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:** • `Art Will Make You Happy!` • `First Grade Fun`
`project_grade_category`	Grade level of students for which the project is targeted. One of the following enumerated values: • `Grades PreK-2` • `Grades 3-5` • `Grades 6-8` • `Grades 9-12`
`project_subject_categories`	One or more (comma-separated) subject categories for the project from the following enumerated list of values: • `Applied Learning` • `Care & Hunger` • `Health & Sports` • `History & Civics` • `Literacy & Language` • `Math & Science` • `Music & The Arts` • `Special Needs` • `Warmth` **Examples:** • `Music & The Arts` • `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:** • `Literacy` • `Literature & Writing, Social Sciences`
`project_resource_summary`	An explanation of the resources needed for the project. **Example:** • `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: • `nan` • `Dr.` • `Mr.` • `Mrs.` • `Ms.` • `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 [1]:

```
%matplotlib inline

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
```

```
C:\Users\Kamlesh\Anaconda3\lib\site-packages\gensim\utils.py:1197: UserWarning: detected Windows; aliasing chunkize to chunkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

In [2]:

```
import warnings
warnings.filterwarnings("ignore")
```

1.1 Reading Data

In [ ]:

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

In [ ]:

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

In [ ]:

```
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 [ ]:

```
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')
        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 [ ]:

```
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 [ ]:

```
# 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 [ ]:

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

In [ ]:

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

In [ ]:

```
# 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 [ ]:

```
# 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 [ ]:

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

In [ ]:

```
# \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 [ ]:

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

In [ ]:

```
# https://gist.github.com/sebleier/554280
```

```
# 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', "d
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [ ]:

```
# 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 [ ]:

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

1.4 Preprocessing of `project_title`

In [ ]:

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

1.5 Preparing data for models

In [ ]:

```
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 [ ]:

```
# 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 [ ]:

```
# 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 [ ]:

```
# 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 [ ]:

```
# 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 [ ]:

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

1.5.2.2 TFIDF vectorizer

In [ ]:

```
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 [ ]:

```
'''
# 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 [ ]:

```
# 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 [ ]:

```
# 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
    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 [ ]:

```
# 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 [ ]:

```
# 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 [ ]:

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

1.5.3 Vectorizing Numerical features

In [ ]:

```
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 [ ]:

```
# 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_normalized = 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_normalized = price_scalar.transform(project_data['price'].values.reshape(-1, 1))
```

In [ ]:

```
price_normalized
```

1.5.4 Merging all the above features

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

In [ ]:

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

In [ ]:

```
# 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_normalized))
X.shape
```

Computing Sentiment Scores

In [ ]:

```
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 w
ith the biggest enthusiasm \
for learning my students learn in many different ways using all of our senses and multiple intelli
gences i use a wide range\
of techniques to help all my students succeed students in my class come from a variety of differen
t backgrounds which makes\
for wonderful sharing of experiences and cultures including native americans our school is a carin
g 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 ki
ndergarten curriculum\
montana is the perfect place to learn about agriculture and nutrition my students love to role pla
y 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 co
oking delicious healthy \
food for snack time my students will have a grounded appreciation for the work that went into maki
```

```
ng the food and knowledge \
of where the ingredients came from as well as how it is healthy for their bodies this project woul
d expand our learning of \
nutrition and agricultural cooking recipes by having us peel our own apples to make homemade apple
sauce make our own bread \
and mix up healthy plants from our classroom garden in the spring we will also create our own cook
books to be printed and \
shared with families students will gain math and literature skills as well as a life long enjoymen
t 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 [3]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

In [4]:

```
# Loading Data Sets
project_data = pd.read_csv('train_data.csv')
resource_data = pd.read_csv('resources.csv')
```

In [5]:

```
# merging project_data and resource_data
price_data = resource_data.groupby(by = 'id')
price_data = price_data.agg({'price' : 'sum', 'quantity' : 'sum'}).reset_index()
project_data = pd.merge(project_data, price_data, on = 'id', how = 'left')
```

In [6]:

```
# summary table for storing AUC scores of every model

# http://zetcode.com/python/prettytable/
from prettytable import PrettyTable
summary_table = PrettyTable(field_names = ['Vectorizer', 'Regularizer', 'Hyper parameter "C"',
                                           'AUC on Train Data', 'AUC on Test Data'])
```

In [7]:

```
data = project_data
y = list(data.project_is_approved.values)
data.drop(columns = 'project_is_approved', axis = 1, inplace = True)
```

In [8]:

```
# splitting the data into train, test, and cross validation
from sklearn.model_selection import train_test_split

X_data, X_test, y_data, y_test = train_test_split(data, y, test_size = 0.33, random_state = 0, stra
tify = y)
X_train, X_cv, y_train, y_cv = train_test_split(X_data, y_data, test_size = 0.3, random_state = 0,
stratify = y_data)
```

In [9]:

```
c = Counter()
for i in y_train:
```

```
for i in y_train:
    c.update(str(i))
dict(c)
```

Out[9]:

```
{'1': 43479, '0': 7758}
```

2.2 Make Data Model Ready: encoding numerical, categorical features

In [10]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# make sure you featurize train and test data separatly

# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

2.2.1 Encoding Numerical data

In [11]:

```
from sklearn.preprocessing import Normalizer
```

2.2.1.1 quantity

In [12]:

```
# train data
norm = Normalizer()
quantity_normalized = norm.fit_transform(X_train.quantity.values.reshape(1, -1))
```

In [13]:

```
# cross validation data
cv_quantity_normalized = norm.transform(X_cv.quantity.values.reshape(1, -1))
```

In [14]:

```
# test data
test_quantity_normalized = norm.transform(X_test.quantity.values.reshape(1, -1))
```

2.2.1.2 teacher_number_of_previously_posted_projects

In [15]:

```
# train data
norm = Normalizer()
prev_posted_project_normalized =
norm.fit_transform(X_train.teacher_number_of_previously_posted_projects.values.reshape(1, -1))
```

In [16]:

```
# cross validation data
cv_prev_posted_project_normalized =
norm.transform(X_cv.teacher_number_of_previously_posted_projects.values.reshape(1, -1))
```

In [17]:

In [17]:

```
# test data
test_prev_posted_project_normalized =
norm.transform(X_test.teacher_number_of_previously_posted_projects.values.reshape(1, -1))
```

2.2.1.3 price

In [18]:

```
# train data
norm = Normalizer()
price_normalized = norm.fit_transform(X_train.price.values.reshape(1, -1))
```

In [19]:

```
# cross validation data
cv_price_normalized = norm.transform(X_cv.price.values.reshape(1, -1))
```

In [20]:

```
# test data
test_price_normalized = norm.transform(X_test.price.values.reshape(1, -1))
```

2.2.1.4 number of words in the title

In [21]:

```
# train data
norm = Normalizer()
word_count_title = norm.fit_transform(X_train.project_title.map(lambda x : len(x)).values.reshape(1
, -1))
```

In [22]:

```
# cross validation data
cv_word_count_title = norm.transform(X_cv.project_title.map(lambda x : len(x)).values.reshape(1, -1
))
```

In [23]:

```
# cross validation data
test_word_count_title = norm.transform(X_test.project_title.map(lambda x : len(x)).values.reshape(1
, -1))
```

2.2.1.5 number of words in the combine essays

In [24]:

```
# train data
norm = Normalizer()

# combine all essay (train)
essay = X_train.project_essay_1.map(str) + \
        X_train.project_essay_2.map(str) + \
        X_train.project_essay_3.map(str) + \
        X_train.project_essay_4.map(str)

word_count_essay = norm.fit_transform(essay.map(len).values.reshape(1, -1))
```

In [25]:

```
# cross validation data

# combine all essay (cross validation)
essay = X_cv.project_essay_1.map(str) + \
```

```
        X_cv.project_essay_2.map(str) + \
        X_cv.project_essay_3.map(str) + \
        X_cv.project_essay_4.map(str)
cv_word_count_essay = norm.transform(essay.map(lambda x : len(x)).values.reshape(1, -1))
```

In [26]:

```
# cross validation data
# combine all essay (test)
essay = X_test.project_essay_1.map(str) + \
        X_test.project_essay_2.map(str) + \
        X_test.project_essay_3.map(str) + \
        X_test.project_essay_4.map(str)
test_word_count_essay = norm.transform(essay.map(lambda x : len(x)).values.reshape(1, -1))
```

2.2.1.6 sentiment score's of each of the essay

In [27]:

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

nltk.download('vader_lexicon')

ss = SentimentIntensityAnalyzer()
```

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

In [28]:

```
def convert_to_array(ps):
    """ Return array represntation of polarity scores"""
    sentiment = list()
    for p in ps:
        temp = list()
        for k in p.keys():
            temp.append(p[k])
        sentiment.append(np.array(temp))
    return np.array(sentiment)
```

2.2.1.7.1 train data

In [29]:

```
# Finding polarity of each essay
# essay1
ps1 = convert_to_array(X_train.project_essay_1.map(lambda x : ss.polarity_scores(str(x))))
# essay2
ps2 = convert_to_array(X_train.project_essay_2.map(lambda x : ss.polarity_scores(str(x))))
# essay3
ps3 = convert_to_array(X_train.project_essay_3.map(lambda x : ss.polarity_scores(str(x))))
# essay4
ps4 = convert_to_array(X_train.project_essay_4.map(lambda x : ss.polarity_scores(str(x))))
```

2.2.1.7.2 cross validation data

In [30]:

```
# Finding polarity of each essay
# essay1
cv_ps1 = convert_to_array(X_cv.project_essay_1.map(lambda x : ss.polarity_scores(str(x))))
# essay2
cv_ps2 = convert_to_array(X_cv.project_essay_2.map(lambda x : ss.polarity_scores(str(x))))
# essay3
cv_ps3 = convert_to_array(X_cv.project_essay_3.map(lambda x : ss.polarity_scores(str(x))))
# essay4
```

```
cv_ps4 = convert_to_array(X_cv.project_essay_4.map(lambda x : ss.polarity_scores(str(x))))
```

2.2.1.7.1 test data

In [31]:

```
# Finding polarity of each essay
# essay1
test_ps1 = convert_to_array(X_test.project_essay_1.map(lambda x : ss.polarity_scores(str(x))))
# essay2
test_ps2 = convert_to_array(X_test.project_essay_2.map(lambda x : ss.polarity_scores(str(x))))
# essay3
test_ps3 = convert_to_array(X_test.project_essay_3.map(lambda x : ss.polarity_scores(str(x))))
# essay4
test_ps4 = convert_to_array(X_test.project_essay_4.map(lambda x : ss.polarity_scores(str(x))))
```

2.2.2 Preprocessing and Encoding Categorical Data

In [32]:

```
from sklearn.feature_extraction.text import CountVectorizer
```

2.2.2.1 project_subject_category

In [33]:

```
# replace & with _, remove spaces
def preprocess_subj_cat(subj_cat):
    subj_cat_list = list()
    for s in subj_cat:
        temp = ''
        for j in s.split(','):
            if 'The' in j.split(' '):
                j = j.replace('The', '')
            j = j.replace(' ', '')
            j = j.replace('&', '_')
            temp += j.strip() + ' '
        subj_cat_list.append(temp.strip())
    return subj_cat_list
```

In [34]:

```
# preprocessing train data
X_train['clean_category'] = preprocess_subj_cat(list(X_train.project_subject_categories.values))
X_train.drop(columns = 'project_subject_categories', axis = 1, inplace = True)
```

In [35]:

```
# preprocessing cross validation data
X_cv['clean_category'] = preprocess_subj_cat(list(X_cv.project_subject_categories.values))
X_cv.drop(columns = 'project_subject_categories', axis = 1, inplace = True)
```

In [36]:

```
# preprocessing test data
X_test['clean_category'] = preprocess_subj_cat(list(X_test.project_subject_categories.values))
X_test.drop(columns = 'project_subject_categories', axis = 1, inplace = True)
```

In [37]:

```
# encoding train data
cat_vectorizer = CountVectorizer(lowercase = False, binary = True)
category_one_hot = cat_vectorizer.fit_transform(X_train.clean_category.values)

print('Feature : ', cat_vectorizer.get_feature_names())
print('Shape of matrix after one hot encoding : ', category_one_hot.shape)
```

```
Feature :  ['AppliedLearning', 'Care_Hunger', 'Health_Sports', 'History_Civics',
'Literacy_Language', 'Math_Science', 'Music_Arts', 'SpecialNeeds', 'Warmth']
Shape of matrix after one hot encoding :  (51237, 9)
```

In [38]:

```
# encoding cross validation data
cv_category_one_hot = cat_vectorizer.transform(X_cv.clean_category.values)
print('Shape of matrix after one hot encoding : ', cv_category_one_hot.shape)
```

```
Shape of matrix after one hot encoding :  (21959, 9)
```

In [39]:

```
# encoding test data
test_category_one_hot = cat_vectorizer.transform(X_test.clean_category.values)
print('Shape of matrix after one hot encoding : ', test_category_one_hot.shape)
```

```
Shape of matrix after one hot encoding :  (36052, 9)
```

2.2.2.2 project_subject_subcategory

In [40]:

```
# replace & with _; remove spaces
def preprocess_subj_subcat(subj_subcat):
    subj_subcat_list = list()
    for s in subj_subcat:
        temp = ''
        for j in s.split(','):
            if 'The' in j.split(' '):
                j = j.replace('The', '')
            j = j.replace(' ', '')
            j = j.replace('&', '_')
            temp += j.strip() + ' '
        subj_subcat_list.append(temp.strip())
    return subj_subcat_list
```

In [41]:

```
# preprocessing train data
X_train['clean_subcategory'] = preprocess_subj_subcat(list(X_train.project_subject_subcategories.v
alues))
X_train.drop(columns = 'project_subject_subcategories', axis = 1, inplace = True)
```

In [42]:

```
# preprocessing cross validation data
X_cv['clean_subcategory'] = preprocess_subj_subcat(list(X_cv.project_subject_subcategories.values)
)
X_cv.drop(columns = 'project_subject_subcategories', axis = 1, inplace = True)
```

In [43]:

```
# preprocessing test data
X_test['clean_subcategory'] =
preprocess_subj_subcat(list(X_test.project_subject_subcategories.values))
X_test.drop(columns = 'project_subject_subcategories', axis = 1, inplace = True)
```

In [44]:

```
# encoding train data
subcat_vectorizer = CountVectorizer(lowercase = False, binary = True)
subcategory_one_hot = subcat_vectorizer.fit_transform(X_train.clean_subcategory.values)

print('Feature : ', subcat_vectorizer.get_feature_names())
```

```
print('Shape of matrix after one hot encoding : ', subcategory_one_hot.shape)
```

```
Feature :  ['AppliedSciences', 'Care_Hunger', 'CharacterEducation', 'Civics_Government',
'College_CareerPrep', 'CommunityService', 'ESL', 'EarlyDevelopment', 'Economics',
'EnvironmentalScience', 'Extracurricular', 'FinancialLiteracy', 'ForeignLanguages', 'Gym_Fitness',
'Health_LifeScience', 'Health_Wellness', 'History_Geography', 'Literacy', 'Literature_Writing', 'M
athematics', 'Music', 'NutritionEducation', 'Other', 'ParentInvolvement', 'PerformingArts', 'Socia
lSciences', 'SpecialNeeds', 'TeamSports', 'VisualArts', 'Warmth']
Shape of matrix after one hot encoding :  (51237, 30)
```

In [45]:

```
# encoding cross validation data
cv_subcategory_one_hot = subcat_vectorizer.transform(X_cv.clean_subcategory.values)
print('Shape of matrix after one hot encoding : ', cv_subcategory_one_hot.shape)
```

```
Shape of matrix after one hot encoding :  (21959, 30)
```

In [46]:

```
# encoding test data
test_subcategory_one_hot = subcat_vectorizer.transform(X_test.clean_subcategory.values)
print('Shape of matrix after one hot encoding : ', test_subcategory_one_hot.shape)
```

```
Shape of matrix after one hot encoding :  (36052, 30)
```

2.2.2.3 project_grade_category

In [47]:

```
# replacing space by '_' and '-' by '_' in 'project_grade_category'
def preprocess_project_grade_category(grade_cat):
    project_grade_category = list()
    for p in grade_cat:
        p = p.strip()
        p = p.replace(' ', '_')
        p = p.replace('-', '_')
        project_grade_category.append(p.strip())
    return project_grade_category
```

In [48]:

```
# preprocessing train data
X_train['clean_grade_category'] =
preprocess_project_grade_category(list(X_train.project_grade_category.values))
X_train.drop(columns = 'project_grade_category', axis = 1, inplace = True)
```

In [49]:

```
# preprocessing cross validation data
X_cv['clean_grade_category'] = preprocess_project_grade_category(list(X_cv.project_grade_category.
values))
X_cv.drop(columns = 'project_grade_category', axis = 1, inplace = True)
```

In [50]:

```
# preprocessing test data
X_test['clean_grade_category'] =
preprocess_project_grade_category(list(X_test.project_grade_category.values))
X_test.drop(columns = 'project_grade_category', axis = 1, inplace = True)
```

In [51]:

```
# encoding train data
grade_vectorizer = CountVectorizer(lowercase = False, binary = True)
project_grade_category_one_hot =
grade_vectorizer.fit_transform(X_train.clean_grade_category.values)
```

```
print('Features : ', grade_vectorizer.get_feature_names())
print('Shape of matrix after one hot encoding : ', project_grade_category_one_hot.shape)
```

```
Features :  ['Grades_3_5', 'Grades_6_8', 'Grades_9_12', 'Grades_PreK_2']
Shape of matrix after one hot encoding :  (51237, 4)
```

In [52]:

```
# encoding cross validation data
cv_project_grade_category_one_hot = grade_vectorizer.transform(X_cv.clean_grade_category.values)
print('Shape of matrix after one hot encoding : ', cv_project_grade_category_one_hot.shape)
```

```
Shape of matrix after one hot encoding :  (21959, 4)
```

In [53]:

```
# encoding test data
test_project_grade_category_one_hot =
grade_vectorizer.transform(X_test.clean_grade_category.values)
print('Shape of matrix after one hot encoding : ', test_project_grade_category_one_hot.shape)
```

```
Shape of matrix after one hot encoding :  (36052, 4)
```

2.2.2.4 teacher_prefix

In [54]:

```
def preprocess_teacher_prefix(data):
    # replacing nan value by 'nan'
    data.teacher_prefix.replace(to_replace = np.nan, value = 'nan', inplace = True)

    # removing '.'
    clean_teacher_prefix = list()
    for tp in data.teacher_prefix.values:
        tp = tp.replace('.', '')
        clean_teacher_prefix.append(tp)

    data.teacher_prefix = clean_teacher_prefix
```

In [55]:

```
# preprocessing train data
preprocess_teacher_prefix(X_train)
```

In [56]:

```
# preprocessing cross validation data
preprocess_teacher_prefix(X_cv)
```

In [57]:

```
# preprocessing test data
preprocess_teacher_prefix(X_test)
```

In [58]:

```
# encoding train data
teacher_vectorizer = CountVectorizer(lowercase = False, binary = True)
teacher_prefix_one_hot = teacher_vectorizer.fit_transform(X_train.teacher_prefix.values)

print('Features : ', teacher_vectorizer.get_feature_names())
print('Shape of matrix after one hot encoding : ', teacher_prefix_one_hot.shape)
```

```
Features :  ['Dr', 'Mr', 'Mrs', 'Ms', 'Teacher', 'nan']
Shape of matrix after one hot encoding :  (51237, 6)
```

In [59]:

```
# encoding cross validation data
cv_teacher_prefix_one_hot = teacher_vectorizer.transform(X_cv.teacher_prefix.values)
print('Shape of matrix after one hot encoding : ', cv_teacher_prefix_one_hot.shape)
```

```
Shape of matrix after one hot encoding :  (21959, 6)
```

In [60]:

```
# encoding test data
test_teacher_prefix_one_hot = teacher_vectorizer.transform(X_test.teacher_prefix.values)
print('Shape of matrix after one hot encoding : ', test_teacher_prefix_one_hot.shape)
```

```
Shape of matrix after one hot encoding :  (36052, 6)
```

2.2.2.5 school_sates

In [61]:

```
# encoding train data
state_vectorizer = CountVectorizer(lowercase = False, binary = True)
school_states_one_hot = state_vectorizer.fit_transform(X_train.school_state.values)

print('Feature : ', state_vectorizer.get_feature_names())
print('Shape of matrix after one hot encoding : ', school_states_one_hot.shape)
```

```
Feature :  ['AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA', 'HI', 'IA', 'ID', 'II
', 'IN', 'KS', 'KY', 'LA', 'MA', 'MD', 'ME', 'MI', 'MN', 'MO', 'MS', 'MT', 'NC', 'ND', 'NE', 'NH',
'NJ', 'NM', 'NV', 'NY', 'OH', 'OK', 'OR', 'PA', 'RI', 'SC', 'SD', 'TN', 'TX', 'UT', 'VA', 'VT', 'WA
', 'WI', 'WV', 'WY']
Shape of matrix after one hot encoding :  (51237, 51)
```

In [62]:

```
# encoding cross validation data
cv_school_states_one_hot = state_vectorizer.transform(X_cv.school_state.values)
print('Shape of matrix after one hot encoding : ', cv_school_states_one_hot.shape)
```

```
Shape of matrix after one hot encoding :  (21959, 51)
```

In [63]:

```
# encoding test data
test_school_states_one_hot = state_vectorizer.transform(X_test.school_state.values)
print('Shape of matrix after one hot encoding : ', test_school_states_one_hot.shape)
```

```
Shape of matrix after one hot encoding :  (36052, 51)
```

2.3 Make Data Model Ready: encoding eassay, and project_title

In [64]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# make sure you featurize train and test data separatly

# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

2.3.1 preprocessing essay and title

In [65]:

```
# https://stackoverflow.com/a/47091490
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 [66]:

```
import re

def remove_escape_sequences(phrase):
    # \r \n \t remove from string python: http://texthandler.com/info/remove-line-breaks-python/
    phrase = re.sub(r'\\"', ' ', phrase)
    phrase = re.sub(r'\\n', ' ', phrase)
    phrase = re.sub(r'\\r', ' ', phrase)
    phrase = re.sub(r'\\t', ' ', phrase)
    return phrase

def remove_special_characters(phrase):
    return re.sub(r'[^A-Za-z0-9]+', ' ', phrase)
```

In [67]:

```
# 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', 'each', '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', "doesn'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"]
```

2.3.1.1 preprocessing essay

In [68]:

```
def preprocess_essay(dataframe):
    # essay train data
    dataframe_essay = dataframe.project_essay_1.map(str) + \
                      dataframe.project_essay_2.map(str) + \
                      dataframe.project_essay_3.map(str) + \
                      dataframe.project_essay_4.map(str)

    # removing stop words, escape sequences, special character
    preprocessed_essay = list()
    for e in tqdm(dataframe_essay):
        e = decontracted(e)
        e = remove_escape_sequences(e)
        e = remove_special_characters(e)

        temp = ' '.join([word.lower() for word in e.split() if word.lower() not in set(stopwords)])
        preprocessed_essay.append(temp)
    # adding new column of preprocessed essay in dataframe
    dataframe['preprocessed_essay'] = preprocessed_essay
    # removing project_essay columns from dataframe
    dataframe.drop(columns = ['project_essay_1', 'project_essay_2', 'project_essay_3',
'project_essay_4'], axis = 1, inplace = True)
```

In [69]:

```
# essay train data
preprocess_essay(X_train)
```

```
100%|████████████████████████████████████████████████████████████| 51237/51237
[00:47<00:00, 1080.88it/s]
```

In [70]:

```
# essay cross validation data
preprocess_essay(X_cv)
```

```
100%|████████████████████████████████████████████████████████████| 21959/21959
[00:20<00:00, 1079.30it/s]
```

In [71]:

```
# essay test data
preprocess_essay(X_test)
```

```
100%|████████████████████████████████████████████████████████████| 36052/36052
[00:33<00:00, 1084.29it/s]
```

2.3.1.2 preprocessing title

In [72]:

```
def preprocess_title(dataframe):
    # removing stop words, escape sequences, special character
    preprocessed_title = list()
    for e in tqdm(dataframe.project_title):
        e = decontracted(e)
        e = remove_escape_sequences(e)
        e = remove_special_characters(e)

        temp = ' '.join([word.lower() for word in e.split() if word.lower() not in set(stopwords)])
        preprocessed_title.append(temp)
    # adding new column of preprocessed_title in dataframe
    dataframe['preprocessed_title'] = preprocessed_title
    # removing project_title column from dataframe
    dataframe.drop(columns = ['project_title'], axis = 1, inplace = True)
```

In [73]:

```
# project_title train data
preprocess_title(X_train)
```

```
100%|████████████████████████████████████████████████████████████| 51237/51237
[00:01<00:00, 32136.42it/s]
```

In [74]:

```
# project_title cross validation data
preprocess_title(X_cv)
```

```
100%|████████████████████████████████████████████████████████████| 21959/21959
[00:00<00:00, 31229.68it/s]
```

In [75]:

```
# project_title test data
preprocess_title(X_test)
```

```
100%|████████████████████████████████████████████████████████████| 36052/36052
[00:01<00:00, 32017.79it/s]
```

2.3.2 Vectorizing text data (train)

2.3.2.1 Bag Of Words (BOW)

In [76]:

```
# preprocessed_essay train data

# Considering the words which appeared in at least min_df documents(rows)
# using n_gram = (1, 2) means uni-gram and bi-gram
essay_bow_vectorizer = CountVectorizer(min_df = 10, ngram_range = (1, 2), max_features = 5000)
essay_bow = essay_bow_vectorizer.fit_transform(X_train.preprocessed_essay)
print("Shape of matrix after BOW ", essay_bow.shape)
```

```
Shape of matrix after BOW  (51237, 5000)
```

In [77]:

```
# preprocessed_essay cross validation data
cv_essay_bow = essay_bow_vectorizer.transform(X_cv.preprocessed_essay)
print("Shape of matrix after BOW ", cv_essay_bow.shape)
```

```
Shape of matrix after BOW  (21959, 5000)
```

In [78]:

```
# preprocessed_essay test data
test_essay_bow = essay_bow_vectorizer.transform(X_test.preprocessed_essay)
print("Shape of matrix after BOW ", test_essay_bow.shape)
```

```
Shape of matrix after BOW  (36052, 5000)
```

In [79]:

```
# preprocessed_title train data

# Considering the words which appeared in at least min_df documents(rows or projects).
# using n_gram = (1, 2) means uni-gram and bi-gram
title_bow_vectorizer = CountVectorizer(min_df = 10, ngram_range = (1, 2), max_features = 5000)
```

```
title_bow = title_bow_vectorizer.fit_transform(X_train.preprocessed_title)
print("Shape of matrix after BOW ", title_bow.shape)
```

```
Shape of matrix after BOW  (51237, 3366)
```

In [80]:

```
# cv_preprocessed_title cross validation data
cv_title_bow = title_bow_vectorizer.transform(X_cv.preprocessed_title)
print("Shape of matrix after BOW ", cv_title_bow.shape)
```

```
Shape of matrix after BOW  (21959, 3366)
```

In [81]:

```
# cv_preprocessed_title test data
test_title_bow = title_bow_vectorizer.transform(X_test.preprocessed_title)
print("Shape of matrix after BOW ", test_title_bow.shape)
```

```
Shape of matrix after BOW  (36052, 3366)
```

2.3.2.2 Term Frequency Inverse Documnet Frequency (TFIDF)

In [82]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [83]:

```
# preprocessed_essay train data

# Considering the words which appeared in at least min_df documents(rows or projects).
# using n_gram = 1
essay_tfidf_vectorizer = TfidfVectorizer(min_df = 10, ngram_range = (1, 2), max_features = 5000)
essay_tfidf = essay_tfidf_vectorizer.fit_transform(X_train.preprocessed_essay)
print("Shape of matrix after TFIDF ", essay_tfidf.shape)
```

```
Shape of matrix after TFIDF  (51237, 5000)
```

In [84]:

```
# preprocessed_essay cross validation data
cv_essay_tfidf = essay_tfidf_vectorizer.transform(X_cv.preprocessed_essay)
print("Shape of matrix after TFIDF ", cv_essay_tfidf.shape)
```

```
Shape of matrix after TFIDF  (21959, 5000)
```

In [85]:

```
# preprocessed_essay test data
test_essay_tfidf = essay_tfidf_vectorizer.transform(X_test.preprocessed_essay)
print("Shape of matrix after TFIDF ", test_essay_tfidf.shape)
```

```
Shape of matrix after TFIDF  (36052, 5000)
```

In [86]:

```
# preprocessed_title train data

# Considering the words which appeared in at least min_df documents(rows or projects).
# using n_gram = 1
title_tfidf_vectorizer = TfidfVectorizer(min_df = 10, ngram_range = (1, 2), max_features = 5000)
title_tfidf = title_tfidf_vectorizer.fit_transform(X_train.preprocessed_title)
print("Shape of matrix after TFIDF ", title_tfidf.shape)
```

```
Shape of matrix after TFIDF  (51237, 3366)
```

In [87]:

```
# preprocessed_title cross validation data
cv_title_tfidf = title_tfidf_vectorizer.transform(X_cv.preprocessed_title)
print("Shape of matrix after TFIDF ", cv_title_tfidf.shape)
```

```
Shape of matrix after TFIDF  (21959, 3366)
```

In [88]:

```
# preprocessed_title test data
test_title_tfidf = title_tfidf_vectorizer.transform(X_test.preprocessed_title)
print("Shape of matrix after TFIDF ", test_title_tfidf.shape)
```

```
Shape of matrix after TFIDF  (36052, 3366)
```

2.3.2.3 Average Word 2 Vec (W2V)

In [89]:

```
# importing glove w2v model
import pickle
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f) # model is a dictionary
    glove_words = set(model.keys())
```

In [90]:

```
def avg_w2v(data, model, glove_words):
    ''' Return the avg_w2v representaion of data'''
    avg_w2v_list = list()
    for sent in tqdm(data):
        temp_w2v = np.zeros(300)
        count_words = 0
        for word in sent.split():
            if word in glove_words:
                temp_w2v += model[word]
                count_words += 1
        if count_words != 0:
            temp_w2v /= count_words
        avg_w2v_list.append(temp_w2v)
    return avg_w2v_list
```

In [91]:

```
# preprocessed_essay train data
essay_avg_w2v = avg_w2v(X_train.preprocessed_essay, model, glove_words)
```

```
100%|████████████████████████████████████████████████████████████████| 51237/51237
[00:11<00:00, 4444.18it/s]
```

In [92]:

```
# preprocessed_essay cross validation data
cv_essay_avg_w2v = avg_w2v(X_cv.preprocessed_essay, model, glove_words)
```

```
100%|████████████████████████████████████████████████████████████████| 21959/21959
[00:04<00:00, 4402.33it/s]
```

In [93]:

```
# preprocessed_essay test data
test_essay_avg_w2v = avg_w2v(X_test.preprocessed_essay, model, glove_words)
```

```
100%|████████████████████████████████████████████████████████████████| 36052/36052
[00:08<00:00, 4438.99it/s]
```

In [94]:

```
# preprocessed_title train data
title_avg_w2v = avg_w2v(X_train.preprocessed_title, model, glove_words)
```

```
100%|████████████████████████████████████████████████████████████████| 51237/51237
[00:00<00:00, 85331.81it/s]
```

In [95]:

```
# preprocessed_title cross validation data
cv_title_avg_w2v = avg_w2v(X_cv.preprocessed_title, model, glove_words)
```

```
100%|████████████████████████████████████████████████████████████████| 21959/21959
[00:00<00:00, 84359.22it/s]
```

In [96]:

```
# preprocessed_title test data
test_title_avg_w2v = avg_w2v(X_test.preprocessed_title, model, glove_words)
```

```
100%|████████████████████████████████████████████████████████████████| 36052/36052
[00:00<00:00, 84801.27it/s]
```

2.3.2.4 TFIDF weighted Word 2 Vec (W2V)

In [97]:

```
def tfidf_w2v(data, model, glove_words, tfidf_model, tfidf_words):
    ''' Return the tfidf weighted w2v representaion of data'''
    tfidf_w2v_list = list()
    for sent in tqdm(data):
        temp_w2v = np.zeros(300)
        tfidf_weight = 0
        for word in sent.split():
            if word in glove_words and word in tfidf_words:
                vec = model[word]
                tfidf = tfidf_model[word] * (sent.count(word) / len(sent.split()))
                temp_w2v += vec * tfidf
                tfidf_weight += tfidf
        if tfidf_weight != 0:
            temp_w2v /= tfidf_weight
        tfidf_w2v_list.append(temp_w2v)
    return tfidf_w2v_list
```

In [98]:

```
# preprocessed_essay train data

vectorizer = TfidfVectorizer()
essay_tfidf_vec = vectorizer.fit(X_train.preprocessed_essay)

# Making a dictionary with key as word and value as idf value of that word
essay_tfidf_model = dict(zip(essay_tfidf_vec.get_feature_names(), essay_tfidf_vec.idf_))
essay_tfidf_words = set(essay_tfidf_vec.get_feature_names())

essay_tfidf_w2v = tfidf_w2v(X_train.preprocessed_essay, model, glove_words, essay_tfidf_model,
essay_tfidf_words)
```

```
100%|████████████████████████████████████████████████████████████████| 51237/51237 [01:
22<00:00, 617.97it/s]
```

In [99]:

```
# preprocessed_essay cross validation data
cv_essay_tfidf_w2v = tfidf_w2v(X_cv.preprocessed_essay, model, glove_words, essay_tfidf_model,
essay_tfidf_words)
```

```
100%|████████████████████████████████████████████████████████████████| 21959/21959 [00:
35<00:00, 617.54it/s]
```

In [100]:

```
# preprocessed_essay cross validation data
test_essay_tfidf_w2v = tfidf_w2v(X_test.preprocessed_essay, model, glove_words, essay_tfidf_model,
essay_tfidf_words)
```

```
100%|████████████████████████████████████████████████████████████████| 36052/36052 [00:
58<00:00, 618.80it/s]
```

In [101]:

```
# preprocessed_title train data

vectorizer = TfidfVectorizer()
title_tfidf_vec = vectorizer.fit(X_train.preprocessed_title)

# Making a dictionary with key as word and value as idf value of that word
title_tfidf_model = dict(zip(title_tfidf_vec.get_feature_names(), title_tfidf_vec.idf_))
title_tfidf_words = set(title_tfidf_vec.get_feature_names())

title_tfidf_w2v = tfidf_w2v(X_train.preprocessed_title, model, glove_words, title_tfidf_model,
title_tfidf_words)
```

```
100%|████████████████████████████████████████████████████████████████| 51237/51237
[00:01<00:00, 41597.51it/s]
```

In [102]:

```
# preprocessed_title cross validation data
cv_title_tfidf_w2v = tfidf_w2v(X_cv.preprocessed_title, model, glove_words, title_tfidf_model,
title_tfidf_words)
```

```
100%|████████████████████████████████████████████████████████████████| 21959/21959
[00:00<00:00, 41938.49it/s]
```

In [103]:

```
# preprocessed_title test data
test_title_tfidf_w2v = tfidf_w2v(X_test.preprocessed_title, model, glove_words, title_tfidf_model,
title_tfidf_words)
```

```
100%|████████████████████████████████████████████████████████████████| 36052/36052
[00:00<00:00, 41478.81it/s]
```

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

In [104]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# when you plot any graph make sure you use
```

```
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

In [105]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score, roc_curve
```

In [106]:

```
# plot confusion matrix python; https://scikit-
learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html
from sklearn.metrics import confusion_matrix
from sklearn.utils.multiclass import unique_labels

def plot_confusion_matrix(y_true, y_pred, classes, title = None, cmap = plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    """
    if not title:
        title = 'Confusion matrix'

    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    # Only use the labels that appear in the data
    classes = classes[unique_labels(y_true, y_pred)]
    print('Confusion matrix')
    print(cm)

    fig, ax = plt.subplots()
    im = ax.imshow(cm, interpolation = 'nearest', cmap = cmap)
    ax.figure.colorbar(im, ax = ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           # ... and label them with the respective list entries
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha = "right", rotation_mode = "anchor")

    # Loop over data dimensions and create text annotations.
    fmt = 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    plt.grid()
    plt.show()
    return ax
```

2.4.1 Applying Logistic Regression on BOW, SET 1

In [107]:

```
# Please write all the code with proper documentation
```

In [108]:

```
from sklearn.preprocessing import PolynomialFeatures
```

In [109]:

```
# train data

# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

# feature engineering; https://jakevdp.github.io/PythonDataScienceHandbook/05.04-feature-
engineering.html
pf = PolynomialFeatures(degree = 2, include_bias = False)
featured_data = pf.fit_transform(np.hstack((quantity_normalized.reshape(-1, 1), price_normalized.re
shape(-1, 1),
                                            prev_posted_project_normalized.reshape(-1, 1))))

# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_train = hstack((school_states_one_hot, category_one_hot, subcategory_one_hot, project_grade_categ
ory_one_hot,
                  teacher_prefix_one_hot, featured_data, essay_bow, title_bow)).tocsr()

print('X_train shape : ', X_train.shape)

y_train = np.array(y_train)
print('y_train shape : ', y_train.shape)
```

```
X_train shape :  (51237, 8475)
y_train shape :  (51237,)
```

In [110]:

```
# cross validation data

# feature engineering https://jakevdp.github.io/PythonDataScienceHandbook/05.04-feature-
engineering.html
cv_featured_data = pf.transform(np.hstack((cv_quantity_normalized.reshape(-1, 1),
cv_price_normalized.reshape(-1, 1),
                                           cv_prev_posted_project_normalized.reshape(-1, 1))))

# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
X_cv = hstack((cv_school_states_one_hot, cv_category_one_hot, cv_subcategory_one_hot,
cv_project_grade_category_one_hot,
               cv_teacher_prefix_one_hot, cv_featured_data, cv_essay_bow, cv_title_bow)).tocsr()
print('X_cv shape : ', X_cv.shape)

y_cv = np.array(y_cv)
print('y_cv shape : ', y_cv.shape)
```

```
X_cv shape :  (21959, 8475)
y_cv shape :  (21959,)
```

In [111]:

```
# test data

# feature engineering https://jakevdp.github.io/PythonDataScienceHandbook/05.04-feature-
engineering.html
test_featured_data = pf.transform(np.hstack((test_quantity_normalized.reshape(-1, 1), test_price_no
rmalized.reshape(-1, 1),
                                             test_prev_posted_project_normalized.reshape(-1, 1))))

# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
X_test = hstack((test_school_states_one_hot, test_category_one_hot, test_subcategory_one_hot,
                 test_project_grade_category_one_hot, test_teacher_prefix_one_hot,
                 test_featured_data, test_essay_bow, test_title_bow)).tocsr()
print('X_test shape : ', X_test.shape)

y_test = np.array(y_test)
print('y_test shape : ', y_test.shape)
```

```
X_test shape :  (36052, 8475)
y_test shape :  (36052,)
```

In [112]:

```
def batch_predict(model, X, batch_size = 1000):
```

```
    predicted = list()
    for i in range(0, X.shape[0] - batch_size + 1, batch_size):
        pred = model.predict_proba(X[i : i + batch_size])[:, 1]
        predicted.extend(pred)
    if X.shape[0] % batch_size != 0:
        predicted.extend(model.predict_proba(X[X.shape[0] - X.shape[0] % batch_size : ])[:, 1])
    return np.array(predicted)
```

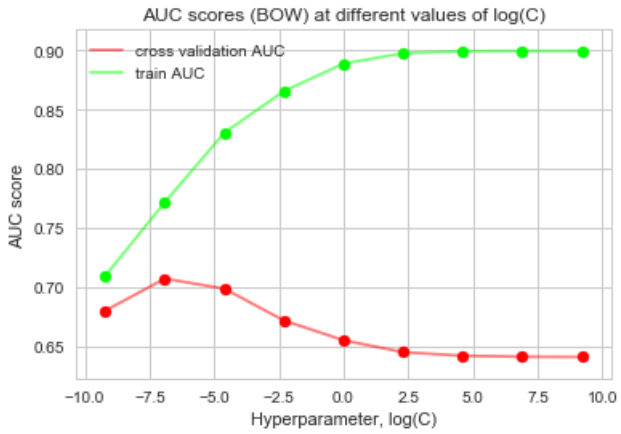
In [113]:

```
# training the LR_model_bow
C_range = list(map(lambda x : 10 ** x, list(np.arange(-4, 5, 1.))))
cv_auc_scores = dict()
train_auc_scores = dict()

for C in tqdm(C_range):
    LR_model_bow = LogisticRegression(penalty = 'l2', C = C, class_weight = 'balanced', n_jobs = -1
, tol = 0.01)
    # fitting the train data
    LR_model_bow.fit(X_train, y_train)
    # predicting the proability scores of train data, cross validation data
    predicted_train = batch_predict(LR_model_bow, X_train, batch_size = 2000)
    predicted_cv = batch_predict(LR_model_bow, X_cv, batch_size = 2000)

    # calculating AUC score for cross validation and test data
    cv_auc_scores[C] = roc_auc_score(y_cv, predicted_cv)
    train_auc_scores[C] = roc_auc_score(y_train, predicted_train)
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 9/9 [03
:33<00:00, 41.64s/it]
```

In [114]:

```
# plotting AUC scores
sns.set(style = 'whitegrid')
plt.plot(np.log(list(cv_auc_scores.keys())), list(cv_auc_scores.values()), color = '#FF000088', lab
el = 'cross validation AUC')
plt.plot(np.log(list(train_auc_scores.keys())), list(train_auc_scores.values()), color =
'#00FF0088', label = 'train AUC')
plt.scatter(np.log(list(cv_auc_scores.keys())), list(cv_auc_scores.values()), color = '#FF0000FF')
plt.scatter(np.log(list(train_auc_scores.keys())), list(train_auc_scores.values()), color=
'#00FF00FF')
plt.legend()
plt.xlabel('Hyperparameter, log(C)')
plt.ylabel('AUC score')
plt.title('AUC scores (BOW) at different values of log(C)')
plt.show()
```



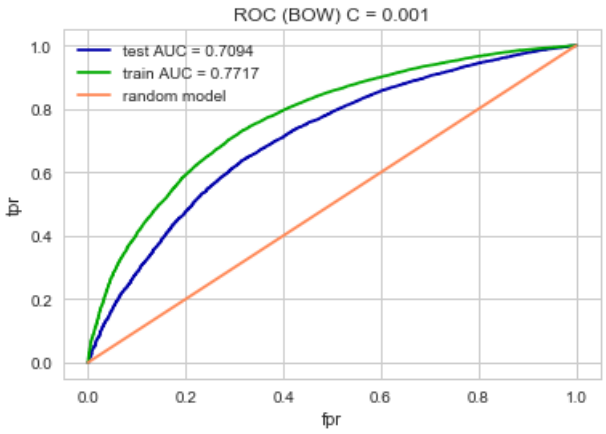
In [115]:

```
best_C = C_range[np.argmax(np.array(list(cv_auc_scores.values())))]
# best LR model (BOW)
best_LR_model_bow = LogisticRegression(penalty = 'l2', C = best_C, class_weight = 'balanced', n_job
s = -1)
# fitting the train data
best_LR_model_bow.fit(X_train, y_train)
```

```
# predicting the proability scores of train data and test data
predicted_test = batch_predict(best_LR_model_bow, X_test, 400)
predicted_train = batch_predict(best_LR_model_bow, X_train, 400)

# calculates fpr and tpr
test_fpr, test_tpr, test_th = roc_curve(y_test, predicted_test)
train_fpr, train_tpr, train_th = roc_curve(y_train, predicted_train)
```

In [116]:

```
# Plotting ROC curve
sns.set(style = 'whitegrid')
plt.plot(test_fpr, test_tpr, color = '#0000AA', label = 'test AUC = %.4f' % (roc_auc_score(y_test,
predicted_test)))
plt.plot(train_fpr, train_tpr, color = '#00AA00', label = 'train AUC = %.4f' % (roc_auc_score(y_tra
in, predicted_train)))
plt.plot([0, 1], [0, 1], color = '#FF8855', label = 'random model')
plt.legend()
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title(f'ROC (BOW) C = {best_C}')
plt.show()
```



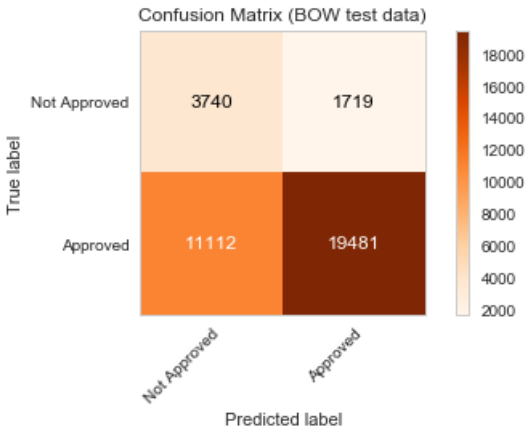
In [117]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def predict(proba, threshold, fpr, tpr):

    t = threshold[np.argmax(tpr*(1-fpr))]

    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high

    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [118]:

```
# adding AUC scores to summary table
summary_table.add_row(['BOW', 'L2', best_C, '%.4f' % (roc_auc_score(y_train, predicted_train)),
                       '%.4f' % (roc_auc_score(y_test, predicted_test))])

# predicting class labels for test data
pred_test = predict(predicted_test, test_th, test_fpr, test_tpr)

# Plotting confusion matrix for test data
class_names = np.array(['Not Approved', 'Approved'])
plot_confusion_matrix(y_test, pred_test, classes = class_names,
                      title = 'Confusion Matrix (BOW test data)', cmap = plt.cm.Oranges)
```

```
the maximum value of tpr*(1-fpr) 0.4362623028911924 for threshold 0.519
Confusion matrix
[[ 3740  1719]
 [11112 19481]]
```



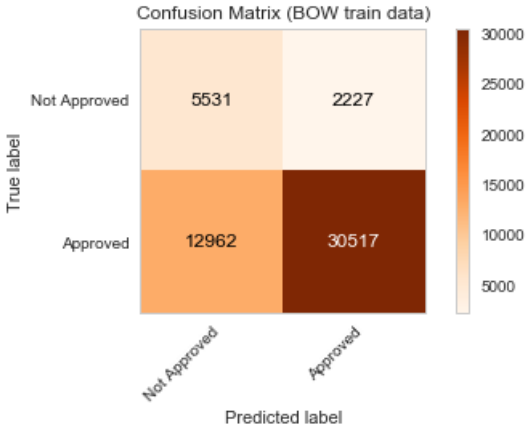
Out[118]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e549a090b8>
```

In [119]:

```
# predicting class labels for train data
pred_train = predict(predicted_train, train_th, train_fpr, train_tpr)

# Plotting confusion matrix for train data
class_names = np.array(['Not Approved', 'Approved'])
plot_confusion_matrix(y_train, pred_train, classes = class_names,
                      title = 'Confusion Matrix (BOW train data)', cmap = plt.cm.Oranges)
```

```
the maximum value of tpr*(1-fpr) 0.5003987013942857 for threshold 0.496
Confusion matrix
[[ 5531  2227]
 [12962 30517]]
```



Out[119]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e54d4aa160>
```

2.4.2 Applying Logistic Regression on TFIDF, SET 2

In [120]:

```
# Please write all the code with proper documentation
```

In [121]:

```
# train data

# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

# feature engineering https://jakevdp.github.io/PythonDataScienceHandbook/05.04-feature-
engineering.html
pf = PolynomialFeatures(degree = 2, include_bias = False, interaction_only = False)
featured_data = pf.fit_transform(np.hstack((quantity_normalized.reshape(-1, 1), price_normalized.re
shape(-1, 1),
                                            prev_posted_project_normalized.reshape(-1, 1))))

# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_train = hstack((school_states_one_hot, category_one_hot, subcategory_one_hot, project_grade_categ
ory_one_hot,
                  teacher_prefix_one_hot, featured_data, essay_tfidf, title_tfidf)).tocsr()

print('X_train shape : ', X_train.shape)

y_train = np.array(y_train)
print('y_train shape : ', y_train.shape)
```

```
X_train shape :  (51237, 8475)
y_train shape :  (51237,)
```

In [122]:

```
# cross validation data


# feature engineering https://jakevdp.github.io/PythonDataScienceHandbook/05.04-feature-
engineering.html
cv_featured_data = pf.transform(np.hstack((cv_quantity_normalized.reshape(-1, 1),
cv_price_normalized.reshape(-1, 1),
                                           cv_prev_posted_project_normalized.reshape(-1, 1))))

# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
X_cv = hstack((cv_school_states_one_hot, cv_category_one_hot, cv_subcategory_one_hot,
cv_project_grade_category_one_hot,
               cv_teacher_prefix_one_hot, cv_featured_data, cv_essay_tfidf, cv_title_tfidf)).tocsr(
)
print('X_cv shape : ', X_cv.shape)

y_cv = np.array(y_cv)
print('y_cv shape : ', y_cv.shape)
```

```
X_cv shape :  (21959, 8475)
y_cv shape :  (21959,)
```

In [123]:

```
# test data

# feature engineering https://jakevdp.github.io/PythonDataScienceHandbook/05.04-feature-
engineering.html
test_featured_data = pf.transform(np.hstack((test_quantity_normalized.reshape(-1, 1), test_price_no
rmalized.reshape(-1, 1),
                                             test_prev_posted_project_normalized.reshape(-1, 1))))

# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
X_test = hstack((test_school_states_one_hot, test_category_one_hot, test_subcategory_one_hot,
                 test_project_grade_category_one_hot, test_teacher_prefix_one_hot,
                 test_featured_data, test_essay_tfidf, test_title_tfidf)).tocsr()
print('X_test shape : ', X_test.shape)

y_test = np.array(y_test)
print('y_test shape : ', y_test.shape)
```

```
X_test shape :  (36052, 8475)
y_test shape :  (36052,)
```

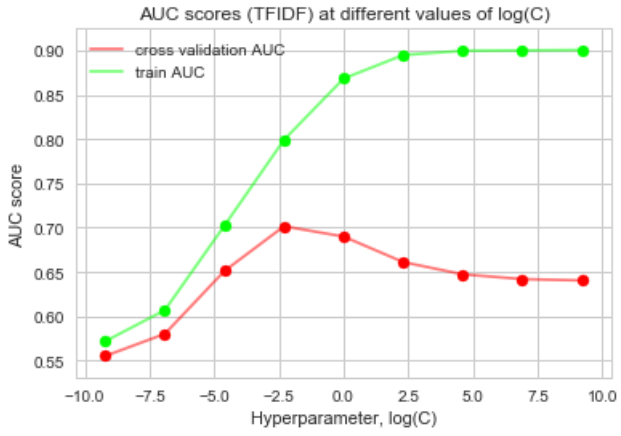
In [124]:

```
# training the LR_model_tfidf
C_range = list(map(lambda x : 10 ** x, range(-4, 5, 1)))
cv_auc_scores = dict()
train_auc_scores = dict()

for C in tqdm(C_range):
    LR_model_tfidf = LogisticRegression(penalty = 'l2', C = C, class_weight = 'balanced', n_jobs =
-1)
    # fitting the train data
    LR_model_tfidf.fit(X_train, y_train)
    # predicting the proability scores of train data, cross validation data
    predicted_train = batch_predict(LR_model_tfidf, X_train, batch_size = 400)
    predicted_cv = batch_predict(LR_model_tfidf, X_cv, batch_size = 400)

    # calculating AUC score for cross validation and test data
    cv_auc_scores[C] = roc_auc_score(y_cv, predicted_cv)
    train_auc_scores[C] = roc_auc_score(y_train, predicted_train)
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 9/9 [06
:16<00:00, 92.10s/it]
```

In [125]:

```
# plotting AUC scores
sns.set(style = 'whitegrid')
plt.plot(np.log(list(cv_auc_scores.keys())), list(cv_auc_scores.values()), color = '#FF000088', lab
el = 'cross validation AUC')
plt.plot(np.log(list(train_auc_scores.keys())), list(train_auc_scores.values()), color =
'#00FF0088', label = 'train AUC')
plt.scatter(np.log(list(cv_auc_scores.keys())), list(cv_auc_scores.values()), color = '#FF0000FF')
plt.scatter(np.log(list(train_auc_scores.keys())), list(train_auc_scores.values()), color=
'#00FF00FF')
plt.legend()
plt.xlabel('Hyperparameter, log(C)')
plt.ylabel('AUC score')
plt.title('AUC scores (TFIDF) at different values of log(C)')
plt.show()
```

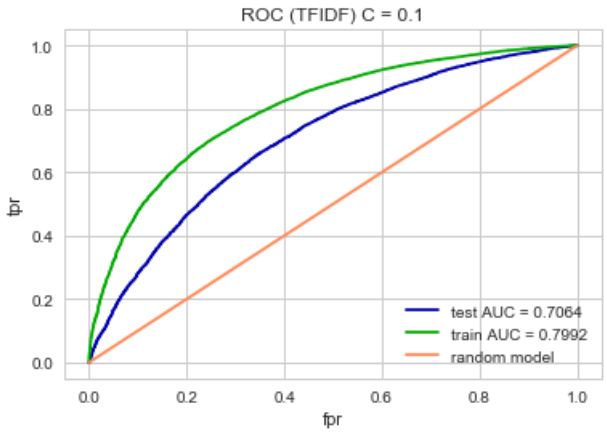


In [126]:

```
best_C = C_range[np.argmax(np.array(list(cv_auc_scores.values())))]
# best LR model (TFIDF)
best_LR_model_tfidf = LogisticRegression(penalty = 'l2', C = best_C, class_weight = 'balanced', n_j
obs = -1)
# fitting the train data
best_LR_model_tfidf.fit(X_train, y_train)

# predicting the proability scores of train data and test data
predicted_test = batch_predict(best_LR_model_tfidf, X_test, 400)
predicted_train = batch_predict(best_LR_model_tfidf, X_train, 400)

# calculates fpr and tpr
test_fpr, test_tpr, test_th = roc_curve(y_test, predicted_test)
train_fpr, train_tpr, train_th = roc_curve(y_train, predicted_train)
```

In [127]:

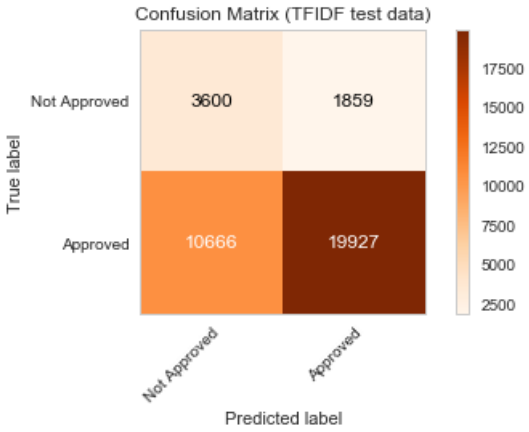
```
# Plotting ROC curve
sns.set(style = 'whitegrid')
plt.plot(test_fpr, test_tpr, color = '#0000AA', label = 'test AUC = %.4f' % (roc_auc_score(y_test,
predicted_test)))
plt.plot(train_fpr, train_tpr, color = '#00AA00', label = 'train AUC = %.4f' % (roc_auc_score(y_tra
in, predicted_train)))
plt.plot([0, 1], [0, 1], color = '#FF8855', label = 'random model')
plt.legend()
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title(f'ROC (TFIDF) C = {best_C}')
plt.show()
```



In [128]:

```
# adding AUC scores to summary table
summary_table.add_row(['TFIDF', 'L2', best_C, '%.4f' % (roc_auc_score(y_train, predicted_train)),
                       '%.4f' % (roc_auc_score(y_test, predicted_test))])

# predicting class labels for test data
pred_test = predict(predicted_test, test_th, test_fpr, test_tpr)

# Plotting confusion matrix for test data
class_names = np.array(['Not Approved', 'Approved'])
plot_confusion_matrix(y_test, pred_test, classes = class_names,
                      title = 'Confusion Matrix (TFIDF test data)', cmap = plt.cm.Oranges)
```

```
the maximum value of tpr*(1-fpr) 0.4295455859633155 for threshold 0.518
Confusion matrix
[[ 3600  1859]
 [10666 19927]]
```



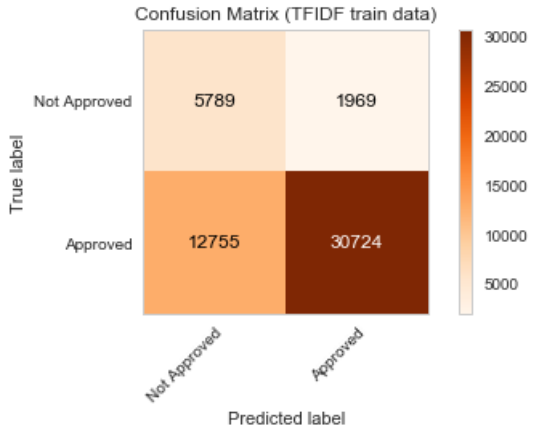
Out[128]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e54cc06da0>
```

In [129]:

```
# predicting class labels for train data
pred_train = predict(predicted_train, train_th, train_fpr, train_tpr)

# Plotting confusion matrix for train data
class_names = np.array(['Not Approved', 'Approved'])
plot_confusion_matrix(y_train, pred_train, classes = class_names,
                      title = 'Confusion Matrix (TFIDF train data)', cmap = plt.cm.Oranges)
```

```
the maximum value of tpr*(1-fpr) 0.5272929731166469 for threshold 0.503
Confusion matrix
[[ 5789  1969]
 [12755 30724]]
```



Out[129]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e54bef7dd8>
```

2.4.3 Applying Logistic Regression on AVG W2V, SET 3

In [130]:

```
# Please write all the code with proper documentation
```

In [131]:

```
# train data

# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

# feature engineering https://jakevdp.github.io/PythonDataScienceHandbook/05.04-feature-
engineering.html
pf = PolynomialFeatures(degree = 2, include_bias = False, interaction_only = False)
featured_data = pf.fit_transform(np.hstack((quantity_normalized.reshape(-1, 1), price_normalized.re
shape(-1, 1),
                                            prev_posted_project_normalized.reshape(-1, 1))))

# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_train = hstack((school_states_one_hot, category_one_hot, subcategory_one_hot, project_grade_categ
ory_one_hot,
                  teacher_prefix_one_hot, featured_data, essay_avg_w2v, title_avg_w2v)).tocsr()
print('X_train shape : ', X_train.shape)

y_train = np.array(y_train)
print('y_train shape : ', y_train.shape)
```

```
X_train shape :  (51237, 709)
y_train shape :  (51237,)
```

In [132]:

```
# cross validation data


# feature engineering https://jakevdp.github.io/PythonDataScienceHandbook/05.04-feature-
engineering.html
cv_featured_data = pf.transform(np.hstack((cv_quantity_normalized.reshape(-1, 1),
cv_price_normalized.reshape(-1, 1),
                                           cv_prev_posted_project_normalized.reshape(-1, 1))))

# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
X_cv = hstack((cv_school_states_one_hot, cv_category_one_hot, cv_subcategory_one_hot,
cv_project_grade_category_one_hot,
               cv_teacher_prefix_one_hot, cv_featured_data, cv_essay_avg_w2v, cv_title_avg_w2v)).to
csr()
print('X_cv shape : ', X_cv.shape)

y_cv = np.array(y_cv)
print('y_cvn shape : ', y_cv.shape)
```

```
X_cv shape :  (21959, 709)
y_cvn shape :  (21959,)
```

In [133]:

```
# test data

# feature engineering https://jakevdp.github.io/PythonDataScienceHandbook/05.04-feature-
engineering.html
test_featured_data = pf.transform(np.hstack((test_quantity_normalized.reshape(-1, 1), test_price_no
rmalized.reshape(-1, 1),
                                             test_prev_posted_project_normalized.reshape(-1, 1))))

# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
X_test = hstack((test_school_states_one_hot, test_category_one_hot, test_subcategory_one_hot,
                 test_project_grade_category_one_hot, test_teacher_prefix_one_hot,
                 test_featured_data, test_essay_avg_w2v, test_title_avg_w2v)).tocsr()
print('X_test shape : ', X_test.shape)

y_test = np.array(y_test)
print('y_test shape : ', y_test.shape)
```

```
X_test shape :  (36052, 709)
y_test shape :  (36052,)
```

In [134]:

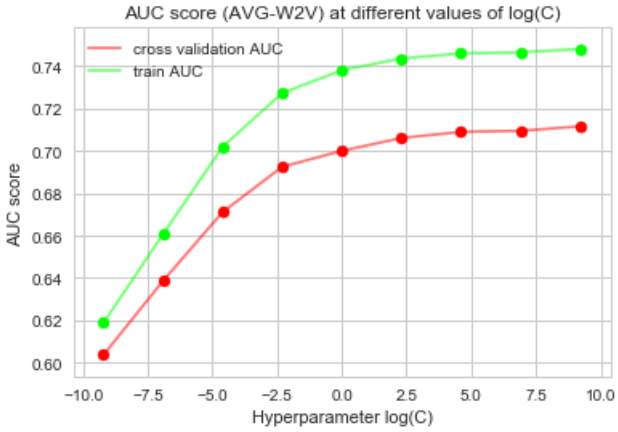
```
# training the LR_model_avg_w2v
C_range = list(map(lambda x : 10 ** x, range(-4, 5, 1)))
cv_auc_scores = dict()
train_auc_scores = dict()

for C in tqdm(C_range):
    LR_model_avg_w2v = LogisticRegression(penalty = 'l2', C = C, class_weight = 'balanced', n_jobs
= -1)
    # fitting the train data
    LR_model_avg_w2v.fit(X_train, y_train)
    # predicting the proability scores of train data, cross validation data
    predicted_train = batch_predict(LR_model_avg_w2v, X_train, batch_size = 400)
    predicted_cv = batch_predict(LR_model_avg_w2v, X_cv, batch_size = 400)

    # calculating AUC score for cross validation and test data
    cv_auc_scores[C] = roc_auc_score(y_cv, predicted_cv)
    train_auc_scores[C] = roc_auc_score(y_train, predicted_train)
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 9/9
[1:00:14<00:00, 976.46s/it]
```

In [135]:

```
# plotting auc curve
sns.set(style = 'whitegrid')
```

```
plt.plot(np.log(list(cv_auc_scores.keys())), list(cv_auc_scores.values()), color = '#FF000088', lab
el = 'cross validation AUC')
plt.plot(np.log(list(train_auc_scores.keys())), list(train_auc_scores.values()), color =
'#00FF0088', label = 'train AUC')
plt.scatter(np.log(list(cv_auc_scores.keys())), list(cv_auc_scores.values()), color = '#FF0000FF')
plt.scatter(np.log(list(train_auc_scores.keys())), list(train_auc_scores.values()), color=
'#00FF00FF')
plt.legend()
plt.xlabel('Hyperparameter log(C)')
plt.ylabel('AUC score')
plt.title('AUC score (AVG-W2V) at different values of log(C)')
plt.show()
```

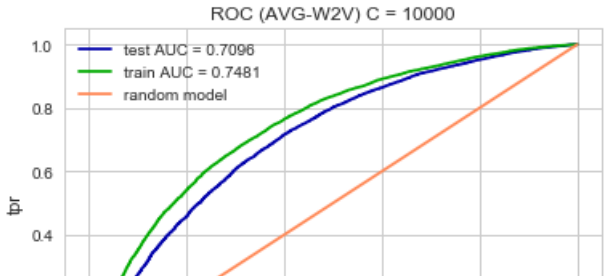


In [136]:

```
best_C = C_range[np.argmax(np.array(list(cv_auc_scores.values())))]
# best LR model (AVG-W2V)
best_LR_model_avg_w2v = LogisticRegression(penalty = 'l2', C = best_C, class_weight = 'balanced', n
_jobs = -1)
# fitting the train data
best_LR_model_avg_w2v.fit(X_train, y_train)

# predicting the proability scores of train data and test data
predicted_test = batch_predict(best_LR_model_avg_w2v, X_test, 400)
predicted_train = batch_predict(best_LR_model_avg_w2v, X_train, 400)

# calculates fpr and tpr
test_fpr, test_tpr, test_th = roc_curve(y_test, predicted_test)
train_fpr, train_tpr, train_th = roc_curve(y_train, predicted_train)
```

In [137]:

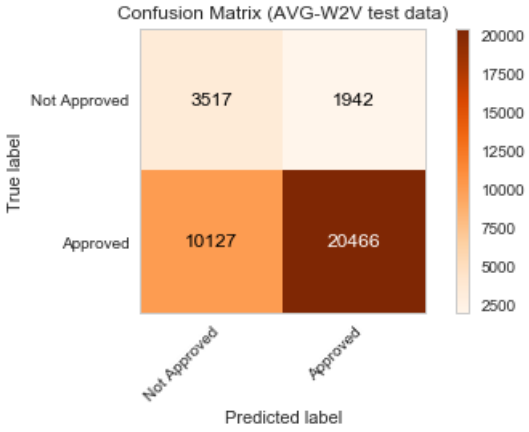
```
# Plots ROC curve
sns.set(style = 'whitegrid')
plt.plot(test_fpr, test_tpr, color = '#0000AA', label = 'test AUC = %.4f' % (roc_auc_score(y_test,
predicted_test)))
plt.plot(train_fpr, train_tpr, color = '#00AA00', label = 'train AUC = %.4f' % (roc_auc_score(y_tra
in, predicted_train)))
plt.plot([0, 1], [0, 1], color = '#FF8855', label = 'random model')
plt.legend()
plt.xlabel('fpr')
plt.ylabel('tpr')

plt.title(f'ROC (AVG-W2V) C = {best_C}')
plt.show()
```



In [138]:

```
# adding AUC scores to summary table
summary_table.add_row(['AVG-W2V', 'L2', best_C, '%.4f' % (roc_auc_score(y_train, predicted_train)),

                       '%.4f' % (roc_auc_score(y_test, predicted_test))])

# predicting class labels for test data
pred_test = predict(predicted_test, test_th, test_fpr, test_tpr)

# Plotting confusion matrix for test data
class_names = np.array(['Not Approved', 'Approved'])
plot_confusion_matrix(y_test, pred_test, classes = class_names,
                      title = 'Confusion Matrix (AVG-W2V test data)', cmap = plt.cm.Oranges)
```

```
the maximum value of tpr*(1-fpr) 0.43099296079994454 for threshold 0.489
Confusion matrix
[[ 3517  1942]
 [10127 20466]]
```



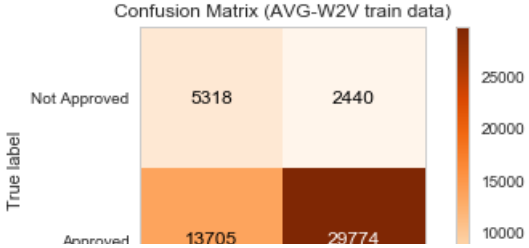
Out[138]:

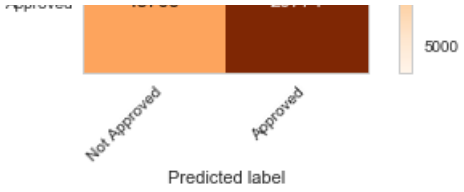
```
<matplotlib.axes._subplots.AxesSubplot at 0x1e54de34f28>
```

In [139]:

```
# predicting class labels for train data
pred_train = predict(predicted_train, train_th, train_fpr, train_tpr)

# Plotting confusion matrix for train data
class_names = np.array(['Not Approved', 'Approved'])
plot_confusion_matrix(y_train, pred_train, classes = class_names,
                      title = 'Confusion Matrix (AVG-W2V train data)', cmap = plt.cm.Oranges)
```

```
the maximum value of tpr*(1-fpr) 0.46941416948219183 for threshold 0.495
Confusion matrix
[[ 5318  2440]
 [13705 29774]]
```





Out[139]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e549cebef0>
```

2.4.4 Applying Logistic Regression on TFIDF W2V, SET 4

In [140]:

```
# Please write all the code with proper documentation
```

In [141]:

```
# train data

# feature engineering https://jakevdp.github.io/PythonDataScienceHandbook/05.04-feature-
engineering.html
pf = PolynomialFeatures(degree = 2, include_bias = False, interaction_only = False)
featured_data = pf.fit_transform(np.hstack((quantity_normalized.reshape(-1, 1), price_normalized.re
shape(-1, 1),
                                            prev_posted_project_normalized.reshape(-1, 1))))

# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_train = hstack((school_states_one_hot, category_one_hot, subcategory_one_hot, project_grade_categ
ory_one_hot,
                  teacher_prefix_one_hot, featured_data, essay_tfidf_w2v, title_tfidf_w2v)).tocsr()
print('X_train shape : ', X_train.shape)

y_train = np.array(y_train)
print('y_train shape : ', y_train.shape)
```

```
X_train shape :  (51237, 709)
y_train shape :  (51237,)
```

In [142]:

```
# cross validation data

# feature engineering https://jakevdp.github.io/PythonDataScienceHandbook/05.04-feature-
engineering.html
cv_featured_data = pf.transform(np.hstack((cv_quantity_normalized.reshape(-1, 1),
cv_price_normalized.reshape(-1, 1),
                                           cv_prev_posted_project_normalized.reshape(-1, 1))))

# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
X_cv = hstack((cv_school_states_one_hot, cv_category_one_hot, cv_subcategory_one_hot,
cv_project_grade_category_one_hot,
               cv_teacher_prefix_one_hot, cv_featured_data, cv_essay_tfidf_w2v, cv_title_tfidf_w2v)
).tocsr()
print('X_cv shape : ', X_cv.shape)

y_cv = np.array(y_cv)
print('y_cvn shape : ', y_cv.shape)
```

```
X_cv shape :  (21959, 709)
y_cvn shape :  (21959,)
```

In [143]:

```
# test data
```

```
# feature engineering https://jakevdp.github.io/PythonDataScienceHandbook/05.04-feature-
engineering.html
test_featured_data = pf.transform(np.hstack((test_quantity_normalized.reshape(-1, 1), test_price_no
rmalized.reshape(-1, 1),
                                             test_prev_posted_project_normalized.reshape(-1, 1))))

# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
X_test = hstack((test_school_states_one_hot, test_category_one_hot, test_subcategory_one_hot,
                 test_project_grade_category_one_hot, test_teacher_prefix_one_hot,
test_featured_data,
                 test_essay_tfidf_w2v, test_title_tfidf_w2v)).tocsr()
print('X_test shape : ', X_test.shape)

y_test = np.array(y_test)
print('y_test shape : ', y_test.shape)
```

```
X_test shape :  (36052, 709)
y_test shape :  (36052,)
```

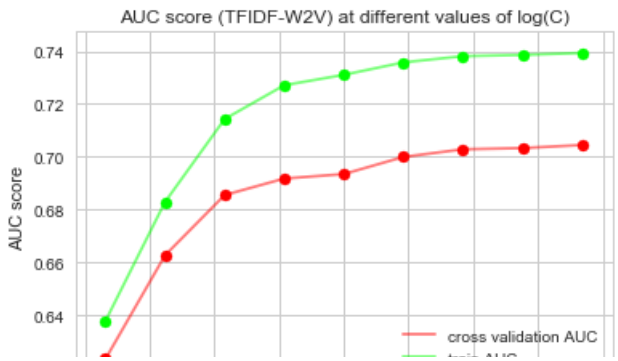
In [144]:

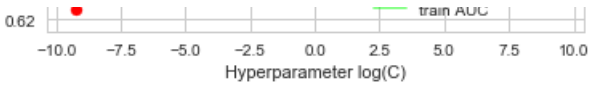
```
# training the LR_model_tfidf_w2v
C_range = list(map(lambda x : 10 ** x, range(-4, 5, 1)))
cv_auc_scores = dict()
train_auc_scores = dict()

for C in tqdm(C_range):
    LR_model_tfidf_w2v = LogisticRegression(penalty = 'l2', C = C, class_weight = 'balanced', n_job
s = -1)
    # fitting the train data
    LR_model_tfidf_w2v.fit(X_train, y_train)
    # predicting the proability scores of train data, cross validation data
    predicted_train = batch_predict(LR_model_tfidf_w2v, X_train, batch_size = 400)
    predicted_cv = batch_predict(LR_model_tfidf_w2v, X_cv, batch_size = 400)

    # calculating AUC score for cross validation and test data
    cv_auc_scores[C] = roc_auc_score(y_cv, predicted_cv)
    train_auc_scores[C] = roc_auc_score(y_train, predicted_train)
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 9/9 [45:
13<00:00, 695.07s/it]
```

In [145]:

```
# plotting auc curve
sns.set(style = 'whitegrid')
plt.plot(np.log(list(cv_auc_scores.keys())), list(cv_auc_scores.values()), color = '#FF000088', lab
el = 'cross validation AUC')
plt.plot(np.log(list(train_auc_scores.keys())), list(train_auc_scores.values()), color =
'#00FF0088', label = 'train AUC')
plt.scatter(np.log(list(cv_auc_scores.keys())), list(cv_auc_scores.values()), color = '#FF0000FF')
plt.scatter(np.log(list(train_auc_scores.keys())), list(train_auc_scores.values()), color=
'#00FF00FF')
plt.legend()
plt.xlabel('Hyperparameter log(C)')
plt.ylabel('AUC score')
plt.title('AUC score (TFIDF-W2V) at different values of log(C)')
plt.show()
```



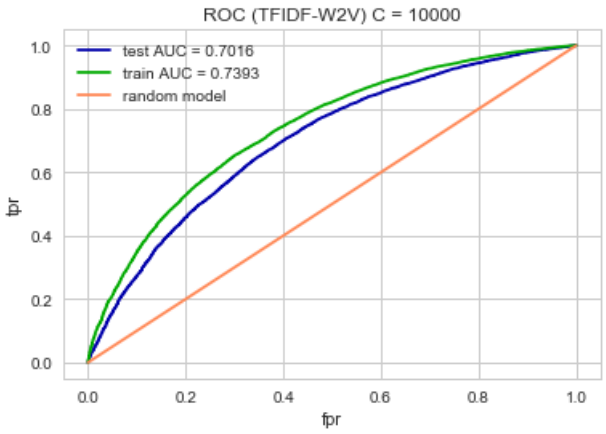


In [146]:

```
best_C = C_range[np.argmax(np.array(list(cv_auc_scores.values())))]
# best LR model (TFIDF-W2V)
best_LR_model_tfidf_w2v = LogisticRegression(penalty = 'l2', C = best_C, class_weight = 'balanced',
n_jobs = -1)
# fitting the train data
best_LR_model_tfidf_w2v.fit(X_train, y_train)

# predicting the proability scores of train data and test data
predicted_test = batch_predict(best_LR_model_tfidf_w2v, X_test, 400)
predicted_train = batch_predict(best_LR_model_tfidf_w2v, X_train, 400)

# calculates fpr and tpr
test_fpr, test_tpr, test_th = roc_curve(y_test, predicted_test)
train_fpr, train_tpr, train_th = roc_curve(y_train, predicted_train)
```

In [147]:

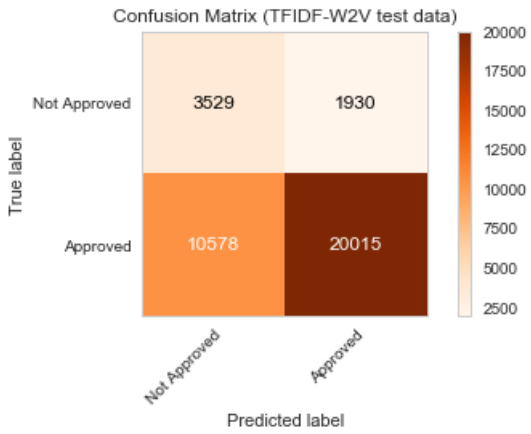
```
# Plots ROC curve
sns.set(style = 'whitegrid')
plt.plot(test_fpr, test_tpr, color = '#0000AA', label = 'test AUC = %.4f' % (roc_auc_score(y_test,
predicted_test)))
plt.plot(train_fpr, train_tpr, color = '#00AA00', label = 'train AUC = %.4f' % (roc_auc_score(y_tra
in, predicted_train)))
plt.plot([0, 1], [0, 1], color = '#FF8855', label = 'random model')
plt.legend()
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title(f'ROC (TFIDF-W2V) C = {best_C}')
plt.show()
```



In [148]:

```
# adding AUC scores to summary table
summary_table.add_row(['TFIDF-W2V', 'L2', best_C, '%.4f' % (roc_auc_score(y_train, predicted_train)
),
                       '%.4f' % (roc_auc_score(y_test, predicted_test))])

# predicting class labels for test data
pred_test = predict(predicted_test, test_th, test_fpr, test_tpr)

# Plotting confusion matrix for test data
class_names = np.array(['Not Approved', 'Approved'])
plot_confusion_matrix(y_test, pred_test, classes = class_names,
                      title = 'Confusion Matrix (TFIDF-W2V test data)', cmap = plt.cm.Oranges)
```

```
the maximum value of tpr*(1-fpr) 0.4229335052508848 for threshold 0.486
Confusion matrix
[[ 3529  1930]
 [10578 20015]]
```



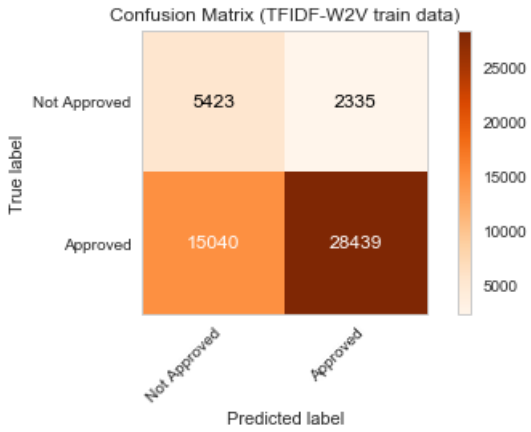
Out[148]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e545b3d0f0>
```

In [149]:

```
# predicting class labels for train data
pred_train = predict(predicted_train, train_th, train_fpr, train_tpr)

# Plotting confusion matrix for train data
class_names = np.array(['Not Approved', 'Approved'])
plot_confusion_matrix(y_train, pred_train, classes = class_names,
                      title = 'Confusion Matrix (TFIDF-W2V train data)', cmap = plt.cm.Oranges)
```

```
the maximum value of tpr*(1-fpr) 0.45721935165875066 for threshold 0.502
Confusion matrix
[[ 5423  2335]
 [15040 28439]]
```



Out[149]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e546b54860>
```

2.5 Logistic Regression with added Features `Set 5`

In [150]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
```

2.5.1 Encoding data for set 5

In [151]:

```
# train data

# feature engineering https://jakevdp.github.io/PythonDataScienceHandbook/05.04-feature-
engineering.html
pf = PolynomialFeatures(degree = 2, include_bias = False, interaction_only = False)
featured_data = pf.fit_transform(np.hstack((quantity_normalized.reshape(-1, 1), price_normalized.re
shape(-1, 1),
                                            prev_posted_project_normalized.reshape(-1, 1),
word_count_essay.reshape(-1, 1),
                                            word_count_title.reshape(-1, 1), ps1, ps2, ps3, ps4)))

# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_train = hstack((school_states_one_hot, category_one_hot, subcategory_one_hot, project_grade_categ
ory_one_hot,
                  teacher_prefix_one_hot, featured_data)).tocsr()
print('X_train shape : ', X_train.shape)

y_train = np.array(y_train)
print('y_train shape : ', y_train.shape)
```

```
X_train shape :  (51237, 352)
y_train shape :  (51237,)
```

In [152]:

```
# cross validation data

# feature engineering https://jakevdp.github.io/PythonDataScienceHandbook/05.04-feature-
engineering.html
cv_featured_data = pf.transform(np.hstack((cv_quantity_normalized.reshape(-1, 1),
cv_price_normalized.reshape(-1, 1),
                                           cv_prev_posted_project_normalized.reshape(-1, 1),
                                           cv_word_count_essay.reshape(-1, 1),
cv_word_count_title.reshape(-1, 1),
                                           cv_ps1, cv_ps2, cv_ps3, cv_ps4)))

# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_cv = hstack((cv_school_states_one_hot, cv_category_one_hot, cv_subcategory_one_hot,
cv_project_grade_category_one_hot,
               cv_teacher_prefix_one_hot, cv_featured_data)).tocsr()
print('X_cv shape : ', X_cv.shape)

y_cv = np.array(y_cv)
print('y_cv shape : ', y_cv.shape)
```

```
X_cv shape :  (21959, 352)
y_cv shape :  (21959,)
```

In [153]:

```
# cross validation data

# feature engineering https://jakevdp.github.io/PythonDataScienceHandbook/05.04-feature-
engineering.html
test_featured_data = pf.transform(np.hstack((test_quantity_normalized.reshape(-1, 1), test_price_no
rmalized.reshape(-1, 1),
                                             test_prev_posted_project_normalized.reshape(-1, 1),
                                             test_word_count_essay.reshape(-1, 1),
test_word_count_title.reshape(-1, 1),
                                             test_ps1, test_ps2, test_ps3, test_ps4)))

# with the same hstack function we are concatinating a sparse matrix and a dense matirx :)
X_test = hstack((test_school_states_one_hot, test_category_one_hot, test_subcategory_one_hot,
                 test_project_grade_category_one_hot, test_teacher_prefix_one_hot,
```

```
test_featured_data)).tocsr()
print('X_test shape : ', X_test.shape)

y_test = np.array(y_test)
print('y_test shape : ', y_test.shape)
```

```
X_test shape :  (36052, 352)
y_test shape :  (36052,)
```

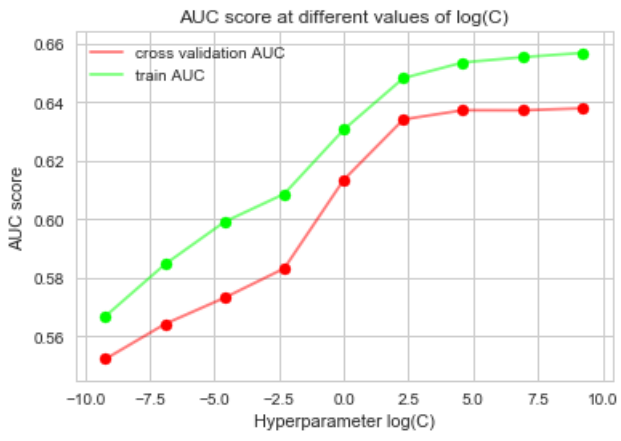
In [154]:

```
# training the LR_model
C_range = list(map(lambda x : 10 ** x, range(-4, 5, 1)))
cv_auc_scores = dict()
train_auc_scores = dict()

for C in tqdm(C_range):
    LR_model = LogisticRegression(penalty = 'l2', C = C, class_weight = 'balanced', n_jobs = -1)
    # fitting the train data
    LR_model.fit(X_train, y_train)
    # predicting the proability scores of train data, cross validation data
    predicted_train = batch_predict(LR_model, X_train, batch_size = 400)
    predicted_cv = batch_predict(LR_model, X_cv, batch_size = 400)

    # calculating AUC score for cross validation and test data
    cv_auc_scores[C] = roc_auc_score(y_cv, predicted_cv)
    train_auc_scores[C] = roc_auc_score(y_train, predicted_train)
```

```
100%|██████████████████████████████████████████████████████████████████████████████| 9/9 [09:
32<00:00, 139.16s/it]
```

In [155]:

```
# plotting auc curve
sns.set(style = 'whitegrid')
plt.plot(np.log(list(cv_auc_scores.keys())), list(cv_auc_scores.values()), color = '#FF000088', lab
el = 'cross validation AUC')
plt.plot(np.log(list(train_auc_scores.keys())), list(train_auc_scores.values()), color =
'#00FF0088', label = 'train AUC')
plt.scatter(np.log(list(cv_auc_scores.keys())), list(cv_auc_scores.values()), color = '#FF0000FF')
plt.scatter(np.log(list(train_auc_scores.keys())), list(train_auc_scores.values()), color=
'#00FF00FF')
plt.legend()
plt.xlabel('Hyperparameter log(C)')
plt.ylabel('AUC score')
plt.title('AUC score at different values of log(C)')
plt.show()
```



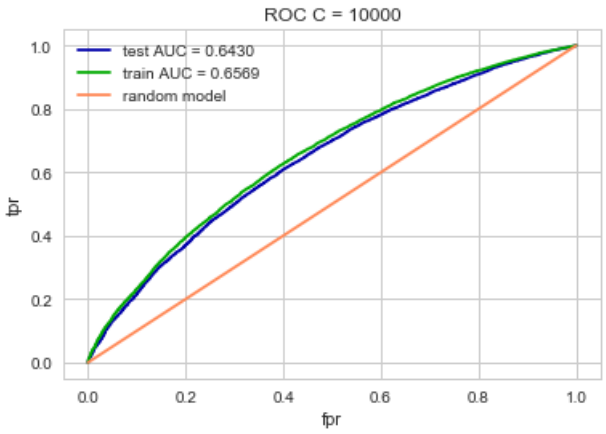
In [156]:

```
best_C = C_range[np.argmax(np.array(list(cv_auc_scores.values())))]
# best LR model
best_LR_model = LogisticRegression(penalty = 'l2', C = best_C, class_weight = 'balanced', n_jobs =
-1)
# fitting the train data
```

```
# fitting the train data
best_LR_model.fit(X_train, y_train)

# predicting the proability scores of train data and test data
predicted_test = batch_predict(best_LR_model, X_test, 2000)
predicted_train = batch_predict(best_LR_model, X_train, 2000)

# calculates fpr and tpr
test_fpr, test_tpr, test_th = roc_curve(y_test, predicted_test)
train_fpr, train_tpr, train_th = roc_curve(y_train, predicted_train)
```

In [157]:

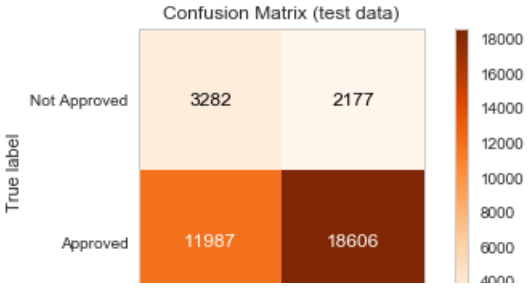
```
# Plots ROC curve
sns.set(style = 'whitegrid')
plt.plot(test_fpr, test_tpr, color = '#0000AA', label = 'test AUC = %.4f' % (roc_auc_score(y_test,
predicted_test)))
plt.plot(train_fpr, train_tpr, color = '#00AA00', label = 'train AUC = %.4f' % (roc_auc_score(y_tra
in, predicted_train)))
plt.plot([0, 1], [0, 1], color = '#FF8855', label = 'random model')
plt.legend()
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title(f'ROC C = {best_C}')
plt.show()
```



In [158]:

```
# adding AUC scores to summary table
summary_table.add_row(['Sentiment Score', 'L2', best_C, '%.4f' % (roc_auc_score(y_train, predicted_
train)),
                       '%.4f' % (roc_auc_score(y_test, predicted_test))])

# predicting class labels for test data
pred_test = predict(predicted_test, test_th, test_fpr, test_tpr)

# Plotting confusion matrix for test data
class_names = np.array(['Not Approved', 'Approved'])
plot_confusion_matrix(y_test, pred_test, classes = class_names,
                      title = 'Confusion Matrix (test data)', cmap = plt.cm.Oranges)
```

```
the maximum value of tpr*(1-fpr) 0.3656423001723872 for threshold 0.525
Confusion matrix
[[ 3282  2177]
 [11987 18606]]
```



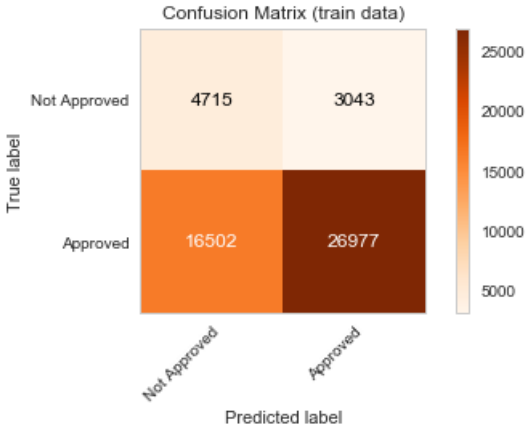
Out[158]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e545896278>
```

In [159]:

```
# predicting class labels for train data
pred_train = predict(predicted_train, train_th, train_fpr, train_tpr)

# Plotting confusion matrix for train data
class_names = np.array(['Not Approved', 'Approved'])
plot_confusion_matrix(y_train, pred_train, classes = class_names,
                      title = 'Confusion Matrix (train data)', cmap = plt.cm.Oranges)
```

```
the maximum value of tpr*(1-fpr) 0.3770908780603836 for threshold 0.498
Confusion matrix
[[ 4715  3043]
 [16502 26977]]
```



Out[159]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e54c893390>
```

3. Conclusion

In [160]:

```
# Please compare all your models using Prettytable library
```

In [161]:

```
print(summary_table)
```

```
+-----------------+-------------+---------------------+--------------------+-------------------+
|    Vectorizer   | Regularizer | Hyper parameter "C" | AUC on Train Data  | AUC on Test Data  |
+-----------------+-------------+---------------------+--------------------+-------------------+
|       BOW       |      L2     |        0.001        |       0.7717       |       0.7094      |
|      TFIDF      |      L2     |         0.1         |       0.7992       |       0.7064      |
|     AVG-W2V     |      L2     |        10000        |       0.7481       |       0.7096      |
|    TFIDF-W2V    |      L2     |        10000        |       0.7393       |       0.7016      |
| Sentiment Score |      L2     |        10000        |       0.6569       |       0.6430      |
+-----------------+-------------+---------------------+--------------------+-------------------+
```

Feature Engineering:

Made Polynomial features of degree 2 using the numerical data

Before Feature Engineering

1. By using BOW vectorizer on text data, both Train and Test AUC scores are high compare to others
2. Every model (except sentiment score) gives almost similar performance on test data but model(using TFIDF) gives highest AUC score on train data

After Feature Engineering

1. Every model give the almost same AUC score on test data except Sentiment Score model.
2. But Sentiment Score model improved alot. Before feature Engineering AUC score on test data was 0.5847 and after feature engineering AUC score on test data is 0.643.
3. AUC score of TFIDF, AVG-W2V, and TFIDF-W2V model improved slighlty.
4. AUC score of BOW model is decreased slighlty.