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*
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]:
```
%config IPCompleter.greedy=True
%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
from sklearn.metrics import confusion_matrix
```

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/49702
492/4084039
project_data['Date'] = pd.to_datetime(project_data['project_submitted_datetim
e'])
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/13148611/4
084039
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_sub
categories', '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_
86221	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA
18308	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT

In [4]:

```
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' 's
chool_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']
```

In [5]:
```
# 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 [6]:
```
# 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', 'it
self', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 't
hat', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have',
'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and','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', 'a
ll', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'tha
n', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "shoul
d've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn',
"didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'm
a', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shoul
dn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

1.2 preprocessing of `project_subject_categories`

In [7]:

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

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-fr
om-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-strin
g-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 & Scienc
e", "Warmth", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on
space "Math & Science"=> "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 ''(emp
ty) ex:"Math & Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the tra
iling 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 [8]:
```
sorted_cat_dict
```

Out[8]:
```
{'Warmth': 1388,
 'Care_Hunger': 1388,
 'History_Civics': 5914,
 'Music_Arts': 10293,
 'AppliedLearning': 12135,
 'SpecialNeeds': 13642,
 'Health_Sports': 14223,
 'Math_Science': 41421,
 'Literacy_Language': 52239}
```

1.3 preprocessing of `project_subject_subcategories`

In [9]:
```
sub_catogories = list(project_data['project_subject_subcategories'].values)
# remove special characters from list of strings python: https://stackoverflo
w.com/a/47301924/4084039

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-fr
om-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-strin
g-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 & Scienc
e", "Warmth", "Care & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on
space "Math & Science"=> "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 ''(emp
ty) ex:"Math & Science"=>"Math&Science"
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the tra
iling 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/2289859
5/4084039
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 [10]:
```
sorted_sub_cat_dict
```

Out[10]:
```
{'Economics': 269,
 'CommunityService': 441,
 'FinancialLiteracy': 568,
 'ParentInvolvement': 677,
 'Extracurricular': 810,
 'Civics_Government': 815,
 'ForeignLanguages': 890,
 'NutritionEducation': 1355,
 'Warmth': 1388,
 'Care_Hunger': 1388,
 'SocialSciences': 1920,
 'PerformingArts': 1961,
 'CharacterEducation': 2065,
 'TeamSports': 2192,
 'Other': 2372,
 'College_CareerPrep': 2568,
 'Music': 3145,
 'History_Geography': 3171,
 'Health_LifeScience': 4235,
 'EarlyDevelopment': 4254,
 'ESL': 4367,
 'Gym_Fitness': 4509,
 'EnvironmentalScience': 5591,
 'VisualArts': 6278,
```

```
'Health_Wellness': 10234,
'AppliedSciences': 10816,
'SpecialNeeds': 13642,
'Literature_Writing': 22179,
'Mathematics': 28074,
'Literacy': 33700}
```

1.3 Text preprocessing

In [11]:
```
# merge two column text dataframe:
project_data["essay"] = project_data["project_essay_1"].map(str) +\
                        project_data["project_essay_2"].map(str) + \
                        project_data["project_essay_3"].map(str) + \
                        project_data["project_essay_4"].map(str)
```

In [12]:
```
project_data.head(2)
```

Out[12]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_
86221	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA
18308	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT

In [13]:
```
# printing some random reviews
print(project_data['essay'].values[0])
print("="*50)
```

```
I have been fortunate enough to use the Fairy Tale STEM kits in my classroom
as well as the STEM journals, which my students really enjoyed.  I would lov
```

```
e to implement more of the Lakeshore STEM kits in my classroom for the next
school year as they provide excellent and engaging STEM lessons.My students
come from a variety of backgrounds, including language and socioeconomic sta
tus.  Many of them don't have a lot of experience in science and engineering
and these kits give me the materials to provide these exciting opportunities
for my students.Each month I try to do several science or STEM/STEAM project
s.  I would use the kits and robot to help guide my science instruction in e
ngaging and meaningful ways.  I can adapt the kits to my current language ar
ts pacing guide where we already teach some of the material in the kits like
tall tales (Paul Bunyan) or Johnny Appleseed.  The following units will be t
aught in the next school year where I will implement these kits: magnets, mo
tion, sink vs. float, robots.  I often get to these units and don't know If
I am teaching the right way or using the right materials.    The kits will g
ive me additional ideas, strategies, and lessons to prepare my students in s
cience.It is challenging to develop high quality science activities.  These
kits give me the materials I need to provide my students with science activi
ties that will go along with the curriculum in my classroom.  Although I hav
e some things (like magnets) in my classroom, I don't know how to use them e
ffectively.  The kits will provide me with the right amount of materials and
show me how to use them in an appropriate way.
==================================================
```

In [14]:

```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_essays = []
# tqdm is for printing the status bar
for sentance1 in tqdm(project_data['essay'].values):
    sent= sentance1.lower()
    sent = decontracted(sent)
    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 [01:29<00:00, 1215.18it/s]
```

In [15]:

```
# after preprocesing
project_data['essay']=preprocessed_essays
project_data.drop(['project_essay_1'], axis=1, inplace=True)
project_data.drop(['project_essay_2'], axis=1, inplace=True)
project_data.drop(['project_essay_3'], axis=1, inplace=True)
```

```
project_data.drop(['project_essay_4'], axis=1, inplace=True)
print(project_data['essay'].values[0])
```

```
fortunate enough use fairy tale stem kits classroom well stem journals stude
nts really enjoyed would love implement lakeshore stem kits classroom next s
chool year provide excellent engaging stem lessons students come variety bac
kgrounds including language socioeconomic status many not lot experience sci
ence engineering kits give materials provide exciting opportunities students
month try several science stem steam projects would use kits robot help guid
e science instruction engaging meaningful ways adapt kits current language a
rts pacing guide already teach material kits like tall tales paul bunyan joh
nny appleseed following units taught next school year implement kits magnets
motion sink vs float robots often get units not know teaching right way usin
g right materials kits give additional ideas strategies lessons prepare stud
ents science challenging develop high quality science activities kits give m
aterials need provide students science activities go along curriculum classr
oom although things like magnets classroom not know use effectively kits pro
vide right amount materials show use appropriate way
```

In [16]:
```
project_data['essay'].values[0]
```

Out[16]:
```
'fortunate enough use fairy tale stem kits classroom well stem journals stud
ents really enjoyed would love implement lakeshore stem kits classroom next
school year provide excellent engaging stem lessons students come variety ba
ckgrounds including language socioeconomic status many not lot experience sc
ience engineering kits give materials provide exciting opportunities student
s month try several science stem steam projects would use kits robot help gu
ide science instruction engaging meaningful ways adapt kits current language
arts pacing guide already teach material kits like tall tales paul bunyan jo
hnny appleseed following units taught next school year implement kits magnet
s motion sink vs float robots often get units not know teaching right way us
ing right materials kits give additional ideas strategies lessons prepare st
udents science challenging develop high quality science activities kits give
materials need provide students science activities go along curriculum class
room although things like magnets classroom not know use effectively kits pr
ovide right amount materials show use appropriate way'
```

1.4 Preprocessing of `project_title`

In [17]:
```
# Combining all the above statemennts
from tqdm import tqdm
preprocessed_titles = []
# tqdm is for printing the status bar
for sentence2 in tqdm(project_data['project_title'].values):
```

```
    sent = sentence2.lower()
    sent = decontracted(sent)
    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 [00:04<00:00, 24435.44it/s]
```

In [18]:
```
# after preprocesing
project_data['project_title']=preprocessed_titles
print(project_data['project_title'][0])
```

```
not 21st century learners across ocean
```

In [19]:
```
#project_resource_summary
preprocessed_project_resource_summary=[]

# similarly you can preprocess the titles also
# Combining all the above statemennts
# Combining all the above statemennts
from tqdm import tqdm
preprocessed_titles = []
# tqdm is for printing the status bar
for sentence3 in tqdm(project_data['project_resource_summary'].values):
    sent = sentence3.lower()
    sent = decontracted(sent)
    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_project_resource_summary.append(sent.lower().strip())
```

```
100%|
| 109248/109248 [00:08<00:00, 13143.78it/s]
```

In [20]:
```
# after preprocesing
project_data['project_resource_summary']=preprocessed_project_resource_summary
```

```
#print(project_data['project_resource_summary'][0])
print(project_data['project_resource_summary'][0])
```

```
students need headphones supplemental supplies help individualize learning
```

In [21]:
```
#Preprocessing the project_grade_category
project_grade_category_cleaned=[]
for grade in tqdm(project_data['project_grade_category'].values):
    grade = grade.replace(' ', '_')
    grade = grade.replace('-', '_')
    project_grade_category_cleaned.append(grade)
project_data['Project_grade_category']=project_grade_category_cleaned
```

```
100%|██████████████████████████████████████████████████████████████████████████████|
109248/109248 [00:00<00:00, 670649.59it/s]
```

In [22]:
```
price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'
}).reset_index()
project_data = pd.merge(project_data, price_data, on='id', how='left')
```

In [23]:
```
project_data.head(2)
```

Out[23]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state
0	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA
1	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT

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.


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

Note: Data Leakage

1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
2. To avoid the issue of data-leakag, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method fit_transform() on you train data, and apply the method transform() on cv/test data.
4. For more details please go through this link.

2. Naive Bayes

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

In [24]:
```
from sklearn.model_selection import train_test_split
# split the data set into train and test respectively 80% and 20%
y=project_data['project_is_approved']
project_data.drop(['project_is_approved'],axis=1, inplace=True)
x=project_data
X_temp,X_test,Y_temp,Y_test=train_test_split(x,y,test_size=0.33,random_state=1
)
# split the data set into train and cv respectively 60% and 20%
X_train,X_cv,Y_train,Y_cv=train_test_split(X_temp,Y_temp,test_size=0.33,random
_state=1)
print("Shape of Train data set X={} Y={}".format(X_train.shape,Y_train.shape))
print("Shape of Test data set X={} Y={}".format(X_test.shape,Y_test.shape))
print("Shape of CV data set X={} Y={}".format(X_cv.shape,Y_cv.shape))
```

```
Shape of Train data set X=(49041, 16) Y=(49041,)
Shape of Test data set X=(36052, 16) Y=(36052,)
Shape of CV data set X=(24155, 16) Y=(24155,)
```

1.5 Preparing data for models

In [25]:
```
X_train.columns
```

Out[25]:
```
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', 'clean_categories',
       'clean_subcategories', 'essay', 'Project_grade_category', 'price',
       'quantity'],
      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

2.2 Make Data Model Ready: encoding numerical, categorical features

1.5.1 Vectorizing Categorical data

In [26]:
```
# we use count vectorizer to convert the values into one hot encoded features
# Project categories
from sklearn.feature_extraction.text import CountVectorizer
vectorizer_categories = CountVectorizer(vocabulary=list(sorted_cat_dict.keys
()),lowercase=False, binary=True)

tr_categories_one_hot=vectorizer_categories.fit_transform(X_train['clean_categ
ories'].values)
print(vectorizer_categories.get_feature_names())
```

```
cv_categories_one_hot =vectorizer_categories.transform(X_cv['clean_categories'
].values)
te_categories_one_hot =vectorizer_categories.transform(X_test['clean_categorie
s'].values)

print(tr_categories_one_hot.toarray()[0:1])
print("\nShape of matrix after one hot encodig for 'Project categories'\nTrain
data-{},\nCV data\t-{}\nTest data-{}".format(tr_categories_one_hot.shape,cv_ca
tegories_one_hot.shape,te_categories_one_hot.shape))
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning',
'SpecialNeeds', 'Health_Sports', 'Math_Science', 'Literacy_Language']
[[0 0 0 0 0 0 1 0 0]]

Shape of matrix after one hot encodig for 'Project categories'
Train data-(49041, 9),
CV data -(24155, 9)
Test data-(36052, 9)
```

In [27]:

```
# we use count vectorizer to convert the values into one hot encoded features
# Project subcategories
vectorizer_subcategories = CountVectorizer(vocabulary=list(sorted_sub_cat_dict
.keys()), lowercase=False, binary=True)

tr_sub_categories_one_hot=vectorizer_subcategories.fit_transform(X_train['clea
n_subcategories'].values)
print(vectorizer_subcategories.get_feature_names())

cv_sub_categories_one_hot = vectorizer_subcategories.transform(X_cv['clean_sub
categories'].values)
te_sub_categories_one_hot = vectorizer_subcategories.transform(X_test['clean_s
ubcategories'].values)

print(tr_sub_categories_one_hot.toarray()[0:2])
print("\nShape of matrix after one hot encodig for 'Project sub categories'\nT
rain data-{},\nCV data\t-{}\nTest data-{}".format(tr_sub_categories_one_hot.sh
ape,cv_sub_categories_one_hot.shape,te_sub_categories_one_hot.shape))
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement',
'Extracurricular', 'Civics_Government', 'ForeignLanguages', 'NutritionEducat
ion', 'Warmth', 'Care_Hunger', 'SocialSciences', 'PerformingArts', 'Characte
rEducation', 'TeamSports', 'Other', 'College_CareerPrep', 'Music', 'History_
Geography', 'Health_LifeScience', 'EarlyDevelopment', 'ESL', 'Gym_Fitness',
'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
[[0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0]
```

```
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]]

Shape of matrix after one hot encodig for 'Project sub categories'
Train data-(49041, 30),
CV data -(24155, 30)
Test data-(36052, 30)
```

In [28]:
```
# you can do the similar thing with state, teacher_prefix and project_grade_ca
tegory also
# we use count vectorizer to convert the values into one hot encoded features
#teacher_prefix
vectorizer_teacher_prefix = CountVectorizer(lowercase=False, binary=True)
tr_teacher_prefix_one_hot=vectorizer_teacher_prefix.fit_transform(X_train['tea
cher_prefix'].values.astype('str'))
print(vectorizer_teacher_prefix.get_feature_names())

cv_teacher_prefix_one_hot = vectorizer_teacher_prefix.transform(X_cv['teacher_
prefix'].values.astype('str'))
te_teacher_prefix_one_hot = vectorizer_teacher_prefix.transform(X_test['teache
r_prefix'].values.astype('str'))

print(tr_teacher_prefix_one_hot.toarray()[0:1])
print("\nShape of matrix after one hot encodig for 'teacher_prefix'\nTrain dat
a-{},\nCV data\t-{}\nTest data-{}".format(tr_teacher_prefix_one_hot.shape,cv_t
eacher_prefix_one_hot.shape,te_teacher_prefix_one_hot.shape))
```

```
['Dr', 'Mr', 'Mrs', 'Ms', 'Teacher', 'nan']
[[0 0 0 1 0 0]]

Shape of matrix after one hot encodig for 'teacher_prefix'
Train data-(49041, 6),
CV data -(24155, 6)
Test data-(36052, 6)
```

In [29]:
```
# we use count vectorizer to convert the values into one hot encoded features
#school_state
vectorizer_school_state = CountVectorizer(lowercase=False, binary=True)
tr_school_state_one_hot=vectorizer_school_state.fit_transform(X_train['school_
state'].values.astype('str'))
print(vectorizer_school_state.get_feature_names())

cv_school_state_one_hot = vectorizer_school_state.transform(X_cv['school_stat
e'].values.astype('str'))
te_school_state_one_hot = vectorizer_school_state.transform(X_test['school_sta
te'].values.astype('str'))
```

```
print(tr_school_state_one_hot.toarray()[0:1])
print("\nShape of matrix after one hot encodig for 'teacher_prefix'\nTrain dat
a-{},\nCV data\t-{}\nTest data-{}".format(tr_school_state_one_hot.shape,cv_sch
ool_state_one_hot.shape,te_school_state_one_hot.shape))
```

```
['AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA', 'HI', 'I
A', 'ID', 'IL', 'IN', 'KS', 'KY', 'LA', 'MA', 'MD', 'ME', 'MI', 'MN', 'MO',
'MS', 'MT', 'NC', 'ND', 'NE', 'NH', 'NJ', 'NM', 'NV', 'NY', 'OH', 'OK', 'O
R', 'PA', 'RI', 'SC', 'SD', 'TN', 'TX', 'UT', 'VA', 'VT', 'WA', 'WI', 'WV',
'WY']
[[0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]]

Shape of matrix after one hot encodig for 'teacher_prefix'
Train data-(49041, 51),
CV data -(24155, 51)
Test data-(36052, 51)
```

In [30]:
```
# we use count vectorizer to convert the values into one hot encoded features
#project_grade_category
vectorizer_grade_category = CountVectorizer(lowercase=False, binary=True)
tr_grade_category_one_hot=vectorizer_grade_category.fit_transform(X_train['Pro
ject_grade_category'])
print(vectorizer_grade_category.get_feature_names())

cv_grade_category_one_hot = vectorizer_grade_category.transform(X_cv['Project_
grade_category'])
te_grade_category_one_hot = vectorizer_grade_category.transform(X_test['Projec
t_grade_category'])

print(tr_grade_category_one_hot.toarray()[0:1])
print(cv_grade_category_one_hot.toarray()[0:1])
print(te_grade_category_one_hot.toarray()[0:1])
print("\nShape of matrix after one hot encodig for 'project_grade_category'\nT
rain data-{},\nCV data\t-{}\nTest data-{}".format(tr_grade_category_one_hot.sh
ape,cv_grade_category_one_hot.shape,te_grade_category_one_hot.shape))
```

```
['Grades_3_5', 'Grades_6_8', 'Grades_9_12', 'Grades_PreK_2']
[[1 0 0 0]]
[[1 0 0 0]]
[[0 0 0 1]]

Shape of matrix after one hot encodig for 'project_grade_category'
Train data-(49041, 4),
```

```
CV data -(24155, 4)
Test data-(36052, 4)
```

1.5.2 normalizing Numerical features

In [31]:
```
# https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html#sklearn.preprocessing.MinMaxScaler

from sklearn.preprocessing import Normalizer
# price_normalized = Normalizer.fit(X_train['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ... 399.   287.73   5.5 ].
# Reshape your data either using array.reshape(1,-1)

normalizer_price = Normalizer(copy=True, norm='l2')

tr_price_normalized=normalizer_price.fit_transform(X_train['price'].values.reshape(1,-1))
# Now standardize the data with above maen and variance.
cv_price_normalized = normalizer_price.transform(X_cv['price'].values.reshape(1, -1))
te_price_normalized = normalizer_price.transform(X_test['price'].values.reshape(1,- 1))
#Reshaping price
tr_price_normalized=tr_price_normalized.reshape(-1,1)
cv_price_normalized=cv_price_normalized.reshape(-1,1)
te_price_normalized=te_price_normalized.reshape(-1,1)
```

In [32]:
```
print("\nShape of matrix after column standardization for 'price'\nTrain data-{},\nCV data\t-{}\nTest data-{}".format(tr_price_normalized.shape,cv_price_normalized.shape,te_price_normalized.shape))
print(tr_price_normalized)
```

```
Shape of matrix after column standardization for 'price'
Train data-(49041, 1),
CV data -(24155, 1)
Test data-(36052, 1)
[[0.00516108]
 [0.00353829]
 [0.00115414]
 ...
 [0.00896599]
```

```
 [0.00087946]
 [0.00208166]]
```

In [33]:

```
#quantity
normalizer_quantity =  Normalizer(copy=True, norm='l2')
tr_quantity_normalized=normalizer_quantity.fit_transform(X_train['quantity'].v
alues.reshape(1,-1)) # finding the mean and standard deviation of this data
# Normalizing the quantity.
cv_quantity_normalized = normalizer_quantity.transform(X_cv['quantity'].values
.reshape(1, -1))
te_quantity_normalized = normalizer_quantity.transform(X_test['quantity'].valu
es.reshape(1,-1))
#reshaping data
tr_quantity_normalized=tr_quantity_normalized.reshape(-1,1)
cv_quantity_normalized=cv_quantity_normalized.reshape(-1,1)
te_quantity_normalized=te_quantity_normalized.reshape(-1,1)
print("\nShape of matrix after column standardization for 'quantity'\nTrain da
ta-{},\nCV data\t-{}\nTest data-{}".format(tr_quantity_normalized.shape,cv_qua
ntity_normalized.shape,te_quantity_normalized.shape))
print(tr_quantity_normalized)
```

```
Shape of matrix after column standardization for 'quantity'
Train data-(49041, 1),
CV data -(24155, 1)
Test data-(36052, 1)
[[0.00057768]
 [0.00057768]
 [0.00101094]
 ...
 [0.004477  ]
 [0.00158861]
 [0.00043326]]
```

In [34]:

```
#teacher_number_of_previously_posted_projects
normalizer_teacher_number_of_previously_posted_projects =Normalizer(copy=True,
norm='l2')
# Normalizing the teacher_number_of_previously_posted_projects.
tr_teacher_number_of_previously_posted_projects_normalized=normalizer_teacher_
number_of_previously_posted_projects.fit_transform(X_train['teacher_number_of_
previously_posted_projects'].values.reshape(1,-1)) # finding the mean and stan
dard deviation of this data
cv_teacher_number_of_previously_posted_projects_normalized = normalizer_teache
r_number_of_previously_posted_projects.transform(X_cv['teacher_number_of_previ
ously_posted_projects'].values.reshape(-1, 1))
te_teacher_number_of_previously_posted_projects_normalized = normalizer_teache
```

```
r_number_of_previously_posted_projects.transform(X_test['teacher_number_of_pre
viously_posted_projects'].values.reshape(-1, 1))
tr_teacher_number_of_previously_posted_projects_normalized=tr_teacher_number_o
f_previously_posted_projects_normalized.reshape(-1,1)
cv_teacher_number_of_previously_posted_projects_normalized=cv_teacher_number_o
f_previously_posted_projects_normalized.reshape(-1,1)
te_teacher_number_of_previously_posted_projects_normalized=te_teacher_number_o
f_previously_posted_projects_normalized.reshape(-1,1)
print("\nShape of matrix after column standardization for 'teacher_number_of_p
reviously_posted_projects'\nTrain data-{},\nCV data\t-{}\nTest data-{}".format
(tr_teacher_number_of_previously_posted_projects_normalized.shape,cv_teacher_n
umber_of_previously_posted_projects_normalized.shape,te_teacher_number_of_prev
iously_posted_projects_normalized.shape))
print(tr_teacher_number_of_previously_posted_projects_normalized)
```

```
Shape of matrix after column standardization for 'teacher_number_of_previous
ly_posted_projects'
Train data-(49041, 1),
CV data -(24155, 1)
Test data-(36052, 1)
[[0.01009535]
 [0.        ]
 [0.00474176]
 ...
 [0.00015296]
 [0.        ]
 [0.00336512]]
```

2.3 Make Data Model Ready: encoding eassay, and project_title

1.5.2 Vectorizing Text data

1.5.2.1

Bag of Words on `preprocessed_essay`

In [35]:

```
#Bag of words of Project essays
# We are considering only the words which appeared in at least 10 documents(ro
ws or projects) and max feature is 000.
```

```
#Fitting train data because we need all and transforming  train ,cv and test v
ector shape should be same.
vectorizer_essays = CountVectorizer(min_df=10,max_features=5000,ngram_range=(1
, 4))#max_features=5000
tr_text_bow=vectorizer_essays.fit_transform(X_train['essay']) # fitting train
 data

#transforming train,cv and test data

cv_text_bow = vectorizer_essays.transform(X_cv['essay'])
te_text_bow = vectorizer_essays.transform(X_test['essay'])
print("Shape of matrix after one hot encodig \nTrain data-{},\nCV data\t-{}\nT
est data-{}".format(tr_text_bow.shape,cv_text_bow.shape,te_text_bow.shape))
```

```
Shape of matrix after one hot encodig
Train data-(49041, 5000),
CV data -(24155, 5000)
Test data-(36052, 5000)
```

In [36]:
```
print('Some feature names of bag of words of the essays')
print('='*50)
print(vectorizer_essays.get_feature_names()[1000:1020])
print(tr_text_bow.toarray()[0:1])
```

```
Some feature names of bag of words of the essays
==================================================
['current', 'current events', 'currently', 'currently not', 'currently stude
nts', 'curricular', 'curriculum', 'curriculum students', 'cushions', 'cut',
'cuts', 'cutting', 'cycle', 'cycles', 'daily', 'daily basis', 'daily basis s
tudents', 'daily lives', 'daily students', 'dance']
[[0 0 0 ... 0 0 0]]
```

Bag of Words on `project_title`

In [37]:
```
#Bag of words project_title
# We are considering only the words which appeared in at least 5 documents(row
s or projects) and max number of feature is 5000.
#Fitting train data and transforming  train ,cv and test vector shape should b
e same.
vectorizer_title = CountVectorizer(min_df=10,ngram_range=(1, 4),max_features=5
000)
tr_text_bow_title=vectorizer_title.fit_transform(X_train['project_title'])
cv_text_bow_title = vectorizer_title.transform(X_cv['project_title'])
te_text_bow_title = vectorizer_title.transform(X_test['project_title'])
```

```
print("Shape of matrix after one hot encodig \nTrain data-{},\nCV data\t-{}\nTest data-{}".format(tr_text_bow_title.shape,cv_text_bow_title.shape,te_text_bow_title.shape))
```

```
Shape of matrix after one hot encodig
Train data-(49041, 3388),
CV data -(24155, 3388)
Test data-(36052, 3388)
```

In [38]:
```
print('Some feature names of bag of words of the project title')
print('='*50)
print(vectorizer_title.get_feature_names()[1000:1020])
print(tr_text_bow_title.toarray()[0:2])
```

```
Some feature names of bag of words of the project title
==================================================
['fall love', 'falling', 'families', 'family', 'family fun', 'fantastic', 'farm', 'fast', 'favorite', 'fear', 'feed', 'feeding', 'feel', 'feeling', 'feet', 'fiction', 'fiction books', 'fidget', 'fidgeting', 'fidgets']
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

Bag of Words on `project_resource_summary`

In [39]:
```
#Bag of words project_resource_summary
# We are considering only the words which appeared in at least 5 documents(rows or projects) and max number of feature is 5000.
#Fitting train data and transforming  train ,cv and test vector shape should be same.
vectorizer_summary = CountVectorizer(min_df=10,ngram_range=(1, 4),max_features=5000)
tr_text_bow_summary=vectorizer_summary.fit_transform(X_train['project_resource_summary'])
cv_text_bow_summary = vectorizer_summary.transform(X_cv['project_resource_summary'])
te_text_bow_summary = vectorizer_summary.transform(X_test['project_resource_summary'])
print("Shape of matrix after one hot encodig \nTrain data-{},\nCV data\t-{}\nTest data-{}".format(tr_text_bow_summary.shape,cv_text_bow_summary.shape,te_text_bow_summary.shape))
```

```
Shape of matrix after one hot encodig
Train data-(49041, 5000),
CV data -(24155, 5000)
Test data-(36052, 5000)
```

In [40]:
```
print('Some feature names of bag of words of the project resource summary')
print('='*50)
print(vectorizer_summary.get_feature_names()[1000:1020])
print(tr_text_bow_summary.toarray()[0:2])
```

```
Some feature names of bag of words of the project resource summary
==================================================
['dash robots', 'data', 'date', 'day', 'day long', 'days', 'decorations', 'd
eepen', 'deeper', 'dell', 'demonstrate', 'deodorant', 'depth', 'deserve', 'd
esign', 'design build', 'designated', 'designed', 'designing', 'designs']
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

1.5.2.2 TFIDF vectorizer

TFIDF Vectorizer on `preprocessed_essay`

In [41]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vectorizer_essays = TfidfVectorizer(min_df=10)
#Fitting train data and transforming  train ,cv and test vector shape should b
e same.
tr_text_tfidf=tfidf_vectorizer_essays.fit_transform(X_train['essay'])
cv_text_tfidf = tfidf_vectorizer_essays.transform(X_cv['essay'])
te_text_tfidf = tfidf_vectorizer_essays.transform(X_test['essay'])
print("Shape of matrix TFIDF Vectorizer on essays \nTrain data-{},\nCV data\t-
{}\nTest data-{}".format(tr_text_tfidf.shape,cv_text_tfidf.shape,te_text_tfidf
.shape))
```

```
Shape of matrix TFIDF Vectorizer on essays
Train data-(49041, 12021),
CV data -(24155, 12021)
Test data-(36052, 12021)
```

In [42]:
```
print('Sample of TFIDF Vectorizer on essays')
print('='*50)
print(tr_text_tfidf.toarray()[0:1])
print(tfidf_vectorizer_essays.get_feature_names()[300:310])
```

```
Sample of TFIDF Vectorizer on essays
==================================================
[[0. 0. 0. ... 0. 0. 0.]]
['accurate', 'accurately', 'accustomed', 'ace', 'acer', 'ache', 'acheive',
'aches', 'achievable', 'achieve']
```

1.4.2.4 TFIDF Vectorizer on `project_title`

In [43]:
```
# Similarly you can vectorize for title also
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vectorizer_title = TfidfVectorizer(min_df=10)
#Fitting train data and transforming  train ,cv and test vector shape should b
e same.
tr_title_tfidf=tfidf_vectorizer_title.fit_transform(X_train['project_title'])

cv_title_tfidf = tfidf_vectorizer_title.transform(X_cv['project_title'])
te_title_tfidf = tfidf_vectorizer_title.transform(X_test['project_title'])

print("Shape of matrix TFIDF Vectorizer on essays \nTrain data-{},\nCV data\t-
{}\nTest data-{}".format(tr_title_tfidf.shape,cv_title_tfidf.shape,te_title_tf
idf.shape))
```

```
Shape of matrix TFIDF Vectorizer on essays
Train data-(49041, 1978),
CV data -(24155, 1978)
Test data-(36052, 1978)
```

In [44]:
```
print('Sample of TFIDF Vectorizer on `project_title`')
print('='*50)
print(tr_title_tfidf.toarray()[0:1])
print(tfidf_vectorizer_title.get_feature_names()[100:110])
```

```
Sample of TFIDF Vectorizer on `project_title`
==================================================
[[0. 0. 0. ... 0. 0. 0.]]
['authors', 'autism', 'autistic', 'avid', 'award', 'awareness', 'away', 'awe
some', 'baby', 'back']
```

1.4.2.4 TFIDF Vectorizer on `project_resource_summary`

In [45]:
```
# Similarly you can vectorize for summary also
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vectorizer_summary = TfidfVectorizer(min_df=10)
#Fitting train data and transforming  train ,cv and test vector shape should b
e same.
tr_summary_tfidf=tfidf_vectorizer_summary.fit_transform(X_train['project_resou
rce_summary'])
```

```
cv_summary_tfidf = tfidf_vectorizer_summary.transform(X_cv['project_resource_summary'])
te_summary_tfidf = tfidf_vectorizer_summary.transform(X_test['project_resource_summary'])

print("Shape of matrix TFIDF Vectorizer on essays \nTrain data-{},\nCV data\t-{}\nTest data-{}".format(tr_summary_tfidf.shape,cv_summary_tfidf.shape,te_summary_tfidf.shape))
```

```
Shape of matrix TFIDF Vectorizer on essays
Train data-(49041, 3875),
CV data -(24155, 3875)
Test data-(36052, 3875)
```

In [46]:
```
print('Sample of TFIDF Vectorizer on `project_title`')
print('='*50)
print(tr_summary_tfidf.toarray()[0:1])
print(tfidf_vectorizer_summary.get_feature_names()[100:110])
```

```
Sample of TFIDF Vectorizer on `project_title`
==================================================
[[0. 0. 0. ... 0. 0. 0.]]
['activity', 'actual', 'actually', 'adapt', 'adaptations', 'adapted', 'adapter', 'adaptive', 'add', 'added']
```

1.5.4 Merging all the above features

- we need to merge all the numerical vectors i.e catogorical, text, numerical vectors

In [47]:
```
print(tr_school_state_one_hot.shape)
print(tr_categories_one_hot.shape)
#print(sub_categories.shape)
print(tr_sub_categories_one_hot.shape)
print(tr_teacher_prefix_one_hot.shape)
print(tr_grade_category_one_hot.shape)
print(tr_text_bow_title.shape)
print(tr_text_bow.shape)
print(tr_price_normalized.shape)
```

```
(49041, 51)
(49041, 9)
(49041, 30)
(49041, 6)
(49041, 4)
```

```
(49041, 3388)
(49041, 5000)
(49041, 1)
```

In [48]:

```
%%time
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
#categorical, numerical features + project_title(BOW)
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a den
se matirx :)
tr_X_BOW= hstack((tr_school_state_one_hot,tr_categories_one_hot,tr_sub_categor
ies_one_hot,tr_teacher_prefix_one_hot,tr_grade_category_one_hot,tr_price_norma
lized,tr_teacher_number_of_previously_posted_projects_normalized,tr_text_bow_t
itle,tr_text_bow,tr_text_bow_summary)).tocsr()
cv_X_BOW= hstack((cv_school_state_one_hot,cv_categories_one_hot,cv_sub_categor
ies_one_hot,cv_teacher_prefix_one_hot,cv_grade_category_one_hot,cv_price_norma
lized,cv_teacher_number_of_previously_posted_projects_normalized,cv_text_bow_t
itle,cv_text_bow,cv_text_bow_summary)).tocsr()
te_X_BOW= hstack((te_school_state_one_hot,te_categories_one_hot,te_sub_categor
ies_one_hot,te_teacher_prefix_one_hot,te_grade_category_one_hot,te_price_norma
lized,te_teacher_number_of_previously_posted_projects_normalized,te_text_bow_t
itle,te_text_bow,te_text_bow_summary)).tocsr()
tr_X_BOW=tr_X_BOW.toarray()
cv_X_BOW=cv_X_BOW.toarray()
te_X_BOW=te_X_BOW.toarray()
print(tr_X_BOW.shape)
print(cv_X_BOW.shape)
print(te_X_BOW.shape)
```

```
(49041, 13490)
(24155, 13490)
(36052, 13490)
Wall time: 8.33 s
```

In [49]:

```
#adding all set 1 features to BOW_features
def add_features(vectorizer,list_a):

    for fea in list(vectorizer.get_feature_names()):
        list_a.append(fea)
    return list_a


BOW_features=[]
add_features(vectorizer_school_state,BOW_features)
add_features(vectorizer_categories,BOW_features)
```

```
add_features(vectorizer_subcategories,BOW_features)
add_features(vectorizer_teacher_prefix,BOW_features)
add_features(vectorizer_grade_category,BOW_features)
BOW_features.append("Price")
BOW_features.append("teacher number of previously posted projects")
add_features(vectorizer_title,BOW_features)
add_features(vectorizer_essays,BOW_features)
add_features(vectorizer_summary,BOW_features)
print("Len of the BOW features",len(BOW_features))
```

```
Len of the BOW features 13490
```

In [50]:

```
%%time
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
#categorical, numerical features + project_title(TFIDF)
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a den
se matirx :)
tr_X_TFIDF= hstack((tr_school_state_one_hot,tr_categories_one_hot, tr_sub_cate
gories_one_hot,tr_teacher_prefix_one_hot,tr_grade_category_one_hot,tr_price_no
rmalized,tr_teacher_number_of_previously_posted_projects_normalized,tr_title_t
fidf,tr_text_tfidf,tr_summary_tfidf))
cv_X_TFIDF= hstack((cv_school_state_one_hot,cv_categories_one_hot, cv_sub_cate
gories_one_hot,cv_teacher_prefix_one_hot,cv_grade_category_one_hot,cv_price_no
rmalized,cv_teacher_number_of_previously_posted_projects_normalized,cv_title_t
fidf,cv_text_tfidf,cv_summary_tfidf))
te_X_TFIDF= hstack((te_school_state_one_hot,te_categories_one_hot, te_sub_cate
gories_one_hot,te_teacher_prefix_one_hot,te_grade_category_one_hot,te_price_no
rmalized,te_teacher_number_of_previously_posted_projects_normalized,te_title_t
fidf,te_text_tfidf,te_summary_tfidf))

tr_X_TFIDF=tr_X_TFIDF.toarray()
cv_X_TFIDF=cv_X_TFIDF.toarray()
te_X_TFIDF=te_X_TFIDF.toarray()
print(tr_X_TFIDF.shape)
print(cv_X_TFIDF.shape)
print(te_X_TFIDF.shape)
```

```
(49041, 17976)
(24155, 17976)
(36052, 17976)
Wall time: 40 s
```

In [51]:

```
#adding all set 2 features to TFIDF_features
def add_features(vectorizer,list_a):
```

```
    for fea in list(vectorizer.get_feature_names()):
        list_a.append(fea)
    return list_a


TFIDF_features=[]
add_features(vectorizer_school_state,TFIDF_features)
add_features(vectorizer_categories,TFIDF_features)
add_features(vectorizer_subcategories,TFIDF_features)
add_features(vectorizer_teacher_prefix,TFIDF_features)
add_features(vectorizer_grade_category,TFIDF_features)
TFIDF_features.append("Price")
TFIDF_features.append("teacher number of previously posted projects")
add_features(tfidf_vectorizer_title,TFIDF_features)
add_features(tfidf_vectorizer_essays,TFIDF_features)
add_features(tfidf_vectorizer_summary,TFIDF_features)
print("Len of the TFIDF features",len(TFIDF_features))
```

```
Len of the TFIDF features 17976
```

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 [52]:

```
#https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matri
x
#Drawing confusion matrix
def draw_confusion_matrix(clf,threshold,y_true,y_hat,tpr,fpr,t):
    result=[]
    y_pred=[]

    #finding threshold which maximises the tpr and minimises the fpr
    thr=threshold[np.argmax((tpr*(1-fpr)))]

    for probab in y_hat:

        if probab >= thr:
            y_pred.append(1)
        else:
            y_pred.append(0)
```

```
    result=confusion_matrix(y_true,y_pred,labels=[0,1])
    df_cm = pd.DataFrame(result,range(2),range(2))
    df_cm.columns = ['Predicted NO','Predicted YES']
    df_cm = df_cm.rename({0: 'Actual NO', 1: 'Actual YES'})
    plt.figure(figsize = (5,3))
    plt.title(t)
    sns.heatmap(df_cm, annot=True,annot_kws={"size": 12}, fmt='g')
```

In [53]:
```
#function to plot lines
def plot_curve(train_auc_scores_tmp,validation_auc_scores_tmp,k_n,title):
    plt.xscale('log')
    plt.plot(k_n,train_auc_scores_tmp,label="Train curve")
    plt.plot(k_n,validation_auc_scores_tmp,label="Validation curve")
    plt.scatter(k_n, train_auc_scores_tmp, label='Train AUC points')
    plt.scatter(k_n, validation_auc_scores_tmp, label='CV AUC points')
    plt.title(title)
    plt.xlabel("Alpha-hyper paramters")
    plt.ylabel("AUC")
    plt.legend()
    plt.show()
```

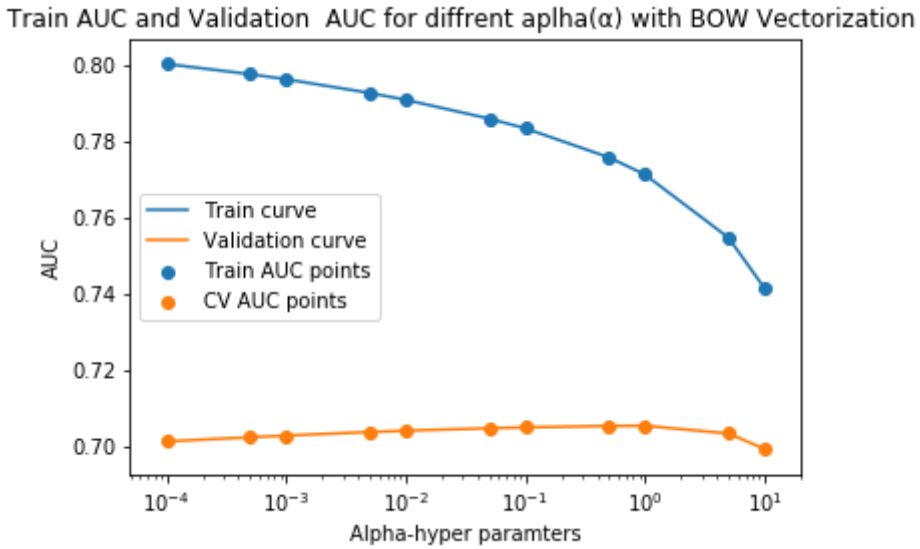
In [54]:
```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import GridSearchCV

def roc_auc_compute(x_train,y_train,x_test_temp,y_test_temp,alpa_n,title,title
2):
    #grid_params = {'mnb__alpha': al,'mnb__fit_prior': [True, False],'scorin
g':'roc_auc','cv':3,'class_prior':[[0.7,0.2],[0.5,0.5]]}
    n_neighbors=alpa_n
    train_auc_scores=[]
    validation_auc_scores=[]
    train_cv_scores=[]
    validation_cv_scores=[]
    best_cv_auc_scores=0
    for alpa in tqdm(n_neighbors):
        parameters = {'alpha':[alpa]}
        trained_mulNB = MultinomialNB(alpha=alpa,class_prior=[0.5,0.5])
        #trainning model
        trained_mulNB.fit(x_train,y_train)

        # predict the response on the cross validation
        pradicted_labels=trained_mulNB.predict_proba(x_test_temp)
```

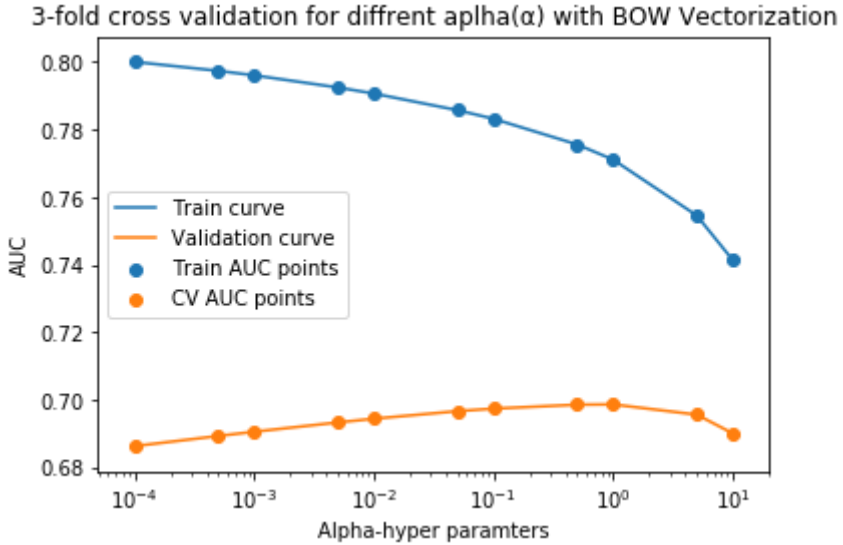
```
        #Calculating validation auc scores
        validation_auc=roc_auc_score(y_test_temp,pradicted_labels[:,1]) #1-roc
_auc_score for validation error

        # predict the response on the train and calculating the train auc
        train_auc=roc_auc_score(y_train,trained_mulNB.predict_proba(x_train)
[:,1]) #1-roc_auc_score for train error

        # K-flod cross validation
        gs = GridSearchCV(trained_mulNB,parameters,cv=3,scoring='roc_auc')
        gs.fit(x_train,y_train)
        train_auc= gs.cv_results_['mean_train_score']
        cv_auc = gs.cv_results_['mean_test_score']

        train_cv_scores.append(train_auc)
        validation_cv_scores.append(cv_auc)
        train_auc_scores.append(train_auc)
        validation_auc_scores.append(validation_auc)
        if cv_auc>=best_cv_auc_scores:
            best_cv_auc_scores=cv_auc

    log_alpa=np.log10(n_neighbors)#appling log10 on alpha
    plot_curve(train_auc_scores,validation_auc_scores,n_neighbors,title)
    plot_curve(train_cv_scores,validation_cv_scores,n_neighbors,title2)
    print("best AUC",best_cv_auc_scores)
    print("best alpha",alpa_n[validation_cv_scores.index(best_cv_auc_scores)])
```

2.4.1 Applying Naive Bayes on BOW, SET 1

In [65]:
```
%%time
title="Train AUC and Validation  AUC for diffrent aplha(α) with BOW Vectorizat
ion "
title2="3-fold cross validation for diffrent aplha(α) with BOW Vectorization"
#Reason for choosing odd k's is there will not be difficulties while calculati
ng majority votes for data point.
#Subseting the trian and cv data because my laptop has only 8GB of RAM
roc_auc_compute(tr_X_BOW,Y_train,cv_X_BOW,Y_cv,[10,5,1,0.5,0.1,0.05,0.01,0.00
5,0.001,0.0005,0.0001],title,title2)
```

```
100%|
| 11/11 [09:34<00:00, 46.80s/it]
```



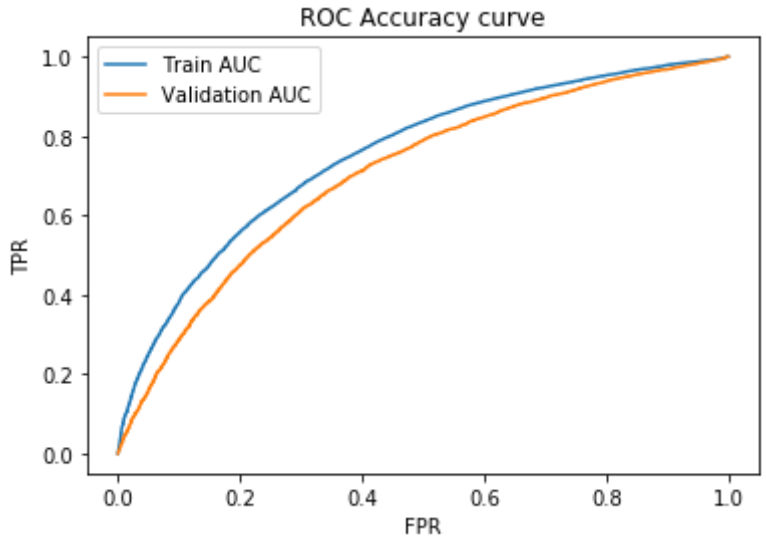



```
best AUC [0.69858042]
best alpha 1
Wall time: 9min 48s
```

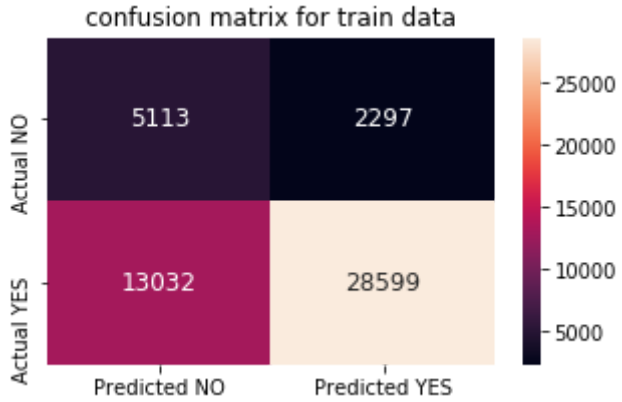
In [66]:

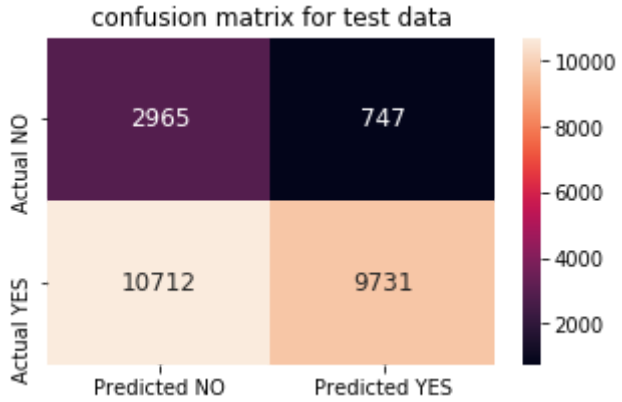
```
%%time
#https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for
-classification-in-python/
from sklearn.metrics import roc_curve, auc
#training model with best K-Hyper paramter
trainedmulNB_BOW=MultinomialNB(alpha=1,class_prior=[0.5,0.5])
#trainning model
```

```
trainedmulNB_BOW.fit(tr_X_BOW,Y_train)
# predict the response on the train data
predicted_labels_train=trainedmulNB_BOW.predict_proba(tr_X_BOW)
# predict the response on the test data
predicted_labels_test=trainedmulNB_BOW.predict_proba(cv_X_BOW)
#Calculating FPR and TPR for train and test data
tr_fpr,tr_tpr,tr_threshold=roc_curve(Y_train,predicted_labels_train[:,1])
te_fpr,te_tpr,te_threshold=roc_curve(Y_cv,predicted_labels_test[:,1])
#drawing ROC ROC Accuracy curve for test and train data
plt.plot(tr_fpr,tr_tpr,label="Train AUC")
plt.plot(te_fpr,te_tpr,label="Validation AUC")
plt.title("ROC Accuracy curve")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.legend()
plt.show()
print("Train AUC =",round(auc(tr_fpr,tr_tpr),2))
print("Test AUC =",round(auc(te_fpr,te_tpr),2))
#drawing confusion matrix for test and train data
t2="confusion matrix for train data"
draw_confusion_matrix(trainedmulNB_BOW,tr_threshold,Y_train,predicted_labels_t
rain[:,1],tr_tpr,tr_fpr,t2)
t1="confusion matrix for test data"
draw_confusion_matrix(trainedmulNB_BOW,tr_threshold,Y_cv,predicted_labels_test
[:,1],te_tpr,te_fpr,t1)
```




```
Train AUC = 0.75
Test AUC = 0.71
Wall time: 7.61 s
```





2.4.1.1 Top 20 important features of positive class from SET 1

In [67]:
```
#Getting feature index which will contribute more toward classifying positive
 class
pos_class_prob=trainedmulNB_BOW.feature_log_prob_[1,:]
#applying argsort() on posative class probabilities
pos_class_prob_index=pos_class_prob.argsort()
# getting top 20 features
top_20_positive_features_index=pos_class_prob_index[-1:-21:-1]
#printing top 20 features
print(top_20_positive_features_index)
#list(BOW_features[top_20_positive_features_index])
print("Top 20 features for positive class")
print("="*40)
print("index\tfeatures")
print("="*15)
```

```
for index in top_20_positive_features_index:
    print((index),(BOW_features[index]))
```

```
[ 7605  7166  5873  4183  6474  5819  5435 12493 10943 12508  6159  6373
  8383  6388  6977  8193  6051  4514  3584  4275]
Top 20 features for positive class
========================================
index   features
===============
7605 students
7166 school
5873 learning
4183 classroom
6474 not
5819 learn
5435 help
12493 students
10943 need
12508 students need
6159 many
6373 nannan
8383 work
6388 need
6977 reading
8193 use
6051 love
4514 day
3584 able
4275 come
```

2.4.1.2 Top 20 important features of negative class from SET 1

In [71]:
```
#Getting feature index which will contribute more toward classifying negative
 class
neg_class_prob=trainedmulNB_BOW.feature_log_prob_[0,:]
#applying argsort() on negative class probabilities
neg_class_prob_index=neg_class_prob.argsort()
# getting top 20 features
top_20_negative_features_index=neg_class_prob_index[1:21]
#printing top 20 features
print(top_20_negative_features_index)
#list(BOW_features[top_20_negative_features_index])
print("Top 20 features for negative class")
print("="*40)
print("index\tfeatures")
```

```
print("="*15)
for index in top_20_negative_features_index:
    print(index,BOW_features[index])
```

```
[10243 10242 10240  1379  2261  1378  2264  1377 12922 12923  1355  1352
  2307 12953  1349 11648  2339  2366  2372  2374]
Top 20 features for negative class
=========================================
index   features
===============
10243 hokki stools help
10242 hokki stools allow
10240 hokki stool
1379 graphing calculators
2261 north
1378 graphing
2264 not hear
1377 graphics
12922 students need stand
12923 students need stand desks
1355 got move
1352 google chromebooks
2307 orchestra
12953 students need ti
1349 good books
11648 pair
2339 papers
2366 personal space
2372 phones
2374 phonics fun
```

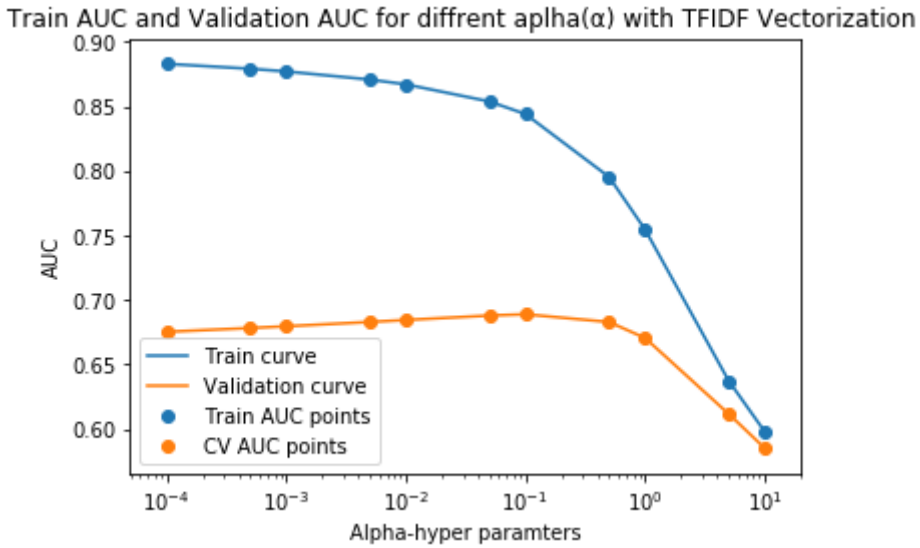
2.4.2 Applying Naive Bayes on TFIDF, SET 2

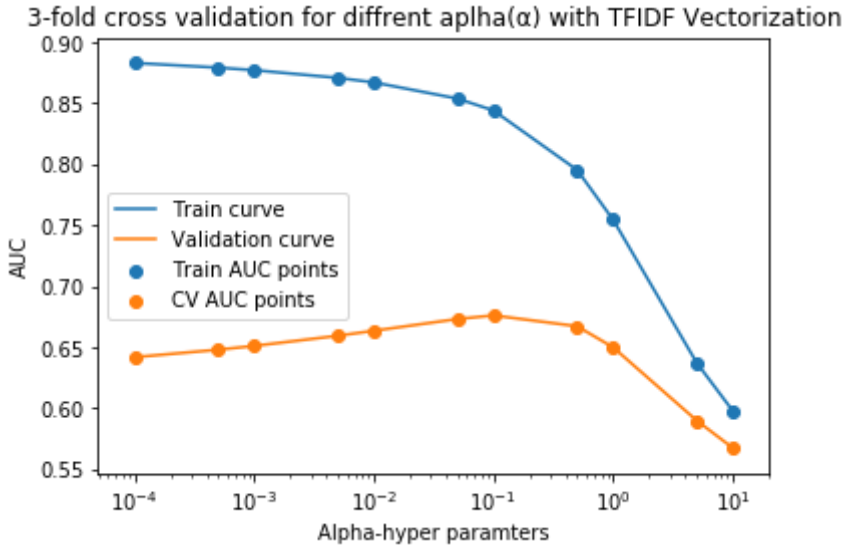
In [64]:

```
%%time
title="Train AUC and Validation AUC for diffrent aplha(α) with TFIDF Vectoriza
tion "
title2="3-fold cross validation for diffrent aplha(α) with TFIDF Vectorizatio
n"
#Reason for choosing odd k's is there will not be difficulties while calculati
ng majority votes for data point.
#Subseting the trian and cv data because my laptop has only 8GB of RAM
roc_auc_compute(tr_X_TFIDF,Y_train,cv_X_TFIDF,Y_cv,[10,5,1,0.5,0.1,0.05,0.01,
0.005,0.001,0.0005,0.0001],title,title2)
```

```
100%|
```

[redacted]| 11/11 [20:43<00:00, 112.71s/it]



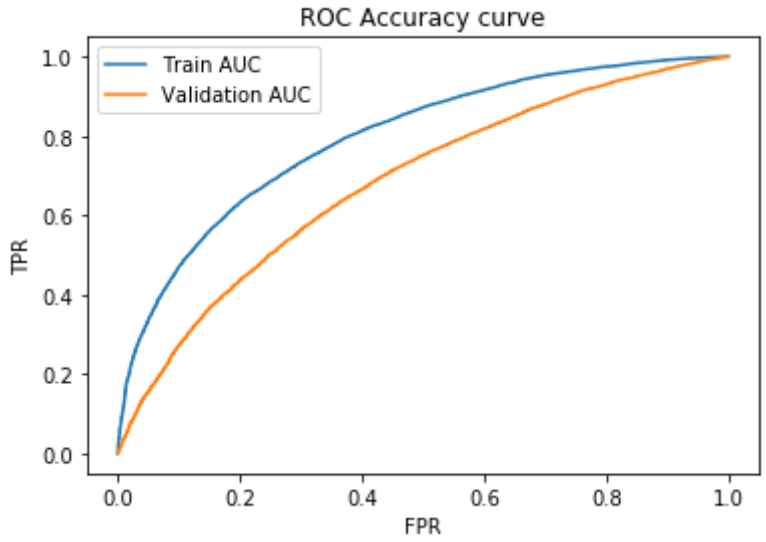



```
best AUC [0.67600028]
best alpha 0.1
Wall time: 20min 44s
```

In [76]:

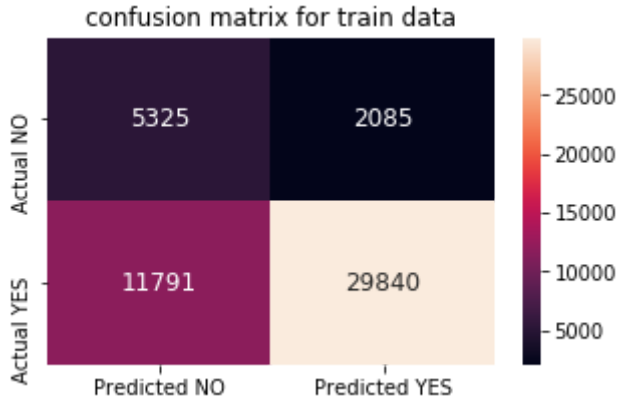
```
%%time
#https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for
-classification-in-python/
from sklearn.metrics import roc_curve, auc
#training model with best K-Hyper paramter
```

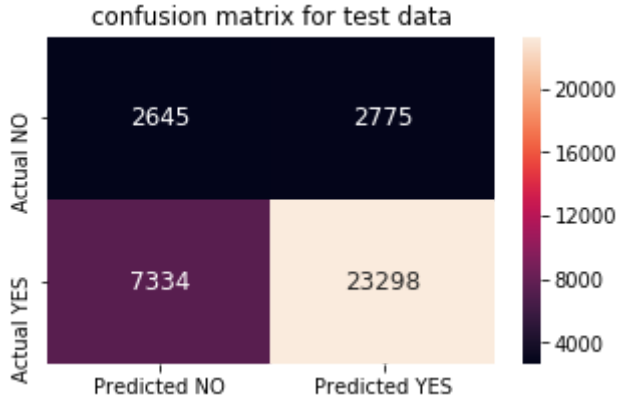
```
trainedmulNB_TFIDF=MultinomialNB(alpha=0.25,class_prior=[0.5,0.5])
#trainning model
trainedmulNB_TFIDF.fit(tr_X_TFIDF,Y_train)
# predict the response on the train data
predicted_labels_train=trainedmulNB_TFIDF.predict_proba(tr_X_TFIDF)
# predict the response on the test data
predicted_labels_test=trainedmulNB_TFIDF.predict_proba(te_X_TFIDF)
#Calculating FPR and TPR for train and test data
tr_fpr,tr_tpr,tr_threshold=roc_curve(Y_train,predicted_labels_train[:,1])
te_fpr,te_tpr,te_threshold=roc_curve(Y_test,predicted_labels_test[:,1])
#drawing ROC ROC Accuracy curve for test and train data
plt.plot(tr_fpr,tr_tpr,label="Train AUC")
plt.plot(te_fpr,te_tpr,label="Validation AUC")
plt.title("ROC Accuracy curve")
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.legend()
plt.show()
print("Train AUC =",round(auc(tr_fpr,tr_tpr),3))
print("Test AUC =",round(auc(te_fpr,te_tpr),3))
#drawing confusion matrix for test and train data
t2="confusion matrix for train data"
draw_confusion_matrix(trainedmulNB_TFIDF,tr_threshold,Y_train,predicted_labels
_train[:,1],tr_tpr,tr_fpr,t2)
t1="confusion matrix for test data"
draw_confusion_matrix(trainedmulNB_TFIDF,tr_threshold,Y_test,predicted_labels_
test[:,1],te_tpr,te_fpr,t1)
```




```
Train AUC = 0.795
Test AUC = 0.682
```

```
Wall time: 19.8 s
```





2.4.2.1 Top 10 important features of positive class from SET 2

In [77]:
```
#Getting feature index which will contribute more toward classifying positive
 class
pos_class_prob=trainedmulNB_TFIDF.feature_log_prob_[1,:]
#applying argsort() on posative class probabilities
pos_class_prob_index=pos_class_prob.argsort()
# getting top 20 features
top_20_positive_features_index=pos_class_prob_index[-1:-21:-1]
#printing top 20 features
print(top_20_positive_features_index)
#list(BOW_features[top_20_positive_features_index])
print("Top 20 features for positive class")
print("="*40)
print("index\tfeatures")
```

```
for index in top_20_positive_features_index:
    print((index),(TFIDF_features[index]))
```

```
[   92    59    99    58    93    96    89    88    87    97     4 12455
    57    56    86    55    98    85    91    54]
Top 20 features for positive class
========================================
index   features
92 Mrs
59 Literacy_Language
99 Grades_PreK_2
58 Math_Science
93 Ms
96 Grades_3_5
89 Literacy
88 Mathematics
87 Literature_Writing
97 Grades_6_8
4 CA
12455 students
57 Health_Sports
56 SpecialNeeds
86 SpecialNeeds
55 AppliedLearning
98 Grades_9_12
85 AppliedSciences
91 Mr
54 Music_Arts
```

2.4.2.2 Top 10 important features of negative class from SET 2

In [82]:
```
#Getting feature index which will contribute more toward classifying negative
 class
neg_class_prob=trainedmulNB_TFIDF.feature_log_prob_[0,:]
#applying argsort() on negative class probabilities
neg_class_prob_index=neg_class_prob.argsort()
# getting top 20 features
top_20_negative_features_index=neg_class_prob_index[1:21]
#printing top 20 features
print(top_20_negative_features_index)
#list(BOW_features[top_20_negative_features_index])
print("Top 20 features for negative class")
print("="*40)
print("index\tfeatures")
```

```
for index in top_20_negative_features_index:
    print(index,TFIDF_features[index])
```

```
[ 5426 14146 10516   698  5429 10515  3648 13808 12889 16730 14143  3650
 12039 11335 11336 11338 13226  4179  2229 13225]
Top 20 features for negative class
=========================================
index   features
5426 doubles
14146 65
10516 proximal
698 epic
5429 doubts
10515 prowess
3648 busses
13808 wedges
12889 thesaurus
16730 position
14143 500
3650 bustle
12039 sooner
11335 rubik
11336 rubric
11338 ruby
13226 triumphs
4179 colds
2229 580
13225 triumphant
```

In [83]:
```
# Please compare all your models using Prettytable library
from prettytable import PrettyTable

table = PrettyTable()

table.field_names = ["Vectorizer", "Hyper parameter", "Test AUC"]

table.add_row(["BOW", "1","0.71" ])
table.add_row(["TFIDF", "0.5", "0.68"])
print(table)
```

```
+------------+-----------------+----------+
| Vectorizer | Hyper parameter | Test AUC |
+------------+-----------------+----------+
|    BOW     |        1        |   0.71   |
|   TFIDF    |       0.5       |   0.68   |
+------------+-----------------+----------+
```