# DonorsChoose

DonorsChoose.org receives hundreds of thousands of project proposals each year for classroom projects in need of funding. Right now, a large number of volunteers is needed to manually screen each submission before it's approved to be posted on the DonorsChoose.org website.

Next year, DonorsChoose.org expects to receive close to 500,000 project proposals. As a result, there are three main problems they need to solve:

- How to scale current manual processes and resources to screen 500,000 projects so that they can be posted as quickly and as efficiently as possible
- How to increase the consistency of project vetting across different volunteers to improve the experience for teachers
- How to focus volunteer time on the applications that need the most assistance

The goal of the competition is to predict whether or not a DonorsChoose.org project proposal submitted by a teacher will be approved, using the text of project descriptions as well as additional metadata about the project, teacher, and school. DonorsChoose.org can then use this information to identify projects most likely to need further review before approval.

## About the DonorsChoose Data Set

The `train.csv` data set provided by DonorsChoose contains the following features:

| Feature | Description |
|---:|---:|
| `project_id` | A unique identifier for the proposed project. **Example:** `p036502` |
| `project_title` | Title of the project. **Examples:**<br>• `Art Will Make You Happy!`<br>• `First Grade Fun` |
| `project_grade_category` | Grade level of students for which the project is targeted. One of the following enumerated values:<br>• `Grades PreK-2`<br>• `Grades 3-5`<br>• `Grades 6-8`<br>• `Grades 9-12` |
| `project_subject_categories` | One or more (comma-separated) subject categories for the project from the following enumerated list of values:<br>• `Applied Learning`<br>• `Care & Hunger`<br>• `Health & Sports`<br>• `History & Civics`<br>• `Literacy & Language`<br>• `Math & Science`<br>• `Music & The Arts`<br>• `Special Needs`<br>• `Warmth`<br>**Examples:**<br>• `Music & The Arts`<br>• `Literacy & Language, Math & Science` |
| `school_state` | State where school is located (Two-letter U.S. postal code). **Example:** `WY` |
| `project_subject_subcategories` | One or more (comma-separated) subject subcategories for the project. **Examples:**<br>• `Literacy`<br>• `Literature & Writing, Social Sciences` |
| `project_resource_summary` | An explanation of the resources needed for the project. **Example:**<br>• `My students need hands on literacy materials to manage sensory needs!` |
| `project_essay_1` | First application essay* |
| `project_essay_2` | Second application essay* |
| `project_essay_3` | Third application essay* |

| Feature | Description |
|---|---|
| `project_essay_4` | Fourth application essay |
| `project_submitted_datetime` | Datetime when project application was submitted. **Example:** `2016-04-28 12:43:56.245` |
| `teacher_id` | A unique identifier for the teacher of the proposed project. **Example:** `bdf8baa8fedef6bfeec7ae4ff1c15c56` |
| `teacher_prefix` | Teacher's title. One of the following enumerated values:<br>• `nan`<br>• `Dr.`<br>• `Mr.`<br>• `Mrs.`<br>• `Ms.`<br>• `Teacher.` |
| `teacher_number_of_previously_posted_projects` | Number of project applications previously submitted by the same teacher. **Example:** `2` |

[*] See the section **Notes on the Essay Data** for more details about these features.

Additionally, the `resources.csv` data set provides more data about the resources required for each project. Each line in this file represents a resource required by a project:

| Feature | Description |
|---|---|
| `id` | A `project_id` value from the `train.csv` file. **Example:** `p036502` |
| `description` | Desciption of the resource. **Example:** `Tenor Saxophone Reeds, Box of 25` |
| `quantity` | Quantity of the resource required. **Example:** `3` |
| `price` | Price of the resource required. **Example:** `9.95` |

**Note:** Many projects require multiple resources. The `id` value corresponds to a `project_id` in train.csv, so you use it as a key to retrieve all resources needed for a project:

The data set contains the following label (the value you will attempt to predict):

| Label | Description |
|---|---|
| `project_is_approved` | A binary flag indicating whether DonorsChoose approved the project. A value of `0` indicates the project was not approved, and a value of `1` indicates the project was approved. |

## Notes on the Essay Data

Prior to May 17, 2016, the prompts for the essays were as follows:

- __project_essay_1:__ "Introduce us to your classroom"
- __project_essay_2:__ "Tell us more about your students"
- __project_essay_3:__ "Describe how your students will use the materials you're requesting"
- __project_essay_3:__ "Close by sharing why your project will make a difference"

Starting on May 17, 2016, the number of essays was reduced from 4 to 2, and the prompts for the first 2 essays were changed to the following:

- __project_essay_1:__ "Describe your students: What makes your students special? Specific details about their background, your neighborhood, and your school are all helpful."
- __project_essay_2:__ "About your project: How will these materials make a difference in your students' learning and improve their school lives?"

For all projects with project_submitted_datetime of 2016-05-17 and later, the values of project_essay_3 and project_essay_4 will be NaN.

In [77]:

```
%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
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

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

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

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

from tqdm import tqdm
import os

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

## 1.1 Reading Data

In [0]:

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

In [0]:

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

In [0]:

```
print("Number of data points in train data", resource_data.shape)
print(resource_data.columns.values)
resource_data.head(2)
```

## 1.2 preprocessing of `project_subject_categories`

In [0]:

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

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
cat_list = []
for i in catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & H
unger"]
        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')
```

```
.e removing 'The')
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math &
Science"=>"Math&Science"
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())

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

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

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

## 1.3 preprocessing of `project_subject_subcategories`

In [0]:

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

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python

sub_cat_list = []
for i in sub_catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & H
unger"]
        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(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math &
Science"=>"Math&Science"
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

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

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

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

## 1.3 Text preprocessing

In [0]:

```
# 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 [0]:

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

In [0]:

```
#### 1.4.2.3 Using Pretrained Models: TFIDF weighted W2V
```

In [0]:

```
# printing some random reviews
print(project_data['essay'].values[0])
print("="*50)
print(project_data['essay'].values[150])
print("="*50)
print(project_data['essay'].values[1000])
print("="*50)
print(project_data['essay'].values[20000])
print("="*50)
print(project_data['essay'].values[99999])
print("="*50)
```

In [0]:

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

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [0]:

```
sent = decontracted(project_data['essay'].values[20000])
print(sent)
print("="*50)
```

In [0]:

```
# \r \n \t remove from string python: http://texthandler.com/info/remove-line-breaks-python/
sent = sent.replace('\\r', ' ')
sent = sent.replace('\\"', ' ')
sent = sent.replace('\\n', ' ')
print(sent)
```

In [0]:

```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
print(sent)
```

In [0]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",
\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "do
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [0]:

```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(project_data['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())
```

In [0]:

```
# after preprocesing
preprocessed_essays[20000]
```

## 1.4 Preprocessing of `project_title`

In [0]:

```
# similarly you can preprocess the titles also
```

## 1.5 Preparing data for models

In [0]:

```
project_data.columns
```

we are going to consider

```
       - school_state : categorical data
       - clean_categories : categorical data
       - clean_subcategories : categorical data
       - project_grade_category : categorical data
       - teacher_prefix : categorical data

       - project_title : text data
       - text : text data
       - project_resource_summary: text data (optinal)
```

```
      - quantity : numerical (optinal)
      - teacher_number_of_previously_posted_projects : numerical
      - price : numerical
```

### 1.5.1 Vectorizing Categorical data

- https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/

In [0]:

```
# we use count vectorizer to convert the values into one
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(vocabulary=list(sorted_cat_dict.keys()), lowercase=False, binary=True
)
categories_one_hot = vectorizer.fit_transform(project_data['clean_categories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",categories_one_hot.shape)
```

In [0]:

```
# we use count vectorizer to convert the values into one
vectorizer = CountVectorizer(vocabulary=list(sorted_sub_cat_dict.keys()), lowercase=False, binary=
True)
sub_categories_one_hot = vectorizer.fit_transform(project_data['clean_subcategories'].values)
print(vectorizer.get_feature_names())
print("Shape of matrix after one hot encodig ",sub_categories_one_hot.shape)
```

In [0]:

```
# you can do the similar thing with state, teacher_prefix and project_grade_category also
```

### 1.5.2 Vectorizing Text data

#### 1.5.2.1 Bag of words

In [0]:

```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer = CountVectorizer(min_df=10)
text_bow = vectorizer.fit_transform(preprocessed_essays)
print("Shape of matrix after one hot encodig ",text_bow.shape)
```

In [0]:

```
# you can vectorize the title also
# before you vectorize the title make sure you preprocess it
```

#### 1.5.2.2 TFIDF vectorizer

In [0]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10)
text_tfidf = vectorizer.fit_transform(preprocessed_essays)
print("Shape of matrix after one hot encodig ",text_tfidf.shape)
```

#### 1.5.2.3 Using Pretrained Models: Avg W2V

In [0]:

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

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

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

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

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

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

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

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


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

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


'''
```

In [0]:

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

In [0]:

```
# average Word2Vec
# compute average word2vec for each review.
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
            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]))
```

#### 1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [0]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
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())
```

In [0]:

```
# average Word2Vec
# compute average word2vec for each review.
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]))
```

In [0]:

```
# Similarly you can vectorize for title also
```

### 1.5.3 Vectorizing Numerical features

In [0]:

```
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 [0]:

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

# price_standardized = standardScalar.fit(project_data['price'].values)
# this will rise the error
# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329.   ... 399.   287.
73   5.5 ].
# Reshape your data either using array.reshape(-1, 1)

price_scalar = StandardScaler()
price_scalar.fit(project_data['price'].values.reshape(-1,1)) # finding the mean and standard
deviation of this data
```

```
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")

# Now standardize the data with above maen and variance.
price_standardized = price_scalar.transform(project_data['price'].values.reshape(-1, 1))
```

In [0]:

```
price_standardized
```

### 1.5.4 Merging all the above features

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

In [0]:

```
print(categories_one_hot.shape)
print(sub_categories_one_hot.shape)
print(text_bow.shape)
print(price_standardized.shape)
```

In [0]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X = hstack((categories_one_hot, sub_categories_one_hot, text_bow, price_standardized))
X.shape
```

**Computing Sentiment Scores**

In [0]:

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

# import nltk
# nltk.download('vader_lexicon')

sid = SentimentIntensityAnalyzer()

for_sentiment = 'a person is a person no matter how small dr seuss i teach the smallest students with the biggest enthusiasm \
for learning my students learn in many different ways using all of our senses and multiple intelligences i use a wide range\
of techniques to help all my students succeed students in my class come from a variety of different backgrounds which makes\
for wonderful sharing of experiences and cultures including native americans our school is a caring community of successful \
learners which can be seen through collaborative student project based learning in and out of the classroom kindergarteners \
in my class love to work with hands on materials and have many different opportunities to practice a skill before it is\
mastered having the social skills to work cooperatively with friends is a crucial aspect of the kindergarten curriculum\
montana is the perfect place to learn about agriculture and nutrition my students love to role play in our pretend kitchen\
in the early childhood classroom i have had several kids ask me can we try cooking with real food i will take their idea \
and create common core cooking lessons where we learn important math and writing concepts while cooking delicious healthy \
food for snack time my students will have a grounded appreciation for the work that went into making the food and knowledge \
of where the ingredients came from as well as how it is healthy for their bodies this project would expand our learning of \
nutrition and agricultural cooking recipes by having us peel our own apples to make homemade applesauce make our own bread \
and mix up healthy plants from our classroom garden in the spring we will also create our own cookbooks to be printed and \
shared with families students will gain math and literature skills as well as a life long enjoyment for healthy cooking \
e for healthy cooking (
nannan'
ss = sid.polarity_scores(for_sentiment)

for k in ss:
    print('{0}: {1}, '.format(k, ss[k]), end='')

# we can use these 4 things as features/attributes (neg, neu, pos, compound)
# neg: 0.0, neu: 0.753, pos: 0.247, compound: 0.93
```

# Assignment 5: Logistic Regression

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

2. **Hyper paramter tuning (find best hyper parameters corresponding the algorithm that you choose)**
   - Find the best hyper parameter which will give the maximum AUC value
   - Find the best hyper paramter using k-fold cross validation or simple cross validation data
   - Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning

3. **Representation of results**
   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure.
   - Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
   - Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.

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

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

6. Conclusion
   - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link

**Note: Data Leakage**

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

# 2. Logistic Regression

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

In [1]:

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

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

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

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

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

from tqdm import tqdm
import os

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

In [2]:

```
from google.colab import drive
drive.mount("/content/drive")

project_data = pd.read_csv('/content/drive/My Drive/Assignments_DonorsChoose_2018/train_data.csv')
resource_data = pd.read_csv('/content/drive/My Drive/Assignments_DonorsChoose_2018/resources.csv')

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

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6
qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%
b&scope=email%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.
2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fww
ogleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code

Enter your authorization code:
··········
Mounted at /content/drive
```

Out[2]:

```
Unnamed: 0                                        0
id                                                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]:

```
from sklearn.model_selection import train_test_split

#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=0.20, random_state=42)

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

```
(87398, 19)   (87398,)
(21850, 19)   (21850,)
```

## 2.2 Make Data Model Ready: encoding numerical, categorical features

**Preprocessing categorical Features**

1. project subject categories

In [0]:

```
#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. project subject sub_categories

In [0]:

```
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 [7]:

```
#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': 11, 'Teacher': 1900, 'Mr': 8519, 'Ms': 31168, 'Mrs': 45800}
```

1. Project Grade Category

In [8]:

```
# 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': 8709, 'six_eight': 13487, 'three_five': 29679, 'prek_two': 35523}
```

1. School State

In [0]:

```
#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

### Preprocessing Numerical Feature

Standardizing price

In [10]:

```
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
```

Standardizing quantity

In [11]:

```
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))
```

```
Mean of Quantity : 16.949598388979155, Standard deviation of Quantity : 26.00482033345183
```

Standardizing number of ppp

In [12]:

```
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)
)
```

```
Mean : 11.102897091466623, Standard deviation : 27.572082372998246
```

**Vectorizing of Categorical data**

1. Vectorizing project categories and subcategories

In [13]:

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

# fitting on train data
vectorizer.fit(X_train['clean_categories'].values)
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']
Shape of matrix after one hot encodig  (87398, 9)
```

1. Vectorizing project subcategories

In [14]:

```
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',
'Civics_Government', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Other',
'College_CareerPrep', 'History_Geography', 'Music', 'Health_LifeScience', 'EarlyDevelopment', 'ESL
', 'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
Shape of matrix after one hot encodig  (87398, 30)
```

1. vectorizing teacher prefix

In [15]:

```
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)
```

```
['Mrs', 'Ms', 'Teacher', 'Mr', 'Dr']
Shape of matrix after one hot encodig  (87398, 5)
```

In [16]:

```
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', 'three_five', 'six_eight', 'nine_twelve']
Shape of matrix after one hot encodig  (87398, 4)
['VT', 'WY', 'ND', 'MT', 'RI', 'NE', 'SD', 'AK', 'DE', 'NH', 'WV', 'ME', 'DC', 'HI', 'NM', 'KS', 'I
D', 'IA', 'AR', 'CO', 'MN', 'OR', 'KY', 'MS', 'NV', 'MD', 'CT', 'TN', 'UT', 'AL', 'WI', 'VA', 'AZ',
'NJ', 'OK', 'WA', 'LA', 'MA', 'OH', 'MO', 'IN', 'PA', 'MI', 'GA', 'SC', 'IL', 'NC', 'FL', 'NY', 'TX
', 'CA']
```

## 2.3 Make Data Model Ready: encoding eassay, and project_title

**Preprocessing of Text Feature for both teat and train data**

In [0]:

```
#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"]
```

preprocessing of project essay

In [18]:

```
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%|██████████| 87398/87398 [00:53<00:00, 1630.86it/s]
100%|██████████| 21850/21850 [00:13<00:00, 1626.63it/s]
```

preprocessing of project title

In [19]:

```
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%|██████████| 87398/87398 [00:02<00:00, 33840.78it/s]
100%|██████████| 21850/21850 [00:00<00:00, 34053.30it/s]
```

### Vectorizing Text Feature

1. BOW

In [20]:

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

# 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)

# 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 :  (87398, 5000)
Shape of test matrix :  (21850, 5000)
Shape of train matrix :  (87398, 3305)
Shape of test matrix :  (21850, 3305)
```

1. TFIDF

In [21]:

```
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)


# for title
vectorizer.fit(preprocessed_title)

# 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 :  (87398, 5000)
Shape of test matrix :  (21850, 5000)
Shape of train matrix :  (87398, 3305)
Shape of test matrix :  (21850, 3305)
```

1. Avg W2v

In [0]:

```
with open('/content/drive/My Drive/Assignments_DonorsChoose_2018/glove_vectors', 'rb') as f:
```

```
with open('/content/drive/My Drive/Assignments_DonorsChoose_2018/glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [23]:

```
# 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%|██████████| 87398/87398 [00:29<00:00, 2938.49it/s]
  1%|▏         | 320/21850 [00:00<00:06, 3189.77it/s]
```

```
87398
300
```

```
100%|██████████| 21850/21850 [00:07<00:00, 3100.65it/s]
```

```
21850
300
```

In [24]:

```
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%|██████████| 87398/87398 [00:01<00:00, 58616.97it/s]
 25%|██▍       | 5439/21850 [00:00<00:00, 54387.07it/s]
```

```
87398
300
```

```
100%|██████████| 21850/21850 [00:00<00:00, 55322.97it/s]
```

```
21850
300
```

1. TFIDF avgw2v

In [25]:

```
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
            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%|██████████| 21850/21850 [00:44<00:00, 491.69it/s]
```

```
21850
300
```

```
100%|██████████| 21850/21850 [00:00<00:00, 25048.16it/s]
```

```
21850
```

In [26]:

```
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%|██████████| 87398/87398 [03:01<00:00, 480.77it/s]
```

```
100%|██████████| 87398/87398 [03:01<00:00, 480.71it/s]
```

```
87398
300
```

```
100%|██████████| 87398/87398 [00:03<00:00, 22942.59it/s]
```

```
87398
```

Printing all

In [27]:

```
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)



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)
```

```
**********************************************************************
Categorical Features that are considered :- 
Subject Categories :-  (87398, 9)
Subject Sub-Categories :-  (87398, 30)
Sudent Grade :-  (87398, 4)
School State :-  (87398, 51)
Teacher Prefix :-  (87398, 5)
**********************************************************************
Text Features that are considered :- 
**********************************************************************
Project Essay BOW:-  (87398, 5000)
Project Essay TFIDF:-  (87398, 5000)
**********************************************************************
Project Title BOW:-  (87398, 3305)
Project Title TFIDF:-  (87398, 3305)
**********************************************************************
```

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

Apply Logistic Regression on different kind of featurization as mentioned in the instructions
For Every model that you work on make sure you do the step 2 and step 3 of instrucations

sets

In [28]:

```
#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,text_b
ow,title_bow,price_standardized,quantity_standardized,number_ppp_standardized))
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_text_bow,test_title_bow,test_price_standardized,test_quantity_standardizec
,test_number_ppp_standardized))

set2 =
hstack((categories_one_hot,sub_categories_one_hot,prefix_one_hot,state_one_hot,grade_one_hot,price_
standardized,quantity_standardized,number_ppp_standardized,text_tfidf,title_tfidf))
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_price_standardized,test_quantity_standardized,test_number_ppp_standardizec
,test_text_tfidf,test_title_tfidf))

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)
```

```
(87398, 8407)    (21850, 8407)
(87398, 8407)    (21850, 8407)
(87398, 702)     (21850, 702)
(87398, 702)     (21850, 702)
```

**SET1 (SGD+log loss)**

In [0]:

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

In [0]:

```
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 [0]:

```
#sgd = SGDClassifier(loss="log")
sgd_bal = SGDClassifier(loss="log",class_weight="balanced")

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

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

In [32]:

```
grid.fit(set1,y_train)
```

Out[32]:

```
GridSearchCV(cv=10, error_score='raise-deprecating',
       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='log', 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),
       fit_params=None, iid='warn', n_jobs=-1,
       param_grid={'penalty': ['l1', 'l2'], 'alpha': [1e-05, 0.0001, 0.001, 0.1, 1, 10, 100, 1000,
10000]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

In [33]:

```
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='log', 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'}
0.6863020256964402
```

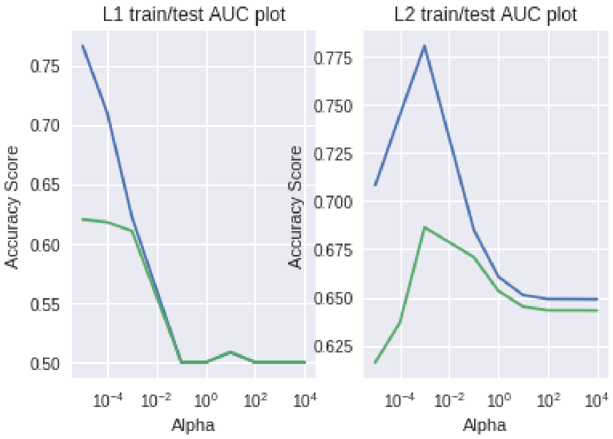
In [34]:

```
#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)
```

```
[0.7665012334501414, 0.7097069630618585, 0.6223570036062789, 0.5000133046999016, 0.5,
0.5084011915322625, 0.5, 0.5, 0.5]
[0.6204503209962282, 0.6180054547545122, 0.6106688112417824, 0.5002596528085683, 0.5,
0.5088472891949261, 0.5, 0.5, 0.5]
[0.7082221553333538, 0.744655681756323, 0.7803324954316638, 0.6849884160153576,
0.6606892418023488, 0.6512144403814121, 0.6491938721011484, 0.6491443979469249,
0.6490450754309043]
[0.6163090344501247, 0.6370165979334412, 0.6863020256964402, 0.6708228628307158,
0.6533699718771292, 0.6452440723755151, 0.6433532007525467, 0.6433305268971765,
0.6432599971722877]
```

In [35]:

```
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()
```

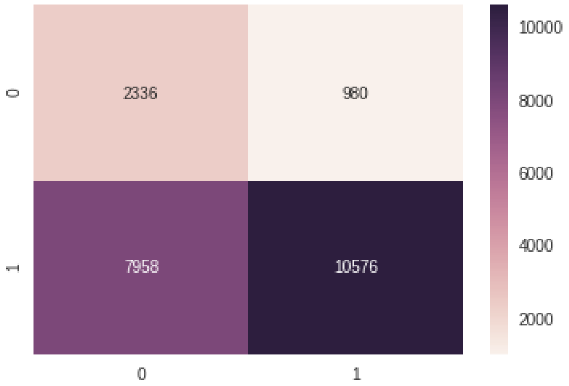


In [36]:

```
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")
```

Out[36]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1b35017400>
```



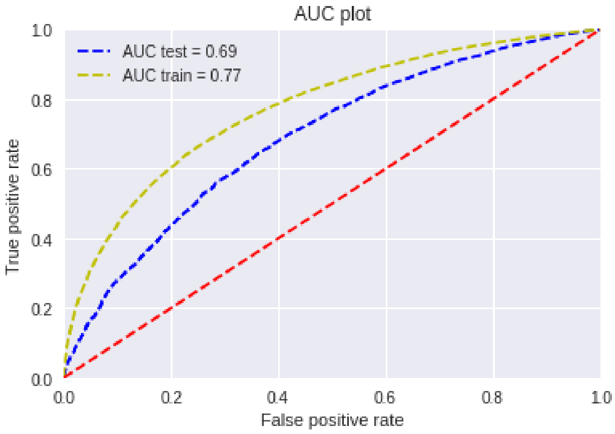
AUC plotting

In [0]:

```
# probabilities calcultion
y1_predict_prob = grid.predict_proba(set1_t)[:,1]
y1_predict_prob_train = grid.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 [38]:

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

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



### Set2

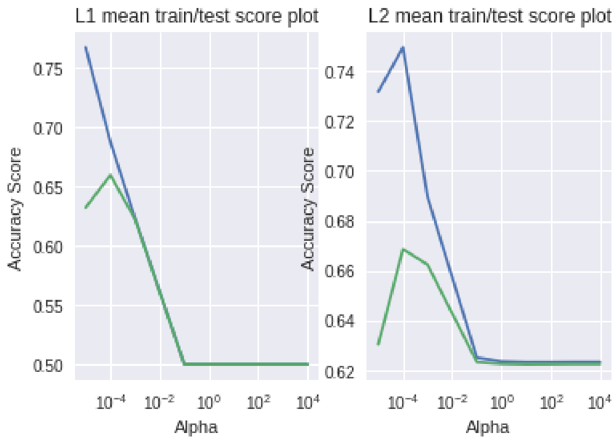
In [39]:

```
grid.fit(set2,y_train)


#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)
```

```
[0.7670394576616517, 0.6874333724208211, 0.6233012442461681, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]
[0.6320377762753624, 0.6595073078941649, 0.62242656095661, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]
[0.7317591440891145, 0.7494837954187982, 0.6895431845896567, 0.6250885697536744,
0.6235664439694502, 0.6232998229960337, 0.6233072152462487, 0.6233524500813271,
0.6233497893542191]
[0.6304572393265719, 0.6685893398131704, 0.6623609240101538, 0.6233202737699608,
0.6225873156162208, 0.6224077577275576, 0.6224555133817674, 0.6225184965585062,
0.6225138803364589]
```

In [40]:

```
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()
```

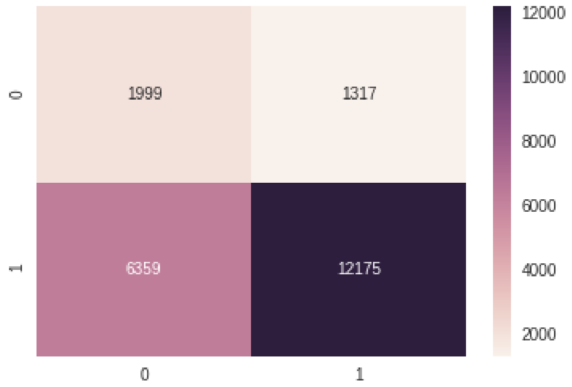


In [41]:

```
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")
```

Out[41]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1b353fdb70>
```



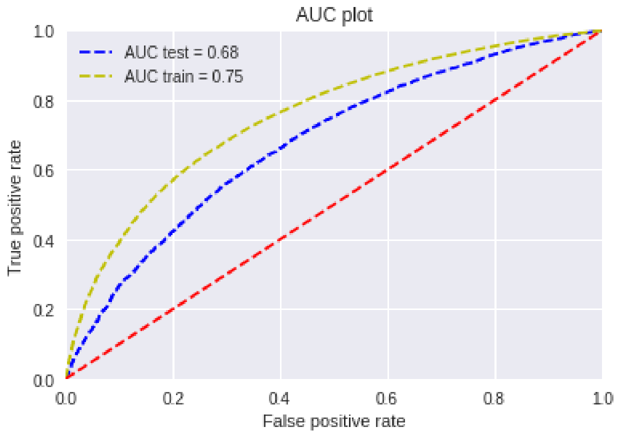
In [0]:

```
# probabilities calcultion
y2_predict_prob = grid.predict_proba(set2_t)[:,1]
y2_predict_prob_train = grid.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 [43]:

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

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



## SET3

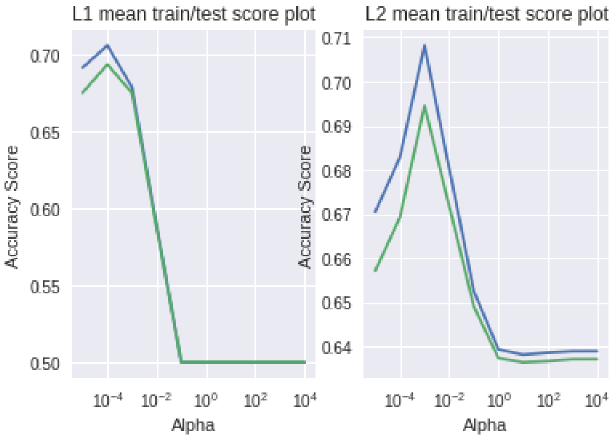
In [44]:

```
grid.fit(set3,y_train)


#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)
```

```
[0.6913540935273976, 0.7057311480766606, 0.6786460378765267, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]
[0.6750872227130325, 0.6934172422591691, 0.6748878226331454, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]
[0.6704657447075538, 0.682856049659837, 0.7082024534603193, 0.6524785632831892,
0.6392556765248021, 0.6380663600979547, 0.6385193454942704, 0.638836906201995, 0.6388482269350175]
[0.6570829888571634, 0.6692318584039729, 0.6945377401406352, 0.6490118912463474,
0.6372809722625842, 0.6363302461921232, 0.6365878509547602, 0.637034508241373, 0.6370380592811131]
```

In [45]:

```
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()
```

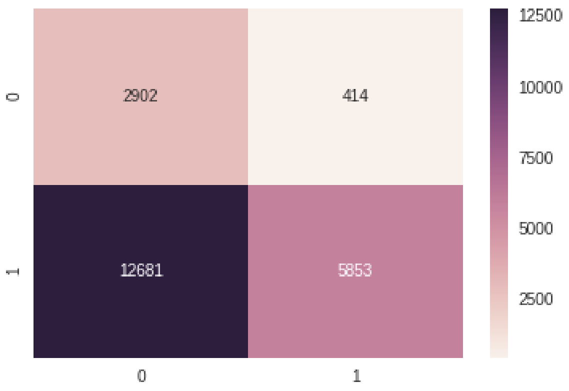


In [46]:

```
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")
```

Out[46]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1b310a3eb8>
```



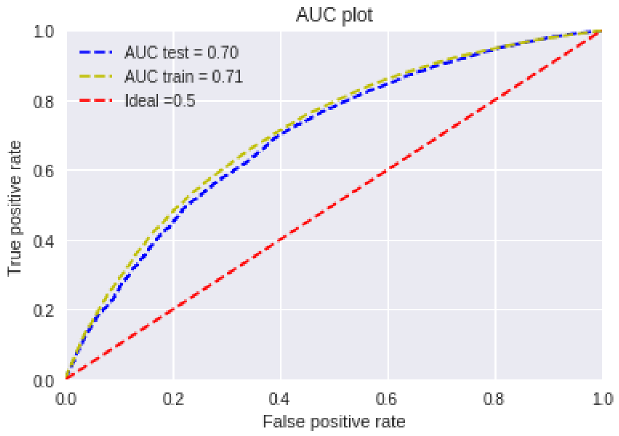
AUC plotting

In [0]:

```
# probabilities calcultion
y3_predict_prob = grid.predict_proba(set3_t)[:,1]
y3_predict_prob_train = grid.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 [48]:

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

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

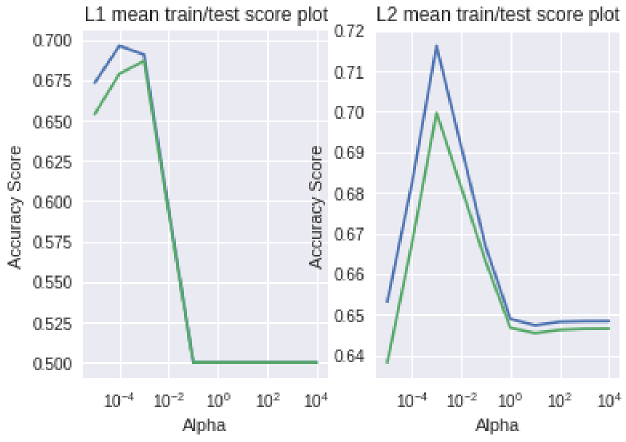
In [49]:

```
grid.fit(set4,y_train)


#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_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)
```

```
[0.673291575253504, 0.6962503434179482, 0.6908555151328969, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]
[0.6538734259560557, 0.6789476230246501, 0.6868842972082144, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]
[0.6531566146782035, 0.6823115109225448, 0.7161836570329465, 0.6665688334872922,
0.6488997271813136, 0.6473497054785988, 0.6482097285292261, 0.6483354367030696,
0.6483595759087908]
[0.6381984873427359, 0.6675056985995755, 0.6997220005525585, 0.6629165252532221,
0.6467143040874936, 0.645422452684753, 0.6462127938251826, 0.6464623902769049, 0.6464998626775139]
```

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()
```

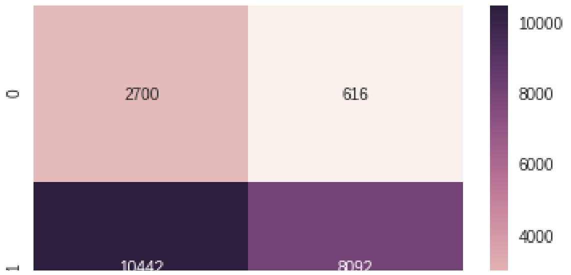


In [51]:

```
y4_predict = grid.predict(set4_t)
cm4 = confusion_matrix(y_test,y4_predict)
# https://seaborn.pydata.org/generated/seaborn.heatmap.html
sns.heatmap(cm4, annot=True, fmt="d")
```

Out[51]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1b377565f8>
```

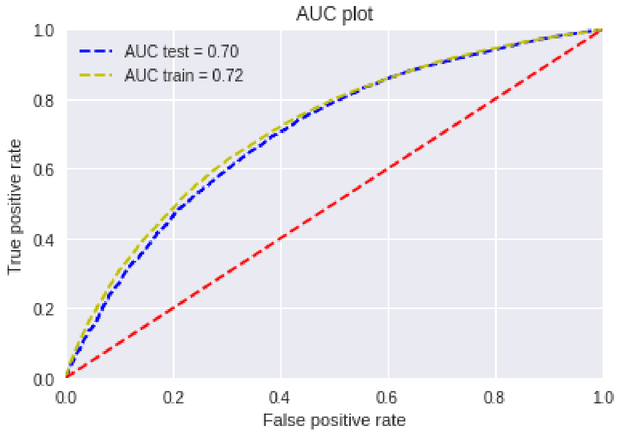


In [0]:

```
# probabilities calcultion
y4_predict_prob = grid.predict_proba(set4_t)[:,1]
y4_predict_prob_train = grid.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 [53]:

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

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



## 2.5 Logistic Regression with added Features `Set 5`

**Sentiment analysis**

In [54]:

```
from textblob import TextBlob

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

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

Out[54]:

```
True
```

for train data

In [55]:

```
# 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']:
  blob = TextBlob(i)
  essay1.append(blob.sentiment.polarity)

for i in X_train['clean_essay_2']:
  blob = TextBlob(i)
  essay2.append(blob.sentiment.polarity)


for i in X_train['clean_essay_3']:
  blob = TextBlob(i)
  essay3.append(blob.sentiment.polarity)


for i in X_train['clean_essay_4']:
  blob = TextBlob(i)
  essay4.append(blob.sentiment.polarity)



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

```
100%|██████████| 87398/87398 [00:05<00:00, 17399.19it/s]
100%|██████████| 87398/87398 [00:05<00:00, 14593.69it/s]
100%|██████████| 87398/87398 [00:01<00:00, 65126.70it/s]
100%|██████████| 87398/87398 [00:01<00:00, 68001.66it/s]
```

```
87398
87398
87398
87398
```

for test data

In [56]:

```
# 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_descr=project_data['clean_essay_1']

for i in X_test['clean_essay_1']:
  blob = TextBlob(i)
  essay1_test.append(blob.sentiment.polarity)

for i in X_test['clean_essay_2']:
  blob = TextBlob(i)
  essay2_test.append(blob.sentiment.polarity)


for i in X_test['clean_essay_3']:
  blob = TextBlob(i)
  essay3_test.append(blob.sentiment.polarity)


for i in X_test['clean_essay_4']:
  blob = TextBlob(i)
  essay4_test.append(blob.sentiment.polarity)


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

```
100%|██████████| 21850/21850 [00:01<00:00, 17347.65it/s]
100%|██████████| 21850/21850 [00:01<00:00, 14637.68it/s]
100%|██████████| 21850/21850 [00:00<00:00, 66540.48it/s]
100%|██████████| 21850/21850 [00:00<00:00, 71009.37it/s]
```

```
21850
21850
21850
21850
```

In [0]:

```
# as lenght of preprocessed array and text have same lenghts
# to store sum of counts of words for title and essay
# 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 [58]:

```
print(len(title_sum))
print(len(essay_sum))
print(len(title_sum_test))
print(len(essay_sum_test))
```

```
87398
87398
21850
21850
```

In [0]:

```
title_sum = np.array(title_sum)
title_sum_test = np.array(title_sum_test)
essay_sum = np.array(essay_sum)
essay_sum_test = np.array(essay_sum_test)
```

In [0]:

```
scalar = StandardScaler()

#train/test data title-sum standardization
title_sum_standardized = scalar.fit_transform(title_sum.reshape(-1, 1))
test_title_sum_standardized = scalar.transform(title_sum_test.reshape(-1, 1))

#train/test data essay-sum standardization
scalar = StandardScaler()
essay_sum_standardized = scalar.fit_transform(essay_sum.reshape(-1, 1))
test_essay_sum_standardized = scalar.transform(essay_sum_test.reshape(-1, 1))
```

In [0]:

```
# conveting to np array
essay1 = np.array(essay1).reshape(-1,1)
essay1_test = np.array(essay1_test).reshape(-1,1)
essay2 = np.array(essay2).reshape(-1,1)
essay2_test = np.array(essay2_test).reshape(-1,1)
```

```
essay2_test = np.array(essay2_test).reshape(-1,1)
essay3 = np.array(essay3).reshape(-1,1)
essay3_test = np.array(essay3_test).reshape(-1,1)
essay4 = np.array(essay4).reshape(-1,1)
essay4_test = np.array(essay4_test).reshape(-1,1)
```

In [0]:

```
set5 =
hstack((categories_one_hot,sub_categories_one_hot,prefix_one_hot,state_one_hot,grade_one_hot,price_
standardized,quantity_standardized,number_ppp_standardized,essay1,essay2,essay3,essay4,title_sum_st
andardized,essay_sum_standardized))
set5_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_standardized
,essay1_test,essay2_test,essay3_test,essay4_test,test_title_sum_standardized,test_essay_sum_standar
dized))
```

Set5 Analysis

In [63]:

```
grid = GridSearchCV(sgd_bal,par_grid,scoring="roc_auc",n_jobs=-1,cv=10)
grid.fit(set5,y_train)
```

Out[63]:

```
GridSearchCV(cv=10, error_score='raise-deprecating',
       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='log', 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),
       fit_params=None, iid='warn', n_jobs=-1,
       param_grid={'penalty': ['l1', 'l2'], 'alpha': [1e-05, 0.0001, 0.001, 0.1, 1, 10, 100, 1000,
10000]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

In [64]:

```
grid.best_params_
```

Out[64]:

```
{'alpha': 0.1, 'penalty': 'l2'}
```

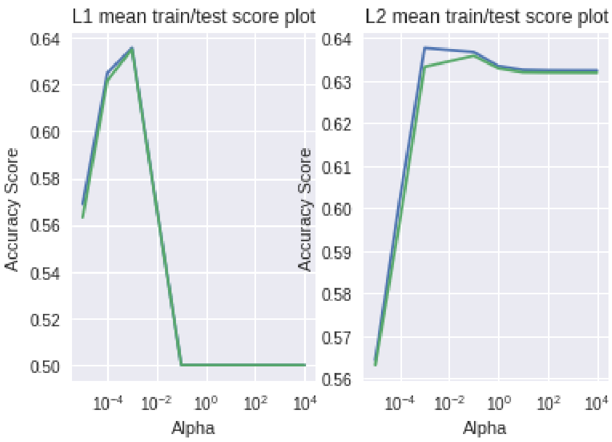
In [65]:

```
#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)
```

```
[0.5689961755587576, 0.6250727375537865, 0.6357291320201085, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]
[0.5631263845018426, 0.6218113824606912, 0.6351097147023935, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]
[0.5643863858899041, 0.6024088195498788, 0.6377001824713203, 0.6367466313802609,
0.6334356152330071, 0.6325501392699556, 0.6324366861129064, 0.6324143644921371, 0.632407687128017]
[0.5631555646076274, 0.5971504218293914, 0.6332489440769077, 0.6358654471445696,
0.6328890880977429, 0.631973958545361, 0.6319135967260345, 0.6318933739053194, 0.6318729016640645]
```

In [66]:

```
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()
```

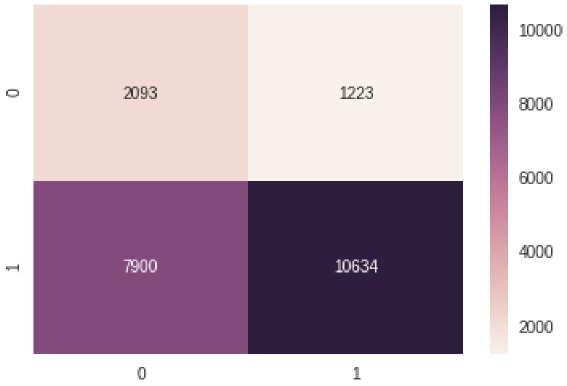


In [67]:

```
y5_predict = grid.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")
```

Out[67]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1b3bddd2b0>
```



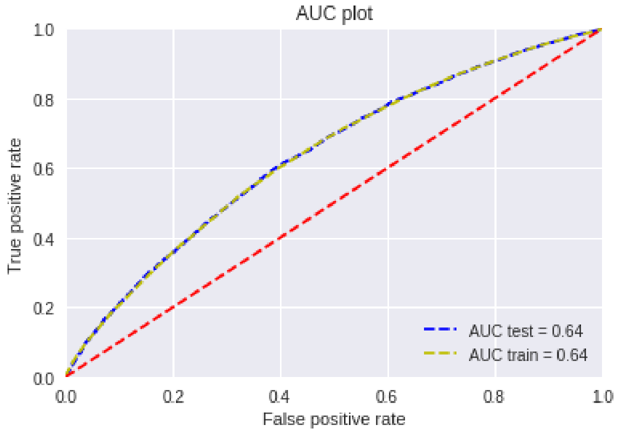
AUC plot

In [0]:

```
# probabilities calcultion
y5_predict_prob = grid.predict_proba(set5_t)[:,1]
y5_predict_prob_train = grid.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 [69]:

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

# auc calculation for train data
roc_auc_train = 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_auc)
plt.plot(fpr_train,tpr_train,"y--",label = 'AUC train = %0.2f'%roc_auc_train)
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 = "lower right")
plt.show()
```



## 3. Conclusion

In [0]:

```
from prettytable import PrettyTable
```

In [0]:

```
# to referance from http://zetcode.com/python/prettytable/
summary = PrettyTable()
```

In [0]:

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

In [0]:

```
summary.add_row(["set1","BOW","SGD+logloss","alpha = 0.001, penalty = 'l2'","test={}\ntrain={}".fo
rmat(0.69,0.77)])
summary.add_row(["set2","TFIDF","SGD+logloss","alpha = 0.001, penalty = 'l2'","test={}\ntrain={}".
format(0.69,0.75)])
summary.add_row(["set3","W2V","SGD+logloss","alpha = 0.001, penalty = 'l2'","test={}\ntrain={}".fo
rmat(0.69,0.71)])
summary.add_row(["set4","TFIDF-W2V","SGD+logloss","alpha = 0.001, penalty = 'l2'","test={}\ntrain=
{}".format(0.71,0.72)])
summary.add_row(["set5","Sentiment Analysis","SGD+logloss","alpha = 0.1, penalty = 'l2'","test={}\
ntrain={}".format(0.64,0.64)])
```

In [74]:

```
print(summary)
```

```
+-------+--------------------+-------------+--------------------------------+------------+
|  Set  |     Vectorizer     |    Model    |         Hyperparameter         |    AUC     |
+-------+--------------------+-------------+--------------------------------+------------+
|  set1 |        BOW         | SGD+logloss | alpha = 0.001, penalty = 'l2'  | test=0.69  |
|       |                    |             |                                | train=0.77 |
|  set2 |       TFIDF        | SGD+logloss | alpha = 0.001, penalty = 'l2'  | test=0.69  |
|       |                    |             |                                | train=0.75 |
|  set3 |        W2V         | SGD+logloss | alpha = 0.001, penalty = 'l2'  | test=0.69  |
|       |                    |             |                                | train=0.71 |
|  set4 |     TFIDF-W2V      | SGD+logloss | alpha = 0.001, penalty = 'l2'  | test=0.71  |
|       |                    |             |                                | train=0.72 |
|  set5 | Sentiment Analysis | SGD+logloss |  alpha = 0.1, penalty = 'l2'   | test=0.64  |
|       |                    |             |                                | train=0.64 |
+-------+--------------------+-------------+--------------------------------+------------+
```

In [0]:

```
############### Conclusion #############################
# 1. Best Results found for set4 i.e TFIDF-W2V
# 2. Dimentionality was less for sentiment analysis
# 3. For sentiment analysis i have used TextBlob library
# 4. AUC for test data is same for BOW/TFIDF/W2V
# 5. Highest test accuracy was found in case of TFIDF-W2V
# 6. Parameters remained same for set 1 to 4
# 7. alpha found to be 0.1 for sentiment analysis
# 8. In confusiong matrix TPR for all set was found to be high

##################################################
# 1. All referances are mentioned in respective code section
# 2. Please let me know if my approach for sentiment analysis was right or not
```