Logistic Regression on 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.

In [1]:

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

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

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

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

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

from tqdm import tqdm
import os
import chart_studio.plotly as py

import chart_studio.plotly as py


from collections import Counter
```

1. LOAD AND PROCESS DATA

1.1 Reading Data

In [2]:

```
data=pd.read_csv("train_data.csv")
resource_data=pd.read_csv("resources.csv")
data.columns
```

Out[2]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category',
       'project_subject_categories', 'project_subject_subcategories',
       'project_title', 'project_essay_1', 'project_essay_2',
       'project_essay_3', 'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved'],
      dtype='object')
```

In [3]:

```
price_data=resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()
```

In [4]:

```
project_data=pd.merge(data, price_data, on='id', how='left')
```

In [5]:

```
project_data.columns
```

Out[5]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category',
       'project_subject_categories', 'project_subject_subcategories',
       'project_title', 'project_essay_1', 'project_essay_2',
       'project_essay_3', 'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'price', 'quantity'],
      dtype='object')
```

1.2 process Project Essay

In [6]:

```
project_data.head(3)
```

Out[6]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_category	pr
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	2016-12-05 13:43:57	Grades PreK-2	
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	2016-10-25 09:22:10	Grades 6-8	
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms.	AZ	2016-08-31 12:03:56	Grades 6-8	

In [7]:

```
project_data["essay"] = project_data["project_essay_1"].map(str) +\
                project_data["project_essay_2"].map(str) + \
                project_data["project_essay_3"].map(str) + \
                project_data["project_essay_4"].map(str)
```

In [8]:

```
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 [9]:

```
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', 'the:
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', ':
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do',
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while'
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'befor
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'aga
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each',
```

```
            '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',
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", '
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'w
            'won', "won't", 'wouldn', "wouldn't"]
```

In [10]:

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

```
100%|██████████| 109248/109248 [00:57<00:00, 1894.86it/s]
```

1.2 process Project Title

In [11]:

```
# https://stackoverflow.com/a/47091490/4084039
from tqdm import tqdm
preprocessed_title = []
# tqdm is for printing the status bar
for sentance in tqdm(data['project_title'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_title.append(sent.lower().strip())
project_data['cleaned_project_title']=preprocessed_title
```

```
100%|██████████| 109248/109248 [00:02<00:00, 38323.91it/s]
```

1.3 teacher_prefix

In [12]:

```
temp1=data.teacher_prefix.apply(lambda x: str(x).replace('.', ''))
project_data['teacher_prefix']=temp1
project_data['teacher_prefix'].value_counts()
```

Out[12]:

```
Mrs        57269
Ms         38955
Mr         10648
Teacher     2360
Dr            13
nan            3
Name: teacher_prefix, dtype: int64
```

1.4 project grade

In [13]:

```
project_data.project_grade_category.value_counts()
```

Out[13]:

```
Grades PreK-2    44225
Grades 3-5       37137
Grades 6-8       16923
Grades 9-12      10963
Name: project_grade_category, dtype: int64
```

In [14]:

```
grade_list=[]
for i in project_data['project_grade_category'].values:
    i=i.replace(' ','_')
    i=i.replace('-','_')
    grade_list.append(i.strip())
```

```
project_data['project_grade_category']=grade_list
```

In [15]:

```
project_data['project_grade_category'].value_counts()
```

Out[15]:

```
Grades_PreK_2    44225
Grades_3_5       37137
Grades_6_8       16923
Grades_9_12      10963
Name: project_grade_category, dtype: int64
```

1.5 project_subject_categories

In [16]:

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

# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string
# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python
cat_list = []
for i in catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & Hunger'
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Science"=> '
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i.e ren
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math & Science"=>"M
        temp+=j.strip()+" " #" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_') # we are replacing the & value into
    cat_list.append(temp.strip())

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

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

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

1.6 project_subject_subcategories

In [17]:

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

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

sub_cat_list = []
for i in sub_catogories:
    temp = ""
    # consider we have text like this "Math & Science, Warmth, Care & Hunger"
    for j in i.split(','): # it will split it in three parts ["Math & Science", "Warmth", "Care & Hunger'
        if 'The' in j.split(): # this will split each of the catogory based on space "Math & Science"=> '
            j=j.replace('The','') # if we have the words "The" we are going to replace it with ''(i.e ren
        j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:"Math & Science"=>"M
        temp +=j.strip()+" "#" abc ".strip() will return "abc", remove the trailing spaces
        temp = temp.replace('&','_')
    sub_cat_list.append(temp.strip())

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

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

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

1.7 counting words in title

In [18]:

```
#https://stackoverflow.com/questions/49984905/count-number-of-words-per-row
project_data['totalwords_title'] = project_data['cleaned_project_title'].str.split().str.len()
```

1.8 number of words in the essay

In [19]:

```
project_data['totalwords_essay'] = project_data['cleaned_essay'].str.split().str.len()
```

1.9 sentiment score's of each of the essay

In [20]:

```
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
analyser = SentimentIntensityAnalyzer()
neg=[]
compound=[]
pos=[]
neu=[]
for sent in (project_data['cleaned_essay'].values):
    score = analyser.polarity_scores(sent)
    neg.append(score.get('neg'))
    neu.append(score.get('neu'))
    pos.append(score.get('pos'))
    compound.append(score.get('compound'))
project_data['neg']=neg
project_data['neu']=neu
project_data['pos']=pos
project_data['compound']=compound
```

1.10 droping unnecesarry columns

In [21]:

```
project_data.drop(['project_title'], axis=1, inplace=True)
project_data.drop(['project_essay_1'], axis=1, inplace=True)
project_data.drop(['project_essay_2'], axis=1, inplace=True)
project_data.drop(['project_essay_3'], axis=1, inplace=True)
project_data.drop(['project_essay_4'], axis=1, inplace=True)
```

In [22]:

```
project_data.head(3)
```

Out[22]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_category	pr
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs	IN	2016-12-05 13:43:57	Grades_PreK_2	
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr	FL	2016-10-25 09:22:10	Grades_6_8	pr
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms	AZ	2016-08-31 12:03:56	Grades_6_8	

3 rows × 23 columns

1.11 Making dependant(label) and independant variables

In [23]:

```
y = project_data['project_is_approved'].values
project_data.drop(['project_is_approved'], axis=1, inplace=True)
project_data.head(1)
x=project_data
x.head(3)
```

Out[23]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	project_submitted_datetime	project_grade_category	pr
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs	IN	2016-12-05 13:43:57	Grades_PreK_2	
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr	FL	2016-10-25 09:22:10	Grades_6_8	pr
2	21895	p182444	3465aaf82da834c0582ebd0ef8040ca0	Ms	AZ	2016-08-31 12:03:56	Grades_6_8	

3 rows × 22 columns

1.12 Traing and Test split

In [24]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(x, y, test_size=0.33, stratify=y,random_state=42)
X_train, X_cv, Y_train, Y_cv = train_test_split(X_train, Y_train, test_size=0.33, stratify=Y_train,random
```

2.Text Vectorization and encoding catagories,normalization numerical features

2.1 converting the essay to vectors using BOW

In [25]:

```
print(X_train.shape, Y_train.shape)
print(X_cv.shape, Y_cv.shape)
print(X_test.shape, Y_test.shape)

print("="*100)

from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,2), max_features=5000)
vectorizer.fit(X_train['cleaned_essay'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_bow = vectorizer.transform(X_train['cleaned_essay'].values)
X_cv_essay_bow = vectorizer.transform(X_cv['cleaned_essay'].values)
X_test_essay_bow = vectorizer.transform(X_test['cleaned_essay'].values)

print("After vectorizations")
print(X_train_essay_bow.shape, Y_train.shape)
print(X_cv_essay_bow.shape, Y_cv.shape)
print(X_test_essay_bow.shape, Y_test.shape)
print("="*100)
```

```
(49041, 22) (49041,)
(24155, 22) (24155,)
(36052, 22) (36052,)
====================================================================================================
After vectorizations
(49041, 5000) (49041,)
(24155, 5000) (24155,)
(36052, 5000) (36052,)
====================================================================================================
```

2.2 converting the title to vectors using BOW

In [26]:

```
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,2), max_features=5000)
vectorizer.fit(X_train['cleaned_project_title'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_title_bow = vectorizer.transform(X_train['cleaned_project_title'].values)
X_cv_title_bow = vectorizer.transform(X_cv['cleaned_project_title'].values)
X_test_title_bow = vectorizer.transform(X_test['cleaned_project_title'].values)
```

```
print("After vectorizations")
print(X_train_title_bow.shape, Y_train.shape)
print(X_cv_title_bow.shape, Y_cv.shape)
print(X_test_title_bow.shape, Y_test.shape)
print("="*100)
```

```
After vectorizations
(49041, 3750) (49041,)
(24155, 3750) (24155,)
(36052, 3750) (36052,)
====================================================================================================
```

2.3 converting the essay to vectors using TFIDF

In [27]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,2), max_features=5000)
vectorizer.fit(X_train['cleaned_essay'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_essay_tfidf = vectorizer.transform(X_train['cleaned_essay'].values)
X_cv_essay_tfidf = vectorizer.transform(X_cv['cleaned_essay'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['cleaned_essay'].values)

print("After vectorizations")
print(X_train_essay_tfidf.shape, Y_train.shape)
print(X_cv_essay_tfidf.shape, Y_cv.shape)
print(X_test_essay_tfidf.shape, Y_test.shape)
print("="*100)
```

```
After vectorizations
(49041, 5000) (49041,)
(24155, 5000) (24155,)
(36052, 5000) (36052,)
====================================================================================================
```

2.4 converting the title to vectors using TFIDF

In [28]:

```
vectorizer = TfidfVectorizer(min_df=10)
vectorizer.fit(X_train['cleaned_project_title'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_title_tfidf = vectorizer.transform(X_train['cleaned_project_title'].values)
X_cv_title_tfidf = vectorizer.transform(X_cv['cleaned_project_title'].values)
X_test_title_tfidf = vectorizer.transform(X_test['cleaned_project_title'].values)

print("After vectorizations")
print(X_train_title_tfidf.shape, Y_train.shape)
print(X_cv_title_tfidf.shape, Y_cv.shape)
print(X_test_title_tfidf.shape, Y_test.shape)
print("="*100)
```

```
After vectorizations
(49041, 2080) (49041,)
(24155, 2080) (24155,)
(36052, 2080) (36052,)
====================================================================================================
```

2.5 load glove model for AvgW2V

load glove model

In [29]:

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

```
model = loadGloveModel('glove.42B.300d.txt')

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

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

```
764it [00:00, 7637.71it/s]
Loading Glove Model
1917495it [04:00, 7971.72it/s]
Done. 1917495  words loaded!
```

Out[29]:

```
'Output:\n    \nLoading Glove Model\n1917495it [06:32, 4879.69it/s]\nDone. 1917495  words loaded!\n'
```

In [30]:

```
words = []
for i in X_train['cleaned_essay'].values:
    words.extend(i.split(' '))

for i in X_train['cleaned_project_title'].values:
    words.extend(i.split(' '))
print("all the words in the coupus", len(words))
words = set(words)
print("the unique words in the coupus", len(words))

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

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


# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-and

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

```
all the words in the coupus 7628532
the unique words in the coupus 42937
The number of words that are present in both glove vectors and our coupus 39195 ( 91.285 %)
word 2 vec length 39195
```

In [31]:

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

2.6 Avg w2v on essay using glove model

In [32]:

```
Text_avg_w2v_train_essay= []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['cleaned_essay'].values): # 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
    Text_avg_w2v_train_essay.append(vector)

print(len(Text_avg_w2v_train_essay))
print(len(Text_avg_w2v_train_essay[0]))
```

```
100%|██████████| 49041/49041 [00:17<00:00, 2844.63it/s]
49041
300
```

In [33]:

```
Text_avg_w2v_cv_essay= []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['cleaned_essay'].values): # 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
    Text_avg_w2v_cv_essay.append(vector)

print(len(Text_avg_w2v_cv_essay))
print(len(Text_avg_w2v_cv_essay[0]))
```

```
100%|██████████| 24155/24155 [00:08<00:00, 2733.54it/s]
24155
300
```

In [34]:

```
Text_avg_w2v_test_essay= []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['cleaned_essay'].values): # 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
    Text_avg_w2v_test_essay.append(vector)

print(len(Text_avg_w2v_test_essay))
print(len(Text_avg_w2v_test_essay[0]))
```

```
100%|██████████| 36052/36052 [00:12<00:00, 2897.42it/s]
36052
300
```

2.7 Avg w2v on title using glove model

In [35]:

```
Text_avg_w2v_train_title= []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['cleaned_project_title'].values): # 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
    Text_avg_w2v_train_title.append(vector)

print(len(Text_avg_w2v_train_title))
print(len(Text_avg_w2v_train_title[0]))
```

```
100%|██████████| 49041/49041 [00:00<00:00, 66021.20it/s]
49041
300
```

In [36]:

```
Text_avg_w2v_cv_title= []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_cv['cleaned_project_title'].values): # 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
    Text_avg_w2v_cv_title.append(vector)

print(len(Text_avg_w2v_cv_title))
```

```
print(len(Text_avg_w2v_cv_title[0]))
```

```
100%|██████████| 24155/24155 [00:00<00:00, 60725.04it/s]
24155
300
```

In [37]:

```
Text_avg_w2v_test_title= []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['cleaned_project_title'].values): # 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
    Text_avg_w2v_test_title.append(vector)

print(len(Text_avg_w2v_test_title))
print(len(Text_avg_w2v_test_title[0]))
```

```
100%|██████████| 36052/36052 [00:00<00:00, 63421.00it/s]
36052
300
```

2.8 Using Pretrained Models: TFIDF weighted W2V on essay

In [38]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['cleaned_essay'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [39]:

```
Text_tfidf_w2v_train_essay= []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_train['cleaned_essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word),
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf va
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    Text_tfidf_w2v_train_essay.append(vector)

print(len(Text_tfidf_w2v_train_essay))
print(len(Text_tfidf_w2v_train_essay[0]))
```

```
100%|██████████| 49041/49041 [02:03<00:00, 397.06it/s]
49041
300
```

In [40]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_cv['cleaned_essay'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [41]:

```
Text_tfidf_w2v_cv_essay= [];
for sentence in tqdm(X_cv['cleaned_essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word),
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf va
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
```

```
    Text_tfidf_w2v_cv_essay.append(vector)

print(len(Text_tfidf_w2v_cv_essay))
print(len(Text_tfidf_w2v_cv_essay[0]))
```

```
100%|██████████| 24155/24155 [01:01<00:00, 393.66it/s]
24155
300
```

In [42]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_test['cleaned_essay'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [43]:

```
Text_tfidf_w2v_test_essay= [];
for sentence in tqdm(X_test['cleaned_essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word),
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf va
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    Text_tfidf_w2v_test_essay.append(vector)

print(len(Text_tfidf_w2v_test_essay))
print(len(Text_tfidf_w2v_test_essay[0]))
```

```
100%|██████████| 36052/36052 [01:31<00:00, 394.96it/s]
36052
300
```

2.9 TFIDF weighted W2V on title

In [44]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train['cleaned_project_title'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [45]:

```
Text_tfidf_w2v_train_title= [];
for sentence in tqdm(X_train['cleaned_project_title'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word),
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf va
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    Text_tfidf_w2v_train_title.append(vector)

print(len(Text_tfidf_w2v_train_title))
print(len(Text_tfidf_w2v_train_title[0]))
```

```
100%|██████████| 49041/49041 [00:01<00:00, 27214.41it/s]
49041
300
```

In [46]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_cv['cleaned_project_title'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [47]:

```
Text_tfidf_w2v_cv_title= [];
for sentence in tqdm(X_cv['cleaned_project_title'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word),
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf va
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    Text_tfidf_w2v_cv_title.append(vector)

print(len(Text_tfidf_w2v_cv_title))
print(len(Text_tfidf_w2v_cv_title[0]))
```

```
100%|██████████| 24155/24155 [00:00<00:00, 27138.71it/s]
24155
300
```

In [48]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_test['cleaned_essay'].values)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))
tfidf_words = set(tfidf_model.get_feature_names())
```

In [49]:

```
Text_tfidf_w2v_test_title= [];
for sentence in tqdm(X_test['cleaned_essay'].values): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word),
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf va
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    Text_tfidf_w2v_test_title.append(vector)

print(len(Text_tfidf_w2v_test_title))
print(len(Text_tfidf_w2v_test_title[0]))
```

```
100%|██████████| 36052/36052 [01:31<00:00, 393.79it/s]
36052
300
```

2.10 one hot encoding the catogorical features: teacher_prefix

In [50]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['teacher_prefix'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_ohe = vectorizer.transform(X_train['teacher_prefix'].values)
X_cv_teacher_ohe = vectorizer.transform(X_cv['teacher_prefix'].values)
X_test_teacher_ohe = vectorizer.transform(X_test['teacher_prefix'].values)

print("After vectorizations")
print(X_train_teacher_ohe.shape, Y_train.shape)
print(X_cv_teacher_ohe.shape, Y_cv.shape)
print(X_test_teacher_ohe.shape, Y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(49041, 6) (49041,)
(24155, 6) (24155,)
(36052, 6) (36052,)
['dr', 'mr', 'mrs', 'ms', 'nan', 'teacher']
====================================================================================================
```

2.11 one hot encoding the catogorical features: project Grade

In [51]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['project_grade_category'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_ohe = vectorizer.transform(X_train['project_grade_category'].values)
X_cv_grade_ohe = vectorizer.transform(X_cv['project_grade_category'].values)
X_test_grade_ohe = vectorizer.transform(X_test['project_grade_category'].values)

print("After vectorizations")
print(X_train_grade_ohe.shape, Y_train.shape)
print(X_cv_grade_ohe.shape, Y_cv.shape)
print(X_test_grade_ohe.shape, Y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(49041, 4) (49041,)
(24155, 4) (24155,)
(36052, 4) (36052,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
====================================================================================================
```

2.12 one hot encoding the catogorical features: state

In [52]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['school_state'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_state_ohe = vectorizer.transform(X_train['school_state'].values)
X_cv_state_ohe = vectorizer.transform(X_cv['school_state'].values)
X_test_state_ohe = vectorizer.transform(X_test['school_state'].values)

print("After vectorizations")
print(X_train_state_ohe.shape, Y_train.shape)
print(X_cv_state_ohe.shape, Y_cv.shape)
print(X_test_state_ohe.shape, Y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(49041, 51) (49041,)
(24155, 51) (24155,)
(36052, 51) (36052,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'ks', 'k
y', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'mo', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', 'nv', 'ny',
'oh', 'ok', 'or', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv', 'wy']
====================================================================================================
```

2.13 one hot encoding the catogorical features:clean_categories

In [53]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_categories'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_categories_ohe = vectorizer.transform(X_train['clean_categories'].values)
X_cv_clean_categories_ohe = vectorizer.transform(X_cv['clean_categories'].values)
X_test_clean_categories_ohe = vectorizer.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_clean_categories_ohe.shape, Y_train.shape)
print(X_cv_clean_categories_ohe.shape, Y_cv.shape)
print(X_test_clean_categories_ohe.shape, Y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(49041, 9) (49041,)
(24155, 9) (24155,)
(36052, 9) (36052,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language',
'math_science', 'music_arts', 'specialneeds', 'warmth']
====================================================================================================
```

2.14 one hot encoding the catogorical features:clean_subcategories

In [54]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_subcategories'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_subcategories_ohe = vectorizer.transform(X_train['clean_subcategories'].values)
X_cv_clean_subcategories_ohe = vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_clean_subcategories_ohe = vectorizer.transform(X_test['clean_subcategories'].values)

print("After vectorizations")
print(X_train_clean_subcategories_ohe.shape, Y_train.shape)
print(X_cv_clean_subcategories_ohe.shape, Y_cv.shape)
print(X_test_clean_subcategories_ohe.shape, Y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(49041, 30) (49041,)
(24155, 30) (24155,)
(36052, 30) (36052,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government', 'college_careerprep',
'communityservice', 'earlydevelopment', 'economics', 'environmentalscience', 'esl', 'extracurricular', 'f
inancialliteracy', 'foreignlanguages', 'gym_fitness', 'health_lifescience', 'health_wellness',
'history_geography', 'literacy', 'literature_writing', 'mathematics', 'music', 'nutritioneducation', 'ot
her', 'parentinvolvement', 'performingarts', 'socialsciences', 'specialneeds', 'teamsports', 'visualarts'
, 'warmth']
====================================================================================================
```

2.15 Normalizing the numerical features: Price

In [55]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

normalizer.fit(X_train['price'].values.reshape(-1,1))

X_train_price_norm = normalizer.transform(X_train['price'].values.reshape(-1,1))
X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(-1,1))
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_price_norm.shape, Y_train.shape)
print(X_cv_price_norm.shape, Y_cv.shape)
print(X_test_price_norm.shape, Y_test.shape)
print("="*100)
```

```
After vectorizations
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
====================================================================================================
```

2.16 Normalizing the numerical features:teacher_number_of_previously_posted_projects

In [56]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

X_train_TPPP_norm = normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].values.r
X_cv_TPPP_norm = normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].values.reshape
X_test_TPPP_norm = normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.res

print("After vectorizations")
print(X_train_TPPP_norm.shape, Y_train.shape)
print(X_cv_TPPP_norm.shape, Y_cv.shape)
print(X_test_TPPP_norm.shape, Y_test.shape)
print("="*100)
```

```
After vectorizations
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
====================================================================================================
```

2.17 Normalizing the numerical features: quantity

In [57]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

normalizer.fit(X_train['quantity'].values.reshape(-1,1))

X_train_quantity_norm = normalizer.transform(X_train['quantity'].values.reshape(-1,1))
X_cv_quantity_norm = normalizer.transform(X_cv['quantity'].values.reshape(-1,1))
X_test_quantity_norm = normalizer.transform(X_test['quantity'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_quantity_norm.shape, Y_train.shape)
print(X_cv_quantity_norm.shape, Y_cv.shape)
print(X_test_quantity_norm.shape, Y_test.shape)
print("="*100)
```

```
After vectorizations
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
====================================================================================================
```

2.18 Normalizing the numerical features: totalwords_title

In [58]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

normalizer.fit(X_train['totalwords_title'].values.reshape(-1,1))

X_train_totalwords_title_norm = normalizer.transform(X_train['totalwords_title'].values.reshape(-1,1))
X_cv_totalwords_title_norm = normalizer.transform(X_cv['totalwords_title'].values.reshape(-1,1))
X_test_totalwords_title_norm = normalizer.transform(X_test['totalwords_title'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_totalwords_title_norm.shape, Y_train.shape)
print(X_cv_totalwords_title_norm.shape, Y_cv.shape)
print(X_test_totalwords_title_norm.shape, Y_test.shape)
print("="*100)
```

```
After vectorizations
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
====================================================================================================
```

2.19 adding sentimental score: sentimental score of essay

In [59]:

```
X_train_essay_sentiment_neg = X_train['neg']
X_train_essay_sentiment_neu = X_train['neu']
X_train_essay_sentiment_pos = X_train['pos']
X_train_essay_sentiment_compound = X_train['compound']

X_cv_essay_sentiment_neg = X_cv['neg']
X_cv_essay_sentiment_neu = X_cv['neu']
X_cv_essay_sentiment_pos = X_cv['pos']
X_cv_essay_sentiment_compound = X_cv['compound']

X_test_essay_sentiment_neg = X_test['neg']
X_test_essay_sentiment_neu = X_test['neu']
X_test_essay_sentiment_pos = X_test['pos']
X_test_essay_sentiment_compound = X_test['compound']


print("After vectorizations")
print(X_train_essay_sentiment_neg.shape, Y_train.shape)
print(X_cv_essay_sentiment_neg.shape, Y_cv.shape)
print(X_test_essay_sentiment_neg.shape, Y_test.shape)
```

```
print("="*100)
```

```
After vectorizations
(49041,) (49041,)
(24155,) (24155,)
(36052,) (36052,)
====================================================================================================
```

2.20 Normalizing the numerical features: totalwords_essay

In [60]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

normalizer.fit(X_train['totalwords_essay'].values.reshape(-1,1))

X_train_totalwords_essay_norm = normalizer.transform(X_train['totalwords_essay'].values.reshape(-1,1))
X_cv_totalwords_essay_norm = normalizer.transform(X_cv['totalwords_essay'].values.reshape(-1,1))
X_test_totalwords_essay_norm = normalizer.transform(X_test['totalwords_essay'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_totalwords_essay_norm.shape, Y_train.shape)
print(X_cv_totalwords_essay_norm.shape, Y_cv.shape)
print(X_test_totalwords_essay_norm.shape, Y_test.shape)
print("="*100)
```

```
After vectorizations
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
====================================================================================================
```

3. Logistic Regression on BOW

3.1 BOW:Concatinating all the features

In [61]:

```
from scipy.sparse import hstack


X_tr_bow=hstack((X_train_state_ohe,X_train_clean_categories_ohe,X_train_clean_subcategories_ohe,X_train_g
X_cr_bow=hstack((X_cv_state_ohe,X_cv_clean_categories_ohe,X_cv_clean_subcategories_ohe,X_cv_grade_ohe,X_c
X_te_bow=hstack((X_test_state_ohe,X_test_clean_categories_ohe,X_test_clean_subcategories_ohe,X_test_grade
print("Final Data matrix")
print(X_tr_bow.shape, Y_train.shape)
print(X_cr_bow.shape, Y_cv.shape)
print(X_te_bow.shape, Y_test.shape)
print("="*100)
```

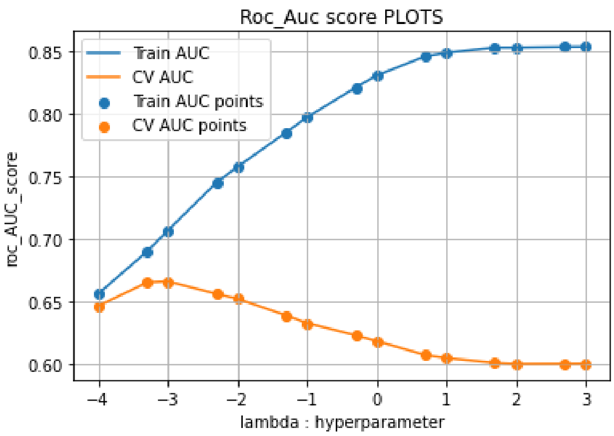
```
Final Data matrix
(49041, 8853) (49041,)
(24155, 8853) (24155,)
(36052, 8853) (36052,)
====================================================================================================
```

3.2 Hyper parameter Tuning:simple for loop for Train and cross validation

In [62]:

```
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score

train_auc = []
cv_auc = []

lambda_hyperparameter =[1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001]

for i in lambda_hyperparameter:

    model = LogisticRegression(C=i,solver='liblinear',random_state=0, class_weight='balanced')
    model.fit(X_tr_bow,Y_train)
    y_tr_prob = model.predict(X_tr_bow)
    y_cr_prob =model.predict(X_cr_bow)

    train_auc.append(roc_auc_score(Y_train,y_tr_prob))
```

```
    cv_auc.append(roc_auc_score(Y_cv,y_cr_prob))


plt.plot(np.log10(lambda_hyperparameter), train_auc, label='Train AUC')
plt.plot(np.log10(lambda_hyperparameter), cv_auc, label='CV AUC')

plt.scatter(np.log10(lambda_hyperparameter), train_auc, label='Train AUC points')
plt.scatter(np.log10(lambda_hyperparameter), cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("lambda : hyperparameter")
plt.ylabel("roc_AUC_score")
plt.title("Roc_Auc score PLOTS")
plt.grid()
plt.show()
```



Observations

1.By observing plot of auc score of train and cross validation we understand lambda=0.05 is best hyperparameter as cross validation auc is very high and does not cause overfit and underfit at lambda=0.05.

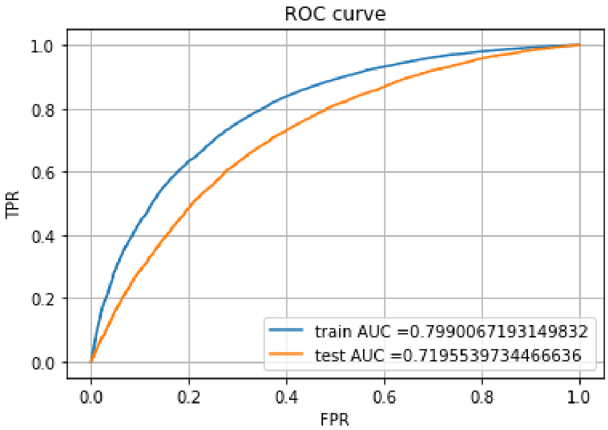
3.3 ROC curve with best lambda

In [63]:

```
def batch_predict(clf, data):


    data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000

    for i in range(0, tr_loop, 1000):
        data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])

    data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return data_pred
```

In [64]:

```
from sklearn.metrics import roc_curve, auc
lambda_bow=0.005

lambda_val = LogisticRegression(solver='liblinear',C=lambda_bow)
lambda_val.fit(X_tr_bow, Y_train)


y_train_pred = batch_predict(lambda_val, X_tr_bow)
y_test_pred = batch_predict(lambda_val, X_te_bow)

train_fpr, train_tpr, tr_thresholds = roc_curve(Y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(Y_test, y_test_pred)
auc_bow=auc(test_fpr, test_tpr)
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC curve")
plt.grid()
plt.show()
```



Observations

1.By looking ROC curve of Training FPR and TPR it looks sensible as it is greater than diagonal line or 0.5

2.By looking ROC curve of Test FPR and TPR is sensible .Model is generalize model

3.4 confusion matrix

In [65]:

```
# https://tatwan.github.io/How-To-Plot-A-Confusion-Matrix-In-Python/
def myplot_matrix1(data):
    plt.clf()
    plt.imshow(data, interpolation='nearest', cmap=plt.cm.Wistia)
    classNames = ['Negative','Positive']
    plt.title('Approved not approved matrix')
    tick_marks = np.arange(len(classNames))

    plt.xticks(tick_marks, classNames, rotation=45)
    plt.yticks(tick_marks, classNames)
    s = [['TN','FN'], ['FP', 'TP']]
    for i in range(2):
        for j in range(2):
            plt.text(j,i, str(s[i][j])+" = "+str(data[i][j]))
    plt.show()
```

In [66]:

```
def predict(proba, threshould, fpr, tpr):

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


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

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

In [67]:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

y_train_predicted_withthroshold=predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)
y_test_predicted_withthroshold=predict(y_test_pred, tr_thresholds, test_fpr, test_tpr)

cm_train=confusion_matrix(Y_train,y_train_predicted_withthroshold,labels=[0, 1])



print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(cm_train)
print("="*100)
```

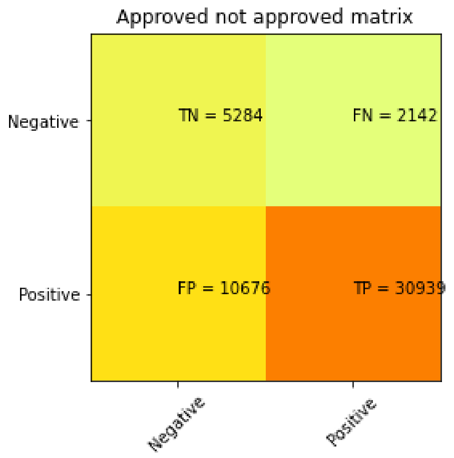
```
print("Accuracy score  for Train")
print(accuracy_score(Y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)))
print("="*100)

cm_test=confusion_matrix(Y_test,y_test_predicted_withthroshold,labels=[0, 1])

print("Test confusion matrix")
print(cm_test)
print("="*100)
print("Accuracy score  for Test")
accuracy_score_bow=accuracy_score(Y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr))
print(accuracy_score_bow)
print("="*100)
```

```
====================================================================================================
Train confusion matrix
[[ 5284  2142]
 [10676 30939]]
====================================================================================================
Accuracy score  for Train
0.7386268632368834
====================================================================================================
Test confusion matrix
[[ 3636  1823]
 [10193 20400]]
====================================================================================================
Accuracy score  for Test
0.6667036502829247
====================================================================================================
```

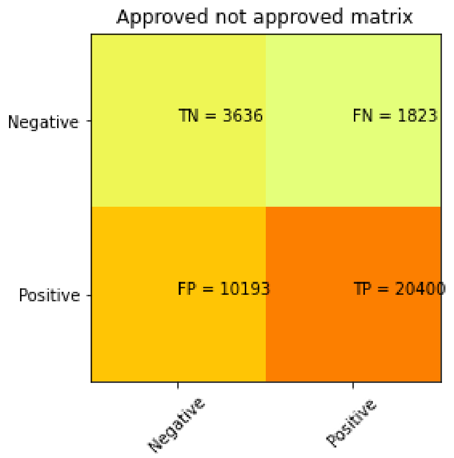
In [68]:

```
print("confusion matrix for train data")
print("="*100)
myplot_matrix1(cm_train)
print("confusion matrix for Test data")

print("="*100)
myplot_matrix1(cm_test)
```

```
confusion matrix for train data
====================================================================================================
```




```
confusion matrix for Test data
====================================================================================================
```



observations

1.TN and TP of train data and test data is higher.

2.Accuracy score on train data is 73% and test data is 66%.

3.TPR rate of test data is 91% .FPR rate of test data is 45%.TPR rate of test data is more than FPR rate of test data

4.TNR rate of testdata is 26% .FNR of test data is 8%.TNR rate of test data is more than FNR rate of test data.

5.TPR and TNR is higher than FPR and FNR so model is sensible for lambda=0.05.

4. Logistic Regression on TFIDF

4.1 TFIDF:Concatinating all the features

In [69]:

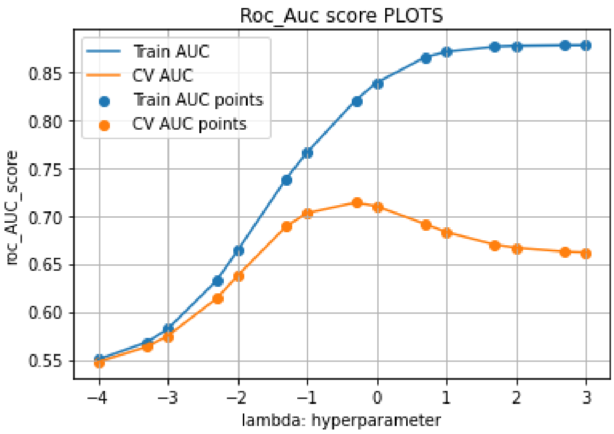
```
X_tr_tfidf=hstack((X_train_essay_tfidf,X_train_title_tfidf,X_train_state_ohe,X_train_clean_categories_ohe
X_cr_tfidf=hstack((X_cv_essay_tfidf,X_cv_title_tfidf,X_cv_state_ohe,X_cv_clean_categories_ohe,X_cv_clean_
X_te_tfidf=hstack((X_test_essay_tfidf,X_test_title_tfidf,X_test_state_ohe,X_test_clean_categories_ohe,X_t

print("Final Data matrix")
print(X_tr_tfidf.shape, Y_train.shape)
print(X_cr_tfidf.shape, Y_cv.shape)
print(X_te_tfidf.shape, Y_test.shape)
print("="*100)
```

```
Final Data matrix
(49041, 7183) (49041,)
(24155, 7183) (24155,)
(36052, 7183) (36052,)
====================================================================================================
```

4.2 Hyper parameter Tuning:simple for loop for Train and cross validation

In [70]:

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

train_auc = []
cv_auc = []


lambda_hyperparameter =[1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001]

for i in lambda_hyperparameter:
    model = LogisticRegression(C=i,solver='liblinear')
    model.fit(X_tr_tfidf,Y_train)
    y_tr_prob = batch_predict(model,X_tr_tfidf)
    y_cr_prob =batch_predict(model, X_cr_tfidf)

    train_auc.append(roc_auc_score(Y_train,y_tr_prob))
    cv_auc.append(roc_auc_score(Y_cv,y_cr_prob))


plt.plot(np.log10(lambda_hyperparameter), train_auc, label='Train AUC')
plt.plot(np.log10(lambda_hyperparameter), cv_auc, label='CV AUC')

plt.scatter(np.log10(lambda_hyperparameter), train_auc, label='Train AUC points')
plt.scatter(np.log10(lambda_hyperparameter), cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("lambda: hyperparameter")
plt.ylabel("roc_AUC_score")
plt.title("Roc_Auc score PLOTS")
plt.grid()
plt.show()
```



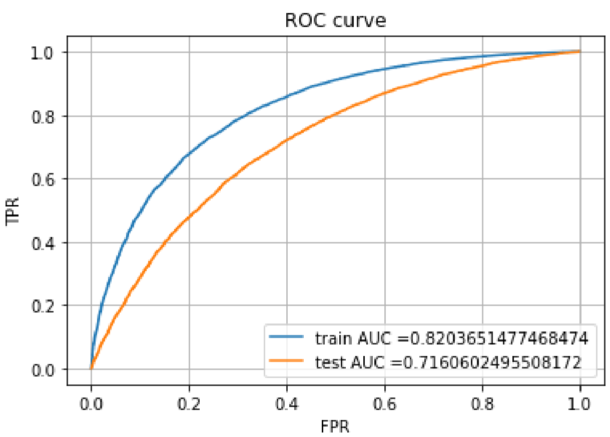
Observations

1.By observing plot of auc score of train and cross validation we understand lambda=0.5 is best hyperparameter as cross validation auc is very high and does not cause overfit and underfit at lambda=0.5.

4.3 ROC curve with best lambda

In [71]:

```
# by grid search
from sklearn.metrics import roc_curve, auc
lambda_tfidf=0.5

lambda_val = LogisticRegression(solver='liblinear',C=lambda_tfidf)
lambda_val.fit(X_tr_tfidf, Y_train)


y_train_pred = batch_predict(lambda_val, X_tr_tfidf)
y_test_pred = batch_predict(lambda_val, X_te_tfidf)

train_fpr, train_tpr, tr_thresholds = roc_curve(Y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(Y_test, y_test_pred)
auc_tfidf=auc(test_fpr, test_tpr)
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC curve")
plt.grid()
plt.show()
```



Observations

1.By looking ROC curve of Training FPR and TPR it looks sensible as it is greater than diagonal line or 0.5

2.By looking ROC curve of Test FPR and TPR is sensible .Model is generalize model

4.4 confusion matrix

In [72]:

```
from sklearn.metrics import accuracy_score
```

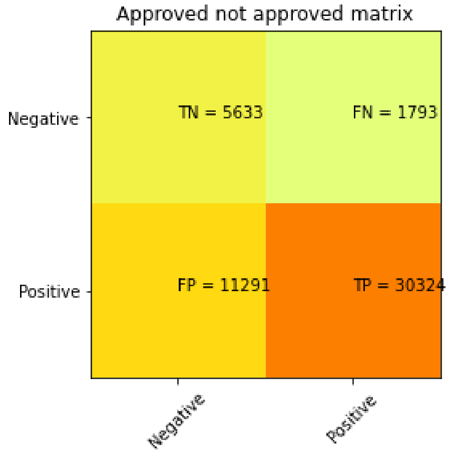
```
from sklearn.metrics import classification_report

y_train_predicted_withthroshold=predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)
y_test_predicted_withthroshold=predict(y_test_pred, tr_thresholds, test_fpr, test_tpr)

cm_train=confusion_matrix(Y_train,y_train_predicted_withthroshold,labels=[0, 1])



print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(cm_train)
print("="*100)
print("Accuracy score  for Train")
print(accuracy_score(Y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)))
print("="*100)

cm_test=confusion_matrix(Y_test,y_test_predicted_withthroshold,labels=[0, 1])

print("Test confusion matrix")
print(cm_test)
print("="*100)
print("Accuracy score  for Test")
accuracy_score_tfidf=accuracy_score(Y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr))
print(accuracy_score_tfidf)
print("="*100)
```

```
====================================================================================================
Train confusion matrix
[[ 5633  1793]
 [11291 30324]]
====================================================================================================
Accuracy score  for Train
0.7332028302848637
====================================================================================================
Test confusion matrix
[[ 3428  2031]
 [ 9454 21139]]
====================================================================================================
Accuracy score  for Test
0.6814323754576722
====================================================================================================
```

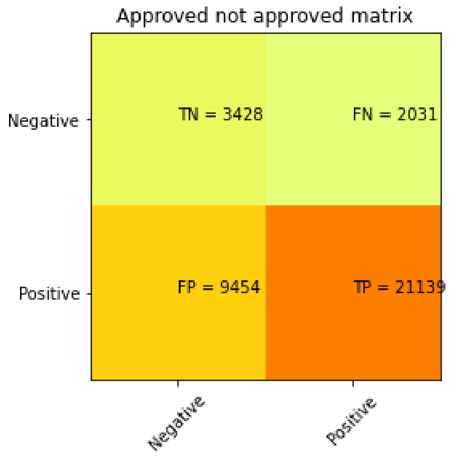
In [73]:

```
print("confusion matrix for train data")
print("="*100)
myplot_matrix1(cm_train)
print("confusion matrix for Test data")

print("="*100)
myplot_matrix1(cm_test)
```

```
confusion matrix for train data
====================================================================================================
```




```
confusion matrix for Test data
====================================================================================================
```



observations

1.TN and TP of train data and test data is higher.

2.Accuracy score on train data is 73% and test data is 68%.

3.TPR rate of test data is 91% .FPR rate of test data is 68%.TPR rate of test data is more than FPR rate of test data

4.TNR rate of testdata is 24% .FNR of test data is 8%.TNR rate of test data is more than FNR rate of test data.

5.TPR and TNR is higher than FPR and FNR so model is sensible for lambda=0.05.

5. Logistic Regression on AVGW2V

5.1 Avgw2v:Concatinating all the features

In [74]:

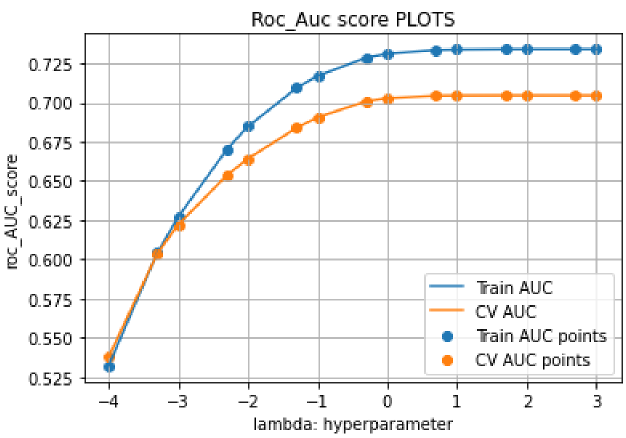
```
X_tr_avgw2v=hstack((Text_avg_w2v_train_essay,Text_avg_w2v_train_title,X_train_state_ohe,X_train_clean_cat
X_cr_avgw2v=hstack((Text_avg_w2v_cv_essay,Text_avg_w2v_cv_title,X_cv_state_ohe,X_cv_clean_categories_ohe,
X_te_avgw2v=hstack((Text_avg_w2v_test_essay,Text_avg_w2v_test_title,X_test_state_ohe,X_test_clean_categor

print("Final Data matrix")
print(X_tr_avgw2v.shape, Y_train.shape)
print(X_cr_avgw2v.shape, Y_cv.shape)
print(X_te_avgw2v.shape, Y_test.shape)
print("="*100)
```

```
Final Data matrix
(49041, 703) (49041,)
(24155, 703) (24155,)
(36052, 703) (36052,)
====================================================================================================
```

5.2 Hyper parameter Tuning:simple for loop for Train and cross validation

In [75]:

```
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_auc_score

train_auc = []
cv_auc = []


lambda_hyperparameter =[1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001]

for i in lambda_hyperparameter:
    model = LogisticRegression(C=i,solver='liblinear')
    model.fit(X_tr_avgw2v,Y_train)
    y_tr_prob = batch_predict(model,X_tr_avgw2v)
    y_cr_prob =batch_predict(model, X_cr_avgw2v)

    train_auc.append(roc_auc_score(Y_train,y_tr_prob))
    cv_auc.append(roc_auc_score(Y_cv,y_cr_prob))


plt.plot(np.log10(lambda_hyperparameter), train_auc, label='Train AUC')
plt.plot(np.log10(lambda_hyperparameter), cv_auc, label='CV AUC')

plt.scatter(np.log10(lambda_hyperparameter), train_auc, label='Train AUC points')
plt.scatter(np.log10(lambda_hyperparameter), cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("lambda: hyperparameter")
plt.ylabel("roc_AUC_score")
plt.title("Roc_Auc score PLOTS")
plt.grid()
plt.show()
```



Observations

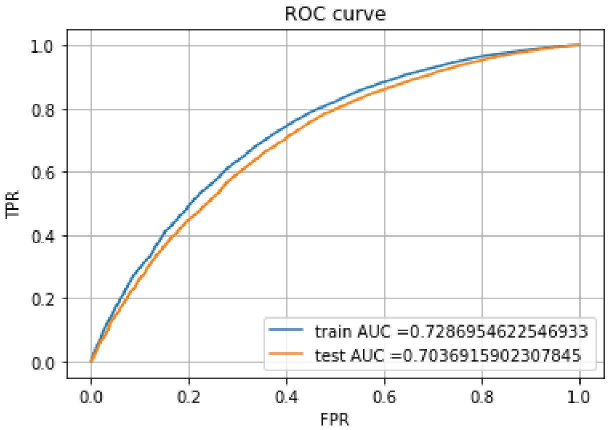
1.By observing plot of auc score of train and cross validation we understand lambda=0.5 is best hyperparameter as cross validation auc is very high and does not cause overfit and underfit at lambda=0.5.

5.3 ROC curve with best lambda

In [76]:

```
from sklearn.metrics import roc_curve, auc
lambda_avgw2v=0.5

lambda_val = LogisticRegression(solver='liblinear',C=lambda_avgw2v)
lambda_val.fit(X_tr_avgw2v, Y_train)


y_train_pred = batch_predict(lambda_val, X_tr_avgw2v)
y_test_pred = batch_predict(lambda_val, X_te_avgw2v)

train_fpr, train_tpr, tr_thresholds = roc_curve(Y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(Y_test, y_test_pred)
auc_avgw2v=auc(test_fpr, test_tpr)
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
```

```
plt.title("ROC curve")
plt.grid()
plt.show()
```



Observations

1.By looking ROC curve of Training FPR and TPR it looks sensible as it is greater than diagonal line or 0.5

2.By looking ROC curve of Test FPR and TPR is sensible .Model is generalize model

5.4 confusion matrix

In [77]:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

y_train_predicted_withthroshold=predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)
y_test_predicted_withthroshold=predict(y_test_pred, tr_thresholds, test_fpr, test_tpr)

cm_train=confusion_matrix(Y_train,y_train_predicted_withthroshold,labels=[0, 1])



print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(cm_train)
print("="*100)
print("Accuracy score  for Train")
print(accuracy_score(Y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)))
print("="*100)

cm_test=confusion_matrix(Y_test,y_test_predicted_withthroshold,labels=[0, 1])

print("Test confusion matrix")
print(cm_test)
print("="*100)
print("Accuracy score  for Test")
accuracy_score_avgw2v=accuracy_score(Y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr))
print(accuracy_score_avgw2v)
print("="*100)
```

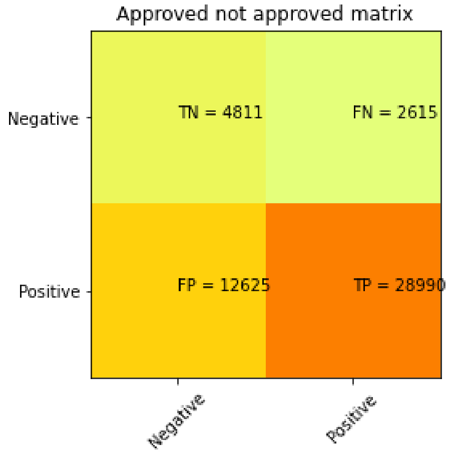
```
====================================================================================================
Train confusion matrix
[[ 4811  2615]
 [12625 28990]]
====================================================================================================
Accuracy score  for Train
0.689239615831651
====================================================================================================
Test confusion matrix
[[ 3932  1527]
 [13208 17385]]
====================================================================================================
Accuracy score  for Test
0.5912848108288028
====================================================================================================
```

In [78]:

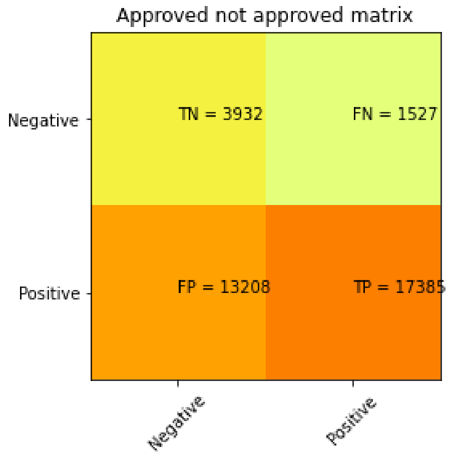
```
print("confusion matrix for train data")
```

```
print("="*100)
myplot_matrix1(cm_train)
print("confusion matrix for Test data")

print("="*100)
myplot_matrix1(cm_test)
```

```
confusion matrix for train data
====================================================================================================
```




```
confusion matrix for Test data
====================================================================================================
```



observations

1.TN and TP of train data and test data is higher.

2.Accuracy score on train data is 68% and test data is 59%.

3.TPR rate of test data is 91% .FPR rate of test data is 77%.TPR rate of test data is more than FPR rate of test data

4.TNR rate of testdata is 22% .FNR of test data is 8%.TNR rate of test data is more than FNR rate of test data.

5.TPR and TNR is higher than FPR and FNR so model is sensible for lambda=0.05.

6. Logistic Regression on TFIDF W2V

6.1 TFIDF:Concatinating all the features

In [79]:

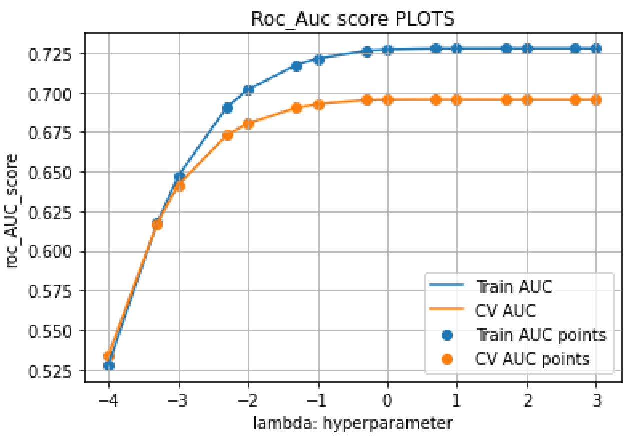
```
X_tr_tfidfw2v=hstack((Text_tfidf_w2v_train_essay,Text_tfidf_w2v_train_title,X_train_state_ohe,X_train_cle
X_cr_tfidfw2v=hstack((Text_tfidf_w2v_cv_essay,Text_tfidf_w2v_cv_title,X_cv_state_ohe,X_cv_clean_categorie
X_te_tfidfw2v=hstack((Text_tfidf_w2v_test_essay,Text_tfidf_w2v_test_title,X_test_state_ohe,X_test_clean_c

print("Final Data matrix")
print(X_tr_tfidfw2v.shape, Y_train.shape)
print(X_cr_tfidfw2v.shape, Y_cv.shape)
print(X_te_tfidfw2v.shape, Y_test.shape)
print("="*100)
```

```
Final Data matrix
(49041, 703) (49041,)
(24155, 703) (24155,)
(36052, 703) (36052,)
====================================================================================================
```

6.2 Hyper parameter Tuning:simple for loop for Train and cross validation

In [80]:

```
import matplotlib.pyplot as plt

from sklearn.metrics import roc_auc_score

train_auc = []
cv_auc = []


lambda_hyperparameter =[1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001]

for i in lambda_hyperparameter:
    model = LogisticRegression(C=i,solver='liblinear')
    model.fit(X_tr_tfidfw2v,Y_train)
    y_tr_prob = batch_predict(model,X_tr_tfidfw2v)
    y_cr_prob =batch_predict(model, X_cr_tfidfw2v)

    train_auc.append(roc_auc_score(Y_train,y_tr_prob))
    cv_auc.append(roc_auc_score(Y_cv,y_cr_prob))


plt.plot(np.log10(lambda_hyperparameter) , train_auc, label='Train AUC')
plt.plot(np.log10(lambda_hyperparameter) , cv_auc, label='CV AUC')

plt.scatter(np.log10(lambda_hyperparameter) , train_auc, label='Train AUC points')
plt.scatter(np.log10(lambda_hyperparameter) , cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("lambda: hyperparameter")
plt.ylabel("roc_AUC_score")
plt.title("Roc_Auc score PLOTS")
plt.grid()
plt.show()
```



Observations

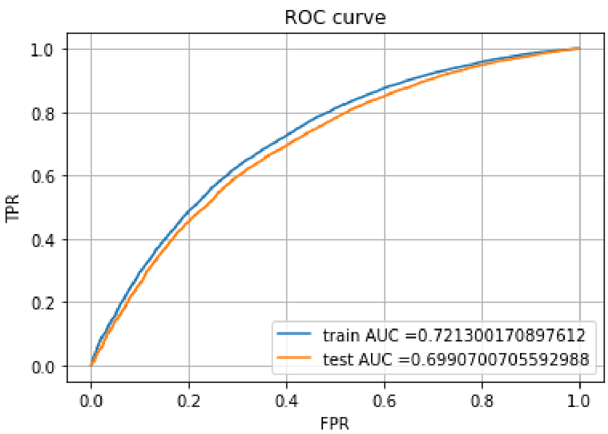
1.By observing plot of auc score of train and cross validation we understand lambda=0.1 is best hyperparameter as cross validation auc is very high and does not cause overfit and underfit at lambda=0.1.

6.3 ROC curve with best lambda

In [81]:

```
# by grid search
from sklearn.metrics import roc_curve, auc
lambda_tfidfw2v=0.1

lambda_val = LogisticRegression(solver='liblinear',C=lambda_tfidfw2v)
lambda_val.fit(X_tr_tfidfw2v, Y_train)


y_train_pred = batch_predict(lambda_val,X_tr_tfidfw2v)
```

```
y_test_pred = batch_predict(lambda_val,X_te_tfidfw2v)

train_fpr, train_tpr, tr_thresholds = roc_curve(Y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(Y_test, y_test_pred)
auc_tfidfw2v=auc(test_fpr, test_tpr)
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC curve")
plt.grid()
plt.show()
```



Observations

1.By looking ROC curve of Training FPR and TPR it looks sensible as it is greater than diagonal line or 0.1

2.By looking ROC curve of Test FPR and TPR is sensible .Model is generalize model

6.4 confusion matrix

In [82]:

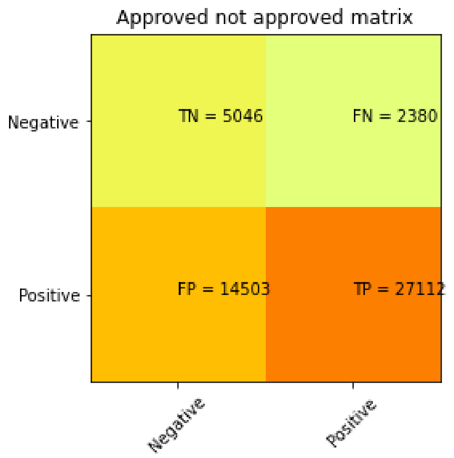
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

y_train_predicted_withthroshold=predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)
y_test_predicted_withthroshold=predict(y_test_pred, tr_thresholds, test_fpr, test_tpr)

cm_train=confusion_matrix(Y_train,y_train_predicted_withthroshold,labels=[0, 1])



print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(cm_train)
print("="*100)
print("Accuracy score  for Train")
print(accuracy_score(Y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)))
print("="*100)

cm_test=confusion_matrix(Y_test,y_test_predicted_withthroshold,labels=[0, 1])

print("Test confusion matrix")
print(cm_test)
print("="*100)
print("Accuracy score  for Test")
accuracy_score_tfidfw2v=accuracy_score(Y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr))
print(accuracy_score_tfidfw2v)
print("="*100)
```

```
====================================================================================================
Train confusion matrix
[[ 5046  2380]
 [14503 27112]]
====================================================================================================
Accuracy score  for Train
0.6557370363573337
====================================================================================================
Test confusion matrix
[[ 4077  1382]
 [14381 16212]]
====================================================================================================
Accuracy score  for Test
0.5627704426938867
====================================================================================================
```

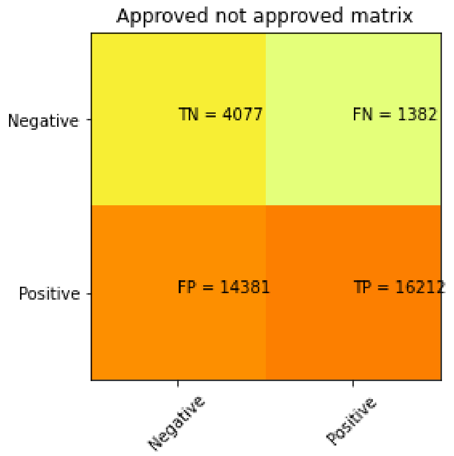
In [83]:

```
print("confusion matrix for train data")
print("="*100)
myplot_matrix1(cm_train)
print("confusion matrix for Test data")

print("="*100)
myplot_matrix1(cm_test)
```

```
confusion matrix for train data
====================================================================================================
```




```
confusion matrix for Test data
====================================================================================================
```



observations

1.TN and TP of train data and test data is higher.

2.Accuracy score on train data is 65% and test data is 56%.

3.TPR rate of test data is 91% .FPR rate of test data is 77%.TPR rate of test data is more than FPR rate of test data

4.TNR rate of testdata is 22% .FNR of test data is 7%.TNR rate of test data is more than FNR rate of test data.

5.TPR and TNR is higher than FPR and FNR so model is sensible for lambda=0.05.

7.Considering new features for analysis

7.1 Concatinating all the features

In [84]:

```
X_train_essay_sentiment_neg = X_train['neg'].values.reshape(-1,1)
X_train_essay_sentiment_neu = X_train['neu'].values.reshape(-1,1)
X_train_essay_sentiment_pos = X_train['pos'].values.reshape(-1,1)
X_train_essay_sentiment_compound = X_train['compound'].values.reshape(-1,1)

X_cv_essay_sentiment_neg = X_cv['neg'].values.reshape(-1,1)
X_cv_essay_sentiment_neu = X_cv['neu'].values.reshape(-1,1)
X_cv_essay_sentiment_pos = X_cv['pos'].values.reshape(-1,1)
X_cv_essay_sentiment_compound = X_cv['compound'].values.reshape(-1,1)

X_test_essay_sentiment_neg = X_test['neg'].values.reshape(-1,1)
X_test_essay_sentiment_neu = X_test['neu'].values.reshape(-1,1)
X_test_essay_sentiment_pos = X_test['pos'].values.reshape(-1,1)
X_test_essay_sentiment_compound = X_test['compound'].values.reshape(-1,1)


print("After vectorizations")
print(X_train_essay_sentiment_neg.shape, Y_train.shape)
print(X_cv_essay_sentiment_neg.shape, Y_cv.shape)
print(X_test_essay_sentiment_neg.shape, Y_test.shape)


print("="*100)
```

```
After vectorizations
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
====================================================================================================
```

In [85]:

```
from scipy.sparse import hstack
X_tr_newfeatures=hstack((X_train_state_ohe,X_train_clean_categories_ohe,X_train_clean_subcategories_ohe,X

#X_tr_newfeatures=hstack((X_train_state_ohe,X_train_clean_categories_ohe,X_train_clean_subcategories_ohe
X_te_newfeatures=hstack((X_test_state_ohe,X_test_clean_categories_ohe,X_test_clean_subcategories_ohe,X_te
X_cv_newfeatures=hstack((X_cv_state_ohe,X_cv_clean_categories_ohe,X_cv_clean_subcategories_ohe,X_cv_grade
print("Final Data matrix")
print(X_tr_newfeatures.shape, Y_train.shape)
print(X_te_newfeatures.shape, Y_cv.shape)
print(X_cv_newfeatures.shape, Y_test.shape)
print("="*100)
```

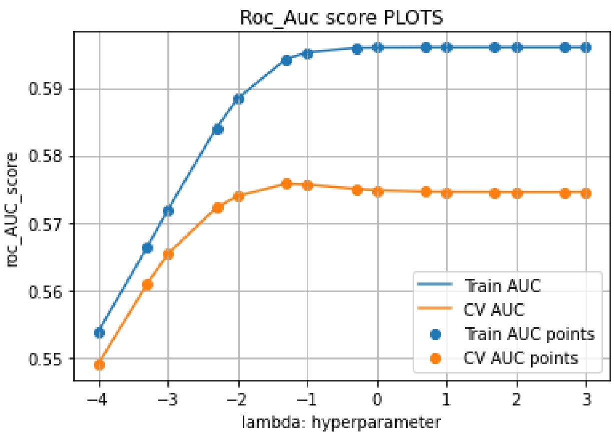
```
Final Data matrix
(49041, 109) (49041,)
(36052, 109) (24155,)
(24155, 109) (36052,)
====================================================================================================
```

7.2 Hyper parameter Tuning:simple for loop for Train and cross validation

In [86]:

```
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score

train_auc = []
cv_auc = []


lambda_hyperparameter =[1000,500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001]

for i in lambda_hyperparameter:
    model = LogisticRegression(C=i,solver='liblinear')
    model.fit(X_tr_newfeatures,Y_train)
    y_tr_prob = batch_predict(model,X_tr_newfeatures)
    y_cr_prob =batch_predict(model, X_cv_newfeatures)

    train_auc.append(roc_auc_score(Y_train,y_tr_prob))
    cv_auc.append(roc_auc_score(Y_cv,y_cr_prob))
```

```
plt.plot(np.log10(lambda_hyperparameter) , train_auc, label='Train AUC')
plt.plot(np.log10(lambda_hyperparameter) , cv_auc, label='CV AUC')

plt.scatter(np.log10(lambda_hyperparameter) , train_auc, label='Train AUC points')
plt.scatter(np.log10(lambda_hyperparameter) , cv_auc, label='CV AUC points')

plt.legend()
plt.xlabel("lambda: hyperparameter")
plt.ylabel("roc_AUC_score")
plt.title("Roc_Auc score PLOTS")
plt.grid()
plt.show()
```



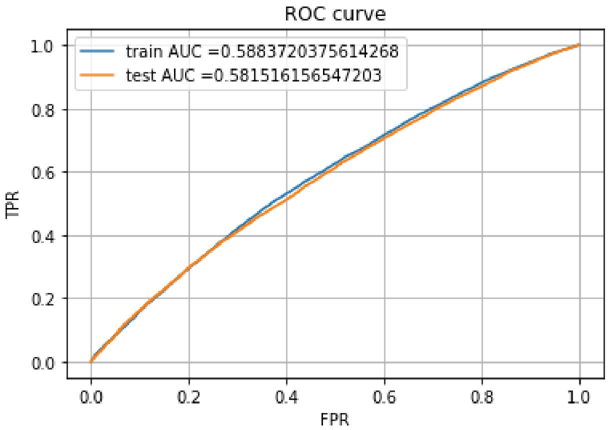
Observations

1.By observing plot of auc score of train and cross validation we understand lambda=0.01 is best hyperparameter as cross validation auc is very high and does not cause overfit and underfit at lambda=0.01.

7.3 ROC curve with best lambda

In [87]:

```
# by grid search
from sklearn.metrics import roc_curve, auc
lambda_newFetures=0.01

lambda_val = LogisticRegression(solver='liblinear',C=lambda_newFetures)
lambda_val.fit(X_tr_newfeatures, Y_train)


y_train_pred = batch_predict(lambda_val,X_tr_newfeatures)
y_test_pred = batch_predict(lambda_val,X_te_newfeatures)

train_fpr, train_tpr, tr_thresholds = roc_curve(Y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(Y_test, y_test_pred)
auc_newfeature=auc(test_fpr, test_tpr)
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC curve")
plt.grid()
plt.show()
```



Observations

1.By looking ROC curve of Training FPR and TPR it looks sensible as it is greater than diagonal line or 0.01

2.By looking ROC curve of Test FPR and TPR is sensible .Model is generalize model

7.4 confusion matrix

In [88]:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

y_train_predicted_withthroshold=predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)
y_test_predicted_withthroshold=predict(y_test_pred, tr_thresholds, test_fpr, test_tpr)

cm_train=confusion_matrix(Y_train,y_train_predicted_withthroshold,labels=[0, 1])



print("="*100)
from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(cm_train)
print("="*100)
print("Accuracy score  for Train")
print(accuracy_score(Y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_tpr)))
print("="*100)

cm_test=confusion_matrix(Y_test,y_test_predicted_withthroshold,labels=[0, 1])

print("Test confusion matrix")
print(cm_test)
print("="*100)
print("Accuracy score  for Test")
print(accuracy_score(Y_test, predict(y_test_pred, tr_thresholds, test_fpr, test_tpr)))
print("="*100)
```

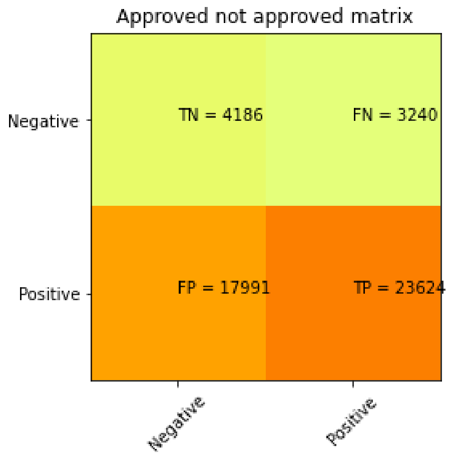
```
====================================================================================================
Train confusion matrix
[[ 4186  3240]
 [17991 23624]]
====================================================================================================
Accuracy score  for Train
0.5670765278032667
====================================================================================================
Test confusion matrix
[[ 3702  1757]
 [17326 13267]]
====================================================================================================
Accuracy score  for Test
0.47068123821147234
====================================================================================================
```

In [89]:

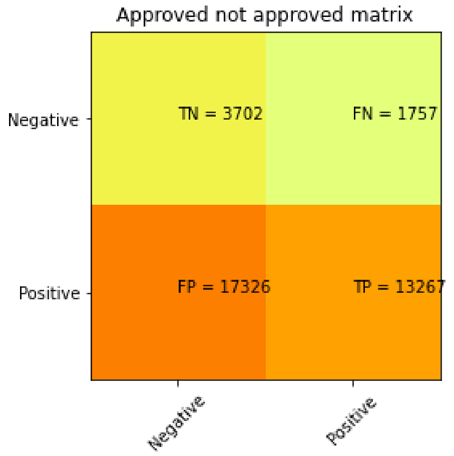
```
print("confusion matrix for train data")
print("="*100)
myplot_matrix1(cm_train)
print("confusion matrix for Test data")
```

```
print("="*100)
myplot_matrix1(cm_test)
```

```
confusion matrix for train data
====================================================================================================
```




```
confusion matrix for Test data
====================================================================================================
```



observations

1.TN and TP of train data and test data is higher.

2.Accuracy score on train data is 56% and test data is47%.

3.TPR rate of test data is 88% .FPR rate of test data is 20%.TPR rate of test data is more than FPR rate of test data

4.TNR rate of testdata is 96% .FNR of test data is 11%.TNR rate of test data is more than FNR rate of test data.

5.model is not sensible .

8.Model Performance Table

In [90]:

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

x.field_names = ["Vectorizer", "Hyper Parameter(lambda)", "AUC"]
x.add_row([" Logistic Regression with Bow",lambda_bow,auc_bow])
x.add_row([" Logistic Regression with TFIDF",lambda_tfidf,auc_tfidf])
x.add_row([" Logistic Regression with AVGW2V", lambda_avgw2v,auc_avgw2v])
x.add_row([" Logistic Regressionwith TFIDF W2V",lambda_tfidfw2v,auc_tfidfw2v])
x.add_row([" Logistic Regressionwith new features",lambda_newFetures,auc_newfeature])


print(x)
```

Vectorizer	Hyper Parameter(lambda)	AUC
Logistic Regression with Bow	0.005	0.7195539734466636
Logistic Regression with TFIDF	0.5	0.7160602495508172
Logistic Regression with AVGW2V	0.5	0.7036915902307845
Logistic Regressionwith TFIDF W2V	0.1	0.6990700705592988
Logistic Regressionwith new features	0.01	0.581516156547203

observations

1.By looking AUC score plots with diiferent lambda values we get best value for hyperparameter lambda=0.005

2.All Models performs good on training data but poor performence on unseen data(test data).