Assignment 7: SVM

Data splitting and pre-processing

In [2]:

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

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


import pickle
from tqdm import tqdm
import os

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

In [3]:

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

In [4]:

```
project_data.isnull().sum()
```

Out[4]:

```
Unnamed: 0                                      0
id                                              0
teacher_id                                      0
teacher_prefix                                  3
school_state                                    0
project_submitted_datetime                      0
project_grade_category                          0
project_subject_categories                      0
project_subject_subcategories                   0
project_title                                   0
project_essay_1                                 0
project_essay_2                                 0
```

```
project_essay_3                                 105490
project_essay_4                                 105490
project_resource_summary                             0
teacher_number_of_previously_posted_projects         0
project_is_approved                                  0
dtype: int64
```

In [5]:

```
#filling 3 null teacher prefix values with Teacher

project_data["teacher_prefix"].fillna("Teacher",inplace = True)
project_data.isnull().sum()
```

Out[5]:

```
Unnamed: 0                                           0
id                                                   0
teacher_id                                           0
teacher_prefix                                       0
school_state                                         0
project_submitted_datetime                           0
project_grade_category                               0
project_subject_categories                           0
project_subject_subcategories                        0
project_title                                        0
project_essay_1                                      0
project_essay_2                                      0
project_essay_3                                 105490
project_essay_4                                 105490
project_resource_summary                             0
teacher_number_of_previously_posted_projects         0
project_is_approved                                  0
dtype: int64
```

In [6]:

```
# 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 [7]:

```
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 [8]:

```
project_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 109248 entries, 0 to 109247
Data columns (total 20 columns):
Unnamed: 0                                      109248 non-null int64
id                                              109248 non-null object
teacher_id                                      109248 non-null object
teacher_prefix                                  109248 non-null object
school_state                                    109248 non-null object
project_submitted_datetime                      109248 non-null object
project_grade_category                          109248 non-null object
project_subject_categories                      109248 non-null object
project_subject_subcategories                   109248 non-null object
project_title                                   109248 non-null object
project_essay_1                                 109248 non-null object
project_essay_2                                 109248 non-null object
project_essay_3                                 3758 non-null object
project_essay_4                                 3758 non-null object
project_resource_summary                        109248 non-null object
teacher_number_of_previously_posted_projects    109248 non-null int64
project_is_approved                             109248 non-null int64
```

```
essay                                           109248 non-null object
price                                           109248 non-null float64
quantity                                        109248 non-null int64
dtypes: float64(1), int64(4), object(15)
memory usage: 17.5+ MB
```

In [9]:

```
#splitting data as 30% to test
y = project_data["project_is_approved"]
X = project_data.drop("project_is_approved",axis = 1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)
```

In [10]:

```
print(X_train.shape," ",y_train.shape)
print(X_test.shape," ",y_test.shape)
```

```
(76473, 19)    (76473,)
(32775, 19)    (32775,)
```

Preprocessing categorical Features

1. project subject categories

In [11]:

```
#using code from assignment
# project subject categories
catogories = list(X_train['project_subject_categories'].values)

cat_list = []
for i in catogories:
    temp = ""
    for j in i.split(','):
        if 'The' in j.split():
            j=j.replace('The','')
        j = j.replace(' ','')
        temp+=j.strip()+" "
        temp = temp.replace('&','_')
    cat_list.append(temp.strip())

X_train['clean_categories'] = cat_list
X_train.drop(['project_subject_categories'], axis=1, inplace=True)

from collections import Counter
my_counter = Counter()
for word in X_train['clean_categories'].values:
    my_counter.update(word.split())

cat_dict = dict(my_counter)
sorted_cat_dict = dict(sorted(cat_dict.items(), key=lambda kv: kv[1]))


# project subject categories for test data

catogories = list(X_test['project_subject_categories'].values)

cat_list = []
for i in catogories:
    temp = ""
    for j in i.split(','):
        if 'The' in j.split():
            j=j.replace('The','')
        j = j.replace(' ','')
        temp+=j.strip()+" "
        temp = temp.replace('&','_')
    cat_list.append(temp.strip())

X_test['clean_categories'] = cat_list
X_test.drop(['project_subject_categories'], axis=1, inplace=True)
```

```
X_test.drop(['project_subject_categories'], axis=1, inplace=True)
```

1. project subject sub_categories

In [12]:

```
sub_catogories = list(X_train['project_subject_subcategories'].values)
sub_cat_list = []
for i in sub_catogories:
    temp = ""
    for j in i.split(','):
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','')
        temp +=j.strip()+" "
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

X_train['clean_subcategories'] = sub_cat_list
X_train.drop(['project_subject_subcategories'], axis=1, inplace=True)

# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039
my_counter = Counter()
for word in X_train['clean_subcategories'].values:
    my_counter.update(word.split())

sub_cat_dict = dict(my_counter)
sorted_sub_cat_dict = dict(sorted(sub_cat_dict.items(), key=lambda kv: kv[1]))


sub_catogories = list(X_test['project_subject_subcategories'].values)
sub_cat_list = []
for i in sub_catogories:
    temp = ""
    for j in i.split(','):
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Scienc
e"=> "Math","&", "Science"
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i
.e removing 'The')
        j = j.replace(' ','')
        temp +=j.strip()+" "
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

X_test['clean_subcategories'] = sub_cat_list
X_test.drop(['project_subject_subcategories'], axis=1, inplace=True)
```

1. Teacher Prefix

In [13]:

```
#preprocessing teacher prefix
prefix = list(X_train['teacher_prefix'].values)
prefix_list = []
for i in prefix:
    temp = ""
    if "." in i:
            i=i.replace('.','')
    temp+=i.strip()+" "
    prefix_list.append(temp.strip())

X_train['clean_prefix'] = prefix_list

my_counter = Counter()
for word in X_train['clean_prefix'].values:
  my_counter.update(word.split())

prefix_dict = dict(my_counter)
sorted_prefix_dict = dict(sorted(prefix_dict.items(), key=lambda kv: kv[1]))
print(sorted_prefix_dict)
```

```
#preprocessing teacher prefix for test data
prefix = list(X_test['teacher_prefix'].values)
prefix_list = []
for i in prefix:
    temp = ""
    if "." in i:
            i=i.replace('.','')
    temp+=i.strip()+" "
    prefix_list.append(temp.strip())

X_test['clean_prefix'] = prefix_list
```

```
{'Dr': 9, 'Teacher': 1647, 'Mr': 7457, 'Ms': 27320, 'Mrs': 40040}
```

1. Project Grade Category

In [14]:

```
# preprocessing of grade category for train data

grade = list(X_train['project_grade_category'].values)
grade_list = []
for i in grade:
    temp = ""
    if "Grades" in i:
      i = i.replace("Grades","")
    if "6-8" in i:
      i = i.replace("6-8","six_eight")
    if "3-5" in i:
      i = i.replace("3-5","three_five")
    if "9-12" in i:
      i = i.replace("9-12","nine_twelve")
    if "PreK-2" in i:
      i = i.replace("PreK-2","prek_two")
    temp+=i.strip()+" "
    grade_list.append(temp.strip())

X_train['clean_grade'] = grade_list

my_counter = Counter()
for word in X_train['clean_grade'].values:
  my_counter.update(word.split())

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

# preprocessing of grade category for test data

grade = list(X_test['project_grade_category'].values)
grade_list = []
for i in grade:
    temp = ""
    if "Grades" in i:
      i = i.replace("Grades","")
    if "6-8" in i:
      i = i.replace("6-8","six_eight")
    if "3-5" in i:
      i = i.replace("3-5","three_five")
    if "9-12" in i:
      i = i.replace("9-12","nine_twelve")
    if "PreK-2" in i:
      i = i.replace("PreK-2","prek_two")
    temp+=i.strip()+" "
    grade_list.append(temp.strip())

X_test['clean_grade'] = grade_list
```

```
{'nine_twelve': 7670, 'six_eight': 11843, 'three_five': 25958, 'prek_two': 31002}
```

1. School State

In [15]:

```
#no need of preprocessing on school state

state = X_train["school_state"].value_counts()
sorted_state = dict(state)
sorted_state_dict = dict(sorted(sorted_state.items(), key=lambda kv: kv[1]))
X_train["clean_state"] = X_train["school_state"]

#similarly for X_test
X_test["clean_state"] = X_test["school_state"]
```

Preprocessing Numerical Feature

1. Standardizing price

In [16]:

```
from sklearn.preprocessing import StandardScaler


price_scalar = StandardScaler()
price_scalar.fit(project_data['price'].values.reshape(-1,1))
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")

#train data price standardization
price_standardized = price_scalar.transform(X_train['price'].values.reshape(-1, 1))

#test data price stanardization. Fit method applied on X_train
test_price_standardized = price_scalar.transform(X_test['price'].values.reshape(-1, 1))
```

```
Mean : 298.1193425966608, Standard deviation : 367.49634838483496
```

1. Standardizing quantity

In [17]:

```
price_scalar = StandardScaler()
price_scalar.fit(X_train["quantity"].values.reshape(-1, 1))
print(f"Mean of Quantity : {price_scalar.mean_[0]}, Standard deviation of Quantity :
{np.sqrt(price_scalar.var_[0])}")

#train data quantity standardization
quantity_standardized = price_scalar.transform(X_train["quantity"].values.reshape(-1, 1))

#test data quantity stanardization. Fit method applied on X_train
test_quantity_standardized = price_scalar.transform(X_test["quantity"].values.reshape(-1, 1))
```

```
C:\Users\rdbz3b\AppData\Local\Continuum\anaconda3\lib\site-
packages\sklearn\utils\validation.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.


Mean of Quantity : 16.951629987054254, Standard deviation of Quantity : 25.894395919389655

C:\Users\rdbz3b\AppData\Local\Continuum\anaconda3\lib\site-
packages\sklearn\utils\validation.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.

C:\Users\rdbz3b\AppData\Local\Continuum\anaconda3\lib\site-
packages\sklearn\utils\validation.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.
```

1. Standardizing number of ppp

In [18]:

```
price_scalar = StandardScaler()
price_scalar.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")

#train data ppp standardization
number_ppp_standardized =
price_scalar.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,
1))

#test data price stanardization. Fit method applied on X_train
test_number_ppp_standardized =
price_scalar.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1)
)
```

```
C:\Users\rdbz3b\AppData\Local\Continuum\anaconda3\lib\site-
packages\sklearn\utils\validation.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.
```

```
Mean : 11.09995684751481, Standard deviation : 27.56154092526415
```

```
C:\Users\rdbz3b\AppData\Local\Continuum\anaconda3\lib\site-
packages\sklearn\utils\validation.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.

C:\Users\rdbz3b\AppData\Local\Continuum\anaconda3\lib\site-
packages\sklearn\utils\validation.py:595: DataConversionWarning:

Data with input dtype int64 was converted to float64 by StandardScaler.
```

Preprocessing of Text Feature for both test and train data

In [19]:

```
#using function and stopwords form assignemnt

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

# 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', "d
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"]
```

1. preprocessing of project essay

In [20]:

```
from tqdm import tqdm

#for train data
preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(X_train['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_essays.append(sent.lower().strip())

test_preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(X_test['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    test_preprocessed_essays.append(sent.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████| 76473/76473
[00:45<00:00, 1678.67it/s]
100%|████████████████████████████████████████████████████████████████| 32775/32775
[00:20<00:00, 1634.39it/s]
```

1. preprocessing of project title

In [21]:

```
preprocessed_title = []
# tqdm is for printing the status bar
for sentance in tqdm(X_train['project_title'].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_title.append(sent.lower().strip())
```

```
# for test data
test_preprocessed_title = []
# tqdm is for printing the status bar
for sentance in tqdm(X_test['project_title'].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)
    test_preprocessed_title.append(sent.lower().strip())
```

```
100%|████████████████████████████████████████████████████████████████| 76473/76473
[00:02<00:00, 36634.63it/s]
100%|████████████████████████████████████████████████████████████████| 32775/32775
[00:00<00:00, 35620.39it/s]
```

Vectorizing of Categorical data

1. Vectorizing project categories and subcategories

In [22]:

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

# fitting on train data
vectorizer.fit(X_train['clean_categories'].values)
print(vectorizer.get_feature_names())
categories_feature = vectorizer.get_feature_names()

# for train data
categories_one_hot = vectorizer.transform(X_train['clean_categories'].values)

print("Shape of matrix after one hot encodig ",categories_one_hot.shape)

# for test data
test_categories_one_hot = vectorizer.transform(X_test['clean_categories'].values)
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds',
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encodig  (76473, 9)
```

1. Vectorizing project subcategories

In [23]:

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

# fitting on train data
vectorizer.fit(X_train['clean_subcategories'].values)
print(vectorizer.get_feature_names())
subcategories_feature = vectorizer.get_feature_names()

# for train data
sub_categories_one_hot = vectorizer.transform(X_train['clean_subcategories'].values)
print("Shape of matrix after one hot encodig ",sub_categories_one_hot.shape)

# for test data
test_sub_categories_one_hot = vectorizer.transform(X_test['clean_subcategories'].values)
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'Music', 'History_Geography', 'Health_LifeScience', 'EarlyDevelopment', 'Gym
_Fitness', 'ESL', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
```

```
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix after one hot encodig  (76473, 30)
```

1. vectorizing teacher prefix

In [24]:

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

# fitting on train data
vectorizer.fit(X_train['clean_prefix'].values)
print(vectorizer.get_feature_names())
prefix_feature = vectorizer.get_feature_names()
# for train data
prefix_one_hot = vectorizer.transform(X_train['clean_prefix'].values)
print("Shape of matrix after one hot encodig ",prefix_one_hot.shape)

# for test data
test_prefix_one_hot = vectorizer.transform(X_test['clean_prefix'].values)
```

```
['Ms', 'Mrs', 'Mr', 'Teacher', 'Dr']
Shape of matrix after one hot encodig  (76473, 5)
```

1. Vectorizing school state and grade

In [25]:

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

# fitting on train data
vectorizer.fit(X_train['clean_grade'].values)
print(vectorizer.get_feature_names())
grade_feature = vectorizer.get_feature_names()
# for train data
grade_one_hot = vectorizer.transform(X_train['clean_grade'].values)
print("Shape of matrix after one hot encodig ",grade_one_hot.shape)

# for test data
test_grade_one_hot = vectorizer.transform(X_test['clean_grade'].values)


vectorizer = CountVectorizer(vocabulary=list(sorted_state_dict.keys()), lowercase=False, binary=Tr
ue)
vectorizer.fit(X_train['clean_state'].values)
print(vectorizer.get_feature_names())
state_one_hot = vectorizer.transform(X_train['clean_state'].values)
state_feature = vectorizer.get_feature_names()
test_state_one_hot = vectorizer.transform(X_test['clean_state'].values)
```

```
['prek_two', 'nine_twelve', 'three_five', 'six_eight']
Shape of matrix after one hot encodig  (76473, 4)
['VT', 'WY', 'ND', 'MT', 'RI', 'NE', 'SD', 'DE', 'AK', 'NH', 'WV', 'ME', 'HI', 'DC', 'NM', 'KS', 'I
A', 'ID', 'AR', 'CO', 'MN', 'KY', 'OR', 'MS', 'NV', 'MD', 'CT', 'TN', 'UT', 'AL', 'WI', 'VA', 'AZ',
'OK', 'NJ', 'WA', 'MA', 'LA', 'OH', 'MO', 'IN', 'PA', 'MI', 'SC', 'GA', 'IL', 'NC', 'FL', 'NY', 'TX
', 'CA']
```

Vectorizing Text Feature

1. BOW

In [26]:

```
vectorizer = CountVectorizer(min_df=10,ngram_range=(2,2),max_features=5000)
#fit using train data
vectorizer.fit(preprocessed_essays)
essay_feature = vectorizer.get_feature_names()
```

```
# for train data
text_bow = vectorizer.transform(preprocessed_essays)
print("Shape of train matrix : ",text_bow.shape)
# for test data
test_text_bow = vectorizer.transform(test_preprocessed_essays)
print("Shape of test matrix : ",test_text_bow.shape)


# for title
vectorizer.fit(preprocessed_title)
title_feature = vectorizer.get_feature_names()

# for train data
title_bow = vectorizer.transform(preprocessed_title)
print("Shape of train matrix : ",title_bow.shape)
# for test data
test_title_bow = vectorizer.transform(test_preprocessed_title)
print("Shape of test matrix : ",test_title_bow.shape)
```

```
Shape of train matrix :  (76473, 5000)
Shape of test matrix :  (32775, 5000)
Shape of train matrix :  (76473, 2843)
Shape of test matrix :  (32775, 2843)
```

1. TFIDF

In [27]:

```
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(2,2),max_features=5000)
#fit using train data
vectorizer.fit(preprocessed_essays)
essay_feature_tfidf = vectorizer.get_feature_names()
# for train data
text_tfidf = vectorizer.transform(preprocessed_essays)
print("Shape of train matrix : ",text_tfidf.shape)
# for test data
test_text_tfidf = vectorizer.transform(test_preprocessed_essays)
print("Shape of test matrix : ",test_text_tfidf.shape)


# for title
vectorizer.fit(preprocessed_title)
title_feature_tfidf = vectorizer.get_feature_names()

# for train data
title_tfidf = vectorizer.transform(preprocessed_title)
print("Shape of train matrix : ",title_tfidf.shape)
# for test data
test_title_tfidf = vectorizer.transform(test_preprocessed_title)
print("Shape of test matrix : ",test_title_tfidf.shape)
```

```
Shape of train matrix :  (76473, 5000)
Shape of test matrix :  (32775, 5000)
Shape of train matrix :  (76473, 2843)
Shape of test matrix :  (32775, 2843)
```

1. Avg W2v

In [28]:

```
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [29]:

```
# for train data
avg_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays): # 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)
print(len(avg_w2v_vectors))
print(len(avg_w2v_vectors[0]))

# for test data
test_avg_w2v_vectors = [] # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(test_preprocessed_essays): # 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
    test_avg_w2v_vectors.append(vector)

print(len(test_avg_w2v_vectors))
print(len(test_avg_w2v_vectors[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 76473/76473
[00:25<00:00, 3032.42it/s]
```

```
76473
300
```

```
100%|██████████████████████████████████████████████████████████████████████████| 32775/32775
[00:11<00:00, 2824.59it/s]
```

```
32775
300
```

In [30]:

```
title_avg_w2v_vectors = []
for sentence in tqdm(preprocessed_title):
    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
    title_avg_w2v_vectors.append(vector)

print(len(title_avg_w2v_vectors))
print(len(title_avg_w2v_vectors[0]))

# for test data
test_title_avg_w2v_vectors = []
for sentence in tqdm(test_preprocessed_title):
    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
    test_title_avg_w2v_vectors.append(vector)

print(len(test_title_avg_w2v_vectors))
print(len(test_title_avg_w2v_vectors[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 76473/76473
```

```
[00:01<00:00, 56320.76it/s]
```

```
76473
300
```

```
100%|██████████| 32775/32775
[00:00<00:00, 61539.59it/s]
```

```
32775
300
```

1. TFIDF avgw2v

In [31]:

```
test_tfidf_model = TfidfVectorizer()
test_tfidf_model.fit(preprocessed_essays)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(test_tfidf_model.get_feature_names(), list(test_tfidf_model.idf_)))
tfidf_words = set(test_tfidf_model.get_feature_names())

test_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(test_preprocessed_essays): # 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
    test_tfidf_w2v_vectors.append(vector)

print(len(test_tfidf_w2v_vectors))
print(len(test_tfidf_w2v_vectors[0]))


# for title
test_tfidf_model.fit(preprocessed_title)

dictionary = dict(zip(test_tfidf_model.get_feature_names(), list(test_tfidf_model.idf_)))
tfidf_words = set(test_tfidf_model.get_feature_names())


test_title_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(test_preprocessed_title): # 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
    test_title_tfidf_w2v_vectors.append(vector)

print(len(test_title_tfidf_w2v_vectors))
```

```
100%|██████████| 32775/32775 [01:
10<00:00, 465.12it/s]
```

```
32775
300
```

```
100%|██████████| 32775/32775 [00:01<00:00, 32773.61it/s]
```

```
32775
```

In [32]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(preprocessed_essays)
# 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())

tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays): # 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.append(vector)

print(len(tfidf_w2v_vectors))
print(len(tfidf_w2v_vectors[0]))

# for title
tfidf_model.fit(preprocessed_title)

dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())


title_tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_title): # 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
    title_tfidf_w2v_vectors.append(vector)

print(len(title_tfidf_w2v_vectors))
```

```
100%|██████████| 76473/76473 [02:48<00:00, 452.88it/s]
```

```
76473
300
```

```
100%|██████████| 76473/76473 [00:02<00:00, 28290.89it/s]
```

```
76473
```

Printing all

In [33]:

```
print("*"*70)
print("Categorical Features that are considered :- ")
print("Subject Categories :- ",categories_one_hot.shape)
print("Subject Sub-Categories :- ",sub_categories_one_hot.shape)
print("Sudent Grade :- ",grade_one_hot.shape)
print("School State :- ",state_one_hot.shape)
print("Teacher Prefix :- ",prefix_one_hot.shape)
print("*"*70)
```

```
**********************************************************************
Categorical Features that are considered :-
Subject Categories :-  (76473, 9)
Subject Sub-Categories :-  (76473, 30)
Sudent Grade :-  (76473, 4)
School State :-  (76473, 51)
Teacher Prefix :-  (76473, 5)
**********************************************************************
```

In [34]:

```
print("Text Features that are considered :- ")
print("*"*70)
print("Project Essay BOW:- ",text_bow.shape)
print("Project Essay TFIDF:- ",text_tfidf.shape)
print("*"*70)
print("Project Title BOW:- ",title_bow.shape)
print("Project Title TFIDF:- ",title_tfidf.shape)
print("*"*70)
```

```
Text Features that are considered :-
**********************************************************************
Project Essay BOW:-  (76473, 5000)
Project Essay TFIDF:-  (76473, 5000)
**********************************************************************
Project Title BOW:-  (76473, 2843)
Project Title TFIDF:-  (76473, 2843)
**********************************************************************
```

sets

In [35]:

```
#combining all feature into one
from scipy.sparse import hstack

set1 =
hstack((categories_one_hot,sub_categories_one_hot,prefix_one_hot,grade_one_hot,state_one_hot,price_
standardized,quantity_standardized,number_ppp_standardized,text_bow,title_bow))
set1_t =
hstack((test_categories_one_hot,test_sub_categories_one_hot,test_prefix_one_hot,test_grade_one_hot
,test_state_one_hot,test_price_standardized,test_quantity_standardized,test_number_ppp_standardizec
,test_text_bow,test_title_bow))

set2 =
hstack((categories_one_hot,sub_categories_one_hot,prefix_one_hot,state_one_hot,grade_one_hot,text_t
fidf,title_tfidf,price_standardized,quantity_standardized,number_ppp_standardized))
set2_t =
hstack((test_categories_one_hot,test_sub_categories_one_hot,test_prefix_one_hot,test_state_one_hot
,test_grade_one_hot,test_text_tfidf,test_title_tfidf,test_price_standardized,test_quantity_standarc
ized,test_number_ppp_standardized))

set3 =
hstack((categories_one_hot,sub_categories_one_hot,prefix_one_hot,state_one_hot,grade_one_hot,price_
standardized,quantity_standardized,number_ppp_standardized,avg_w2v_vectors,title_avg_w2v_vectors))
set3_t =
```

```
hstack((test_categories_one_hot,test_sub_categories_one_hot,test_prefix_one_hot,test_state_one_hot
,test_grade_one_hot,test_price_standardized,test_quantity_standardized,test_number_ppp_standardizec
,test_avg_w2v_vectors,test_title_avg_w2v_vectors))

set4 =
hstack((categories_one_hot,sub_categories_one_hot,prefix_one_hot,state_one_hot,grade_one_hot,price_
standardized,quantity_standardized,number_ppp_standardized,tfidf_w2v_vectors,title_tfidf_w2v_vector
s))
set4_t =
hstack((test_categories_one_hot,test_sub_categories_one_hot,test_prefix_one_hot,test_state_one_hot
,test_grade_one_hot,test_price_standardized,test_quantity_standardized,test_number_ppp_standardizec
,test_tfidf_w2v_vectors,test_title_tfidf_w2v_vectors))


print(set1.shape,"\t",set1_t.shape)
print(set2.shape,"\t",set2_t.shape)
print(set3.shape,"\t",set3_t.shape)
print(set4.shape,"\t",set4_t.shape)
```

```
(76473, 7945)    (32775, 7945)
(76473, 7945)    (32775, 7945)
(76473, 702)    (32775, 702)
(76473, 702)    (32775, 702)
```

In [36]:

```
set_feature = categories_feature + subcategories_feature + prefix_feature + grade_feature + state_f
eature + essay_feature_tfidf + title_feature_tfidf


set_feature.append("price")
set_feature.append("quantity")
set_feature.append("number")
print(len(set_feature))
```

```
7945
```

In [37]:

```
par_grid = dict(penalty = ["l1","l2"],alpha=[0.00001,0.0001,0.001,0.1,1,10,100,1000,10000])
alpha=[0.00001,0.0001,0.001,0.1,1,10,100,1000,10000]
```

In [38]:

```
from sklearn.calibration import CalibratedClassifierCV
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve
from sklearn.linear_model import SGDClassifier
```

SET1 (BOW)

In [39]:

```
sgd = SGDClassifier(class_weight="balanced")

#using balanced class weight = "balanced" as result using it was much better than "none"

grid = GridSearchCV(sgd,par_grid,scoring="roc_auc",n_jobs=-1,cv=10)
```

In [40]:

```
grid.fit(set1,y_train)
print(grid.best_estimator_)
print(grid.best_index_)
print(grid.best_params_)
print(grid.best_score_)
```

```
SGDClassifier(alpha=0.001, average=False, class_weight='balanced',
```

```
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False)
5
{'alpha': 0.001, 'penalty': 'l2'}
0.6713670013694405
```

In [41]:

```
sgd = SGDClassifier(alpha=0.001,class_weight="balanced")
ccv = CalibratedClassifierCV(sgd,cv=10)
ccv.fit(set1,y_train)
print(ccv.get_params)
```

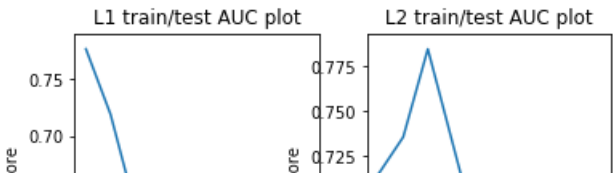
```
<bound method BaseEstimator.get_params of
CalibratedClassifierCV(base_estimator=SGDClassifier(alpha=0.001, average=False,
class_weight='balanced',
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False),
            cv=10, method='sigmoid')>
```

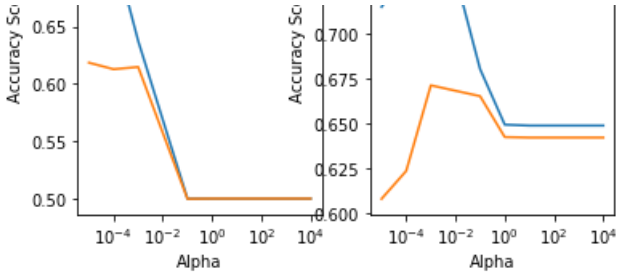
In [42]:

```
#converting results to dataframe
df = pd.DataFrame(data = grid.cv_results_)

# getting into list
l1_train_score = []
l1_test_score = []
l2_train_score = []
l2_test_score = []

for i in range(len(df)):
  if df.iloc[i]["param_penalty"] =="l1":
    l1_test_score.append(df.iloc[i]["mean_test_score"])
    l1_train_score.append(df.iloc[i]["mean_train_score"])

  if df.iloc[i]["param_penalty"] =="l2":
    l2_test_score.append(df.iloc[i]["mean_test_score"])
    l2_train_score.append(df.iloc[i]["mean_train_score"])
```

In [43]:

```
plt.figure()
plt.subplot(121)
plt.plot(alpha,l1_train_score)
plt.plot(alpha,l1_test_score)
plt.xscale("log")
plt.xlabel("Alpha")
plt.ylabel("Accuracy Score")
plt.title("L1 train/test AUC plot")
plt.subplot(122)
plt.plot(alpha,l2_train_score)
plt.plot(alpha,l2_test_score)
plt.xscale("log")
plt.xlabel("Alpha")
plt.ylabel("Accuracy Score")
plt.title("L2 train/test AUC plot")
plt.show()
```





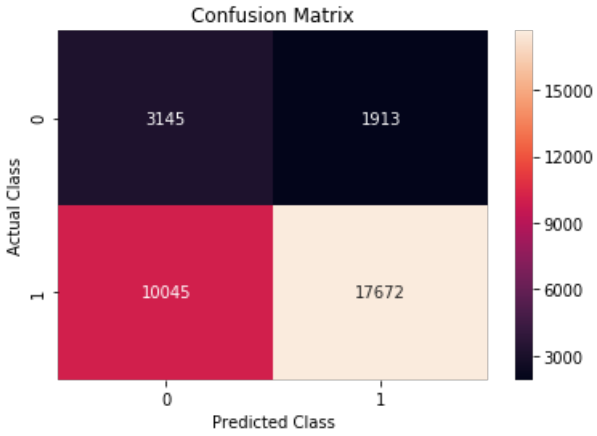
Confusion Matrix

In [44]:

```
y1_predict = grid.predict(set1_t)
cm1 = confusion_matrix(y_test,y1_predict)
# https://seaborn.pydata.org/generated/seaborn.heatmap.html
sns.heatmap(cm1, annot=True, fmt="d")
plt.ylabel("Actual Class")
plt.xlabel("Predicted Class")
plt.title("Confusion Matrix")
```

Out[44]:

```
Text(0.5, 1.0, 'Confusion Matrix')
```



AUC plotting

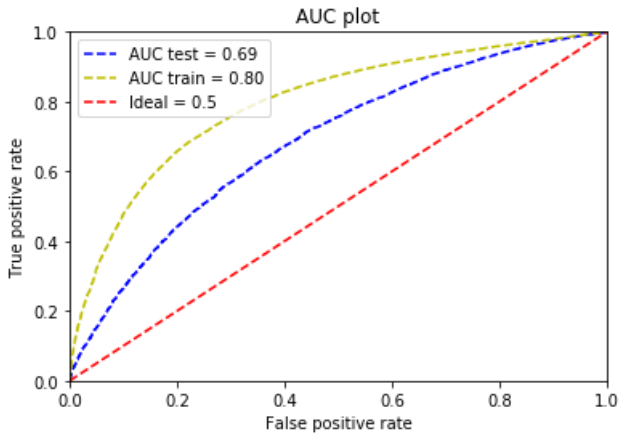
In [45]:

```
# probabilities calcultion
y1_predict_prob = ccv.predict_proba(set1_t)[:,1]
y1_predict_prob_train = ccv.predict_proba(set1)[:,1]

# took referance from https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
#fpr,tpr
fpr,tpr,thre = roc_curve(y_test,y1_predict_prob)

# am i doing it right here.....?
fpr_train,tpr_train,thre_train = roc_curve(y_train,y1_predict_prob_train)
```

In [46]:

```
# auc calculation for test data
roc_auc1 = metrics.auc(fpr,tpr)

# auc calculation for train data
roc_auc_train1 = metrics.auc(fpr_train,tpr_train)

# took referance from https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
plt.plot(fpr,tpr,"b--",label = 'AUC test = %0.2f'%roc_auc1)
plt.plot(fpr_train,tpr_train,"y--",label = 'AUC train = %0.2f'%roc_auc_train1)
plt.title("AUC plot")
```

```
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
plt.plot([0, 1], [0, 1],'r--',label = "Ideal = 0.5")
plt.xlim([0,1])
plt.ylim([0,1])
plt.legend(loc = "upper left")
plt.show()
```



Set2 (TFIDF)

In [47]:

```
grid.fit(set2,y_train)
print(grid.best_estimator_)
print(grid.best_index_)
print(grid.best_params_)
print(grid.best_score_)
```

```
SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l1',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False)
2
{'alpha': 0.0001, 'penalty': 'l1'}
0.6643859989303158
```

In [48]:

```
sgd =
SGDClassifier(alpha=grid.best_params_["alpha"],class_weight="balanced",penalty=grid.best_params_["
penalty"])
ccv = CalibratedClassifierCV(sgd,cv=10)
ccv.fit(set2,y_train)
print(ccv.get_params)
```

```
<bound method BaseEstimator.get_params of
CalibratedClassifierCV(base_estimator=SGDClassifier(alpha=0.0001, average=False,
class_weight='balanced',
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l1',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False),
            cv=10, method='sigmoid')>
```

In [49]:

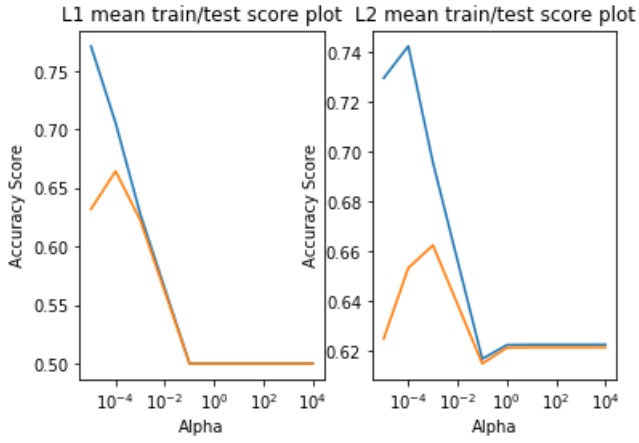
```
#converting results to dataframe
df = pd.DataFrame(data = grid.cv_results_)

# getting into list
l1_train_score = []
```

```
l1_train_score = []
l1_test_score = []
l2_train_score = []
l2_test_score = []

for i in range(len(df)):
  if df.iloc[i]["param_penalty"] =="l1":
    l1_test_score.append(df.iloc[i]["mean_test_score"])
    l1_train_score.append(df.iloc[i]["mean_train_score"])

  if df.iloc[i]["param_penalty"] =="l2":
    l2_test_score.append(df.iloc[i]["mean_test_score"])
    l2_train_score.append(df.iloc[i]["mean_train_score"])

print(l1_train_score)
print(l1_test_score )
print(l2_train_score)
print(l2_test_score)
```

```
[0.7711956820598286, 0.7056481370322893, 0.6278956430004208, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]
[0.6319330965945366, 0.6643859989303158, 0.6226427908628587, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]
[0.729627638378256, 0.7424725006884214, 0.6956403582683837, 0.616897971263666, 0.6225206342773695,
0.6226272610502457, 0.6226273492253667, 0.6226273492253667, 0.6226273492253667]
[0.6250498058608464, 0.6534576279197091, 0.6625108823454687, 0.6150018560995887,
0.6213947934698029, 0.6215429509403063, 0.6215430714363522, 0.6215430714363522,
0.6215430714363522]
```

In [50]:

```
plt.figure()
plt.subplot(121)
plt.plot(alpha,l1_train_score)
plt.plot(alpha,l1_test_score)
plt.xscale("log")
plt.xlabel("Alpha")
plt.ylabel("Accuracy Score")
plt.title("L1 mean train/test score plot")
plt.subplot(122)
plt.plot(alpha,l2_train_score)
plt.plot(alpha,l2_test_score)
plt.xscale("log")
plt.xlabel("Alpha")
plt.ylabel("Accuracy Score")
plt.title("L2 mean train/test score plot")
plt.show()
```



Confusion matrix

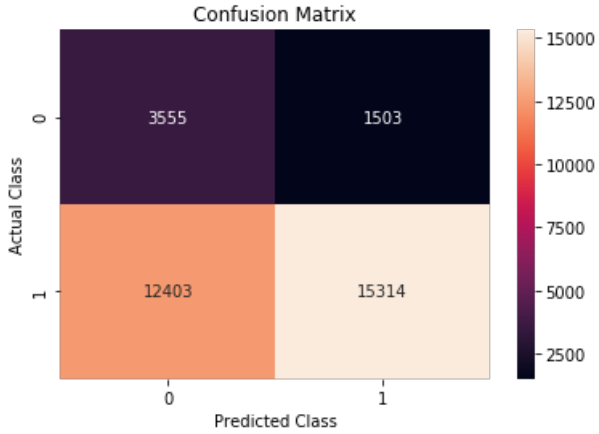
In [51]:

```
y2_predict = grid.predict(set2_t)
cm2 = confusion_matrix(y_test,y2_predict)
# https://seaborn.pydata.org/generated/seaborn.heatmap.html
sns.heatmap(cm2, annot=True, fmt="d")
plt.ylabel("Actual Class")
plt.xlabel("Predicted Class")
```

```
plt.title("Confusion Matrix")
```

Out[51]:

```
Text(0.5, 1.0, 'Confusion Matrix')
```



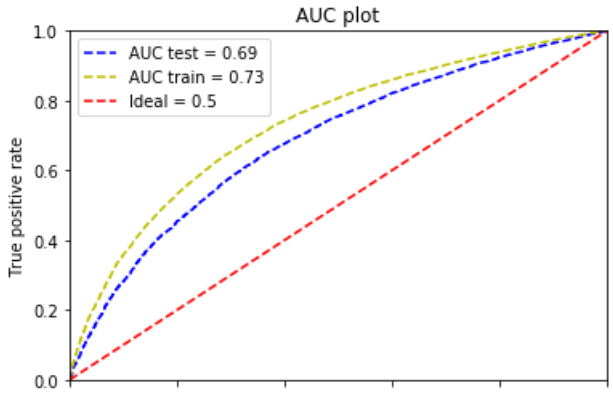
AUC

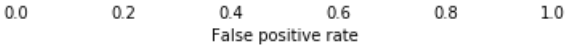
In [52]:

```
# probabilities calcultion
y2_predict_prob = ccv.predict_proba(set2_t)[:,1]
y2_predict_prob_train = ccv.predict_proba(set2)[:,1]

# took referance from https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
#fpr,tpr
fpr,tpr,thre = roc_curve(y_test,y2_predict_prob)

# am i doing it right here......?
fpr_train,tpr_train,thre_train = roc_curve(y_train,y2_predict_prob_train)
```

In [53]:

```
# auc calculation for test data
roc_auc2 = metrics.auc(fpr,tpr)

# auc calculation for train data
roc_auc_train2 = metrics.auc(fpr_train,tpr_train)

# took referance from https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
plt.plot(fpr,tpr,"b--",label = 'AUC test = %0.2f'%roc_auc2)
plt.plot(fpr_train,tpr_train,"y--",label = 'AUC train = %0.2f'%roc_auc_train2)
plt.title("AUC plot")
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
plt.plot([0, 1], [0, 1],'r--',label ="Ideal = 0.5")
plt.xlim([0,1])
plt.ylim([0,1])
plt.legend(loc = "upper left")
plt.show()
```





Set3 (Avg W2V)

In [54]:

```
grid = GridSearchCV(sgd,par_grid,scoring="roc_auc",n_jobs=-1,cv=10)
grid.fit(set3,y_train)
print(grid.best_estimator_)
print(grid.best_index_)
print(grid.best_params_)
print(grid.best_score_)
```

```
SGDClassifier(alpha=0.001, average=False, class_weight='balanced',
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False)
5
{'alpha': 0.001, 'penalty': 'l2'}
0.6898722244176072
```

In [55]:

```
sgd =
SGDClassifier(alpha=grid.best_params_["alpha"],class_weight="balanced",penalty=grid.best_params_["
penalty"])
ccv = CalibratedClassifierCV(sgd,cv=10)
ccv.fit(set3,y_train)
print(ccv.get_params)
```

```
<bound method BaseEstimator.get_params of
CalibratedClassifierCV(base_estimator=SGDClassifier(alpha=0.001, average=False,
class_weight='balanced',
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False),
            cv=10, method='sigmoid')>
```

In [56]:

```
#converting results to dataframe
df = pd.DataFrame(data = grid.cv_results_)

# getting into list
l1_train_score = []
l1_test_score = []
l2_train_score = []
l2_test_score = []

for i in range(len(df)):
  if df.iloc[i]["param_penalty"] =="l1":
    l1_test_score.append(df.iloc[i]["mean_test_score"])
    l1_train_score.append(df.iloc[i]["mean_train_score"])

  if df.iloc[i]["param_penalty"] =="l2":
    l2_test_score.append(df.iloc[i]["mean_test_score"])
    l2_train_score.append(df.iloc[i]["mean_train_score"])

print(l1_train_score)
print(l1_test_score )
print(l2_train_score)
print(l2_test_score)
```

```
[0.6952675553506407, 0.7023841517798984, 0.6910450895707014, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]
[0.6731981961850698, 0.6875299992462821, 0.6830276960516257, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]
[0.6674706271249987, 0.6819791106428462, 0.7053584602089862, 0.6443475210746894,
```

```
0.6384886081060215, 0.6385540012326143, 0.63855407921698, 0.6385534867118401, 0.638553579156604]
[0.6513451514851318, 0.6685588686395242, 0.6898722244176072, 0.6406571319779433,
0.6363145752346241, 0.6364610563592599, 0.6364600639967104, 0.6364604524898966,
0.6364599565536718]
```

In [57]:

```
plt.figure()
plt.subplot(121)
plt.plot(alpha,l1_train_score)
plt.plot(alpha,l1_test_score)
plt.xscale("log")
plt.xlabel("Alpha")
plt.ylabel("Accuracy Score")
plt.title("L1 mean train/test score plot")
plt.subplot(122)
plt.plot(alpha,l2_train_score)
plt.plot(alpha,l2_test_score)
plt.xscale("log")
plt.xlabel("Alpha")
plt.ylabel("Accuracy Score")
plt.title("L2 mean train/test score plot")
plt.show()
```

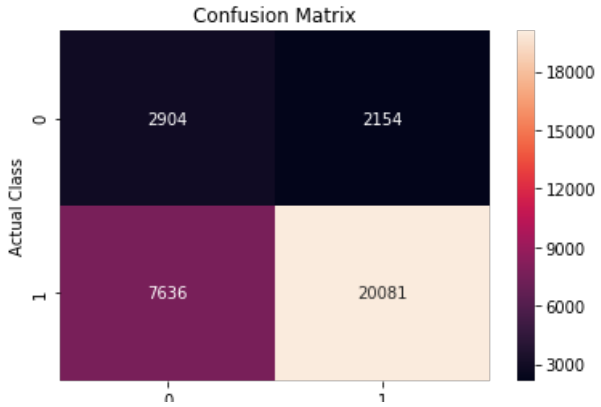
confusion matrix

In [58]:

```
y3_predict = grid.predict(set3_t)
cm3 = confusion_matrix(y_test,y3_predict)
# https://seaborn.pydata.org/generated/seaborn.heatmap.html
sns.heatmap(cm3, annot=True, fmt="d")
plt.ylabel("Actual Class")
plt.xlabel("Predicted Class")
plt.title("Confusion Matrix")
```

Out[58]:

```
Text(0.5, 1.0, 'Confusion Matrix')
```



Predicted Class

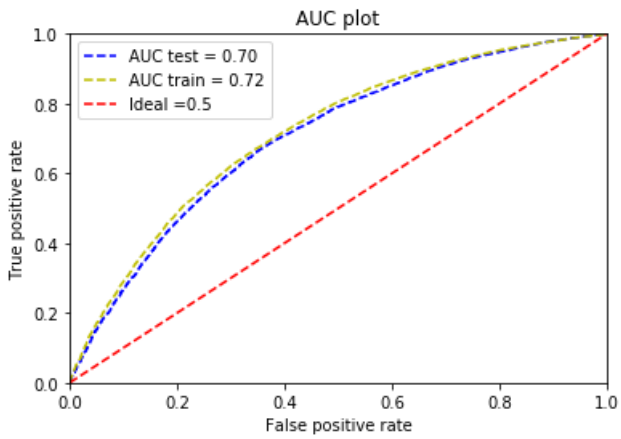
AUC CUrve

In [59]:

```
# probabilities calcultion
y3_predict_prob = ccv.predict_proba(set3_t)[:,1]
y3_predict_prob_train = ccv.predict_proba(set3)[:,1]

# took referance from https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
#fpr,tpr
fpr,tpr,thre = roc_curve(y_test,y3_predict_prob)

# am i doing it right here......?
fpr_train,tpr_train,thre_train = roc_curve(y_train,y3_predict_prob_train)
```

In [60]:

```
# auc calculation for test data
roc_auc3 = metrics.auc(fpr,tpr)

# auc calculation for train data
roc_auc_train3 = metrics.auc(fpr_train,tpr_train)

# took referance from https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
plt.plot(fpr,tpr,"b--",label = 'AUC test = %0.2f'%roc_auc3)
plt.plot(fpr_train,tpr_train,"y--",label = 'AUC train = %0.2f'%roc_auc_train3)
plt.title("AUC plot")
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
plt.plot([0, 1], [0, 1],'r--',label = "Ideal =0.5")
plt.xlim([0,1])
plt.ylim([0,1])
plt.legend(loc = "upper left")
plt.show()
```



Set4 (TFIDF Avg_w2v)

In [61]:

```
grid = GridSearchCV(sgd,par_grid,scoring="roc_auc",n_jobs=-1,cv=10)
grid.fit(set4,y_train)
print(grid.best_estimator_)
print(grid.best_index_)
print(grid.best_params_)
print(grid.best_score_)
```

```
SGDClassifier(alpha=0.001, average=False, class_weight='balanced',
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
```

```
       validation_fraction=0.1, verbose=0, warm_start=False)
5
{'alpha': 0.001, 'penalty': 'l2'}
0.6968222367442298
```

In [62]:

```
sgd =
SGDClassifier(alpha=grid.best_params_["alpha"],class_weight="balanced",penalty=grid.best_params_["
penalty"])
ccv = CalibratedClassifierCV(sgd,cv=10)
ccv.fit(set4,y_train)
print(ccv.get_params)
```

```
<bound method BaseEstimator.get_params of
CalibratedClassifierCV(base_estimator=SGDClassifier(alpha=0.001, average=False,
class_weight='balanced',
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False),
            cv=10, method='sigmoid')>
```

In [63]:

```
#converting results to dataframe
df = pd.DataFrame(data = grid.cv_results_)

# getting into list
l1_train_score = []
l1_test_score = []
l2_train_score = []
l2_test_score = []

for i in range(len(df)):
  if df.iloc[i]["param_penalty"] =="l1":
    l1_test_score.append(df.iloc[i]["mean_test_score"])
    l1_train_score.append(df.iloc[i]["mean_train_score"])

  if df.iloc[i]["param_penalty"] =="l2":
    l2_test_score.append(df.iloc[i]["mean_test_score"])
    l2_train_score.append(df.iloc[i]["mean_train_score"])

print(l1_train_score)
print(l1_test_score )
print(l2_train_score)
print(l2_test_score)
```

```
[0.6696517912765817, 0.6940314360150116, 0.6995576237021873, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]
[0.6405377724273922, 0.6772690762178323, 0.6924008180309225, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]
[0.6552380574732946, 0.6750022815200984, 0.7142898985112826, 0.6585109002490036,
0.6484185323882781, 0.6479936252088834, 0.6479923810460161, 0.6479927170289383, 0.647992301850221]
[0.6389208678761469, 0.6611515236641259, 0.6968222367442298, 0.6544395808014649,
0.6459914617241826, 0.6456755453684208, 0.6456751028230016, 0.6456759063258709,
0.6456746065732635]
```

In [64]:

```
plt.figure()
plt.subplot(121)
plt.plot(alpha,l1_train_score)
plt.plot(alpha,l1_test_score)
plt.xscale("log")
plt.xlabel("Alpha")
plt.ylabel("Accuracy Score")
plt.title("L1 mean train/test score plot")
plt.subplot(122)
plt.plot(alpha,l2_train_score)
plt.plot(alpha,l2_test_score)
plt.xscale("log")
plt.xlabel("Alpha")
```

```
plt.ylabel("Accuracy Score")
plt.title("L2 mean train/test score plot")
plt.show()
```

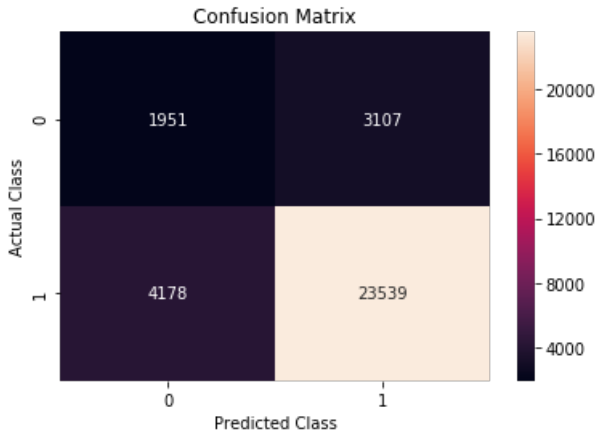
Confusion Matrix

In [65]:

```
y4_predict = grid.predict(set4_t)
cm1 = confusion_matrix(y_test,y4_predict)
# https://seaborn.pydata.org/generated/seaborn.heatmap.html
sns.heatmap(cm1, annot=True, fmt="d")
plt.ylabel("Actual Class")
plt.xlabel("Predicted Class")
plt.title("Confusion Matrix")
```

Out[65]:

```
Text(0.5, 1.0, 'Confusion Matrix')
```



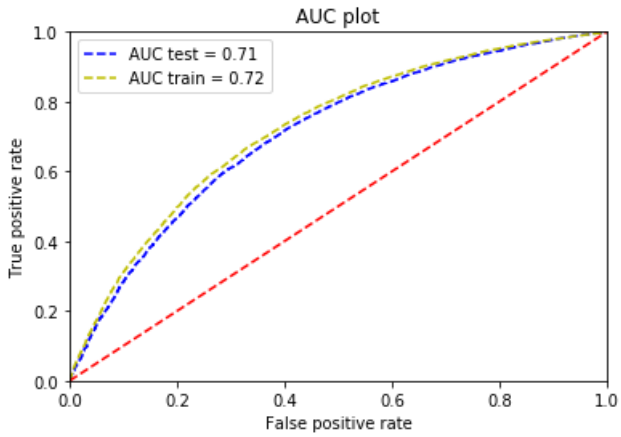
AUC Curve

In [66]:

```
# probabilities calcultion
y4_predict_prob = ccv.predict_proba(set4_t)[:,1]
y4_predict_prob_train = ccv.predict_proba(set4)[:,1]

# took referance from https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
#fpr,tpr
fpr,tpr,thre = roc_curve(y_test,y4_predict_prob)

# am i doing it right here......?
fpr_train,tpr_train,thre_train = roc_curve(y_train,y4_predict_prob_train)
```

In [67]:

```
# auc calculation for test data
roc_auc4 = metrics.auc(fpr,tpr)

# auc calculation for train data
roc_auc_train4 = metrics.auc(fpr_train,tpr_train)

# took referance from https://stackoverflow.com/questions/25009284/how-to-plot-roc-curve-in-python
plt.plot(fpr,tpr,"b--",label = 'AUC test = %0.2f'%roc_auc4)
plt.plot(fpr_train,tpr_train,"y--",label = 'AUC train = %0.2f'%roc_auc_train4)
plt.title("AUC plot")
plt.xlabel("False positive rate")
plt.ylabel("True positive rate")
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0,1])
plt.ylim([0,1])
plt.legend(loc = "upper left")
plt.show()
```



Task-2

Please refer attachments

DonorsChoose_7 SVM(Set5)

Observation

1. Best model found is Avg-W2v with Test AUC of 0.705 and Train AUC of 0.716
2. Maximum Train AUC found is 0.795 in case of BOW
3. 40k Training data point considered in Task2 due to Memory Error

Summary

In [77]:

```
from prettytable import PrettyTable
summary = PrettyTable()
```

In [78]:

```
summary.field_names = ["Set", "Vectorizer", "Model", "Hyperparameter","Test","Train"]
```

In [79]:

```
summary.add_row(["set1","BOW","SVM(Hinge Loss)","'alpha': 0.001, 'penalty': 'l2'","%0.3f"%roc_auc1
,"%0.3f"%roc_auc_train1])
summary.add_row(["set2","TFIDF","SVM(Hinge Loss)","'alpha': 0.001, 'penalty':
'l2'","%0.3f"%roc_auc2,"%0.3f"%roc_auc_train2])
summary.add_row(["set1","Avg-W2v","SVM(Hinge Loss)","'alpha': 0.001, 'penalty': 'l2'","%0.3f"%roc_
auc3,"%0.3f"%roc_auc_train3])
summary.add_row(["set2","TFIDF W2V","SVM(Hinge Loss)","'alpha': 0.001, 'penalty': 'l2'","%0.3f"%ro
```

```
c_auc4,"%0.3f"%roc_auc_train4])
summary.add_row(["set1","TFIDF (Truncated SVD)","SVM(Hinge Loss)","'alpha': 0.0001, 'penalty': 'l2
'","0.64","0.63"])
```

In [80]:

```
print(summary)
```

```
+------+-----------------------+-----------------+-----------------------------------+-------+------
| Set  |       Vectorizer      |      Model      |           Hyperparameter          |  Test | Train
|
+------+-----------------------+-----------------+-----------------------------------+-------+------
| set1 |          BOW          | SVM(Hinge Loss) | 'alpha': 0.001, 'penalty': 'l2'   | 0.686 | 0.795
|
| set2 |         TFIDF         | SVM(Hinge Loss) | 'alpha': 0.001, 'penalty': 'l2'   | 0.686 | 0.731
|
| set1 |        Avg-W2v        | SVM(Hinge Loss) | 'alpha': 0.001, 'penalty': 'l2'   | 0.705 | 0.716
|
| set2 |       TFIDF W2V       | SVM(Hinge Loss) | 'alpha': 0.001, 'penalty': 'l2'   | 0.709 | 0.724
|
| set1 | TFIDF (Truncated SVD) | SVM(Hinge Loss) | 'alpha': 0.0001, 'penalty': 'l2'  |  0.64 |
0.63 |
+------+-----------------------+-----------------+-----------------------------------+-------+------
```

In [ ]: