016) (36051,)
=======================================================================================================================
=
Wall time: 29 s
```

2. clean_titles

In [31]:

```
# Vectorizing the project_title column

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_selection import SelectKBest, chi2

vectorizer_tfidf_title = TfidfVectorizer(min_df = 10)

# We will fit the train data only
vectorizer_tfidf_title.fit(X_train['clean_titles'].values)

# we use the fitted CountVectorizer to convert the text to vector
X_train_titles_tfidf = vectorizer_tfidf_title.transform(X_train['clean_titles'].values)
X_cv_titles_tfidf = vectorizer_tfidf_title.transform(X_cv['clean_titles'].values)
X_test_titles_tfidf = vectorizer_tfidf_title.transform(X_test['clean_titles'].values)


print("Titles vectorized")
print(X_train_titles_tfidf.shape, y_train.shape)
print(X_cv_titles_tfidf.shape, y_cv.shape)
print(X_test_titles_tfidf.shape, y_test.shape)
print("="*120)
```

```
Titles vectorized
(49039, 2080) (49039,)
(24155, 2080) (24155,)
(36051, 2080) (36051,)
=======================================================================================================================
=
```

6. Appling NB() on different kind of featurization as mentioned in the instructions

1. Applying Naive Bayes on BOW, SET 1

Merging the categorical, numerical and text features

In [32]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
# https://stackoverflow.com/questions/54226138/constructing-sparse-csr-matrix-directly-vs-using-coo-tocsr-scipy
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)

from scipy.sparse import hstack

# Training data
X_tr = hstack((X_train_essay_bow, X_train_titles_bow, X_train_clean_category, X_train_clean_subcategories,
               X_train_project_grade, X_train_school_state, X_train_teacher_prefix,
               X_train_previous_projects, X_train_price, X_train_quantity)).tocsr()
# CV data
X_cr = hstack((X_cv_essay_bow, X_cv_titles_bow, X_cv_clean_category, X_cv_clean_subcategories,
               X_cv_project_grade, X_cv_school_state, X_cv_teacher_prefix,
               X_cv_previous_projects, X_cv_price, X_cv_quantity)).tocsr()
# Test data
X_te = hstack((X_test_essay_bow, X_test_titles_bow, X_test_clean_category, X_test_clean_subcategories,
               X_test_project_grade, X_test_school_state, X_test_teacher_prefix,
               X_test_previous_projects, X_test_price, X_test_quantity)).tocsr()


print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
```

```
Final Data matrix
(49039, 14238) (49039,)
(24155, 14238) (24155,)
(36051, 14238) (36051,)
```

Dimensions of the hstacked features

In [33]:

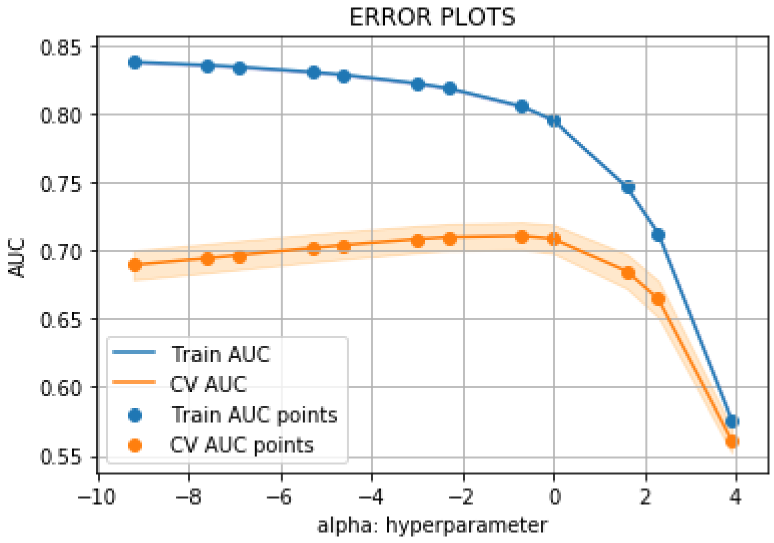
```
print('Training DATA\n')
print('ESSAY : ', X_train_essay_bow.shape)
print('Title : ', X_train_titles_bow.shape)
print('Categorical Data : ', (X_train_clean_category.shape + X_train_clean_subcategories.shape + X_train_project_grade.s
                              X_train_school_state.shape + X_train_teacher_prefix.shape))
print('Numerical Data : ', (X_train_previous_projects.shape + X_train_price.shape + X_train_quantity.shape))

print('\n','='*120)

print('CV DATA\n')
print('ESSAY : ', X_cv_essay_bow.shape)
print('Title : ', X_cv_titles_bow.shape)
print('Categorical Data : ', (X_cv_clean_category.shape + X_cv_clean_subcategories.shape + X_cv_project_grade.shape + \
                              X_cv_school_state.shape + X_cv_teacher_prefix.shape))
print('Numerical Data : ', (X_cv_previous_projects.shape + X_cv_price.shape + X_cv_quantity.shape))

print('\n','='*120)

print('Test DATA\n')
print('ESSAY : ', X_test_essay_bow.shape)
print('Title : ', X_test_titles_bow.shape)
print('Categorical Data : ', (X_test_clean_category.shape + X_test_clean_subcategories.shape + X_test_project_grade.shap
                              X_test_school_state.shape + X_test_teacher_prefix.shape))
print('Numerical Data : ', (X_test_previous_projects.shape + X_test_price.shape + X_test_quantity.shape))

print('\n','='*120)
```

```
Training DATA

ESSAY :  (49039, 12054)
Title :  (49039, 2082)
Categorical Data :  (49039, 9, 49039, 30, 49039, 4, 49039, 51, 49039, 5)
Numerical Data :  (49039, 1, 49039, 1, 49039, 1)

 ======================================================================================================================
==
CV DATA

ESSAY :  (24155, 12054)
Title :  (24155, 2082)
Categorical Data :  (24155, 9, 24155, 30, 24155, 4, 24155, 51, 24155, 5)
Numerical Data :  (24155, 1, 24155, 1, 24155, 1)

 ======================================================================================================================
==
Test DATA

ESSAY :  (36051, 12054)
Title :  (36051, 2082)
Categorical Data :  (36051, 9, 36051, 30, 36051, 4, 36051, 51, 36051, 5)
Numerical Data :  (36051, 1, 36051, 1, 36051, 1)

 ======================================================================================================================
==
```

Hyper paramter tuning to find best α (APLHA) (Using GridSearchCV)

In [34]:

```
%%time
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB

# creating naive bayes classifier
nb = MultinomialNB()

# Alpha values
parameters = {'alpha':[50, 10, 5, 1, 0.5, 0.1, 0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001]}

# Finding the best parameter using gridsearchcv and 10-folds
clf = GridSearchCV(nb, parameters, cv=10, scoring='roc_auc', return_train_score=True)
clf.fit(X_tr, 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']


# We use log(alpha) values so as to get a more distinguishable graph because log is monotonous function
# and it won't affect our results

plt.plot(np.log(parameters['alpha']), train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(np.log(parameters['alpha']),train_auc - train_auc_std, train_auc + train_auc_std, alpha=0.2,
                       color='darkblue')

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

plt.scatter(np.log(parameters['alpha']), train_auc, label='Train AUC points')
plt.scatter(np.log(parameters['alpha']), cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



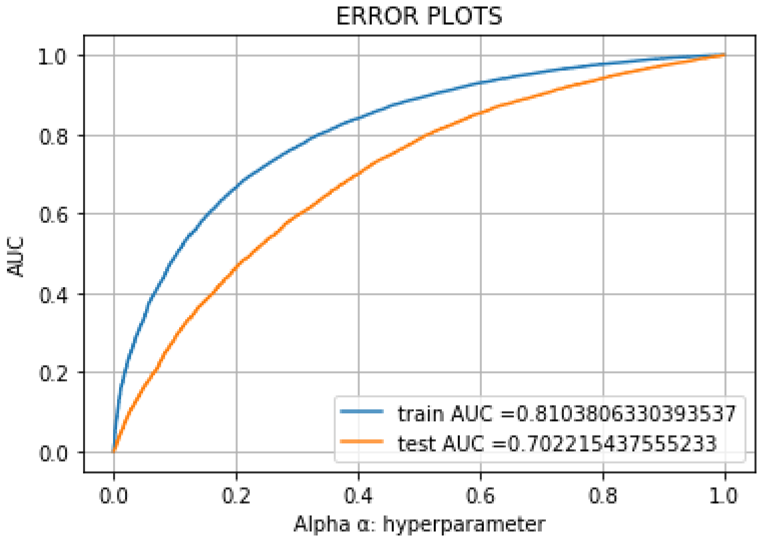
```
Wall time: 19.4 s
```

Now creating the model with best α

In [41]:

```
# From the error plot we choose a such that, we will have maximum AUC on cv data and gap between the train and cv is les

# Here we are choosing the best_a based on GridSearchCV results

best_α = 0.1
```

In [42]:
```
%%time
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
from sklearn.naive_bayes import MultinomialNB

# Creating the classifier
nb = MultinomialNB(alpha=best_α)
nb.fit(X_tr, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

# Getting the predicted probability scores for test and train values
y_train_pred = nb.predict_proba(X_tr)[:,1]
y_test_pred = nb.predict_proba(X_te)[:,1]


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("Alpha α: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



```
Wall time: 286 ms
```

NOTE:

```
- As we can see from the graph.The AUC curve is lower for the test set than the train set.
- The AUC scores for the Train and Test data are : 81% and 70% respectively
- We choose the α value equal to 0.1 because it has maximum AUC on the CV data
```

In [43]:
```
# we are writing our own function for predict, with defined threshold
# we will pick a threshold that will give the least fpr
def predict(proba, threshold, fpr, tpr):

    t = threshold[np.argmax(tpr*(1-fpr))]

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

    #print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [44]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

from sklearn.metrics import classification_report

y_pred_new = nb.predict(X_te)
target_names = ['class 0', 'class 1']
print(classification_report(y_test, y_pred_new, target_names = target_names))
```

```
              precision    recall  f1-score   support

     class 0       0.30      0.49      0.37      5459
     class 1       0.90      0.80      0.84     30592

   micro avg       0.75      0.75      0.75     36051
   macro avg       0.60      0.64      0.61     36051
weighted avg       0.81      0.75      0.77     36051
```

In [45]:

```
%%time
# https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score

y_pred_new = nb.predict(X_te)
print("Accuracy on test set: {}".format(accuracy_score(y_test, y_pred_new)))
print("Precision on test set: {}".format(precision_score(y_test, y_pred_new)))
print("Recall on test set: {}".format(recall_score(y_test, y_pred_new)))
print("F1-Score on test set: {}".format(f1_score(y_test, y_pred_new)))
```

```
Accuracy on test set: 0.7510193891986353
Precision on test set: 0.8971777157136558
Recall on test set: 0.7980517782426778
F1-Score on test set: 0.8447166286070168
Wall time: 63.8 ms
```

In [46]:

```
from sklearn.metrics import confusion_matrix

print("="*120)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)))
print("="*120)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr)))
```

```
========================================================================================================================
=
Train confusion matrix
[[ 5534  1891]
 [11322 30292]]
========================================================================================================================
=
Test confusion matrix
[[ 3268  2191]
 [ 9196 21396]]
```

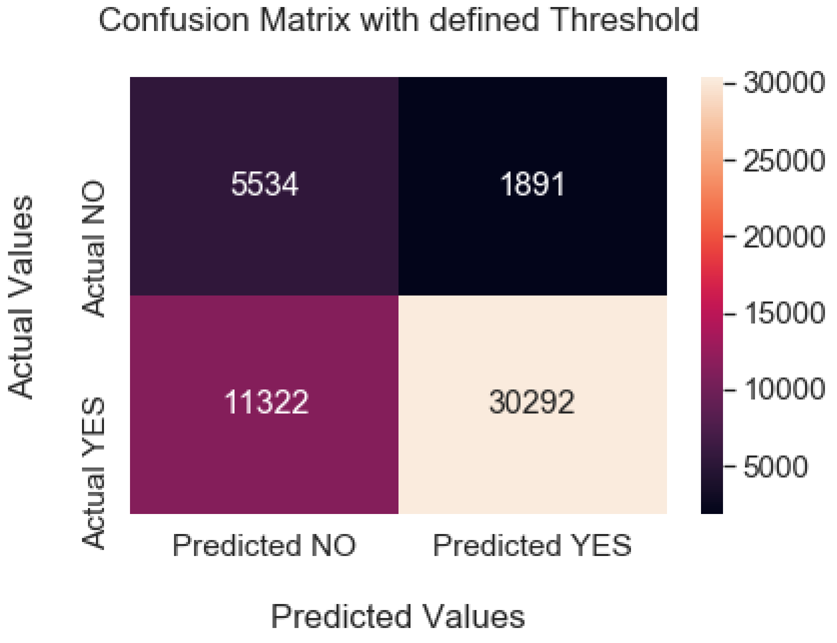
Function to create the confusion matrix

In [47]:

```
# https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix

def get_confusion_matrix(y_tr_ts, y_tr_ts_pred, tr_ts_thresholds, tr_ts_fpr, tr_ts_tpr):

    """Function to get heatmap confusion matrix"""

    # Creating the confusion matrix dataframe
    df_cm = pd.DataFrame(confusion_matrix(y_tr_ts, predict(y_tr_ts_pred, tr_ts_thresholds, tr_ts_fpr, tr_ts_tpr))
                     , range(2),range(2))

    df_cm.columns = ['Predicted NO','Predicted YES']
    df_cm = df_cm.rename({0: 'Actual NO', 1: 'Actual YES'})

    plt.title('Confusion Matrix with defined Threshold\n ')

    sns.set(font_scale=1.4)#for label size
    sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
```

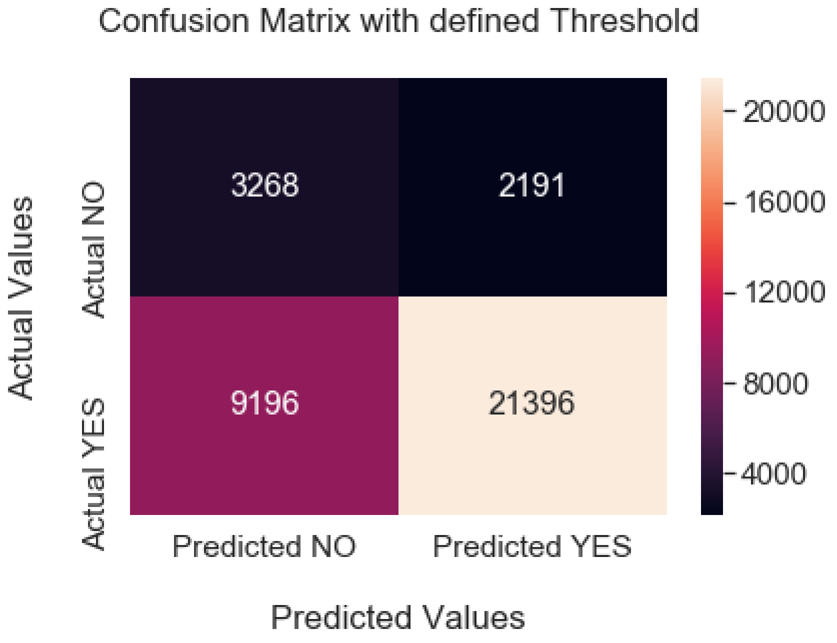
Confusion Matrix on train data

In [49]:
```
get_confusion_matrix(y_train, y_train_pred, tr_thresholds, train_fpr, train_tpr)
plt.xlabel('\nPredicted Values')
plt.ylabel('Actual Values\n')
plt.show()
```



Confusion Matrix on test data

In [50]:
```
get_confusion_matrix(y_test, y_test_pred, tr_thresholds, test_fpr, test_tpr)
plt.xlabel('\nPredicted Values')
plt.ylabel('Actual Values\n')
plt.show()
```



Feature importance

Top 10 important features of positive and negative class from SET 1

Merging the categorical, numerical and text features names

In [51]:

```
bow_features_names = []

for a in vectorizer_clean_category.get_feature_names() :
    bow_features_names.append(a)

for a in vectorizer_clean_subcategories.get_feature_names() :
    bow_features_names.append(a)

for a in vectorizer_essay_bow.get_feature_names() :
    bow_features_names.append(a)

for a in vectorizer_project_grade_category.get_feature_names() :
    bow_features_names.append(a)

for a in vectorizer_school_state.get_feature_names() :
    bow_features_names.append(a)

for a in vectorizer_teacher_prefix.get_feature_names() :
    bow_features_names.append(a)

for a in vectorizer_title_bow.get_feature_names() :
    bow_features_names.append(a)

bow_features_names.append("price")
bow_features_names.append("quantity")
bow_features_names.append("prev_proposed_projects")

len(bow_features_names)
```

Out[51]:

```
14238
```

NOTE:

```
- There are total 14238 features and now we have to find the most imformative features using the coef_ parameter
```

Creating the function to print most informative features

In [52]:

```
## Function to print the most important features using coef_
# n=10 -> prints top 10 features

#https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
#http://zetcode.com/python/prettytable/
#https://stackoverflow.com/questions/30017491/problems-obtaining-most-informative-features-with-scikit-learn
#https://stackoverflow.com/questions/26976362/how-to-get-most-informative-features-for-scikit-learn-classifier-for-diffe

from prettytable import PrettyTable

def most_informative_feature(feature_names, classifier, n=10):

    """Function to print the most informative features"""

    # Class Labels
    class_labels = classifier.classes_

    top10_class1 = sorted(zip(classifier.coef_[0], feature_names))[:n]
    top10_class2 = sorted(zip(classifier.coef_[0], feature_names))[-n:]

    print("\nMost Important Features of Negative class : \n")

    x = PrettyTable()
    x.field_names = ["Class label", "Negative-value", "Negative Features"]

    for (coef_0, feature_names_0) in top10_class1:
        x.add_row([class_labels[0], coef_0, feature_names_0])
    print(x)

    print('\n', '='*120)

    print("\nMost Important Features of Positive class : \n")

    x = PrettyTable()
    x.field_names = ["Class label", "Positive-value", "Positive Features"]

    for (coef_1, feature_names_1) in reversed(top10_class2):
        x.add_row([class_labels[1], coef_1, feature_names_1])
    print(x)

    print('\n', '='*120)
```

In [53]:

```
# Function call

most_informative_feature(bow_features_names, nb)
```

```
Most Important Features of Negative class :

+-------------+---------------------+-------------------+
| Class label |    Negative-value   | Negative Features |
+-------------+---------------------+-------------------+
|      0      |  -17.93472452010426 |        why        |
|      0      |  -17.93472452010426 |        wide       |
|      0      |  -17.93472452010426 |       wiggle      |
|      0      |  -17.93472452010426 |      wigglers     |
|      0      | -14.002898887379937 |       curve       |
|      0      | -14.002898887379937 |    embarrassing   |
|      0      | -14.002898887379937 |       harry       |
|      0      | -14.002898887379937 |        hip        |
|      0      | -14.002898887379937 |       intent      |
|      0      | -14.002898887379937 |      laptops      |
+-------------+---------------------+-------------------+

 ======================================================================================================================
==

Most Important Features of Positive class :

+-------------+---------------------+-------------------+
| Class label |    Positive-value   | Positive Features |
+-------------+---------------------+-------------------+
|      1      | -3.0081656556220313 |     stressing     |
|      1      |  -4.148271060256096 |       saying      |
|      1      |  -4.517349497315605 |       launch      |
|      1      | -4.5368209858698005 |    circulatory    |
|      1      |  -4.803062509303945 |      nintendo     |
|      1      |  -4.854765638715428 |      laughed      |
|      1      |  -4.87916141967999  |      headsets     |
|      1      |  -5.02278345320499  |       males       |
|      1      |  -5.042375745717369 |      mundane      |
|      1      |  -5.152148066832993 |       rapid       |
+-------------+---------------------+-------------------+

 ======================================================================================================================
==
```

NOTE:

1. As we can see in the table, we have the top 10 features (both positive and negative class) which are the 10 most important features.
2. The F1_Score obtained in the test data : 0.8447166286070168

2. Applying Naive Bayes on TFIDF, SET 2

Merging the categorical, numerical and text features

In [32]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
# https://stackoverflow.com/questions/54226138/constructing-sparse-csr-matrix-directly-vs-using-coo-tocsr-scipy
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)

from scipy.sparse import hstack

# Training data
X_tr = hstack((X_train_essay_tfidf, X_train_titles_tfidf, X_train_clean_category, X_train_clean_subcategories,
               X_train_project_grade, X_train_school_state, X_train_teacher_prefix,
               X_train_previous_projects, X_train_price, X_train_quantity)).tocsr()
# CV data
X_cr = hstack((X_cv_essay_tfidf, X_cv_titles_tfidf, X_cv_clean_category, X_cv_clean_subcategories,
               X_cv_project_grade, X_cv_school_state, X_cv_teacher_prefix,
               X_cv_previous_projects, X_cv_price, X_cv_quantity)).tocsr()
# Test data
X_te = hstack((X_test_essay_tfidf, X_test_titles_tfidf, X_test_clean_category, X_test_clean_subcategories,
               X_test_project_grade, X_test_school_state, X_test_teacher_prefix,
               X_test_previous_projects, X_test_price, X_test_quantity)).tocsr()
```

In [33]:
```
## Print the final data matrix

print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
```

```
Final Data matrix
(49039, 14198) (49039,)
(24155, 14198) (24155,)
(36051, 14198) (36051,)
```

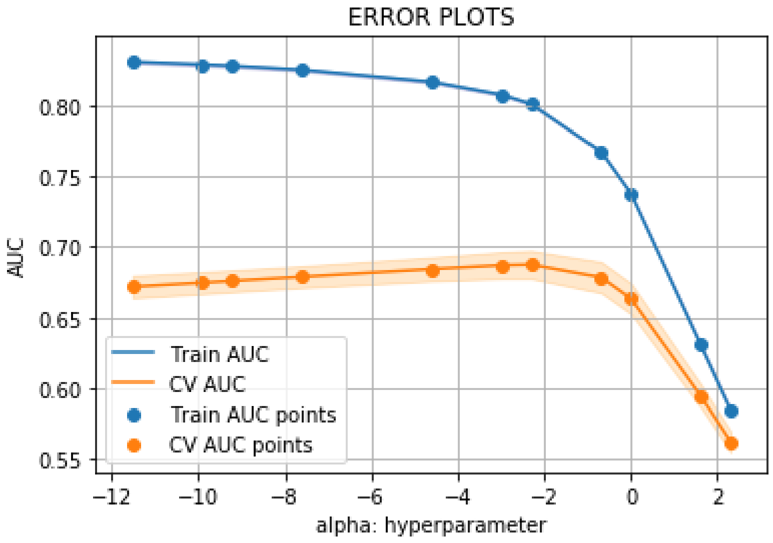
Dimensions of the hstacked features

In [34]:
```
print('Training DATA\n')
print('ESSAY : ', X_train_essay_tfidf.shape)
print('Title : ', X_train_titles_tfidf.shape)
print('Categorical Data : ', (X_train_clean_category.shape + X_train_clean_subcategories.shape + X_train_project_grade.s
                              X_train_school_state.shape + X_train_teacher_prefix.shape))
print('Numerical Data : ', (X_train_previous_projects.shape + X_train_price.shape + X_train_quantity.shape))

print('\n','='*120)

print('CV DATA\n')
print('ESSAY : ', X_cv_essay_tfidf.shape)
print('Title : ', X_cv_titles_tfidf.shape)
print('Categorical Data : ', (X_cv_clean_category.shape + X_cv_clean_subcategories.shape + X_cv_project_grade.shape + \
                              X_cv_school_state.shape + X_cv_teacher_prefix.shape))
print('Numerical Data : ', (X_cv_previous_projects.shape + X_cv_price.shape + X_cv_quantity.shape))

print('\n','='*120)

print('Test DATA\n')
print('ESSAY : ', X_test_essay_tfidf.shape)
print('Title : ', X_test_titles_tfidf.shape)
print('Categorical Data : ', (X_test_clean_category.shape + X_test_clean_subcategories.shape + X_test_project_grade.shap
                              X_test_school_state.shape + X_test_teacher_prefix.shape))
print('Numerical Data : ', (X_test_previous_projects.shape + X_test_price.shape + X_test_quantity.shape))

print('\n','='*120)
```

```
Training DATA

ESSAY :  (49039, 12016)
Title :  (49039, 2080)
Categorical Data :  (49039, 9, 49039, 30, 49039, 4, 49039, 51, 49039, 5)
Numerical Data :  (49039, 1, 49039, 1, 49039, 1)

 ======================================================================================================================
==
CV DATA

ESSAY :  (24155, 12016)
Title :  (24155, 2080)
Categorical Data :  (24155, 9, 24155, 30, 24155, 4, 24155, 51, 24155, 5)
Numerical Data :  (24155, 1, 24155, 1, 24155, 1)

 ======================================================================================================================
==
Test DATA

ESSAY :  (36051, 12016)
Title :  (36051, 2080)
Categorical Data :  (36051, 9, 36051, 30, 36051, 4, 36051, 51, 36051, 5)
Numerical Data :  (36051, 1, 36051, 1, 36051, 1)

 ======================================================================================================================
==
```

Hyper paramter tuning to find best α (APLHA) (Using GridSearchCV)

In [35]:
```
%%time
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB

# creating naive bayes classifier
nb = MultinomialNB()

# Alpha values
parameters = {'alpha':[10, 5, 1, 0.5, 0.1, 0.05, 0.01, 0.0005, 0.0001, 0.00005, 0.00001]}

# Finding the best parameter using gridsearchcv and 10-folds
clf = GridSearchCV(nb, parameters, cv=10, scoring='roc_auc', return_train_score=True)
clf.fit(X_tr, 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']


# We use log(alpha) values so as to get a more distinguishable graph because log is monotonous function
# and it won't affect our results

plt.plot(np.log(parameters['alpha']), train_auc, label='Train AUC')
# this code is copied from here: https://stackoverflow.com/a/48803361/4084039
plt.gca().fill_between(np.log(parameters['alpha']),train_auc - train_auc_std,train_auc + train_auc_std,alpha=0.2,color='

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

plt.scatter(np.log(parameters['alpha']), train_auc, label='Train AUC points')
plt.scatter(np.log(parameters['alpha']), cv_auc, label='CV AUC points')


plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

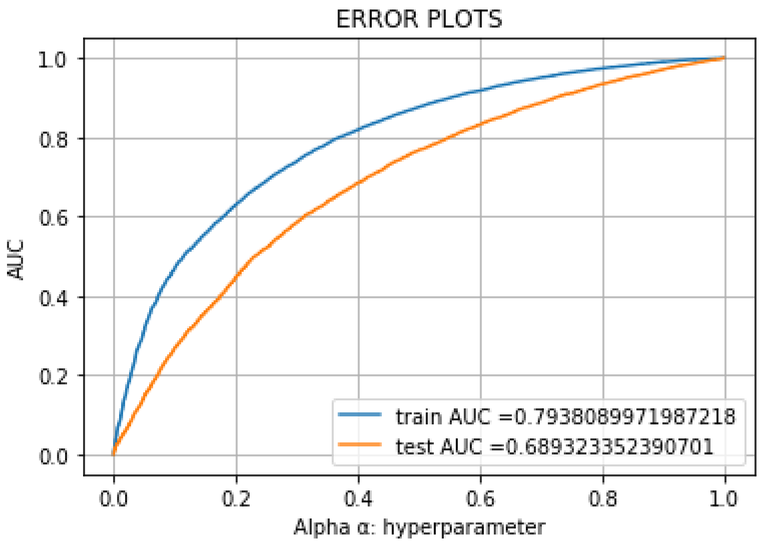


```
Wall time: 18.7 s
```

Now creating the model with best α

In [36]:
```
# From the error plot we choose a such that, we will have maximum AUC on cv data and gap between the train and cv is les

# Here we are choosing the best_a based on GridSearchCV results

best_α = 0.1
```

In [37]:
```
%%time
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc
from sklearn.naive_bayes import MultinomialNB

# Creating the classifier
nb = MultinomialNB(alpha=best_α)
nb.fit(X_tr, y_train)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
# not the predicted outputs

# Getting the predicted probability scores for test and train values
y_train_pred = nb.predict_proba(X_tr)[:,1]
y_test_pred = nb.predict_proba(X_te)[:,1]


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("Alpha α: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



```
Wall time: 287 ms
```

NOTE:

- As we can see from the graph.The AUC curve is lower for the test set than the train set.
- The AUC scores for the Train and Test data are : 79% and 69% respectively
- We choose the α value equal to 0.1 because it has maximum AUC on the CV data

In [38]:
```
# we are writing our own function for predict, with defined threshold
# we will pick a threshold that will give the least fpr
def predict(proba, threshold, fpr, tpr):

    t = threshold[np.argmax(tpr*(1-fpr))]

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

    # print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [39]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

from sklearn.metrics import classification_report

y_pred_new = nb.predict(X_te)
target_names = ['class 0', 'class 1']
print(classification_report(y_test, y_pred_new, target_names = target_names))
```

```
              precision    recall  f1-score   support

     class 0       0.40      0.08      0.13      5459
     class 1       0.86      0.98      0.91     30592

   micro avg       0.84      0.84      0.84     36051
   macro avg       0.63      0.53      0.52     36051
weighted avg       0.79      0.84      0.79     36051
```

In [40]:

```
%%time
# https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score

y_pred_new = nb.predict(X_te)
print("Accuracy on test set: {}".format(accuracy_score(y_test, y_pred_new)))
print("Precision on test set: {}".format(precision_score(y_test, y_pred_new)))
print("Recall on test set: {}".format(recall_score(y_test, y_pred_new)))
print("F1-Score on test set: {}".format(f1_score(y_test, y_pred_new)))
```

```
Accuracy on test set: 0.8427505478350115
Precision on test set: 0.8559716628102951
Recall on test set: 0.9795044456066946
F1-Score on test set: 0.9135809997103614
Wall time: 61.8 ms
```

In [41]:

```
from sklearn.metrics import confusion_matrix

print("="*120)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)))
print("="*120)
print("Test confusion matrix")
print(confusion_matrix(y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr)))
```

```
=======================================================================================================================
=
Train confusion matrix
[[ 5415  2010]
 [11951 29663]]
=======================================================================================================================
=
Test confusion matrix
[[ 3313  2146]
 [ 9892 20700]]
```

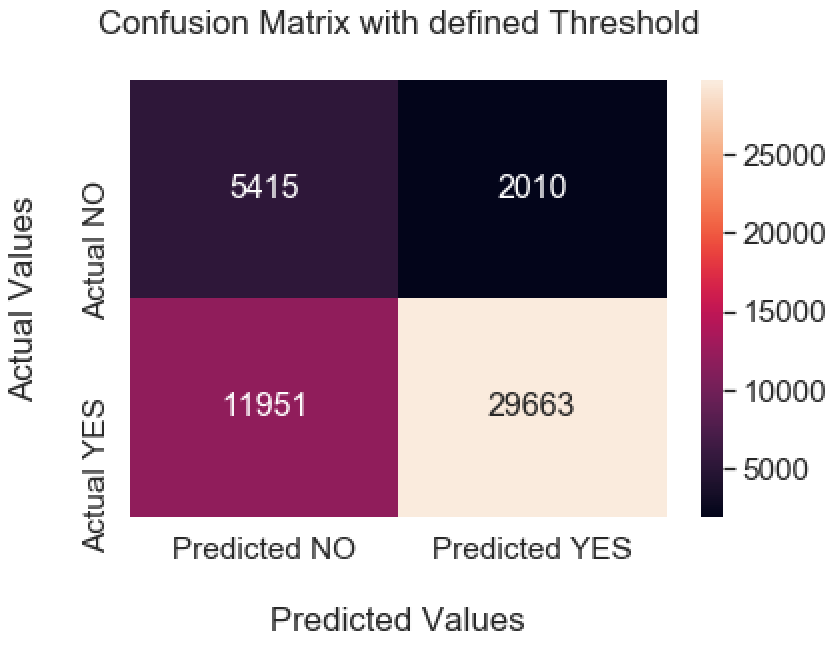
Function to create the confusion matrix

In [42]:

```
# https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix

def get_confusion_matrix(y_tr_ts, y_tr_ts_pred, tr_ts_thresholds, tr_ts_fpr, tr_ts_tpr):

    """Function to get heatmap confusion matrix"""

    # Creating the confusion matrix dataframe
    df_cm = pd.DataFrame(confusion_matrix(y_tr_ts, predict(y_tr_ts_pred, tr_ts_thresholds, tr_ts_fpr, tr_ts_tpr))
                     , range(2),range(2))

    df_cm.columns = ['Predicted NO','Predicted YES']
    df_cm = df_cm.rename({0: 'Actual NO', 1: 'Actual YES'})

    plt.title('Confusion Matrix with defined Threshold\n ')

    sns.set(font_scale=1.4)#for label size
    sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
```

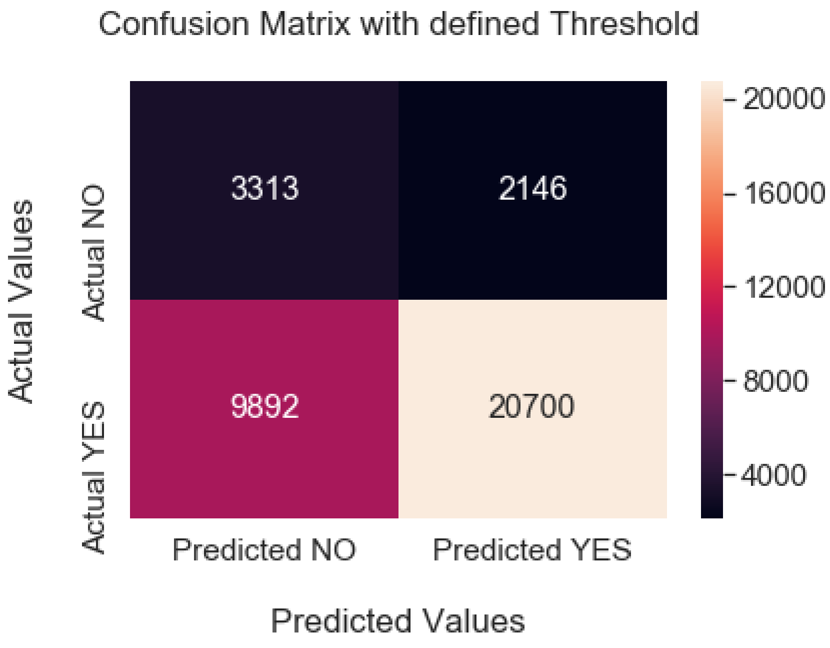
Confusion Matrix on train data

In [44]:
```
get_confusion_matrix(y_train, y_train_pred, tr_thresholds, train_fpr, train_tpr)
plt.xlabel('\nPredicted Values')
plt.ylabel('Actual Values\n')
plt.show()
```



Confusion Matrix on test data

In [45]:
```
get_confusion_matrix(y_test, y_test_pred, tr_thresholds, test_fpr, test_tpr)
plt.xlabel('\nPredicted Values')
plt.ylabel('Actual Values\n')
plt.show()
```



Feature importance

Top 10 important features of positive and negative class from SET 2

Merging the categorical, numerical and text features names

In [46]:

```
tfidf_features_names = []

for a in vectorizer_clean_category.get_feature_names() :
    tfidf_features_names.append(a)

for a in vectorizer_clean_subcategories.get_feature_names() :
    tfidf_features_names.append(a)

for a in vectorizer_tfidf_essay.get_feature_names() :
    tfidf_features_names.append(a)

for a in vectorizer_project_grade_category.get_feature_names() :
    tfidf_features_names.append(a)

for a in vectorizer_school_state.get_feature_names() :
    tfidf_features_names.append(a)

for a in vectorizer_teacher_prefix.get_feature_names() :
    tfidf_features_names.append(a)

for a in vectorizer_tfidf_title.get_feature_names() :
    tfidf_features_names.append(a)

tfidf_features_names.append("price")
tfidf_features_names.append("quantity")
tfidf_features_names.append("prev_proposed_projects")

len(tfidf_features_names)
```

Out[46]:

```
14198
```

NOTE:

```
- There are total 14198 features and now we have to find the most imformative features using the coef_ parameter
```

Creating the function to print most informative features

In [47]:

```
## Function to print the most important features using coef_
# n=10 -> prints top 10 features

#https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html
#http://zetcode.com/python/prettytable/
#https://stackoverflow.com/questions/30017491/problems-obtaining-most-informative-features-with-scikit-learn
#https://stackoverflow.com/questions/26976362/how-to-get-most-informative-features-for-scikit-learn-classifier-for-diffe

from prettytable import PrettyTable

def most_informative_feature(feature_names, classifier, n=10):

    """Function to print the most informative features"""

    # Class Labels
    class_labels = classifier.classes_

    top10_class1 = sorted(zip(classifier.coef_[0], feature_names))[:n]
    top10_class2 = sorted(zip(classifier.coef_[0], feature_names))[-n:]

    print("\nMost Important Features of Negative class : \n")

    x = PrettyTable()
    x.field_names = ["Class label", "Negative-value", "Negative Features"]

    for (coef_0, feature_names_0) in top10_class1:
        x.add_row([class_labels[0], coef_0, feature_names_0])
    print(x)

    print('\n', '='*120)

    print("\nMost Important Features of Positive class : \n")

    x = PrettyTable()
    x.field_names = ["Class label", "Positive-value", "Positive Features"]

    for (coef_1, feature_names_1) in top10_class2:
        x.add_row([class_labels[1], coef_1, feature_names_1])
    print(x)

    print('\n', '='*120)
```

In [48]:

```
# Function call

most_informative_feature(tfidf_features_names, nb)
```

```
Most Important Features of Negative class :

+-------------+---------------------+-------------------+
| Class label |    Negative-value   | Negative Features |
+-------------+---------------------+-------------------+
|      0      | -15.790094832764924 |        wild       |
|      0      | -15.790094832764924 |        will       |
|      0      | -15.790094832764924 |        win        |
|      0      | -15.790094832764924 |        wind       |
|      0      | -13.652900560120722 |      studied      |
|      0      |  -13.62190776436671 |      stomach      |
|      0      | -13.594510789966495 |        dose       |
|      0      | -13.578511853519208 |   communications  |
|      0      | -13.570431115305498 |     preparing     |
|      0      | -13.563620887749224 |     systematic    |
+-------------+---------------------+-------------------+

 ========================================================================================================================
==

Most Important Features of Positive class :

+-------------+---------------------+------------------------+
| Class label |    Positive-value   |   Positive Features    |
+-------------+---------------------+------------------------+
|      1      |  -4.436974336359533 |       whiteboard       |
|      1      |  -4.207820559660709 |      whiteboards       |
|      1      |  -4.006371535277159 |         white          |
|      1      | -3.8948978297361396 |          zone          |
|      1      | -3.8246870949234744 |         waste          |
|      1      | -3.6985125336672553 |         price          |
|      1      |  -3.573522502353647 |          was           |
|      1      |  -3.48670946839626  |         youth          |
|      1      | -3.3297444415245803 | prev_proposed_projects |
|      1      |  -3.106573834555711 |        quantity        |
+-------------+---------------------+------------------------+

 ========================================================================================================================
==
```

NOTE:

1. As we can see in the table, we have the top 10 features (both positive and negative class) which are the 10 most important features.
2. The F1_Score obtained in the test data : 0.9135809997103614

CONCLUSION

In [49]:

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

x = PrettyTable()
x.field_names = ["Vectorizer", "NAIVE BAYES Model" ,"Hyper parameter [α-value]", "AUC Score [train]", "AUC Score [test]"
x.add_row(["Bag of Words", "MultinomialNB", 0.1, 0.81, 0.70])
x.add_row(["TF-IDF", "MultinomialNB", 0.1, 0.79, 0.69])

print(x)
```

```
+--------------+-------------------+---------------------------+-------------------+------------------+
|  Vectorizer  | NAIVE BAYES Model | Hyper parameter [α-value] | AUC Score [train] | AUC Score [test] |
+--------------+-------------------+---------------------------+-------------------+------------------+
| Bag of Words |   MultinomialNB   |            0.1            |        0.81       |       0.7        |
|    TF-IDF    |   MultinomialNB   |            0.1            |        0.79       |       0.69       |
+--------------+-------------------+---------------------------+-------------------+------------------+
```

NOTE:

1. The auc scores of both BOW and TF-IDF are very close to each other.
2. The hyperparameter value for both the vectorizers are same i.e 0.1
3. Naive Bayes is very fast and its time and space complexity is much better than KNN
4. The results took very less time to get displayed but for KNN it took a lot of time to get executed.
5. The scores of Naive Bayes is better than KNN - Both AUC and f1_Score are better for Naive Bayes.
6. We could train our model with the entire data. We didn't have to sample the data as Naive bayes performs very well with huge data.

In [ ]: