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`
`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*

Feature	Description
`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."
- __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
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

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]:

```
project_data.shape
```

Out[3]:

```
(109248, 17)
```

In [4]:

```
resource_data.shape
```

Out[4]:

```
(1541272, 4)
```

In [5]:

```
print("Number of data points in train data", project_data.shape)
print('-'*50)
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 [6]:

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

Out[6]:

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

In [7]:

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


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


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


project_data.head(2)
```

Out[7]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_s
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 [8]:

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

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

Out[8]:

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

1.2 preprocessing of `project_subject_categories`

In [9]:

```
print(project_data['project_subject_categories'].head(5))
```

```
55660          Math & Science
76127           Special Needs
51140    Literacy & Language
473         Applied Learning
41558    Literacy & Language
Name: project_subject_categories, dtype: object
```

In [10]:

```
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]))
```

In [11]:

```
print(project_data['clean_categories'].head(5))
```

```
55660          Math_Science
76127          SpecialNeeds
51140      Literacy_Language
473          AppliedLearning
41558      Literacy_Language
Name: clean_categories, dtype: object
```

1.3 preprocessing of `project_subject_subcategories`

In [12]:

```
print(project_data['project_subject_subcategories'].head(5))
```

```
55660    Applied Sciences, Health & Life Science
76127                              Special Needs
51140                                   Literacy
473                        Early Development
41558                                   Literacy
Name: project_subject_subcategories, dtype: object
```

In [13]:

```
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:
```

```
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]))
```

In [14]:

```
print(project_data['clean_subcategories'].head(5))
```

```
55660    AppliedSciences Health_LifeScience
76127                          SpecialNeeds
51140                              Literacy
473                        EarlyDevelopment
41558                              Literacy
Name: clean_subcategories, dtype: object
```

1.4 preprocessing of `school_state`

In [15]:

```
my_counter = Counter()
for word in project_data['school_state'].values:
    my_counter.update(word.split())

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

1.5 preprocessing of `project_grade_category`

In [16]:

```
preproc = []
# tqdm is for printing the status bar
for sent in project_data['project_grade_category']:
    sent = sent.replace('Grades ', 'Grades_')
    preproc.append(sent)
project_data['project_grade_category']=preproc
```

In [17]:

```
my_counter = Counter()
for word in project_data['project_grade_category'].values:
    my_counter.update(word.split())

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

1.6 preprocessing of `teacher_prefix`

In [18]:

```
project_data['teacher_prefix']=project_data['teacher_prefix'].fillna('')
my_counter = Counter()
for word in project_data['teacher_prefix'].values:
    my_counter.update(word.split())

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

1.3 Text preprocessing

In [19]:

```
# 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 [20]:

```
# 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 [21]:

```
# 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 [22]:

```
# 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 [01:08<00:00, 1603.01it/s]
```

In [23]:

```
# after preprocesing
preprocessed_essays[2000]
```

Out[23]:

```
'creativity intelligence fun albert einstein elementary library greenville elementary anything qui
et hushed space place collaboration research place incorporating technology place innovation place
creating school serves 350 third fourth graders primarily live rural poverty stricken areas
community title school approximately 85 receive free reduced lunch inquisitive creative eager
learn love visiting library check books hear stories create digital stories use computer lab learn
ing fun want build library makerspace activities revolving around art literacy provide engaging ha
nds activities want begin makerspace fridays school recently received 1000 grant books arts
integrated makerspace received titles origami everyone make stuff ducktape cool engineering activi
ties girls need supplies correlate new informational texts adding art craft supplies students able
design create masterpieces related coursework example studying native americans students use looms
yarn recreate navajo pueblo weaving weaving also integrated literacy greek mythology story arachne
creating art perler beads many possibilities students design animals studying characteristics use
symmetry patterning create one kind originals origami reinforces geometry thinking skills
fractions problem solving fun science students need able apply read learn read book apply reading
hands art activity actually create product crucial skill real world creating designing
masterpieces using many critical thinking skills students become analytical thinkers'
```

In [24]:

```
project_data['essay']=preprocessed_essays
```

1.4 Preprocessing of `project_title`

In [25]:

```
# similarly you can preprocess the titles also
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, 36071.50it/s]
```

In [26]:

```
project_data['project_title']=preprocessed_titles
```

1.5 Preparing data for models

In [27]:

```
project_data.columns
```

Out[27]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'Date', 'project_grade_category', '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',
       'clean_categories', 'clean_subcategories', 'essay'],
      dtype='object')
```

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

Assignment 4: Naive Bayes

1. **Apply Multinomial NaiveBayes on these feature sets**
    - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_eassay (BOW)
    - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)
2. **The hyper paramter tuning(find best Alpha)**
    - Find the best hyper parameter which will give the maximum AUC 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

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

2. Naive Bayes

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

In [28]:

```
y = project_data['project_is_approved']
print(y.shape)
```

```
(109248,)
```

In [29]:

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

In [30]:

```
X=project_data
print(X.shape)
```

```
(109248, 17)
```

In [31]:

```
#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=0.20, stratify=y)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.20, stratify=y_train)
```

2.3 Make Data Model Ready: encoding numerical and categorical features

Vectorizing Numerical features

In [32]:

```
features=[]
```

In [33]:

```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
```

In [34]:

```
price_data.head(5)
```

Out[34]:

	id	quantity	price
0	p000001	7	459.56
1	p000002	21	515.89
2	p000003	4	298.97
3	p000004	98	1113.69

	id	quantity	price
4	p000005	8	485.99

In [35]:

```
X_train=pd.merge(X_train,price_data,on='id',how='left')
X_test=pd.merge(X_test,price_data,on='id',how='left')
X_cv=pd.merge(X_cv,price_data,on='id',how='left')
```

In [36]:

```
X_train=X_train.fillna(0)
X_cv=X_cv.fillna(0)
X_test=X_test.fillna(0)
```

Normalizing the numerical features: Price

In [37]:

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

X_train_price_norm = normalizer.transform(X_train['price'].values.reshape(-1,1))
X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(-1,1))
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(-1,1))
features += ['price']
print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)
print(X_cv_price_norm.shape, y_cv.shape)
print(X_test_price_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(69918, 1) (69918,)
(17480, 1) (17480,)
(21850, 1) (21850,)
====================================================================================================
```

Normalizing the numerical features: Number of previously posted projects

In [38]:

```
normalizer = Normalizer()
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
X_train_project_norm = normalizer.transform(X_train['teacher_number_of_previously_posted_projects'
].values.reshape(-1,1))
X_cv_project_norm = normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].valu
es.reshape(-1,1))
X_test_project_norm = normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].
values.reshape(-1,1))
features += ['teacher_number_of_previously_posted_projects']
print("After vectorizations")
print(X_train_project_norm.shape, y_train.shape)
print(X_cv_project_norm.shape, y_cv.shape)
print(X_test_project_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(69918, 1) (69918,)
(17480, 1) (17480,)
(21850, 1) (21850,)
```

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

Vectorizing Categorical features

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

Vectorizing Categorical features: project grade category

In [39]:

```
from sklearn.feature_extraction.text import CountVectorizer
```

In [40]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_grade_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)
features += vectorizer.get_feature_names()
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)
```

```
After vectorizations
(69918, 4) (69918,)
(17480, 4) (17480,)
(21850, 4) (21850,)
['Grades_PreK-2', 'Grades_3-5', 'Grades_9-12', 'Grades_6-8']
====================================================================================================
```

Vectorizing Categorical features: teacher prefix

In [41]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_teacher_dict.keys()), lowercase=False, binary=True)
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)
features += vectorizer.get_feature_names()
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)
```

```
After vectorizations
(69918, 5) (69918,)
(17480, 5) (17480,)
(21850, 5) (21850,)
```

```
['Teacher', 'Dr.', 'Ms.', 'Mr.', 'Mrs.']
====================================================================================================
```

Vectorizing Categorical features: school state

In [42]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_state_dict.keys()), lowercase=False, binary=Tr
ue)
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)
features += vectorizer.get_feature_names()
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)
```

```
After vectorizations
(69918, 51) (69918,)
(17480, 51) (17480,)
(21850, 51) (21850,)
['WI', 'KS', 'MI', 'MN', 'VT', 'CO', 'NV', 'TN', 'MS', 'TX', 'VA', 'NM', 'NY', 'ME', 'MD', 'AL', 'I
D', 'SD', 'GA', 'OK', 'WY', 'UT', 'IL', 'CT', 'NC', 'IN', 'MO', 'OH', 'KY', 'WV', 'AZ', 'ND', 'DE',
'HI', 'AK', 'AR', 'IA', 'CA', 'FL', 'PA', 'LA', 'OR', 'NJ', 'SC', 'MA', 'MT', 'NE', 'WA', 'RI', 'DC
', 'NH']
====================================================================================================
```

Vectorizing Categorical features: clean categories

In [43]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True
)
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_cat_ohe = vectorizer.transform(X_train['clean_categories'].values)
X_cv_cat_ohe = vectorizer.transform(X_cv['clean_categories'].values)
X_test_cat_ohe = vectorizer.transform(X_test['clean_categories'].values)
features += vectorizer.get_feature_names()
print("After vectorizations")
print(X_train_cat_ohe.shape, y_train.shape)
print(X_cv_cat_ohe.shape, y_cv.shape)
print(X_test_cat_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(69918, 9) (69918,)
(17480, 9) (17480,)
(21850, 9) (21850,)
['History_Civics', 'SpecialNeeds', 'AppliedLearning', 'Care_Hunger', 'Music_Arts',
'Health_Sports', 'Warmth', 'Math_Science', 'Literacy_Language']
====================================================================================================
```

Vectorizing Categorical features: clean subcategories

In [44]:

```
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=
True)
```

```
True)
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_sub_ohe = vectorizer.transform(X_train['clean_subcategories'].values)
X_cv_sub_ohe = vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_sub_ohe = vectorizer.transform(X_test['clean_subcategories'].values)
features += vectorizer.get_feature_names()
print("After vectorizations")
print(X_train_sub_ohe.shape, y_train.shape)
print(X_cv_sub_ohe.shape, y_cv.shape)
print(X_test_sub_ohe.shape, y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(69918, 30) (69918,)
(17480, 30) (17480,)
(21850, 30) (21850,)
['AppliedSciences', 'Literacy', 'Health_LifeScience', 'CommunityService', 'Literature_Writing', 'M
athematics', 'SocialSciences', 'PerformingArts', 'Other', 'Extracurricular', 'ESL',
'SpecialNeeds', 'Health_Wellness', 'TeamSports', 'FinancialLiteracy', 'Music',
'History_Geography', 'College_CareerPrep', 'VisualArts', 'Gym_Fitness', 'Economics',
'ForeignLanguages', 'EarlyDevelopment', 'Warmth', 'ParentInvolvement', 'EnvironmentalScience',
'Care_Hunger', 'NutritionEducation', 'CharacterEducation', 'Civics_Government']
====================================================================================================
```

2.2 Make Data Model Ready: encoding eassay, and project_title

In [45]:

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

```
(69918, 19) (69918,)
(17480, 19) (17480,)
(21850, 19) (21850,)
====================================================================================================
```

Encoding of Text Data

In [46]:

```
from sklearn.feature_extraction.text import CountVectorizer
```

In [47]:

```
features_bow = features
features_tfidf = features
```

BOW of Essay

In [48]:

```
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
```

In [49]:

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

Out[49]:

```
CountVectorizer(analyzer='word', binary=False, decode_error='strict',
CountVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=5000, min_df=10,
        ngram_range=(1, 4), preprocessor=None, stop_words=None,
        strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
        tokenizer=None, vocabulary=None)
```

In [50]:

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

In [51]:

```
X_cv_essay_bow = vectorizer.transform(X_cv['essay'].values)
```

In [52]:

```
X_test_essay_bow = vectorizer.transform(X_test['essay'].values)
```

In [53]:

```
features_bow += vectorizer.get_feature_names()
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)
```

```
After vectorizations
(69918, 5000) (69918,)
(17480, 5000) (17480,)
(21850, 5000) (21850,)
====================================================================================================
```

BOW of Title

In [54]:

```
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
```

In [55]:

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

Out[55]:

```
CountVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=5000, min_df=10,
        ngram_range=(1, 4), preprocessor=None, stop_words=None,
        strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
        tokenizer=None, vocabulary=None)
```

In [56]:

```
# we use the fitted CountVectorizer to convert the text to vector
X_train_title_bow = vectorizer.transform(X_train['project_title'].values)
```

In [57]:

```
X_cv_title_bow = vectorizer.transform(X_cv['project_title'].values)
```

In [58]:

```
X_test_title_bow = vectorizer.transform(X_test['project_title'].values)
X_test_title_bow = vectorizer.transform(X_test['project_title'].values)
```

In [59]:

```
features_bow += vectorizer.get_feature_names()
print("After vectorizations")
print(X_train_title_bow.shape, y_train.shape)
print(X_cv_title_bow.shape, y_cv.shape)
print(X_test_title_bow.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(69918, 5000) (69918,)
(17480, 5000) (17480,)
(21850, 5000) (21850,)
====================================================================================================
```

TFIDF of Essay

In [60]:

```
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
```

In [61]:

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

Out[61]:

```
TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=5000, min_df=10,
        ngram_range=(1, 4), norm='l2', preprocessor=None, smooth_idf=True,
        stop_words=None, strip_accents=None, sublinear_tf=False,
        token_pattern='(?u)\\b\\w\\w+\\b', tokenizer=None, use_idf=True,
        vocabulary=None)
```

In [62]:

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

In [63]:

```
X_cv_essay_tfidf = vectorizer.transform(X_cv['essay'].values)
```

In [64]:

```
X_test_essay_tfidf = vectorizer.transform(X_test['essay'].values)
```

In [65]:

```
features_tfidf += vectorizer.get_feature_names()
print("After vectorizations")
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("="*100)
```

```
After vectorizations
(69918, 5000) (69918,)
(17480, 5000) (17480,)
(21850, 5000) (21850,)
====================================================================================================
```

TFIDF of Title

In [66]:

```
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
```

In [67]:

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

Out[67]:

```
TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=5000, min_df=10,
        ngram_range=(1, 4), norm='l2', preprocessor=None, smooth_idf=True,
        stop_words=None, strip_accents=None, sublinear_tf=False,
        token_pattern='(?u)\\b\\w\\w+\\b', tokenizer=None, use_idf=True,
        vocabulary=None)
```

In [68]:

```
# we use the fitted CountVectorizer to convert the text to vector
X_train_title_tfidf = vectorizer.transform(X_train['project_title'].values)
```

In [69]:

```
X_cv_title_tfidf = vectorizer.transform(X_cv['project_title'].values)
```

In [70]:

```
X_test_title_tfidf = vectorizer.transform(X_test['project_title'].values)
```

In [71]:

```
features_tfidf += vectorizer.get_feature_names()
print("After vectorizations")
print(X_train_title_tfidf.shape, y_train.shape)
print(X_cv_title_tfidf.shape, y_cv.shape)
print(X_test_title_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(69918, 5000) (69918,)
(17480, 5000) (17480,)
(21850, 5000) (21850,)
====================================================================================================
```

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

2.4.1 Applying Naive Bayes on BOW, SET 1

In [162]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr = hstack((X_train_essay_bow,X_train_title_bow, X_train_state_ohe, X_train_teacher_ohe,
X_train_grade_ohe,X_train_cat_ohe,X_train_sub_ohe, X_train_price_norm,X_train_project_norm)).tocsr
()
```

```
X_cr = hstack((X_cv_essay_bow,X_cv_title_bow, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe,X_cv_cat_ohe,X_cv_sub_ohe, X_cv_price_norm,X_cv_project_norm)).tocsr()
X_te = hstack((X_test_essay_bow,X_test_title_bow, X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe,X_test_cat_ohe,X_test_sub_ohe, X_test_price_norm,X_test_project_norm)).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
(69918, 10101) (69918,)
(17480, 10101) (17480,)
(21850, 10101) (21850,)
====================================================================================================
```

In [163]:

```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive 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
```

Hyperparameter Tuning: Simple for loop (if you are having memory limitations use this)

In [164]:

```
import matplotlib.pyplot as plt
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
import math
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or non-thresholded measure of
decisions (as returned by “decision_function” on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
log_alphas=[]
alphas = [1e-4, 1e-3, 1e-2, 1e-1, 1.0, 1e1, 1e2, 1e3, 1e4]

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

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

    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive 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))
```

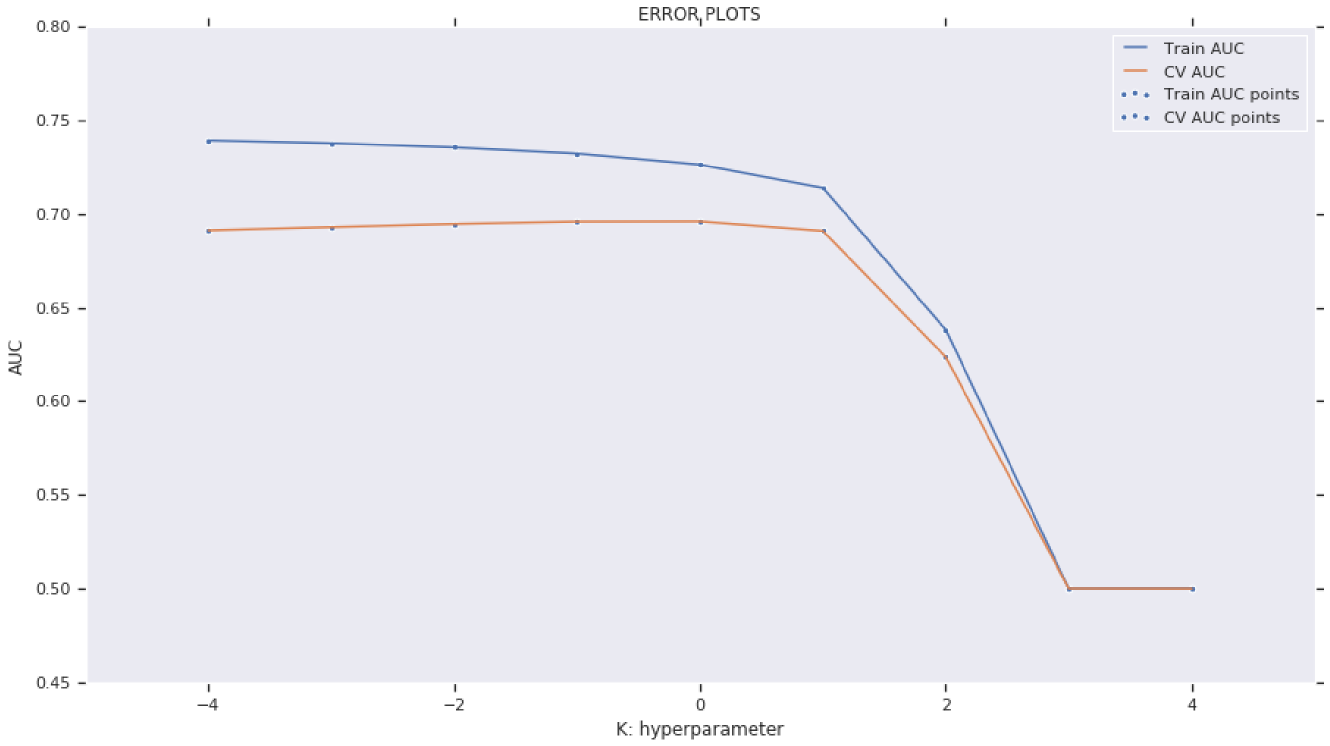
```
for a in tqdm(alphas):
    b = math.log10(a)
    log_alphas.append(b)
print(log_alphas)
```

```
100%|██████████| 9/9 [00:02<00:00,  4.09it/s]
100%|██████████| 9/9 [00:00<00:00, 26772.15it/s]
```

```
[-4.0, -3.0, -2.0, -1.0, 0.0, 1.0, 2.0, 3.0, 4.0]
```

In [165]:

```
plt.figure(figsize=(15,8))
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.autoscale(enable = True)
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")

plt.grid()
plt.show()
```



Here we can see that the smaller values of log_alphas seem to work very well on train data but not on cross validation data. The values close to log_alphas=-5 works pretty well both on Train data and Cross Validation data. So the value of alpha is antilog or exponential of log_alphas.

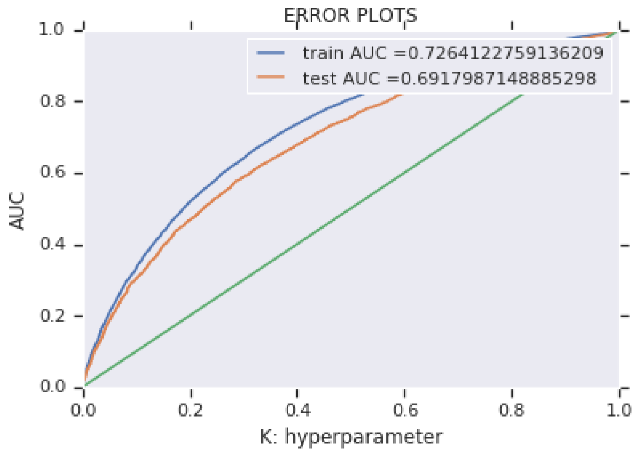
In [166]:

```
best_alpha=1
```

Train The Model

In [167]:

```
from sklearn.metrics import roc_curve, auc


neigh = MultinomialNB(alpha=best_alpha)
neigh.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(neigh, X_tr)
y_test_pred = batch_predict(neigh, 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)

x=[0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]

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.plot(x,x)
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



Confusion Matrix

In [168]:

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

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

    # (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
```

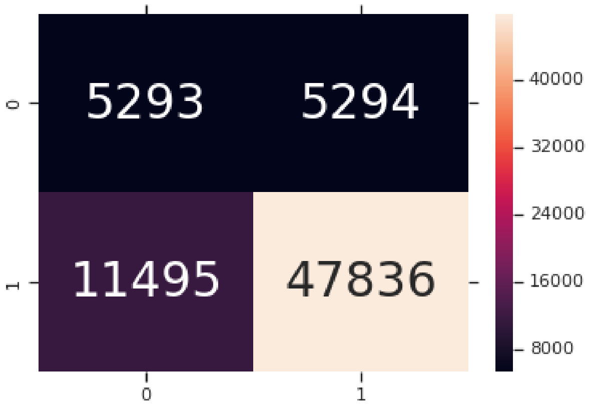
In [169]:

```
print("Train confusion matrix")
conf_matr_df_train_2=pd.DataFrame(confusion_matrix(y_train,predict(y_train_pred,tr_thresholds,train_fpr,train_fpr)),range(2),range(2))
sns.set(font_scale=1)#for label size
sns.heatmap(conf_matr_df_train_2,annot=True,annot_kws={"size":30},fmt='g')
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999776954132 for threshold 0.181
```

Out[169]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f31a3dff0f0>
```



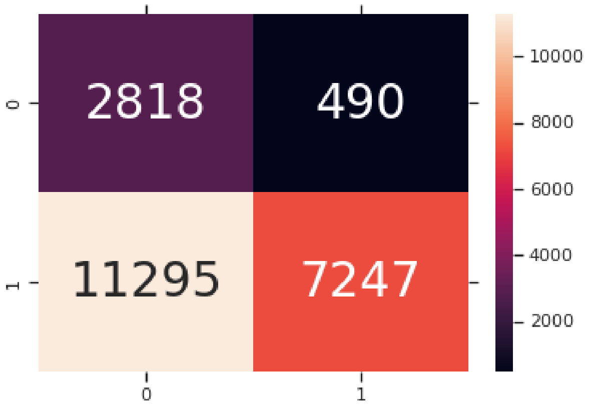
In [170]:

```python
print("Test confusion matrix")
conf_matr_df_train_2=pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred,tr_thresholds,test_fpr,test_fpr)),range(2),range(2))
sns.set(font_scale=1)#for label size
sns.heatmap(conf_matr_df_train_2,annot=True,annot_kws={"size":30},fmt='g')
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.24999990861624524 for threshold 0.999
```

Out[170]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f31800dc860>
```



2.4.1.1 Top 10 important features of positive class from SET 1

In [172]:

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

pos_imp = neigh.feature_log_prob_[1]
imp_feature = zip(pos_imp, features_bow)

# sort a list of tuples, https://stackoverflow.com/a/10695161
imp_feature = sorted(imp_feature, reverse = True, key = lambda x: x[0])
print('Top 10 most important features of positive class are: ')

pt = PrettyTable()
pt.field_names = ['Priority', 'Feature', 'Log probability']
```

```
for i in range(1,11):
    pt.add_row([i, imp_feature[i-1][1], imp_feature[i-1][0]])

print(pt)
```

```
Top 10 most important features of positive class are: 
+----------+-------------------+---------------------+
| Priority |      Feature      |   Log probability   |
+----------+-------------------+---------------------+
|    1     |      spending     |  -3.159794292332453 |
|    2     | reluctant readers |   -4.3031436506819  |
|    3     |        knew       |  -4.665804056859782 |
|    4     |      centered     |  -4.692508842051618 |
|    5     |        name       | -4.9549916678258565 |
|    6     |     ipad minis    |  -5.008426487274717 |
|    7     |  graders students |  -5.036728791571706 |
|    8     |        zone       |  -5.148689536637908 |
|    9     |     love learn    | -5.1765811525852214 |
|    10    |       meets       |  -5.194487485397294 |
+----------+-------------------+---------------------+
```

2.4.1.2 Top 10 important features of negative class from SET 1

In [173]:

```
neg_imp = neigh.feature_log_prob_[0]
imp_feature = zip(neg_imp, features_bow)

# sort a list of tuples, https://stackoverflow.com/a/10695161
imp_feature = sorted(imp_feature, reverse = True, key = lambda x: x[0])
print('Top 10 most important features of negative class are: ')

pt = PrettyTable()
pt.field_names = ['Priority', 'Feature', 'Log probability']

for i in range(1,11):
    pt.add_row([i, imp_feature[i-1][1], imp_feature[i-1][0]])

print(pt)
```

```
Top 10 most important features of negative class are: 
+----------+-------------------+---------------------+
| Priority |      Feature      |   Log probability   |
+----------+-------------------+---------------------+
|    1     |      spending     | -3.1786015242163206 |
|    2     | reluctant readers | -4.2681987290849595 |
|    3     |        knew       |  -4.584850607041833 |
|    4     |      centered     |  -4.740506886099583 |
|    5     |        name       |  -4.931142407326135 |
|    6     |     ipad minis    |  -4.955280814419675 |
|    7     |  graders students |  -4.98783642160525  |
|    8     |        zone       |  -5.100105395095168 |
|    9     |       meets       |  -5.148480080797405 |
|    10    |     love learn    |  -5.166383244383262 |
+----------+-------------------+---------------------+
```

2.4.2 Applying Naive Bayes on TFIDF, SET 2

In [174]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr = hstack((X_train_essay_tfidf,X_train_title_tfidf, X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe,X_train_cat_ohe,X_train_sub_ohe, X_train_price_norm,X_train_project_norm)).tocsr()
X_cr = hstack((X_cv_essay_tfidf,X_cv_title_tfidf, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe,
X_cv_cat_ohe,X_cv_sub_ohe, X_cv_price_norm,X_cv_project_norm)).tocsr()
X_te = hstack((X_test_essay_tfidf,X_test_title_tfidf, X_test_state_ohe, X_test_teacher_ohe,
X_test_grade_ohe,X_test_cat_ohe,X_test_sub_ohe, X_test_price_norm,X_test_project_norm)).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
(69918, 10101) (69918,)
(17480, 10101) (17480,)
(21850, 10101) (21850,)
====================================================================================================
```

In [175]:

```
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
```

Hyperparameter Tuning: Simple for loop (if you are having memory limitations use this)

In [176]:

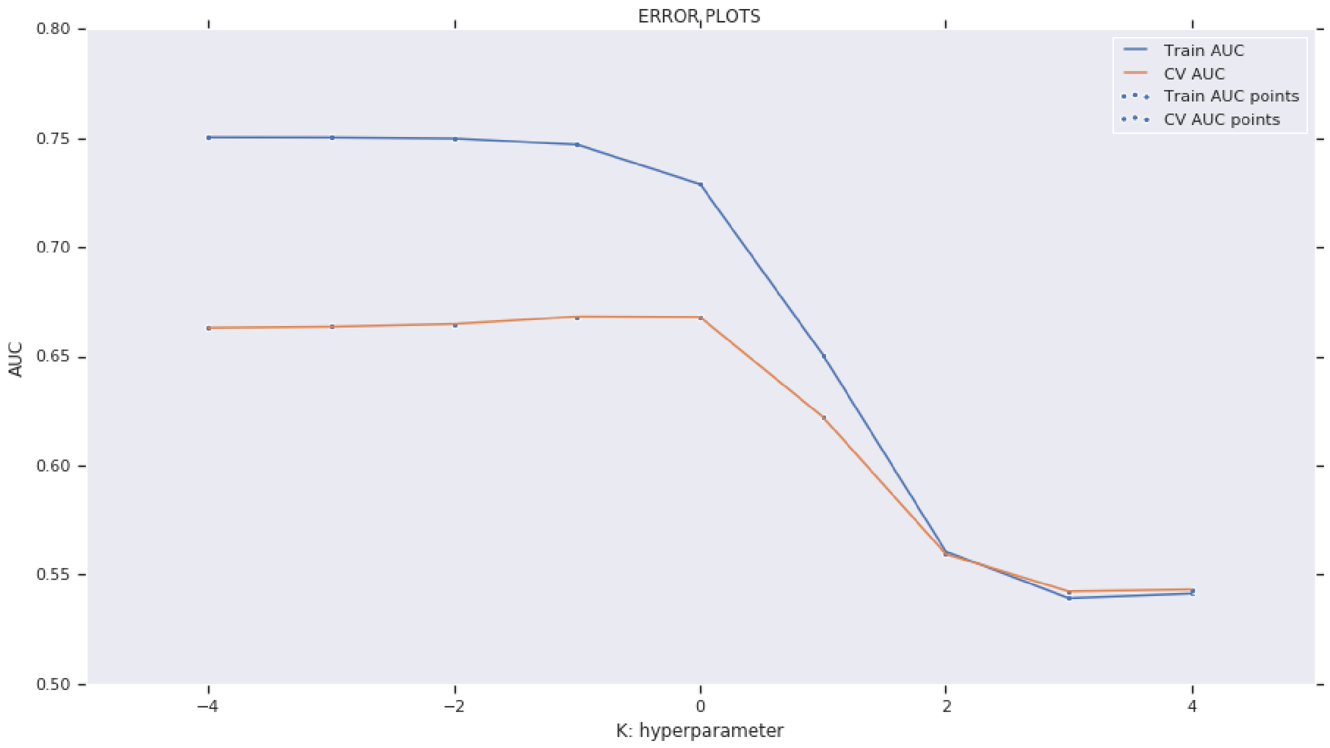
```
import matplotlib.pyplot as plt
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
import math
"""
y_true : array, shape = [n_samples] or [n_samples, n_classes]
True binary labels or binary label indicators.

y_score : array, shape = [n_samples] or [n_samples, n_classes]
Target scores, can either be probability estimates of the positive class, confidence values, or no
n-thresholded measure of
decisions (as returned by “decision_function” on some classifiers).
For binary y_true, y_score is supposed to be the score of the class with greater label.

"""

train_auc = []
cv_auc = []
log_alphas=[]
alphas=[1e-4, 1e-3, 1e-2, 1e-1, 1.0, 1e1, 1e2, 1e3, 1e4]

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

    y_train_pred = batch_predict(neigh, X_tr)
    y_cv_pred = batch_predict(neigh, 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.log10(a)
    log_alphas.append(b)
print(log_alphas)
```

```
100%|██████████| 9/9 [00:02<00:00,  4.05it/s]
100%|██████████| 9/9 [00:00<00:00, 26944.14it/s]
```

```
[-4.0, -3.0, -2.0, -1.0, 0.0, 1.0, 2.0, 3.0, 4.0]
```

In [177]:

```
plt.figure(figsize=(15,8))
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("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")

plt.grid()
plt.show()
```



In [178]:

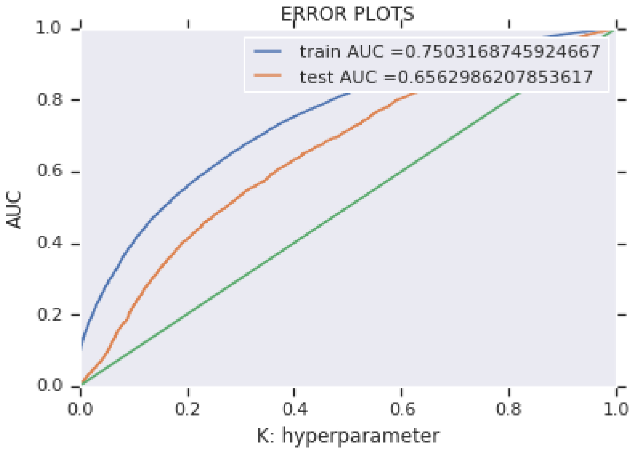
```
best_alpha=0.001
```

Train The Model

In [179]:

```
from sklearn.metrics import roc_curve, auc


neigh = MultinomialNB(alpha=best_alpha)
neigh.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(neigh, X_tr)
y_test_pred = batch_predict(neigh, 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)

x=[0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]

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.plot(x,x)
plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



Confusion Matrix

In [180]:

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

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

    # (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
```

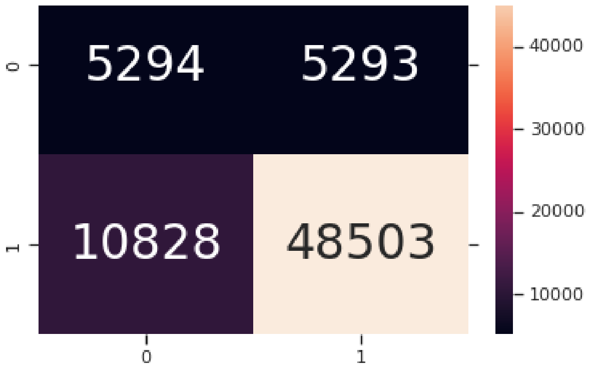
In [181]:

```
print("Train confusion matrix")
conf_matr_df_train_2=pd.DataFrame(confusion_matrix(y_train,predict(y_train_pred,tr_thresholds,trai
n_fpr,train_fpr)),range(2),range(2))
sns.set(font_scale=1)#for label size
sns.heatmap(conf_matr_df_train_2,annot=True,annot_kws={"size":30},fmt='g')
```

```
Train confusion matrix
the maximum value of tpr*(1-fpr) 0.24999999776954132 for threshold 0.761
```

Out[181]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f319e00deb8>
```



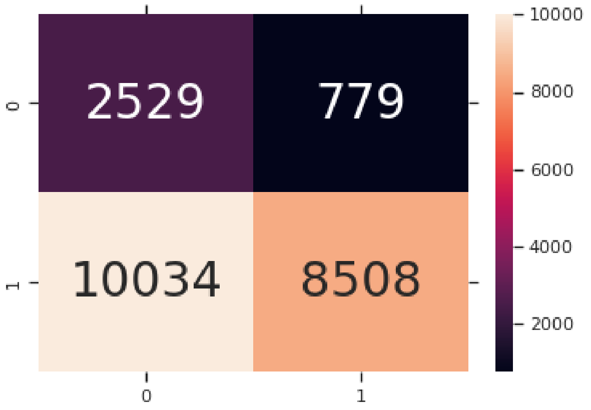
In [182]:

```
print("Test confusion matrix")
conf_matr_df_train_2=pd.DataFrame(confusion_matrix(y_test,predict(y_test_pred,tr_thresholds,test_fp
r,test_fpr)),range(2),range(2))
sns.set(font_scale=1)#for label size
sns.heatmap(conf_matr_df_train_2,annot=True,annot_kws={"size":30},fmt='g')
```

```
Test confusion matrix
the maximum value of tpr*(1-fpr) 0.25 for threshold 0.911
```

Out[182]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f31c8953710>
```



2.4.2.1 Top 10 important features of positive class from SET 2

In [183]:

```
# Please write all the code with proper documentation
```

In [184]:

```
pos_imp = neigh.feature_log_prob_[1]
imp_feature = zip(pos_imp, features_tfidf)

# sort a list of tuples, https://stackoverflow.com/a/10695161
imp_feature = sorted(imp_feature, reverse = True, key = lambda x: x[0])
print('Top 10 most important features of positive class are: ')

pt = PrettyTable()
pt.field_names = ['Priority', 'Feature', 'Log probability']

for i in range(1,11):
    pt.add_row([i, imp_feature[i-1][1], imp_feature[i-1][0]])

print(pt)
```

```
Top 10 most important features of positive class are:
+----------+--------------+--------------------+
| Priority |    Feature    |   Log probability   |
+----------+---------------+---------------------+
|    1     |      zone     | -2.8726063386468255 |
|    2     |      zoom     | -3.1801078870641923 |
|    3     |  you help us  | -3.5898917516174187 |
|    4     |    you help   |  -3.850210306331684 |
|    5     |   you learn   |  -4.022573095995963 |
|    6     |    you see    |  -4.22638325765201  |
|    7     | you read more |  -4.453646734053821 |
|    8     |    year old   |  -4.825128343490345 |
|    9     | you hear what |  -4.911664763167254 |
|    10    |    spending   |  -4.948264179114741 |
+----------+---------------+---------------------+
```

2.4.2.2 Top 10 important features of negative class from SET 2

In [185]:

```
# Please write all the code with proper documentation
```

In [186]:

```
neg_imp = neigh.feature_log_prob_[0]
imp_feature = zip(neg_imp, features_tfidf)

# sort a list of tuples, https://stackoverflow.com/a/10695161
imp_feature = sorted(imp_feature, reverse = True, key = lambda x: x[0])
print('Top 10 most important features of negative class are: ')

pt = PrettyTable()
pt.field_names = ['Priority', 'Feature', 'Log probability']

for i in range(1,11):
    pt.add_row([i, imp_feature[i-1][1], imp_feature[i-1][0]])

print(pt)
```

```
Top 10 most important features of negative class are:
+----------+---------------+---------------------+
| Priority |    Feature    |   Log probability   |
+----------+---------------+---------------------+
|    1     |      zone     |  -2.855828606215624 |
|    2     |      zoom     | -3.2455497399507927 |
|    3     |  you help us  | -3.7151628457941364 |
|    4     |    you help   | -3.7760942148965864 |
|    5     |    you see    |  -4.19696892870236  |
|    6     |   you learn   |  -4.205674094772173 |
|    7     | you read more |  -4.524810982437492 |
|    8     |    you hear   |  -4.84665371873306  |
|    9     | young authors |  -4.84665371873306  |
|    10    | you hear what |  -4.875417519503098 |
+----------+---------------+---------------------+
```

3. Conclusions

In [187]:

```
# http://zetcode.com/python/prettytable/

from prettytable import PrettyTable
#If you get a ModuleNotFoundError error , install prettytable using: pip3 install prettytable
x=PrettyTable()
x.field_names=["Vectorizer","Model","Hyper Parameter","AUC"]
x.add_row(["BOW","Naive Bayes",1,0.69])
x.add_row(["TFIDF","Naive Bayes",0.001,0.66])
print(x)
```

```
+------------+-------------+-----------------+------+
| Vectorizer |    Model    | Hyper Parameter | AUC  |
+------------+-------------+-----------------+------+
|    BOW     | Naive Bayes |        1        | 0.69 |
|   TFIDF    | Naive Bayes |      0.001      | 0.66 |
+------------+-------------+-----------------+------+
```

```
In [ ]:
```