Apply Support Vector Machine

In [1]:

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

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

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

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

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

from tqdm import tqdm
import os

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

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

In [2]:

```
project_data = pd.read_csv("train_data.csv", nrows = 30000)
resource_data = pd.read_csv("resources.csv", nrows = 30000)
```

In [3]:

```
print("Number of data points in train data", project_data.shape)
print('-'*50)
print("The attributes of data :", project_data.columns.values)
```

```
Number of data points in train data (30000, 17)
--------------------------------------------------
The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'school_state'
 'project_submitted_datetime' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'project_resource_summary'
 'teacher_number_of_previously_posted_projects' 'project_is_approved']
```

In [4]:

```
# Let's check for any "null" or "missing" values
project_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 17 columns):
Unnamed: 0                                      30000 non-null int64
id                                              30000 non-null object
teacher_id                                      30000 non-null object
teacher_prefix                                  29999 non-null object
school_state                                    30000 non-null object
project_submitted_datetime                      30000 non-null object
project_grade_category                          30000 non-null object
project_subject_categories                      30000 non-null object
project_subject_subcategories                   30000 non-null object
project_title                                   30000 non-null object
project_essay_1                                 30000 non-null object
project_essay_2                                 30000 non-null object
project_essay_3                                 1013 non-null object
project_essay_4                                 1013 non-null object
project_resource_summary                        30000 non-null object
teacher_number_of_previously_posted_projects    30000 non-null int64
project_is_approved                             30000 non-null int64
dtypes: int64(3), object(14)
memory usage: 3.9+ MB
```

In [5]:

```
project_data['teacher_prefix'].isna().sum()
```

Out[5]:

```
1
```

In [6]:

```
# "teacher_prefix" seems to contain 3 "missing" values, let't use mode replacement strategy to fil
l those missing values
project_data['teacher_prefix'].mode()
```

Out[6]:

```
0    Mrs.
dtype: object
```

In [7]:

```
# Let's replace the missing values with "Mrs." , as it is the mode of the "teacher_prefix"
project_data['teacher_prefix'] = project_data['teacher_prefix'].fillna('Mrs.')
```

In [8]:

```
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 [9]:

```
# Let's select only the selected features or columns, dropping "project_resource_summary" as it is
optional
#
project_data.drop(['id','teacher_id','project_submitted_datetime','project_resource_summary'],axis
=1, inplace=True)
project_data.columns
```

Out[9]:

```
Index(['Unnamed: 0', 'teacher_prefix', 'school_state',
       'project_grade_category', 'project_subject_categories',
```

```
       'project_subject_subcategories', 'project_title', 'project_essay_1',
       'project_essay_2', 'project_essay_3', 'project_essay_4',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'price', 'quantity'],
      dtype='object')
```

In [10]:

```
# Data seems to be highly imbalanced since the ratio of "class 1" to "class 0" is nearly 5.5
project_data['project_is_approved'].value_counts()
```

Out[10]:

```
1    25380
0     4620
Name: project_is_approved, dtype: int64
```

In [11]:

```
number_of_approved = project_data['project_is_approved'][project_data['project_is_approved'] == 1].
count()
number_of_not_approved = project_data['project_is_approved'][project_data['project_is_approved'] =
= 0].count()

print("Ratio of Project approved to Not approved is:", number_of_approved/number_of_not_approved)
```

```
Ratio of Project approved to Not approved is: 5.4935064935064934
```

Let's first merge all the project_essays into single columns

In [12]:

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

In [13]:

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

Out[13]:

	Unnamed: 0	teacher_prefix	school_state	project_grade_category	project_subject_categories	project_subject_subcatego
0	160221	Mrs.	IN	Grades PreK-2	Literacy & Language	ESL, Literacy
1	140945	Mr.	FL	Grades 6-8	History & Civics, Health & Sports	Civics & Government, Team Sports

In [14]:

```
# Let's drop the project essay columns from the dadaset now, as we have captured the essay text da
ta into single "essay" column
```

```
project_data.drop(['project_essay_1','project_essay_2','project_essay_3','project_essay_4'],axis=1
, inplace=True)
```

In [15]:

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

Out[15]:

	Unnamed: 0	teacher_prefix	school_state	project_grade_category	project_subject_categories	project_subject_subcatego
0	160221	Mrs.	IN	Grades PreK-2	Literacy & Language	ESL, Literacy

In [16]:

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

2) Make Data Model Ready: encoding numerical, categorical features

In [17]:

```
def cleaning_text_data(list_text_feature,df,old_col_name,new_col_name):

    # remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039
    # https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
    # https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
    # https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
    feature_list = []
    for i in list_text_feature:
        temp = ""
        # consider we have text like this "Math & Science, Warmth, Care & Hunger"
        for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care
& Hunger"]
            if 'The' in j.split(): # this will split each of the catogory based on space "Math & Sc
ience"=> "Math","&", "Science"
                j=j.replace('The','') # if we have the words "The" we are going to replace it with
''(i.e removing 'The')
            j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math & Sc
ience"=>"Math&Science"
            temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
            temp = temp.replace('&','_') # we are replacing the & value into
        feature_list.append(temp.strip())

    df[new_col_name] = feature_list
    df.drop([old_col_name], axis=1, inplace=True)

    from collections import Counter
    my_counter = Counter()
    for word in df[new_col_name].values:
        my_counter.update(word.split())

    feature_dict = dict(my_counter)
    sorted_feature_dict = dict(sorted(feature_dict.items(), key=lambda kv: kv[1]))
    return sorted_feature_dict
```

In [18]:

```
def clean_project_grade(list_text_feature,df,old_col_name,new_col_name):

    # remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039
    # https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
    # https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
    # https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
    feature_list = []
    for i in list_text_feature:
        temp = i.split(' ')
        last_dig = temp[-1].split('-')
        fin = [temp[0]]
        fin.extend(last_dig)
        feature = '_'.join(fin)
        feature_list.append(feature.strip())

    df[new_col_name] = feature_list
    df.drop([old_col_name], axis=1, inplace=True)

    from collections import Counter
    my_counter = Counter()
    for word in df[new_col_name].values:
        my_counter.update(word.split())

    feature_dict = dict(my_counter)
    sorted_feature_dict = dict(sorted(feature_dict.items(), key=lambda kv: kv[1]))
    return sorted_feature_dict
```

2.1) Text Preprocessing: project_subject_categories

In [19]:

```
x_train_sorted_category_dict = cleaning_text_data(X_train['project_subject_categories'],X_train,'project_subject_categories','clean_categories')
x_test_sorted_category_dict =
cleaning_text_data(X_test['project_subject_categories'],X_test,'project_subject_categories','clean_categories')
```

2.2) Text Preprocessing : project_subject_subcategories

In [20]:

```
x_train_sorted_subcategories = cleaning_text_data(X_train['project_subject_subcategories'],X_train,'project_subject_subcategories','clean_subcategories')
x_test_sorted_subcategories = cleaning_text_data(X_test['project_subject_subcategories'],X_test,'project_subject_subcategories','clean_subcategories')
```

2.3) Text Preprocessing: project_grade_category

In [21]:

```
x_train_sorted_grade =
clean_project_grade(X_train['project_grade_category'],X_train,'project_grade_category','clean_grade')
x_test_sorted_grade =
clean_project_grade(X_test['project_grade_category'],X_test,'project_grade_category','clean_grade')
```

2.4) Text Preprocessing (stowords): project_essay, project_title

In [22]:

```
# https://stackoverflow.com/a/47091490/4084039
import re
```

```python
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 [23]:

```python
# 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 [24]:

```python
# Combining all the above stundents
from tqdm import tqdm
def process_text(df,col_name):
    preprocessed_feature = []
    # tqdm is for printing the status bar
    for sentance in tqdm(df[col_name].values):
        sent = decontracted(sentance)
        sent = sent.replace('\\r', ' ')
        sent = sent.replace('\\"', ' ')
        sent = sent.replace('\\n', ' ')
        sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
        # https://gist.github.com/sebleier/554280
        sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
        preprocessed_feature.append(sent.lower().strip())
    return preprocessed_feature
```

In [25]:

```python
x_train_essay_preprocessed = process_text(X_train,'essay')
x_test_essay_preprocessed = process_text(X_test,'essay')
```

```
100%|██████████████████████████████████████████████████| 20100/20100
```

```
[00:13<00:00, 1520.46it/s]
100%|██████████████████████████████████████████████████████████████████████████| 9900/9900
[00:06<00:00, 1519.22it/s]
```

In [26]:

```
x_train_title_preprocessed = process_text(X_train,'project_title')
x_test_title_preprocessed = process_text(X_test,'project_title')
```

```
100%|███████████████████████████████████████████████████████████████████████| 20100/20100
[00:00<00:00, 29992.84it/s]
100%|█████████████████████████████████████████████████████████████████████████| 9900/9900
[00:00<00:00, 30696.04it/s]
```

2.5) Vectorizing Categorical Data

project_subject_categories (clean_categories)

In [27]:

```
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
def cat_vectorizer(X_train,df,col_name):
    vectorizer = CountVectorizer()
    vectorizer.fit(X_train[col_name].values)
    feature_one_hot = vectorizer.transform(df[col_name].values)
    print(vectorizer.get_feature_names())
    return feature_one_hot, vectorizer.get_feature_names()
```

In [28]:

```
x_train_cat_one_hot, x_train_cat_feat_list = cat_vectorizer(X_train,X_train,'clean_categories')
x_test_cat_one_hot, x_test_cat_feat_list = cat_vectorizer(X_train,X_test,'clean_categories')
```

```
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language',
'math_science', 'music_arts', 'specialneeds', 'warmth']
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language',
'math_science', 'music_arts', 'specialneeds', 'warmth']
```

In [29]:

```
# shape after categorical one hot encoding
print(x_train_cat_one_hot.shape)
print(x_test_cat_one_hot.shape)
```

```
(20100, 9)
(9900, 9)
```

project_subject_subcategory (clean_subcategory)

In [30]:

```
x_train_subcat_one_hot, x_train_subcat_feat_list =
cat_vectorizer(X_train,X_train,'clean_subcategories')
x_test_subcat_one_hot, x_test_subcat_feat_list =
cat_vectorizer(X_train,X_test,'clean_subcategories')
```

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

'esl', 'extracurricular', 'financialliteracy', 'foreignlanguages', 'gym_fitness',
'health_lifescience', 'health_wellness', 'history_geography', 'literacy', 'literature_writing', 'm
athematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'socia
lsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']

In [31]:

```
# shape after categorical one hot encoding
print(x_train_subcat_one_hot.shape)
print(x_test_subcat_one_hot.shape)
```

```
(20100, 30)
(9900, 30)
```

school_state

In [32]:

```
# we use count vectorizer to convert the values into one hot encoding
# CountVectorizer for "school_state"
x_train_state_one_hot, x_train_state_feat_list = cat_vectorizer(X_train,X_train,'school_state')
x_test_state_one_hot, x_test_state_feat_list = cat_vectorizer(X_train,X_test,'school_state')
```

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

In [33]:

```
# shape after categorical one hot encoding
print(x_train_state_one_hot.shape)
print(x_test_state_one_hot.shape)
```

```
(20100, 51)
(9900, 51)
```

teacher_prefix

In [34]:

```
# we use count vectorizer to convert the values into one hot encoding
# CountVectorizer for teacher_prefix
x_train_teacher_prefix_one_hot,x_train_teacher_prefix_feat_list = cat_vectorizer(X_train,X_train,'
teacher_prefix')
x_test_teacher_prefix_one_hot,x_test_teacher_prefix_feat_list =
cat_vectorizer(X_train,X_test,'teacher_prefix')
```

```
['mr', 'mrs', 'ms', 'teacher']
['mr', 'mrs', 'ms', 'teacher']
```

In [35]:

```
# shape after categorical one hot encoding
print(x_train_teacher_prefix_one_hot.shape)
print(x_test_teacher_prefix_one_hot.shape)
```

```
(20100, 4)
(9900, 4)
```

project_grade_category

In [36]:

```
# using count vectorizer for one-hot encoding of project_grade_category
x_train_grade_one_hot, x_train_grade_feat_list = cat_vectorizer(X_train,X_train,'clean_grade')
x_test_grade_one_hot, x_test_grade_feat_list = cat_vectorizer(X_train,X_test,'clean_grade')
```

```
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
```

In [37]:

```
# shape after categorical one hot encoding
print(x_train_grade_one_hot.shape)
print(x_test_grade_one_hot.shape)
```

```
(20100, 4)
(9900, 4)
```

2.6) Vectorizing Text Data

2.6.1) Bag of Words (essay)

In [38]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
def bow_vectorizer(X_train,col_name,df):
    vectorizer = CountVectorizer(min_df=10)
    vectorizer.fit(X_train[col_name].values)
    df_bow = vectorizer.transform(df[col_name].values)
    return df_bow, vectorizer.get_feature_names()
```

In [39]:

```
x_train_essay_bow, x_train_essay_feat = bow_vectorizer(X_train,'essay',X_train)
x_test_essay_bow,  x_test_essay_feat  = bow_vectorizer(X_train,'essay',X_test)
```

In [40]:

```
print(x_train_essay_bow.shape)
print(x_test_essay_bow.shape)
```

```
(20100, 8760)
(9900, 8760)
```

2.6.2) Bag of Words (title)

In [41]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
def bow_vectorizer_title(X_train,col_name,df):
    vectorizer = CountVectorizer(min_df=10)
    vectorizer.fit(X_train[col_name].values)
    df_bow = vectorizer.transform(df[col_name].values)
    return df_bow, vectorizer.get_feature_names()
```

In [42]:

```
x_train_title_bow, x_train_title_feat = bow_vectorizer_title(X_train,'project_title',X_train)
x_test_title_bow,  x_test_title_feat  = bow_vectorizer_title(X_train,'project_title',X_test)
```

In [43]:

```
print(x_train_title_bow.shape)
print(x_test_title_bow.shape)
```

```
(20100, 1145)
(9900, 1145)
```

2.6.3) TFIDF (essay)

In [44]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
# We are considering only the words which appeared in at least 10 documents(rows or projects).
def tfidf_vectorizer(X_train,col_name,df):
    vectorizer = TfidfVectorizer(min_df=10)
    vectorizer.fit(X_train[col_name].values)
    df_tfidf = vectorizer.transform(df[col_name].values)
    return df_tfidf, vectorizer.get_feature_names()
```

In [45]:

```
# Lets vectorize essay
x_train_essay_tfidf, x_train_essay_tfidf_feat = tfidf_vectorizer(X_train,'essay',X_train)
x_test_essay_tfidf, x_test_essay_tfidf_feat = tfidf_vectorizer(X_train,'essay',X_test)
```

In [46]:

```
print(x_train_essay_tfidf.shape)
print(x_test_essay_tfidf.shape)
```

```
(20100, 8760)
(9900, 8760)
```

2.6.4) TFIDF (title)

In [47]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
# We are considering only the words which appeared in at least 10 documents(rows or projects).
def tfidf_vectorizer_title(X_train,col_name,df):
    vectorizer = TfidfVectorizer(min_df=10)
    vectorizer.fit(X_train[col_name].values)
    df_tfidf = vectorizer.transform(df[col_name].values)
    return df_tfidf, vectorizer.get_feature_names()
```

In [48]:

```
# Lets vectorize essay
x_train_title_tfidf, x_train_title_tfidf_feat =
tfidf_vectorizer_title(X_train,'project_title',X_train)
x_test_title_tfidf, x_test_title_tfidf_feat =
tfidf_vectorizer_title(X_train,'project_title',X_test)
```

In [49]:

```
print(x_train_title_tfidf.shape)
print(x_test_title_tfidf.shape)
```

```
(20100, 1145)
(9900, 1145)
```

2.6.5) Using Pretrained Models: Avg W2V

In [50]:

```
'''
```

```
'''
# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039
def loadGloveModel(gloveFile):
    print ("Loading Glove Model")
    f = open(gloveFile,'r', encoding="utf8")
    model = {}
    for line in tqdm(f):
        splitLine = line.split()
        word = splitLine[0]
        embedding = np.array([float(val) for val in splitLine[1:]])
        model[word] = embedding
    print ("Done.",len(model)," words loaded!")
    return model
model = loadGloveModel('glove.42B.300d.txt')

# ============================
Output:

Loading Glove Model
1917495it [06:32, 4879.69it/s]
Done. 1917495  words loaded!

# ============================

words = []
for i in preproced_texts:
    words.extend(i.split(' '))

for i in preproced_titles:
    words.extend(i.split(' '))
print("all the words in the coupus", len(words))
words = set(words)
print("the unique words in the coupus", len(words))

inter_words = set(model.keys()).intersection(words)
print("The number of words that are present in both glove vectors and our coupus", \
      len(inter_words),"(",np.round(len(inter_words)/len(words)*100,3),"%)")

words_courpus = {}
words_glove = set(model.keys())
for i in words:
    if i in words_glove:
        words_courpus[i] = model[i]
print("word 2 vec length", len(words_courpus))


# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-sa
ve-and-load-variables-in-python/

import pickle
with open('glove_vectors', 'wb') as f:
    pickle.dump(words_courpus, f)


'''
```

Out[50]:

```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039\ndef
loadGloveModel(gloveFile):\n    print ("Loading Glove Model")\n    f = open(gloveFile,\'r\',
encoding="utf8")\n    model = {}\n    for line in tqdm(f):\n        splitLine = line.split()\n
word = splitLine[0]\n        embedding = np.array([float(val) for val in splitLine[1:]])\n        m
odel[word] = embedding\n    print ("Done.",len(model)," words loaded!")\n    return model\nmodel =
loadGloveModel(\'glove.42B.300d.txt\')\n\n# ============================\nOutput:\n    \nLoading G
love Model\n1917495it [06:32, 4879.69it/s]\nDone. 1917495  words loaded!\n\n#
============================\n\nwords = []\nfor i in preproced_texts:\n    words.extend(i.split(\'
\'))\n\nfor i in preproced_titles:\n    words.extend(i.split(\' \'))\nprint("all the words in the
coupus", len(words))\nwords = set(words)\nprint("the unique words in the coupus",
len(words))\n\ninter_words = set(model.keys()).intersection(words)\nprint("The number of words tha
t are present in both glove vectors and our coupus",      len(inter_words),"
(",np.round(len(inter_words)/len(words)*100,3),"%)")\n\nwords_courpus = {}\nwords_glove =
set(model.keys())\nfor i in words:\n    if i in words_glove:\n        words_courpus[i] = model[i]\n
print("word 2 vec length", len(words_courpus))\n\n\n# stronging variables into pickle files python
: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-python/\n\nimport pic
kle\nwith open(\'glove_vectors\', \'wb\') as f:\n    pickle.dump(words_courpus, f)\n\n\n'
```

In [51]:

```
# 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 [52]:

```
# Combining all the above stundents
from tqdm import tqdm
def preprocess_essay(df,col_name):
    preprocessed_essays = []
    # tqdm is for printing the status bar
    for sentance in tqdm(df[col_name].values):
        sent = decontracted(sentance)
        sent = sent.replace('\\r', ' ')
        sent = sent.replace('\\"', ' ')
        sent = sent.replace('\\n', ' ')
        sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
        # https://gist.github.com/sebleier/554280
        sent = ' '.join(e for e in sent.split() if e not in stopwords)
        preprocessed_essays.append(sent.lower().strip())
    return preprocessed_essays
```

In [53]:

```
# average Word2Vec
# compute average word2vec for each review.
def compute_avg_W2V(preprocessed_feature):
    avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
    for sentence in tqdm(preprocessed_feature): # 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.append(vector)
    return avg_w2v_vectors
```

In [54]:

```
x_train_preprocessed_essay = preprocess_essay(X_train,'essay')
x_test_preprocessed_essay = preprocess_essay(X_test,'essay')
```

```
100%|███████████████████████████████████████████████████████████████████████████| 20100/20100
[00:15<00:00, 1281.10it/s]
100%|█████████████████████████████████████████████████████████████████████████████| 9900/9900
[00:07<00:00, 1237.78it/s]
```

In [55]:

```
x_train_preprocessed_title = preprocess_essay(X_train,'project_title')
x_test_preprocessed_title = preprocess_essay(X_test,'project_title')
```

```
100%|██████████████████████████████████████████████████████████████████████████| 20100/20100
[00:00<00:00, 24048.91it/s]
100%|████████████████████████████████████████████████████████████████████████████| 9900/9900
[00:00<00:00, 21985.55it/s]
```

In [56]:

```
x_train_avg_w2v_essay = compute_avg_W2V(x_train_preprocessed_essay)
x_test_avg_w2v_essay = compute_avg_W2V(x_test_preprocessed_essay)
```

```
100%|███████████████████████████████████████████████████████████████████████████| 20100/20100
```

```
100%|██████████████████████████████████████████████████| 20100/20100
[00:11<00:00, 1748.52it/s]
100%|██████████████████████████████████████████████████| 9900/9900
[00:05<00:00, 1786.46it/s]
```

In [57]:

```
x_train_avg_w2v_title = compute_avg_W2V(x_train_preprocessed_title)
x_test_avg_w2v_title = compute_avg_W2V(x_test_preprocessed_title)
```

```
100%|██████████████████████████████████████████████████| 20100/20100
[00:00<00:00, 41830.47it/s]
100%|██████████████████████████████████████████████████| 9900/9900
[00:00<00:00, 41147.36it/s]
```

2.6.6) Using Pretrained Models: TFIDF Weighted W2V

In [58]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
def get_tfidf_dict(preprocessed_feature_train, preprocessed_feature_test,data_type):
    tfidf_model = TfidfVectorizer()
    if data_type == 'train':
        tfidf_model.fit(preprocessed_feature_train)
    elif data_type == 'test':
        tfidf_model.fit(preprocessed_feature_train)
        tfidf_model.transform(preprocessed_feature_test)
    # we are converting a dictionary with word as a key, and the idf as a value
    dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
    tfidf_words = set(tfidf_model.get_feature_names())
    return dictionary, tfidf_words
```

In [59]:

```
# average Word2Vec
# compute average word2vec for each review.
def compute_tfidf_w2v_vectors(preprocessed_feature_train, preprocessed_feature_test, data_type):
    tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
    dictionary, tfidf_words = get_tfidf_dict(preprocessed_feature_train, preprocessed_feature_test,
data_type)
    if data_type == 'train':
        preprocessed_feature = preprocessed_feature_train
    else:
        preprocessed_feature = preprocessed_feature_test
    for sentence in tqdm(preprocessed_feature): # 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 tfidf 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.append(vector)
    return tfidf_w2v_vectors
```

In [60]:

```
dictionary_train, tfidf_words_train = get_tfidf_dict(x_train_essay_preprocessed,
x_test_essay_preprocessed, 'train')
dictionary_test, tfidf_words_test = get_tfidf_dict(x_train_essay_preprocessed,
x_test_essay_preprocessed, 'test')
```

In [61]:

```
x_train_weighted_w2v_essay = compute_tfidf_w2v_vectors(x_train_essay_preprocessed,
```

```
x_train_weighted_w2v_essay = compute_tfidf_w2v_vectors(x_train_essay_preprocessed,
x_test_essay_preprocessed, 'train')
x_test_weighted_w2v_essay= compute_tfidf_w2v_vectors(x_train_essay_preprocessed,
x_test_essay_preprocessed, 'test')
```

```
100%|████████████████████████████████████████████████████████████████████████████| 20100/20100 [01:
02<00:00, 321.09it/s]
100%|██████████████████████████████████████████████████████████████████████████████| 9900/9900
[00:31<00:00, 317.91it/s]
```

In [62]:

```
x_train_weighted_w2v_title = compute_tfidf_w2v_vectors(x_train_preprocessed_title,
x_test_preprocessed_title, 'train')
x_test_weighted_w2v_title= compute_tfidf_w2v_vectors(x_train_preprocessed_title,
x_test_preprocessed_title, 'test')
```

```
100%|██████████████████████████████████████████████████████████████████████████| 20100/20100
[00:01<00:00, 14805.24it/s]
100%|████████████████████████████████████████████████████████████████████████████| 9900/9900
[00:00<00:00, 17312.03it/s]
```

2.6.7) Vectorizing Numerical Features

We have 2 numerical features left, "price" and "teacher_number_of_previously_posted_projects". Let's check for the "missing" or "NaN" values present in those numerical features and use "Mean Replacement" for "price" and "Mode Replacement" for "teacher_number_of_previously_posted_projects".

In [63]:

```
print("Total number of \"Missing\" Values present in X_train price:",X_train['price'].isna().sum()
)
print("Total number of \"Missing\" Values present in X_test price:",X_test['price'].isna().sum())
```

```
Total number of "Missing" Values present in X_train price: 19705
Total number of "Missing" Values present in X_test price: 9728
```

In [64]:

```
print("Total number of \"Missing\" Values present in X_train previous teacher number:",X_train['te
acher_number_of_previously_posted_projects'].isna().sum())
print("Total number of \"Missing\" Values present in X_test previous teacher number:",X_test['teac
her_number_of_previously_posted_projects'].isna().sum())
```

```
Total number of "Missing" Values present in X_train previous teacher number: 0
Total number of "Missing" Values present in X_test previous teacher number: 0
```

In [65]:

```
print("Total number of \"Missing\" Values present in X_train quantity:",X_train['quantity'].isna()
.sum())
print("Total number of \"Missing\" Values present in X_test quantity:",X_test['quantity'].isna().s
um())
```

```
Total number of "Missing" Values present in X_train quantity: 19705
Total number of "Missing" Values present in X_test quantity: 9728
```

"teacher_number_of_previously_posted_projects" does not have any "missing" values.

In [66]:

```
X_train['price'].mean()
```

Out[66]:

```
289.77784810126604
```

In [67]:

```
X_train['price'] = X_train['price'].fillna(289.7778)
```

In [68]:

```
X_test['price'].mean()
```

Out[68]:

```
253.14540697674417
```

In [69]:

```
X_test['price'] = X_test['price'].fillna(253.1454)
```

In [70]:

```
print(X_train['quantity'].mean())
print(X_test['quantity'].mean())
```

```
19.245569620253164
17.302325581395348
```

In [71]:

```
X_train['quantity'] = X_train['quantity'].fillna(19.2455)
X_test['quantity'] = X_test['quantity'].fillna(17.3023)
```

In [72]:

```
# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s
# https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
from sklearn.preprocessing import StandardScaler
def scaler_function(df,col_name):

    scaler = StandardScaler()
    scaler.fit(df[col_name].values.reshape(-1,1)) # finding the mean and standard deviation of this
data

    # Now standardize the data with above maen and variance.
    print(f"Mean : {scaler.mean_[0]}, Standard deviation : {np.sqrt(scaler.var_[0])}")
    scaled = scaler.transform(df[col_name].values.reshape(-1, 1))
    return scaled
```

teacher_number_of_previously_posted_projects

In [73]:

```
x_train_teacher_number = scaler_function(X_train,'teacher_number_of_previously_posted_projects')
x_test_teacher_number = scaler_function(X_test,'teacher_number_of_previously_posted_projects')
```

```
Mean : 11.264776119402985, Standard deviation : 28.24534202913663
Mean : 11.103737373737374, Standard deviation : 27.277127099453022
```

price

In [74]:

```
x_train_price = scaler_function(X_train,'price')
x_test_price = scaler_function(X_test,'price')
```

```
Mean : 289.77780094527355, Standard deviation : 63.027727401964185
Mean : 253.14540012121205, Standard deviation : 30.465281737091416
```

In [75]:

```
x_train_quantity = scaler_function(X_train,'quantity')
x_test_quantity = scaler_function(X_test,'quantity')
```

```
Mean : 19.2455013681592, Standard deviation : 3.9008174914914933
Mean : 17.302300444444445, Standard deviation : 3.157231223233005
```

2.7) Merging all the features and building the sets

In [76]:

```
# train dataset
print("After Vectorization and One hot encoding train dataset shape becomes:")
print(x_train_cat_one_hot.shape)
print(x_train_subcat_one_hot.shape)
print(x_train_state_one_hot.shape)
print(x_train_teacher_prefix_one_hot.shape)
print(x_train_grade_one_hot.shape)
print(x_train_essay_bow.shape)
print(x_train_title_bow.shape)
print(x_train_essay_tfidf.shape)
print(x_train_title_tfidf.shape)
print(np.asarray(x_train_avg_w2v_essay).shape)
print(np.asarray(x_train_avg_w2v_title).shape)
print(np.asarray(x_train_weighted_w2v_essay).shape)
print(np.asarray(x_train_weighted_w2v_title).shape)
print(x_train_teacher_number.shape)
print(x_train_price.shape)
print(x_train_quantity.shape)
print("="*50)

# test dataset
print("After Vectorization and One hot encoding test dataset shape becomes:")
print(x_test_cat_one_hot.shape)
print(x_test_subcat_one_hot.shape)
print(x_test_state_one_hot.shape)
print(x_test_teacher_prefix_one_hot.shape)
print(x_test_grade_one_hot.shape)
print(x_test_essay_bow.shape)
print(x_test_title_bow.shape)
print(x_test_essay_tfidf.shape)
print(x_test_title_tfidf.shape)
print(np.asarray(x_test_avg_w2v_essay).shape)
print(np.asarray(x_test_avg_w2v_title).shape)
print(np.asarray(x_test_weighted_w2v_essay).shape)
print(np.asarray(x_test_weighted_w2v_title).shape)
print(x_test_teacher_number.shape)
print(x_test_price.shape)
print(x_test_quantity.shape)
print("="*50)
```

```
After Vectorization and One hot encoding train dataset shape becomes:
(20100, 9)
(20100, 30)
(20100, 51)
(20100, 4)
(20100, 4)
(20100, 8760)
(20100, 1145)
(20100, 8760)
(20100, 1145)
(20100, 300)
(20100, 300)
(20100, 300)
(20100, 300)
(20100, 1)
(20100, 1)
(20100, 1)
==================================================
After Vectorization and One hot encoding test dataset shape becomes:
(9900, 9)
```

```
(9900, 30)
(9900, 51)
(9900, 4)
(9900, 4)
(9900, 8760)
(9900, 1145)
(9900, 8760)
(9900, 1145)
(9900, 300)
(9900, 300)
(9900, 300)
(9900, 300)
(9900, 1)
(9900, 1)
(9900, 1)
==================================================
```

2.7.1) Set 1: categorical, numerical features + project_title(BOW) + preprocessed_eassay (BOW)

In [77]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_train_set_1 = 
hstack((x_train_cat_one_hot,x_train_subcat_one_hot,x_train_state_one_hot,x_train_teacher_prefix_one
_hot,\

x_train_grade_one_hot,x_train_teacher_number,x_train_price,x_train_quantity,x_train_title_bow,x_tr
ain_essay_bow)).tocsr()
X_test_set_1 = 
hstack((x_test_cat_one_hot,x_test_subcat_one_hot,x_test_state_one_hot,x_test_teacher_prefix_one_hot
,\
                         x_test_grade_one_hot,x_test_teacher_number,x_test_price,x_test_quantity,x_
test_title_bow,x_test_essay_bow)).tocsr()
```

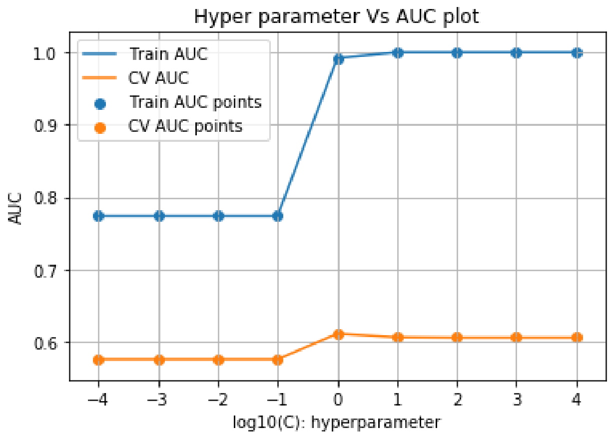
In [78]:

```
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
from sklearn.svm import SVC
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import roc_curve, auc
import math
```

In [81]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
# for class prior i referred https://scikit-
learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html,\
# https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

svm_ = SVC(class_weight = "balanced")
parameters = {'C':[10**x for x in range(-4,5)]}
clf = RandomizedSearchCV(svm_, parameters,n_iter = 9, scoring='roc_auc', return_train_score = True)
clf.fit(X_train_set_1, y_train)

results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_C'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
C_ =  results['param_C'].apply(lambda x: math.log10(x))

plt.plot(C_, train_auc, label='Train AUC')
```

```
plt.plot(C_, cv_auc, label='CV AUC')


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


plt.legend()
plt.xlabel("log10(C): hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()

results
```



Out[81]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params	split0_test_score	split1_test_score	s
0	0.845405	0.034177	0.204710	0.003416	0.0001	{'C': 0.0001}	0.587135	0.637908	0
1	0.832014	0.015563	0.205299	0.007661	0.001	{'C': 0.001}	0.587135	0.637908	0
2	0.833384	0.014308	0.202407	0.003843	0.01	{'C': 0.01}	0.587135	0.637908	0
3	0.858559	0.037666	0.221014	0.024869	0.1	{'C': 0.1}	0.587135	0.637908	0
4	0.766323	0.018562	0.188920	0.011463	1	{'C': 1}	0.576637	0.687536	0
5	0.808437	0.024412	0.172512	0.011876	10	{'C': 10}	0.577400	0.690399	0
6	0.808952	0.022179	0.155909	0.002025	100	{'C': 100}	0.576637	0.690399	0
7	0.812931	0.024074	0.157807	0.005505	1000	{'C': 1000}	0.576637	0.690399	0
8	0.792830	0.012511	0.156429	0.004660	10000	{'C': 10000}	0.576637	0.690399	0

9 rows × 21 columns

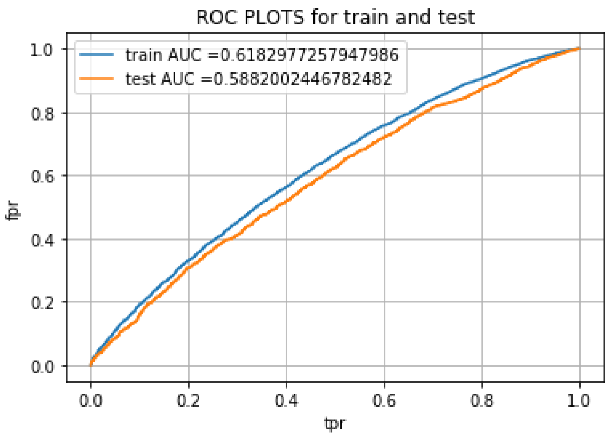
In [82]:

```
# From the AUC plot, we find that the best value for "C" - "Inverse of Regularization Strength" fo
r the LogisticRegression is 0.01
best_C = 0.1
```

In [83]:

```
# https://scikit-
```

```
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


svm_= SVC(C= best_C, class_weight = "balanced", probability = True)
svm_.fit(X_train_set_1, y_train)


y_train_pred = svm_.predict_proba(X_train_set_1)
y_test_pred =  svm_.predict_proba(X_test_set_1)

y_train_pred_prob = []
y_test_pred_prob = []

for index in range(len(y_train_pred)):
    y_train_pred_prob.append(y_train_pred[index][1])

for index in range(len(y_test_pred)):
    y_test_pred_prob.append(y_test_pred[index][1])


train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred_prob)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred_prob)

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("tpr")
plt.ylabel("fpr")
plt.title("ROC PLOTS for train and test")
plt.grid()
plt.show()
```



In [84]:

```
# 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 [85]:

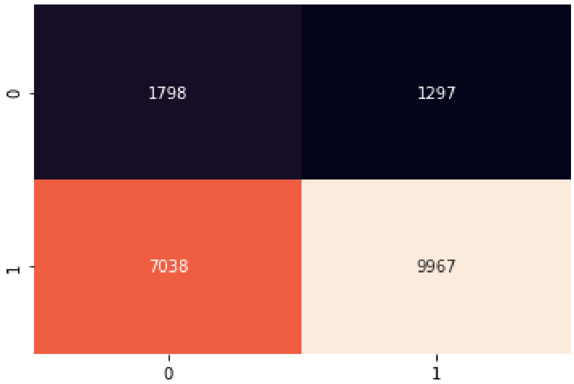
```
print("="*100)
from sklearn.metrics import confusion_matrix
```

```
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
sns.heatmap(confusion_matrix(y_train, predict_with_best_t(y_train_pred_prob, best_t)),annot = True,
fmt = "d", cbar=False)
```

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

the maximum value of tpr*(1-fpr) 0.3404997959832207 for threshold 0.834
Train confusion matrix
```

Out[85]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x19f152d9080>
```



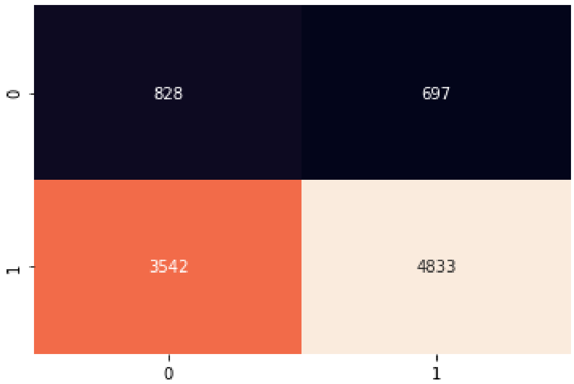
In [86]:

```
print("Test confusion matrix")
sns.heatmap(confusion_matrix(y_test, predict_with_best_t(y_test_pred_prob, best_t)), annot = True,
fmt = "d", cbar=False)
```

```
Test confusion matrix
```

Out[86]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x19f120982b0>
```



2.7.2) Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)

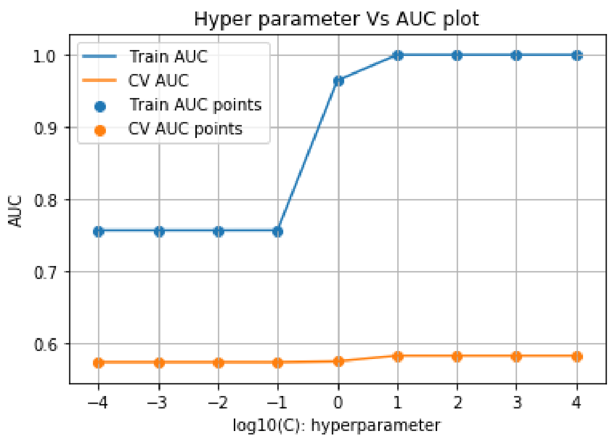
In [87]:

```
X_train_set_2 =
hstack((x_train_cat_one_hot,x_train_subcat_one_hot,x_train_state_one_hot,x_train_teacher_prefix_one
_hot,\

x_train_grade_one_hot,x_train_teacher_number,x_train_price,x_train_quantity,x_train_title_tfidf,x_
train_essay_tfidf)).tocsr()
X_test_set_2 =
hstack((x_test_cat_one_hot,x_test_subcat_one_hot,x_test_state_one_hot,x_test_teacher_prefix_one_hot
,\
```

```
                          x_test_grade_one_hot,x_test_teacher_number,x_test_price,x_test_quantity,x_
test_title_tfidf,x_test_essay_tfidf)).tocsr()
# X_cv_set_2 =
hstack((x_cv_cat_one_hot,x_cv_subcat_one_hot,x_cv_state_one_hot,x_cv_teacher_prefix_one_hot,\
#
x_cv_grade_one_hot,x_cv_teacher_number,x_cv_price,x_cv_quantity,x_cv_title_tfidf,x_cv_essay_tfidf),
sr()
```

In [88]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
# for class prior i referred https://scikit-
learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html,\
# https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

svm_ = SVC(class_weight = "balanced", probability = True)
parameters = {'C':[10**x for x in range(-4,5)]}
clf = RandomizedSearchCV(svm_, parameters,n_iter = 9, scoring='roc_auc', return_train_score=True)
clf.fit(X_train_set_2, y_train)

results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_C'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
C_ =  results['param_C'].apply(lambda x: math.log10(x))

plt.plot(C_, train_auc, label='Train AUC')


plt.plot(C_, cv_auc, label='CV AUC')


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


plt.legend()
plt.xlabel("log10(C): hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()

results
```



Out[88]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params	split0_test_score	split1_test_score	s
0	4.335892	0.221849	0.210016	0.012581	0.0001	{'C': 0.0001}	0.577782	0.543043	0
1	4.188767	0.130759	0.206819	0.012110	0.001	{'C': 0.001}	0.577782	0.543043	0

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params	split0_test_score	split1_test_score	s
2	4.148769	0.113357	0.202932	0.004408	0.01	{'C': 0.01}	0.577782	0.543043	0
3	4.474311	0.154637	0.220212	0.011951	0.1	{'C': 0.1}	0.577782	0.542661	0
4	3.770089	0.049563	0.186412	0.005369	1	{'C': 1}	0.536362	0.568620	0
5	4.198255	0.305808	0.173230	0.016630	10	{'C': 10}	0.538080	0.583317	0
6	4.337163	0.215913	0.196990	0.039839	100	{'C': 100}	0.538271	0.585608	0
7	4.120537	0.176513	0.168860	0.012009	1000	{'C': 1000}	0.538271	0.585608	0
8	4.151167	0.129624	0.190840	0.022912	10000	{'C': 10000}	0.538271	0.585608	0

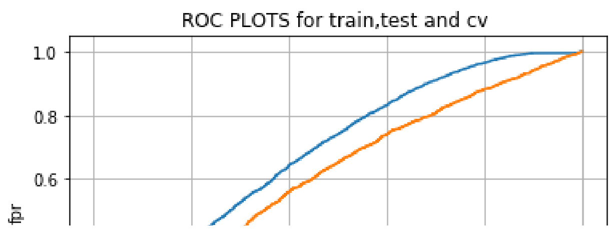
9 rows × 21 columns

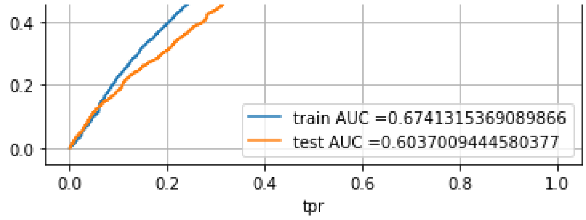
In [93]:

```
# From the AUC plot, we find that the best value for "C" - "Inverse of Regularization Strength" fo
r the LogisticRegression is 0.01
best_C = 1
```

In [94]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


svm = SVC(C=best_C, class_weight = "balanced", probability = True)
svm.fit(X_train_set_2, y_train)


y_train_pred = svm.predict_proba(X_train_set_2)
y_test_pred =  svm.predict_proba(X_test_set_2)

y_train_pred_prob = []
y_test_pred_prob = []


for index in range(len(y_train_pred)):
    y_train_pred_prob.append(y_train_pred[index][1])

for index in range(len(y_test_pred)):
    y_test_pred_prob.append(y_test_pred[index][1])


train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred_prob)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred_prob)


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("tpr")
plt.ylabel("fpr")
plt.title("ROC PLOTS for train,test and cv ")
plt.grid()
plt.show()
```





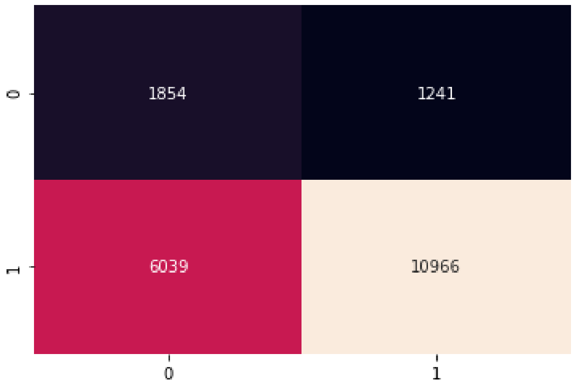
In [95]:

```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
sns.heatmap(confusion_matrix(y_train, predict_with_best_t(y_train_pred_prob, best_t)),annot = True,
fmt = "d", cbar=False)
```

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

the maximum value of tpr*(1-fpr) 0.38629641856737945 for threshold 0.831
Train confusion matrix
```

Out[95]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x19f12084978>
```



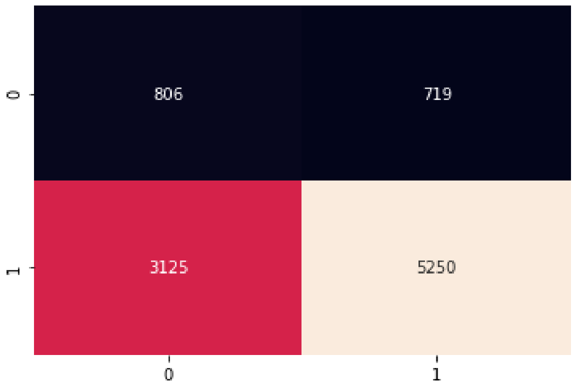
In [96]:

```
print("Test confusion matrix")
sns.heatmap(confusion_matrix(y_test, predict_with_best_t(y_test_pred_prob, best_t)), annot = True,
fmt = "d", cbar=False)
```

```
Test confusion matrix
```

Out[96]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x19f153facc0>
```



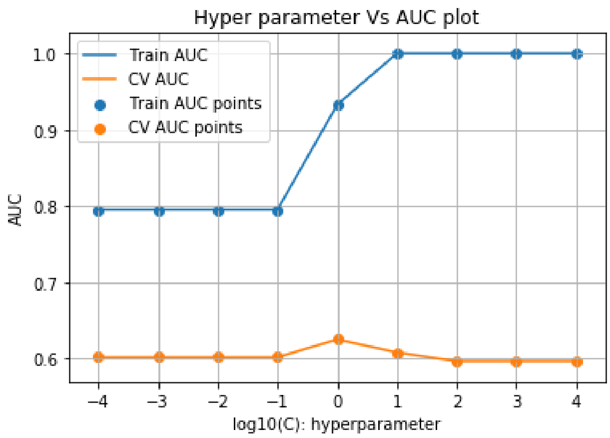
2.7.3) Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V)

In [97]:

```
X_train_set_3 =
hstack((x_train_cat_one_hot,x_train_subcat_one_hot,x_train_state_one_hot,x_train_teacher_prefix_one
_hot,\

x_train_grade_one_hot,x_train_teacher_number,x_train_price,x_train_quantity,x_train_avg_w2v_title,
x_train_avg_w2v_essay)).tocsr()
X_test_set_3 =
hstack((x_test_cat_one_hot,x_test_subcat_one_hot,x_test_state_one_hot,x_test_teacher_prefix_one_hot
,\
                              x_test_grade_one_hot,x_test_teacher_number,x_test_price,x_test_quantity,x_
test_avg_w2v_title,x_test_avg_w2v_essay)).tocsr()
```

In [98]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
# for class prior i referred https://scikit-
learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html,\
# https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
svm_ = SVC(class_weight = "balanced")
parameters = {'C':[10**x for x in range(-4,5)]}
clf = RandomizedSearchCV(svm_, parameters,n_iter = 9, scoring='roc_auc', return_train_score = True)
clf.fit(X_train_set_3, y_train)

results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_C'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
C_ =  results['param_C'].apply(lambda x: math.log10(x))

plt.plot(C_, train_auc, label='Train AUC')


plt.plot(C_, cv_auc, label='CV AUC')


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


plt.legend()
plt.xlabel("log10(C): hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()

results
```



Out[98]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params	split0_test_score	split1_test_score	s
0	1.398747	0.215468	0.352142	0.086094	0.0001	{'C': 0.0001}	0.571101	0.577782	0
1	1.332115	0.131517	0.304615	0.032164	0.001	{'C': 0.001}	0.571101	0.577782	0
2	1.333656	0.110522	0.356893	0.070069	0.01	{'C': 0.01}	0.571101	0.577782	0
3	1.261492	0.046357	0.307056	0.011949	0.1	{'C': 0.1}	0.571101	0.577782	0
4	1.100885	0.017553	0.288134	0.033315	1	{'C': 1}	0.586944	0.611567	0
5	0.956846	0.115930	0.199069	0.029583	10	{'C': 10}	0.548005	0.604505	0
6	1.054335	0.044270	0.192328	0.036437	100	{'C': 100}	0.525673	0.571292	0
7	1.069900	0.042723	0.177537	0.023571	1000	{'C': 1000}	0.525673	0.571292	0
8	1.129869	0.046232	0.180508	0.031636	10000	{'C': 10000}	0.525673	0.571292	0

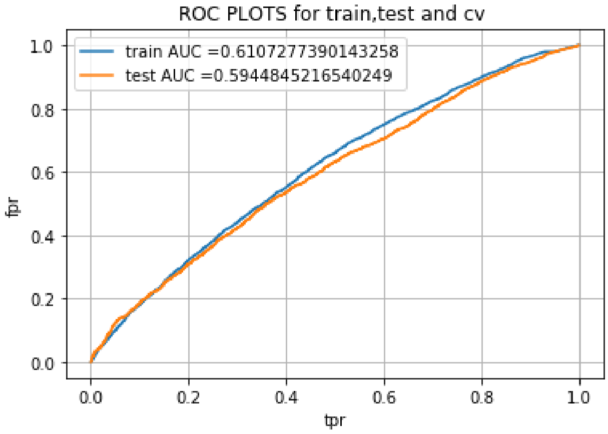
9 rows × 21 columns

In [99]:

```
# From the AUC plot, we find that the best value for "C" - "Inverse of Regularization Strength" fo
r the LogisticRegression is 0.01
best_C = 1
```

In [100]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


svm = SVC(C=best_C, class_weight = "balanced", probability = True)
svm.fit(X_train_set_3, y_train)


y_train_pred = svm.predict_proba(X_train_set_3)
y_test_pred =  svm.predict_proba(X_test_set_3)

y_train_pred_prob = []
y_test_pred_prob = []


for index in range(len(y_train_pred)):
    y_train_pred_prob.append(y_train_pred[index][1])

for index in range(len(y_test_pred)):
    y_test_pred_prob.append(y_test_pred[index][1])


train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred_prob)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred_prob)


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("tpr")
plt.ylabel("fpr")
plt.title("ROC PLOTS for train,test and cv ")
plt.grid()
plt.show()
```



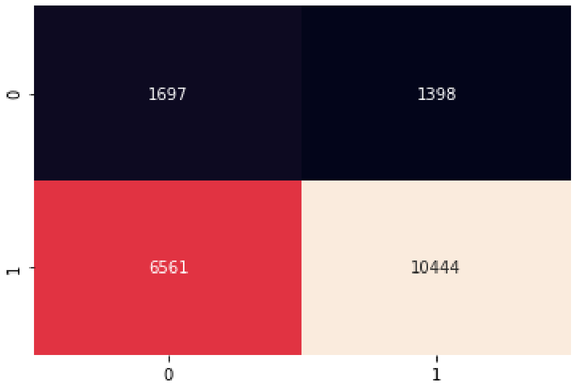
In [101]:

```python
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
sns.heatmap(confusion_matrix(y_train, predict_with_best_t(y_train_pred_prob, best_t)),annot = True,
fmt = "d", cbar=False)
```

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

the maximum value of tpr*(1-fpr) 0.3367529553932394 for threshold 0.833
Train confusion matrix
```

Out[101]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x19f120c7860>
```



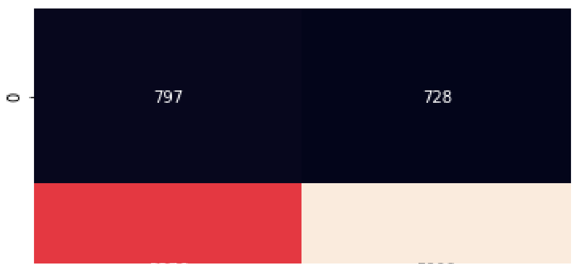
In [102]:

```python
print("Test confusion matrix")
sns.heatmap(confusion_matrix(y_test, predict_with_best_t(y_test_pred_prob, best_t)), annot = True,
fmt = "d", cbar=False)
```

```
Test confusion matrix
```

Out[102]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x19f12217240>
```



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

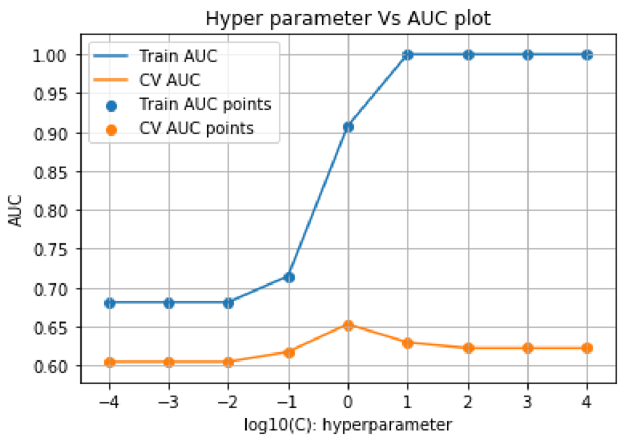
In [103]:

```
X_train_set_4 = 
hstack((x_train_cat_one_hot,x_train_subcat_one_hot,x_train_state_one_hot,x_train_teacher_prefix_one
_hot,\

x_train_grade_one_hot,x_train_teacher_number,x_train_price,x_train_quantity,x_train_weighted_w2v_ti
tle,x_train_weighted_w2v_essay)).tocsr()
X_test_set_4 = 
hstack((x_test_cat_one_hot,x_test_subcat_one_hot,x_test_state_one_hot,x_test_teacher_prefix_one_hot
,\
                         x_test_grade_one_hot,x_test_teacher_number,x_test_price,x_test_quantity,x_
test_weighted_w2v_title,x_test_weighted_w2v_essay)).tocsr()
# X_cv_set_4 = 
hstack((x_cv_cat_one_hot,x_cv_subcat_one_hot,x_cv_state_one_hot,x_cv_teacher_prefix_one_hot,\
# 
x_cv_grade_one_hot,x_cv_teacher_number,x_cv_price,x_cv_quantity,x_cv_weighted_w2v_title,x_cv_weigh
2v_essay)).tocsr()
```

In [105]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
# for class prior i referred https://scikit-
learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html,\
# https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
from sklearn.svm import SVC
import math


svm_ = SVC(class_weight = "balanced")
parameters = {'C':[10**x for x in range(-4,5)]}
clf = RandomizedSearchCV(svm_, parameters,n_iter = 9, scoring='roc_auc', return_train_score = True)
clf.fit(X_train_set_4, y_train)

results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_C'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
C_ =  results['param_C'].apply(lambda x: math.log10(x))

plt.plot(C_, train_auc, label='Train AUC')


plt.plot(C_, cv_auc, label='CV AUC')


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


plt.legend()
plt.xlabel("log10(C): hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
```

```
results.head()
```



Out[105]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params	split0_test_score	split1_test_score	s
0	35.134050	3.979885	8.050346	0.462151	0.0001	{'C': 0.0001}	0.610244	0.618615	0
1	31.116918	1.375108	7.439106	0.252824	0.001	{'C': 0.001}	0.610244	0.618615	0
2	30.181647	0.837014	7.158909	0.497679	0.01	{'C': 0.01}	0.610244	0.618615	0
3	29.608674	0.090885	6.831105	0.426381	0.1	{'C': 0.1}	0.623355	0.626846	0
4	26.262037	0.260796	6.191987	0.107065	1	{'C': 1}	0.644892	0.670905	0

5 rows × 21 columns

In [106]:

```
results
```

Out[106]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params	split0_test_score	split1_test_score	s
0	35.134050	3.979885	8.050346	0.462151	0.0001	{'C': 0.0001}	0.610244	0.618615	(
1	31.116918	1.375108	7.439106	0.252824	0.001	{'C': 0.001}	0.610244	0.618615	(
2	30.181647	0.837014	7.158909	0.497679	0.01	{'C': 0.01}	0.610244	0.618615	(
3	29.608674	0.090885	6.831105	0.426381	0.1	{'C': 0.1}	0.623355	0.626846	(
4	26.262037	0.260796	6.191987	0.107065	1	{'C': 1}	0.644892	0.670905	(
5	288.246052	534.938997	4.990601	0.316941	10	{'C': 10}	0.628328	0.657910	(
6	2209.564291	4377.014745	3.565383	0.304112	100	{'C': 100}	0.616846	0.654566	(
7	19.321585	0.698173	3.347260	0.205880	1000	{'C': 1000}	0.616846	0.654566	(
8	19.263534	0.685195	3.230254	0.063087	10000	{'C': 10000}	0.616846	0.654566	(

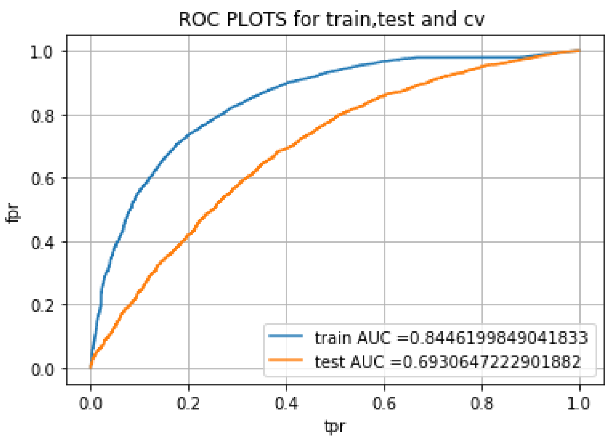
9 rows × 21 columns

In [107]:

```
# From the AUC plot, we find that the best value for "C" - "Inverse of Regularization Strength" fo
r the LogisticRegression is 0.01
best_C = 1
```

In [109]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc


svm = SVC(C=best_C, class_weight = "balanced", probability = True)
svm.fit(X_train_set_4, y_train)


y_train_pred = svm.predict_proba(X_train_set_4)
y_test_pred =  svm.predict_proba(X_test_set_4)

y_train_pred_prob = []
y_test_pred_prob = []


for index in range(len(y_train_pred)):
    y_train_pred_prob.append(y_train_pred[index][1])

for index in range(len(y_test_pred)):
    y_test_pred_prob.append(y_test_pred[index][1])


train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred_prob)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred_prob)


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("tpr")
plt.ylabel("fpr")
plt.title("ROC PLOTS for train,test and cv ")
plt.grid()
plt.show()
```



In [110]:

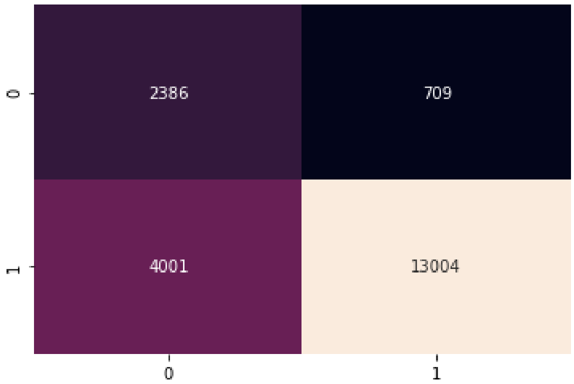
```
print("="*100)
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
sns.heatmap(confusion_matrix(y_train, predict_with_best_t(y_train_pred_prob, best_t)),annot = True,
fmt = "d", cbar=False)
```

==

```
the maximum value of tpr*(1-fpr) 0.5895357015113392 for threshold 0.818
Train confusion matrix
```

Out[110]:

<matplotlib.axes._subplots.AxesSubplot at 0x1e0beb6d358>



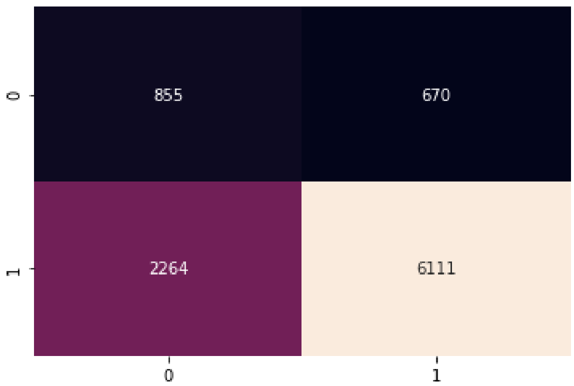
In [111]:

```
print("Test confusion matrix")
sns.heatmap(confusion_matrix(y_test, predict_with_best_t(y_test_pred_prob, best_t)), annot = True,
fmt = "d", cbar=False)
```

```
Test confusion matrix
```

Out[111]:

<matplotlib.axes._subplots.AxesSubplot at 0x1e0beb65ef0>



2.7.5) Calculate Sentiment Score for each essay (combined)

In [112]:

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

# import nltk
# nltk.download('vader_lexicon')
def compute_sentiment_score(df):
    score_list = []
    sid = SentimentIntensityAnalyzer()
    for essay in df['essay']:
        ss = sid.polarity_scores(essay)
        score_list.append(ss)
    return score_list
# we can use these 4 things as features/attributes (neg, neu, pos, compound)
# neg: 0.0, neu: 0.753, pos: 0.247, compound: 0.93
```

In [113]:

```
x_train_score = compute_sentiment_score(X_train)
x_test_score = compute_sentiment_score(X_test)
```

In [114]:

```
def populate_list(score_dicts):

    neg_score = []
    neu_score = []
    pos_score = []
    compound_score = []
    for dict_ in score_dicts:
        neg_score.append(dict_['neg'])
        neu_score.append(dict_['neu'])
        pos_score.append(dict_['pos'])
        compound_score.append(dict_['compound'])
    return neg_score, neu_score, pos_score, compound_score
```

In [115]:

```
x_train_neg, x_train_neu, x_train_pos, x_train_compound = populate_list(x_train_score)
x_test_neg, x_test_neu, x_test_pos, x_test_compound = populate_list(x_test_score)
# x_cv_neg, x_cv_neu, x_cv_pos, x_cv_compound = populate_list(x_cv_score)
```

In [116]:

```
X_train['words_project_title'] = X_train['project_title'].apply(lambda x: len(x.split()))
X_train['words_essay'] = X_train['essay'].apply(lambda x: len(x.split()))
```

In [117]:

```
X_test['words_project_title'] = X_test['project_title'].apply(lambda x: len(x.split()))
X_test['words_essay'] = X_test['essay'].apply(lambda x: len(x.split()))
```

Let's join the all the sentiment scores to the respective dataframes

In [118]:

```
# for training set
X_train['neg'] = x_train_neg
X_train['neu'] = x_train_neu
X_train['pos'] = x_train_pos
X_train['compound'] = x_train_compound

# for testing set
X_test['neg'] = x_test_neg
X_test['neu'] = x_test_neu
X_test['pos'] = x_test_pos
X_test['compound'] = x_test_compound
```

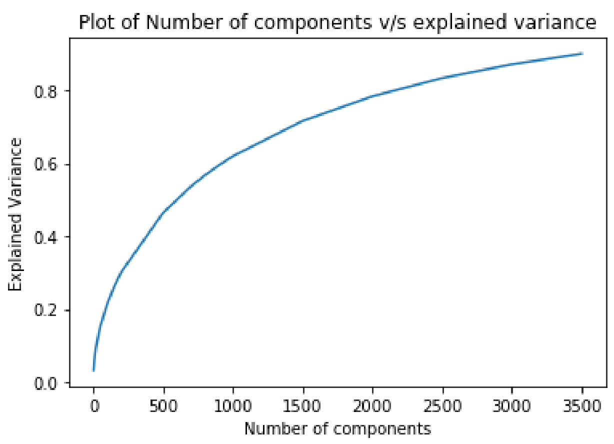
Applying truncatedSVD on Tfidf essay text

In [119]:

```
# Program to find the optimal number of components for Truncated SVD
# https://medium.com/swlh/truncated-singular-value-decomposition-svd-using-amazon-food-reviews-891
d97af5d8d
from sklearn.decomposition import TruncatedSVD
n_comp = [4,10,15,20,50,100,150,200,500,700,800,900,1000,1500,2000,2500,3000,3500] # list containin
g different values of components
explained = [] # explained variance ratio for each component of Truncated SVD
for x in n_comp:
    svd = TruncatedSVD(n_components=x)
    svd.fit(x_train_essay_tfidf)
    explained.append(svd.explained_variance_ratio_.sum())
    print("Number of components = %r and explained variance = %r"%(x,svd.explained_variance_ratio_
.sum()))
plt.plot(n_comp, explained)
```

```
plt.xlabel('Number of components')
plt.ylabel("Explained Variance")
plt.title("Plot of Number of components v/s explained variance")
plt.show()
```

```
Number of components = 4 and explained variance = 0.03125929895297198
Number of components = 10 and explained variance = 0.06107551180954257
Number of components = 15 and explained variance = 0.07862197951718307
Number of components = 20 and explained variance = 0.09290311719899019
Number of components = 50 and explained variance = 0.1504683693409316
Number of components = 100 and explained variance = 0.21331507982147596
Number of components = 150 and explained variance = 0.2606556273159695
Number of components = 200 and explained variance = 0.3001826777803587
Number of components = 500 and explained variance = 0.4628298246054989
Number of components = 700 and explained variance = 0.5360332728366037
Number of components = 800 and explained variance = 0.5664760413268588
Number of components = 900 and explained variance = 0.593938436624162
Number of components = 1000 and explained variance = 0.618724823238755
Number of components = 1500 and explained variance = 0.7155434520982675
Number of components = 2000 and explained variance = 0.7832046781722717
Number of components = 2500 and explained variance = 0.8331143430893263
Number of components = 3000 and explained variance = 0.871020598957353
Number of components = 3500 and explained variance = 0.9003365631505804
```



In [120]:

```
x_train_tfidf_essay_final = x_train_essay_tfidf[:,:2500]
x_train_tfidf_essay_final.shape
```

Out[120]:

```
(20100, 2500)
```

In [121]:

```
x_test_tfidf_essay_final = x_test_essay_tfidf[:,:2500]
x_test_tfidf_essay_final.shape
```

Out[121]:

```
(9900, 2500)
```

Prepare data set 5 and Apply SVM

NOTE: I am using same x_train_tfidf_essay_final for dataset 5 train and test part

In [122]:

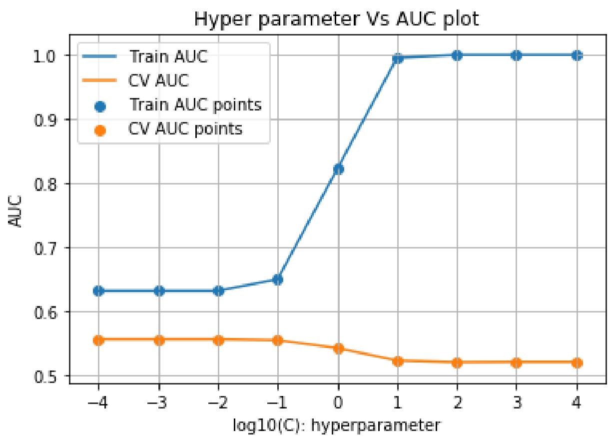
```
from scipy.sparse import hstack
```

In [123]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_train_set_5 =
hstack((x_train_cat_one_hot,x_train_subcat_one_hot,x_train_state_one_hot,x_train_teacher_prefix_one
_hot,\

x_train_grade_one_hot,x_train_teacher_number,x_train_price,x_train_quantity,\
                       X_train['neg'].values.reshape(-1,1), X_train['pos'].values.reshape(-1,1), X_t
rain['neu'].values.reshape(-1,1),\
                       X_train['compound'].values.reshape(-1,1),
x_train_tfidf_essay_final)).tocsr()
X_test_set_5 =
hstack((x_test_cat_one_hot,x_test_subcat_one_hot,x_test_state_one_hot,x_test_teacher_prefix_one_hot
,\
                         x_test_grade_one_hot,x_test_teacher_number,x_test_price,x_test_quantity,X_
test['neg'].values.reshape(-1,1),\
                      X_test['pos'].values.reshape(-1,1), X_test['neu'].values.reshape(-1,1), X_test
['compound'].values.reshape(-1,1),\
                      x_test_tfidf_essay_final)).tocsr()
```

In [129]:

```
svm_ = SVC(class_weight = "balanced")
parameters = {'C':[10**x for x in range(-4,5)]}
clf = RandomizedSearchCV(svm_, parameters,n_iter = 9, scoring='roc_auc', return_train_score = True)
clf.fit(X_train_set_5, y_train)

results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_C'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
C_ =  results['param_C'].apply(lambda x: math.log10(x))

plt.plot(C_, train_auc, label='Train AUC')


plt.plot(C_, cv_auc, label='CV AUC')


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


plt.legend()
plt.xlabel("log10(C): hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()

results
```



Out[129]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params	split0_test_score	split1_test_score	s
0	4.011217	0.386549	0.921342	0.085058	0.0001	{'C': 0.0001}	0.519868	0.580859	0
1	3.764693	0.182195	0.880555	0.048312	0.001	{'C': 0.001}	0.519868	0.580859	0
2	3.866731	0.171593	0.954829	0.105674	0.01	{'C': 0.01}	0.519868	0.580859	0
3	3.732203	0.313938	0.913110	0.096982	0.1	{'C': 0.1}	0.527205	0.577685	0
4	3.413536	0.181338	0.760193	0.019466	1	{'C': 1}	0.565764	0.550332	0
5	2.962850	0.347432	0.675489	0.082303	10	{'C': 10}	0.547347	0.484526	0
6	4.286309	0.088510	0.626696	0.049192	100	{'C': 100}	0.546737	0.497540	0
7	3.385508	0.330455	0.528986	0.035626	1000	{'C': 1000}	0.546737	0.497309	0
8	3.660696	0.539229	0.514163	0.022179	10000	{'C': 10000}	0.546737	0.497309	0

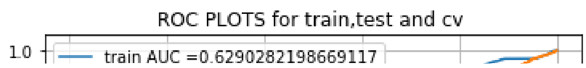
9 rows × 21 columns

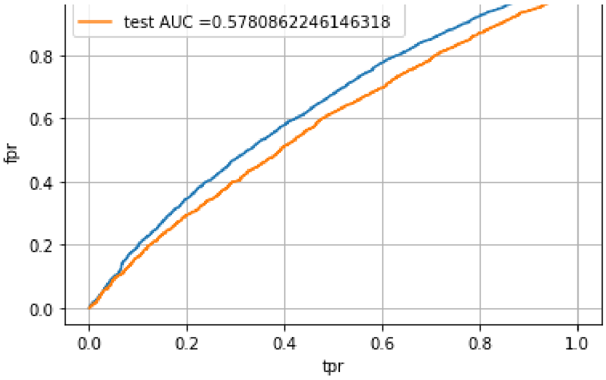
In [130]:

```
# Best value for Inverse Regularization Parameter from the table is 100
best_C = 1
```

In [132]:

```
from sklearn.metrics import roc_curve, auc


svm = SVC(C=best_C, class_weight = "balanced", probability = True)
svm.fit(X_train_set_5, y_train)


y_train_pred = svm.predict_proba(X_train_set_5)
y_test_pred =  svm.predict_proba(X_test_set_5)

y_train_pred_prob = []
y_test_pred_prob = []


for index in range(len(y_train_pred)):
    y_train_pred_prob.append(y_train_pred[index][1])

for index in range(len(y_test_pred)):
    y_test_pred_prob.append(y_test_pred[index][1])


train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred_prob)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred_prob)


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("tpr")
plt.ylabel("fpr")
plt.title("ROC PLOTS for train,test and cv ")
plt.grid()
plt.show()
```





In [133]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["Vectorizer", "Model", "Hyper-parameter", "Train AUC", "Test AUC"]

x.add_row(["BOW", "SVM (Brute)", 0.1, 0.6182, 0.5882])
x.add_row(["TFIDF", "SVM (Brute)", 1, 0.6741, 0.6037])
x.add_row(["AVG_W2V", "SVM (Brute)", 1, 0.6102, 0.5944])
x.add_row(["TFDIF_W2V", "SVM (Brute)", 1, 0.8446, 0.6930])
x.add_row(["TFIDF SVD Features", "SVM (Brute)", 1, 0.6290, 0.5780])

print(x)
```

```
+--------------------+-------------+-----------------+-----------+----------+
|     Vectorizer     |    Model    | Hyper-parameter | Train AUC | Test AUC |
+--------------------+-------------+-----------------+-----------+----------+
|        BOW         | SVM (Brute) |       0.1       |   0.6182  |  0.5882  |
|       TFIDF        | SVM (Brute) |        1        |   0.6741  |  0.6037  |
|      AVG_W2V       | SVM (Brute) |        1        |   0.6102  |  0.5944  |
|     TFDIF_W2V      | SVM (Brute) |        1        |   0.8446  |  0.693   |
| TFIDF SVD Features | SVM (Brute) |        1        |   0.629   |  0.578   |
+--------------------+-------------+-----------------+-----------+----------+
```