DonorsChoose

1. Importing All necessary LIbs to Work

In [100]:

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

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

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

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

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

from tqdm import tqdm
import os

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


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

2. Reading Data

In [101]:

```
project_data = pd.read_csv('train_data.csv', nrows=10000)
resource_data = pd.read_csv('resources.csv', nrows=10000)
```

In [102]:

```
data = project_data[['id', 'teacher_prefix', 'school_state', 'project_grade_category',
                     'project_subject_categories', 'project_subject_subcategories', 'project_title',
                     'project_essay_1', 'project_essay_2', 'project_essay_3', 'project_essay_4',
                     'teacher_number_of_previously_posted_projects', 'project_is_approved']].copy()


#data = resource_data[['quantity','price']].copy()
```

In [103]:

```
# I have to add to data frames to spilt it properly so i seached as below
# gSearch Key: df add column from other df
# https://stackoverflow.com/a/20603020/6000190
data = data.join(resource_data[['quantity','price']])
```

In [104]:

```
# Lets preprocess a little bit.. like creating one eassy and removing 4 parts of it.
# Dropping ID too.. Cause We might dont need id for anything.


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

In [105]:

```
# Just to ensure no empty data should pass on
data.fillna("", inplace=True)
```

In [106]:

```
data.head(2)
```

Out[106]:

	id	teacher_prefix	school_state	project_grade_category	project_subject_categories	project_subject_subcategories	project_title
0	p253737	Mrs.	IN	Grades PreK-2	Literacy & Language	ESL, Literacy	Educational Support for English Learners at Home
1	p258326	Mr.	FL	Grades 6-8	History & Civics, Health & Sports	Civics & Government, Team Sports	Wanted: Projector for Hungry Learners

In [107]:

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

```
Number of data points in train data (10000, 16)
--------------------------------------------------
The attributes of data : ['id' 'teacher_prefix' 'school_state' 'project_grade_category'
 'project_subject_categories' 'project_subject_subcategories'
 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'
 'project_essay_4' 'teacher_number_of_previously_posted_projects'
 'project_is_approved' 'quantity' 'price' 'essay']
```

In [ ]:

```

```

3. Text preprocessing

1.3.1 Essay Text

In [108]:

```
# 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 [109]:

```
# 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 [110]:

```
def preProcessTextData(feature):
    print("Preprocessing {}: ".format(feature))
    # Combining all the above statemennts
    processed_list = []
    # tqdm is for printing the status bar
    for sentance in tqdm(data['{}'.format(feature)].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)
        processed_list.append(sent.lower().strip())
    data['{}'.format(feature)] = processed_list
    print("="*10 +"  Preprocessing of {} 100 % Completed.".format(feature))
    return processed_list
```

In [111]:

```
def preProcessCategoricalTextData(feature):
    print("\033[0m Preprocessing {}: ".format(feature))
    listOfDataOfGivenFeature = list(data['{}'.format(feature)].values)

    cat_list = []
    for i in listOfDataOfGivenFeature:
        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 & Sc
ience"=> "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 & Sc
ience"=>"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())

    data['{}'.format(feature)] = cat_list
    print(" ")
    print("\033[0m Preprocessing of {} 100 % Completed.".format(feature))
    print("\033[0m preprocessed text of {}[0] is :: '\033[1m' {}".format(feature, cat_list[0]))
    print("="*10)
    return cat_list
```

In [112]:

```
# Polarity Score
# Searched for check of Empty String:
# https://stackoverflow.com/questions/9573244/how-to-check-if-the-string-is-empty

def replaceTextWithCompoundScore(feature):
    print("\033[0m Storing Polarity Score for {}: ".format(feature))
    ss = []
    sid = SentimentIntensityAnalyzer()
    for i in range(len(data['{}'.format(feature)])):
        singleLine = data['{}'.format(feature)][i]
        if ( (singleLine is None) or (str(singleLine).strip()=="") ):
            ss.append(0)
        else:
            ss.append(sid.polarity_scores(data['{}'.format(feature)][i])['compound'])


    data['{}'.format(feature)] = ss
    print("\033[0m Storing Polarity Score for {} 100 % Completed.".format(feature))
    print("\033[0m Coumpound score of {}[0] is :: '\033[1m' {}".format(feature, ss[0]))
    print("="*10)
    return ss
```

In [113]:

```
# Essay
processedList_essay = preProcessTextData('essay')
```

```
  1%|          | 90/10000 [00:00<00:11, 898.13it/s]
```

```
Preprocessing essay:
```

```
100%|██████████| 10000/10000 [00:08<00:00, 1222.41it/s]
```

```
==========  Preprocessing of essay 100 % Completed.
```

In [114]:

```
# Project Title
processedList_title = preProcessTextData('project_title')
```

```
 21%|██▏         | 2138/10000 [00:00<00:00, 17673.21it/s]
```

```
Preprocessing project_title:
```

```
100%|██████████| 10000/10000 [00:00<00:00, 21888.59it/s]
```

```
==========  Preprocessing of project_title 100 % Completed.
```

In [115]:

```
#Shape of Processed Text data
print("Shape of essay data: {}".format(np.array(processedList_essay).shape))
print("Shape of Title data: {}".format(np.array(processedList_title).shape))
```

```
Shape of essay data: (10000,)
Shape of Title data: (10000,)
```

In [116]:

```
# Visualize processed data

print("After Processed Essay Data")
print("="*10)
print(""+processedList_essay[0])
print("="*100)
print(" ")

print("After Processed Project Title Data")
print("="*10)
print(""+processedList_title[0])
print("="*100)
print(" ")
```

```
After Processed Essay Data
==========
my students english learners working english second third languages we melting pot refugees immigr
ants native born americans bringing gift language school we 24 languages represented english learn
er program students every level mastery we also 40 countries represented families within school ea
ch student brings wealth knowledge experiences us open eyes new cultures beliefs respect the limit
s language limits world ludwig wittgenstein our english learner strong support system home begs re
sources many times parents learning read speak english along side children sometimes creates
barriers parents able help child learn phonetics letter recognition reading skills by providing dv
d players students able continue mastery english language even no one home able assist all familie
s students within level 1 proficiency status offered part program these educational videos
specially chosen english learner teacher sent home regularly watch the videos help child develop e
arly reading skills parents not access dvd player opportunity check dvd player use year the plan u
se videos educational dvd years come el students nannan
====================================================================================================


After Processed Project Title Data
==========
educational support english learners home
====================================================================================================
```

In [117]:

```
# Project Subject Categories
processedList_project_subject_categories =
preProcessCategoricalTextData('project_subject_categories')

# Project Subject Subcategories
processedList_project_subject_subcategories =
```

```
preprocessCategoricalTextData('project_subject_subcategories')
```

```
 Preprocessing project_subject_categories:

 Preprocessing of project_subject_categories 100 % Completed.
 preprocessed text of project_subject_categories[0] is :: '' Literacy_Language
==========
 Preprocessing project_subject_subcategories:

 Preprocessing of project_subject_subcategories 100 % Completed.
 preprocessed text of project_subject_subcategories[0] is :: '' ESL Literacy
==========
```

In [118]:

```
# # Feature Set for Set 5 Task 2. Will not going to use for other computation

essay_1 = replaceTextWithCompoundScore('project_essay_1')

essay_2 = replaceTextWithCompoundScore('project_essay_2')

essay_3 = replaceTextWithCompoundScore('project_essay_3')

essay_4 = replaceTextWithCompoundScore('project_essay_4')
```

```
 Storing Polarity Score for project_essay_1:
 Storing Polarity Score for project_essay_1 100 % Completed.
 Coumpound score of project_essay_1[0] is :: '' 0.8481
==========
 Storing Polarity Score for project_essay_2:
 Storing Polarity Score for project_essay_2 100 % Completed.
 Coumpound score of project_essay_2[0] is :: '' 0.9242
==========
 Storing Polarity Score for project_essay_3:
 Storing Polarity Score for project_essay_3 100 % Completed.
 Coumpound score of project_essay_3[0] is :: '' 0
==========
 Storing Polarity Score for project_essay_4:
 Storing Polarity Score for project_essay_4 100 % Completed.
 Coumpound score of project_essay_4[0] is :: '' 0
==========
```

4. Spliting Data

In [119]:

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

Out[119]:

	id	teacher_prefix	school_state	project_grade_category	project_subject_categories	project_subject_subcategories	project_title
0	p253737	Mrs.	IN	Grades PreK-2	Literacy_Language	ESL Literacy	educational support english learners home

In [120]:

```
X = data
```

In [121]:

```
# train test split
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)
```

```
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)
```

5. Vectorizing Data

In [122]:

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

In [123]:

```
# Vectorizing Text Data
def text2VecBOW(feature):
    vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000) # ,
max_features=5000
    vectorizer.fit(X_train['{}'.format(feature)].values) # fit has to happen only on train data

    # we use the fitted CountVectorizer to convert the text to vector
    X_train_list_bow = vectorizer.transform(X_train['{}'.format(feature)].values)
    X_cv_list_bow = vectorizer.transform(X_cv['{}'.format(feature)].values)
    X_test_list_bow = vectorizer.transform(X_test['{}'.format(feature)].values)

    print("After vectorizations")
    print(X_train_list_bow.shape, y_train.shape)
    print(X_cv_list_bow.shape, y_cv.shape)
    print(X_test_list_bow.shape, y_test.shape)
    print("="*10)
    print(X_train_list_bow.toarray()[:2,:2])
    print(X_cv_list_bow.toarray()[:2,:2])
    print(X_test_list_bow.toarray()[:2,:2])

    return X_train_list_bow, X_cv_list_bow, X_test_list_bow
```

In [124]:

```
# Vectorizing Categorical Data
def cat2Vec(feature, vocabularyData=None):
    vectorizer = CountVectorizer(vocabulary=vocabularyData, lowercase=False)
    vectorizer.fit(X_train['{}'.format(feature)].values) # fit has to happen only on train data

    # we use the fitted CountVectorizer to convert the text to vector
    X_train_ohe = vectorizer.transform(X_train['{}'.format(feature)].values)
    X_cv_ohe = vectorizer.transform(X_cv['{}'.format(feature)].values)
    X_test_ohe = vectorizer.transform(X_test['{}'.format(feature)].values)

    print("After vectorizations")
    print(X_train_ohe.shape, y_train.shape)
    print(X_cv_ohe.shape, y_cv.shape)
    print(X_test_ohe.shape, y_test.shape)
    print(vectorizer.get_feature_names())
    print("="*10)
    print(X_train_ohe.toarray()[:2,:2])
    print(X_cv_ohe.toarray()[:2,:2])
    print(X_test_ohe.toarray()[:2,:2])
    return X_train_ohe, X_cv_ohe, X_test_ohe
```

In [125]:

```
# Vectorizing Numerical Data
def num2Vec(feature):
    normalizer = Normalizer()

    normalizer.fit(X_train['{}'.format(feature)].values.reshape(1,-1))

    X_train_num_norm = normalizer.transform(X_train['{}'.format(feature)].values.reshape(-1,1))
    X_cv_num_norm = normalizer.transform(X_cv['{}'.format(feature)].values.reshape(-1,1))
    X_test_num_norm = normalizer.transform(X_test['{}'.format(feature)].values.reshape(-1,1))

    print("After vectorizations")
    print(X_train_num_norm.shape, y_train.shape)
    print(X_cv_num_norm.shape, y_cv.shape)
```

```
    print(X_cv_num_norm.shape, y_cv.shape)
    print(X_test_num_norm.shape, y_test.shape)
    print(X_train_num_norm)
    print("="*10)
#     print(X_train_num_norm.toarray()[:2,:2])
#     print(X_cv_num_norm.toarray()[:2,:2])
#     print(X_test_num_norm.toarray()[:2,:2])
    return X_train_num_norm, X_cv_num_norm, X_test_num_norm
```

In [126]:

```
# Text Data
```

In [127]:

```
# Vectorizing Essay
X_train_essay_bow, X_cv_essay_bow, X_test_essay_bow = text2VecBOW('essay')
```

```
After vectorizations
(4489, 5000) (4489,)
(2211, 5000) (2211,)
(3300, 5000) (3300,)
==========
[[0 0]
 [0 0]]
[[0 0]
 [0 0]]
[[0 0]
 [0 0]]
```

In [128]:

```
# Vectorizing Title
X_train_title_bow, X_cv_title_bow, X_test_title_bow = text2VecBOW('project_title')
```

```
After vectorizations
(4489, 407) (4489,)
(2211, 407) (2211,)
(3300, 407) (3300,)
==========
[[0 0]
 [0 0]]
[[0 0]
 [0 0]]
[[0 0]
 [0 0]]
```

In [ ]:

```
```

In [129]:

```
# Vectorizing School State
X_train_school_ohe, X_cv_school_ohe, X_test_school_ohe = cat2Vec('school_state')
```

```
After vectorizations
(4489, 51) (4489,)
(2211, 51) (2211,)
(3300, 51) (3300,)
['AK', 'AL', 'AR', 'AZ', 'CA', 'CO', 'CT', 'DC', 'DE', 'FL', 'GA', 'HI', 'IA', 'ID', 'IL', 'IN', 'K
S', 'KY', 'LA', 'MA', 'MD', 'ME', 'MI', 'MN', 'MO', 'MS', 'MT', 'NC', 'ND', 'NE', 'NH', 'NJ', 'NM',
'NV', 'NY', 'OH', 'OK', 'OR', 'PA', 'RI', 'SC', 'SD', 'TN', 'TX', 'UT', 'VA', 'VT', 'WA', 'WI', 'WV
', 'WY']
==========
[[0 0]
 [0 0]]
[[0 0]
 [0 0]]
```

```
[[0 0]
 [0 0]]
```

In [130]:

```
# Vectorizing Project Grade Category

my_counter_project_grade = Counter()
for word in X_train['project_grade_category'].values:
        my_counter_project_grade.update(word.split(","))

cat_dict_procat = dict(my_counter_project_grade)
sorted_procat = dict(sorted(cat_dict_procat.items(), key=lambda kv: kv[1]))

vocabulary=list(sorted_procat.keys())

X_train_pgc_ohe, X_cv_pgc_ohe, X_test_pgc_ohe = cat2Vec('project_grade_category', vocabularyData=vo
cabulary)
```

```
After vectorizations
(4489, 4) (4489,)
(2211, 4) (2211,)
(3300, 4) (3300,)
['Grades 9-12', 'Grades 6-8', 'Grades 3-5', 'Grades PreK-2']
==========
[[0 0]
 [0 0]]
[[0 0]
 [0 0]]
[[0 0]
 [0 0]]
```

In [131]:

```
# teacher_prefix

X_train_tp_ohe, X_cv_tp_ohe, X_test_tp_ohe = cat2Vec('teacher_prefix')
```

```
After vectorizations
(4489, 4) (4489,)
(2211, 4) (2211,)
(3300, 4) (3300,)
['Mr', 'Mrs', 'Ms', 'Teacher']
==========
[[0 1]
 [0 0]]
[[0 1]
 [0 1]]
[[0 0]
 [0 1]]
```

In [132]:

```
# Vectorizing Project Subject Category

my_counter_project_grade = Counter()
for word in X_train['project_subject_categories'].values:
        my_counter_project_grade.update(word.split("_"))

cat_dict_procat = dict(my_counter_project_grade)
sorted_procat = dict(sorted(cat_dict_procat.items(), key=lambda kv: kv[1]))

vocabulary=list(sorted_procat.keys())

X_train_psc_ohe, X_cv_psc_ohe, X_test_psc_ohe = cat2Vec('project_subject_categories',
vocabularyData=vocabulary)
```

```
After vectorizations
(4489, 51) (4489,)
(2211, 51) (2211,)
(3300, 51) (3300,)
['Arts SpecialNeeds', 'Arts AppliedLearning', 'Sports History', 'Arts Warmth Care', 'Arts
```

```
History', 'Science Warmth Care', 'Language Warmth Care', 'AppliedLearning Warmth Care', 'SpecialNe
eds Health', 'Civics AppliedLearning', 'Sports AppliedLearning', 'AppliedLearning History',
'Civics SpecialNeeds', 'Sports Music', 'Sports Math', 'SpecialNeeds Music', 'Civics Math', 'Civics
Music', 'Science Health', 'Language AppliedLearning', 'Science History', 'AppliedLearning Health',
'Language History', 'Sports Literacy', 'AppliedLearning Music', 'AppliedLearning Math', 'Warmth Ca
re', 'Science AppliedLearning', 'Sports SpecialNeeds', 'Hunger', 'Science Music', 'Civics
Literacy', 'Language Music', 'AppliedLearning SpecialNeeds', 'AppliedLearning Literacy', 'Science
SpecialNeeds', 'Science Literacy', 'Civics', 'AppliedLearning', 'Language SpecialNeeds',
'SpecialNeeds', 'History', 'Music', 'Arts', 'Sports', 'Health', 'Language Math', 'Math',
'Language', 'Science', 'Literacy']
==========
[[0 0]
 [0 0]]
[[0 0]
 [0 0]]
[[0 0]
 [0 0]]
```

In [ ]:

```
```

In [133]:

```
# # Vectorizing Project Subject Sub Category

my_counter_project_grade = Counter()
for word in X_train['project_subject_subcategories'].values:
        my_counter_project_grade.update(word.split(" "))

cat_dict_procat = dict(my_counter_project_grade)
sorted_procat = dict(sorted(cat_dict_procat.items(), key=lambda kv: kv[1]))

vocabulary=list(sorted_procat.keys())

X_train_pssc_ohe, X_cv_pssc_ohe, X_test_pssc_ohe = cat2Vec('project_subject_subcategories',
vocabularyData=vocabulary)
```

```
After vectorizations
(4489, 30) (4489,)
(2211, 30) (2211,)
(3300, 30) (3300,)
['FinancialLiteracy', 'Economics', 'CommunityService', 'Civics_Government', 'Extracurricular',
'ParentInvolvement', 'ForeignLanguages', 'NutritionEducation', 'Warmth', 'Care_Hunger',
'SocialSciences', 'CharacterEducation', 'PerformingArts', 'TeamSports', 'Other', 'Music',
'College_CareerPrep', 'History_Geography', 'ESL', 'Health_LifeScience', 'EarlyDevelopment',
'Gym_Fitness', 'EnvironmentalScience', 'VisualArts', 'Health_Wellness', 'AppliedSciences',
'SpecialNeeds', 'Literature_Writing', 'Mathematics', 'Literacy']
==========
[[0 0]
 [0 0]]
[[0 0]
 [0 0]]
[[0 0]
 [0 0]]
```

In [ ]:

```
```

In [134]:

```
# Normalizing Numerical Data
X_train_noppp_norm, X_cv_noppp_norm, X_test_noppp_norm =
num2Vec('teacher_number_of_previously_posted_projects')
```

```
After vectorizations
(4489, 1) (4489,)
(2211, 1) (2211,)
(3300, 1) (3300,)
[[1.]
 [1.]
```

```
 [1.]
 ...
 [0.]
 [0.]
 [1.]]
==========
```

In [135]:

```
# Normalizing Numerical Data
X_train_q_norm, X_cv_q_norm, X_test_q_norm = num2Vec('quantity')
```

```
After vectorizations
(4489, 1) (4489,)
(2211, 1) (2211,)
(3300, 1) (3300,)
[[1.]
 [1.]
 [1.]
 ...
 [1.]
 [1.]
 [1.]]
==========
```

In [136]:

```
# Normalizing Numerical Data
X_train_price_norm, X_cv_price_norm, X_test_price_norm = num2Vec('price')
```

```
After vectorizations
(4489, 1) (4489,)
(2211, 1) (2211,)
(3300, 1) (3300,)
[[1.]
 [1.]
 [1.]
 ...
 [1.]
 [1.]
 [1.]]
==========
```

In [ ]:

In [137]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039

from scipy.sparse import hstack
X_tr = hstack(
    (X_train_essay_bow,
     X_train_title_bow,
     X_train_school_ohe,
     X_train_pgc_ohe,
     X_train_tp_ohe,
     X_train_psc_ohe,
     X_train_pssc_ohe,
     X_train_noppp_norm,
     X_train_q_norm,
     X_train_price_norm)).tocsr()


X_cr = hstack(
    (X_cv_essay_bow,
     X_cv_title_bow,
     X_cv_school_ohe,
     X_cv_pgc_ohe,
     X_cv_tp_ohe,
     X_cv_psc_ohe,
```

```
     X_cv_pssc_one,
     X_cv_noppp_norm,
     X_cv_q_norm,
     X_cv_price_norm)).tocsr()



X_te = hstack(
    (X_test_essay_bow,
     X_test_title_bow,
     X_test_school_ohe,
     X_test_pgc_ohe,
     X_test_tp_ohe,
     X_test_psc_ohe,
     X_test_pssc_ohe,
     X_test_noppp_norm,
     X_test_q_norm,
     X_test_price_norm)).tocsr()



print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(4489, 5550) (4489,)
(2211, 5550) (2211,)
(3300, 5550) (3300,)
====================================================================================================
```

6. Let's Build a model with above data

In [138]:

```
# SKlearn imports
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV
from sklearn import linear_model
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import confusion_matrix
from sklearn.linear_model import LogisticRegression
```

In [139]:

```
print( [10**x for x in range(-4,4)])
```

```
[0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
```

In [140]:

```
# Lets Define Some Function

# def plotHyperParameterErrorTuningGraph(X_train_hytu, y_train_hytu):
#     neigh = LogisticRegression()
#     parameters = {'C':
[np.log10(0.0001),np.log10(0.001),np.log10(0.01),np.log10(0.1),np.log10(1),np.log10(10),np.log10(1(
p.log10(1000),np.log10(10000)]}
#     clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc')
#     clf.fit(X_train_hytu, y_train_hytu)

#     train_auc= clf.cv_results_['mean_train_score']
#     train_auc_std= clf.cv_results_['std_train_score']
#     cv_auc = clf.cv_results_['mean_test_score']
#     cv_auc_std= clf.cv_results_['std_test_score']

#     plt.plot(parameters['C'], train_auc, label='Train AUC')
```

```
#     # this code is copied from here: https://stackoverflow.com/a/48803361/4084039
#     plt.gca().fill_between(parameters['C'],train_auc - train_auc_std,train_auc +
train_auc_std,alpha=0.2,color='blue')

#     plt.plot(parameters['C'], cv_auc, label='CV AUC')
#     # this code is copied from here: https://stackoverflow.com/a/48803361/4084039
#     plt.gca().fill_between(parameters['C'],cv_auc - cv_auc_std,cv_auc +
cv_auc_std,alpha=0.2,color='orange')

#     plt.scatter(parameters['C'], train_auc, label='Train AUC points')
#     plt.scatter(parameters['C'], cv_auc, label='CV AUC points')


#     plt.legend()
#     plt.xlabel("Alpha: hyperparameter")
#     plt.ylabel("AUC")
#     plt.title("ERROR PLOTS")
#     plt.grid()
#     plt.show()

def plotHyperParameterErrorTuningGraph(X_train_hytu, y_train_hytu):
    neigh = LogisticRegression()
    parameters = {'C':[0.0001,0.001,0.01,0.1,1,10,100,1000,10000]}
    clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc')
    clf.fit(X_train_hytu, y_train_hytu)

    train_auc= clf.cv_results_['mean_train_score']
    train_auc_std= clf.cv_results_['std_train_score']
    cv_auc = clf.cv_results_['mean_test_score']
    cv_auc_std= clf.cv_results_['std_test_score']

    plt.plot(np.log10(parameters['C']), train_auc, label='Train AUC')
    # this code is copied from here: https://stackoverflow.com/a/48803361/4084039
    plt.gca().fill_between(np.log10(parameters['C']),train_auc - train_auc_std,train_auc + train_au
c_std,alpha=0.2,color='blue')

    plt.plot(np.log10(parameters['C']), cv_auc, label='CV AUC')
    # this code is copied from here: https://stackoverflow.com/a/48803361/4084039
    plt.gca().fill_between(np.log10(parameters['C']),cv_auc - cv_auc_std,cv_auc + cv_auc_std,alpha=
0.2,color='orange')

    plt.scatter(np.log10(parameters['C']), train_auc, label='Train AUC points')
    plt.scatter(np.log10(parameters['C']), cv_auc, label='CV AUC points')


    plt.legend()
    plt.xlabel("Alpha: hyperparameter")
    plt.ylabel("AUC")
    plt.title("ERROR PLOTS")
    plt.grid()
    plt.savefig('plotted_graph.png')
    plt.show()
```

In [141]:

```
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)
    #print(predictions)
    return predictions
```

In [153]:

```
# https://stackoverflow.com/a/42265865/6000190
```

```
def plotConfusionMatrix(cm):
#     cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    sns.set(font_scale=1.4)#for label size
#     sns.heatmap(cm_normalized, cmap="Blues", annot=True,annot_kws={"size": 16})# font size
    sns.heatmap(cm, annot = True,cmap="Blues",annot_kws={"size": 16}, fmt='d')
    TP, TN, FP, FN = cm[0][0],cm[0][1],cm[1][0],cm[1][1]
    print("True Positive Value: {}".format(TP))
    print("True Negative Value: {}".format(TN))
    print("False Positive Value: {}".format(FP))
    print("False Negative Value: {}".format(FN))
```

In [154]:

```
def plotROCCurveGraph(X_train_roc, y_train_roc, X_test_roc, y_test_roc, best_alpha):
    # for i in tqdm(parameters):
    neigh = LogisticRegression( C=best_alpha, class_weight='balanced')
    neigh.fit(X_train_roc, y_train_roc)

    y_train_pred = neigh.predict_proba(X_train_roc)[:,1]
    y_test_pred = neigh.predict_proba(X_test_roc)[:,1]

    train_fpr, train_tpr, tr_thresholds = roc_curve(y_train_roc, y_train_pred)
    test_fpr, test_tpr, te_thresholds = roc_curve(y_test_roc, y_test_pred)

    m_Auc = str(auc(train_fpr, train_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()

    print("Train AUC values: "+str(auc(train_fpr, train_tpr)))
    print("Train AUC values: "+str(auc(test_fpr, test_tpr)))

    cm_train = confusion_matrix(y_train_roc, predict(y_train_pred, tr_thresholds, train_fpr, train_
fpr))
    cm_test = confusion_matrix(y_test_roc, predict(y_test_pred, tr_thresholds, test_fpr, test_fpr))

    print("Train confusion matrix")
    print(cm_train)
    print("Test confusion matrix")
    print(cm_test)


    return best_alpha, m_Auc, cm_train, cm_test
```

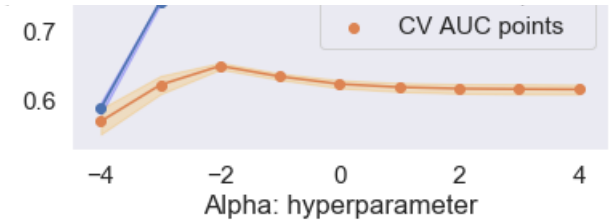
In [ ]:

```
```

Applying LogisticRegression on BOW, SET 1

In [155]:

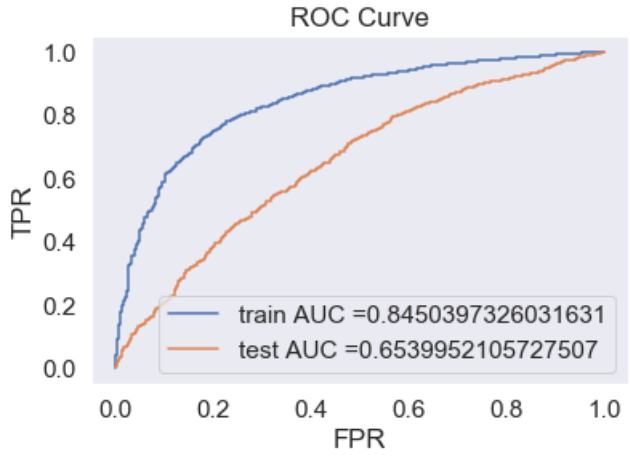
```
plotHyperParameterErrorTuningGraph(X_tr, y_train)
```



- As per the error Plot The minimum gap represents the best alpha value.
- And in this plot 2 is having less space between Train and Test Auc Graphs

In [157]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
best_alpha_m1, m1_Auc, cm_train, cm_test = plotROCCurveGraph(X_tr, y_train, X_te, y_test, 0.001)
```

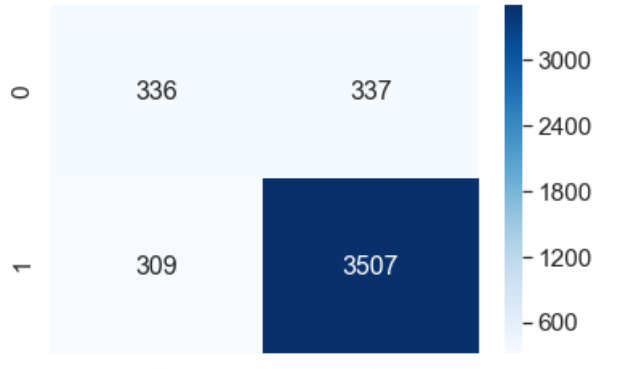



```
Train AUC values: 0.8450397326031631
Train AUC values: 0.6539952105727507
the maximum value of tpr*(1-fpr) 0.24999944803710958 for threshold 0.444
the maximum value of tpr*(1-fpr) 0.2499989796959494 for threshold 0.467
Train confusion matrix
[[ 336  337]
 [ 309 3507]]
Test confusion matrix
[[ 182  313]
 [ 468 2337]]
```

In [158]:

```
# Ploting Confusion Matrix for Train
plotConfusionMatrix(cm_train)
```

```
True Positive Value: 336
True Negative Value: 337
False Positive Value: 309
False Negative Value: 3507
```

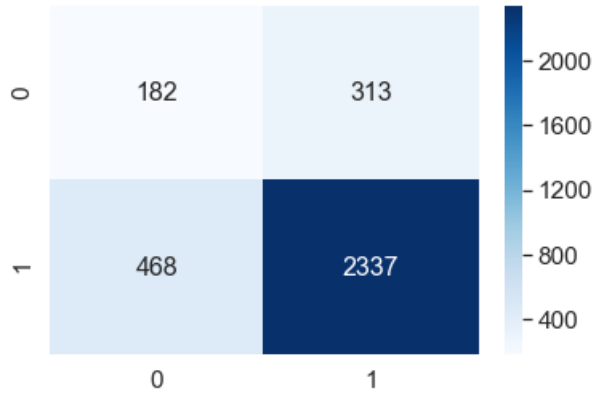


0 1

In [159]:

```
# Ploting Confusion Matrix for Test
plotConfusionMatrix(cm_test)
```

```
True Positive Value: 182
True Negative Value: 313
False Positive Value: 468
False Negative Value: 2337
```



Applying LR on AVG W2V, SET 3

FEaturization to Text for W2v

In [160]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-python/
# make sure you have the glove_vectors file

#https://drive.google.com/open?id=14nf-h6aYdhL_01I8DVg9CFZ5aMqAXeTi

with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [161]:

```
# THis is w2v vectorization type.

def makew2vList(xtrainEassyArray):
    avg_w2v_vector_eassy = []; # the avg-w2v for each sentence/review is stored in this list
    print(len(xtrainEassyArray))
    for sentence in tqdm(xtrainEassyArray): # 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_vector_eassy.append(vector)
    return avg_w2v_vector_eassy
```

In [162]:

```
def text2Vecw2v(feature):

    # we use the fitted CountVectorizer to convert the text to vector
    X_train_list_w2v = makew2vList(X_train['{}'.format(feature)].values)
```

```
    X_cv_list_w2v = makew2vList(X_cv['{}'.format(feature)].values)
    X_test_list_w2v = makew2vList(X_test['{}'.format(feature)].values)

    print("After vectorizations")
    print(np.array(X_train_list_w2v).shape, y_train.shape)
    print(np.array(X_cv_list_w2v).shape, y_cv.shape)
    print(np.array(X_test_list_w2v).shape, y_test.shape)
    print("="*10)
    print("X_train_list_w2v :")
    print(np.array(X_train_list_w2v)[:2,:2])
    print("X_cv_list_w2v :")
    print(np.array(X_cv_list_w2v)[:2,:2])
    print("X_test_list_w2v :")
    print(np.array(X_test_list_w2v)[:2,:2])

    return X_train_list_w2v, X_cv_list_w2v, X_test_list_w2v
```

- Now Combining all of the feature.. to put into the model

In [163]:

```
#  text2Vecw2v for train cv test
x_tr_w2v_eassy,x_cv_w2v_eassy, x_tes_w2v_eassy = text2Vecw2v('essay')
```

```
  2%|▌          | 77/4489 [00:00<00:05, 762.66it/s]
```

```
4489
```

```
100%|██████████| 4489/4489 [00:02<00:00, 2072.82it/s]
 20%|██        | 444/2211 [00:00<00:00, 2141.19it/s]
```

```
2211
```

```
100%|██████████| 2211/2211 [00:01<00:00, 2187.35it/s]
  6%|▌         | 203/3300 [00:00<00:01, 2026.09it/s]
```

```
3300
```

```
100%|██████████| 3300/3300 [00:01<00:00, 2502.99it/s]
```

```
After vectorizations
(4489, 300) (4489,)
(2211, 300) (2211,)
(3300, 300) (3300,)
==========
X_train_list_w2v :
[[ 0.01262111  0.03116973]
 [-0.05003747  0.0012176 ]]
X_cv_list_w2v :
[[0.00046231 0.01453527]
 [0.01295594 0.02829579]]
X_test_list_w2v :
[[ 0.01151679  0.03309468]
 [-0.02614215  0.0270397 ]]
```

In [164]:

```
#  text2Vecw2v for train cv test
x_tr_w2v_title ,x_cv_w2v_title, x_tes_w2v_title = text2Vecw2v('project_title')
```

```
100%|██████████| 4489/4489 [00:00<00:00, 39235.86it/s]
100%|██████████| 2211/2211 [00:00<00:00, 44088.44it/s]
  0%|          | 0/3300 [00:00<?, ?it/s]
```

```
4489
2211
3300
```

```
100%|██████████| 3300/3300 [00:00<00:00, 37820.60it/s]
```

```
After vectorizations
(4489, 300) (4489,)
(2211, 300) (2211,)
(3300, 300) (3300,)
==========
X_train_list_w2v :
[[-0.313065    0.00619667]
 [-0.0472486  -0.2644098 ]]
X_cv_list_w2v :
[[-0.269375    0.18040619]
 [-0.21621043 -0.01101843]]
X_test_list_w2v :
[[-0.06986692  0.2524068 ]
 [-0.17980825 -0.30345225]]
```

In [165]:

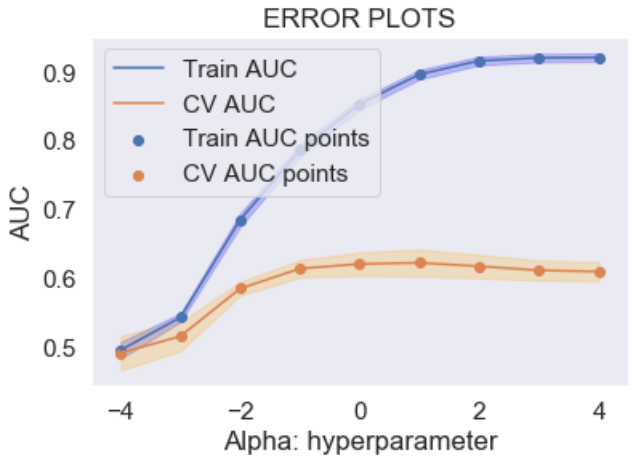
```
# # combining all features
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039

from scipy.sparse import hstack
X_tr = hstack(
    (np.array(x_tr_w2v_eassy),
     np.array(x_tr_w2v_title),
     X_train_school_ohe,
     X_train_pgc_ohe,
     X_train_tp_ohe,
     X_train_psc_ohe,
     X_train_pssc_ohe,
     X_train_noppp_norm,
     X_train_q_norm,
     X_train_price_norm)).tocsr()


X_cr = hstack(
    (np.array(x_cv_w2v_eassy),
     np.array(x_cv_w2v_title),
     X_cv_school_ohe,
     X_cv_pgc_ohe,
     X_cv_tp_ohe,
     X_cv_psc_ohe,
     X_cv_pssc_ohe,
     X_cv_noppp_norm,
     X_cv_q_norm,
     X_cv_price_norm)).tocsr()




X_te = hstack(
    (np.array(x_tes_w2v_eassy),
     np.array(x_tes_w2v_title),
     X_test_school_ohe,
     X_test_pgc_ohe,
     X_test_tp_ohe,
     X_test_psc_ohe,
     X_test_pssc_ohe,
     X_test_noppp_norm,
     X_test_q_norm,
     X_test_price_norm)).tocsr()




print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(4489, 743) (4489,)
```

```
(2211, 743) (2211,)
(3300, 743) (3300,)
==========================================================================================================
```

In [166]:

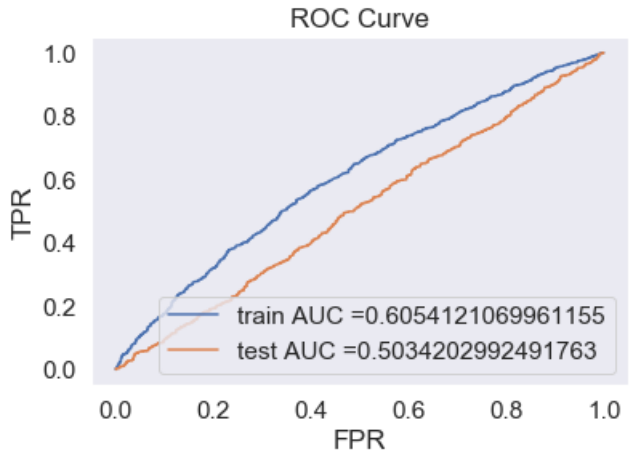
```
plotHyperParameterErrorTuningGraph(X_tr, y_train)
```



- As per the error Plot The minimum gap represents the best alphavalue.
- And in this plot 2 is having less space between Train and Test Auc Graphs

In [209]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
best_alpha_m3, m3_Auc, cm_train, cm_test = plotROCCurveGraph(X_tr, y_train, X_te, y_test, np.log10(
2))
```




```
Train AUC values: 0.6054121069961155
Train AUC values: 0.5034202992491763
the maximum value of tpr*(1-fpr) 0.24999944803710958 for threshold 0.489
the maximum value of tpr*(1-fpr) 0.2499989796959494 for threshold 0.522
Train confusion matrix
[[ 337  336]
 [1330 2486]]
Test confusion matrix
[[ 294  201]
 [1662 1143]]
```

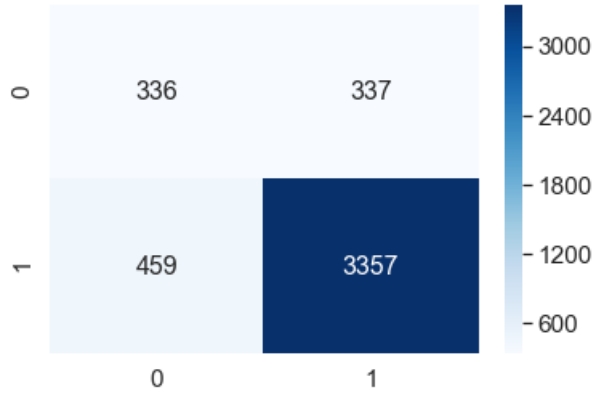
This model Performance is Not Good At all. Cause its Showing result test AuC of <50%.

In [168]:

```
# Plotting Confusion Matrix for Train
```

```
# Ploting Confusion Matrix for Train
plotConfusionMatrix(cm_train)
```

```
True Positive Value: 336
True Negative Value: 337
False Positive Value: 459
False Negative Value: 3357
```

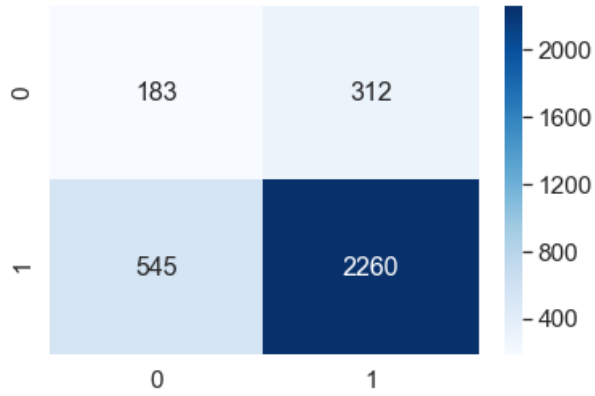


- Train AUC values is 0.5
- Test AUC value is 0.5
- Comparing to Bow model this model is not performing in the test data sets.

In [169]:

```
# Ploting Confusion Matrix for Test
plotConfusionMatrix(cm_test)
```

```
True Positive Value: 183
True Negative Value: 312
False Positive Value: 545
False Negative Value: 2260
```



This model Performance is Not Good. FP Value should be lower Comparing to FN .

Applying KNN brute force on TFIDF W2V, SET 4

In [ ]:

```

```

In [170]:

```
# Similarly you can vectorize for title also

def maketfidfw2vList(feature, xEassyArray):
```

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


    tfidf_w2v_vectors_project_text = []; # the avg-w2v for each sentence/review is stored in this
list
    for sentence in tqdm(xEassyArray): # for each review/sentence
        vector = np.zeros(300) # as word vectors are of zero length
        tf_idf_weight =0; # num of words with a valid vector in the sentence/review
        for word in sentence.split(): # for each word in a review/sentence
            if (word in glove_words) and (word in tfidf_words):
                vec = model[word] # getting the vector for each word
                # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
                tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting
the 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_project_text.append(vector)
    return tfidf_w2v_vectors_project_text
```

In [171]:

```
def text2VecTfidfW2v(feature):

    # we use the fitted CountVectorizer to convert the text to vector
    X_train_list_tfidfw2v = maketfidfw2vList(feature, X_train['{}'.format(feature)].values)
    X_cv_list_tfidfw2v = maketfidfw2vList(feature, X_cv['{}'.format(feature)].values)
    X_test_list_tfidfw2v = maketfidfw2vList(feature, X_test['{}'.format(feature)].values)

    print("After vectorizations")
    print(np.array(X_train_list_tfidfw2v).shape, y_train.shape)
    print(np.array(X_cv_list_tfidfw2v).shape, y_cv.shape)
    print(np.array(X_test_list_tfidfw2v).shape, y_test.shape)
    print("="*10)
    print("X_train_list_w2v :")
    print(np.array(X_train_list_tfidfw2v)[:2,:2])
    print("X_cv_list_w2v :")
    print(np.array(X_cv_list_tfidfw2v)[:2,:2])
    print("X_test_list_w2v :")
    print(np.array(X_test_list_tfidfw2v)[:2,:2])

    return X_train_list_tfidfw2v, X_cv_list_tfidfw2v, X_test_list_tfidfw2v
```

In [172]:

```
# maketfidfw2vList
x_tr_tfidfw2v_eassy, x_cv_tfidfw2v_eassy, x_tes_tfidfw2v_eassy = text2VecTfidfW2v('essay')
```

```
100%|██████████| 4489/4489 [00:14<00:00, 311.84it/s]
100%|██████████| 2211/2211 [00:06<00:00, 321.00it/s]
100%|██████████| 3300/3300 [00:10<00:00, 318.73it/s]
```

```
After vectorizations
(4489, 300) (4489,)
(2211, 300) (2211,)
(3300, 300) (3300,)
==========
X_train_list_w2v :
[[ 0.01264499  0.04120872]
 [-0.04308857 -0.04988383]]
X_cv_list_w2v :
[[-0.00785342 -0.03295695]
 [-0.01446013  0.02852146]]
X_test_list_w2v :
[[ 0.01774244  0.01680825]
```

```
 [-0.06206841  0.03701767]]
```

In [173]:

```
# maketfidfw2vList
x_tr_tfidfw2v_title, x_cv_tfidfw2v_title, x_tes_tfidfw2v_title = text2VecTfidfW2v('project_title')
```

```
100%|██████████| 4489/4489 [00:00<00:00, 23153.74it/s]
100%|██████████| 2211/2211 [00:00<00:00, 19259.23it/s]
100%|██████████| 3300/3300 [00:00<00:00, 19000.32it/s]
```

```
After vectorizations
(4489, 300) (4489,)
(2211, 300) (2211,)
(3300, 300) (3300,)
==========
X_train_list_w2v :
[[-0.41494333 -0.37394416]
 [-0.01384593 -0.26695051]]
X_cv_list_w2v :
[[-0.21799728  0.24316173]
 [-0.2342455   0.01625889]]
X_test_list_w2v :
[[-0.05173604  0.23991109]
 [-0.21683831 -0.27908929]]
```

- Combining All Vecotrizers to train cv and test to put into model

In [174]:

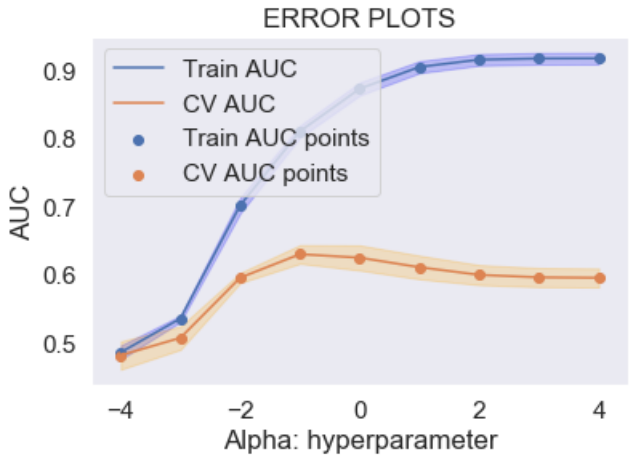
```
# combining all features

# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039

# # combining all features
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039

from scipy.sparse import hstack
X_tr = hstack(
    (np.array(x_tr_tfidfw2v_eassy),
     np.array(x_tr_tfidfw2v_title),
     X_train_school_ohe,
     X_train_pgc_ohe,
     X_train_tp_ohe,
     X_train_psc_ohe,
     X_train_pssc_ohe,
     X_train_noppp_norm,
     X_train_q_norm,
     X_train_price_norm)).tocsr()


X_cr = hstack(
    (np.array(x_cv_tfidfw2v_eassy),
     np.array(x_cv_tfidfw2v_title),
     X_cv_school_ohe,
     X_cv_pgc_ohe,
     X_cv_tp_ohe,
     X_cv_psc_ohe,
     X_cv_pssc_ohe,
     X_cv_noppp_norm,
     X_cv_q_norm,
     X_cv_price_norm)).tocsr()




X_te = hstack(
    (np.array(x_tes_tfidfw2v_eassy),
     np.array(x_tes_tfidfw2v_eassy),
     X_test_school_ohe,
     X_test_pgc_ohe,
     X_test_tp_ohe,
     X_test_psc_ohe
```

```
      X_test_pssc_ohe,
      X_test_noppp_norm,
      X_test_q_norm,
      X_test_price_norm)).tocsr()



print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(4489, 743) (4489,)
(2211, 743) (2211,)
(3300, 743) (3300,)
====================================================================================================
```

In [175]:

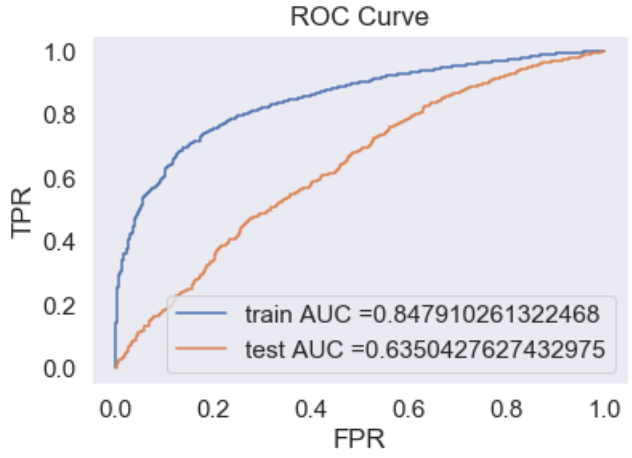
```
plotHyperParameterErrorTuningGraph(X_tr, y_train)
```



- As per the error Plot The minimum gap represents the best alpha value.
- And in this plot 0.01 is having less space between Train and Test Auc Graphs

In [176]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
best_alpha_m4, m4_Auc, cm_train, cm_test = plotROCCurveGraph(X_tr, y_train, X_te, y_test, np.log(2)
)
```



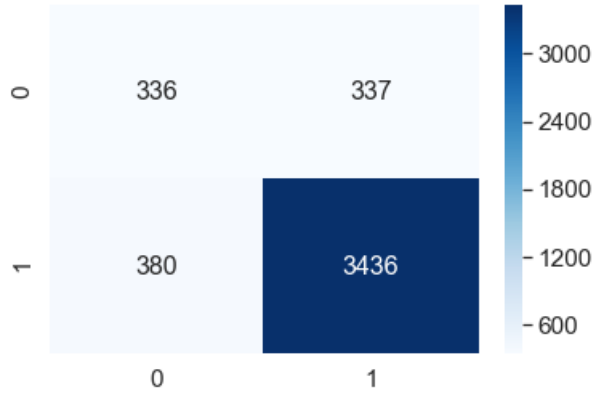
```
Train AUC values: 0.847910261322468
Train AUC values: 0.6350427627432975
the maximum value of tpr*(1-fpr) 0.24999944803710958 for threshold 0.324
the maximum value of tpr*(1-fpr) 0.2499989796959494 for threshold 0.393
Train confusion matrix
[[ 336  337]
 [ 380 3436]]
Test confusion matrix
[[ 140  355]
 [ 339 2466]]
```

This model Performance is Not Good. Test Auc is less then 50% .

In [177]:

```
# Ploting Confusion Matrix for Train
plotConfusionMatrix(cm_train)
```

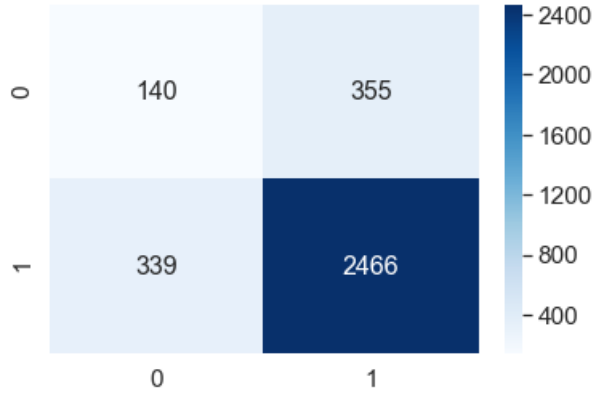
```
True Positive Value: 336
True Negative Value: 337
False Positive Value: 380
False Negative Value: 3436
```



In [178]:

```
# Ploting Confusion Matrix for Test
plotConfusionMatrix(cm_test)
```

```
True Positive Value: 140
True Negative Value: 355
False Positive Value: 339
False Negative Value: 2466
```



- FP Value should be lower Comparing to FN . And TP has Heigher value the TF. Which is good.

In [ ]:

In [ ]:

```
```

Applying KNN brute force on TFIDF, SET 2 With Best 20 Features

In [179]:

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

In [180]:

```
# Making DataMatrix with eassay and title with TFIDF


def text2Vectfidf(feature):
    vectorizerTfIDF = TfidfVectorizer(min_df=10, max_features=5000)
    vectorizerTfIDF.fit(X_train['{}'.format(feature)].values)

    # we use the fitted CountVectorizer to convert the text to vector
    X_train_list_tfidf = vectorizerTfIDF.transform(X_train['{}'.format(feature)].values)
    X_cv_list_tfidf = vectorizerTfIDF.transform(X_cv['{}'.format(feature)].values)
    X_test_list_tfidf = vectorizerTfIDF.transform(X_test['{}'.format(feature)].values)

    print("Shape of matrix Shape of matrix TFIDFg ",X_train_list_tfidf.shape)

    print("After vectorizations")
    print(X_train_list_tfidf.shape, y_train.shape)
    print(X_cv_list_tfidf.shape, y_cv.shape)
    print(X_test_list_tfidf.shape, y_test.shape)
    print("="*50)

    return X_train_list_tfidf, X_cv_list_tfidf, X_test_list_tfidf
```

In [181]:

```
X_train_essay_tfidf, X_cv_essay_tfidf, X_test_essay_tfidf = text2Vectfidf('essay')
```

```
Shape of matrix Shape of matrix TFIDFg  (4489, 4130)
After vectorizations
(4489, 4130) (4489,)
(2211, 4130) (2211,)
(3300, 4130) (3300,)
==================================================
```

In [182]:

```
X_train_title_tfidf, X_cv_title_tfidf, X_test_title_tfidf = text2Vectfidf('project_title')
```

```
Shape of matrix Shape of matrix TFIDFg  (4489, 337)
After vectorizations
(4489, 337) (4489,)
(2211, 337) (2211,)
(3300, 337) (3300,)
==================================================
```

In [183]:

```
X_cv_essay_tfidf.shape
```

Out[183]:

```
(2211, 4130)
```

In [184]:

```
```

```
X_train_essay_tfidf.shape
```

Out[184]:

```
(4489, 4130)
```

In [185]:

```
X_cv_price_norm.shape
```

Out[185]:

```
(2211, 1)
```

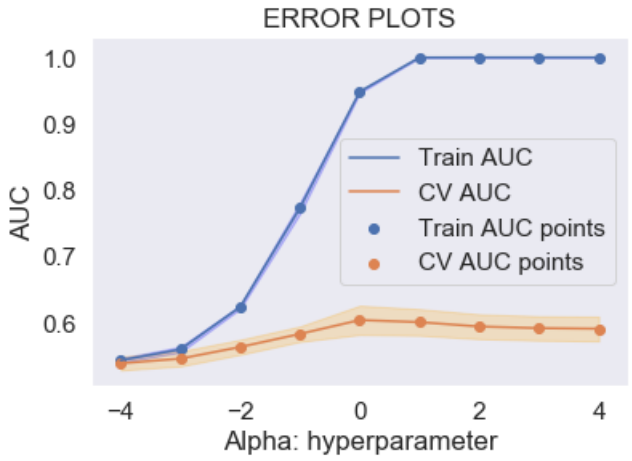
In [186]:

```
# combining all features

X_tr = hstack(
    (X_train_essay_tfidf,
     X_train_title_tfidf,
     X_train_school_ohe,
     X_train_pgc_ohe,
     X_train_tp_ohe,
     X_train_psc_ohe,
     X_train_pssc_ohe,
     X_train_noppp_norm,
     X_train_q_norm,
     X_train_price_norm)).tocsr()


X_cr = hstack(
    (X_cv_essay_tfidf,
     X_cv_title_tfidf,
     X_cv_school_ohe,
     X_cv_pgc_ohe,
     X_cv_tp_ohe,
     X_cv_psc_ohe,
     X_cv_pssc_ohe,
     X_cv_noppp_norm,
     X_cv_q_norm,
     X_cv_price_norm)).tocsr()




X_te = hstack(
    (X_test_essay_tfidf,
     X_test_title_tfidf,
     X_test_school_ohe,
     X_test_pgc_ohe,
     X_test_tp_ohe,
     X_test_psc_ohe,
     X_test_pssc_ohe,
     X_test_noppp_norm,
     X_test_q_norm,
     X_test_price_norm)).tocsr()




print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(4489, 4610) (4489,)
(2211, 4610) (2211,)
(3300, 4610) (3300,)
====================================================================================================
```

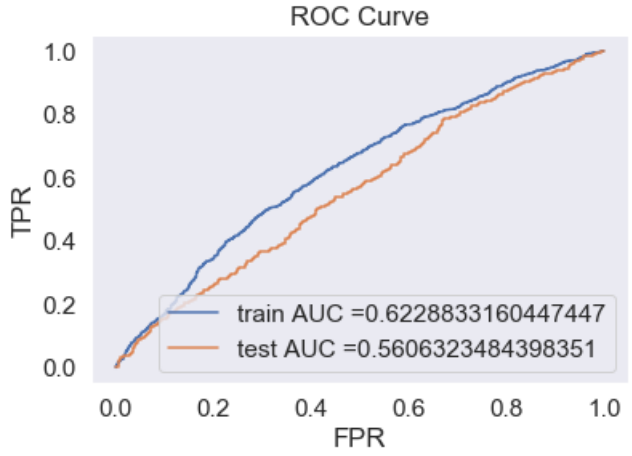
In [76]:

```
plotHyperParameterErrorTuningGraph(X_tr, y_train)
```



In [188]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
best_alpha_m2, m2_Auc, cm_train, cm_test = plotROCCurveGraph(X_tr, y_train, X_te, y_test,  0.001)
```



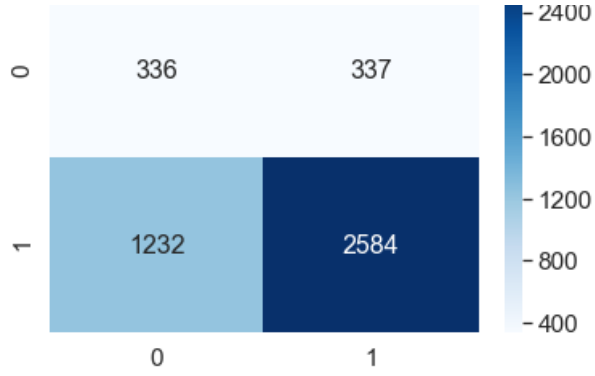

```
Train AUC values: 0.6228833160447447
Train AUC values: 0.5606323484398351
the maximum value of tpr*(1-fpr) 0.24999944803710958 for threshold 0.5
the maximum value of tpr*(1-fpr) 0.2499989796959494 for threshold 0.505
Train confusion matrix
[[ 336  337]
 [1232 2584]]
Test confusion matrix
[[ 290  205]
 [1400 1405]]
```

This Model Showed Better Result the TFIDF weighed Models

In [189]:

```
# Ploting Confusion Matrix for Train
plotConfusionMatrix(cm_train)
```

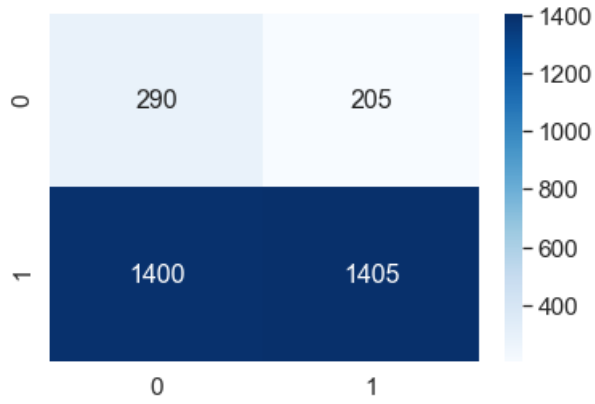
```
True Positive Value: 336
True Negative Value: 337
False Positive Value: 1232
False Negative Value: 2584
```



In [190]:

```
# Ploting Confusion Matrix for Test
plotConfusionMatrix(cm_test)
```

```
True Positive Value: 290
True Negative Value: 205
False Positive Value: 1400
False Negative Value: 1405
```



</p>Test Results Didn't showed Any Major Difference Comparing to other Models. TP Value is 0.61 which is good But FN Should be heigher Too.</p>

In [ ]:

```

```

Set 5

- **[Task-2] Apply Logistic Regression on the below feature set Set 5 by finding the best hyper parameter as suggested in step 2 and step 3.**
- Consider these set of features Set 5 :

  - **school_state** : categorical data
  - **clean_categories** : categorical data
  - **clean_subcategories** : categorical data
  - **project_grade_category** :categorical data
  - **teacher_prefix** : categorical data
  - **quantity** : numerical data
  - **teacher_number_of_previously_posted_projects** : numerical data
  - **price** : numerical data
  - **sentiment score's of each of the essay** : numerical data
  - **number of words in the title** : numerical data
  - **number of words in the combine essays** : numerical data

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

In [ ]:

```
```

In [191]:

```
data.head(1)
```

Out[191]:

	id	teacher_prefix	school_state	project_grade_category	project_subject_categories	project_subject_subcategories	project_title
0	p253737	Mrs.	IN	Grades PreK-2	Literacy_Language	ESL Literacy	educational support english learners home

In [192]:

```
# Lets do preprocess and make the data as in Task5

def replaceTextWithWordCount(feature):
    print("\033[0m Storing Polarity Score for {}: ".format(feature))
    wordCount = []
    for i in range(len(data['{}'.format(feature)])):
        singleLine = data['{}'.format(feature)][i]
        if ( (singleLine is None) or (str(singleLine).strip()=="") ):
            wordCount.append(0)
        else:
            wordCount.append(len(str(singleLine).strip()))
#         print("wordCount: {}".format(wordCount))


    data['{}'.format(feature)] = wordCount
    print("\033[0m Storing Polarity Score for {} 100 % Completed.".format(feature))
    print("\033[0m Coumpound score of {}[0] is :: '\033[1m' {}".format(feature, wordCount[0]))
    print("="*10)
    return wordCount
```

In [193]:

```
wordCountList = replaceTextWithWordCount('project_title')
```

```
 Storing Polarity Score for project_title:
 Storing Polarity Score for project_title 100 % Completed.
 Coumpound score of project_title[0] is :: '' 41
==========
```

In [194]:

```
wordCountListEssay = replaceTextWithWordCount('essay')
```

```
 Storing Polarity Score for essay:
 Storing Polarity Score for essay 100 % Completed.
 Coumpound score of essay[0] is :: '' 1121
==========
```

In [195]:

```
data.head(1)
```

Out[195]:

	id	teacher_prefix	school_state	project_grade_category	project_subject_categories	project_subject_subcategories	project_title
0	p253737	Mrs.	IN	Grades PreK-2	Literacy_Language	ESL Literacy	41

In [196]:

```
X = data
```

In [197]:

```
# train test split
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)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)
```

In [ ]:

```

```

In [198]:

```
# Normalizing Numerical Data Project Essay
X_train_num_essay_norm, X_cv_num_essay_norm, X_test_num_essay_norm = num2Vec('essay')
```

```
After vectorizations
(4489, 1) (4489,)
(2211, 1) (2211,)
(3300, 1) (3300,)
[[1.]
 [1.]
 [1.]
 ...
 [1.]
 [1.]
 [1.]]
==========
```

In [199]:

```
# Normalizing New Numerical Data

# Normalizing Numerical Data Project title
X_train_num_title_norm, X_cv_num_title_norm, X_test_num_title_norm = num2Vec('project_title')
```

```
After vectorizations
(4489, 1) (4489,)
(2211, 1) (2211,)
(3300, 1) (3300,)
[[1.]
 [1.]
 [1.]
 ...
 [1.]
 [1.]
 [1.]]
==========
```

In [ ]:

```

```

In [200]:

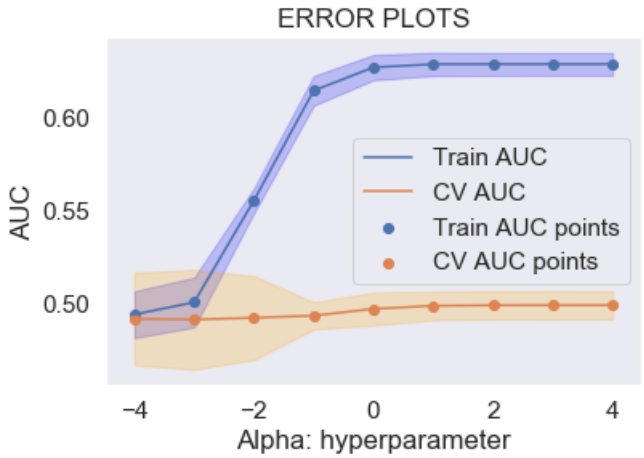
```
# combining all features

X_tr = hstack(
    (X_train_num_essay_norm,
     X_train_num_title_norm,
     X_train_school_ohe,
     X_train_pgc_ohe,
     X_train_tp_ohe,
     X_train_psc_ohe,
     X_train_pssc_ohe
```

```
     X_train_noppp_norm,
     X_train_q_norm,
     X_train_price_norm)).tocsr()


X_cr = hstack(
    (X_cv_num_essay_norm,
     X_cv_num_title_norm,
     X_cv_school_ohe,
     X_cv_pgc_ohe,
     X_cv_tp_ohe,
     X_cv_psc_ohe,
     X_cv_pssc_ohe,
     X_cv_noppp_norm,
     X_cv_q_norm,
     X_cv_price_norm)).tocsr()



X_te = hstack(
    (X_test_num_essay_norm,
     X_test_num_title_norm,
     X_test_school_ohe,
     X_test_pgc_ohe,
     X_test_tp_ohe,
     X_test_psc_ohe,
     X_test_pssc_ohe,
     X_test_noppp_norm,
     X_test_q_norm,
     X_test_price_norm)).tocsr()



print("Final Data matrix")
print(X_tr.shape, y_train.shape)
print(X_cr.shape, y_cv.shape)
print(X_te.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(4489, 145) (4489,)
(2211, 145) (2211,)
(3300, 145) (3300,)
====================================================================================================
```

In [201]:

```
# Applying on set5
plotHyperParameterErrorTuningGraph(X_tr, y_train)
```

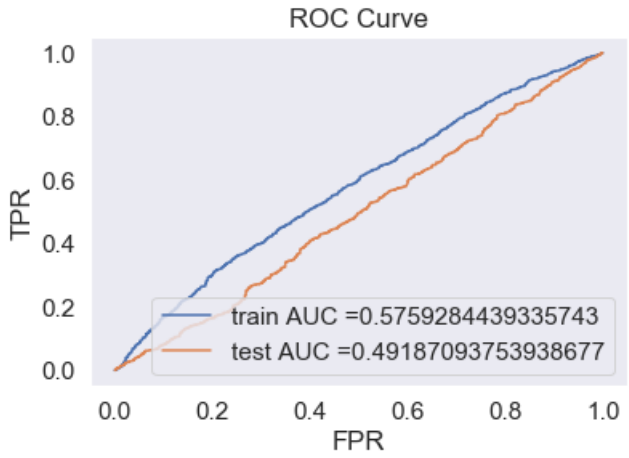


- As per the error Plot The minimum gap represents the best alpha value.
- And in this plot 5 is having less space between Train and Test Auc Graphs

- And in this plot we is having less space between Train and Test Auc Graphs

In [202]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
best_alpha_m5, m5_Auc, cm_train, cm_test = plotROCCurveGraph(X_tr, y_train, X_te, y_test,  0.001)
```



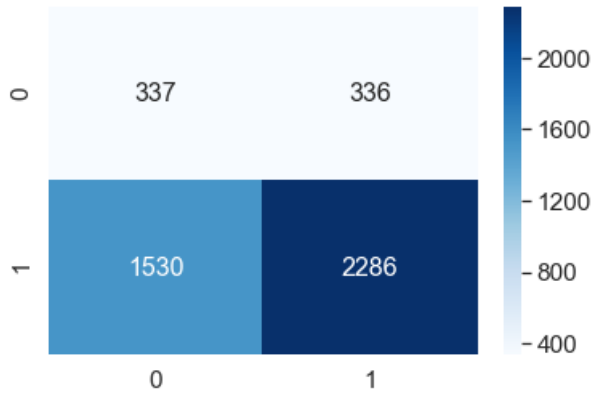

```
Train AUC values: 0.5759284439335743
Train AUC values: 0.49187093753938677
the maximum value of tpr*(1-fpr) 0.24999944803710958 for threshold 0.499
the maximum value of tpr*(1-fpr) 0.2499989796959494 for threshold 0.501
Train confusion matrix
[[ 337  336]
 [1530 2286]]
Test confusion matrix
[[ 301  194]
 [1691 1114]]
```

Not At all a good Model . Test AUC came just 50%.

In [203]:

```
# Ploting Confusion Matrix for Train
plotConfusionMatrix(cm_train)
```

```
True Positive Value: 337
True Negative Value: 336
False Positive Value: 1530
False Negative Value: 2286
```

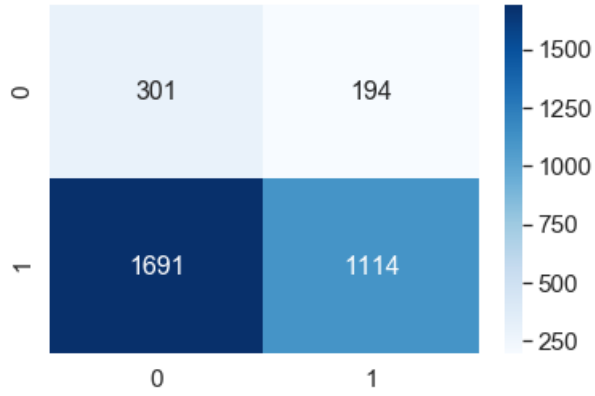


In [204]:

```
# Ploting Confusion Matrix for Test
plotConfusionMatrix(cm_test)
```

```
True Positive Value: 301
```

```
True Negative Value: 194
False Positive Value: 1691
False Negative Value: 1114
```



As same as the Previous Models this Model didn't worked By Converting Text data to numerical.

In [ ]:

```
```

In [ ]:

```
```

Final Table

In [205]:

```
# http://zetcode.com/python/prettytable/
from prettytable import PrettyTable
```

In [210]:

```
#If you get a ModuleNotFoundError error , install prettytable using: pip3 install prettytable

x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Hyper Parameter", "AUC"]


x.add_row(["BOW", "LR", str(best_alpha_m1)[:5], m1_Auc])
x.add_row([" ", " ", " ", " "])
x.add_row(["TFIDF", "LR", str(best_alpha_m2)[:5], m2_Auc])
x.add_row([" ", " ", " ", " "])
x.add_row(["W2V", "LR", str(best_alpha_m3)[:5], m3_Auc])
x.add_row([" ", " ", " ", " "])
x.add_row(["TFIDFW2V", "LR", str(best_alpha_m4)[:5], m4_Auc])
x.add_row([" ", " ", " ", " "])
x.add_row(["Witout Txt Data", "", str(best_alpha_m5)[:5], m5_Auc])


print(x)
```

```
+-----------------+-------+-----------------+--------------------+
|    Vectorizer   | Model | Hyper Parameter |        AUC         |
+-----------------+-------+-----------------+--------------------+
|       BOW       |   LR  |       0.001     | 0.8450397326031631 |
|                 |       |                 |                    |
|      TFIDF      |   LR  |       0.001     | 0.6228833160447447 |
|                 |       |                 |                    |
|       W2V       |   LR  |       0.301     | 0.6054121069961155 |
|                 |       |                 |                    |
|     TFIDFW2V    |   LR  |       0.693     | 0.847910261322468  |
|                 |       |                 |                    |
| Witout Txt Data |       |       0.001     | 0.5759284439335743 |
```

```
+-----------------+-------+-----------------+--------------------+
```

In [ ]:

```
```

Observations

- 1. We Have applied Logistic Regression For this Dataset.
- 1. We have taken 1Lac Data points for this computation
- 1. BOW weighted Model (Set 4) is doing good.
- 1. Model without text data set is not performing Good.
- 1. Removing Text features didn't shown any best performance instead went performance went lower.

In [ ]:

```
```