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`

Feature	Description
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 (https://en.wikipedia.org/wiki/List_of_U.S._state_abbreviations#Postal_codes)). **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!</code`
project_essay_1	First application essay*
project_essay_2	Second application essay*
project_essay_3	Third application essay*
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 [0]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer
import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle
from tqdm import tqdm
import os
import chart_studio.plotly
# from plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
from scipy.sparse import hstack,vstack
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn import model_selection
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import RandomizedSearchCV
#from sklearn.impute import SimpleImputer
from sklearn.datasets import load_digits
#from sklearn.feature_selection import SelectKBest, chi2
```

```
#from sklearn.model_selection import GridSearchCV
from sklearn.feature_selection import SelectKBest,f_classif
from prettytable import PrettyTable
from sklearn.naive_bayes import MultinomialNB
from sklearn.preprocessing import Normalizer
from sklearn.metrics import confusion_matrix
#import math
#from sklearn.linear_model import LogisticRegression
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from sklearn.linear_model import SGDClassifier
nltk.download('vader_lexicon')
import pdb
from sklearn.decomposition import TruncatedSVD
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
```

1.1 Reading Data

In [0]:

```
Project_data = pd.read_csv('train_data.csv')
Resource_data = pd.read_csv('resources.csv')
print(Project_data.shape)
print(Resource_data.shape)
```

```
(109248, 17)
(1541272, 4)
```

In [0]:

```
# how to replace elements in list python: https://stackoverflow.com/a/2582163/4084039
cols = ['Date' if x=='project_submitted_datetime' else x for x in list(Project_data.columns)]
#sort dataframe based on time pandas python: https://stackoverflow.com/a/49702492/4084039
Project_data['Date'] = pd.to_datetime(Project_data['project_submitted_datetime'])
Project_data.drop('project_submitted_datetime', axis=1, inplace=True)
Project_data.sort_values(by=['Date'], inplace=True)
# how to reorder columns pandas python: https://stackoverflow.com/a/13148611/4084039
Project_data = Project_data[cols]
Project_data.head(2)
```

Out[5]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	Date	project_grade_category	project_subject_categories	project_
55660	8393	p205479	2bf07ba08945e5d8b2a3f269b2b3cfe5	Mrs.	CA	2016-04-27 00:27:36	Grades PreK-2	Math & Science	Applie
76127	37728	p043609	3f60494c61921b3b43ab61bdde2904df	Ms.	UT	2016-04-27 00:31:25	Grades 3-5	Special Needs	

1.2 preprocessing of `project_subject_categories`

In [0]:

```
y = Project_data['project_is_approved'].values
Project_data.drop(['project_is_approved'], axis=1, inplace=True)
lpd = len(Project_data)
ys = np.zeros(lpd, dtype=np.int32)
X = Project_data
```

In [0]:

```
#Spliting the Dataset into three Train,CV and Test
X1, X_Test, Y1, Y_Test = train_test_split(X, y, test_size=0.33, random_state=0, stratify=ys)
nx1 = len(X1)
ys1 = np.zeros(nx1, dtype=np.int32)
X_Train, X_CV, Y_Train, Y_CV = train_test_split(X1, Y1, test_size=0.33, random_state=0, stratify=ys1)
print('Shape of the X_Train data is {0} and Y_Train data is: {1}'.format(X_Train.shape,Y_Train.shape[0]))
print('Shape of the X_CV data is    {0} and Y_CV data is   : {1}'.format(X_CV.shape,Y_CV.shape[0]))
print('Shape of the X_Test data is  {0} and Y_Test data is : {1}'.format(X_Test.shape,Y_Test.shape[0]))
```

```
Shape of the X_Train data is (49041, 16) and Y_Train data is: 49041
Shape of the X_CV data is    (24155, 16) and Y_CV data is   : 24155
Shape of the X_Test data is  (36052, 16) and Y_Test data is : 36052
```

In [0]:

```
# 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

#************************************************************Train Data**********************************************
catogories = list(X_Train['project_subject_categories'].values)
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 & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Science"=> "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())

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

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

cat_dict = dict(my_counter)
sorted_cat_dict_Train = dict(sorted(cat_dict.items(), key=lambda kv: kv[1]))
print(len(sorted_cat_dict_Train))
#************************************************************CV Data**********************************************
catogories = list(X_CV['project_subject_categories'].values)
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 & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Science"=> "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())

```

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

#**************************************************************Test Data*****************************************************
catogories = list(X_Test['project_subject_categories'].values)
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 & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Science"=> "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())

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

```

9

1.3 preprocessing of `project_subject_subcategories`

In [0]:

```
# 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
#************************************************************Train Data************************************************************
sub_catogories = list(X_Train['project_subject_subcategories'].values)
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 & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Science"=> "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())

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

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

sub_cat_dict = dict(my_counter)
sorted_sub_cat_dict_Train = dict(sorted(sub_cat_dict.items(), key=lambda kv: kv[1]))
print(len(sorted_sub_cat_dict_Train))
#************************************************************CV Data************************************************************
sub_catogories = list(X_CV['project_subject_subcategories'].values)
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 & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Science"=> "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())

X_CV['clean_subcategories'] = sub_cat_list
```

```
X_CV.drop(['project_subject_subcategories'], axis=1, inplace=True)

#************************************************************Test Data*****************************************************
sub_catogories = list(X_Test['project_subject_subcategories'].values)
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 & Hunger"]
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Science"=> "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())

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

```
30
```

1.3 Text preprocessing

In [0]:

```
# merge two column text dataframe:
X_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)

X_CV["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)

X_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)
```

In [0]:

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

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

In [0]:

```python
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords = ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', '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"]
```

In [0]:

```
# Combining all the above stundents
# tqdm is for printing the status bar

#----------------------------------PreProcessing of Essays in Train data set-------------------------------------------------
preprocessed_essays_Train = []
for sentance in tqdm(X_Train['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
    preprocessed_essays_Train.append(sent.lower().strip())
# pdb.set_trace()

#----------------------------------PreProcessing of Essays in CV data set-------------------------------------------------
preprocessed_essays_CV = []
for sentance in tqdm(X_CV['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.lower() not in stopwords)
    preprocessed_essays_CV.append(sent.lower().strip())
# pdb.set_trace()

#----------------------------------PreProcessing of Essays in Test data set-------------------------------------------------
preprocessed_essays_Test = []
for sentance in tqdm(X_Test['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e.lower() not in stopwords)
    preprocessed_essays_Test.append(sent.lower().strip())
# pdb.set_trace()
```

```
100%|██████████| 49041/49041 [00:26<00:00, 1836.95it/s]
100%|██████████| 24155/24155 [00:13<00:00, 1820.77it/s]
```

```
100%|██████████| 36052/36052 [00:19<00:00, 1823.46it/s]
```

In [0]:

```python
word_count_essay_Train = []
for a in tqdm(X_Train["essay"]) :
    b = len(a.split())
    word_count_essay_Train.append(b)

X_Train["word_count_essay_Train"] = word_count_essay_Train

word_count_essay_CV = []
for a in tqdm(X_CV["essay"]) :
    b = len(a.split())
    word_count_essay_CV.append(b)

X_CV["word_count_essay_CV"] = word_count_essay_CV

word_count_essay_Test = []
for a in tqdm(X_Test["essay"]) :
    b = len(a.split())
    word_count_essay_Test.append(b)

X_Test["word_count_essay_Test"] = word_count_essay_Test
```

```
100%|██████████| 49041/49041 [00:00<00:00, 66616.32it/s]
100%|██████████| 24155/24155 [00:00<00:00, 67353.42it/s]
100%|██████████| 36052/36052 [00:00<00:00, 65733.43it/s]
```

1.4 Preprocessing of `project_title`

In [0]:

```
# Combining all the above stundents
# tqdm is for printing the status bar

#----------------------------------PreProcessing of Project Title in Train data set---------------------------------------
preprocessed_titles_Train = []
for sentance in tqdm(X_Train['project_title'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_titles_Train.append(sent.lower().strip())
# pdb.set_trace()

#----------------------------------PreProcessing of Project Title in CV data set---------------------------------------
preprocessed_titles_CV = []
for sentance in tqdm(X_CV['project_title'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_titles_CV.append(sent.lower().strip())
# pdb.set_trace()

#----------------------------------PreProcessing of Project Title in Test data set---------------------------------------
preprocessed_titles_Test = []
for sentance in tqdm(X_Test['project_title'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_titles_Test.append(sent.lower().strip())
# pdb.set_trace()
```

```
100%|██████████| 49041/49041 [00:01<00:00, 42155.30it/s]
100%|██████████| 24155/24155 [00:00<00:00, 42563.89it/s]
100%|██████████| 36052/36052 [00:00<00:00, 42638.68it/s]
```

In [0]:

```
word_count_title_Train = []
for a in tqdm(X_Train["project_title"]) :
    b = len(a.split())
    word_count_title_Train.append(b)

X_Train["word_count_title_Train"] = word_count_title_Train

word_count_title_CV = []
for a in tqdm(X_CV["project_title"]) :
    b = len(a.split())
    word_count_title_CV.append(b)

X_CV["word_count_title_CV"] = word_count_title_CV

word_count_title_Test = []
for a in tqdm(X_Test["project_title"]) :
    b = len(a.split())
    word_count_title_Test.append(b)

X_Test["word_count_title_Test"] = word_count_title_Test
```

```
100%|██████████| 49041/49041 [00:00<00:00, 862093.25it/s]
100%|██████████| 24155/24155 [00:00<00:00, 862351.90it/s]
100%|██████████| 36052/36052 [00:00<00:00, 901682.45it/s]
```

1.5 Preparing data for models

1.5.1 Vectorizing Categorical data

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

In [0]:

```
#------------------------Vectorizing categorical data for Train,CV and Test-----------------------------------------------

# we use count vectorizer to convert the values into one hot encoding
vectorizer_cat = CountVectorizer(vocabulary=list(sorted_cat_dict_Train.keys()), lowercase=False, binary=True)
vectorizer_cat.fit(X_Train['clean_categories'].values)
categories_one_hot_Train = vectorizer_cat.transform(X_Train['clean_categories'].values)
categories_one_hot_CV = vectorizer_cat.transform(X_CV['clean_categories'].values)
categories_one_hot_Test = vectorizer_cat.transform(X_Test['clean_categories'].values)
print(vectorizer_cat.get_feature_names())
print("-"*120)
print('Shape of Train dataset matrix after one hot encoding is: {0}'.format(categories_one_hot_Train.shape))
print('Shape of CV dataset matrix after one hot encoding is: {0}'.format(categories_one_hot_CV.shape))
print('Shape of Test dataset matrix after one hot encoding is: {0}'.format(categories_one_hot_Test.shape))
```

```
['Warmth', 'Care_Hunger', 'History_Civics', 'Music_Arts', 'AppliedLearning', 'SpecialNeeds', 'Health_Sports', 'Math_Science', 'Li
teracy_Language']
------------------------------------------------------------------------------------------------------------------------
Shape of Train dataset matrix after one hot encoding is: (49041, 9)
Shape of CV dataset matrix after one hot encoding is: (24155, 9)
Shape of Test dataset matrix after one hot encoding is: (36052, 9)
```

In [0]:

```
#-------------------------Vectorizing Sub categorical data for Train,CV and Test-----------------------------------------

# we use count vectorizer to convert the values into one hot encoding
vectorizer_sub_cat = CountVectorizer(vocabulary=list(sorted_sub_cat_dict_Train.keys()), lowercase=False, binary=True)
vectorizer_sub_cat.fit(X_Train['clean_categories'].values)
sub_categories_one_hot_Train = vectorizer_sub_cat.transform(X_Train['clean_subcategories'].values)
sub_categories_one_hot_CV = vectorizer_sub_cat.transform(X_CV['clean_subcategories'].values)
sub_categories_one_hot_Test = vectorizer_sub_cat.transform(X_Test['clean_subcategories'].values)
print(vectorizer_sub_cat.get_feature_names())
print("-"*120)
print('Shape of Train dataset matrix after one hot encoding is: {0}'.format(sub_categories_one_hot_Train.shape))
print('Shape of CV dataset matrix after one hot encoding is: {0}'.format(sub_categories_one_hot_CV.shape))
print('Shape of Test dataset matrix after one hot encoding is: {0}'.format(sub_categories_one_hot_Test.shape))
```

```
['Economics', 'CommunityService', 'FinancialLiteracy', 'ParentInvolvement', 'Extracurricular', 'Civics_Government', 'ForeignLangu
ages', 'Warmth', 'Care_Hunger', 'NutritionEducation', 'SocialSciences', 'PerformingArts', 'CharacterEducation', 'TeamSports', 'Ot
her', 'College_CareerPrep', 'Music', 'History_Geography', 'EarlyDevelopment', 'Health_LifeScience', 'ESL', 'Gym_Fitness', 'Enviro
nmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences', 'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Litera
cy']
------------------------------------------------------------------------------------------------------------------------
Shape of Train dataset matrix after one hot encoding is: (49041, 30)
Shape of CV dataset matrix after one hot encoding is: (24155, 30)
Shape of Test dataset matrix after one hot encoding is: (36052, 30)
```

School State

In [0]:

```
#------------------------Vectorizing categorical data of School state for Train dataset-----------------------------------

school_catogories_Train = list(X_Train['school_state'].values)
school_list_Train = []
for sent in school_catogories_Train:
    school_list_Train.append(sent.lower().strip())
X_Train['school_categories'] = school_list_Train
X_Train.drop(['school_state'], axis=1, inplace=True)

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

# dict sort by value python: https://stackoverflow.com/a/613218/4084039
school_dict_Train = dict(my_counter_school_Train)
sorted_school_dict_Train = dict(sorted(school_dict_Train.items(), key=lambda kv: kv[1]))

vectorizer_school = CountVectorizer(vocabulary=list(sorted_school_dict_Train.keys()), lowercase=False, binary=True)
vectorizer_school.fit(X_Train['school_categories'].values)
#print(vectorizer.get_feature_names())

school_one_hot_Train = vectorizer_school.transform(X_Train['school_categories'].values)

#------------------------Vectorizing categorical data of School state for CV dataset-----------------------------------

school_catogories_CV = list(X_CV['school_state'].values)
school_list_CV = []
for sent in school_catogories_CV:
    school_list_CV.append(sent.lower().strip())
X_CV['school_categories'] = school_list_CV
X_CV.drop(['school_state'], axis=1, inplace=True)

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

# dict sort by value python: https://stackoverflow.com/a/613218/4084039
school_dict_CV = dict(my_counter_school_CV)
sorted_school_dict_CV = dict(sorted(school_dict_CV.items(), key=lambda kv: kv[1]))
school_one_hot_CV = vectorizer_school.transform(X_CV['school_categories'].values)

#------------------------Vectorizing categorical data of School state for Test dataset-----------------------------------

```

```
school_catogories_Test = list(X_Test['school_state'].values)
school_list_Test = []
for sent in school_catogories_Test:
    school_list_Test.append(sent.lower().strip())
X_Test['school_categories'] = school_list_Test
X_Test.drop(['school_state'], axis=1, inplace=True)

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

# dict sort by value python: https://stackoverflow.com/a/613218/4084039
school_dict_Test = dict(my_counter_school_Test)
sorted_school_dict_Test = dict(sorted(school_dict_Test.items(), key=lambda kv: kv[1]))
school_one_hot_Test = vectorizer_school.transform(X_Test['school_categories'].values)
print("-"*120)
print('Shape of Train dataset matrix after one hot encoding is: {0}'.format(school_one_hot_Train.shape))
print('Shape of CV dataset matrix after one hot encoding is: {0}'.format(school_one_hot_CV.shape))
print('Shape of Test dataset matrix after one hot encoding is: {0}'.format(school_one_hot_Test.shape))
```

```
------------------------------------------------------------------------------------------------------------------------
Shape of Train dataset matrix after one hot encoding is: (49041, 51)
Shape of CV dataset matrix after one hot encoding is: (24155, 51)
Shape of Test dataset matrix after one hot encoding is: (36052, 51)
```

Prefix

In [0]:

```
#-------------------------Vectorizing categorical data of Teacher Prefix for Train dataset-----------------------------------

# 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
prefix_catogories_Train = list(X_Train['teacher_prefix'].values)
prefix_list_Train = []
for sent in prefix_catogories_Train:
    sent = re.sub('[^A-Za-z0-9]+', ' ', str(sent))
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split())
    prefix_list_Train.append(sent.lower().strip())
X_Train['prefix_catogories'] = prefix_list_Train
X_Train.drop(['teacher_prefix'], axis=1, inplace=True)

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

# dict sort by value python: https://stackoverflow.com/a/613218/4084039
prefix_dict_Train = dict(my_counter_prefix_Train)
sorted_prefix_dict_Train = dict(sorted(prefix_dict_Train.items(), key=lambda kv: kv[1]))


vectorizer_prefix = CountVectorizer(vocabulary=list(sorted_prefix_dict_Train.keys()), lowercase=False, binary=True)
vectorizer_prefix.fit(X_Train['prefix_catogories'].values)
#print(vectorizer.get_feature_names())

prefix_one_hot_Train = vectorizer_prefix.transform(X_Train['prefix_catogories'].values)
#print("Shape of matrix after one hot encodig ",prefix_one_hot.shape)

#-------------------------Vectorizing categorical data of Teacher Prefix for CV dataset-----------------------------------

prefix_catogories_CV = list(X_CV['teacher_prefix'].values)
prefix_list_CV = []
for sent in prefix_catogories_CV:
    sent = re.sub('[^A-Za-z0-9]+', ' ', str(sent))
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split())
    prefix_list_CV.append(sent.lower().strip())
X_CV['prefix_catogories'] = prefix_list_CV
X_CV.drop(['teacher_prefix'], axis=1, inplace=True)

```

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

# dict sort by value python: https://stackoverflow.com/a/613218/4084039
prefix_dict_CV = dict(my_counter_prefix_CV)
sorted_prefix_dict_CV = dict(sorted(prefix_dict_CV.items(), key=lambda kv: kv[1]))
prefix_one_hot_CV = vectorizer_prefix.transform(X_CV['prefix_catogories'].values)

#------------------------Vectorizing categorical data of Teacher Prefix for Test dataset-----------------------------------

prefix_catogories_Test = list(X_Test['teacher_prefix'].values)
prefix_list_Test = []
for sent in prefix_catogories_Test:
    sent = re.sub('[^A-Za-z0-9]+', ' ', str(sent))
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split())
    prefix_list_Test.append(sent.lower().strip())
X_Test['prefix_catogories'] = prefix_list_Test
X_Test.drop(['teacher_prefix'], axis=1, inplace=True)

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

# dict sort by value python: https://stackoverflow.com/a/613218/4084039
prefix_dict_Test = dict(my_counter_prefix_Test)
sorted_prefix_dict_Test = dict(sorted(prefix_dict_Test.items(), key=lambda kv: kv[1]))
prefix_one_hot_Test = vectorizer_prefix.transform(X_Test['prefix_catogories'].values)
print("-"*120)
print('Shape of Train dataset matrix after one hot encoding is: {0}'.format(prefix_one_hot_Train.shape))
print('Shape of CV dataset matrix after one hot encoding is: {0}'.format(prefix_one_hot_CV.shape))
print('Shape of Test dataset matrix after one hot encoding is: {0}'.format(prefix_one_hot_Test.shape))
```

```
------------------------------------------------------------------------------------------------------------------------
Shape of Train dataset matrix after one hot encoding is: (49041, 6)
Shape of CV dataset matrix after one hot encoding is: (24155, 6)
Shape of Test dataset matrix after one hot encoding is: (36052, 6)
```

project_grade_category

In [0]:

```
#------------------------Vectorizing categorical data of Project Grade for Train dataset-----------------------------------

# 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
grade_catogories_Train = list(X_Train['project_grade_category'].values)
grade_list_Train = []
for sent in grade_catogories_Train:
    sent = sent.replace('-','_')
    sent = sent.replace(' ','_')
    # sent = re.sub('[^A-Za-z0-9]+', ' ', str(sent))
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split())
    grade_list_Train.append(sent.lower().strip())

# temp = temp.replace('-','_')
X_Train['new_grade_category'] = grade_list_Train
X_Train.drop(['project_grade_category'], axis=1, inplace=True)

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

# dict sort by value python: https://stackoverflow.com/a/613218/4084039
grade_dict_Train = dict(my_counter_grade_Train)
sorted_grade_dict_Train = dict(sorted(grade_dict_Train.items(), key=lambda kv: kv[1]))

vectorizer_grade = CountVectorizer(vocabulary=list(sorted_grade_dict_Train.keys()), lowercase=False, binary=True)
vectorizer_grade.fit(X_Train['new_grade_category'].values)
#print(vectorizer.get_feature_names())

grade_one_hot_Train = vectorizer_grade.transform(X_Train['new_grade_category'].values)

#------------------------Vectorizing categorical data of Project Grade for CV dataset-----------------------------------

grade_catogories_CV = list(X_CV['project_grade_category'].values)
grade_list_CV = []
for sent in grade_catogories_CV:
    sent = sent.replace('-','_')
    sent = sent.replace(' ','_')
    # sent = re.sub('[^A-Za-z0-9]+', ' ', str(sent))
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split())
```

```
    grade_list_CV.append(sent.lower().strip())

# temp = temp.replace('-','_')
X_CV['new_grade_category'] = grade_list_CV
X_CV.drop(['project_grade_category'], axis=1, inplace=True)

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

# dict sort by value python: https://stackoverflow.com/a/613218/4084039
grade_dict_CV = dict(my_counter_grade_CV)
sorted_grade_dict_CV = dict(sorted(grade_dict_CV.items(), key=lambda kv: kv[1]))

grade_one_hot_CV = vectorizer_grade.transform(X_CV['new_grade_category'].values)

#-------------------------Vectorizing categorical data of Project Grade for Train dataset----------------------------------

grade_catogories_Test = list(X_Test['project_grade_category'].values)
grade_list_Test = []
for sent in grade_catogories_Test:
    sent = sent.replace('-','_')
    sent = sent.replace(' ','_')
    # sent = re.sub('[^A-Za-z0-9]+', ' ', str(sent))
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split())
    grade_list_Test.append(sent.lower().strip())

# temp = temp.replace('-','_')
X_Test['new_grade_category'] = grade_list_Test
X_Test.drop(['project_grade_category'], axis=1, inplace=True)

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

# dict sort by value python: https://stackoverflow.com/a/613218/4084039
grade_dict_Test = dict(my_counter_grade_Test)
sorted_grade_dict_Test = dict(sorted(grade_dict_Test.items(), key=lambda kv: kv[1]))

grade_one_hot_Test = vectorizer_grade.transform(X_Test['new_grade_category'].values)
print("-"*120)
print('Shape of Train dataset matrix after one hot encoding is: {0}'.format(grade_one_hot_Train.shape))
print('Shape of CV dataset matrix after one hot encoding is: {0}'.format(grade_one_hot_CV.shape))
```

```
print('Shape of Test dataset matrix after one hot encoding is: {0}'.format(grade_one_hot_Test.shape))
```

```
--------------------------------------------------------------------------------------------------------------------------
Shape of Train dataset matrix after one hot encoding is: (49041, 4)
Shape of CV dataset matrix after one hot encoding is: (24155, 4)
Shape of Test dataset matrix after one hot encoding is: (36052, 4)
```

1.5.2 Vectorizing Numerical features

In [0]:
```
price_data = Resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
X_Train = pd.merge(X_Train, price_data, on='id', how='left')
X_CV = pd.merge(X_CV, price_data, on='id', how='left')
X_Test = pd.merge(X_Test, price_data, on='id', how='left')
```

In [0]:
```
price_norm = Normalizer(norm='l2', copy=False)
price_norm.fit(X_Train['price'].values.reshape(1,-1))

p=price_norm.transform(X_Train['price'].values.reshape(1,-1))
price_norm.transform(X_CV['price'].values.reshape(1,-1))
price_norm.transform(X_Test['price'].values.reshape(1,-1))
price_norm_Train = (X_Train['price'].values.reshape(-1,1))
price_norm_CV = (X_CV['price'].values.reshape(-1,1))
price_norm_Test = (X_Test['price'].values.reshape(-1,1))
print("-"*120)
print('Shape of Train normalized price dataset matrix after one hot encoding is: {0}'.format(price_norm_Train.shape))
print('Shape of CV normalized price dataset matrix after one hot encoding is: {0}'.format(price_norm_CV.shape))
print('Shape of Test normalized price dataset matrix after one hot encoding is: {0}'.format(price_norm_Test.shape))
```

```
--------------------------------------------------------------------------------------------------------------------------
Shape of Train normalized price dataset matrix after one hot encoding is: (49041, 1)
Shape of CV normalized price dataset matrix after one hot encoding is: (24155, 1)
Shape of Test normalized price dataset matrix after one hot encoding is: (36052, 1)
```

In [0]:

```
quantity_norm = Normalizer(norm='l2', copy=False)
quantity_norm.fit(X_Train['quantity'].values.reshape(1,-1))

quantity_norm.transform(X_Train['quantity'].values.reshape(1,-1))
quantity_norm.transform(X_CV['quantity'].values.reshape(1,-1))
quantity_norm.transform(X_Test['quantity'].values.reshape(1,-1))
quantity_norm_Train = quantity_norm.transform(X_Train['quantity'].values.reshape(-1,1))
quantity_norm_CV = quantity_norm.transform(X_CV['quantity'].values.reshape(-1,1))
quantity_norm_Test = quantity_norm.transform(X_Test['quantity'].values.reshape(-1,1))
print("-"*120)
print('Shape of Train normalized quantity dataset matrix after one hot encoding is: {0}'.format(quantity_norm_Train.shape))
print('Shape of CV normalized quantity dataset matrix after one hot encoding is: {0}'.format(quantity_norm_CV.shape))
print('Shape of Test normalized quantity dataset matrix after one hot encoding is: {0}'.format(quantity_norm_Test.shape))
```

```
------------------------------------------------------------------------------------------------------------------------
Shape of Train normalized quantity dataset matrix after one hot encoding is: (49041, 1)
Shape of CV normalized quantity dataset matrix after one hot encoding is: (24155, 1)
Shape of Test normalized quantity dataset matrix after one hot encoding is: (36052, 1)
```

In [0]:

```
teacher_prev_post_norm = Normalizer(norm='l2', copy=False)
teacher_prev_post_norm.fit(X_Train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))

teacher_prev_post_norm.transform(X_Train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
teacher_prev_post_norm.transform(X_CV['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
teacher_prev_post_norm.transform(X_Test['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
teacher_prev_post_norm_Train = teacher_prev_post_norm.transform(X_Train['teacher_number_of_previously_posted_projects'].values
teacher_prev_post_norm_CV = teacher_prev_post_norm.transform(X_CV['teacher_number_of_previously_posted_projects'].values.reshap
teacher_prev_post_norm_Test = teacher_prev_post_norm.transform(X_Test['teacher_number_of_previously_posted_projects'].values.re
print("-"*120)
print('Shape of Train normalized previously posted project dataset matrix after one hot encoding is: {0}'.format(teacher_prev_p
print('Shape of CV normalized previously posted project dataset matrix after one hot encoding is: {0}'.format(teacher_prev_post
print('Shape of Test normalized previously posted project dataset matrix after one hot encoding is: {0}'.format(teacher_prev_p
```

```
------------------------------------------------------------------------------------------------------------------------
Shape of Train normalized previously posted project dataset matrix after one hot encoding is: (49041, 1)
Shape of CV normalized previously posted project dataset matrix after one hot encoding is: (24155, 1)
Shape of Test normalized previously posted project dataset matrix after one hot encoding is: (36052, 1)
```

In [0]:
```
title_norm = Normalizer(norm='l2', copy=False)
title_norm.fit(X_Train['word_count_title_Train'].values.reshape(1,-1))
title_norm.transform(X_Train['word_count_title_Train'].values.reshape(1,-1))
title_norm.transform(X_CV['word_count_title_CV'].values.reshape(1,-1))
title_norm.transform(X_Test['word_count_title_Test'].values.reshape(1,-1))
word_count_title_Train = title_norm.transform(X_Train['word_count_title_Train'].values.reshape(-1,1))
word_count_title_CV = title_norm.transform(X_CV['word_count_title_CV'].values.reshape(-1,1))
word_count_title_Test = title_norm.transform(X_Test['word_count_title_Test'].values.reshape(-1,1))
print("-"*120)
print('Shape of Train normalized title dataset matrix after one hot encoding is: {0}'.format(word_count_title_Train.shape))
print('Shape of CV normalized title dataset matrix after one hot encoding is: {0}'.format(word_count_title_CV.shape))
print('Shape of Test normalized title dataset matrix after one hot encoding is: {0}'.format(word_count_title_Test.shape))
```

```
------------------------------------------------------------------------------------------------------------------------
Shape of Train normalized title dataset matrix after one hot encoding is: (49041, 1)
Shape of CV normalized title dataset matrix after one hot encoding is: (24155, 1)
Shape of Test normalized title dataset matrix after one hot encoding is: (36052, 1)
```

In [0]:
```
essay_norm = Normalizer(norm='l2', copy=False)
essay_norm.fit(X_Train['word_count_essay_Train'].values.reshape(1,-1))
essay_norm.transform(X_Train['word_count_essay_Train'].values.reshape(1,-1))
essay_norm.transform(X_CV['word_count_essay_CV'].values.reshape(1,-1))
essay_norm.transform(X_Test['word_count_essay_Test'].values.reshape(1,-1))
word_count_essay_Train = essay_norm.transform(X_Train['word_count_essay_Train'].values.reshape(-1,1))
word_count_essay_CV = essay_norm.transform(X_CV['word_count_essay_CV'].values.reshape(-1,1))
word_count_essay_Test = essay_norm.transform(X_Test['word_count_essay_Test'].values.reshape(-1,1))
print("-"*120)
print('Shape of Train normalized title dataset matrix after one hot encoding is: {0}'.format(word_count_essay_Train.shape))
print('Shape of CV normalized title dataset matrix after one hot encoding is: {0}'.format(word_count_essay_CV.shape))
print('Shape of Test normalized title dataset matrix after one hot encoding is: {0}'.format(word_count_essay_Test.shape))
```

```
------------------------------------------------------------------------------------------------------------------------
Shape of Train normalized title dataset matrix after one hot encoding is: (49041, 1)
Shape of CV normalized title dataset matrix after one hot encoding is: (24155, 1)
Shape of Test normalized title dataset matrix after one hot encoding is: (36052, 1)
```

1.5.3 Vectorizing Text data

1.5.3.1 Bag of words

In [0]:
```
# We are considering only the words which appeared in at least 10 documents(rows or projects).
vectorizer_essays_bow = CountVectorizer(min_df=10)
text_bow_Train = vectorizer_essays_bow.fit_transform(preprocessed_essays_Train)
text_bow_CV = vectorizer_essays_bow.transform(preprocessed_essays_CV)
text_bow_Test = vectorizer_essays_bow.transform(preprocessed_essays_Test)
print("-"*120)
print("Applying Bag Of Words for Text Data")
print("-"*120)
print('Shape of Train dataset matrix after one hot encoding is: {0}'.format(text_bow_Train.shape))
print('Shape of CV dataset matrix after one hot encoding is: {0}'.format(text_bow_CV.shape))
print('Shape of Test dataset matrix after one hot encoding is: {0}'.format(text_bow_Test.shape))
```

```
------------------------------------------------------------------------------------------------------------------------
Applying Bag Of Words for Text Data
------------------------------------------------------------------------------------------------------------------------
Shape of Train dataset matrix after one hot encoding is: (49041, 12097)
Shape of CV dataset matrix after one hot encoding is: (24155, 12097)
Shape of Test dataset matrix after one hot encoding is: (36052, 12097)
```

Bag of Words for Project Title

In [0]:
```
# you can vectorize the title also
# before you vectorize the title make sure you preprocess it
vectorizer_titles_bow = CountVectorizer(min_df=10)
title_bow_Train = vectorizer_titles_bow.fit_transform(preprocessed_titles_Train)
title_bow_CV = vectorizer_titles_bow.transform(preprocessed_titles_CV)
title_bow_Test = vectorizer_titles_bow.transform(preprocessed_titles_Test)
print("-"*120)
print("Applying Bag Of Words for Project Title Data")
print("-"*120)
print('Shape of Train dataset matrix after one hot encoding is: {0}'.format(title_bow_Train.shape))
print('Shape of CV dataset matrix after one hot encoding is: {0}'.format(title_bow_CV.shape))
print('Shape of Test dataset matrix after one hot encoding is: {0}'.format(title_bow_Test.shape))

```

```
------------------------------------------------------------------------------------------------------------------------
Applying Bag Of Words for Project Title Data
------------------------------------------------------------------------------------------------------------------------
Shape of Train dataset matrix after one hot encoding is: (49041, 2083)
Shape of CV dataset matrix after one hot encoding is: (24155, 2083)
Shape of Test dataset matrix after one hot encoding is: (36052, 2083)
```

1.5.2.2 TFIDF vectorizer

In [0]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_essays_tfidf = TfidfVectorizer(min_df=10)
text_tfidf_Train = vectorizer_essays_tfidf.fit_transform(preprocessed_essays_Train)
text_tfidf_CV = vectorizer_essays_tfidf.transform(preprocessed_essays_CV)
text_tfidf_Test = vectorizer_essays_tfidf.transform(preprocessed_essays_Test)
print("-"*120)
print("Applying TFIDF for Text Data")
print("-"*120)
print('Shape of Train dataset matrix after one hot encoding is: {0}'.format(text_tfidf_Train.shape))
print('Shape of CV dataset matrix after one hot encoding is: {0}'.format(text_tfidf_CV.shape))
print('Shape of Test dataset matrix after one hot encoding is: {0}'.format(text_tfidf_Test.shape))
```

```
------------------------------------------------------------------------------------------------------------------------
Applying TFIDF for Text Data
------------------------------------------------------------------------------------------------------------------------
Shape of Train dataset matrix after one hot encoding is: (49041, 12097)
Shape of CV dataset matrix after one hot encoding is: (24155, 12097)
Shape of Test dataset matrix after one hot encoding is: (36052, 12097)
```

TFIDF vectorizer for Project Title

In [0]:
```
vectorizer_titles_tfidf = TfidfVectorizer(min_df=10)
title_tfidf_Train = vectorizer_titles_tfidf.fit_transform(preprocessed_titles_Train)
title_tfidf_CV = vectorizer_titles_tfidf.transform(preprocessed_titles_CV)
title_tfidf_Test = vectorizer_titles_tfidf.transform(preprocessed_titles_Test)
print("-"*120)
print("Applying TFIDF for Project Title")
print("-"*120)
print('Shape of Train dataset matrix after one hot encoding is: {0}'.format(title_tfidf_Train.shape))
print('Shape of CV dataset matrix after one hot encoding is: {0}'.format(title_tfidf_CV.shape))
print('Shape of Test dataset matrix after one hot encoding is: {0}'.format(title_tfidf_Test.shape))
```

```
------------------------------------------------------------------------------------------------------------------------
Applying TFIDF for Project Title
------------------------------------------------------------------------------------------------------------------------
Shape of Train dataset matrix after one hot encoding is: (49041, 2083)
Shape of CV dataset matrix after one hot encoding is: (24155, 2083)
Shape of Test dataset matrix after one hot encoding is: (36052, 2083)
```

1.5.2.3 Using Pretrained Models: Avg W2V

In [0]:

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

In [0]:

```
# average Word2Vec
# compute average word2vec for each review.
avg_w2v_vectors_Train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays_Train): # 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_Train.append(vector)
#------------------------------------------------------------------------------------------

avg_w2v_vectors_CV = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays_CV): # 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_CV.append(vector)
#------------------------------------------------------------------------------------------

avg_w2v_vectors_Test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays_Test): # 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_Test.append(vector)

print(len(avg_w2v_vectors_Test))
print(len(avg_w2v_vectors_Test[1]))
```

```
100%|██████████| 49041/49041 [00:12<00:00, 3941.77it/s]
100%|██████████| 24155/24155 [00:06<00:00, 3925.89it/s]
```

```
100%|██████████| 36052/36052 [00:09<00:00, 3947.34it/s]

36052
300
```

AVG W2V on project_title

In [0]:

```
# Similarly you can vectorize for title also
# compute average word2vec for each title.
avg_w2v_vectors_title_Train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_titles_Train): # for each review/sentence
    vector_title = np.zeros(300) # as word vectors are of zero length
    cnt_title_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_title += model[word]
            cnt_title_words += 1
    if cnt_title_words != 0:
        vector_title /= cnt_title_words
    avg_w2v_vectors_title_Train.append(vector_title)


#-------------------------------------------------------------------------------------------
avg_w2v_vectors_title_CV = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_titles_CV): # for each review/sentence
    vector_title = np.zeros(300) # as word vectors are of zero length
    cnt_title_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_title += model[word]
            cnt_title_words += 1
    if cnt_title_words != 0:
        vector_title /= cnt_title_words
    avg_w2v_vectors_title_CV.append(vector_title)

#-------------------------------------------------------------------------------------------
avg_w2v_vectors_title_Test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_titles_Test): # for each review/sentence
    vector_title = np.zeros(300) # as word vectors are of zero length
    cnt_title_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_title += model[word]
            cnt_title_words += 1
    if cnt_title_words != 0:
        vector_title /= cnt_title_words
    avg_w2v_vectors_title_Test.append(vector_title)

print(len(avg_w2v_vectors_title_Test))
print(len(avg_w2v_vectors_title_Test[0]))
```

```
100%|██████████| 49041/49041 [00:00<00:00, 65685.20it/s]
```

```
100%|██████████| 24155/24155 [00:00<00:00, 66048.26it/s]
100%|██████████| 36052/36052 [00:00<00:00, 66043.95it/s]

36052
300
```

1.5.2.3 Using Pretrained Models: TFIDF weighted W2V

In [0]:

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

In [0]:

```
# average Word2Vec
# compute average word2vec for each review.
tfidf_w2v_vectors_Train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays_Train): # 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_essays):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split()))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_Train.append(vector)

#------------------------------------------------------------------------------------------
tfidf_w2v_vectors_CV = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays_CV): # 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_essays):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split()))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_vectors_CV.append(vector)
#------------------------------------------------------------------------------------------
tfidf_w2v_vectors_Test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_essays_Test): # 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_essays):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split()))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
```

```
        vector /= tf_idf_weight
    tfidf_w2v_vectors_Test.append(vector)

print(len(tfidf_w2v_vectors_Test))
print(len(tfidf_w2v_vectors_Test[0]))

```

```
100%|██████████| 49041/49041 [01:21<00:00, 599.75it/s]
100%|██████████| 24155/24155 [00:40<00:00, 597.22it/s]
100%|██████████| 36052/36052 [01:00<00:00, 596.10it/s]

36052
300
```

Using Pretrained Models: TFIDF weighted W2V on project_title

In [0]:

```
# Similarly you can vectorize for title also
tfidf_model_title = TfidfVectorizer()
tfidf_model_title.fit(preprocessed_titles_Train)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model_title.get_feature_names(), list(tfidf_model_title.idf_)))
tfidf_words_title = set(tfidf_model_title.get_feature_names())

# compute tfidf word2vec for each title.
tfidf_w2v_vectors_title_Train = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_titles_Train): # for each review/sentence
    vector_title = 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_title):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split()))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector_title += (vector_title * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector_title /= tf_idf_weight
    tfidf_w2v_vectors_title_Train.append(vector_title)
#------------------------------------------------------------------------------------------------------------------------

tfidf_w2v_vectors_title_CV = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_titles_CV): # for each review/sentence
    vector_title = 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_title):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split()))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector_title += (vector_title * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector_title /= tf_idf_weight
    tfidf_w2v_vectors_title_CV.append(vector_title)
#------------------------------------------------------------------------------------------------------------------------


tfidf_w2v_vectors_title_Test = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(preprocessed_titles_Test): # for each review/sentence
    vector_title = 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_title):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split()))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word
            vector_title += (vector_title * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector_title /= tf_idf_weight
    tfidf_w2v_vectors_title_Test.append(vector_title)

print(len(tfidf_w2v_vectors_title_Test))
print(len(tfidf_w2v_vectors_title_Test[0]))
```

```
100%|██████████| 49041/49041 [00:01<00:00, 31119.68it/s]
100%|██████████| 24155/24155 [00:00<00:00, 29048.77it/s]
100%|██████████| 36052/36052 [00:01<00:00, 32121.15it/s]

36052
300
```

Calculating the sentiment score's of each of the essay

In [0]:

```
sid = SentimentIntensityAnalyzer()

essays = X_Train['essay']
essays_sentiment_TR_P = []
essays_sentiment_TR_N = []
essays_sentiment_TR_NE = []
essays_sentiment_TR_C = []
for essay in tqdm(essays):
    res = sid.polarity_scores(essay)
    essays_sentiment_TR_P.append(res['pos'])
    essays_sentiment_TR_N.append(res['neg'])
    essays_sentiment_TR_NE.append(res['neu'])
    essays_sentiment_TR_C.append(res['compound'])
X_Train['sentiment_essay_TR_P'] = essays_sentiment_TR_P
X_Train['sentiment_essay_TR_N'] = essays_sentiment_TR_N
X_Train['sentiment_essay_TR_NE'] = essays_sentiment_TR_NE
X_Train['sentiment_essay_TR_C'] = essays_sentiment_TR_C

essays = X_CV['essay']
essays_sentiment_CV_P = []
essays_sentiment_CV_N = []
essays_sentiment_CV_NE = []
essays_sentiment_CV_C = []
for essay in tqdm(essays):
    res = sid.polarity_scores(essay)
    essays_sentiment_CV_P.append(res['pos'])
    essays_sentiment_CV_N.append(res['neg'])
    essays_sentiment_CV_NE.append(res['neu'])
    essays_sentiment_CV_C.append(res['compound'])
X_CV['sentiment_essay_CV_P'] = essays_sentiment_CV_P
X_CV['sentiment_essay_CV_N'] = essays_sentiment_CV_N
X_CV['sentiment_essay_CV_NE'] = essays_sentiment_CV_NE
X_CV['sentiment_essay_CV_C'] = essays_sentiment_CV_C

essays = X_Test['essay']
essays_sentiment_TS_P = []
essays_sentiment_TS_N = []
essays_sentiment_TS_NE = []
essays_sentiment_TS_C = []
for essay in tqdm(essays):
    res = sid.polarity_scores(essay)
    essays_sentiment_TS_P.append(res['pos'])
    essays_sentiment_TS_N.append(res['neg'])
    essays_sentiment_TS_NE.append(res['neu'])
```

```
    essays_sentiment_TS_C.append(res['compound'])
X_Test['sentiment_essay_TS_P'] = essays_sentiment_TS_P
X_Test['sentiment_essay_TS_N'] = essays_sentiment_TS_N
X_Test['sentiment_essay_TS_NE'] = essays_sentiment_TS_NE
X_Test['sentiment_essay_TS_C'] = essays_sentiment_TS_C

sentiment_norm_P = Normalizer(norm='l2', copy=False)
sentiment_norm_N = Normalizer(norm='l2', copy=False)
sentiment_norm_NE = Normalizer(norm='l2', copy=False)
sentiment_norm_C = Normalizer(norm='l2', copy=False)


sentiment_norm_P.fit(X_Train['sentiment_essay_TR_P'].values.reshape(1,-1))
sentiment_norm_N.fit(X_Train['sentiment_essay_TR_N'].values.reshape(1,-1))
sentiment_norm_NE.fit(X_Train['sentiment_essay_TR_NE'].values.reshape(1,-1))
sentiment_norm_C.fit(X_Train['sentiment_essay_TR_C'].values.reshape(1,-1))

sentiment_Train_P = sentiment_norm_P.transform(X_Train['sentiment_essay_TR_P'].values.reshape(1,-1))
sentiment_CV_P = sentiment_norm_P.transform(X_CV['sentiment_essay_CV_P'].values.reshape(1,-1))
sentiment_Test_P = sentiment_norm_P.transform(X_Test['sentiment_essay_TS_P'].values.reshape(1,-1))
sentiment_Train_N = sentiment_norm_N.transform(X_Train['sentiment_essay_TR_N'].values.reshape(1,-1))
sentiment_CV_N = sentiment_norm_N.transform(X_CV['sentiment_essay_CV_N'].values.reshape(1,-1))
sentiment_Test_N = sentiment_norm_N.transform(X_Test['sentiment_essay_TS_N'].values.reshape(1,-1))
sentiment_Train_NE = sentiment_norm_NE.transform(X_Train['sentiment_essay_TR_NE'].values.reshape(1,-1))
sentiment_CV_NE = sentiment_norm_NE.transform(X_CV['sentiment_essay_CV_NE'].values.reshape(1,-1))
sentiment_Test_NE = sentiment_norm_NE.transform(X_Test['sentiment_essay_TS_NE'].values.reshape(1,-1))
sentiment_Train_C = sentiment_norm_C.transform(X_Train['sentiment_essay_TR_C'].values.reshape(1,-1))
sentiment_CV_C = sentiment_norm_C.transform(X_CV['sentiment_essay_CV_C'].values.reshape(1,-1))
sentiment_Test_C = sentiment_norm_C.transform(X_Test['sentiment_essay_TS_C'].values.reshape(1,-1))

sentiment_Train_P = (X_Train['sentiment_essay_TR_P'].values.reshape(-1,1))
sentiment_CV_P = (X_CV['sentiment_essay_CV_P'].values.reshape(-1,1))
sentiment_Test_P = (X_Test['sentiment_essay_TS_P'].values.reshape(-1,1))
sentiment_Train_N = (X_Train['sentiment_essay_TR_N'].values.reshape(-1,1))
sentiment_CV_N = (X_CV['sentiment_essay_CV_N'].values.reshape(-1,1))
sentiment_Test_N = (X_Test['sentiment_essay_TS_N'].values.reshape(-1,1))
sentiment_Train_NE = (X_Train['sentiment_essay_TR_NE'].values.reshape(-1,1))
sentiment_CV_NE = (X_CV['sentiment_essay_CV_NE'].values.reshape(-1,1))
sentiment_Test_NE = (X_Test['sentiment_essay_TS_NE'].values.reshape(-1,1))
sentiment_Train_C = (X_Train['sentiment_essay_TR_C'].values.reshape(-1,1))
sentiment_CV_C = (X_CV['sentiment_essay_CV_C'].values.reshape(-1,1))
sentiment_Test_C = (X_Test['sentiment_essay_TS_C'].values.reshape(-1,1))


print("Shape of sentiment Train matrix after one hot encodig ",sentiment_Train_P.shape)
print("Shape of sentiment CV matrix after one hot encodig ",sentiment_CV_P.shape)
```

```
print("Shape of sentiment Test matrix after one hot encodig ",sentiment_Test_P.shape)
print("Shape of sentiment Train matrix after one hot encodig ",sentiment_Train_N.shape)
print("Shape of sentiment CV matrix after one hot encodig ",sentiment_CV_N.shape)
print("Shape of sentiment Test matrix after one hot encodig ",sentiment_Test_N.shape)
print("Shape of sentiment Train matrix after one hot encodig ",sentiment_Train_NE.shape)
print("Shape of sentiment CV matrix after one hot encodig ",sentiment_CV_NE.shape)
print("Shape of sentiment Test matrix after one hot encodig ",sentiment_Test_NE.shape)
print("Shape of sentiment Train matrix after one hot encodig ",sentiment_Train_C.shape)
print("Shape of sentiment CV matrix after one hot encodig ",sentiment_CV_C.shape)
print("Shape of sentiment Test matrix after one hot encodig ",sentiment_Test_C.shape)
```

```
100%|██████████| 49041/49041 [02:03<00:00, 397.69it/s]
100%|██████████| 24155/24155 [01:00<00:00, 396.71it/s]
100%|██████████| 36052/36052 [01:31<00:00, 395.80it/s]

Shape of sentiment Train matrix after one hot encodig  (49041, 1)
Shape of sentiment CV matrix after one hot encodig  (24155, 1)
Shape of sentiment Test matrix after one hot encodig  (36052, 1)
Shape of sentiment Train matrix after one hot encodig  (49041, 1)
Shape of sentiment CV matrix after one hot encodig  (24155, 1)
Shape of sentiment Test matrix after one hot encodig  (36052, 1)
Shape of sentiment Train matrix after one hot encodig  (49041, 1)
Shape of sentiment CV matrix after one hot encodig  (24155, 1)
Shape of sentiment Test matrix after one hot encodig  (36052, 1)
Shape of sentiment Train matrix after one hot encodig  (49041, 1)
Shape of sentiment CV matrix after one hot encodig  (24155, 1)
Shape of sentiment Test matrix after one hot encodig  (36052, 1)
```

Assignment 7: SVM

1. **[Task-1] Apply Support Vector Machines(SGDClassifier with hinge loss: Linear SVM) on these feature sets**

   - Set 1: categorical, numerical features + project_title(BOW) + preprocessed_essay (BOW)
   - Set 2: categorical, numerical features + project_title(TFIDF)+ preprocessed_essay (TFIDF)
   - Set 3: categorical, numerical features + project_title(AVG W2V)+ preprocessed_essay (AVG W2V)
   - Set 4: categorical, numerical features + project_title(TFIDF W2V)+ preprocessed_essay (TFIDF W2V)

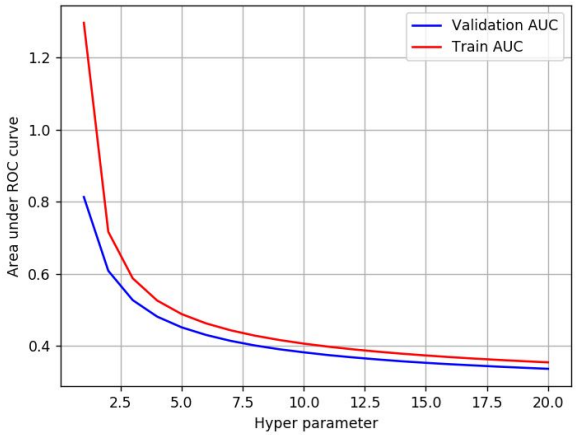
2. **The hyper paramter tuning (best alpha in range [10^-4 to 10^4], and the best penalty among 'l1', 'l2')**

   - Find the best hyper parameter which will give the maximum [AUC (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/)](https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) value
   - Find the best hyper paramter using k-fold cross validation or simple cross validation data

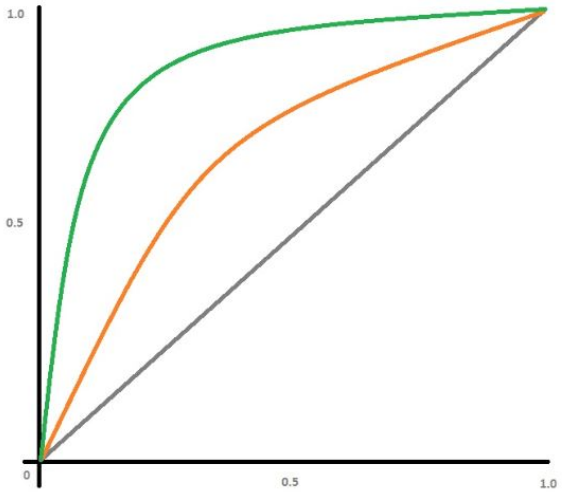
- Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning

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 (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/)](https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points. Please visualize your confusion matrices using [seaborn heatmaps.](https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/)

	Predicted: NO	Predicted: YES
Actual: NO	TN = ??	FP = ??
Actual: YES	FN = ??	TP = ??

(https://seaborn.pydata.org/generated/seaborn.heatmap.html)

4. **[Task-2] Apply the Support Vector Machines on these features by finding the best hyper paramter as suggested in step 2 and step 3**

- 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
  - **Apply TruncatedSVD (http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html) on TfidfVectorizer (https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html) of essay text, choose the number of components (`n_components`) using elbow method (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/pca-code-example-using-non-visualization/)** : numerical data

- **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 (http://zetcode.com/python/prettytable/)

Vectorizer	Model	Hyper parameter	AUC
BOW	Brute	7	0.78
TFIDF	Brute	12	0.79
W2V	Brute	10	0.78
TFIDFW2V	Brute	6	0.78

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. (https://soundcloud.com/applied-ai-course/leakage-bow-and-tfidf)

2.4 Appling Support Vector Machines on different kind of featurization as mentioned in the instructions

Apply Support Vector Machines 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

1.5.4 Merging all the above features

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

In [0]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
BOW_Train = hstack((categories_one_hot_Train,sub_categories_one_hot_Train,school_one_hot_Train,grade_one_hot_Train,prefix_one_h
print(BOW_Train.shape)
TFIDF_Train = hstack((categories_one_hot_Train,sub_categories_one_hot_Train,school_one_hot_Train,grade_one_hot_Train,prefix_one
print(TFIDF_Train.shape)
AVG_W2V_Train = hstack((categories_one_hot_Train,sub_categories_one_hot_Train,school_one_hot_Train,grade_one_hot_Train,prefix_c
print(AVG_W2V_Train.shape)
TFIDF_W2V_Train = hstack((categories_one_hot_Train,sub_categories_one_hot_Train,school_one_hot_Train,grade_one_hot_Train,prefix
print(TFIDF_W2V_Train.shape)
#FiveF_Train = hstack((categories_one_hot_Train,sub_categories_one_hot_Train,school_one_hot_Train,grade_one_hot_Train,prefix_or
#print(FiveF_Train.shape)
```

```
(49041, 14289)
(49041, 14289)
(49041, 709)
(49041, 709)
```

In [0]:

```
BOW_CV = hstack((categories_one_hot_CV,sub_categories_one_hot_CV,school_one_hot_CV,grade_one_hot_CV,prefix_one_hot_CV,text_bow_
print(BOW_CV.shape)
TFIDF_CV = hstack((categories_one_hot_CV,sub_categories_one_hot_CV,school_one_hot_CV,grade_one_hot_CV,prefix_one_hot_CV,text_tf
print(TFIDF_CV.shape)
AVG_W2V_CV = hstack((categories_one_hot_CV,sub_categories_one_hot_CV,school_one_hot_CV,grade_one_hot_CV,prefix_one_hot_CV,avg_w
print(AVG_W2V_CV.shape)
TFIDF_W2V_CV = hstack((categories_one_hot_CV,sub_categories_one_hot_CV,school_one_hot_CV,grade_one_hot_CV,prefix_one_hot_CV,tfi
print(TFIDF_W2V_CV.shape)
#FiveF_CV = hstack((categories_one_hot_CV,sub_categories_one_hot_CV,school_one_hot_CV,grade_one_hot_CV,prefix_one_hot_CV, price
#print(FiveF_CV.shape)
```

```
(24155, 14289)
(24155, 14289)
(24155, 709)
(24155, 709)
```

In [0]:

```
BOW_Test = hstack((categories_one_hot_Test,sub_categories_one_hot_Test,school_one_hot_Test,grade_one_hot_Test,prefix_one_hot_Te
print(BOW_Test.shape)
TFIDF_Test = hstack((categories_one_hot_Test,sub_categories_one_hot_Test,school_one_hot_Test,grade_one_hot_Test,prefix_one_hot_
print(TFIDF_Test.shape)
AVG_W2V_Test = hstack((categories_one_hot_Test,sub_categories_one_hot_Test,school_one_hot_Test,grade_one_hot_Test,prefix_one_ho
print(AVG_W2V_Test.shape)
TFIDF_W2V_Test = hstack((categories_one_hot_Test,sub_categories_one_hot_Test,school_one_hot_Test,grade_one_hot_Test,prefix_one_
print(TFIDF_W2V_Test.shape)
#FiveF_Test = hstack((categories_one_hot_Test,sub_categories_one_hot_Test,school_one_hot_Test,grade_one_hot_Test,prefix_one_hot
#print(FiveF_Test.shape)
```

```
(36052, 14289)
(36052, 14289)
(36052, 709)
(36052, 709)
```

In [0]:

```
import pdb

def batch_predict(clf, data):
    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.decision_function(data[i:i+1000]))
    y_data_pred.extend(clf.decision_function(data[tr_loop:]))
    #pdb.set_trace()
    return y_data_pred
```

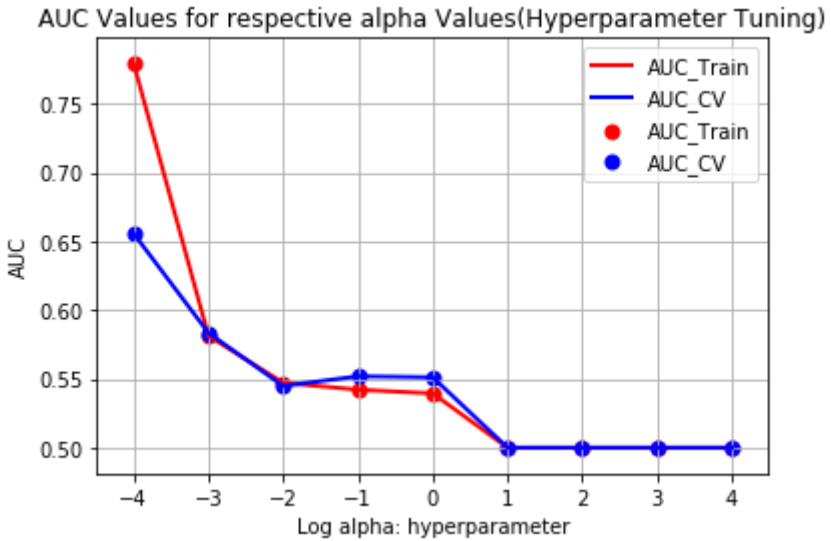
Applying Support Vector Machines(SVM) on BOW with L1 regularization, SET 1

In [0]:

```
%%time

BOW_TR_CSR = BOW_Train.tocsr()
BOW_CV_CSR = BOW_CV.tocsr()
BOW_TS_CSR = BOW_Test.tocsr()

C_Para = [1e-4,1e-3,1e-2,1e-1,1,1e+1,1e+2,1e+3,1e+4]
L_C_Para= []
ACCV = []
AUC_TR = []
AUC_CV = []

for i in tqdm(C_Para):
    SGC = SGDClassifier(loss = 'hinge',penalty='l1', n_jobs=-1,alpha=i)
    SGC.fit(BOW_Train, Y_Train)
    pred = SGC.predict(BOW_CV)
    acc = accuracy_score(Y_CV, pred, normalize=True) * float(100)
    ACCV.append(acc)

    Train_pred = batch_predict(SGC, BOW_TR_CSR)
    a_fpr_train,a_tpr_train,c = roc_curve(Y_Train, Train_pred)
    AUC_TR.append(auc(a_fpr_train, a_tpr_train))

    CV_pred = batch_predict(SGC, BOW_CV_CSR)
    a_fpr_cv,a_tpr_cv,c = roc_curve(Y_CV, CV_pred)

    AUC_CV.append(auc(a_fpr_cv, a_tpr_cv))


for av in tqdm(C_Para):
    b = np.log10(av)
    L_C_Para.append(b)

# Performance of model on Train data and Test data for each hyper parameter.
plt.plot(L_C_Para, AUC_TR, label='AUC_Train',color='red',linewidth=2)
plt.scatter(L_C_Para, AUC_TR, label='AUC_Train',color='red',linewidth=2)
plt.gca()
plt.plot(L_C_Para, AUC_CV, label='AUC_CV',color='blue',linewidth=2)
plt.scatter(L_C_Para, AUC_CV, label='AUC_CV',color='blue',linewidth=2)
plt.gca()
plt.legend()
plt.xlabel("Log alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC Values for respective alpha Values(Hyperparameter Tuning)")
plt.grid()
```

```
plt.show()
```

```
100%|██████████| 9/9 [00:20<00:00,  1.53s/it]
100%|██████████| 9/9 [00:00<00:00, 32044.77it/s]
```




```
CPU times: user 26.3 s, sys: 30.5 s, total: 56.8 s
Wall time: 21.3 s
```

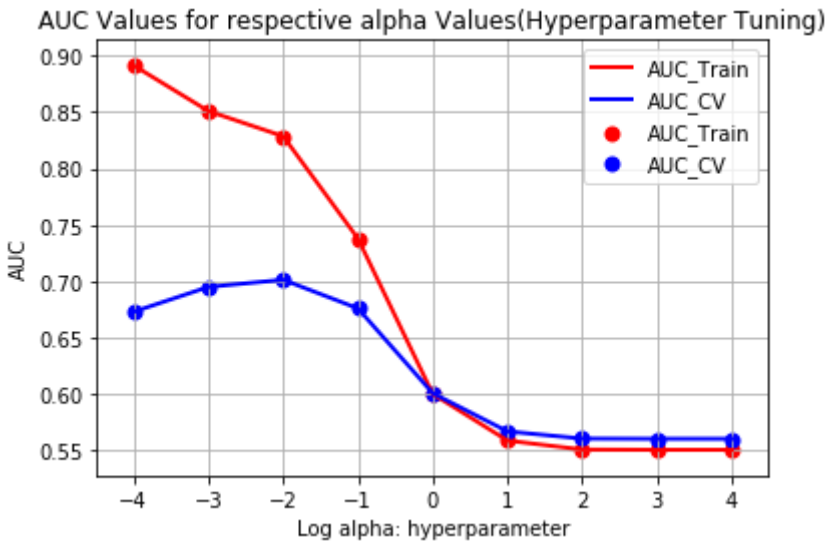
Applying Support Vector Machines(SVM) on BOW with L2 regularization, SET 1

In [0]:

```
%%time

BOW_TR_CSR = BOW_Train.tocsr()
BOW_CV_CSR = BOW_CV.tocsr()
BOW_TS_CSR = BOW_Test.tocsr()

C_Para = [1e-4,1e-3,1e-2,1e-1,1,1e+1,1e+2,1e+3,1e+4]
L_C_Para= []
ACCV = []
AUC_TR = []
AUC_CV = []

for i in tqdm(C_Para):

    SGC = SGDClassifier(loss = 'hinge',penalty='l2', n_jobs=-1,alpha=i)
    SGC.fit(BOW_Train, Y_Train)
    pred = SGC.predict(BOW_CV)
    acc = accuracy_score(Y_CV, pred, normalize=True) * float(100)
    ACCV.append(acc)

    Train_pred = batch_predict(SGC, BOW_TR_CSR)
    a_fpr_train,a_tpr_train,c = roc_curve(Y_Train, Train_pred)
    AUC_TR.append(auc(a_fpr_train, a_tpr_train))

    CV_pred = batch_predict(SGC, BOW_CV_CSR)
    a_fpr_cv,a_tpr_cv,c = roc_curve(Y_CV, CV_pred)

    AUC_CV.append(auc(a_fpr_cv, a_tpr_cv))


for av in tqdm(C_Para):
    b = np.log10(av)
    L_C_Para.append(b)

# Performance of model on Train data and Test data for each hyper parameter.
plt.plot(L_C_Para, AUC_TR, label='AUC_Train',color='red',linewidth=2)
plt.scatter(L_C_Para, AUC_TR, label='AUC_Train',color='red',linewidth=2)
plt.gca()
plt.plot(L_C_Para, AUC_CV, label='AUC_CV',color='blue',linewidth=2)
plt.scatter(L_C_Para, AUC_CV, label='AUC_CV',color='blue',linewidth=2)
plt.gca()
plt.legend()
plt.xlabel("Log alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC Values for respective alpha Values(Hyperparameter Tuning)")
```

```
46 plt.grid()
47 plt.show()
```

```
100%|██████████| 9/9 [00:08<00:00,  1.25it/s]
100%|██████████| 9/9 [00:00<00:00, 43389.35it/s]
```




```
CPU times: user 13.8 s, sys: 28.3 s, total: 42.1 s
Wall time: 9.25 s
```

OBSERVATION:

1. α value has been choose based on the alpha which maximize the AUC CV score.
2. In this case **α = 0.01** is having the highest AUC CV score.
3. Hence **α = 0.01** is choosen as a best α value.

In [0]:

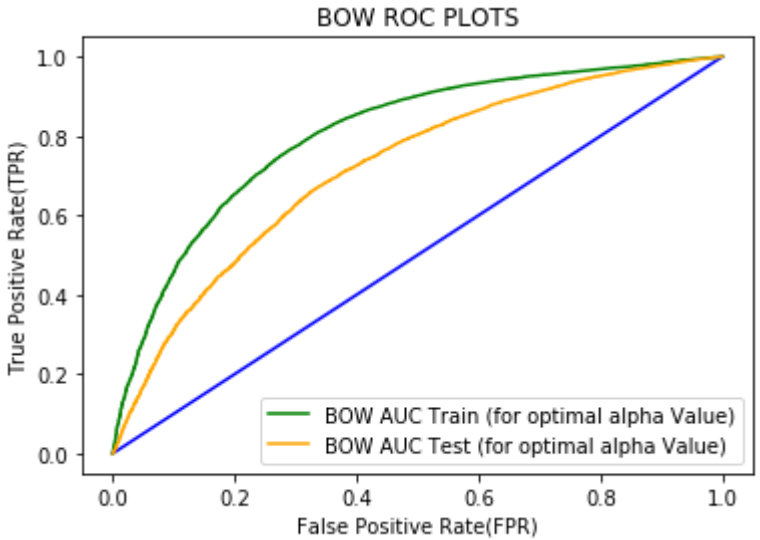
```
SGC_OPT= 0.01
BOW_opt = SGDClassifier(penalty='l2', alpha=SGC_OPT, class_weight='balanced', n_jobs=-1)
BOW_opt.fit(BOW_Train, Y_Train)
pred = BOW_opt.predict(BOW_Test)
acc = accuracy_score(Y_Test, pred, normalize=True) * float(100)
print('\nTest accuracy for alpha Value = {0} is {1}%'.format(SGC_OPT,acc))

a_fpr_train, a_tpr_train, thresholds = roc_curve(Y_Train, BOW_opt.decision_function(BOW_TR_CSR))
a_fpr_Test, a_tpr_Test, thresholds = roc_curve(Y_Test, BOW_opt.decision_function(BOW_TS_CSR))
```

```
Test accuracy for alpha Value = 0.01 is 66.8118273604793%
```

BOW ROC PLOT

In [0]:

```
%%time

#https://stackoverflow.com/questions/52910061/implementing-roc-curves-for-k-nn-machine-learning-algorithm-using-python-and-sci

plt.plot([0,1],[0,1],'k-', color='blue')
plt.plot(a_fpr_train, a_tpr_train, label="BOW AUC Train (for optimal alpha Value)", color='green')
plt.plot(a_fpr_Test, a_tpr_Test, label="BOW AUC Test (for optimal alpha Value)", color='orange')
plt.legend()
plt.ylabel("True Positive Rate(TPR)")
plt.xlabel("False Positive Rate(FPR)")
plt.title("BOW ROC PLOTS")
plt.show()
print("-"*120)
print("AUC Train (for optimal alpha Value) =", auc(a_fpr_train, a_tpr_train))
print("AUC Test (for optimal alpha Value) =", auc(a_fpr_Test, a_tpr_Test))
BOW_AOPT=SGC_OPT
BOW_AUC=round(auc(a_fpr_Test, a_tpr_Test)*100)
pred1 = BOW_opt.predict(BOW_Train)
pred2 = BOW_opt.predict(BOW_Test)

```

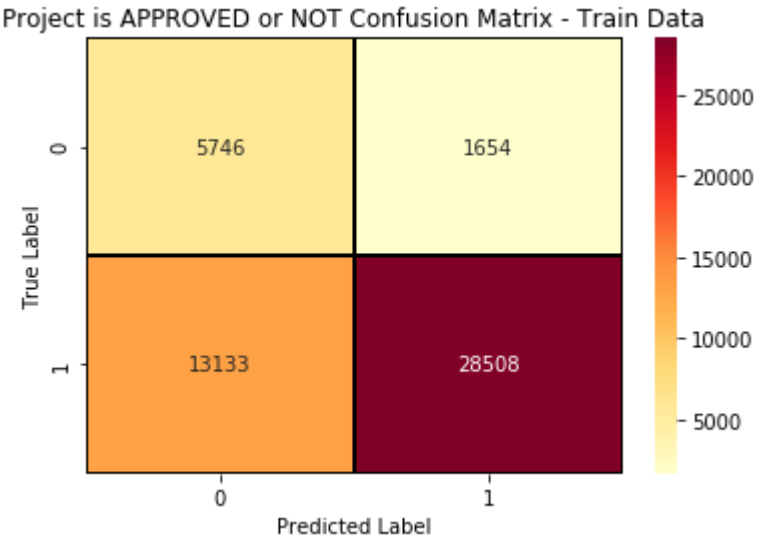



```
------------------------------------------------------------------------------------------------------------------------
AUC Train (for optimal alpha Value) = 0.8043252832934277
AUC Test (for optimal alpha Value) = 0.7185840057086738
CPU times: user 651 ms, sys: 7.82 ms, total: 659 ms
Wall time: 656 ms
```

BOW CONFUSION MATRIX

In [0]:
```
%%time
#https://seaborn.pydata.org/generated/seaborn.heatmap.html
#https://getaravind.com/blog/confusion-matrix-seaborn-heatmap/
%matplotlib inline
from sklearn.metrics import confusion_matrix
Train = confusion_matrix(Y_Train, pred1)
sns.heatmap(Train,annot=True,cbar=True,fmt='d',cmap='YlOrRd',linewidths=1,linecolor='black')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Project is APPROVED or NOT Confusion Matrix - Train Data')
```

```
CPU times: user 104 ms, sys: 34.1 ms, total: 139 ms
Wall time: 94.2 ms
```



OBSERVATION:

True Negative =5746; False Negative =13133; True Positive = 28508; False Positive = 1654

Accuracy (Overall, how often is the classifier correct) = 0.70

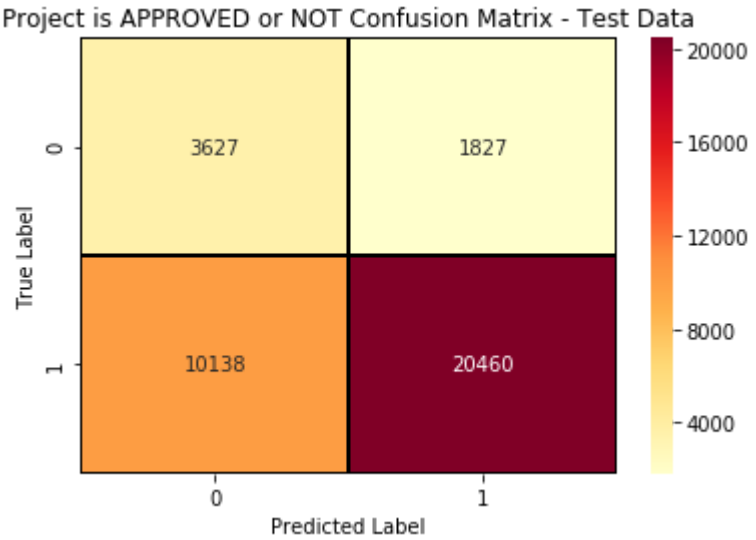
Precision(When it predicts yes, how often is it correct) =0.95

Misclassification (Overall, how often is it wrong) =0.31

In [0]:

```
%%time
#https://seaborn.pydata.org/generated/seaborn.heatmap.html
#https://getaravind.com/blog/confusion-matrix-seaborn-heatmap/
Test = confusion_matrix(Y_Test, pred2)
sns.heatmap(Test,annot=True,cbar=True,fmt='d',cmap='YlOrRd',linewidths=1,linecolor='black')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Project is APPROVED or NOT Confusion Matrix - Test Data')
```

```
CPU times: user 93.9 ms, sys: 37.3 ms, total: 131 ms
Wall time: 85.3 ms
```



OBSERVATION:

True Negative =3627; False Negative =10138; True Positive = 20460; False Positive = 1827

Accuracy (Overall, how often is the classifier correct) = 0.67

Precision(When it predicts yes, how often is it correct) =0.92

Misclassification (Overall, how often is it wrong) =0.34

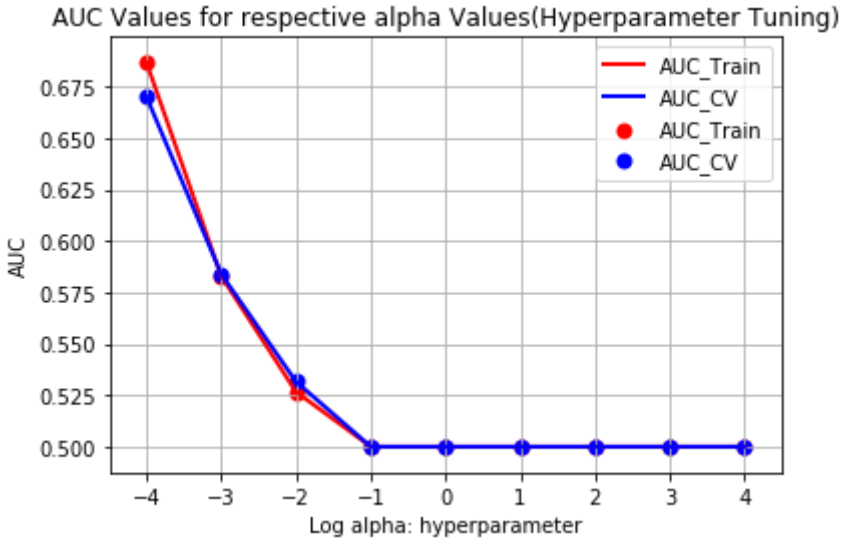
Applying Support Vector Machines(SVM) on TFIDF with L1 regularization, SET 2

In [0]:

```
%%time

TFIDF_TR_CSR = TFIDF_Train.tocsr()
TFIDF_CV_CSR = TFIDF_CV.tocsr()
TFIDF_TS_CSR = TFIDF_Test.tocsr()

C_Para = [1e-4,1e-3,1e-2,1e-1,1,1e+1,1e+2,1e+3,1e+4]
L_C_Para= []
ACCV = []
AUC_TR = []
AUC_CV = []

for i in tqdm(C_Para):
    SGC = SGDClassifier(loss = 'hinge',penalty='l1', n_jobs=-1,alpha=i)
    SGC.fit(TFIDF_Train, Y_Train)
    pred = SGC.predict(TFIDF_CV)
    acc = accuracy_score(Y_CV, pred, normalize=True) * float(100)
    ACCV.append(acc)
    #a_fpr_train, a_tpr_train, thresholds = roc_curve(Y_Train, SGC.predict_proba(TFIDF_TR_CSR) [:,1])
    Train_pred = batch_predict(SGC, TFIDF_TR_CSR)
    a_fpr_train,a_tpr_train,c = roc_curve(Y_Train, Train_pred)
    AUC_TR.append(auc(a_fpr_train, a_tpr_train))

    #a_fpr_cv, a_tpr_cv, thresholds = roc_curve(Y_CV, SGC.predict_proba(TFIDF_CV_CSR) [:,1])
    CV_pred = batch_predict(SGC, TFIDF_CV_CSR)
    a_fpr_cv,a_tpr_cv,c = roc_curve(Y_CV, CV_pred)
    AUC_CV.append(auc(a_fpr_cv, a_tpr_cv))


for av in tqdm(C_Para):
    b = np.log10(av)
    L_C_Para.append(b)

# Performance of model on Train data and Test data for each hyper parameter.
plt.plot(L_C_Para, AUC_TR, label='AUC_Train',color='red',linewidth=2)
plt.scatter(L_C_Para, AUC_TR, label='AUC_Train',color='red',linewidth=2)
plt.gca()
plt.plot(L_C_Para, AUC_CV, label='AUC_CV',color='blue',linewidth=2)
plt.scatter(L_C_Para, AUC_CV, label='AUC_CV',color='blue',linewidth=2)
plt.gca()
plt.legend()
plt.xlabel("Log alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC Values for respective alpha Values(Hyperparameter Tuning)")
plt.grid()
```

```
46  plt.show()
```

```
100%|██████████| 9/9 [00:08<00:00,  1.13it/s]
100%|██████████| 9/9 [00:00<00:00, 24120.60it/s]
```




```
CPU times: user 13.4 s, sys: 28.4 s, total: 41.8 s
Wall time: 8.99 s
```

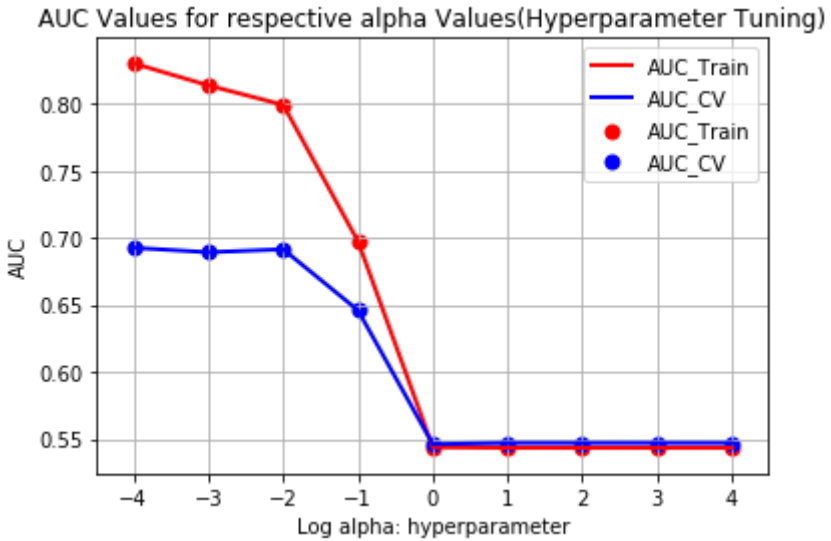
Applying Support Vector Machines(SVM) on TFIDF with L2 regularization, SET 2

In [0]:

```
%%time

TFIDF_TR_CSR = TFIDF_Train.tocsr()
TFIDF_CV_CSR = TFIDF_CV.tocsr()
TFIDF_TS_CSR = TFIDF_Test.tocsr()

C_Para = [1e-4,1e-3,1e-2,1e-1,1,1e+1,1e+2,1e+3,1e+4]
L_C_Para= []
ACCV = []
AUC_TR = []
AUC_CV = []

for i in tqdm(C_Para):
    SGC = SGDClassifier(loss = 'hinge',penalty='l2', n_jobs=-1,alpha=i)
    SGC.fit(TFIDF_Train, Y_Train)
    pred = SGC.predict(TFIDF_CV)
    acc = accuracy_score(Y_CV, pred, normalize=True) * float(100)
    ACCV.append(acc)
    #a_fpr_train, a_tpr_train, thresholds = roc_curve(Y_Train, SGC.predict_proba(TFIDF_TR_CSR) [:,1])
    Train_pred = batch_predict(SGC, TFIDF_TR_CSR)
    a_fpr_train,a_tpr_train,c = roc_curve(Y_Train, Train_pred)
    AUC_TR.append(auc(a_fpr_train, a_tpr_train))

    #a_fpr_cv, a_tpr_cv, thresholds = roc_curve(Y_CV, SGC.predict_proba(TFIDF_CV_CSR) [:,1])
    CV_pred = batch_predict(SGC, TFIDF_CV_CSR)
    a_fpr_cv,a_tpr_cv,c = roc_curve(Y_CV, CV_pred)
    AUC_CV.append(auc(a_fpr_cv, a_tpr_cv))


for av in tqdm(C_Para):
    b = np.log10(av)
    L_C_Para.append(b)

# Performance of model on Train data and Test data for each hyper parameter.
plt.plot(L_C_Para, AUC_TR, label='AUC_Train',color='red',linewidth=2)
plt.scatter(L_C_Para, AUC_TR, label='AUC_Train',color='red',linewidth=2)
plt.gca()
plt.plot(L_C_Para, AUC_CV, label='AUC_CV',color='blue',linewidth=2)
plt.scatter(L_C_Para, AUC_CV, label='AUC_CV',color='blue',linewidth=2)
plt.gca()
plt.legend()
plt.xlabel("Log alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC Values for respective alpha Values(Hyperparameter Tuning)")
plt.grid()
```

```
46 plt.show()
```

```
100%|██████████| 9/9 [00:06<00:00,  1.37it/s]
100%|██████████| 9/9 [00:00<00:00, 20482.22it/s]
```




```
CPU times: user 12 s, sys: 29.8 s, total: 41.9 s
Wall time: 7.42 s
```

OBSERVATION:

1. α value has been choose based on the alpha which maximize the AUC CV score.
2. In this case **α = 0.0001** is having the highest AUC CV score.
3. Hence **α = 0.0001** is choosen as a best α value.

In [0]:

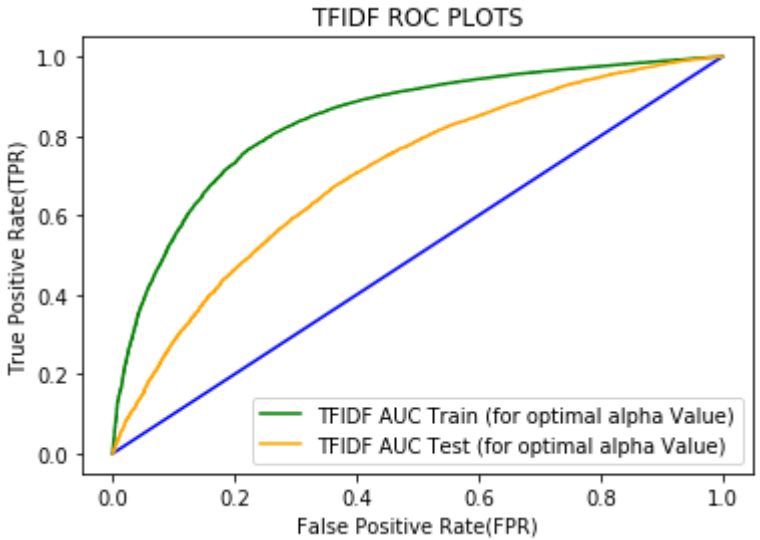
```
SGC_OPT=0.0001
TFIDF_opt =SGDClassifier(penalty='l2', alpha=SGC_OPT, class_weight='balanced', n_jobs=-1)

TFIDF_opt.fit(TFIDF_Train, Y_Train)
pred = TFIDF_opt.predict(TFIDF_Test)
acc = accuracy_score(Y_Test, pred, normalize=True) * float(100)
print('\nTest accuracy for alpha Value = {0} is {1}%'.format(SGC_OPT,acc))

a_fpr_train, a_tpr_train, thresholds = roc_curve(Y_Train, TFIDF_opt.decision_function(TFIDF_TR_CSR))
a_fpr_Test, a_tpr_Test, thresholds = roc_curve(Y_Test, TFIDF_opt.decision_function(TFIDF_TS_CSR))
```

```
Test accuracy for alpha Value = 0.0001 is 74.9112393209808%
```

TFIDF ROC PLOT

In [0]:

```
%%time

#https://stackoverflow.com/questions/52910061/implementing-roc-curves-for-k-nn-machine-learning-algorithm-using-python-and-sci

plt.plot([0,1],[0,1],'k-', color='blue')
plt.plot(a_fpr_train, a_tpr_train, label="TFIDF AUC Train (for optimal alpha Value)", color='green')
plt.plot(a_fpr_Test, a_tpr_Test, label="TFIDF AUC Test (for optimal alpha Value)", color='orange')
plt.legend()
plt.ylabel("True Positive Rate(TPR)")
plt.xlabel("False Positive Rate(FPR)")
plt.title("TFIDF ROC PLOTS")
plt.show()
print("-"*120)
print("AUC Train (for optimal alpha Value) =", auc(a_fpr_train, a_tpr_train))
print("AUC Test (for optimal alpha Value) =", auc(a_fpr_Test, a_tpr_Test))
TFIDF_AOPT=SGC_OPT
TFIDF_AUC=round(auc(a_fpr_Test, a_tpr_Test)*100)
pred3 = TFIDF_opt.predict(TFIDF_Train)
pred4 = TFIDF_opt.predict(TFIDF_Test)

```

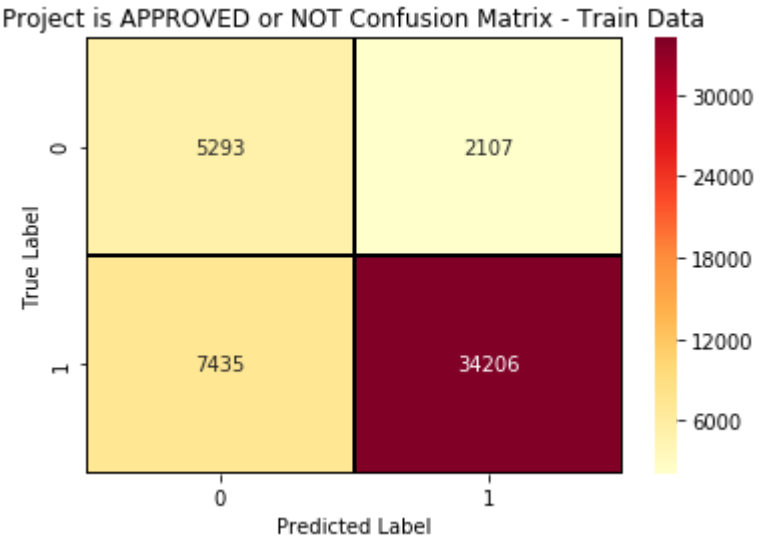



```
------------------------------------------------------------------------------------------------------------------------
AUC Train (for optimal alpha Value) = 0.8392740895959478
AUC Test (for optimal alpha Value) = 0.7051983991130664
CPU times: user 578 ms, sys: 7.82 ms, total: 586 ms
Wall time: 581 ms
```

TFIDF CONFUSION MATRIX

In [0]:
```
%%time
#https://seaborn.pydata.org/generated/seaborn.heatmap.html
#https://getaravind.com/blog/confusion-matrix-seaborn-heatmap/
%matplotlib inline
from sklearn.metrics import confusion_matrix
Train = confusion_matrix(Y_Train, pred3)
sns.heatmap(Train,annot=True,cbar=True,fmt='d',cmap='YlOrRd',linewidths=1,linecolor='black')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Project is APPROVED or NOT Confusion Matrix - Train Data')
```

```
CPU times: user 100 ms, sys: 35.3 ms, total: 136 ms
Wall time: 92.2 ms
```



OBSERVATION:

True Negative =5293; False Negative =7435; True Positive = 34206; False Positive = 2107

Accuracy (Overall, how often is the classifier correct) = 0.81

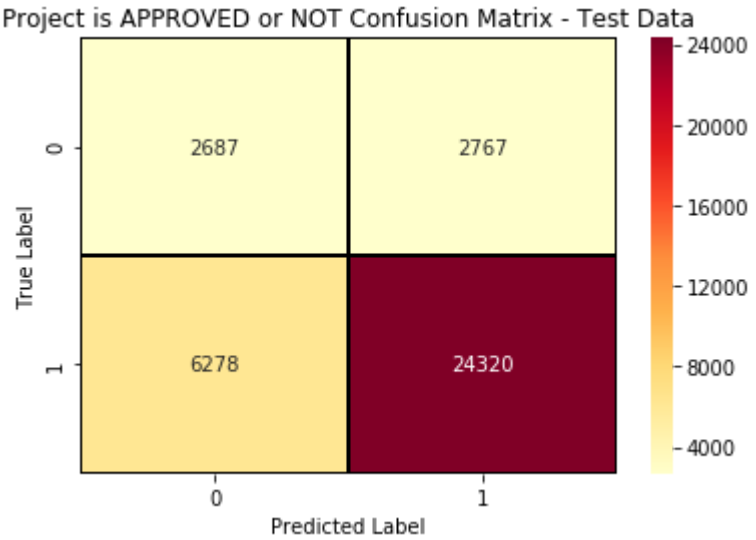
Precision(When it predicts yes, how often is it correct) =0.95

Misclassification (Overall, how often is it wrong) =0.20

In [0]:

```
%%time
#https://seaborn.pydata.org/generated/seaborn.heatmap.html
#https://getaravind.com/blog/confusion-matrix-seaborn-heatmap/
Test = confusion_matrix(Y_Test, pred4)
sns.heatmap(Test,annot=True,cbar=True,fmt='d',cmap='YlOrRd',linewidths=1,linecolor='black')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Project is APPROVED or NOT Confusion Matrix - Test Data')
```

```
CPU times: user 91.9 ms, sys: 38 ms, total: 130 ms
Wall time: 83.1 ms
```



OBSERVATION:

True Negative =2687; False Negative =6278; True Positive = 24320; False Positive = 2767

Accuracy (Overall, how often is the classifier correct) = 0.75

Precision(When it predicts yes, how often is it correct) =0.90

Misclassification (Overall, how often is it wrong) =0.26

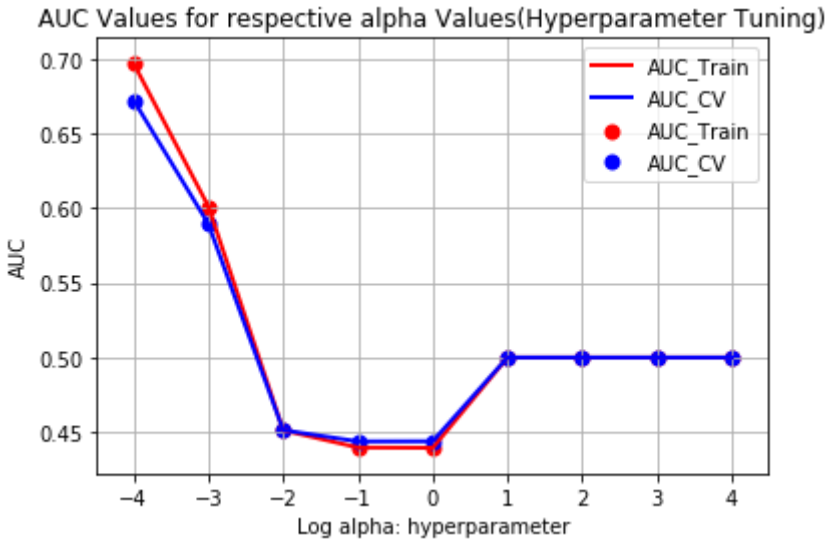
Applying Support Vector Machines(SVM) on AVG W2V with L1 regularization, SET 3

In [0]:

```
%%time
AVG_W2V_TR_CSR = AVG_W2V_Train.tocsr()
AVG_W2V_CV_CSR = AVG_W2V_CV.tocsr()
AVG_W2V_TS_CSR = AVG_W2V_Test.tocsr()

C_Para = [1e-4,1e-3,1e-2,1e-1,1,1e+1,1e+2,1e+3,1e+4]
L_C_Para= []
ACCV = []
AUC_TR = []
AUC_CV = []

for i in tqdm(C_Para):
    SGC = SGDClassifier(loss = 'hinge',penalty='l1', n_jobs=-1,alpha=i)
    SGC.fit(AVG_W2V_Train, Y_Train)
    pred = SGC.predict(AVG_W2V_CV)
    acc = accuracy_score(Y_CV, pred, normalize=True) * float(100)
    ACCV.append(acc)
    Train_pred = batch_predict(SGC, AVG_W2V_TR_CSR)
    a_fpr_train,a_tpr_train,c = roc_curve(Y_Train, Train_pred)
    AUC_TR.append(auc(a_fpr_train, a_tpr_train))

    CV_pred = batch_predict(SGC, AVG_W2V_CV_CSR)
    a_fpr_cv,a_tpr_cv,c = roc_curve(Y_CV, CV_pred)
    AUC_CV.append(auc(a_fpr_cv, a_tpr_cv))


for av in tqdm(C_Para):
    b = np.log10(av)
    L_C_Para.append(b)

# Performance of model on Train data and Test data for each hyper parameter.
plt.plot(L_C_Para, AUC_TR, label='AUC_Train',color='red',linewidth=2)
plt.scatter(L_C_Para, AUC_TR, label='AUC_Train',color='red',linewidth=2)
plt.gca()
plt.plot(L_C_Para, AUC_CV, label='AUC_CV',color='blue',linewidth=2)
plt.scatter(L_C_Para, AUC_CV, label='AUC_CV',color='blue',linewidth=2)
plt.gca()
plt.legend()
plt.xlabel("Log alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC Values for respective alpha Values(Hyperparameter Tuning)")
plt.grid()
plt.show()
```

```
100%|██████████| 9/9 [00:34<00:00,  3.55s/it]
```

```
100%|██████████| 9/9 [00:00<00:00, 37117.73it/s]
```




```
CPU times: user 34.9 s, sys: 684 ms, total: 35.6 s
Wall time: 35.4 s
```

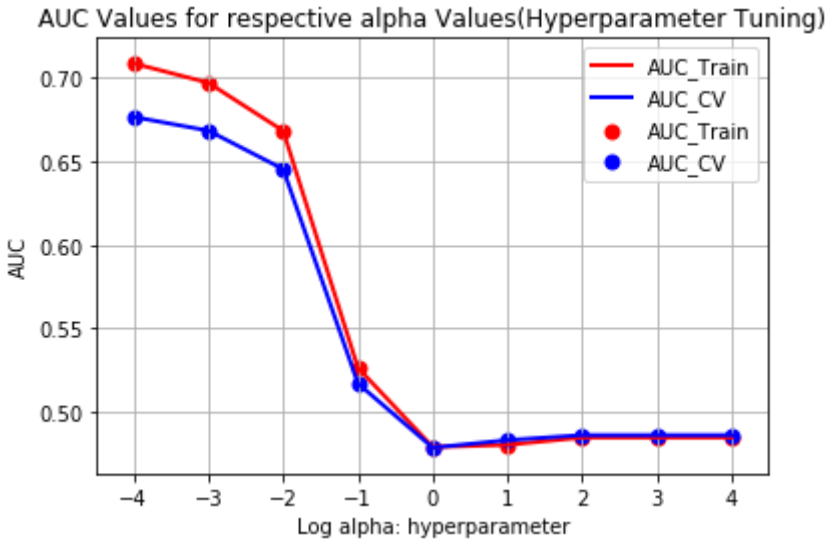
Applying Support Vector Machines(SVM) on AVG W2V with L2 regularization, SET 3

In [0]:

```
%%time
AVG_W2V_TR_CSR = AVG_W2V_Train.tocsr()
AVG_W2V_CV_CSR = AVG_W2V_CV.tocsr()
AVG_W2V_TS_CSR = AVG_W2V_Test.tocsr()

C_Para = [1e-4,1e-3,1e-2,1e-1,1,1e+1,1e+2,1e+3,1e+4]
L_C_Para= []
ACCV = []
AUC_TR = []
AUC_CV = []

for i in tqdm(C_Para):
    SGC = SGDClassifier(loss = 'hinge',penalty='l2', n_jobs=-1,alpha=i)
    SGC.fit(AVG_W2V_Train, Y_Train)
    pred = SGC.predict(AVG_W2V_CV)
    acc = accuracy_score(Y_CV, pred, normalize=True) * float(100)
    ACCV.append(acc)
    Train_pred = batch_predict(SGC, AVG_W2V_TR_CSR)
    a_fpr_train,a_tpr_train,c = roc_curve(Y_Train, Train_pred)
    AUC_TR.append(auc(a_fpr_train, a_tpr_train))

    CV_pred = batch_predict(SGC, AVG_W2V_CV_CSR)
    a_fpr_cv,a_tpr_cv,c = roc_curve(Y_CV, CV_pred)
    AUC_CV.append(auc(a_fpr_cv, a_tpr_cv))


for av in tqdm(C_Para):
    b = np.log10(av)
    L_C_Para.append(b)

# Performance of model on Train data and Test data for each hyper parameter.
plt.plot(L_C_Para, AUC_TR, label='AUC_Train',color='red',linewidth=2)
plt.scatter(L_C_Para, AUC_TR, label='AUC_Train',color='red',linewidth=2)
plt.gca()
plt.plot(L_C_Para, AUC_CV, label='AUC_CV',color='blue',linewidth=2)
plt.scatter(L_C_Para, AUC_CV, label='AUC_CV',color='blue',linewidth=2)
plt.gca()
plt.legend()
plt.xlabel("Log alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC Values for respective alpha Values(Hyperparameter Tuning)")
plt.grid()
plt.show()
```

```
100%|██████████| 9/9 [00:17<00:00,  1.73s/it]
```

```
100%|██████████| 9/9 [00:00<00:00, 27675.03it/s]
```




```
CPU times: user 18.5 s, sys: 265 ms, total: 18.8 s
Wall time: 18.5 s
```

OBSERVATION:

1. α value has been choose based on the alpha which maximize the AUC CV score.
2. In this case **α = 0.0001** is having the highest AUC CV score.
3. Hence **α = 0.0001** is choosen as a best α value.

In [0]:

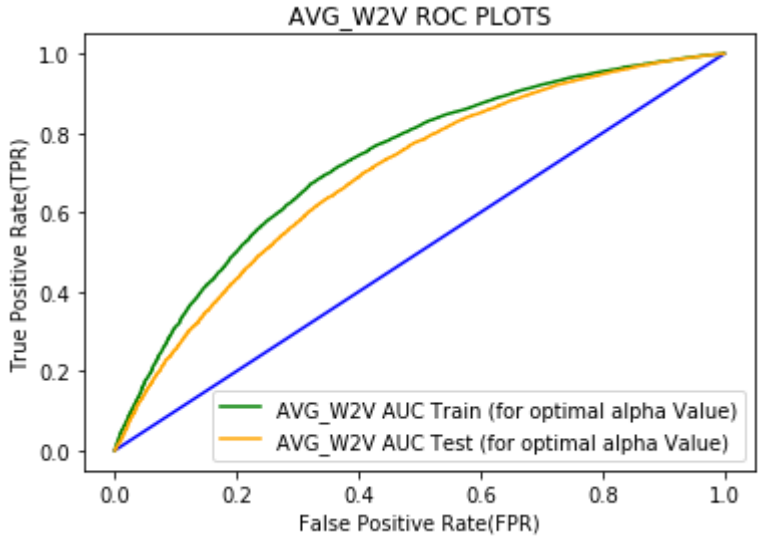
```
SGC_OPT=0.0001
AVG_W2V_opt =SGDClassifier(penalty='l2', alpha=SGC_OPT, class_weight='balanced', n_jobs=-1)
AVG_W2V_opt.fit(AVG_W2V_Train, Y_Train)
pred = AVG_W2V_opt.predict(AVG_W2V_Test)
acc = accuracy_score(Y_Test, pred, normalize=True) * float(100)
print('\nTest accuracy for alpha Value = {0} is {1}%'.format(SGC_OPT,acc))

a_fpr_train, a_tpr_train, thresholds = roc_curve(Y_Train, AVG_W2V_opt.decision_function(AVG_W2V_TR_CSR))
a_fpr_Test, a_tpr_Test, thresholds = roc_curve(Y_Test, AVG_W2V_opt.decision_function(AVG_W2V_TS_CSR))
```

```
Test accuracy for alpha Value = 0.0001 is 73.63530456007989%
```

AVG_W2V ROC PLOT

In [0]:

```
%%time

#https://stackoverflow.com/questions/52910061/implementing-roc-curves-for-k-nn-machine-learning-algorithm-using-python-and-sci

plt.plot([0,1],[0,1],'k-', color='blue')
plt.plot(a_fpr_train, a_tpr_train, label="AVG_W2V AUC Train (for optimal alpha Value)", color='green')
plt.plot(a_fpr_Test, a_tpr_Test, label="AVG_W2V AUC Test (for optimal alpha Value)", color='orange')
plt.legend()
plt.ylabel("True Positive Rate(TPR)")
plt.xlabel("False Positive Rate(FPR)")
plt.title("AVG_W2V ROC PLOTS")
plt.show()
print("-"*120)
print("AUC Train (for optimal alpha Value) =", auc(a_fpr_train, a_tpr_train))
print("AUC Test (for optimal alpha Value) =", auc(a_fpr_Test, a_tpr_Test))
AVG_W2V_AOPT=SGC_OPT
AVG_W2V_AUC=round(auc(a_fpr_Test, a_tpr_Test)*100)
pred5 = AVG_W2V_opt.predict(AVG_W2V_Train)
pred6 = AVG_W2V_opt.predict(AVG_W2V_Test)
```

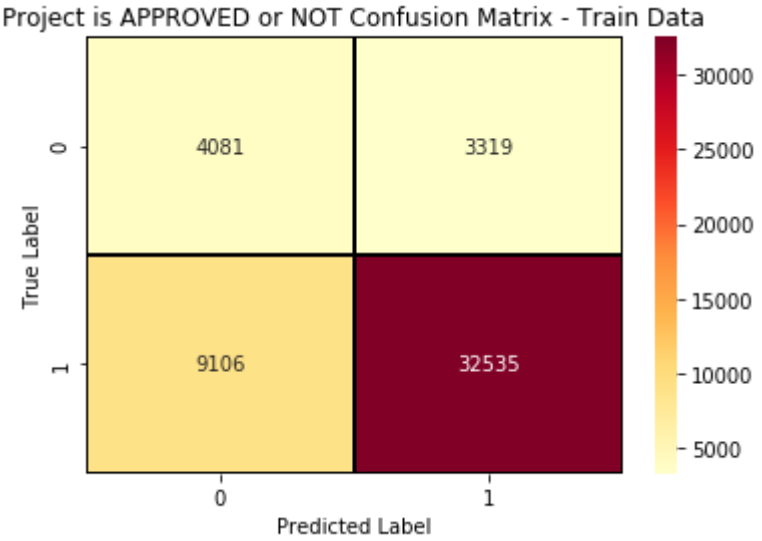



```
------------------------------------------------------------------------------------------------------------------------
AUC Train (for optimal alpha Value) = 0.7281203881050187
AUC Test (for optimal alpha Value) = 0.6950187921378363
CPU times: user 731 ms, sys: 2.77 ms, total: 734 ms
Wall time: 731 ms
```

AVG_W2V CONFUSION MATRIX

In [0]:
```
%%time
#https://seaborn.pydata.org/generated/seaborn.heatmap.html
#https://getaravind.com/blog/confusion-matrix-seaborn-heatmap/
%matplotlib inline
from sklearn.metrics import confusion_matrix
Train = confusion_matrix(Y_Train, pred5)
sns.heatmap(Train,annot=True,cbar=True,fmt='d',cmap='YlOrRd',linewidths=1,linecolor='black')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Project is APPROVED or NOT Confusion Matrix - Train Data')

```

```
CPU times: user 119 ms, sys: 25.9 ms, total: 144 ms
Wall time: 98.2 ms
```



OBSERVATION:

True Negative =4081; False Negative =9106; True Positive = 32535; False Positive = 3319

Accuracy (Overall, how often is the classifier correct) = 0.75

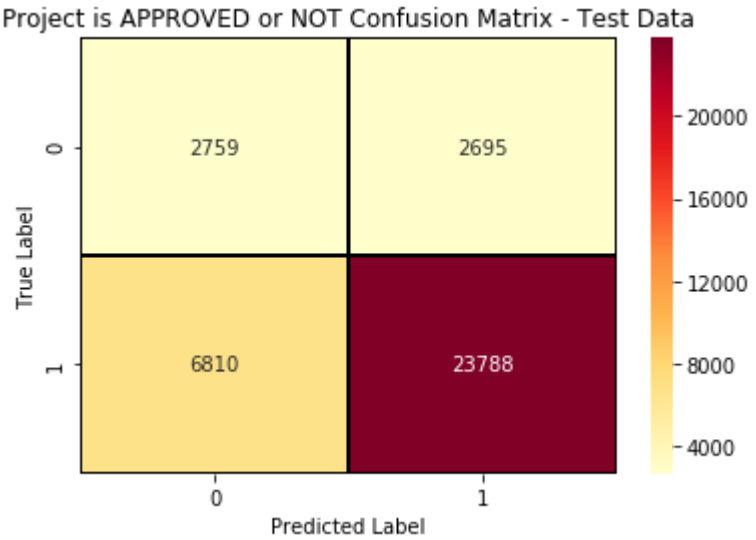
Precision(When it predicts yes, how often is it correct) =0.91

Misclassification (Overall, how often is it wrong) =0.26

In [0]:

```
%%time
#https://seaborn.pydata.org/generated/seaborn.heatmap.html
#https://getaravind.com/blog/confusion-matrix-seaborn-heatmap/
Test = confusion_matrix(Y_Test, pred6)
sns.heatmap(Test,annot=True,cbar=True,fmt='d',cmap='YlOrRd',linewidths=1,linecolor='black')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Project is APPROVED or NOT Confusion Matrix - Test Data')
```

```
CPU times: user 90 ms, sys: 40 ms, total: 130 ms
Wall time: 84.4 ms
```



OBSERVATION:

True Negative =2759; False Negative =6810; True Positive = 23788; False Positive = 2695

Accuracy (Overall, how often is the classifier correct) = 0.74

Precision(When it predicts yes, how often is it correct) =0.90

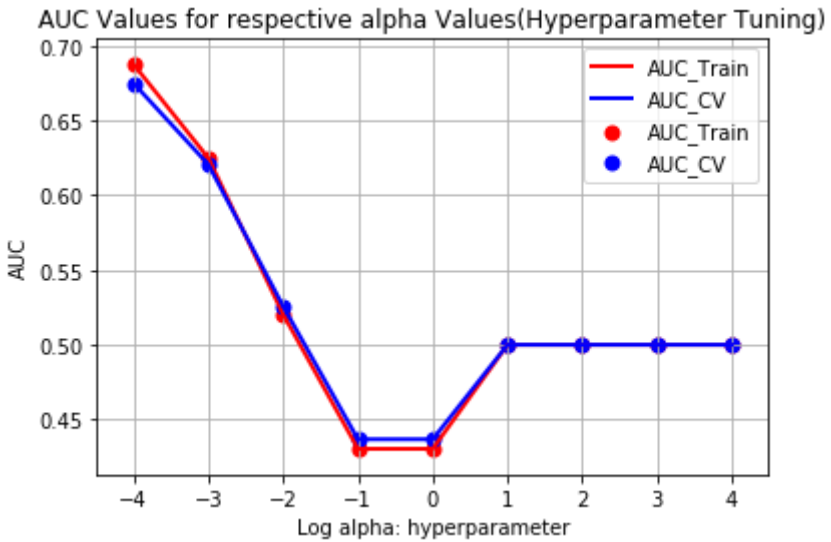
Misclassification (Overall, how often is it wrong) =0.27

Applying Support Vector Machines(SVM) on TFIDF W2V with L1 regularization, SET 4

In [0]:

```
TFIDF_W2V_TR_CSR = TFIDF_W2V_Train.tocsr()
TFIDF_W2V_CV_CSR = TFIDF_W2V_CV.tocsr()
TFIDF_W2V_TS_CSR = TFIDF_W2V_Test.tocsr()

C_Para = [1e-4,1e-3,1e-2,1e-1,1,1e+1,1e+2,1e+3,1e+4]
L_C_Para= []
ACCV = []
AUC_TR = []
AUC_CV = []

for i in tqdm(C_Para):
    SGC = SGDClassifier(loss = 'hinge',penalty='l1', n_jobs=-1,alpha=i)
    SGC.fit(TFIDF_W2V_Train, Y_Train)
    pred = SGC.predict(TFIDF_W2V_CV)
    acc = accuracy_score(Y_CV, pred, normalize=True) * float(100)
    ACCV.append(acc)
    Train_pred = batch_predict(SGC, TFIDF_W2V_TR_CSR)
    a_fpr_train,a_tpr_train,c = roc_curve(Y_Train, Train_pred)
    AUC_TR.append(auc(a_fpr_train, a_tpr_train))

    CV_pred = batch_predict(SGC, TFIDF_W2V_CV_CSR)
    a_fpr_cv,a_tpr_cv,c = roc_curve(Y_CV, CV_pred)
    AUC_CV.append(auc(a_fpr_cv, a_tpr_cv))


for av in tqdm(C_Para):
    b = np.log10(av)
    L_C_Para.append(b)

# Performance of model on Train data and Test data for each hyper parameter.
plt.plot(L_C_Para, AUC_TR, label='AUC_Train',color='red',linewidth=2)
plt.scatter(L_C_Para, AUC_TR, label='AUC_Train',color='red',linewidth=2)
plt.gca()
plt.plot(L_C_Para, AUC_CV, label='AUC_CV',color='blue',linewidth=2)
plt.scatter(L_C_Para, AUC_CV, label='AUC_CV',color='blue',linewidth=2)
plt.gca()
plt.legend()
plt.xlabel("Log alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC Values for respective alpha Values(Hyperparameter Tuning)")
plt.grid()
plt.show()
```

```
100%|██████████| 9/9 [00:17<00:00,  1.85s/it]
100%|██████████| 9/9 [00:00<00:00, 46374.37it/s]
```

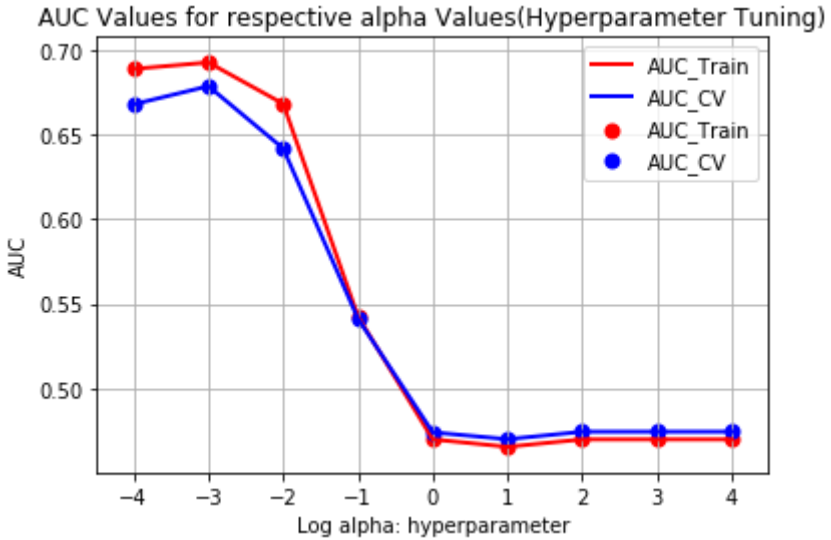


Applying Support Vector Machines(SVM) on TFIDF W2V with L2 regularization, SET 4

In [0]:

```
TFIDF_W2V_TR_CSR = TFIDF_W2V_Train.tocsr()
TFIDF_W2V_CV_CSR = TFIDF_W2V_CV.tocsr()
TFIDF_W2V_TS_CSR = TFIDF_W2V_Test.tocsr()

C_Para = [1e-4,1e-3,1e-2,1e-1,1,1e+1,1e+2,1e+3,1e+4]
L_C_Para= []
ACCV = []
AUC_TR = []
AUC_CV = []

for i in tqdm(C_Para):
    SGC = SGDClassifier(loss = 'hinge',penalty='l2', n_jobs=-1,alpha=i)
    SGC.fit(TFIDF_W2V_Train, Y_Train)
    pred = SGC.predict(TFIDF_W2V_CV)
    acc = accuracy_score(Y_CV, pred, normalize=True) * float(100)
    ACCV.append(acc)
    Train_pred = batch_predict(SGC, TFIDF_W2V_TR_CSR)
    a_fpr_train,a_tpr_train,c = roc_curve(Y_Train, Train_pred)
    AUC_TR.append(auc(a_fpr_train, a_tpr_train))

    CV_pred = batch_predict(SGC, TFIDF_W2V_CV_CSR)
    a_fpr_cv,a_tpr_cv,c = roc_curve(Y_CV, CV_pred)
    AUC_CV.append(auc(a_fpr_cv, a_tpr_cv))


for av in tqdm(C_Para):
    b = np.log10(av)
    L_C_Para.append(b)

# Performance of model on Train data and Test data for each hyper parameter.
plt.plot(L_C_Para, AUC_TR, label='AUC_Train',color='red',linewidth=2)
plt.scatter(L_C_Para, AUC_TR, label='AUC_Train',color='red',linewidth=2)
plt.gca()
plt.plot(L_C_Para, AUC_CV, label='AUC_CV',color='blue',linewidth=2)
plt.scatter(L_C_Para, AUC_CV, label='AUC_CV',color='blue',linewidth=2)
plt.gca()
plt.legend()
plt.xlabel("Log alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC Values for respective alpha Values(Hyperparameter Tuning)")
plt.grid()
plt.show()
```

```
100%|██████████| 9/9 [00:09<00:00,  1.02it/s]
100%|██████████| 9/9 [00:00<00:00, 36972.32it/s]
```



OBSERVATION:

1. α value has been choose based on the alpha which maximize the AUC CV score.
2. In this case **α = 0.001** is having the highest AUC CV score.
3. Hence **α = 0.001** is choosen as a best α value.

In [0]:

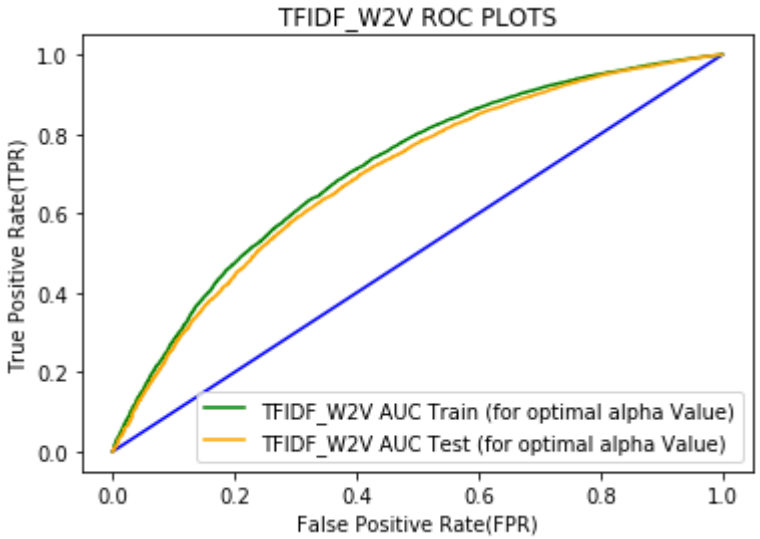
```
SGC_OPT=0.001
TFIDF_W2V_opt =SGDClassifier(penalty='l2', alpha=SGC_OPT, class_weight='balanced', n_jobs=-1)
TFIDF_W2V_opt.fit(TFIDF_W2V_Train, Y_Train)
pred = TFIDF_W2V_opt.predict(TFIDF_W2V_Test)
acc = accuracy_score(Y_Test, pred, normalize=True) * float(100)
print('\nTest accuracy for alpha Value = {0} is {1}%'.format(SGC_OPT,acc))

a_fpr_train, a_tpr_train, thresholds = roc_curve(Y_Train, TFIDF_W2V_opt.decision_function(TFIDF_W2V_TR_CSR))
a_fpr_Test, a_tpr_Test, thresholds = roc_curve(Y_Test, TFIDF_W2V_opt.decision_function(TFIDF_W2V_TS_CSR))
```

```
Test accuracy for alpha Value = 0.001 is 69.52734938422279%
```

TFIDF_W2V ROC PLOT

In [0]:

```
%%time

#https://stackoverflow.com/questions/52910061/implementing-roc-curves-for-k-nn-machine-learning-algorithm-using-python-and-sci

plt.plot([0,1],[0,1],'k-', color='blue')
plt.plot(a_fpr_train, a_tpr_train, label="TFIDF_W2V AUC Train (for optimal alpha Value)", color='green')
plt.plot(a_fpr_Test, a_tpr_Test, label="TFIDF_W2V AUC Test (for optimal alpha Value)", color='orange')
plt.legend()
plt.ylabel("True Positive Rate(TPR)")
plt.xlabel("False Positive Rate(FPR)")
plt.title("TFIDF_W2V ROC PLOTS")
plt.show()
print("-"*120)
print("AUC Train (for optimal alpha Value) =", auc(a_fpr_train, a_tpr_train))
print("AUC Test (for optimal alpha Value) =", auc(a_fpr_Test, a_tpr_Test))
TFIDF_W2V_AOPT=SGC_OPT
TFIDF_W2V_AUC=round(auc(a_fpr_Test, a_tpr_Test)*100)
pred7 = TFIDF_W2V_opt.predict(TFIDF_W2V_Train)
pred8 = TFIDF_W2V_opt.predict(TFIDF_W2V_Test)
```




```
------------------------------------------------------------------------------------------------------------------------
AUC Train (for optimal alpha Value) = 0.7113998596108175
AUC Test (for optimal alpha Value) = 0.6958268266201744
```

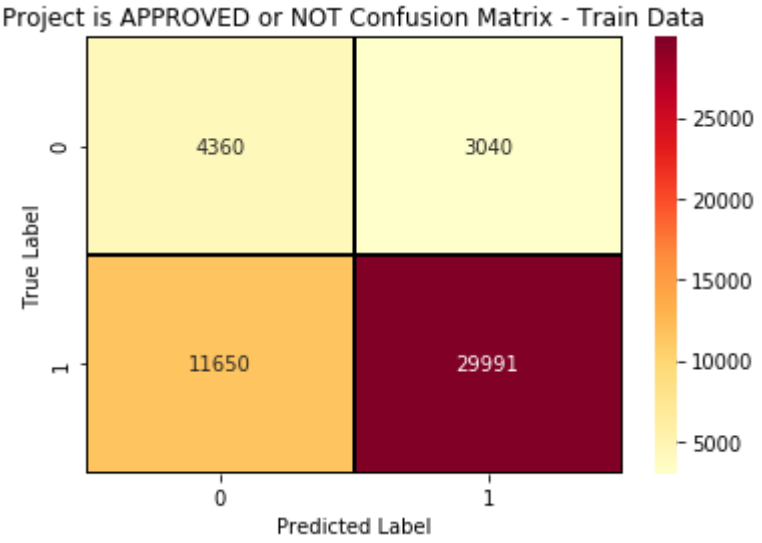
```
CPU times: user 519 ms, sys: 3.2 ms, total: 522 ms
Wall time: 521 ms
```

TFIDF_W2V CONFUSION MATRIX

In [0]:

```
%%time
#https://seaborn.pydata.org/generated/seaborn.heatmap.html
#https://getaravind.com/blog/confusion-matrix-seaborn-heatmap/
%matplotlib inline
from sklearn.metrics import confusion_matrix
Train = confusion_matrix(Y_Train, pred7)
sns.heatmap(Train,annot=True,cbar=True,fmt='d',cmap='YlOrRd',linewidths=1,linecolor='black')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Project is APPROVED or NOT Confusion Matrix - Train Data')
```

```
CPU times: user 102 ms, sys: 41 ms, total: 143 ms
Wall time: 96.8 ms
```



OBSERVATION:

True Negative =4360; False Negative =11650; True Positive = 29991; False Positive = 3040

Accuracy (Overall, how often is the classifier correct) = 0.71

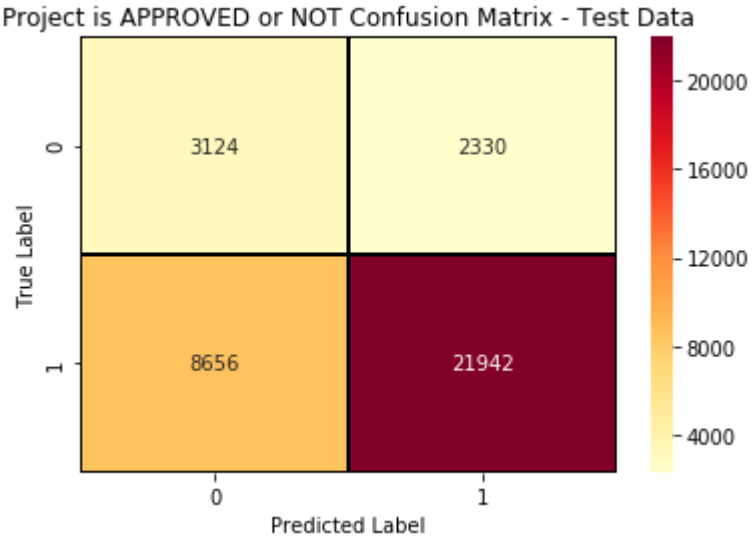
Precision(When it predicts yes, how often is it correct) =0.91

Misclassification (Overall, how often is it wrong) =0.30

In [0]:

```
%%time
#https://seaborn.pydata.org/generated/seaborn.heatmap.html
#https://getaravind.com/blog/confusion-matrix-seaborn-heatmap/
Test = confusion_matrix(Y_Test, pred8)
sns.heatmap(Test,annot=True,cbar=True,fmt='d',cmap='YlOrRd',linewidths=1,linecolor='black')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Project is APPROVED or NOT Confusion Matrix - Test Data')
```

```
CPU times: user 1 s, sys: 21.3 ms, total: 1.02 s
Wall time: 980 ms
```



OBSERVATION:

True Negative =3124; False Negative =8656; True Positive = 21942; False Positive = 2330

Accuracy (Overall, how often is the classifier correct) = 0.70

Precision(When it predicts yes, how often is it correct) =0.91

Misclassification (Overall, how often is it wrong) =0.31

- **[Task-2] Apply the Support Vector Machines on these features by finding the best hyper paramter as suggested in step 2 and step 3**
  - Consider these set of features Set 5 :
    - **school_state** : categorical data
    - **clean_categories** : categorical data
    - **clean_subcategories** : categorical data

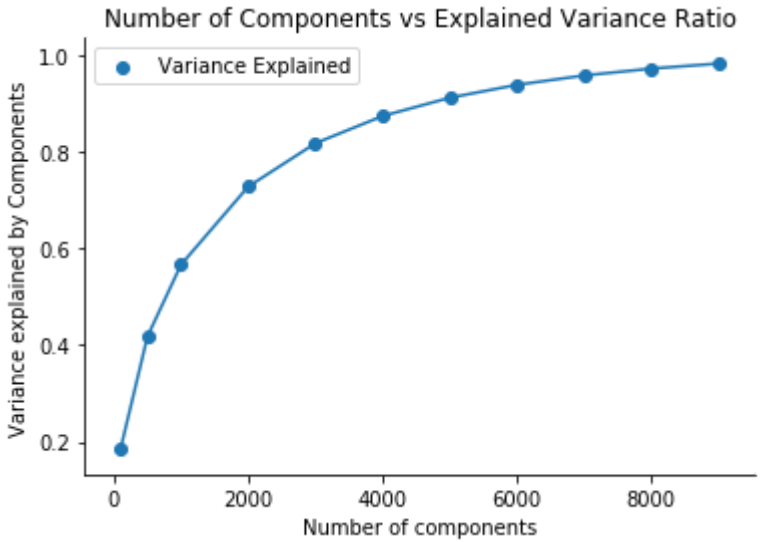
- **project_grade_category** :categorical data
- **teacher_prefix** : categorical data
- **quantity** : numerical data
- **teacher_number_of_previously_posted_projects** : numerical data
- **price** : numerical data
- **sentiment score's of each of the essay** : numerical data
- **number of words in the title** : numerical data
- **number of words in the combine essays** : numerical data
- **Apply TruncatedSVD (http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html) on TfidfVectorizer (https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html) of essay text, choose the number of components (`n_components`) using elbow method (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/pca-code-example-using-non-visualization/)** : numerical data

Choosing the no of components using the Elbow Method

In [0]:

```
%%time
#https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html
#https://www.analyticsvidhya.com/blog/2016/03/practical-guide-principal-component-analysis-python/
#https://medium.com/@bioturing/how-to-read-pca-biplots-and-scree-plots-186246aae063
no_of_comp = [100,500,1000,2000,3000,4000,5000,6000,7000,8000,9000]
agg_variance = []
for i in tqdm(no_of_comp):
    tsvd1 = TruncatedSVD(n_components =i,random_state = 0,n_iter = 3 )
    tsvd1.fit(text_tfidf_Train)
    agg_variance.append(tsvd1.explained_variance_ratio_.sum())

plt.plot(no_of_comp,agg_variance)
plt.scatter(no_of_comp,agg_variance,label = 'Variance Explained')
plt.xlabel("Number of components")
plt.ylabel("Variance explained by Components")
plt.legend()
plt.title("Number of Components vs Explained Variance Ratio")
sns.despine()
plt.show()
```

```
100%|██████████| 11/11 [45:44<00:00, 436.29s/it]
```



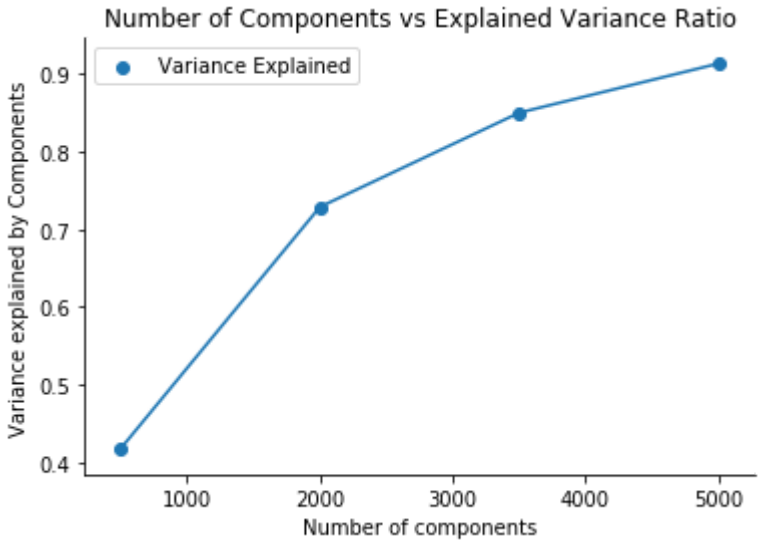

```
CPU times: user 7h 26min 9s, sys: 2h 6min 3s, total: 9h 32min 13s
Wall time: 45min 45s
```

Zoom In the Elbow Curve

In [0]:

```
%%time
no_of_comp = [500,2000,3500,5000]
agg_variance = []
for i in tqdm(no_of_comp):
    tsvd1 = TruncatedSVD(n_components =i,random_state = 0,n_iter = 3 )
    tsvd1.fit(text_tfidf_Train)
    agg_variance.append(tsvd1.explained_variance_ratio_.sum())

plt.plot(no_of_comp,agg_variance)
plt.scatter(no_of_comp,agg_variance,label = 'Variance Explained')
plt.xlabel("Number of components")
plt.ylabel("Variance explained by Components")
plt.legend()
plt.title("Number of Components vs Explained Variance Ratio")
sns.despine()
plt.show()
```

```
100%|██████████| 4/4 [09:39<00:00, 143.96s/it]
```




```
CPU times: user 1h 2min 21s, sys: 34min 21s, total: 1h 36min 42s
Wall time: 9min 39s
```

In [0]:
```
from sklearn.decomposition import TruncatedSVD
tsvd2 = TruncatedSVD(n_components = 2000,n_iter = 5)
tsvd2.fit(text_tfidf_Train)
tsvd_Train = tsvd2.transform(text_tfidf_Train)
tsvd_CV = tsvd2.transform(text_tfidf_CV)
tsvd_Test = tsvd2.transform(text_tfidf_Test)
print(len(tsvd_Train))
print(len(tsvd_Test))
```

```
49041
36052
```

In [0]:
```
FiveF_Train = hstack((categories_one_hot_Train,sub_categories_one_hot_Train,school_one_hot_Train,grade_one_hot_Train,prefix_one
print(FiveF_Train.shape)
FiveF_CV = hstack((categories_one_hot_CV,sub_categories_one_hot_CV,school_one_hot_CV,grade_one_hot_CV,prefix_one_hot_CV,text_t
print(FiveF_CV.shape)
FiveF_Test = hstack((categories_one_hot_Test,sub_categories_one_hot_Test,school_one_hot_Test,grade_one_hot_Test,prefix_one_hot_
print(FiveF_Test.shape)
```

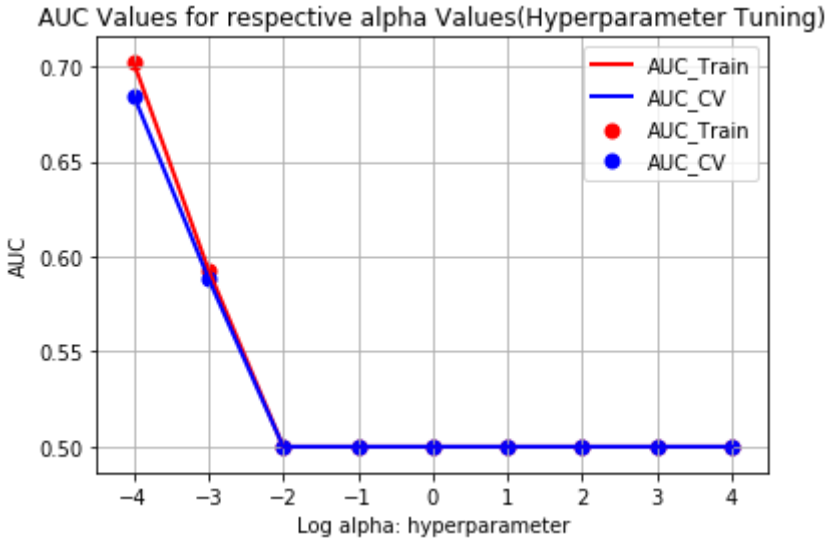
```
(49041, 16289)
(24155, 16289)
(36052, 16289)
```

Applying Support Vector Machines(SVM) on TFIDF for Selected features with L1 regularization, SET 5

In [0]:

```
FiveF_TR_CSR = FiveF_Train.tocsr()
FiveF_CV_CSR = FiveF_CV.tocsr()
FiveF_TS_CSR = FiveF_Test.tocsr()

C_Para = [1e-4,1e-3,1e-2,1e-1,1,1e+1,1e+2,1e+3,1e+4]
L_C_Para= []
ACCV = []
AUC_TR = []
AUC_CV = []

for i in tqdm(C_Para):
    SGC = SGDClassifier(loss = 'hinge',penalty='l1', n_jobs=-1,alpha=i)
    SGC.fit(FiveF_Train, Y_Train)
    pred = SGC.predict(FiveF_CV)
    acc = accuracy_score(Y_CV, pred, normalize=True) * float(100)
    ACCV.append(acc)
    Train_pred = batch_predict(SGC, FiveF_TR_CSR)
    a_fpr_train,a_tpr_train,c = roc_curve(Y_Train, Train_pred)
    AUC_TR.append(auc(a_fpr_train, a_tpr_train))

    CV_pred = batch_predict(SGC, FiveF_CV_CSR)
    a_fpr_cv,a_tpr_cv,c = roc_curve(Y_CV, CV_pred)
    AUC_CV.append(auc(a_fpr_cv, a_tpr_cv))


for av in tqdm(C_Para):
    b = np.log10(av)
    L_C_Para.append(b)

# Performance of model on Train data and Test data for each hyper parameter.
plt.plot(L_C_Para, AUC_TR, label='AUC_Train',color='red',linewidth=2)
plt.scatter(L_C_Para, AUC_TR, label='AUC_Train',color='red',linewidth=2)
plt.gca()
plt.plot(L_C_Para, AUC_CV, label='AUC_CV',color='blue',linewidth=2)
plt.scatter(L_C_Para, AUC_CV, label='AUC_CV',color='blue',linewidth=2)
plt.gca()
plt.legend()
plt.xlabel("Log alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC Values for respective alpha Values(Hyperparameter Tuning)")
plt.grid()
plt.show()
```

```
100%|██████████| 9/9 [02:02<00:00, 13.72s/it]
100%|██████████| 9/9 [00:00<00:00, 26123.69it/s]
```

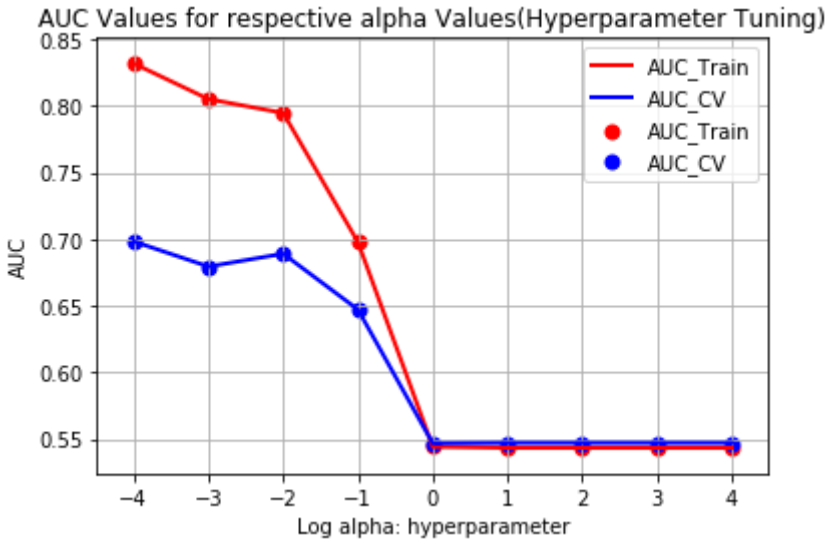


Applying Support Vector Machines(SVM) on TFIDF for Selected features with L2 regularization, SET 5

In [0]:

```
FiveF_TR_CSR = FiveF_Train.tocsr()
FiveF_CV_CSR = FiveF_CV.tocsr()
FiveF_TS_CSR = FiveF_Test.tocsr()

C_Para = [1e-4,1e-3,1e-2,1e-1,1,1e+1,1e+2,1e+3,1e+4]
L_C_Para= []
ACCV = []
AUC_TR = []
AUC_CV = []

for i in tqdm(C_Para):
    SGC = SGDClassifier(loss = 'hinge',penalty='l2', n_jobs=-1,alpha=i)
    SGC.fit(FiveF_Train, Y_Train)
    pred = SGC.predict(FiveF_CV)
    acc = accuracy_score(Y_CV, pred, normalize=True) * float(100)
    ACCV.append(acc)
    Train_pred = batch_predict(SGC, FiveF_TR_CSR)
    a_fpr_train,a_tpr_train,c = roc_curve(Y_Train, Train_pred)
    AUC_TR.append(auc(a_fpr_train, a_tpr_train))

    CV_pred = batch_predict(SGC, FiveF_CV_CSR)
    a_fpr_cv,a_tpr_cv,c = roc_curve(Y_CV, CV_pred)
    AUC_CV.append(auc(a_fpr_cv, a_tpr_cv))


for av in tqdm(C_Para):
    b = np.log10(av)
    L_C_Para.append(b)

# Performance of model on Train data and Test data for each hyper parameter.
plt.plot(L_C_Para, AUC_TR, label='AUC_Train',color='red',linewidth=2)
plt.scatter(L_C_Para, AUC_TR, label='AUC_Train',color='red',linewidth=2)
plt.gca()
plt.plot(L_C_Para, AUC_CV, label='AUC_CV',color='blue',linewidth=2)
plt.scatter(L_C_Para, AUC_CV, label='AUC_CV',color='blue',linewidth=2)
plt.gca()
plt.legend()
plt.xlabel("Log alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("AUC Values for respective alpha Values(Hyperparameter Tuning)")
plt.grid()
plt.show()
```

```
100%|██████████| 9/9 [01:08<00:00,  7.46s/it]
100%|██████████| 9/9 [00:00<00:00, 17839.67it/s]
```



OBSERVATION:

1. α value has been choose based on the alpha which maximize the AUC CV score.
2. In this case **α = 0.0001** is having the highest AUC CV score.
3. Hence **α = 0.0001** is choosen as a best α value.

In [0]:

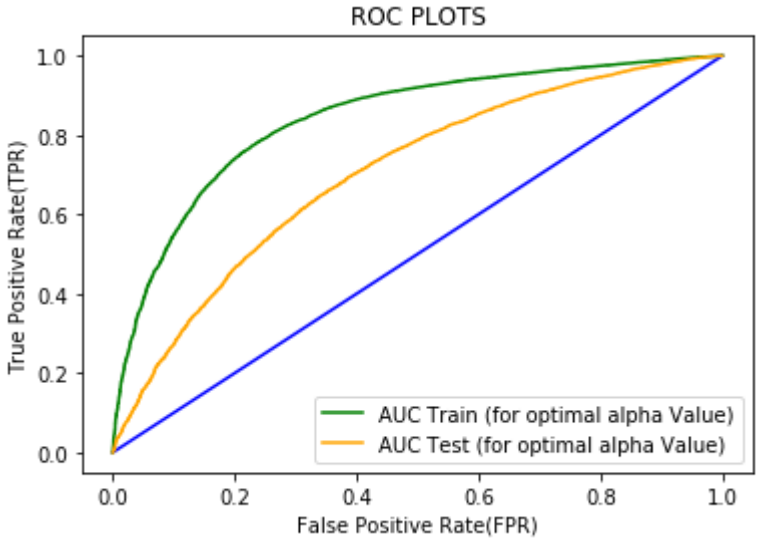
```
SGC_OPT=0.0001
FiveF_opt =SGDClassifier(penalty='l2', alpha=SGC_OPT, class_weight='balanced', n_jobs=-1)
FiveF_opt.fit(FiveF_Train, Y_Train)
pred = FiveF_opt.predict(FiveF_Test)
acc = accuracy_score(Y_Test, pred, normalize=True) * float(100)
print('\nTest accuracy for alpha Value = {0} is {1}%'.format(SGC_OPT,acc))

a_fpr_train, a_tpr_train, thresholds = roc_curve(Y_Train, FiveF_opt.decision_function(FiveF_TR_CSR))
a_fpr_Test, a_tpr_Test, thresholds = roc_curve(Y_Test, FiveF_opt.decision_function(FiveF_TS_CSR))
```

```
Test accuracy for alpha Value = 0.0001 is 73.23310773327417%
```

ROC PLOT FOR SELECTED FEATURES

In [0]:

```
%%time

#https://stackoverflow.com/questions/52910061/implementing-roc-curves-for-k-nn-machine-learning-algorithm-using-python-and-sci

plt.plot([0,1],[0,1],'k-', color='blue')
plt.plot(a_fpr_train, a_tpr_train, label="AUC Train (for optimal alpha Value)", color='green')
plt.plot(a_fpr_Test, a_tpr_Test, label="AUC Test (for optimal alpha Value)", color='orange')
plt.legend()
plt.ylabel("True Positive Rate(TPR)")
plt.xlabel("False Positive Rate(FPR)")
plt.title("ROC PLOTS")
plt.show()
print("-"*120)
print("AUC Train (for optimal alpha Value) =", auc(a_fpr_train, a_tpr_train))
print("AUC Test (for optimal alpha Value) =", auc(a_fpr_Test, a_tpr_Test))
FiveF_AOPT=SGC_OPT
FiveF_AUC=round(auc(a_fpr_Test, a_tpr_Test)*100)
pred9 = FiveF_opt.predict(FiveF_Train)
pred10 = FiveF_opt.predict(FiveF_Test)
```




```
------------------------------------------------------------------------------------------------------------------------
AUC Train (for optimal alpha Value) = 0.8395689555577046
AUC Test (for optimal alpha Value) = 0.7041598897018491
```

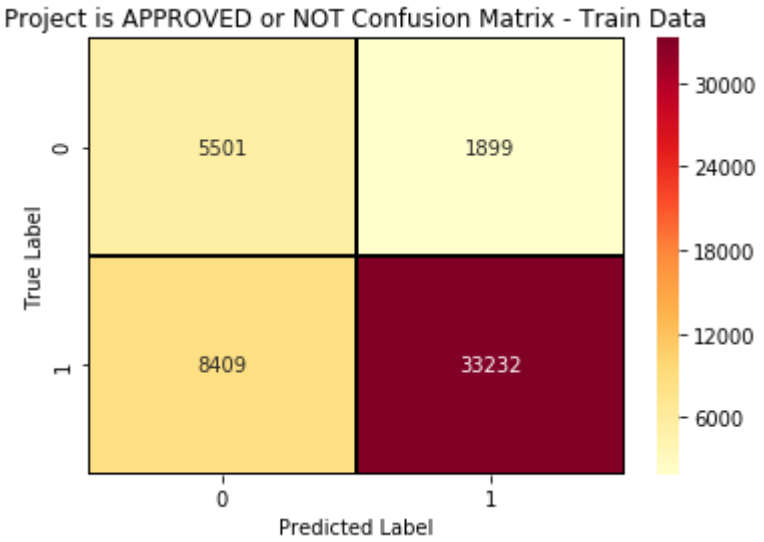
```
CPU times: user 4.48 s, sys: 7.29 ms, total: 4.49 s
Wall time: 4.48 s
```

CONFUSION MATRIX FOR SELECTED FEATURES

In [0]:

```
%%time
#https://seaborn.pydata.org/generated/seaborn.heatmap.html
#https://getaravind.com/blog/confusion-matrix-seaborn-heatmap/
%matplotlib inline
from sklearn.metrics import confusion_matrix
Train = confusion_matrix(Y_Train, pred9)
sns.heatmap(Train,annot=True,cbar=True,fmt='d',cmap='YlOrRd',linewidths=1,linecolor='black')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Project is APPROVED or NOT Confusion Matrix - Train Data')
```

```
CPU times: user 104 ms, sys: 45.9 ms, total: 150 ms
Wall time: 104 ms
```



OBSERVATION:

True Negative =5501; False Negative =8409; True Positive = 33232; False Positive = 1899

Accuracy (Overall, how often is the classifier correct) = 0.80

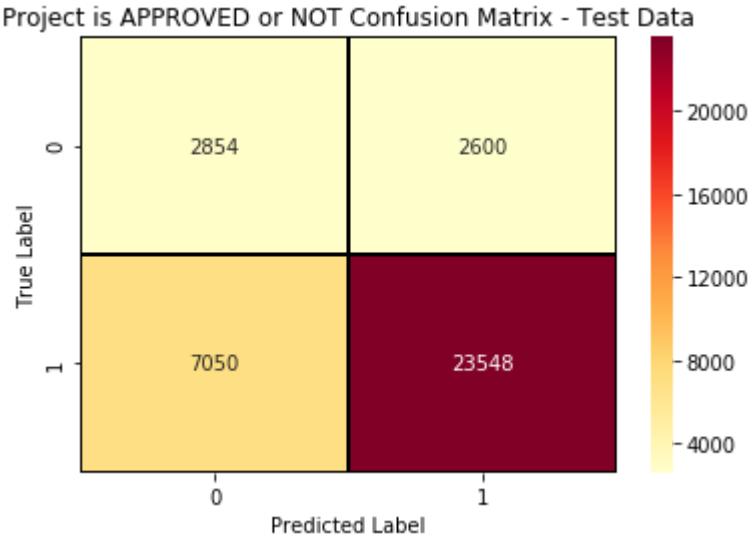
Precision(When it predicts yes, how often is it correct) =0.95

Misclassification (Overall, how often is it wrong) =0.22

In [0]:

```
%%time
#https://seaborn.pydata.org/generated/seaborn.heatmap.html
#https://getaravind.com/blog/confusion-matrix-seaborn-heatmap/
Test = confusion_matrix(Y_Test, pred10)
sns.heatmap(Test,annot=True,cbar=True,fmt='d',cmap='YlOrRd',linewidths=1,linecolor='black')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.title('Project is APPROVED or NOT Confusion Matrix - Test Data')
```

```
CPU times: user 92.8 ms, sys: 34.2 ms, total: 127 ms
Wall time: 82.9 ms
```



OBSERVATION:

True Negative =2854; False Negative =7050; True Positive = 23548; False Positive = 2600

Accuracy (Overall, how often is the classifier correct) = 0.74

Precision(When it predicts yes, how often is it correct) =0.91

Misclassification (Overall, how often is it wrong) =0.27

3. Conclusions

In [0]:

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

pt = PrettyTable()
pt.field_names= ("S.No","Vectorizer", "Model", "HyperParameter", "AUC")
pt.add_row(["1","BOW", "SVM",BOW_AOPT, BOW_AUC])
pt.add_row(["2","TFIDF", "SVM", TFIDF_AOPT, TFIDF_AUC])
pt.add_row(["3","AVG_W2V", "SVM",AVG_W2V_AOPT, AVG_W2V_AUC])
pt.add_row(["4","TFIDF_W2V", "SVM", TFIDF_W2V_AOPT, TFIDF_W2V_AUC])
pt.add_row(["5","SET 5", "SVM",FiveF_AOPT, FiveF_AUC])
print(pt)
```

```
+------+------------+-------+----------------+------+
| S.No | Vectorizer | Model | HyperParameter | AUC  |
+------+------------+-------+----------------+------+
|  1   |    BOW     |  SVM  |      0.01      | 72.0 |
|  2   |   TFIDF    |  SVM  |     0.0001     | 71.0 |
|  3   |  AVG_W2V   |  SVM  |     0.0001     | 70.0 |
|  4   | TFIDF_W2V  |  SVM  |     0.001      | 70.0 |
|  5   |   SET 5    |  SVM  |     0.0001     | 70.0 |
+------+------------+-------+----------------+------+
```

SUMMARY:

1.**"BOW and TFIDF"** performing very wellin acuuracy and precision scores.When compare to other vectorizer such as AVG_W2V and TFIDF_W2V.

2. Hyperparameter such as regularization and the number of iterations needs to be choose very carefully.

3. Linear SVM is fast when compare to the Kernal SVM.

4. For graphical representation applied Log function on alpha values.