DonorsChoose

DonorsChoose.org receives hundreds of thousands of project proposals each year for classroom projects in need of funding. Right now, a large number of volunteers is needed to manually screen each submission before it's approved to be posted on the DonorsChoose.org website.

Next year, DonorsChoose.org expects to receive close to 500,000 project proposals. As a result, there are three main problems they need to solve:

- How to scale current manual processes and resources to screen 500,000 projects so that they can be posted as quickly and as efficiently as possible
- How to increase the consistency of project vetting across different volunteers to improve the experience for teachers
- How to focus volunteer time on the applications that need the most assistance

The goal of the competition 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. DonorsChoose.org can then use this information to identify projects most likely to need further review before approval.

About the DonorsChoose Data Set

The `train.csv` data set provided by DonorsChoose contains the following features:

Feature	Description
`project_id`	A unique identifier for the proposed project. **Example:** `p036502`
`project_title`	Title of the project. **Examples:** • `Art Will Make You Happy!` • `First Grade Fun`
`project_grade_category`	Grade level of students for which the project is targeted. One of the following enumerated values: • `Grades PreK-2` • `Grades 3-5` • `Grades 6-8` • `Grades 9-12`
`project_subject_categories`	One or more (comma-separated) subject categories for the project from the following enumerated list of values: • `Applied Learning` • `Care & Hunger` • `Health & Sports` • `History & Civics` • `Literacy & Language` • `Math & Science` • `Music & The Arts` • `Special Needs` • `Warmth` **Examples:** • `Music & The Arts` • `Literacy & Language, Math & Science`
`school_state`	State where school is located (Two-letter U.S. postal code). **Example:** `WY`
`project_subject_subcategories`	One or more (comma-separated) subject subcategories for the project. **Examples:** • `Literacy`

- Literature & Writing, Social Sciences

Feature	Description
project_resource_summary	An explanation of the resources needed for the project. **Example:** • My students need hands on literacy materials to manage sensory needs!
project_essay_1	First application essay*
project_essay_2	Second application essay*
project_essay_3	Third application essay*
project_essay_4	Fourth application essay*
project_submitted_datetime	Datetime when project application was submitted. **Example:** 2016-04-28 12:43:56.245
teacher_id	A unique identifier for the teacher of the proposed project. **Example:** bdf8baa8fedef6bfeec7ae4ff1c15c56
teacher_prefix	Teacher's title. One of the following enumerated values: • nan • Dr. • Mr. • Mrs. • Ms. • Teacher.
teacher_number_of_previously_posted_projects	Number of project applications previously submitted by the same teacher. **Example:** 2

* See the section **Notes on the Essay Data** for more details about these features.

Additionally, the resources.csv data set provides more data about the resources required for each project. Each line in this file represents a resource required by a project:

Feature	Description
id	A project_id value from the train.csv file. **Example:** p036502
description	Desciption of the resource. **Example:** Tenor Saxophone Reeds, Box of 25
quantity	Quantity of the resource required. **Example:** 3
price	Price of the resource required. **Example:** 9.95

Note: Many projects require multiple resources. The id value corresponds to a project_id in train.csv, so you use it as a key to retrieve all resources needed for a project:

The data set contains the following label (the value you will attempt to predict):

Label	Description
project_is_approved	A binary flag indicating whether DonorsChoose approved the project. A value of 0 indicates the project was not approved, and a value of 1 indicates the project was approved.

Notes on the Essay Data

Prior to May 17, 2016, the prompts for the essays were as follows:

- __project_essay_1:__ "Introduce us to your classroom"
- __project_essay_2:__ "Tell us more about your students"
- __project_essay_3:__ "Describe how your students will use the materials you're requesting"
- __project_essay_3:__ "Close by sharing why your project will make a difference"

Starting on May 17, 2016, the number of essays was reduced from 4 to 2, and the prompts for the first 2 essays were changed to the following:

- __project_essay_1:__ "Describe your students: What makes your students special? Specific details about their background, your neighborhood, and your school are all helpful."

your neighborhood, and your school are all helpful."

- __project_essay_2:__ "About your project: How will these materials make a difference in your students' learning and improve their school lives?"

  For all projects with project_submitted_datetime of 2016-05-17 and later, the values of project_essay_3 and project_essay_4 will be NaN.

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

1.1 Reading Data

In [2]:

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

In [3]:

```
# 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)
project_data.head(2)
```

```
['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_approved']
```

Out[3]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_cate
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Grades PreK-2
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Grades 3-5

In [4]:

```
# 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')
```

1.2 preprocessing of `project_subject_categories`

In [5]:

```
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 & H
unger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            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]))
```

1.3 preprocessing of `project_subject_subcategories`

In [6]:

```
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 & H
unger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            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]))
```

1.3 Text preprocessing

In [7]:

```
# 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 [8]:

```
# 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
```

In [9]:

```
# 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', 'e
ach', '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', "do
esn'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"]
```

In [10]:

```
# 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
[01:24<00:00, 1294.86it/s]
```

In [11]:

```
#Adding processed columns at place of original columns
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)
```

1.4 Preprocessing of `project_title`

In [12]:

```
# 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, 28679.13it/s]
```

In [13]:

```
project_data['clean_titles'] = preprocessed_titles
```

In [14]:

```
# we cannot remove rows where teacher prefix is not available therefore we are replacing 'nan' val
ue with
# 'null'(string)
#https://stackoverflow.com/questions/42224700/attributeerror-float-object-has-no-attribute-split
project_data['teacher_prefix'] = project_data['teacher_prefix'].fillna('null')
```

function_for_getting_confusion_matrix

In [15]:

```
project_data.count()#function to get heatmap confusion matrix
def get_confusion_matrix(clf,X,y):
    y_pred = clf.predict(X)
    df_cm = pd.DataFrame(confusion_matrix(y, 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 [16]:

```
def informative_feature(feature_names,classifier,n=20):
    topn_class1 = sorted(zip(classifier.feature_log_prob_[1], feature_names),reverse=True)[:n]
    topn_class2 = sorted(zip(classifier.feature_log_prob_[0], feature_names),reverse=True)[:n]
    top = zip(topn_class1, topn_class2)
    x = PrettyTable()
    x.field_names = ["P-value","Positives","N-value","Negatives"]
    #print("\t\tPositive\t\t\t\tNegative")
    for (coef_1, fn_1), (coef_2, fn_2) in top:
        x.add_row([coef_1, fn_1, coef_2, fn_2])
    print(x)
```

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

In [17]:

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

In [18]:

```
 #train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)
```

In [19]:

```
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)
print("="*100)
```

```
(49041, 17) (49041,)
(24155, 17) (24155,)
(36052, 17) (36052,)
====================================================================================================
```

1.5 Preparing data for models

1.5.1 Vectorizing Categorical data

In [20]:

```
project_data.columns
```

Out[20]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_grade_category', 'project_title',
       'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'price', 'quantity',
       'clean_categories', 'clean_subcategories', 'essay', 'clean_essays',
       'clean_titles'],
      dtype='object')
```

- https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/

In [21]:

```
feature_list = []
def add_feature(vectorizer,feature_list):
    for feature in vectorizer.get_feature_names():
        feature_list.append(feature)
```

one_hot_encoding_for_clean_categories

In [22]:

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


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

print("After vectorizations")
print(X_train_clean_cat_ohe.shape, y_train.shape)
print(X_cv_clean_cat_ohe.shape, y_cv.shape)
print(X_test_clean_cat_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)

add_feature(vectorizer,feature_list)
```

```
After vectorizations
(49041, 9) (49041,)
(24155, 9) (24155,)
(36052, 9) (36052,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language',
'math_science', 'music_arts', 'specialneeds', 'warmth']
====================================================================================================
```

one_hot_encoding of clean_sub_categories

In [23]:

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


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

print("After vectorizations")
print(X_train_clean_subcat_ohe.shape, y_train.shape)
print(X_cv_clean_subcat_ohe.shape, y_cv.shape)
print(X_test_clean_subcat_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)

add_feature(vectorizer,feature_list)
```

```
After vectorizations
(49041, 30) (49041,)
(24155, 30) (24155,)
(36052, 30) (36052,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government',
'college_careerprep', 'communityservice', 'earlydevelopment', 'economics', 'environmentalscience',
'esl', 'extracurricular', 'financialliteracy', 'foreignlanguages', 'gym_fitness',
'health_lifescience', 'health_wellness', 'history_geography', 'literacy', 'literature_writing', 'm
athematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'socia
lsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
===================================================================================================
```

In [24]:

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

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

print("After vectorizations")
print(X_train_teacher_ohe.shape, y_train.shape)
print(X_cv_teacher_ohe.shape, y_cv.shape)
print(X_test_teacher_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)

add_feature(vectorizer,feature_list)
```

```
After vectorizations
(49041, 6) (49041,)
(24155, 6) (24155,)
(36052, 6) (36052,)
['dr', 'mr', 'mrs', 'ms', 'null', 'teacher']
===================================================================================================
```

one_hot_encoding_for_school_state

In [25]:

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

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

print("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)
print(X_cv_state_ohe.shape, y_cv.shape)
print(X_test_state_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)

add_feature(vectorizer,feature_list)
```

```
After vectorizations
(49041, 51) (49041,)
(24155, 51) (24155,)
(36052, 51) (36052,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'k
s', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'mo', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm',
'nv', 'ny', 'oh', 'ok', 'or', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv
', 'wy']
===================================================================================================
```

In [26]:

```
X_train.head(2)
```

Out[26]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_cat
3906	20259	p238774	29c4b27156dc7b02e87ca46a62d6b44b	Mrs.	NC	2016-05-17 21:54:34	Grades PreK-2
77044	18494	p005457	f70af14d400b66bfafa7beb9bfe40ccf	Mrs.	PA	2016-12-21 13:48:03	Grades 3-5

one_hot_encoding for project_grade_category

In [27]:

```
#This step is to intialize a vectorizer with vocab from train data
from collections import Counter
my_counter4 = Counter()
for word in X_train['project_grade_category'].values:
    my_counter4.update(word.split())

# dict sort by value python: https://stackoverflow.com/a/613218/4084039
project_grade_category_dict = dict(my_counter4)
sorted_project_grade_category_dict = dict(sorted(project_grade_category_dict.items(), key=lambda
kv: kv[1]))
```

In [28]:

In [28]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_project_grade_category_dict.keys()), lowercase
=False, binary=True)
vectorizer.fit(X_train['project_grade_category'].values) # fit has to happen only on train data

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

print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
print(X_cv_grade_ohe.shape, y_cv.shape)
print(X_test_grade_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)

add_feature(vectorizer,feature_list)
```

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

In [29]:

```
X_train.head(2)
```

Out[29]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_cat
3906	20259	p238774	29c4b27156dc7b02e87ca46a62d6b44b	Mrs.	NC	2016-05-17 21:54:34	Grades PreK-2
77044	18494	p005457	f70af14d400b66bfafa7beb9bfe40ccf	Mrs.	PA	2016-12-21 13:48:03	Grades 3-5

1.5.3 Vectorizing Numerical features

In [30]:

```
from sklearn.preprocessing import StandardScaler
standard_vec = 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.
standard_vec.fit(X_train['price'].values.reshape(-1,1))

X_train_price_std = standard_vec.transform(X_train['price'].values.reshape(-1,1))
```

```
X_cv_price_std = standard_vec.transform(X_cv['price'].values.reshape(-1,1))
X_test_price_std = standard_vec.transform(X_test['price'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_price_std.shape, y_train.shape)
print(X_cv_price_std.shape, y_cv.shape)
print(X_test_price_std.shape, y_test.shape)
print("="*100)

# As this is a numerical feature we will add an entry named as price in put feature_list
list/array
feature_list.append("Price")
```

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

vectorizing numerical_feature:teache_number_of_previously_posted_projects

In [31]:

```
from sklearn.preprocessing import StandardScaler
standard_vec = 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.
standard_vec.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

X_train_projects_std =
standard_vec.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1
))
X_cv_projects_std = standard_vec.transform(X_cv['teacher_number_of_previously_posted_projects'].va
lues.reshape(-1,1))
X_test_projects_std = standard_vec.transform(X_test['teacher_number_of_previously_posted_projects'
].values.reshape(-1,1))

print("After vectorizations")
print(X_train_projects_std.shape, y_train.shape)
print(X_cv_projects_std.shape, y_cv.shape)
print(X_test_projects_std.shape, y_test.shape)
print("="*100)

# As this is a numerical feature we will add an entry named as Num previous projects in put featur
e_list list/array
feature_list.append("num_prev_projects")
```

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

vectorizing numerical feature:quantity

In [32]:

```
from sklearn.preprocessing import StandardScaler
standard_vec = StandardScaler(with_mean = False)
# 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.
standard_vec.fit(X_train['quantity'].values.reshape(-1,1))

X_train_qty_std = standard_vec.transform(X_train['quantity'].values.reshape(-1,1))
X_cv_qty_std = standard_vec.transform(X_cv['quantity'].values.reshape(-1,1))
X_test_qty_std = standard_vec.transform(X_test['quantity'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_qty_std.shape, y_train.shape)
print(X_cv_qty_std.shape, y_cv.shape)
print(X_test_qty_std.shape, y_test.shape)
print("="*100)

# As this is a numerical feature we will add an entry named as quantity in put feature_list list/a
rray
feature_list.append("Quantity")
```

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

In [33]:

```
# Creating a copy of feature list for TFIDF
feature_list_2 = feature_list.copy()
```

In [34]:

```
len(feature_list_2)
```

Out[34]:

```
104
```

1.5.2 Vectorizing Text data

1.5.2.1 Bag of words

vectorizing_essay_BoW

In [35]:

```
from sklearn.feature_extraction.text import CountVectorizer
bow = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
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 = bow.transform(X_train['clean_essays'].values)
X_cv_essay_bow = bow.transform(X_cv['clean_essays'].values)
X_test_essay_bow = bow.transform(X_test['clean_essays'].values)

print("After vectorizations")
print(X_train_essay_bow.shape, y_train.shape)
print(X_cv_essay_bow.shape, y_cv.shape)
print(X_test_essay_bow.shape, y_test.shape)
print("="*100)
add_feature(bow,feature_list)
```

```
After vectorizations
(49041, 5000) (49041,)
(24155, 5000) (24155,)
(36052, 5000) (36052,)
====================================================================================================

After vectorizations
```

```
(49041, 4097) (49041,)
(24155, 4097) (24155,)
(36052, 4097) (36052,)
====================================================================================================

After vectorizations
(49041, 5000) (49041,)
(24155, 5000) (24155,)
(36052, 5000) (36052,)
====================================================================================================
```

vectorizing_title_Bow

In [ ]:

```
# we use the fitted CountVectorizer to convert the text to vector
bow_titles = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
bow_titles.fit(X_train['clean_titles'].values) # fit has to happen only on train data

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

print("After vectorizations")
print(X_train_titles_bow.shape, y_train.shape)
print(X_cv_titles_bow.shape, y_cv.shape)
print(X_test_titles_bow.shape, y_test.shape)
print("="*100)
add_feature(bow_titles,feature_list)
```

vectorizing_resource_summary_Bow

In [ ]:

```
# we use the fitted CountVectorizer to convert the text to vector
bow_summary = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
bow_summary.fit(X_train['project_resource_summary'].values) # fit has to happen only on train data

X_train_summary_bow = bow.transform(X_train['project_resource_summary'].values)
X_cv_summary_bow = bow.transform(X_cv['project_resource_summary'].values)
X_test_summary_bow = bow.transform(X_test['project_resource_summary'].values)

print("After vectorizations")
print(X_train_summary_bow.shape, y_train.shape)
print(X_cv_summary_bow.shape, y_cv.shape)
print(X_test_summary_bow.shape, y_test.shape)
print("="*100)
add_feature(bow_summary,feature_list)
```

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

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

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
- text : text data
- project_resource_summary: text data (optinal)

- quantity : numerical (optinal)
- teacher_number_of_previously_posted_projects : numerical
- price : numerical
```

2.4.1 Applying Naive Bayes on BOW, SET 1

1.5.4 Merging all the above features

In [36]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr =
hstack((X_train_clean_cat_ohe,X_train_clean_subcat_ohe,X_train_teacher_ohe,X_train_state_ohe,X_trai
n_grade_ohe,X_train_price_std,X_train_projects_std,X_train_qty_std,X_train_essay_bow,X_train_titles
_bow,X_train_summary_bow)).tocsr()
X_cr =
hstack((X_cv_clean_cat_ohe,X_cv_clean_subcat_ohe,X_cv_teacher_ohe,X_cv_state_ohe,X_cv_grade_ohe,X_c
v_price_std,X_cv_projects_std,X_cv_qty_std,X_cv_essay_bow,X_cv_titles_bow,X_cv_summary_bow)).tocsr
()
X_te = hstack((X_test_clean_cat_ohe,X_test_clean_subcat_ohe,X_test_teacher_ohe,X_test_state_ohe,X_
test_grade_ohe,X_test_price_std,X_test_projects_std,X_test_qty_std,X_test_essay_bow,X_test_titles_b
ow,X_test_summary_bow)).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)
print("="*100)
```

```
Final Data matrix
(49041, 14201) (49041,)
(24155, 14201) (24155,)
(36052, 14201) (36052,)
===================================================================================================
```

In [37]:

```
#length of feature list must be equal to that of final data matrix
len(feature_list)
```

Out[37]:

```
14201
```

2. Naive Bayes

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

Apply Naive Bayes on different kind of featurization as mentioned in the instructions
For Every model that you work on make sure you do the step 2 and step 3 of instrucations

In [38]:

```
#using batch_prediction

def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your cr_loop will be 49041 - 49041%1000 = 49000
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [39]:

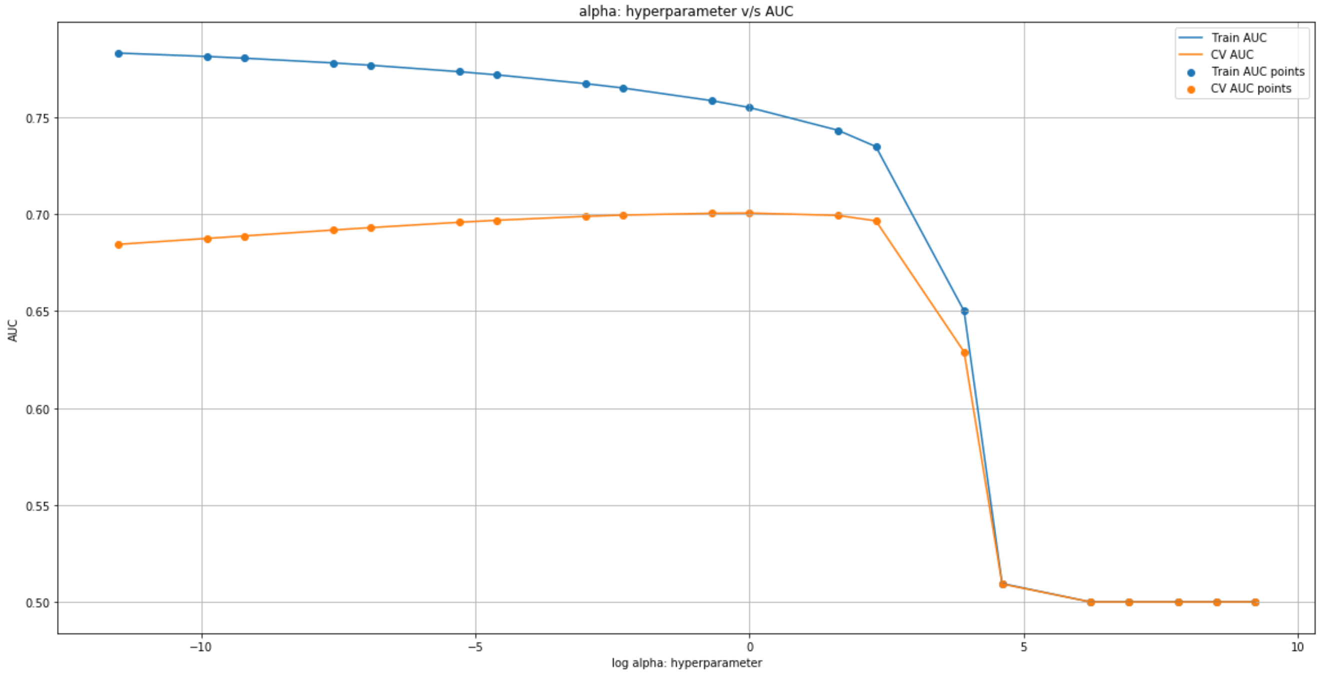
```
import matplotlib.pyplot as plt
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
import math

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

alphas = [0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100, 5
00, 1000, 2500, 5000, 10000]

for i in tqdm(alphas):
    nb = MultinomialNB(alpha = i)
    nb.fit(X_tr, y_train)

    y_train_pred = batch_predict(nb, X_tr)
    y_cv_pred = batch_predict(nb, X_cr)


    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # 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%|██████████████████████████████████████████████████████████████████████████████| 20/20
[00:08<00:00,  2.56it/s]
```

```
100%|████████████████████████████████████████████████████████████████████████████| 2
0/20 [00:00<?, ?it/s]
```

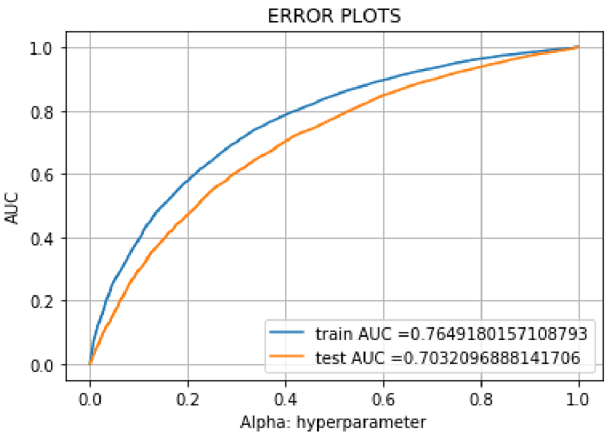
In [40]:

```
plt.figure(figsize=(20,10))
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 [41]:

```
# 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


nb = MultinomialNB(alpha=0.1)
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 outputs1

y_train_pred = batch_predict(nb, X_tr)
y_test_pred = batch_predict(nb, X_te)

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



2.4.1.2 Top 10 important features of negative class from SET 1

In [42]:

```
important_feature = pd.DataFrame({'features': feature_list,'prob_value': nb.coef_[0],})
```

In [43]:

```
#Least important features
important_feature.sort_values(['prob_value'], ascending=[False]).head(20)
```

Out[43]:

	features	prob_value
4215	students	-3.244269
3773	school	-4.385336
2485	learning	-4.750550
801	classroom	-4.784208
3081	not	-5.042365
2429	learn	-5.095693
2049	help	-5.118782
13312	their art	-5.142857
12092	order to create	-5.180687
13490	they can read	-5.226390
100	Grades	-5.231471
2771	many	-5.258679
2981	nannan	-5.275688
2995	need	-5.387361
3583	reading	-5.390117
4996	work	-5.392881
4808	use	-5.456749
101	Price	-5.470686
2663	love	-5.559234
199	able	-5.576185

2.4.1.1 Top 10 important features of positive class from SET 1

In [44]:

```
#Most important features
important_feature.sort_values(['prob_value'], ascending=[True]).head(20)
```

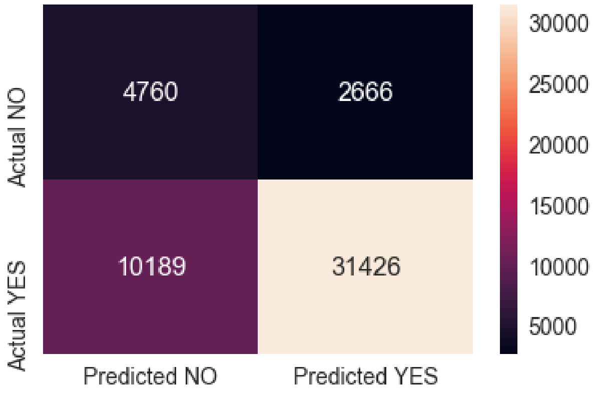
Out[44]:

	features	prob_value
10321	during our	-18.170275
12181	participation	-18.170275
12182	partitions	-18.170275
12183	partner	-18.170275
10022	chromebooks in order	-18.170275
12188	pedometers	-18.170275
10021	chromebooks in	-18.170275
10020	chromebooks for	-18.170275
12198	pens	-18.170275
12180	participate in	-18.170275
12201	people	-18.170275
12209	phonemic	-18.170275
12210	phonemic awareness	-18.170275
12215	photography	-18.170275
12216	photos	-18.170275
12218	physical activity	-18.170275
12219	physical education	-18.170275
10019	chromebooks and	-18.170275
12235	plant	-18.170275
12205	performance	-18.170275

train_confusion_matrix_bow

In [45]:

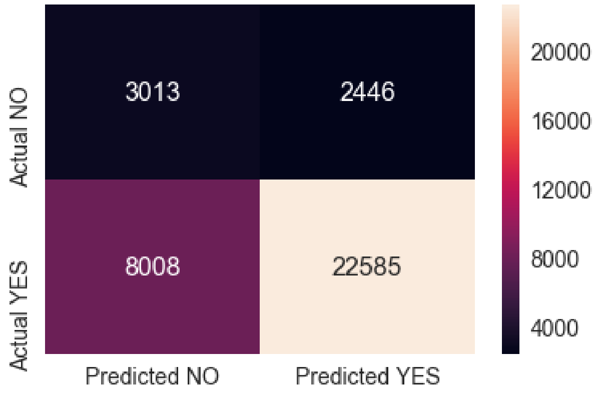
```
get_confusion_matrix(nb,X_tr,y_train)
```



test_confusion_matrix_of_bow

In [46]:

```
get_confusion_matrix(nb,X_te,y_test)
```



essay_tfidf_vectorizing

In [47]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_selection import SelectKBest, chi2
tfidf = TfidfVectorizer(min_df=10)
tfidf.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_tfidf = tfidf.transform(X_train['clean_essays'].values)
X_cv_essay_tfidf = tfidf.transform(X_cv['clean_essays'].values)
X_test_essay_tfidf = tfidf.transform(X_test['clean_essays'].values)


#Selecting top 2000 best features from the generated tfidf features
selector = SelectKBest(chi2, k = 2000 )
selector.fit(X_train_essay_tfidf,y_train)
X_train_essay_2000 = selector.transform(X_train_essay_tfidf)
X_cv_essay_2000 = selector.transform(X_cv_essay_tfidf)
X_test_essay_2000 = selector.transform(X_test_essay_tfidf)
print(X_train_essay_2000.shape)
print(X_cv_essay_2000.shape)
print(X_test_essay_2000.shape)
```

```
(49041, 2000)
(24155, 2000)
(36052, 2000)
```

In [48]:

```
cols = selector.get_support(indices=True)
features = tfidf.get_feature_names()
i = 0
feats=[]
while(i < len(features)+1):
    if i in cols:
        feats.append(features[i])
    i+=1
len(feats)
```

Out[48]:

```
2000
```

In [49]:

```
print(len(feature_list_2))
feature_list_2.extend(feats)
print(len(feature_list_2))
```

```
104
2104
```

title_tfidf_vectorizing

In [50]:

```
tfidf_titles = TfidfVectorizer(min_df=10)
tfidf_titles.fit(X_train['clean_titles'].values) # fit has to happen only on train data

# tfidf_titles = TfidfVectorizer(min_df=5)
# tfidf_titles.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_tfidf = tfidf.transform(X_train['clean_titles'].values)
X_cv_titles_tfidf = tfidf.transform(X_cv['clean_titles'].values)
X_test_titles_tfidf = tfidf.transform(X_test['clean_titles'].values)

selector = SelectKBest(chi2, k = 2000 )
selector.fit(X_train_titles_tfidf,y_train)

X_train_titles_2000 = selector.transform(X_train_titles_tfidf)
X_cv_titles_2000 = selector.transform(X_cv_titles_tfidf)
X_test_titles_2000 = selector.transform(X_test_titles_tfidf)
print(X_train_titles_2000.shape)
print(X_cv_titles_2000.shape)
print(X_test_titles_2000.shape)
```

```
(49041, 2000)
(24155, 2000)
(36052, 2000)
```

In [51]:

```
cols = selector.get_support(indices=True)
features = tfidf.get_feature_names()
i = 0
feats=[]
while(i < len(features)+1):
    if i in cols:
        feats.append(features[i])
    i+=1
len(feats)
```

Out[51]:

```
2000
```

In [52]:

```
print(len(feature_list_2))
feature_list_2.extend(feats)
print(len(feature_list_2))
```

```
2104
4104
```

In [53]:

```
tfidf_summary = TfidfVectorizer(min_df=10)
tfidf_summary.fit(X_train['project_resource_summary'].values) # fit has to happen only on train
data

# we use the fitted CountVectorizer to convert the text to vector
X_train_summary_tfidf = tfidf.transform(X_train['project_resource_summary'].values)
X_cv_summary_tfidf = tfidf.transform(X_cv['project_resource_summary'].values)
X_test_summary_tfidf = tfidf.transform(X_test['project_resource_summary'].values)


selector = SelectKBest(chi2, k = 2000 )
selector.fit(X_train_summary_tfidf,y_train)

X_train_summary_2000 = selector.transform(X_train_summary_tfidf)
```

```
X_cv_summary_2000 = selector.transform(X_cv_summary_tfidf)
X_test_summary_2000 = selector.transform(X_test_summary_tfidf)
print(X_train_summary_2000.shape)
print(X_cv_summary_2000.shape)
print(X_test_summary_2000.shape)
```

```
(49041, 2000)
(24155, 2000)
(36052, 2000)
```

In [54]:

```
cols = selector.get_support(indices=True)
features = tfidf.get_feature_names()
i = 0
feats=[]
while(i < len(features)+1):
    if i in cols:
        feats.append(features[i])
    i+=1
len(feats)
```

Out[54]:

```
2000
```

In [55]:

```
print(len(feature_list_2))
feature_list_2.extend(feats)
print(len(feature_list_2))
```

```
4104
6104
```

2.4.2 Applying Naive Bayes on TFIDF, SET 2

In [56]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr =
hstack((X_train_clean_cat_ohe,X_train_clean_subcat_ohe,X_train_teacher_ohe,X_train_state_ohe,X_trai
n_grade_ohe,X_train_price_std,X_train_projects_std,X_train_qty_std,X_train_essay_2000,X_train_title
s_2000,X_train_summary_2000)).tocsr()
X_cr =
hstack((X_cv_clean_cat_ohe,X_cv_clean_subcat_ohe,X_cv_teacher_ohe,X_cv_state_ohe,X_cv_grade_ohe,X_c
v_price_std,X_cv_projects_std,X_cv_qty_std,X_cv_essay_2000,X_cv_titles_2000,X_cv_summary_2000)).to
csr()
X_te = hstack((X_test_clean_cat_ohe,X_test_clean_subcat_ohe,X_test_teacher_ohe,X_test_state_ohe,X_
test_grade_ohe,X_test_price_std,X_test_projects_std,X_test_qty_std,X_test_essay_2000,X_test_titles_
2000,X_test_summary_2000)).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
(49041, 6104) (49041,)
(24155, 6104) (24155,)
(36052, 6104) (36052,)
```

In [57]:

```
len(feature_list_2)
```

Out[57]:

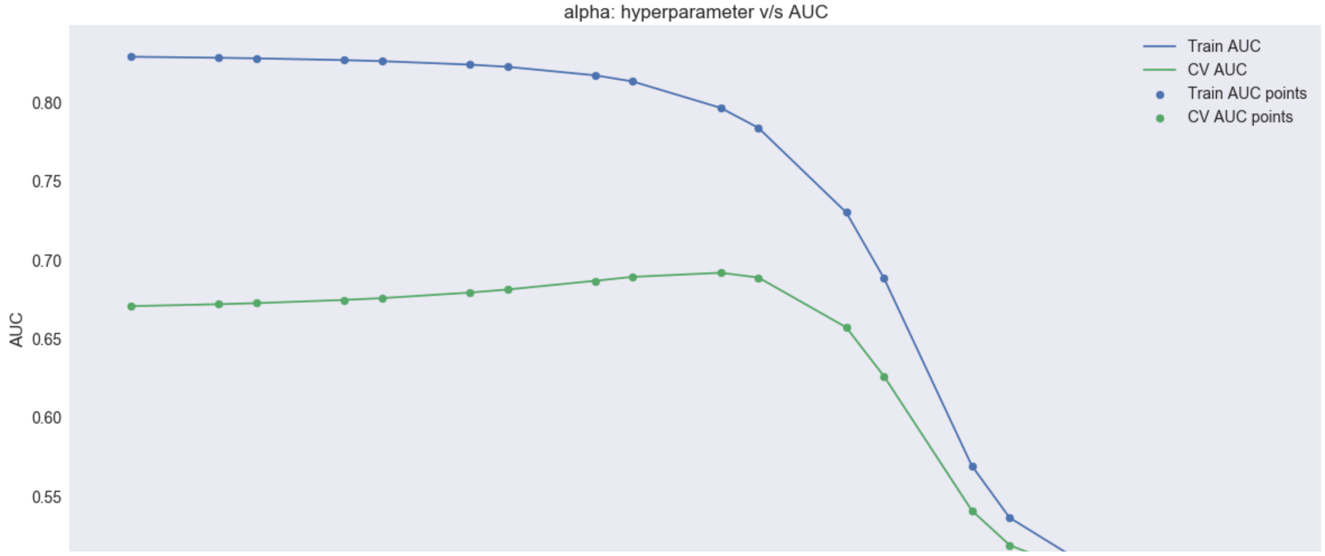
```
6104
```

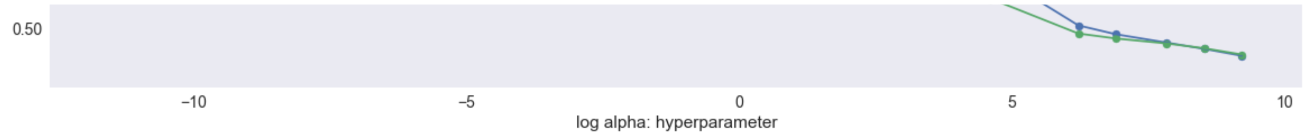
```
6104
```

In [58]:

```python
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 = [0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5, 10, 50, 100, 5
00, 1000, 2500, 5000, 10000]

for i in tqdm(alphas):
    nb_1 = MultinomialNB(alpha = i)
    nb_1.fit(X_tr, y_train)

    y_train_pred = batch_predict(nb_1, X_tr)
    y_cv_pred = batch_predict(nb_1, X_cr)


    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # 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)



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

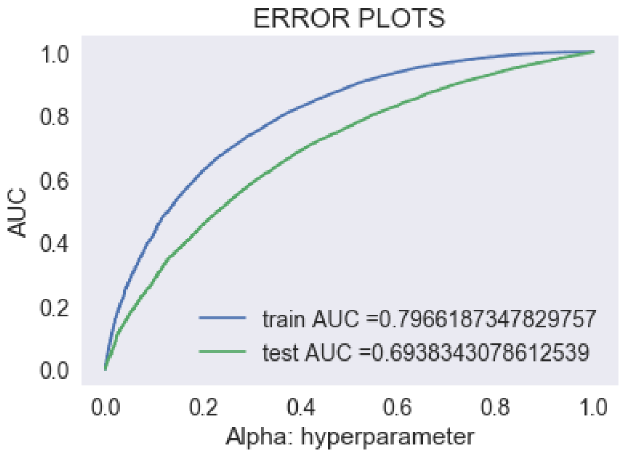
```
100%|██████████████████████████████████████████████████████████| 20/20
[00:03<00:00,  6.85it/s]
100%|████████████████████████████████████████████████████████| 20/20
[00:00<00:00, 20087.66it/s]
```





In [59]:

```
# 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


nb_1 = MultinomialNB(alpha=0.5)
nb_1.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

y_train_pred = batch_predict(nb_1, X_tr)
y_test_pred = batch_predict(nb_1, X_te)

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



2.4.2.2 Top 10 important features of negative class from SET 2

In [60]:

```
feature_importance_2 = pd.DataFrame({'features': feature_list_2,'prob_value': nb_1.coef_[0],})
```

In [61]:

```
#Least important features
feature_importance_2.sort_values(['prob_value'], ascending=[False]).head(20)
```

Out[61]:

	features	prob_value
100	Grades	-2.607522
101	Price	-2.846734

	features	prob_value
103	Quantity	-3.084628
41	mrs	-3.247387
4	literacy_language	-3.326778
102	num_prev_projects	-3.447688
5	math_science	-3.585428
42	ms	-3.644243
26	literacy	-3.765866
28	mathematics	-3.970763
27	literature_writing	-4.184058
49	ca	-4.571863
5313	need	-4.614028
2	health_sports	-4.641205
35	specialneeds	-4.699817
7	specialneeds	-4.699817
0	appliedlearning	-4.821637
9	appliedsciences	-4.936414
40	mr	-4.942600
24	health_wellness	-4.948825

2.4.2.1 Top 10 important features of positive class from SET 2

In [62]:

```
#Most important features
feature_importance_2.sort_values(['prob_value'], ascending=[True]).head(20)
```

Out[62]:

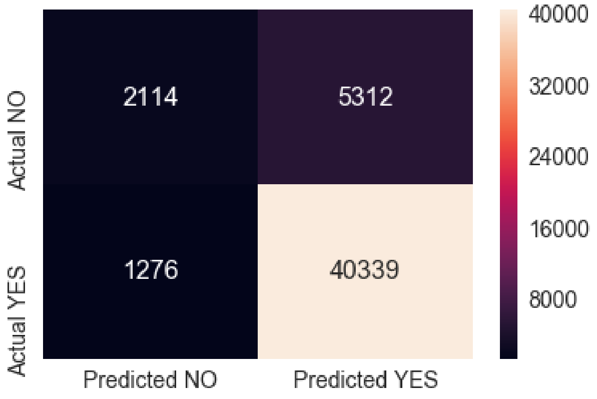
	features	prob_value
2508	conversations	-13.936897
4860	fort	-13.936897
4857	formally	-13.936897
2964	hartford	-13.936897
2981	hired	-13.936897
2984	histories	-13.936897
4843	flows	-13.936897
2990	hometown	-13.936897
2994	hospitality	-13.936897
2999	hunt	-13.936897
3004	identify	-13.936897
3005	ie	-13.936897
4867	frameworks	-13.936897
4828	firing	-13.936897
4813	figuring	-13.936897
3016	impacts	-13.936897
4807	fibers	-13.936897
4805	ffa	-13.936897
4804	fest	-13.936897

	features	prob_value
4801	fees	13.936897

train_confusion_matrix

In [64]:

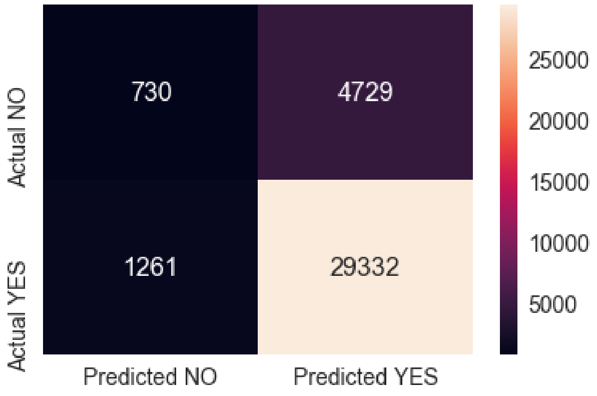
```
get_confusion_matrix(nb_1,X_tr,y_train)
```



test_confusion_matrix

In [65]:

```
get_confusion_matrix(nb_1,X_te,y_test)
```



3. Conclusions

In [71]:

```
#http://zetcode.com/python/prettytable/
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Vectorizer", "Hyperparameter", "AUC"]
x.add_row(["Bag of Words", 0.00001, 0.703])
x.add_row(["TFIDF", 0.5, 0.693])
print(x)
```

```
+--------------+----------------+-------+
|  Vectorizer  | Hyperparameter |  AUC  |
+--------------+----------------+-------+
| Bag of Words |     1e-05      | 0.703 |
|    TFIDF     |      0.5       | 0.693 |
+--------------+----------------+-------+
```

summary

1. Naive_bayes algorithm is very fast computation as compare to k_nearest_neighbors
2. The accuracy of the naive_bayes algorithm is better as compared to the k_nearest_neighbors