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
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
from sklearn.model_selection import train_test_split
from scipy.sparse import hstack
```

Data Preprocessing

In [2]:

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

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

Out[2]:

```
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 [3]:

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

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

# 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)


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')

print(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
None
```

In [4]:

```
#splitting data as 20% 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)

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

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

In [5]:

```
# considering lesser datapoints due to memory error
X_train = X_train[:40000]
y_train = y_train[:40000]
X_test = X_test[:5000]
y_test = y_test[:5000]
print(X_train.shape," ",y_train.shape)
print(X_test.shape," ",y_test.shape)
```

```
(40000, 19)   (40000,)
(5000, 19)   (5000,)
```

Text data TFIDF

In [6]:

```
#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', "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 [7]:

```
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%|████████████████████████████████████████████████████████████████████████████| 40000/40000
[00:23<00:00, 1670.59it/s]
100%|██████████████████████████████████████████████████████████████████████████████| 5000/5000
[00:02<00:00, 1692.17it/s]
```

In [8]:

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

# 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)
```

```
Shape of train matrix :  (40000, 5000)
Shape of test matrix :  (5000, 5000)
```

Implementation of elbow method

using less iterations due to memory error and insufficient RAM.

Tried multiple times but crashed session every time

PLease check attachment for elbow method

seperately executed elbow method to get best n_components ipynb file present in attachment please check here https://imgur.com/wVbiDfM

In [9]:

```
# from graph -->usng 2700 n_components which preserve around 80% variance
# please check here https://imgur.com/wVbiDfM
from sklearn.decomposition import TruncatedSVD
svd = TruncatedSVD(n_components=2700, n_iter=7, random_state=42)
svd.fit(text_tfidf)
truncated_essay = svd.transform(text_tfidf)
truncated_essay_test = svd.transform(test_text_tfidf)
```

In [10]:

```
from scipy.sparse import csr_matrix
truncated_essay = csr_matrix(truncated_essay)
truncated_essay_test = csr_matrix(truncated_essay_test)
print(type(truncated_essay))
```

```
<class 'scipy.sparse.csr.csr_matrix'>
```

Preprocessing categorical Features

1. Preprocessing Categories

In [11]:

```python
#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)
```

1. Preprocessing Subcategories

In [12]:

```python
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():
            j=j.replace('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():
            j=j.replace('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. Preprocessing 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': 6, 'Teacher': 859, 'Mr': 3920, 'Ms': 14407, 'Mrs': 20808}
```

1. Preoprocessing Grade

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:
```

```
    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': 4045, 'six_eight': 6143, 'three_five': 13650, 'prek_two': 16162}
```

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"]
```

Vectorizing of Categorical data

1. Vectorizing Categories

In [16]:

```
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())

# 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']
```

```
'Health_Sports', 'Math_Science', 'Literacy_Language']
Shape of matrix after one hot encodig  (40000, 9)
```

In [17]:

```
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())

# 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',
'ForeignLanguages', 'Civics_Government', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'College_CareerPrep', 'Oth
er', 'Music', 'History_Geography', 'EarlyDevelopment', 'Health_LifeScience', 'ESL', 'Gym_Fitness',
'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences', 'SpecialNeeds',
'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix after one hot encodig  (40000, 30)
```

In [18]:

```
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())

# 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  (40000, 5)
```

In [19]:

```
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())

# 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)
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  (40000, 4)
['VT', 'WY', 'ND', 'MT', 'NE', 'RI', 'AK', 'SD', 'DE', 'NH', 'WV', 'HI', 'ME', 'NM', 'DC', 'KS', 'I
D', 'IA', 'AR', 'CO', 'KY', 'MN', 'OR', 'MS', 'NV', 'MD', 'CT', 'TN', 'UT', 'AL', 'WI', 'VA', 'AZ',
'OK', 'NJ', 'WA', 'LA', 'OH', 'MA', 'MO', 'IN', 'PA', 'MI', 'SC', 'GA', 'IL', 'NC', 'FL', 'TX', 'NY
', 'CA']
```

Preprocessing Numerical Feature

In [20]:

```python
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
```

In [21]:

```python
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 : 16.9483, Standard deviation of Quantity : 25.95974628362149
```

```
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.
```

In [22]:

```python
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 : 11.2195, Standard deviation : 27.84555565525673
```

```
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.
```

SET-5 (TruncatedSVD and Sentiment Analysis)

Data Preperation

In [23]:

```
from textblob import TextBlob

#this is beacuse was getting error. so added it
import nltk
nltk.download('punkt')
nltk.download('vader_lexicon')

from nltk.sentiment.vader import SentimentIntensityAnalyzer
sid = SentimentIntensityAnalyzer()
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\rdbz3b\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\rdbz3b\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

In [24]:

```
# preoprocessing of essay
# took referance form https://monkeylearn.com/sentiment-analysis/
# too referance https://www.kaggle.com/ankkur13/sentiment-analysis-nlp-wordcloud-textblob

essay1 = []
essay2 = []
essay3 = []
essay4 = []

#preprocessing each essay for sentiment analysis. Remooved stop word command

for i in range(1,5):
# tqdm is for printing the status bar
  temp_essay = []
  temp = X_train["project_essay_{}".format(i)].astype(str)
  for sentance in tqdm(temp.values):
      sent = decontracted(sentance)
      sent = sent.replace('\\r', ' ')
      sent = sent.replace('\\"', ' ')
      sent = sent.replace('\\n', ' ')
      sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
      temp_essay.append(sent.lower().strip())

  X_train["clean_essay_{}".format(i)] = temp_essay

# blob.sentimnt.polarity gives polarity of review i.e review is +ve or -ve
# please let me know if if my approach is right

# calculating sentiment analysis for each of essay's
#essay1_descr=project_data['clean_essay_1']



for i in X_train['clean_essay_1']:
  ss = sid.polarity_scores(i)
  essay1.append(ss)

for i in X_train['clean_essay_2']:
  ss = sid.polarity_scores(i)
  essay2.append(ss)

for i in X_train['clean_essay_3']:
  ss = sid.polarity_scores(i)
  essay3.append(ss)


for i in X_train['clean_essay_4']:
  ss = sid.polarity_scores(i)
  essay4.append(ss)



print(len(essay1))
print(len(essay2))
print(len(essay3))
print(len(essay4))
```

```
100%|████████████████████████████████████████████████████████████████████████████| 40000/40000
[00:02<00:00, 15788.10it/s]
100%|████████████████████████████████████████████████████████████████████████████| 40000/40000
[00:03<00:00, 12907.50it/s]
100%|████████████████████████████████████████████████████████████████████████████| 40000/40000
[00:00<00:00, 74969.41it/s]
100%|████████████████████████████████████████████████████████████████████████████| 40000/40000
[00:00<00:00, 79996.26it/s]
```

```
40000
40000
40000
40000
```

In [25]:

```
# converting sentiment analysis of each essay to dataframe.
# adding corresponding columns to reduce dimentionality and computational power
df1 = pd.DataFrame(data=essay1)
df2 = pd.DataFrame(data=essay2)
df3 = pd.DataFrame(data=essay3)
df4 = pd.DataFrame(data=essay4)

df = df1+df2+df3+df4
```

In [26]:

```
# preoprocessing of essay
# took referance form https://monkeylearn.com/sentiment-analysis/
# too referance https://www.kaggle.com/ankkur13/sentiment-analysis-nlp-wordcloud-textblob

essay1_test = []
essay2_test = []
essay3_test = []
essay4_test = []

# preprocessing each essay for sentiment analysis. Remooved stop word command

for i in range(1, 5):
    # tqdm is for printing the status bar
    temp_essay = []
    temp = X_test["project_essay_{}".format(i)].astype(str)
    for sentance in tqdm(temp.values):
        sent = decontracted(sentance)
        sent = sent.replace('\\r', ' ')
        sent = sent.replace('\\"', ' ')
        sent = sent.replace('\\n', ' ')
        sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
        temp_essay.append(sent.lower().strip())

    X_test["clean_essay_{}".format(i)] = temp_essay

# blob.sentimnt.polarity gives polarity of review i.e review is +ve or -ve
# please let me know if if my approach is right

# calculating sentiment analysis for each of essay's
# essay1_test_descr=project_data['clean_essay_1']


for i in X_test['clean_essay_1']:
    ss = sid.polarity_scores(i)
    essay1_test.append(ss)

for i in X_test['clean_essay_2']:
    ss = sid.polarity_scores(i)
    essay2_test.append(ss)

for i in X_test['clean_essay_3']:
    ss = sid.polarity_scores(i)
    essay3_test.append(ss)

for i in X_test['clean_essay_4']:
    ss = sid.polarity_scores(i)
    essay4_test.append(ss)

print(len(essay1_test))
print(len(essay2_test))
print(len(essay3_test))
print(len(essay4_test))
```

```
100%|████████████████████████████████████████████████████████████████████████████| 5000/5000
[00:00<00:00, 15861.18it/s]
100%|████████████████████████████████████████████████████████████████████████████| 5000/5000
[00:00<00:00, 13287.97it/s]
100%|████████████████████████████████████████████████████████████████████████████| 5000/5000
[00:00<00:00, 75701.53it/s]
100%|████████████████████████████████████████████████████████████████████████████| 5000/5000
[00:00<00:00, 78067.26it/s]
```

```
5000
5000
5000
5000
```

In [27]:

```
# converting sentiment analysis of each essay to dataframe.
# adding corresponding columns to reduce dimentionality and computational power
df1_test = pd.DataFrame(data=essay1_test)
df2_test = pd.DataFrame(data=essay2_test)
df3_test = pd.DataFrame(data=essay3_test)
df4_test = pd.DataFrame(data=essay4_test)

df_test = df1_test+df2_test+df3_test+df4_test
```

In [28]:

```
from sklearn.preprocessing import StandardScaler


scalar = StandardScaler()
# fitting train data
scalar.fit(df['compound'].values.reshape(-1,1))

# transforming train and test data
compound = scalar.transform(df['compound'].values.reshape(-1,1))
compound_test = scalar.transform(df_test['compound'].values.reshape(-1,1))
```

```
scalar = StandardScaler()
# fitting train data
scalar.fit(df['pos'].values.reshape(-1,1))

# transforming train and test data
pos = scalar.transform(df['pos'].values.reshape(-1,1))
pos_test = scalar.transform(df_test['pos'].values.reshape(-1,1))

scalar = StandardScaler()
# fitting train data
scalar.fit(df['neg'].values.reshape(-1,1))

# transforming train and test data
neg = scalar.transform(df['neg'].values.reshape(-1,1))
neg_test = scalar.transform(df_test['neg'].values.reshape(-1,1))


scalar = StandardScaler()
# fitting train data
scalar.fit(df['neu'].values.reshape(-1,1))

# transforming train and test data
neu = scalar.transform(df['neu'].values.reshape(-1,1))
neu_test = scalar.transform(df_test['neu'].values.reshape(-1,1))
```

In [29]:

```
# as lenght of preprocessed array and text have same lenghts
# to store sum of counts of words for title and essay

X_train["combine_essay"] = X_train["clean_essay_1"]+
X_train["clean_essay_2"]+X_train["clean_essay_3"]+X_train["clean_essay_4"]
X_test["combine_essay"] = X_test["clean_essay_1"]+ X_test["clean_essay_2"]+X_test["clean_essay_3"]+
X_test["clean_essay_4"]

# For train data

title_sum = []
essay_sum = []

for i in range(len(X_train["combine_essay"])):
  blob = TextBlob(X_train.iloc[i]["combine_essay"])
  a = blob.word_counts
  title_sum.append(sum(a.values()))
  blob = TextBlob(X_train.iloc[i]["project_title"])
  a = blob.word_counts
  essay_sum.append(sum(a.values()))


# for test data

title_sum_test = []
essay_sum_test = []

for i in range(len(X_test["combine_essay"])):
  blob = TextBlob(X_test.iloc[i]["combine_essay"])
  a = blob.word_counts
  title_sum_test.append(sum(a.values()))
  blob = TextBlob(X_test.iloc[i]["project_title"])
  a = blob.word_counts
  essay_sum_test.append(sum(a.values()))
```

In [30]:

```
t_sum = np.array(title_sum).reshape(-1,1)
e_sum = np.array(essay_sum).reshape(-1,1)
```

In [31]:

```
t_sum_test = np.array(title_sum_test).reshape(-1,1)
e_sum_test = np.array(essay_sum_test).reshape(-1,1)
```

In [33]:

```
from scipy.sparse import hstack
set5 =
hstack((categories_one_hot,sub_categories_one_hot,prefix_one_hot,grade_one_hot,state_one_hot,price_
standardized,quantity_standardized,number_ppp_standardized,compound,pos,neg,neu,t_sum,e_sum,truncat
ed_essay))
set5_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
,compound_test,pos_test,neg_test,neu_test,t_sum_test,e_sum_test,truncated_essay_test))
print(set5.shape)
print(set5_t.shape)
```

```
(40000, 2808)
(5000, 2808)
```

In [36]:

```
from sklearn.calibration import CalibratedClassifierCV
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import GridSearchCV
param_grid = dict(penalty=["l1","l2"],alpha = [0.001,0.01,0.1,1,10])
sgd = SGDClassifier(n_jobs=-1)
grid = GridSearchCV(sgd,param_grid,scoring='roc_auc',n_jobs=-1,cv=10)
```

SET5

In [37]:

```
grid.fit(set5,y_train)
```

Out[37]:

```
GridSearchCV(cv=10, error_score='raise-deprecating',
       estimator=SGDClassifier(alpha=0.0001, average=False, class_weight=None,
       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=-1, penalty='l2',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=-1,
       param_grid={'penalty': ['l1', 'l2'], 'alpha': [0.001, 0.01, 0.1, 1, 10]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

In [40]:

```
sgd =
SGDClassifier(alpha=grid.best_params_["alpha"],class_weight="balanced",penalty=grid.best_params_["
penalty"])
ccv = CalibratedClassifierCV(sgd,cv=10)
ccv.fit(set5,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='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 [41]:

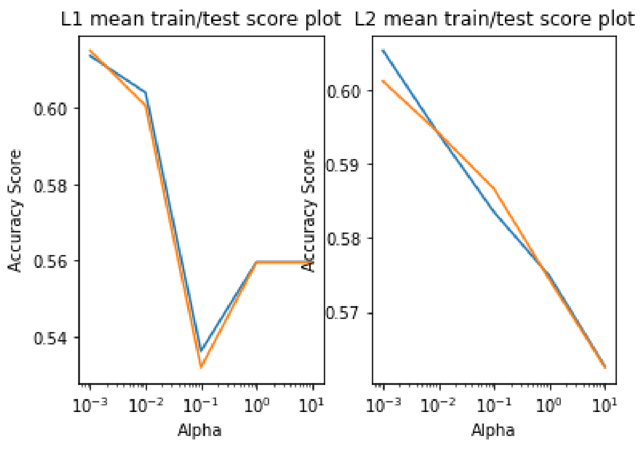
```
#converting results to dataframe
df = pd.DataFrame(data = grid.cv_results_)
```

```
# 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)
```

```
[0.6137085799204108, 0.6040695789214917, 0.5361888889530985, 0.5596099904502754,
0.5596099904502754]
[0.6150966887858336, 0.6006332386135353, 0.5319484131191882, 0.559501445843147, 0.559501445843147]
[0.6053780619822637, 0.5941209214774653, 0.583537652685651, 0.5748864177016759, 0.562626802467141]
[0.601228666737878, 0.5942433880970887, 0.586682805871488, 0.5743670767882715, 0.5625083340415288]
```

In [43]:

```
alpha = [0.001,0.01,0.1,1,10]
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()
```

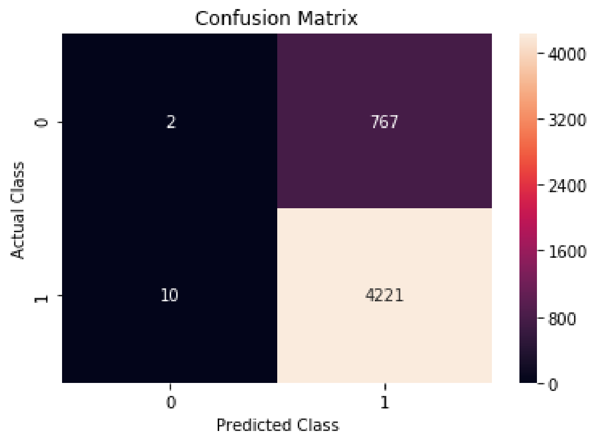


In [47]:

```
y5_predict = ccv.predict(set5_t)
cm5 = confusion_matrix(y_test,y5_predict)
# https://seaborn.pydata.org/generated/seaborn.heatmap.html
sns.heatmap(cm5, annot=True, fmt="d")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Class")
plt.ylabel("Actual Class")
```

Out[47]:

```
Text(33.0, 0.5, 'Actual Class')
```

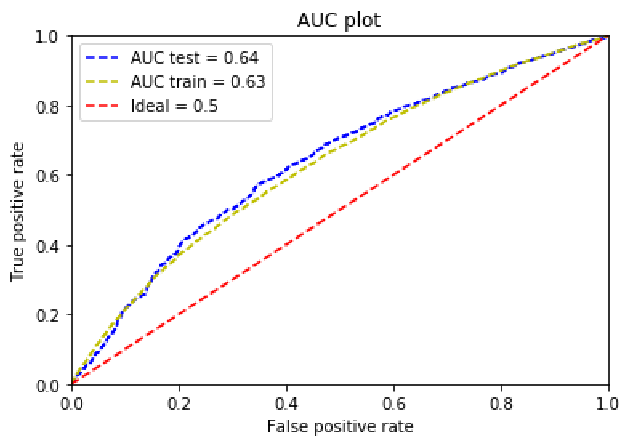


In [45]:

```
# probabilities calcultion
y5_predict_prob = ccv.predict_proba(set5_t)[:,1]
y5_predict_prob_train = ccv.predict_proba(set5)[:,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,y5_predict_prob)

# am i doing it right here......?
fpr_train,tpr_train,thre_train = roc_curve(y_train,y5_predict_prob_train)
```

In [46]:

```
# auc calculation for test data
roc_auc5 = metrics.auc(fpr,tpr)

# auc calculation for train data
roc_auc_train5 = 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_auc5)
plt.plot(fpr_train,tpr_train,"y--",label = 'AUC train = %0.2f'%roc_auc_train5)
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()
```



In [ ]:

```
```