Assignment 5: Logistic Regression

1. **[Task-1] Logistic Regression(either SGDClassifier with log loss, or LogisticRegression) on these feature sets**
    - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_eassay (`BOW with bi-grams` with `min_df=10` and `max_features=5000`)
    - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_eassay (`TFIDF with bi-grams` with `min_df=10` and `max_features=5000`)
    - Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V)
    - Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)

2. **Hyper paramter tuning (find best hyper parameters corresponding the algorithm that you choose)**
    - Find the best hyper parameter which will give the maximum AUC value
    - 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. **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.
    - 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.

4. **[Task-2] Apply Logistic Regression on the below feature set Set 5 by finding the best hyper parameter as suggested in step 2 and step 3.**
5. Consider these set of features Set 5 :
    - **school_state** : categorical data
    - **clean_categories** : categorical data
    - **clean_subcategories** : categorical data
    - **project_grade_category** :categorical data
    - **teacher_prefix** : categorical data
    - **quantity** : numerical data
    - **teacher_number_of_previously_posted_projects** : numerical data
    - **price** : numerical data
    - **sentiment score's of each of the essay** : numerical data
    - **number of words in the title** : numerical data
    - **number of words in the combine essays** : numerical data

    And apply the Logistic regression on these features by finding the best hyper paramter as suggested in step 2 and step 3

6. **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

In [2]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

In [3]:

```
import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [4]:

```
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
```

In [5]:

```
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
```

In [6]:

```
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle
```

In [7]:

```
from tqdm import tqdm
import os
```

In [8]:

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

Loading Data

In [9]:

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

In [10]:

```
print("Number of data points in train data", project_data.shape)
print('-'*75)
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 [11]:

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

```
print(resource_data.columns.values)
resource_data.head(2)
```

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

Out[11]:

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

In [12]:

```
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 [13]:

```
print("Number of data points in train data", project_data.shape)
print('-'*75)
print("The attributes of data :", project_data.columns.values)
```

```
Number of data points in train data (109248, 19)
---------------------------------------------------------------------------
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'
 'price' 'quantity']
```

Preprocessing Categorical Data

project_grade_category

In [14]:

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

Out[14]:

```
Grades PreK-2    44225
Grades 3-5       37137
Grades 6-8       16923
Grades 9-12      10963
Name: project_grade_category, dtype: int64
```

In [15]:

```
# We need to remove the spaces, replace '-' with '_' and convert all the letters to lowercase
# https://stackoverflow.com/questions/36383821/pandas-dataframe-apply-function-to-column-strings-b
ased-on-other-column-value
project_data['project_grade_category'] = project_data['project_grade_category'].str.replace(' ', '_
')
project_data['project_grade_category'] = project_data['project_grade_category'].str.replace('-', '_
')
project_data['project_grade_category'] = project_data['project_grade_category'].str.lower()
project_data['project_grade_category'].value_counts()
```

Out[15]:

```
grades_prek_2    44225
grades_3_5       37137
grades_6_8       16923
```

```
grades_6_8       16923
grades_9_12      10963
Name: project_grade_category, dtype: int64
```

project_subject_categories

In [16]:

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

Out[16]:

```
Literacy & Language                            23655
Math & Science                                 17072
Literacy & Language, Math & Science            14636
Health & Sports                                10177
Music & The Arts                                5180
Special Needs                                   4226
Literacy & Language, Special Needs              3961
Applied Learning                                3771
Math & Science, Literacy & Language             2289
Applied Learning, Literacy & Language           2191
History & Civics                                1851
Math & Science, Special Needs                   1840
Literacy & Language, Music & The Arts           1757
Math & Science, Music & The Arts                1642
Applied Learning, Special Needs                 1467
History & Civics, Literacy & Language           1421
Health & Sports, Special Needs                  1391
Warmth, Care & Hunger                           1309
Math & Science, Applied Learning                1220
Applied Learning, Math & Science                1052
Literacy & Language, History & Civics            809
Health & Sports, Literacy & Language             803
Applied Learning, Music & The Arts               758
Math & Science, History & Civics                 652
Literacy & Language, Applied Learning            636
Applied Learning, Health & Sports                608
Math & Science, Health & Sports                  414
History & Civics, Math & Science                 322
History & Civics, Music & The Arts               312
Special Needs, Music & The Arts                  302
Health & Sports, Math & Science                  271
History & Civics, Special Needs                  252
Health & Sports, Applied Learning                192
Applied Learning, History & Civics               178
Health & Sports, Music & The Arts                155
Music & The Arts, Special Needs                  138
Literacy & Language, Health & Sports              72
Health & Sports, History & Civics                 43
History & Civics, Applied Learning                42
Special Needs, Health & Sports                    42
Health & Sports, Warmth, Care & Hunger            23
Special Needs, Warmth, Care & Hunger              23
Music & The Arts, Health & Sports                 19
Music & The Arts, History & Civics                18
History & Civics, Health & Sports                 13
Math & Science, Warmth, Care & Hunger             11
Music & The Arts, Applied Learning                10
Applied Learning, Warmth, Care & Hunger           10
Literacy & Language, Warmth, Care & Hunger         9
Music & The Arts, Warmth, Care & Hunger            2
History & Civics, Warmth, Care & Hunger            1
Name: project_subject_categories, dtype: int64
```

In [17]:

```
# we need to remove spaces and 'The'
# replace space with '_' , '&' with '_' and ',' with '_'
project_data['project_subject_categories'] =
project_data['project_subject_categories'].str.replace(' The ','')
project_data['project_subject_categories'] =
project_data['project_subject_categories'].str.replace(' ','')
project_data['project_subject_categories'] =
```

```
project_data['project_subject_categories'] =
project_data['project_subject_categories'].str.replace('&','_')
project_data['project_subject_categories'] =
project_data['project_subject_categories'].str.replace(',','_')
project_data['project_subject_categories'] = project_data['project_subject_categories'].str.lower(
)
project_data['project_subject_categories'].value_counts()
```

Out[17]:

```
literacy_language                          23655
math_science                               17072
literacy_language_math_science             14636
health_sports                              10177
music_arts                                  5180
specialneeds                                4226
literacy_language_specialneeds              3961
appliedlearning                             3771
math_science_literacy_language              2289
appliedlearning_literacy_language           2191
history_civics                              1851
math_science_specialneeds                   1840
literacy_language_music_arts                1757
math_science_music_arts                     1642
appliedlearning_specialneeds                1467
history_civics_literacy_language            1421
health_sports_specialneeds                  1391
warmth_care_hunger                          1309
math_science_appliedlearning                1220
appliedlearning_math_science                1052
literacy_language_history_civics             809
health_sports_literacy_language              803
appliedlearning_music_arts                   758
math_science_history_civics                  652
literacy_language_appliedlearning            636
appliedlearning_health_sports                608
math_science_health_sports                   414
history_civics_math_science                  322
history_civics_music_arts                    312
specialneeds_music_arts                      302
health_sports_math_science                   271
history_civics_specialneeds                  252
health_sports_appliedlearning                192
appliedlearning_history_civics               178
health_sports_music_arts                     155
music_arts_specialneeds                      138
literacy_language_health_sports               72
health_sports_history_civics                  43
history_civics_appliedlearning                42
specialneeds_health_sports                    42
health_sports_warmth_care_hunger              23
specialneeds_warmth_care_hunger               23
music_arts_health_sports                      19
music_arts_history_civics                     18
history_civics_health_sports                  13
math_science_warmth_care_hunger               11
appliedlearning_warmth_care_hunger            10
music_arts_appliedlearning                    10
literacy_language_warmth_care_hunger           9
music_arts_warmth_care_hunger                  2
history_civics_warmth_care_hunger              1
Name: project_subject_categories, dtype: int64
```

teacher_prefix

In [18]:

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

Out[18]:

```
Mrs.        57269
Ms.         38955
Mr.         10648
```

```
Teacher     2360
Dr.           13
Name: teacher_prefix, dtype: int64
```

In [19]:

```
# check if we have any nan values are there
print(project_data['teacher_prefix'].isnull().values.any())
print("number of nan values",project_data['teacher_prefix'].isnull().values.sum())
```

```
True
number of nan values 3
```

In [20]:

```
# number of missing values are very less, we can replace it with Mrs. as
# most of the projects are submitted by Mrs.
project_data['teacher_prefix']=project_data['teacher_prefix'].fillna('Mrs.')
project_data['teacher_prefix'].value_counts()
```

Out[20]:

```
Mrs.       57272
Ms.        38955
Mr.        10648
Teacher     2360
Dr.           13
Name: teacher_prefix, dtype: int64
```

In [21]:

```
# remove '.' and convert all the characters to lowercase
project_data['teacher_prefix'] = project_data['teacher_prefix'].str.replace('.','')
project_data['teacher_prefix'] = project_data['teacher_prefix'].str.lower()
project_data['teacher_prefix'].value_counts()
```

Out[21]:

```
mrs        57272
ms         38955
mr         10648
teacher     2360
dr            13
Name: teacher_prefix, dtype: int64
```

project_subject_subcategories

In [22]:

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

Out[22]:

```
Literacy                                     9486
Literacy, Mathematics                        8325
Literature & Writing, Mathematics            5923
Literacy, Literature & Writing               5571
Mathematics                                  5379
                                             ...
Literature & Writing, Nutrition Education       1
Economics, Nutrition Education                  1
Gym & Fitness, Warmth, Care & Hunger            1
Parent Involvement, Team Sports                 1
ESL, Team Sports                                1
Name: project_subject_subcategories, Length: 401, dtype: int64
```

In [23]:

```
# we need to remove spaces and 'The'
# replace space with '_' , '&' with '_' and ',' with '_'
```

```
project_data['project_subject_subcategories'] = project_data['project_subject_subcategories'].str.
replace(' The ','')
project_data['project_subject_subcategories'] = project_data['project_subject_subcategories'].str.
replace(' ','')
project_data['project_subject_subcategories'] = project_data['project_subject_subcategories'].str.
replace('&','_')
project_data['project_subject_subcategories'] = project_data['project_subject_subcategories'].str.
replace(',','_')
project_data['project_subject_subcategories'] = project_data['project_subject_subcategories'].str.
lower()
project_data['project_subject_subcategories'].value_counts()
```

Out[23]:

```
literacy                                   9486
literacy_mathematics                       8325
literature_writing_mathematics             5923
literacy_literature_writing                5571
mathematics                                5379
                                           ...
esl_economics                                 1
communityservice_gym_fitness                  1
economics_foreignlanguages                    1
college_careerprep_warmth_care_hunger         1
gym_fitness_warmth_care_hunger                1
Name: project_subject_subcategories, Length: 401, dtype: int64
```

school_state

In [40]:

```
project_data['school_state'].value_counts(5)
```

Out[40]:

```
ca    0.140854
tx    0.067699
ny    0.066985
fl    0.056614
nc    0.046600
il    0.039818
ga    0.036275
sc    0.036028
mi    0.028934
pa    0.028458
in    0.023982
mo    0.023579
oh    0.022582
la    0.021913
ma    0.021868
wa    0.021364
ok    0.020833
nj    0.020476
az    0.019653
va    0.018719
wi    0.016723
al    0.016128
ut    0.015845
tn    0.015451
ct    0.015222
md    0.013858
nv    0.012513
ms    0.012110
ky    0.011936
or    0.011369
mn    0.011057
co    0.010170
ar    0.009602
id    0.006343
ia    0.006096
ks    0.005803
nm    0.005098
dc    0.004723
hi    0.004641
```

```
hi     0.004641
me     0.004623
wv     0.004604
nh     0.003185
ak     0.003158
de     0.003140
ne     0.002828
sd     0.002746
ri     0.002609
mt     0.002243
nd     0.001309
wy     0.000897
vt     0.000732
Name: school_state, dtype: float64
```

In [25]:

```
#convert all of them to lowercase
project_data['school_state'] = project_data['school_state'].str.lower()
project_data['school_state'].value_counts()
```

Out[25]:

```
ca    15388
tx     7396
ny     7318
fl     6185
nc     5091
il     4350
ga     3963
sc     3936
mi     3161
pa     3109
in     2620
mo     2576
oh     2467
la     2394
ma     2389
wa     2334
ok     2276
nj     2237
az     2147
va     2045
wi     1827
al     1762
ut     1731
tn     1688
ct     1663
md     1514
nv     1367
ms     1323
ky     1304
or     1242
mn     1208
co     1111
ar     1049
id      693
ia      666
ks      634
nm      557
dc      516
hi      507
me      505
wv      503
nh      348
ak      345
de      343
ne      309
sd      300
ri      285
mt      245
nd      143
wy       98
vt       80
Name: school_state, dtype: int64
```

Preprocessing Text features

project_title

In [26]:

```
# 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 [27]:

```
# 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 [28]:

```
project_data['project_title'].head(5)
```

Out[28]:

```
0      Educational Support for English Learners at Home
1                 Wanted: Projector for Hungry Learners
2     Soccer Equipment for AWESOME Middle School Stu...
3                                Techie Kindergarteners
4                                Interactive Math Tools
Name: project_title, dtype: object
```

In [29]:

```
print("printing some random reviews")
print(9, project_data['project_title'].values[9])
print(34, project_data['project_title'].values[34])
print(147, project_data['project_title'].values[147])
```

```
printing some random reviews
9 Just For the Love of Reading--\r\nPure Pleasure
34 \"Have A Ball!!!\"
147 Who needs a Chromebook?\r\nWE DO!!
```

In [30]:

```
from tqdm import tqdm
def preprocess_text(text_data):
    preprocessed_text = []
    # tqdm is for printing the status bar
    for sentance in tqdm(text_data):
        sent = decontracted(sentance)
        sent = sent.replace('\\r', ' ')
        sent = sent.replace('\\n', ' ')
        sent = sent.replace('\\"', ' ')
        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_text.append(sent.lower().strip())
    return preprocessed_text
```

In [31]:

```
preprocessed_titles = preprocess_text(project_data['project_title'].values)
```

```
100%|████████████████████████████████████████████████████████████████████████| 109248/109248
[00:05<00:00, 18486.60it/s]
```

In [32]:

```
print("printing some random reviews")
print(9, preprocessed_titles[9])
print(34, preprocessed_titles[34])
print(147, preprocessed_titles[147])
```

```
printing some random reviews
9 love reading pure pleasure
34 ball
147 needs chromebook
```

essay

In [33]:

```
# we will merge the essay columns and then preprocess it
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 [34]:

```
print(9, project_data['essay'].values[9])
print('-'*50)
print(34, project_data['essay'].values[34])
print('-'*50)
print(147, project_data['essay'].values[147])
```

9 Over 95% of my students are on free or reduced lunch. I have a few who are homeless, but despit

```
9 Over 95% of my students are on free or reduced lunch.  I have a few who are homeless, but despit
e that, they come to school with an eagerness to learn.  My students are inquisitive eager
learners who  embrace the challenge of not having great books and other resources  every day.
Many of them are not afforded the opportunity to engage with these big colorful pages of a book on
a regular basis at home and they don't travel to the public library.  \r\nIt is my duty as a
teacher to do all I can to provide each student an opportunity to succeed in every aspect of life.
\r\nReading is Fundamental! My students will read these books over and over again while boosting t
heir comprehension skills. These books will be used for read alouds, partner reading and for
Independent reading. \r\nThey will engage in reading to build their \"Love for Reading\" by
reading for pure enjoyment. They will be introduced to some new authors as well as some old
favorites. I want my students to be ready for the 21st Century and know the pleasure of holding a
good hard back book in hand. There's nothing like a good book to read!  \r\nMy students will soar
in Reading, and more because of your consideration and generous funding contribution. This will he
lp build stamina and prepare for 3rd grade. Thank you so much for reading our proposal!nannan
--------------------------------------------------
34 My students mainly come from extremely low-income families, and the majority of them come from
homes where both parents work full time. Most of my students are at school from 7:30 am to 6:00 pm
(2:30 to 6:00 pm in the after-school program), and they all receive free and reduced meals for bre
akfast and lunch. \r\n\r\n\r\nI want my students to feel  as comfortable in my classroom as they d
o at home. Many of my students take on multiple roles both at home as well as in school. They are
sometimes the caretakers of younger siblings, cooks, babysitters, academics, friends, and most of
all, they are developing who they are going to become as adults.  I consider it an essential part
of my job to model helping others gain knowledge in a positive manner. As a result, I have a commu
nity of students who love helping each other in and outside of the classroom. They consistently lo
ok for opportunities to support each other's learning in a kind and helpful way.I am excited to be
experimenting with alternative seating in my classroom this school year. Studies have shown that g
iving students the option of where they sit in a classroom increases focus as well as motivation.
\r\n\r\nBy allowing students choice in the classroom, they are able to explore and create in a wel
coming environment. Alternative classroom seating has been experimented with more frequently in re
cent years. I believe (along with many others), that every child learns differently. This does not
only apply to how multiplication is memorized, or a paper is written, but applies to the space in
which they are asked to work. I have had students in the past ask \"Can I work in the library? Can
I work on the carpet?\" My answer was always, \"As long as you're learning, you can work wherever
you want!\" \r\n\r\nWith the yoga balls and the lap-desks, I will be able to increase the options
for seating in my classroom and expand its imaginable space.nannan
--------------------------------------------------
147 My students are eager to learn and make their mark on the world.\r\n\r\nThey come from a Title
1 school and need extra love.\r\n\r\nMy fourth grade students are in a high poverty area and still
come to school every day to get their education. I am trying to make it fun and educational for th
em so they can get the most out of their schooling. I created a caring environment for the student
s to bloom! They deserve the best.\r\nThank you!\r\nI am requesting 1 Chromebook to access online
interventions, differentiate instruction, and get extra practice. The Chromebook will be used to s
upplement ELA and math instruction. Students will play ELA and math games that are engaging and fu
n, as well as participate in assignments online. This in turn will help my students improve their
skills. Having a Chromebook in the classroom would not only allow students to use the programs at
their own pace, but would ensure more students are getting adequate time to use the programs. The
online programs have been especially beneficial to my students with special needs. They are able t
o work at their level as well as be challenged with some different materials. This is making these
students more confident in their abilities.\r\n\r\nThe Chromebook would allow my students to have
daily access to computers and increase their computing skills.\r\nThis will change their lives for
the better as they become more successful in school. Having access to technology in the classroom
would help bridge the achievement gap.nannan
```

In [35]:

```python
#we will use the same process as project_title
preprocessed_essays = preprocess_text(project_data['essay'].values)
```

```
100%|████████████████████████████████████████████████████████████████████████████| 109248/109248
[02:14<00:00, 811.33it/s]
```

In [36]:

```python
print(project_data.columns)
print(project_data.shape)
```

```
Index(['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',
       'price', 'quantity', 'essay'],
      dtype='object')
(109248, 20)
```

```
(109248, 20)
```

In [39]:

```python
print(type(preprocessed_essays))
```

```
<class 'list'>
```

In [41]:

```python
project_data['preprocessed_essays'] = preprocessed_essays
project_data.drop(['essay'], axis=1, inplace=True)
project_data.head(2)
```

Out[41]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	pro
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	mrs	in	2016-12-05 13:43:57	grad
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	mr	fl	2016-10-25 09:22:10	grad

In [42]:

```python
project_data['preprocessed_titles'] = preprocessed_titles
project_data.drop(['project_title'], axis=1, inplace=True)
project_data.head(2)
```

Out[42]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	pro
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	mrs	in	2016-12-05 13:43:57	grad
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	mr	fl	2016-10-25 09:22:10	grad

In [43]:

```python
project_data.drop(['project_essay_1'], axis=1, inplace=True)
project_data.drop(['project_essay_2'], axis=1, inplace=True)
project_data.drop(['project_essay_3'], axis=1, inplace=True)
project_data.drop(['project_essay_4'], axis=1, inplace=True)
project_data.drop(['project_resource_summary'], axis=1, inplace=True)
project_data.drop(['Unnamed: 0'], axis=1, inplace=True)
project_data.drop(['id'], axis=1, inplace=True)
project_data.drop(['teacher_id'], axis=1, inplace=True)
project_data.drop(['project_submitted_datetime'], axis=1, inplace=True)

project_data.head(2)
```

Out[43]:

	teacher_prefix	school_state	project_grade_category	project_subject_categories	project_subject_subcategories	teach

	teacher_prefix	school_state	project_grade_category	project_subject_categories	project_subject_subcategories	teach
0	mrs	in	grades_prek_2	literacy_language	esl_literacy	0
1	mr	fl	grades_6_8	history_civics_health_sports	civics_government_teamsports	7

In [44]:

```
print(project_data.shape)
```

```
(109248, 11)
```

In [45]:

```
# creating a preprocessed data so as to use it when required
project_data.to_csv('preprocessed_data1.csv')
```

1.1 Loading Data

In [46]:

```
data = pd.read_csv('preprocessed_data1.csv', nrows = 50000)
data.head(2)
```

Out[46]:

	Unnamed: 0	teacher_prefix	school_state	project_grade_category	project_subject_categories	project_subject_subcatego
0	0	mrs	in	grades_prek_2	literacy_language	esl_literacy
1	1	mr	fl	grades_6_8	history_civics_health_sports	civics_government_teamspo

In [49]:

```
print(data.columns.values)
print(data.shape)
```

```
['Unnamed: 0' 'teacher_prefix' 'school_state' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'teacher_number_of_previously_posted_projects' 'project_is_approved'
 'price' 'quantity' 'preprocessed_essays' 'preprocessed_titles']
(50000, 12)
```

In [48]:

```
y = data['project_is_approved'].values
x = data.drop(['project_is_approved'], axis = 1)
print(x.shape)
x.head(2)
```

```
(50000, 11)
```

Out[48]:

	Unnamed: 0	teacher_prefix	school_state	project_grade_category	project_subject_categories	project_subject_subcatego

	Unnamed: 0	teacher_prefix	school_state	project_grade_category	project_subject_categories	project_subject_subcatego
0	0	mrs	in	grades_prek_2	literacy_language	esl_literacy
1	1	mr	fl	grades_6_8	history_civics_health_sports	civics_government_teamspo

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

In [50]:

```
#Train Test split (Using GridSearchCV)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size= 0.3, stratify = y)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(35000, 11)
(15000, 11)
(35000,)
(15000,)
```

1.3 Make Data Model Ready: encoding text features

Encoding Essay : BOW

In [56]:

```
vectorizer = CountVectorizer(min_df = 10, ngram_range=(1,2), max_features=5000)
# fit should be done only on train data
vectorizer.fit(X_train['preprocessed_essays'].values)
# we use the fitted countvectorizer to convert the text to vector
X_train_essay_bow = vectorizer.transform(X_train['preprocessed_essays'].values)
X_test_essay_bow = vectorizer.transform(X_test['preprocessed_essays'].values)

print("Shape after vectorization")
print(X_train_essay_bow.shape, y_train.shape)
print(X_test_essay_bow.shape, y_test.shape)
```

```
Shape after vectorization
(35000, 5000) (35000,)
(15000, 5000) (15000,)
```

Encoding Essay : TF-IDF

In [58]:

```
vectorizer = TfidfVectorizer(min_df = 10, ngram_range=(1,2), max_features=5000)
# fit should be done only on train data
vectorizer.fit(X_train['preprocessed_essays'].values)

X_train_essay_tfidf = vectorizer.transform(X_train['preprocessed_essays'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['preprocessed_essays'].values)

print("Shape after vectorization")
print(X_train_essay_tfidf.shape, y_train.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
```

```
Shape after vectorization
(35000, 5000) (35000,)
(15000, 5000) (15000,)
```

Encoding Essay : AVG W2V

In [59]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-sa
ve-and-load-variables-in-python/
# make sure you have the glove_vectors file
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [60]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['preprocessed_essays'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_train.append(vector)

print(len(avg_w2v_vectors_train))
print(len(avg_w2v_vectors_train[0]))
```

```
100%|████████████████████████████████████████████████████████████████████████████| 35000/35000
[00:21<00:00, 1617.24it/s]
```

```
35000
300
```

In [61]:

```
avg_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['preprocessed_essays'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_test.append(vector)


print(len(avg_w2v_vectors_test))
print(len(avg_w2v_vectors_test[0]))
print(avg_w2v_vectors_test[0])
```

```
100%|████████████████████████████████████████████████████████████████████████████| 15000/15000
[00:08<00:00, 1712.97it/s]
```

```
15000
300
[ 1.32635203e-02  6.28973872e-02  3.14925086e-02 -1.19027209e-01
  5.59147734e-02 -2.11207246e-02 -2.96726625e+00  4.84867594e-02
 -1.01097091e-01 -8.00504793e-02  2.58621617e-02  4.38902462e-02
  1.32962759e-01 -1.48805212e-01 -5.69888547e-02 -4.02640078e-02
  8.64900000e-03 -7.19350805e-02  4.87913284e-02 -1.04335062e-02
  4.77429648e-02 -1.97969508e-02 -2.12610508e-02 -1.97588109e-02
  1.16535328e-02 -5.05186867e-02  6.87834289e-02 -2.35002586e-02
 -3.50257266e-03  6.63681656e-02 -3.06255042e-01 -6.19839641e-02
 -5.04795313e-02  5.84906070e-02 -6.34922734e-03 -9.96811773e-02
```

```
 -7.76671000e-02  4.90294039e-02 -1.28522878e-01 -8.46569117e-02
  1.68450820e-02  1.28082735e-01  9.37460008e-02 -1.80464599e-01
 -3.42028594e-03 -4.08903773e-02  1.85623632e-01 -1.05226014e-01
 -1.72498836e-02 -5.78458594e-02  4.11750273e-02  3.47688404e-02
 -8.39564088e-02 -4.75552117e-02  1.17988040e-01 -2.45098305e-02
  5.50474219e-03  1.81335937e-03 -7.38944109e-02  1.14888114e-01
  2.70363508e-02 -6.87556382e-02  5.68925992e-02 -2.88252727e-03
 -1.48727180e-02  9.32849148e-02  4.29447401e-02 -7.25413281e-04
  1.11949526e-01 -1.46841380e-01 -1.24585927e-01  8.55450937e-02
  4.75051391e-02 -1.70273023e-02  8.96821328e-03 -1.15465444e-01
  7.21100273e-02 -3.65222031e-03 -3.22946180e-02  1.39792977e-02
  2.73001719e-02 -3.22060653e-01 -5.29504398e-02  1.07447020e-02
 -8.16735223e-02  1.64391109e-02  1.52299127e-01 -2.84161636e-02
  1.01940048e-01 -4.64044189e-02  1.75678495e-02 -9.74046156e-02
 -7.38631424e-02 -2.23738563e-02  3.99123397e-02 -1.39750672e-01
 -2.01581234e+00  3.79224316e-02  1.28256722e-01  3.35382859e-02
 -4.60455219e-02  5.02693797e-02  8.25727734e-02 -4.95217617e-02
  3.66608164e-02  2.47351953e-02  6.75584063e-03 -1.56760372e-01
  4.68661670e-02 -3.36617219e-02 -5.66556406e-02 -4.98236445e-02
 -7.19577187e-03  1.63786937e-01  1.99979566e-02 -4.09423641e-02
 -2.40623108e-01 -1.43986882e-02 -2.33902148e-02  5.48775578e-02
 -3.70798594e-03  4.80756645e-02  1.06372391e-02 -1.18094534e-01
 -2.99272598e-02 -3.30839227e-03  1.19245101e-01 -8.64493984e-04
 -5.73514197e-02 -6.80395124e-02 -1.02065625e-02  1.19572727e-02
  7.54133594e-04 -2.50781836e-02  4.97003422e-02 -6.83558802e-02
 -2.03350547e-02 -8.77492680e-03  8.13981016e-02  4.16791701e-01
  1.26139600e-01 -9.93575547e-03  1.33320148e-02  4.61481789e-02
 -1.29375782e-01  6.04794141e-03  8.15661016e-03 -4.52377703e-02
  1.01587097e-02 -4.17069969e-02 -1.04053610e-01 -3.83184602e-02
  4.88434770e-02 -8.97665344e-02  7.88015547e-02  1.75266635e-02
 -3.09274086e-02 -1.94229539e-02 -3.89657465e-02 -2.66563617e-02
  4.15954672e-02  1.07159219e-03 -3.18408156e-02 -8.36350391e-03
 -4.76536086e-02 -7.19420328e-02 -1.22740039e-02  6.89487500e-03
  1.83144472e-01 -1.68020538e-01 -9.10061586e-02 -3.02307592e-02
  4.38108281e-03 -1.17663594e-01 -1.20225288e-02  3.83796727e-02
  7.18962109e-02  6.54363148e-02 -8.41482836e-02 -7.74657750e-02
  5.94364609e-02  1.55707791e-01  9.53600141e-02  2.27322789e-02
  2.10144969e-02 -7.72265555e-02 -5.83665242e-02  7.87225586e-02
  3.69872750e-02 -5.16937502e-02  2.05089111e-02 -2.52709531e-02
 -4.91324989e-02 -1.53885467e-01 -4.04820688e-02 -8.92311656e-02
 -6.36658437e-03  7.70220522e-02 -3.41514437e-02  5.71410620e-02
  1.51328581e-01 -8.76068281e-03 -4.03867539e-02  8.07691430e-02
 -1.13218587e-01  7.58385703e-02  3.66625594e-02 -1.03906255e-01
  9.46708281e-02 -2.93966336e-02  5.93476016e-03 -4.60596414e-02
 -5.74824906e-02 -1.30350298e-01  3.36196699e-02 -5.05752578e-02
 -9.84690344e-02 -4.52848547e-02  5.72747609e-02 -8.51141383e-02
 -5.86080031e-02 -1.63701867e-02 -1.60317852e-01 -1.00159328e-02
 -2.18887129e+00  9.18300625e-02 -3.31608203e-02  6.23289016e-02
  4.37766563e-02 -5.33202359e-02  2.22208117e-02 -1.45312437e-02
 -8.89768258e-02 -3.96396937e-02 -1.10311294e-01  1.08048194e-01
  2.07270780e-02 -1.14573859e-02  8.53841406e-03  5.40764070e-02
 -4.88667453e-02  9.75909364e-02 -1.84397942e-01  2.11230867e-01
 -5.55228867e-02 -6.87850547e-03 -1.12661985e-01  8.66879063e-03
  5.21346141e-02 -6.21807672e-03  1.49753941e-02 -1.14021626e-01
  1.20138594e-02 -4.80948639e-02  3.84125541e-02 -4.86184555e-02
  1.20394426e-01 -5.50398222e-02  1.52200000e-02  4.60216469e-02
  2.20623125e-02  2.19706797e-02 -1.05964766e-02 -8.48626672e-02
  1.53488763e-01 -7.46075961e-02 -1.67570615e-01 -1.50602547e-02
  9.48059297e-03  9.11100805e-02 -6.57416519e-02 -8.85630141e-02
 -1.80119617e-02  3.21748594e-02 -1.83609375e-04 -2.17240539e-02
  9.35732641e-02  2.60421178e-02  7.73558125e-03  4.79220859e-02
  2.49802028e-01 -1.17833908e-01 -1.98931344e-02  6.64572498e-02
  1.54701500e-02  1.84673102e-01  4.58783125e-03  5.30087289e-02
 -5.27100359e-02 -6.57981039e-02  4.42322672e-02  2.24040909e-02
 -7.06790078e-02 -7.93999273e-02 -5.19446359e-02  1.86795562e-02
 -2.16950134e-02  6.51086250e-03  2.69527930e-02  2.55086094e-03]
```

Encoding Essay : TFIDF W2V

In [62]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['preprocessed_essays'].values)
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
```

```
tfidf_words = set(tfidf_model.get_feature_names())
```

In [63]:

```
tfidf_w2v_vectors_train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['preprocessed_essays'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_train.append(vector)

print(len(tfidf_w2v_vectors_train))
print(len(tfidf_w2v_vectors_train[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 35000/35000 [02:
34<00:00, 226.38it/s]
```

```
35000
300
```

In [64]:

```
tfidf_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['preprocessed_essays'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_test.append(vector)

print(len(tfidf_w2v_vectors_test))
print(len(tfidf_w2v_vectors_test[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 15000/15000 [01:
05<00:00, 228.93it/s]
```

```
15000
300
```

Encoding Project title : BOW

In [66]:

```
# check if we have any nan values are there
print(X_train['preprocessed_titles'].isnull().values.any())
print("number of nan values",X_train['preprocessed_titles'].isnull().values.sum())
```

```
True
number of nan values 15
```

In [70]:

```
X_train['preprocessed_titles'].value_counts()
```

Out[70]:

```
wiggle work                            133
flexible seating                       101
like move move                          76
hear                                    59
wobble work                             54
                                      ...
help us learn movement                   1
learning listen listening learn          1
alternative eco firendly seating         1
nonfiction reading scholastic news       1
weeble wobbles calm                      1
Name: preprocessed_titles, Length: 31215, dtype: int64
```

In [73]:

```
# replacing missing values with literacy language project as title because
# this has most number of projects belong to this category
X_train['preprocessed_titles']=X_train['preprocessed_titles'].fillna('literacy language project')
print(X_train['preprocessed_titles'].isnull().values.any())
```

```
False
```

In [72]:

```
print(X_test['preprocessed_titles'].isnull().values.any())
print("number of nan values",X_test['preprocessed_titles'].isnull().values.sum())
```

```
True
number of nan values 8
```

In [74]:

```
# replacing missing values with literacy language project as title because
# this has most number of projects belong to this category
X_test['preprocessed_titles']=X_test['preprocessed_titles'].fillna('literacy language project')
print(X_test['preprocessed_titles'].isnull().values.any())
```

```
False
```

In [75]:

```
vectorizer = CountVectorizer(min_df = 10, ngram_range=(1,1), max_features=5000)
# fit should be done only on train data
vectorizer.fit(X_train['preprocessed_titles'].values)
# we use the fitted countvectorizer to convert the text to vector
X_train_titles_bow = vectorizer.transform(X_train['preprocessed_titles'].values)
X_test_titles_bow = vectorizer.transform(X_test['preprocessed_titles'].values)

print("Shape after vectorization")
print(X_train_titles_bow.shape, y_train.shape)
print(X_test_titles_bow.shape, y_test.shape)
```

```
Shape after vectorization
(35000, 1608) (35000,)
(15000, 1608) (15000,)
```

Encoding Project title : TFIDF

In [76]:

```
vectorizer = TfidfVectorizer(min_df = 10)
# fit should be done only on train data
vectorizer.fit(X_train['preprocessed_titles'].values)

X_train_titles_tfidf = vectorizer.transform(X_train['preprocessed_titles'].values)
X_test_titles_tfidf = vectorizer.transform(X_test['preprocessed_titles'].values)

print("Shape after vectorization")
print(X_train_titles_tfidf.shape, y_train.shape)
print(X_train_titles_tfidf.shape, y_test.shape)
```

```
Shape after vectorization
(35000, 1608) (35000,)
(35000, 1608) (15000,)
```

Encoding Project title : AVG W2V

In [77]:

```
avg_w2v_vectors_train_titles = [];
for sentence in tqdm(X_train['preprocessed_titles'].values):
    vector = np.zeros(300)
    cnt_words =0;
    for word in sentence.split():
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_train_titles.append(vector)

print(len(avg_w2v_vectors_train_titles))
print(len(avg_w2v_vectors_train_titles[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 35000/35000
[00:02<00:00, 13288.90it/s]
```

```
35000
300
```

In [78]:

```
avg_w2v_vectors_test_titles = [];
for sentence in tqdm(X_test['preprocessed_titles'].values):
    vector = np.zeros(300)
    cnt_words =0;
    for word in sentence.split():
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_vectors_test_titles.append(vector)


print(len(avg_w2v_vectors_test_titles))
print(len(avg_w2v_vectors_test_titles[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 15000/15000
[00:00<00:00, 25338.77it/s]
```

```
15000
300
```

Encoding Project title : TFIDF W2V

In [79]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['preprocessed_titles'].values)
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [80]:

```
tfidf_w2v_vectors_train_titles = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['preprocessed_titles'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_train_titles.append(vector)

print(len(tfidf_w2v_vectors_train_titles))
print(len(tfidf_w2v_vectors_train_titles[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 35000/35000
[00:02<00:00, 16343.88it/s]
```

```
35000
300
```

In [81]:

```
tfidf_w2v_vectors_test_titles = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['preprocessed_titles'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_test_titles.append(vector)

print(len(tfidf_w2v_vectors_test_titles))
print(len(tfidf_w2v_vectors_test_titles[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 15000/15000
[00:00<00:00, 17481.14it/s]
```

```
15000
300
```

1.4 Make Data Model Ready: encoding numerical, categorical features

encoding categorical features: School State

In [82]:

In [82]:

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

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

print('After Vectorizations:')
print(X_train_state_ohe.shape, y_train.shape)
print(X_test_state_ohe.shape, y_test.shape)
```

```
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'k
s', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'mo', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm',
'nv', 'ny', 'oh', 'ok', 'or', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv
', 'wy']
After Vectorizations:
(35000, 51) (35000,)
(15000, 51) (15000,)
```

encoding categorical features: teacher_prefix

In [83]:

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

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

print("After vectorizations")
print(X_train_teacher_ohe.shape, y_train.shape)
print(X_test_teacher_ohe.shape, y_test.shape)
```

```
['dr', 'mr', 'mrs', 'ms', 'teacher']
After vectorizations
(35000, 5) (35000,)
(15000, 5) (15000,)
```

encoding categorical features: project_grade_category

In [84]:

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

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


print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
print(X_test_grade_ohe.shape, y_test.shape)
```

```
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
After vectorizations
(35000, 4) (35000,)
(15000, 4) (15000,)
```

encoding categorical features: project_subject_categories

encoding categorical features: project_subject_categories

In [89]:

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

# we use the fitted CountVectorizer to convert the text to vector
X_train_project_categories_ohe = vectorizer.transform(X_train['project_subject_categories'].values
)
X_test_project_categories_ohe = vectorizer.transform(X_test['project_subject_categories'].values)

print("After vectorizations")
print(X_train_project_categories_ohe.shape, y_train.shape)
print(X_test_project_categories_ohe.shape, y_test.shape)
```

```
After vectorizations
(35000, 49) (35000,)
(15000, 49) (15000,)
```

encoding categorical features: project_subject_subcategories

In [88]:

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

# we use the fitted CountVectorizer to convert the text to vector
X_train_project_subcategories_ohe = vectorizer.transform(X_train['project_subject_subcategories'].
values)
X_test_project_subcategories_ohe = vectorizer.transform(X_test['project_subject_subcategories'].va
lues)

print("After vectorizations")
print(X_train_project_subcategories_ohe.shape, y_train.shape)
print(X_test_project_subcategories_ohe.shape, y_test.shape)
```

```
After vectorizations
(35000, 366) (35000,)
(15000, 366) (15000,)
```

encoding numerical features: price

In [90]:

```
print(X_train['price'].value_counts())
```

```
479.00    157
149.99    155
269.99    149
399.99    148
49.99     128
         ...
438.40      1
197.56      1
128.66      1
272.91      1
139.70      1
Name: price, Length: 22666, dtype: int64
```

In [91]:

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

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

```
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(1,-1))

print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)
print(X_test_price_norm.shape, y_test.shape)
```

```
After vectorizations
(1, 35000) (35000,)
(1, 15000) (15000,)
```

In [92]:

```
#print(X_train_price_norm)
X_train_price_norm= X_train_price_norm.reshape(-1, 1)
X_test_price_norm = X_test_price_norm.reshape(-1, 1)
print(X_train_price_norm.shape)
print(X_test_price_norm.shape)
```

```
(35000, 1)
(15000, 1)
```

teacher_number_of_previously_posted_projects

In [93]:

```
print(X_train['teacher_number_of_previously_posted_projects'].value_counts())
```

```
0      9534
1      5104
2      3291
3      2229
4      1706
       ...
336       1
304       1
272       1
240       1
271       1
Name: teacher_number_of_previously_posted_projects, Length: 300, dtype: int64
```

In [94]:

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

X_train_teacher_number_of_previously_posted_projects_norm =
normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
X_test_teacher_number_of_previously_posted_projects_norm =
normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))

print("After vectorizations")
print(X_train_teacher_number_of_previously_posted_projects_norm.shape, y_train.shape)
print(X_test_teacher_number_of_previously_posted_projects_norm.shape, y_test.shape)
```

```
After vectorizations
(1, 35000) (35000,)
(1, 15000) (15000,)
```

In [95]:

```
print(X_train_teacher_number_of_previously_posted_projects_norm)
```

```
[[0.00017625 0.0003525  0.0003525  ... 0.         0.         0.00017625]]
```

In [96]:

```
#reshape
```

```
X_train_teacher_number_of_previously_posted_projects_norm=
X_train_teacher_number_of_previously_posted_projects_norm.reshape(-1, 1)
X_test_teacher_number_of_previously_posted_projects_norm =
X_test_teacher_number_of_previously_posted_projects_norm.reshape(-1, 1)
print(X_train_teacher_number_of_previously_posted_projects_norm.shape)
print(X_test_teacher_number_of_previously_posted_projects_norm.shape)
```

```
(35000, 1)
(15000, 1)
```

Appling Logistic Regression on different kind of featurization as mentioned in the instructions

Apply Logistic Regression 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

Set 1: categorical, numerical features + project_title(BOW) + preprocessed_eassay (BOW with bi-grams with min_df=10 and max_features=5000)

In [97]:

```
from scipy.sparse import hstack
X_tr = hstack((X_train_titles_bow, X_train_essay_bow,
X_train_teacher_number_of_previously_posted_projects_norm, X_train_price_norm,
X_train_project_categories_ohe, X_train_project_subcategories_ohe, X_train_grade_ohe,
X_train_teacher_ohe, X_train_state_ohe)).tocsr()
X_te = hstack((X_test_titles_bow, X_test_essay_bow,
X_test_teacher_number_of_previously_posted_projects_norm, X_test_price_norm,
X_test_project_categories_ohe, X_test_project_subcategories_ohe, X_test_grade_ohe,
X_test_teacher_ohe, X_test_state_ohe)).tocsr()

print("Final Data Matrix:")
print(X_tr.shape, y_train.shape)
print(X_te.shape, y_test.shape)
```

```
Final Data Matrix:
(35000, 7085) (35000,)
(15000, 7085) (15000,)
```

Hyperparameter tuning using GridSearchCV

In [104]:

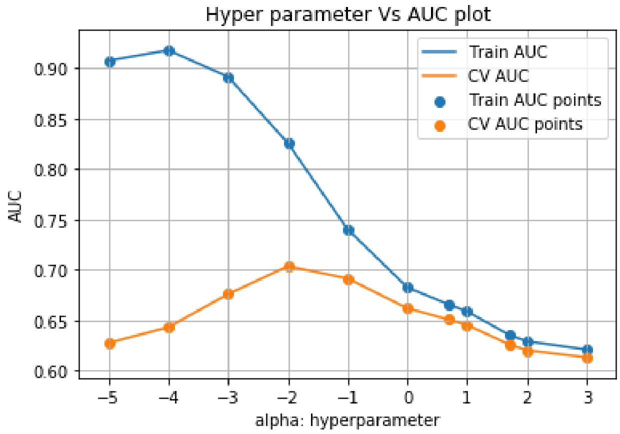
```
import math
from sklearn.model_selection import GridSearchCV
from sklearn import linear_model

lr = linear_model.SGDClassifier(loss='log', penalty='l2', class_weight = 'balanced')
parameters = {'alpha':[0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 5, 10, 50, 100, 1000]}
clf = GridSearchCV(lr, parameters, cv=5, scoring='roc_auc', return_train_score = True)
clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
alpha =  clf.cv_results_['param_alpha']
print(alpha)
```

```
[1e-05 0.0001 0.001 0.01 0.1 1 5 10 50 100 1000]
```

In [105]:

```
#use log10(alpha) on X axis while plotting ROC vs hyperparam plot, as it allows us to understand w
hat's happening in a better way
log_alpha = []
for a in tqdm(alpha):
    b=math.log10(a)
    log_alpha.append(b)

plt.plot(log_alpha, train_auc, label='Train AUC')
plt.plot(log_alpha, cv_auc, label='CV AUC')

plt.scatter(log_alpha, train_auc, label='Train AUC points')
plt.scatter(log_alpha, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 11/11
[00:00<00:00, 22128.22it/s]
```



Summary:

Using gridsearchcv with cv=5 we got the best value of our hyperparameter as 0.1.

Testing the performance of the model on test data, plotting ROC Curves

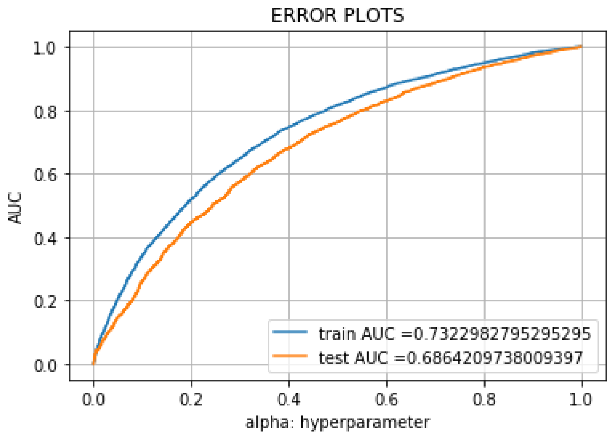
In [106]:

```
def pred_prob(clf, data):
    y_pred = []
    y_pred = clf.predict_proba(data)[:,1]
    return y_pred
```

In [110]:

```
from sklearn.metrics import roc_curve, auc

lr = linear_model.SGDClassifier(loss='log', penalty='l2', alpha=0.1, class_weight = 'balanced')
lr.fit(X_tr, y_train)

y_train_pred = pred_prob(lr, X_tr)
y_test_pred = pred_prob(lr, X_te)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
```

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



Summary:

From the above plot, we observe that at alpha=0.01 we get the train-AUC of 0.732 and test-AUC of 0.686.

In [111]:

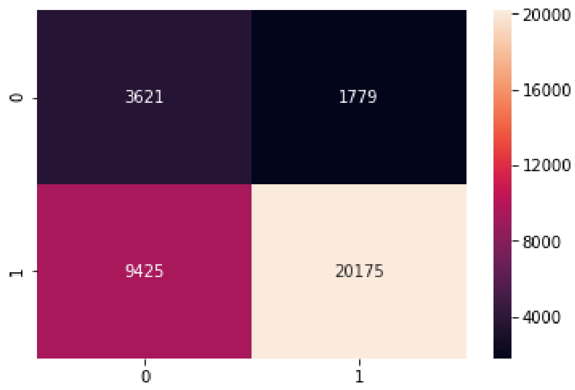
```
#we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [112]:

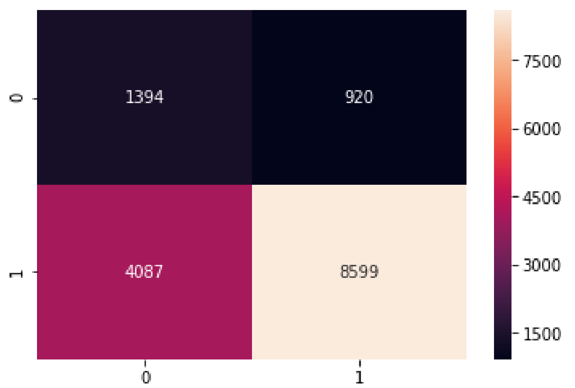
```
from sklearn.metrics import confusion_matrix
fig = plt.figure()
ax = fig.add_subplot(111)
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print('-'*75)
print("Train confusion matrix")
cm = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
sns.heatmap(cm, annot=True, fmt='d')

plt.show(ax)


fig = plt.figure()
ax1 = fig.add_subplot(111)
print('-'*75)
print("Test confusion matrix")
cm = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
sns.heatmap(cm, annot=True, fmt='d')

plt.show(ax1)
```

```
the maximum value of tpr*(1-fpr) 0.4570425112612613 for threshold 0.49
---------------------------------------------------------------------------
```

```
Train confusion matrix
```




```
--------------------------------------------------------------------------
Test confusion matrix
```



In [113]:

```
#https://towardsdatascience.com/demystifying-confusion-matrix-confusion-9e82201592fd
tn, fp, fn, tp = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)).ravel()
print("True Negatives: ",tn)
print("False Positives: ",fp)
print("False Negatives: ",fn)
print("True Positives: ",tp)
```

```
True Negatives:  1394
False Positives:  920
False Negatives:  4087
True Positives:  8599
```

Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF with bi-grams with min_df=10 and max_features=5000)

In [114]:

```
from scipy.sparse import hstack
X_tr = hstack((X_train_titles_tfidf, X_train_essay_tfidf,
X_train_teacher_number_of_previously_posted_projects_norm, X_train_price_norm,
X_train_project_categories_ohe, X_train_project_subcategories_ohe, X_train_grade_ohe,
X_train_teacher_ohe, X_train_state_ohe)).tocsr()
X_te = hstack((X_test_titles_tfidf, X_test_essay_tfidf,
X_test_teacher_number_of_previously_posted_projects_norm, X_test_price_norm,
X_test_project_categories_ohe, X_test_project_subcategories_ohe, X_test_grade_ohe,
X_test_teacher_ohe, X_test_state_ohe)).tocsr()

print("Final Data Matrix:")
print(X_tr.shape, y_train.shape)
print(X_te.shape, y_test.shape)
```

```
Final Data Matrix:
(35000, 7085) (35000,)
(15000, 7085) (15000,)
```

Hyperparameter tuning using GridSearchCV

In [123]:

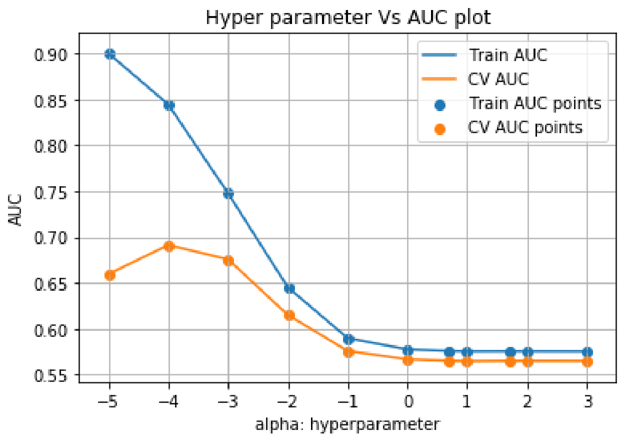
```
import math
from sklearn.model_selection import GridSearchCV
from sklearn import linear_model

lr = linear_model.SGDClassifier(loss='log', penalty='l2', class_weight = 'balanced')
parameters = {'alpha':[0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 5, 10, 50, 100, 1000]}
clf = GridSearchCV(lr, parameters, cv=10, scoring='roc_auc', return_train_score = True)
clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
alpha =  clf.cv_results_['param_alpha']
print(alpha)
```

```
[1e-05 0.0001 0.001 0.01 0.1 1 5 10 50 100 1000]
```

In [124]:

```
#use log10(alpha) on X axis while plotting ROC vs hyperparam plot, as it allows us to understand w
hat's happening in a better way
log_alpha = []
for a in tqdm(alpha):
    b=math.log10(a)
    log_alpha.append(b)

plt.plot(log_alpha, train_auc, label='Train AUC')
plt.plot(log_alpha, cv_auc, label='CV AUC')

plt.scatter(log_alpha, train_auc, label='Train AUC points')
plt.scatter(log_alpha, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
```

```
100%|████████████████████████████████████████████████████████████████████████████████| 11/11
[00:00<00:00, 22117.61it/s]
```



Summary:

Using gridsearchcv with cv=10 we got the best value of our hyperparameter as 0.001.

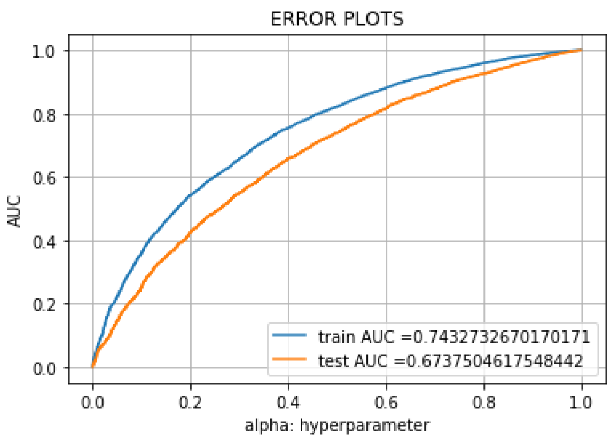
Testing the performance of the model on test data, plotting ROC Curves

In [125]:

```
def pred_prob(clf, data):
    y_pred = []
    y_pred = clf.predict_proba(data)[:,1]
    return y_pred
```

In [127]:

```
from sklearn.metrics import roc_curve, auc

lr = linear_model.SGDClassifier(loss='log', penalty='l2', alpha=0.001, class_weight = 'balanced')
lr.fit(X_tr, y_train)

y_train_pred = pred_prob(lr, X_tr)
y_test_pred = pred_prob(lr, X_te)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



Summary:

From the above plot, we observe that at alpha=0.001 we get the train-AUC of 0.743 and test-AUC of 0.674.

In [128]:

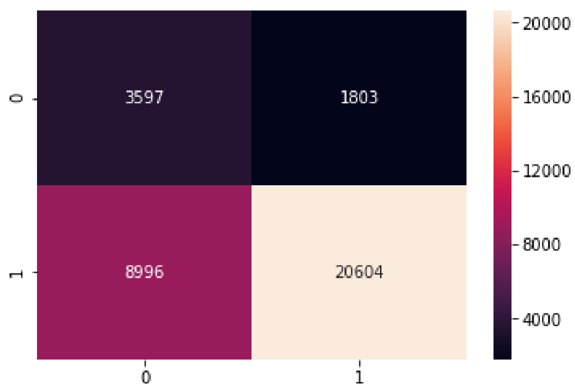
```
#we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
```

```
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

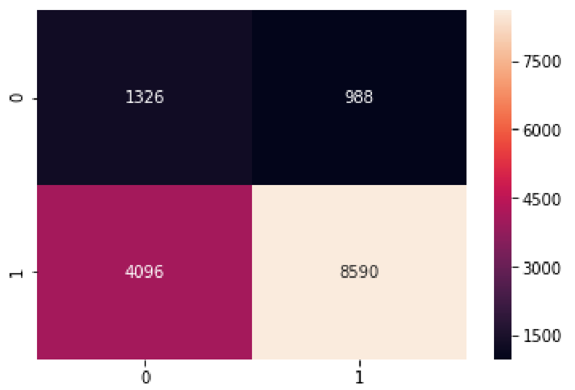
In [129]:

```
from sklearn.metrics import confusion_matrix
fig = plt.figure()
ax = fig.add_subplot(111)
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print('-'*75)
print("Train confusion matrix")
cm = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
sns.heatmap(cm, annot=True, fmt='d')

plt.show(ax)


fig = plt.figure()
ax1 = fig.add_subplot(111)
print('-'*75)
print("Test confusion matrix")
cm = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
sns.heatmap(cm, annot=True, fmt='d')

plt.show(ax1)
```

```
the maximum value of tpr*(1-fpr) 0.46366734234234236 for threshold 0.488
---------------------------------------------------------------------------
Train confusion matrix
```




```
---------------------------------------------------------------------------
Test confusion matrix
```



In [130]:

```
#https://towardsdatascience.com/demystifying-confusion-matrix-confusion-9e82201592fd
tn, fp, fn, tp = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)).ravel()
```

```
print("True Negatives: ",tn)
print("False Positives: ",fp)
print("False Negatives: ",fn)
print("True Positives: ",tp)
```

```
True Negatives:  1326
False Positives:  988
False Negatives:  4096
True Positives:  8590
```

Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V)

In [131]:

```
from scipy.sparse import hstack
X_tr = hstack((avg_w2v_vectors_train_titles, avg_w2v_vectors_train,
X_train_teacher_number_of_previously_posted_projects_norm, X_train_price_norm,
X_train_project_categories_ohe, X_train_project_subcategories_ohe, X_train_grade_ohe,
X_train_teacher_ohe, X_train_state_ohe)).tocsr()
X_te = hstack((avg_w2v_vectors_test_titles, avg_w2v_vectors_test,
X_test_teacher_number_of_previously_posted_projects_norm, X_test_price_norm,
X_test_project_categories_ohe, X_test_project_subcategories_ohe, X_test_grade_ohe,
X_test_teacher_ohe, X_test_state_ohe)).tocsr()

print("Final Data Matrix:")
print(X_tr.shape, y_train.shape)
print(X_te.shape, y_test.shape)
```

```
Final Data Matrix:
(35000, 1077) (35000,)
(15000, 1077) (15000,)
```

Hyperparameter tuning using GridSearchCV

In [132]:

```
import math
from sklearn.model_selection import GridSearchCV
from sklearn import linear_model

lr = linear_model.SGDClassifier(loss='log', penalty='l2', class_weight = 'balanced')
parameters = {'alpha':[0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 5, 10, 50, 100, 1000]}
clf = GridSearchCV(lr, parameters, cv=10, scoring='roc_auc', return_train_score = True)
clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
alpha =  clf.cv_results_['param_alpha']
print(alpha)
```

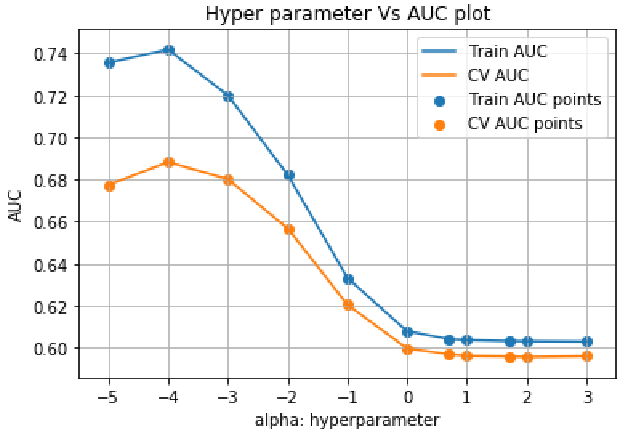
```
[1e-05 0.0001 0.001 0.01 0.1 1 5 10 50 100 1000]
```

In [133]:

```
#use log10(alpha) on X axis while plotting ROC vs hyperparam plot, as it allows us to understand w
hat's happening in a better way
log_alpha = []
for a in tqdm(alpha):
    b=math.log10(a)
    log_alpha.append(b)

plt.plot(log_alpha, train_auc, label='Train AUC')
plt.plot(log_alpha, cv_auc, label='CV AUC')

plt.scatter(log_alpha, train_auc, label='Train AUC points')
plt.scatter(log_alpha, cv_auc, label='CV AUC points')
```

```
plt.scatter(log_alpha, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
```

```
100%|██████████████████████████████████████████████████| 1
1/11 [00:00<?, ?it/s]
```



Summary:

Using gridsearchcv with cv=10 we got the best value of our hyperparameter as 0.001.

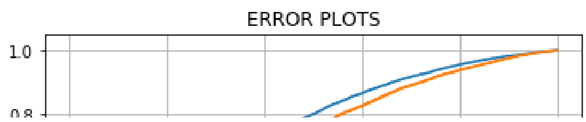
Testing the performance of the model on test data, plotting ROC Curves

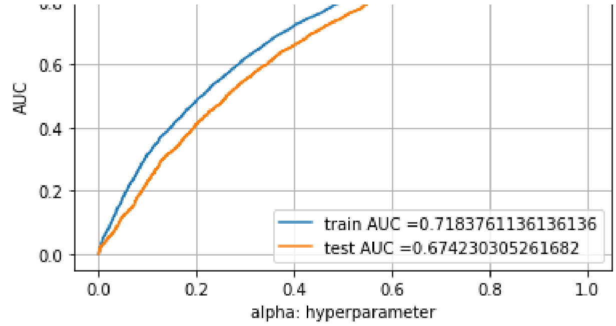
In [134]:

```
def pred_prob(clf, data):
    y_pred = []
    y_pred = clf.predict_proba(data)[:,1]
    return y_pred
```

In [135]:

```
from sklearn.metrics import roc_curve, auc

lr = linear_model.SGDClassifier(loss='log', penalty='l2', alpha=0.001, class_weight = 'balanced')
lr.fit(X_tr, y_train)

y_train_pred = pred_prob(lr, X_tr)
y_test_pred = pred_prob(lr, X_te)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```





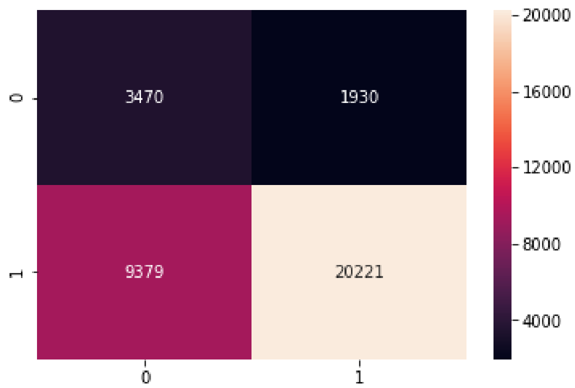
Summary:

From the above plot, we observe that at alpha=0.001 we get the train-AUC of 0.718 and test-AUC of 0.674.

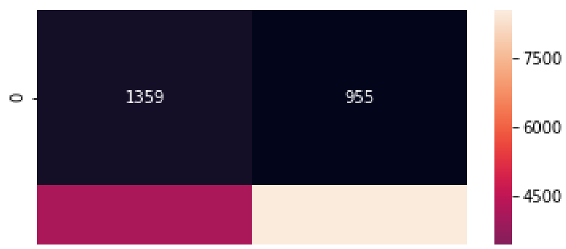
In [136]:

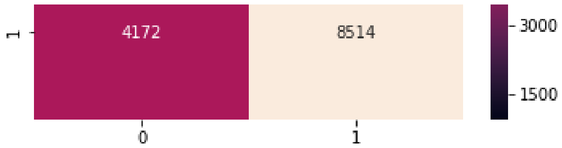
```
from sklearn.metrics import confusion_matrix
fig = plt.figure()
ax = fig.add_subplot(111)
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print('-'*75)
print("Train confusion matrix")
cm = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
sns.heatmap(cm, annot=True, fmt='d')

plt.show(ax)


fig = plt.figure()
ax1 = fig.add_subplot(111)
print('-'*75)
print("Test confusion matrix")
cm = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
sns.heatmap(cm, annot=True, fmt='d')

plt.show(ax1)
```

```
the maximum value of tpr*(1-fpr) 0.43898191941941944 for threshold 0.484
---------------------------------------------------------------------------
Train confusion matrix
```




```
---------------------------------------------------------------------------
Test confusion matrix
```





In [137]:

```
#https://towardsdatascience.com/demystifying-confusion-matrix-confusion-9e82201592fd
tn, fp, fn, tp = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)).ravel()
print("True Negatives: ",tn)
print("False Positives: ",fp)
print("False Negatives: ",fn)
print("True Positives: ",tp)
```

```
True Negatives:  1359
False Positives:  955
False Negatives:  4172
True Positives:  8514
```

Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)

In [138]:

```
from scipy.sparse import hstack
X_tr = hstack((tfidf_w2v_vectors_train_titles, tfidf_w2v_vectors_train,
X_train_teacher_number_of_previously_posted_projects_norm, X_train_price_norm,
X_train_project_categories_ohe, X_train_project_subcategories_ohe, X_train_grade_ohe,
X_train_teacher_ohe, X_train_state_ohe)).tocsr()
X_te = hstack((tfidf_w2v_vectors_test_titles, tfidf_w2v_vectors_test,
X_test_teacher_number_of_previously_posted_projects_norm, X_test_price_norm,
X_test_project_categories_ohe, X_test_project_subcategories_ohe, X_test_grade_ohe,
X_test_teacher_ohe, X_test_state_ohe)).tocsr()

print("Final Data Matrix:")
print(X_tr.shape, y_train.shape)
print(X_te.shape, y_test.shape)
```

```
Final Data Matrix:
(35000, 1077) (35000,)
(15000, 1077) (15000,)
```

Hyperparameter tuning using GridSearchCV

In [139]:

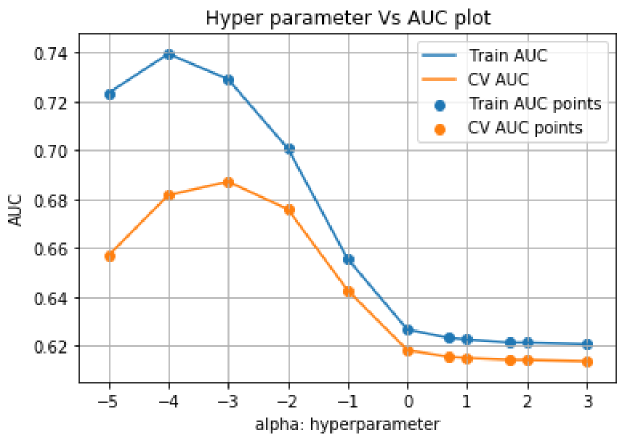
```
import math
from sklearn.model_selection import GridSearchCV
from sklearn import linear_model

lr = linear_model.SGDClassifier(loss='log', penalty='l2', class_weight = 'balanced')
parameters = {'alpha':[0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 5, 10, 50, 100, 1000]}
clf = GridSearchCV(lr, parameters, cv=10, scoring='roc_auc', return_train_score = True)
clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
alpha =  clf.cv_results_['param_alpha']
print(alpha)
```

```
[1e-05 0.0001 0.001 0.01 0.1 1 5 10 50 100 1000]
```

In [141]:

```
#use log10(alpha) on X axis while plotting ROC vs hyperparam plot, as it allows us to understand w
hat's happening in a better way
log_alpha = []
for a in tqdm(alpha):
    b=math.log10(a)
    log_alpha.append(b)

plt.plot(log_alpha, train_auc, label='Train AUC')
plt.plot(log_alpha, cv_auc, label='CV AUC')

plt.scatter(log_alpha, train_auc, label='Train AUC points')
plt.scatter(log_alpha, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
```

```
100%|████████████████████████████████████████████████████████████| 11/11
[00:00<00:00, 22331.73it/s]
```



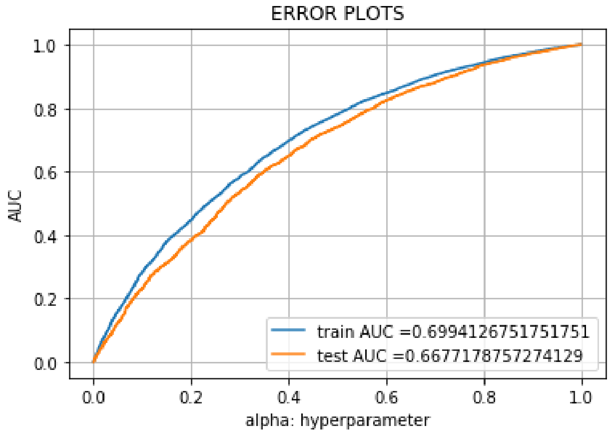
Summary:

Using gridsearchcv with cv=10 we got the best value of our hyperparameter as 0.01.

Testing the performance of the model on test data, plotting ROC Curves

In [142]:

```
from sklearn.metrics import roc_curve, auc

lr = linear_model.SGDClassifier(loss='log', penalty='l2', alpha=0.01, class_weight = 'balanced')
lr.fit(X_tr, y_train)

y_train_pred = pred_prob(lr, X_tr)
y_test_pred = pred_prob(lr, X_te)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



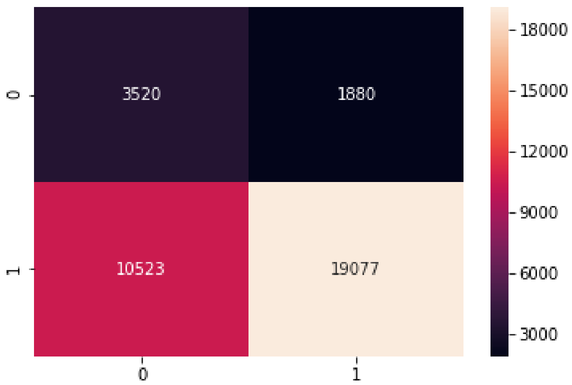
Summary:

From the above plot, we observe that at alpha=0.001 we get the train-AUC of 0.699 and test-AUC of 0.668.

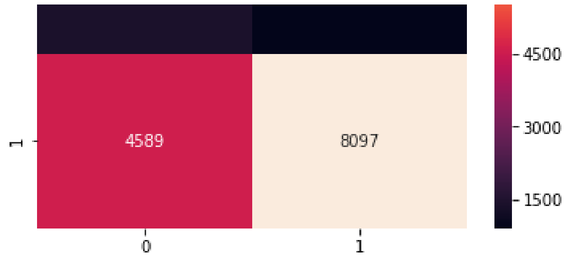
In [143]:

```
from sklearn.metrics import confusion_matrix
fig = plt.figure()
ax = fig.add_subplot(111)
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print('-'*75)
print("Train confusion matrix")
cm = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
sns.heatmap(cm, annot=True, fmt='d')

plt.show(ax)


fig = plt.figure()
ax1 = fig.add_subplot(111)
print('-'*75)
print("Test confusion matrix")
cm = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
sns.heatmap(cm, annot=True, fmt='d')

plt.show(ax1)
```

```
the maximum value of tpr*(1-fpr) 0.4201141141141141 for threshold 0.487
---------------------------------------------------------------------------
Train confusion matrix
```




```
---------------------------------------------------------------------------
Test confusion matrix
```



In [144]:

```
#https://towardsdatascience.com/demystifying-confusion-matrix-confusion-9e82201592fd
tn, fp, fn, tp = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)).ravel()
print("True Negatives: ",tn)
print("False Positives: ",fp)
print("False Negatives: ",fn)
print("True Positives: ",tp)
```

```
True Negatives:  1414
False Positives:  900
False Negatives:  4589
True Positives:  8097
```

SET 5

Number of words in the title

In [149]:

```
X_train_title_wordcount = []
for i in X_train['preprocessed_titles']:
    j = len(i.split())
    X_train_title_wordcount.append(j)
X_train['train_title_wordcount'] = X_train_title_wordcount
X_train.head(2)
```

Out[149]:

	Unnamed: 0	teacher_prefix	school_state	project_grade_category	project_subject_categories	project_subject_su
5164	5164	mrs	al	grades_3_5	literacy_language_specialneeds	literacy_specialneed
24801	24801	mrs	ca	grades_3_5	math_science	mathematics

In [147]:

```
X_test_title_wordcount = []
for i in X_test['preprocessed_titles']:
    j = len(i.split())
    X_test_title_wordcount.append(j)
X_test['title_wordcount'] = X_test_title_wordcount
X_test.head(2)
```

Out[147]:

	Unnamed: 0	teacher_prefix	school_state	project_grade_category	project_subject_categories	project_subject_subc
4623	4623	mrs	mo	grades_9_12	health_sports	teamsports

	Unnamed: 0	teacher_prefix	school_state	project_grade_category	project_subject_categories	project_subject_subc
11156	11156	mrs	tx	grades_prek_2	math_science	mathematics

In [169]:

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

X_train_wc_norm = normalizer.transform(X_train['train_title_wordcount'].values.reshape(1,-1))
X_test_wc_norm = normalizer.transform(X_test['title_wordcount'].values.reshape(1,-1))

print("After vectorizations")
print(X_train_wc_norm.shape, y_train.shape)
print(X_test_wc_norm.shape, y_test.shape)
```

```
After vectorizations
(1, 35000) (35000,)
(1, 15000) (15000,)
```

In [170]:

```
X_train_wc_norm= X_train_wc_norm.reshape(-1, 1)
X_test_wc_norm = X_test_wc_norm.reshape(-1, 1)
print(X_train_wc_norm.shape)
print(X_test_wc_norm.shape)
```

```
(35000, 1)
(15000, 1)
```

number of words in the combine essays

In [151]:

```
X_train_essay_wordcount = []
for i in X_train['preprocessed_essays']:
    j = len(i.split())
    X_train_essay_wordcount.append(j)
X_train['train_essay_wordcount'] = X_train_essay_wordcount
X_train.head(2)
```

Out[151]:

	Unnamed: 0	teacher_prefix	school_state	project_grade_category	project_subject_categories	project_subject_su
5164	5164	mrs	al	grades_3_5	literacy_language_specialneeds	literacy_specialneed
24801	24801	mrs	ca	grades_3_5	math_science	mathematics

In [153]:

```
X_test_essay_wordcount = []
for i in X_test['preprocessed_essays']:
    j = len(i.split())
    X_test_essay_wordcount.append(j)
X_test['test_essay_wordcount'] = X_test_essay_wordcount
X_test.head(2)
```

Out[153]:

	Unnamed: 0	teacher_prefix	school_state	project_grade_category	project_subject_categories	project_subject_subca
4623	4623	mrs	mo	grades_9_12	health_sports	teamsports
11156	11156	mrs	tx	grades_prek_2	math_science	mathematics

In [171]:

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

X_train_essay_wc_norm = normalizer.transform(X_train['train_essay_wordcount'].values.reshape(1,-1))
X_test_essay_wc_norm = normalizer.transform(X_test['test_essay_wordcount'].values.reshape(1,-1))

print("After vectorizations")
print(X_train_essay_wc_norm.shape, y_train.shape)
print(X_test_essay_wc_norm.shape, y_test.shape)
```

```
After vectorizations
(1, 35000) (35000,)
(1, 15000) (15000,)
```

In [172]:

```
X_train_essay_wc_norm= X_train_essay_wc_norm.reshape(-1, 1)
X_test_essay_wc_norm = X_test_essay_wc_norm.reshape(-1, 1)
print(X_train_essay_wc_norm.shape)
print(X_test_essay_wc_norm.shape)
```

```
(35000, 1)
(15000, 1)
```

sentiment score's of each of the essay

In [158]:

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
```

In [157]:

```
nltk.download('vader_lexicon')
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\syeda\AppData\Roaming\nltk_data...
```

Out[157]:

```
True
```

In [163]:

```
#let us call the SentimentIntensityAnalyser object
analyzer = SentimentIntensityAnalyzer()
```

In [164]:

```
#create four lists to store the values of neg, pos, neu, compound
neg = []
```

```
neg = []
pos = []
neu = []
compound = []

for i in tqdm(X_train['preprocessed_essays']):
    a = analyzer.polarity_scores(i)['neg']
    b = analyzer.polarity_scores(i)['pos']
    c = analyzer.polarity_scores(i)['neu']
    d = analyzer.polarity_scores(i)['compound']
    neg.append(a)
    pos.append(b)
    neu.append(c)
    compound.append(d)
```

```
100%|██████████████████████████████████████████████████████████████████████████| 35000/35000 [03:12<00:00, 181.70it/s]
```

In [165]:

```
X_train['neg'] = neg
X_train['pos'] = pos
X_train['neu'] = neu
X_train['compound'] = compound
X_train.head(2)
```

Out[165]:

	Unnamed: 0	teacher_prefix	school_state	project_grade_category	project_subject_categories	project_subject_su
5164	5164	mrs	al	grades_3_5	literacy_language_specialneeds	literacy_specialneed
24801	24801	mrs	ca	grades_3_5	math_science	mathematics

Encoding numerical features:

In [193]:

```
# sentimentintensityanalyzer was applied for X_test['preprocessed_essays'] before encoding, its in below cells.
normalizer = Normalizer()
normalizer.fit(X_train['pos'].values.reshape(1,-1))

X_train_pos_norm = normalizer.transform(X_train['pos'].values.reshape(1,-1))
X_test_pos_norm = normalizer.transform(X_test['pos1'].values.reshape(1,-1))

print("After vectorizations")
print(X_train_pos_norm.shape, y_train.shape)
print(X_test_pos_norm.shape, y_test.shape)


X_train_pos_norm= X_train_pos_norm.reshape(-1, 1)
X_test_pos_norm = X_test_pos_norm.reshape(-1, 1)
print(X_train_pos_norm.shape)
print(X_test_pos_norm.shape)
```

```
After vectorizations
(1, 35000) (35000,)
(1, 15000) (15000,)
(35000, 1)
(15000, 1)
```

In [191]:

```
normalizer = Normalizer()
```

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

X_train_neg_norm = normalizer.transform(X_train['neg'].values.reshape(1,-1))
X_test_neg_norm = normalizer.transform(X_test['neg1'].values.reshape(1,-1))

print("After vectorizations")
print(X_train_neg_norm.shape, y_train.shape)
print(X_test_neg_norm.shape, y_test.shape)
```

```
After vectorizations
(1, 35000) (35000,)
(1, 15000) (15000,)
```

In [192]:

```
X_train_neg_norm= X_train_neg_norm.reshape(-1, 1)
X_test_neg_norm = X_test_neg_norm.reshape(-1, 1)
print(X_train_neg_norm.shape)
print(X_test_neg_norm.shape)
```

```
(35000, 1)
(15000, 1)
```

In [183]:

```
normalizer = Normalizer()
normalizer.fit(X_train['neu'].values.reshape(1,-1))

X_train_neu_norm = normalizer.transform(X_train['neu'].values.reshape(1,-1))
X_test_neu_norm = normalizer.transform(X_test['neu1'].values.reshape(1,-1))

print("After vectorizations")
print(X_train_neu_norm.shape, y_train.shape)
print(X_test_neu_norm.shape, y_test.shape)


X_train_neu_norm= X_train_neu_norm.reshape(-1, 1)
X_test_neu_norm = X_test_neu_norm.reshape(-1, 1)
print(X_train_neu_norm.shape)
print(X_test_neu_norm.shape)
```

```
After vectorizations
(1, 35000) (35000,)
(1, 15000) (15000,)
(35000, 1)
(15000, 1)
```

In [184]:

```
normalizer = Normalizer()
normalizer.fit(X_train['compound'].values.reshape(1,-1))

X_train_compound_norm = normalizer.transform(X_train['compound'].values.reshape(1,-1))
X_test_compound_norm = normalizer.transform(X_test['compound1'].values.reshape(1,-1))

print("After vectorizations")
print(X_train_compound_norm.shape, y_train.shape)
print(X_test_compound_norm.shape, y_test.shape)


X_train_compound_norm= X_train_compound_norm.reshape(-1, 1)
X_test_compound_norm = X_test_compound_norm.reshape(-1, 1)
print(X_train_compound_norm.shape)
print(X_test_compound_norm.shape)
```

```
After vectorizations
(1, 35000) (35000,)
(1, 15000) (15000,)
(35000, 1)
(15000, 1)
```

In [166]:

```
neg = []
pos = []
neu = []
compound = []

for i in tqdm(X_test['preprocessed_essays']):
    a = analyzer.polarity_scores(i)['neg']
    b = analyzer.polarity_scores(i)['pos']
    c = analyzer.polarity_scores(i)['neu']
    d = analyzer.polarity_scores(i)['compound']
    neg.append(a)
    pos.append(b)
    neu.append(c)
    compound.append(d)
```

```
100%|██████████████████████████████████████████████████████████████████████████| 15000/15000 [01:
23<00:00, 180.67it/s]
```

In [167]:

```
X_test['neg1'] = neg
X_test['pos1'] = pos
X_test['neu1'] = neu
X_test['compound1'] = compound
X_test.head(2)
```

Out[167]:

	Unnamed: 0	teacher_prefix	school_state	project_grade_category	project_subject_categories	project_subject_subca
4623	4623	mrs	mo	grades_9_12	health_sports	teamsports
11156	11156	mrs	tx	grades_prek_2	math_science	mathematics

Encoding Quantity:

In [174]:

```
print(X_train['quantity'].isnull().values.any())
print("number of nan values",X_train['quantity'].isnull().values.sum())
print(X_test['quantity'].isnull().values.any())
print("number of nan values",X_test['quantity'].isnull().values.sum())
```

```
False
number of nan values 0
False
number of nan values 0
```

In [176]:

```
X_train['quantity'].value_counts()
```

Out[176]:

```
1       3345
2       2783
4       2419
3       2113
5       1979
        ...
373        1
377        1
```

```
277        1
213        1
232        1
511        1
Name: quantity, Length: 261, dtype: int64
```

In [177]:

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

X_train_quantity_norm = normalizer.transform(X_train['quantity'].values.reshape(1,-1))
X_test_quantity_norm = normalizer.transform(X_test['quantity'].values.reshape(1,-1))

print("After vectorizations")
print(X_train_quantity_norm.shape, y_train.shape)
print(X_test_quantity_norm.shape, y_test.shape)
```

```
After vectorizations
(1, 35000) (35000,)
(1, 15000) (15000,)
```

In [178]:

```
X_train_quantity_norm= X_train_quantity_norm.reshape(-1, 1)
X_test_quantity_norm = X_test_quantity_norm.reshape(-1, 1)
print(X_train_quantity_norm.shape)
print(X_test_quantity_norm.shape)
```

```
(35000, 1)
(15000, 1)
```

In [194]:

```
from scipy.sparse import hstack
X_tr = hstack((X_train_teacher_number_of_previously_posted_projects_norm, X_train_price_norm,
X_train_project_categories_ohe, X_train_project_subcategories_ohe, X_train_grade_ohe,
X_train_teacher_ohe, X_train_state_ohe,X_train_wc_norm, X_train_essay_wc_norm,
X_train_quantity_norm, X_train_pos_norm, X_train_neg_norm, X_train_neu_norm, X_train_compound_norm)
).tocsr()
X_te = hstack((X_test_teacher_number_of_previously_posted_projects_norm, X_test_price_norm,
X_test_project_categories_ohe, X_test_project_subcategories_ohe, X_test_grade_ohe,
X_test_teacher_ohe, X_test_state_ohe, X_test_wc_norm, X_test_essay_wc_norm, X_test_quantity_norm, X
_test_pos_norm, X_test_neg_norm, X_test_neu_norm, X_test_compound_norm)).tocsr()

print("Final Data Matrix:")
print(X_tr.shape, y_train.shape)
print(X_te.shape, y_test.shape)
```

```
Final Data Matrix:
(35000, 484) (35000,)
(15000, 484) (15000,)
```

Hyperparameter tuning using GridSearchCV

In [195]:

```
import math
from sklearn.model_selection import GridSearchCV
from sklearn import linear_model

lr = linear_model.SGDClassifier(loss='log', penalty='l2', class_weight = 'balanced')
parameters = {'alpha':[0.00001, 0.0001, 0.001, 0.01, 0.1, 1, 5, 10, 50, 100, 1000]}
clf = GridSearchCV(lr, parameters, cv=10, scoring='roc_auc', return_train_score = True)
clf.fit(X_tr, y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
```

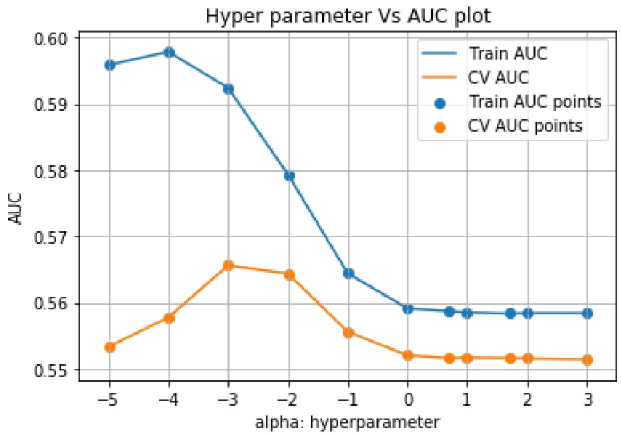
```
cv_auc_std= clf.cv_results_['std_test_score']
alpha =  clf.cv_results_['param_alpha']
print(alpha)
```

```
[1e-05 0.0001 0.001 0.01 0.1 1 5 10 50 100 1000]
```

In [196]:

```
#use log10(alpha) on X axis while plotting ROC vs hyperparam plot, as it allows us to understand w
hat's happening in a better way
log_alpha = []
for a in tqdm(alpha):
    b=math.log10(a)
    log_alpha.append(b)

plt.plot(log_alpha, train_auc, label='Train AUC')
plt.plot(log_alpha, cv_auc, label='CV AUC')

plt.scatter(log_alpha, train_auc, label='Train AUC points')
plt.scatter(log_alpha, cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
```

```
100%|████████████████████████████████████████████████████████████████████████████████| 11/11
[00:00<00:00, 4428.19it/s]
```



Summary:

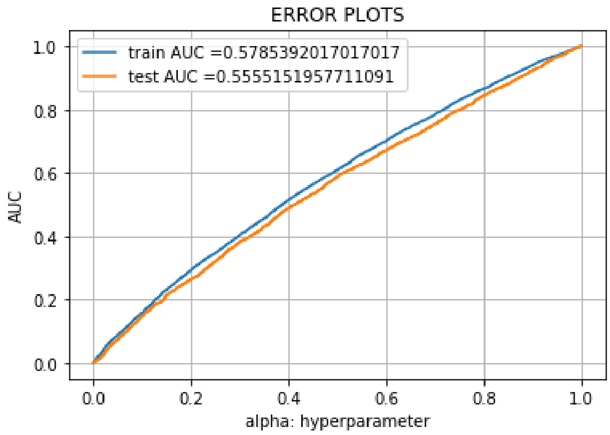
Using gridsearchcv with cv=10 we got the best value of our hyperparameter as 0.01.

Testing the performance of the model on test data, plotting ROC Curves

In [197]:

```
from sklearn.metrics import roc_curve, auc

lr = linear_model.SGDClassifier(loss='log', penalty='l2', alpha=0.01, class_weight = 'balanced')
lr.fit(X_tr, y_train)

y_train_pred = pred_prob(lr, X_tr)
y_test_pred = pred_prob(lr, X_te)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
```

```
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



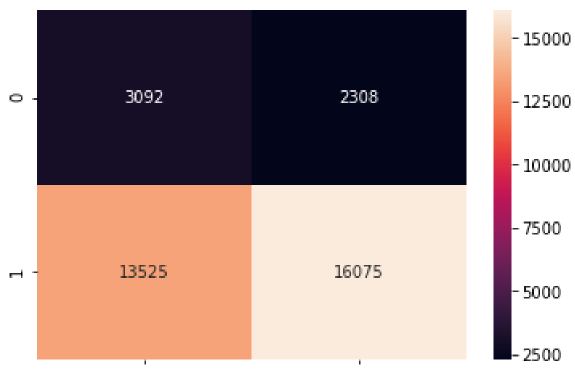
Summary:

From the above plot, we observe that at alpha=0.01 we get the train-AUC of 0.578 and test-AUC of 0.555. The AUC score goes down when text features are excluded. This shows that for this task text features are important in determining whether the project is accepted or not.

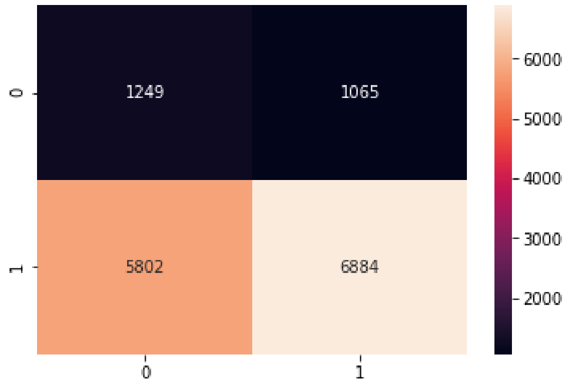
In [198]:

```
from sklearn.metrics import confusion_matrix
fig = plt.figure()
ax = fig.add_subplot(111)
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print('-'*75)
print("Train confusion matrix")
cm = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
sns.heatmap(cm, annot=True, fmt='d')

plt.show(ax)


fig = plt.figure()
ax1 = fig.add_subplot(111)
print('-'*75)
print("Test confusion matrix")
cm = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
sns.heatmap(cm, annot=True, fmt='d')

plt.show(ax1)
```

```
the maximum value of tpr*(1-fpr) 0.3109603353353353 for threshold 0.5
---------------------------------------------------------------------------
Train confusion matrix
```




```
---------------------------------------------------------------------------
Test confusion matrix
```



In [199]:

```python
#https://towardsdatascience.com/demystifying-confusion-matrix-confusion-9e82201592fd
tn, fp, fn, tp = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)).ravel()
print("True Negatives: ",tn)
print("False Positives: ",fp)
print("False Negatives: ",fn)
print("True Positives: ",tp)
```

```
True Negatives:  1249
False Positives:  1065
False Negatives:  5802
True Positives:  6884
```

3. Summary

In [200]:

```python
from prettytable import PrettyTable

x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Hyper Parameter", "Test-AUC"]

x.add_row(["BOW", "Logistic Regression", 0.1, 0.686])
x.add_row(["TFIDF", "Logistic Regression", 0.001, 0.674])
x.add_row(["AVG W2V", "Logistic Regression", 0.001, 0.674])
x.add_row(["TFIDF W2V", "Logistic Regression", 0.01, 0.668])

print(x)
```

```
+------------+---------------------+-----------------+----------+
| Vectorizer |        Model        | Hyper Parameter | Test-AUC |
+------------+---------------------+-----------------+----------+
|    BOW     | Logistic Regression |       0.1       |  0.686   |
|   TFIDF    | Logistic Regression |      0.001      |  0.674   |
|  AVG W2V   | Logistic Regression |      0.001      |  0.674   |
| TFIDF W2V  | Logistic Regression |       0.01      |  0.668   |
+------------+---------------------+-----------------+----------+
```

Using 50000 datapoints and GridSearchCV for Hyperparameter tuning, the better model is 'Logistic Regression with BOW vectorizer', however the difference between the test-auc scores is small.