Assignment-9 Apply Random Forests & GBDT on Donors Choose dataset

In [1]:

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

1.1 Loading Data

In [2]:

```
data = pd.read_csv('preprocessed_data.csv', nrows=50000)
data.head(2)
```

Out[2]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	proj
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	

2 rows × 29 columns

In [3]:

```
data['project_is_approved'].value_counts()
```

Out[3]:

```
1    42286
0     7714
Name: project_is_approved, dtype: int64
```

In [4]:

```
y = data['project_is_approved']
X = data.drop(['project_is_approved'], axis=1)
X.head(2)
```

Out[4]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state	proj
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN	
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL	

2 rows × 28 columns

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

In [5]:

```
# 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)
```

In [6]:

```
X_train['clean_subcategories'].isnull().values.any()
```

Out[6]:

```
False
```

1.3 Make Data Model Ready: encoding essay, and project_title

1.3.1 Vectorizing preprocessed essays & project_title using BOW

In [7]:

```
# preprocessed essays
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)

print("="*100)


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

# we use the fit CountVectorizer to convert the text to vector
X_train_essay_bow = vectorizer.transform(X_train['preprocessed_essays'].values)
X_test_essay_bow = vectorizer.transform(X_test['preprocessed_essays'].values)
```

```
(33500, 28) (33500,)
(16500, 28) (16500,)
==========================================================================
==========================
```

In [8]:

```
f1=vectorizer.get_feature_names()
print("After vectorization")
print(X_train_essay_bow.shape, y_train.shape)
print(X_test_essay_bow.shape, y_test.shape)
print("="*100)
```

```
After vectorization
(33500, 5000) (33500,)
(16500, 5000) (16500,)
==========================================================================
==========================
```

In [9]:

```
#project_title
vectorizer = CountVectorizer(min_df=10,ngram_range=(1,2), max_features=5000)
vectorizer.fit(X_train['preprocessed_titles'].values.astype('U'))

X_train_title_bow = vectorizer.transform(X_train['preprocessed_titles'].values.astype(
'U'))
X_test_title_bow = vectorizer.transform(X_test['preprocessed_titles'].values.astype('U'
))
```

In [10]:

```
f2=vectorizer.get_feature_names()
print("After vectorization")
print(X_train_title_bow.shape, y_train.shape)
print(X_test_title_bow.shape, y_test.shape)
print("="*100)
```

```
After vectorization
(33500, 2340) (33500,)
(16500, 2340) (16500,)
===========================================================================
=========================
```

1.3.2 Vectorizing preprocessed essays & project_title using TFIDF

In [11]:

```
#TFIDF for preprocessed_essays
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,2), max_features=5000)
vectorizer.fit(X_train['preprocessed_essays'].values)

X_train_essay_tfidf = vectorizer.transform(X_train['preprocessed_essays'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['preprocessed_essays'].values)
```

In [12]:

```
f3=vectorizer.get_feature_names()
print("After vectorization")
print(X_train_essay_tfidf.shape, y_train.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorization
(33500, 5000) (33500,)
(16500, 5000) (16500,)
===========================================================================
=========================
```

In [13]:

```
#TFIDF for preprocessed_titles
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,2), max_features=5000)
vectorizer.fit(X_train['preprocessed_titles'].values.astype('U'))

X_train_titles_tfidf = vectorizer.transform(X_train['preprocessed_titles'].values.astyp
e('U'))
X_test_titles_tfidf = vectorizer.transform(X_test['preprocessed_titles'].values.astype(
'U'))
```

In [14]:

```
f4=vectorizer.get_feature_names()
print("After vectorization")
print(X_train_titles_tfidf.shape, y_train.shape)
print(X_test_titles_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorization
(33500, 2340) (33500,)
(16500, 2340) (16500,)
==========================================================================
==========================
```

1.3.3 Vectorizing preprocessed essays & project_title using Avg W2V

1.3.3.1 For preprocessed_titles

In [15]:

```
#Avg W2V for preprocessed_titles
#Train your own Word2Vec model using your own text corpus
import warnings
warnings.filterwarnings("ignore")
#train data
w2v_data= X_train['preprocessed_titles']
split_title_train=[]
for row in w2v_data:
    split_title_train.append([word for word in str(row).split()])     #splitting words

#train your W2v
train_w2v = Word2Vec(split_title_train,min_count=1,size=50, workers=4)
word_vectors_train = train_w2v.wv
w2v_words_train =list(word_vectors_train.vocab)
print(len(w2v_words_train ))
```

```
9594
```

In [16]:

```
# compute average word2vec for each title.
sent_vectors_train = [] # the avg-w2v for each title is stored in this list
for sent in tqdm(split_title_train):   # for each title
    sent_vec = np.zeros(50) # as word vectors are of zero length 50
    cnt_words =0   # num of words with a valid vector in the title
    for word in sent:   # for each word in a title
        if word in w2v_words_train:
            vec = word_vectors_train[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
        sent_vectors_train.append(sent_vec)
print(len(sent_vectors_train))
print(len(sent_vectors_train[3]))
```

```
100%|████████████████████████████████████████| 33500/33500 [00:04<00:00, 7026.69it/s]

33500
50
```

In [17]:

```
# For test data

# compute average word2vec for each title.
sent_vectors_test = [] # the avg-w2v for each title is stored in this list
for sent in tqdm(X_test['preprocessed_titles']):   # for each title
    sent_vec = np.zeros(50) # as word vectors are of zero length 50
    #cnt_words =0   # num of words with a valid vector in the title
    for word in str(sent):   # for each word in a title
        if word in w2v_words_train:
            vec = word_vectors_train[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
        sent_vectors_test.append(sent_vec)
print(len(sent_vectors_test))
print(len(sent_vectors_test[3]))
```

```
100%|████████████████████████████████████████| 16500/16500 [00:57<00:00, 284.50it/s]

16500
50
```

1.3.3.2 For preprocessed_essays

Using Pretrained Models: Avg W2V

In [18]:

```
# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-p
ickle-to-save-and-load-variables-in-python/
# make sure you have the glove_vectors file
with open('C:\\Users\\Admin\\Assignments and case studies\\Mandatory\\Assignment 7-SVM
 on donors choose\\glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
print ("Done.",len(model)," words loaded!")
```

```
Done. 51510  words loaded!
```

In [19]:

```
# Avg W2V for train data
# compute average word2vec for each review.
avg_w2v_essay_train = []   # the avg-w2v for each sentence/review is stored in this lis
t
for sentence in tqdm(X_train['preprocessed_essays']):   # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0 # num of words with a valid vector in the sentence/review
    for word in sentence.split():  # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_essay_train.append(vector)
print(len(avg_w2v_essay_train))
print(len(avg_w2v_essay_train[0]))
```

```
100%|██████████████████████████████████████████████████████████████████
██████████████████████████████████████████████████| 33500/33500 [00:13<0
0:00, 2461.43it/s]

33500
300
```

In [20]:

```
# Avg W2V for test data

avg_w2v_essay_test = []   # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['preprocessed_essays']):    # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0 # num of words with a valid vector in the sentence/review
    for word in sentence.split():  # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    avg_w2v_essay_test.append(vector)
print(len(avg_w2v_essay_test))
print(len(avg_w2v_essay_test[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████████████████████| 16500/16500 [00:07<00:00, 2201.66it/s]

16500
300
```

1.3.4 Vectorizing preprocessed essays & project_title using TFIDF weighted W2V

1.3.4.1 For preprocessed essays

In [21]:

```
# For train data

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

In [22]:

```
# average Word2Vec using pretrained models
# compute average word2vec for each review.
tfidf_w2v_train_essay = [] # the avg-w2v for each sentence/review is stored in this lis
t
for sentence in tqdm(X_train['preprocessed_essays']): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words_essays):
            vec = model[word]  # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sen
tence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
tting 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_train_essay.append(vector)
print(len(tfidf_w2v_train_essay))
print(len(tfidf_w2v_train_essay[0]))
```

```
100%|██████████████████████████████
██████████████████████| 33500/33500 [01:44<0
0:00, 321.24it/s]

33500
300
```

In [23]:

```
# For test data

tfidf_w2v_test_essay = [] # the avg-w2v for each sentence/review is stored in this list
for sentence2 in tqdm(X_test['preprocessed_essays']): # for each review/sentence
    vector2 = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight2 =0; # num of words with a valid vector in the sentence/review
    for word2 in sentence2.split(): # for each word in a review/sentence
        if (word2 in glove_words) and (word2 in tfidf_words_essays):
            vec2 = model[word2]  # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sen
tence.count(word)/len(sentence.split())))
            tf_idf2 = dictionary[word2]*(sentence2.count(word2)/len(sentence2.split()))
# getting the tfidf value for each word
            vector2 += (vec2 * tf_idf2) # calculating tfidf weighted w2v
            tf_idf_weight2 += tf_idf2
    if tf_idf_weight2 != 0:
        vector2 /= tf_idf_weight2
        tfidf_w2v_test_essay.append(vector2)
print(len(tfidf_w2v_test_essay))
print(len(tfidf_w2v_test_essay[0]))
```

```
100%|██████████████████████████████
██████████████████████| 16500/16500 [00:54<0
0:00, 304.92it/s]

16500
300
```

1.3.4.2 For preprocessed titles

Using pretrained models

In [24]:

```
# For train data

tfidf_model1 = TfidfVectorizer()
tfidf_model1.fit(X_train['preprocessed_titles'].values.astype('U'))
#we are converting a dictionary with word as a key, and the idf as a value
dictionary_title = dict(zip(tfidf_model1.get_feature_names(), list(tfidf_model1.idf_)))
tfidf_words_titles = set(tfidf_model1.get_feature_names())
```

In [26]:

```
# average Word2Vec using pretrained models
# compute average word2vec for each review.
tfidf_w2v_train_title = [] # the avg-w2v for each sentence/review is stored in this lis
t
for sentence_title in tqdm(X_train['preprocessed_titles']): # for each review/sentence
    vector3 = np.zeros(300) # as word vectors are of zero length
    #tf_idf_weight3=0; # num of words with a valid vector in the sentence/review
    for word3 in str(sentence_title).split(): # for each word in a review/sentence
        if (word3 in glove_words) and (word3 in tfidf_words_titles):
            vec4 = model[word3]  # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sen
tence.count(word)/len(sentence.split())))
            tf_idf3 = dictionary_title[word3]*(sentence_title.count(word3)/len(str(sent
ence_title).split())) # getting the tfidf value for each word
            vector3 += (vec4 * tf_idf3) # calculating tfidf weighted w2v
            tf_idf_weight3 += tf_idf3
    if tf_idf_weight3 != 0:
        vector3 /= tf_idf_weight3
        tfidf_w2v_train_title.append(vector3)
print(len(tfidf_w2v_train_title))
print(len(tfidf_w2v_train_title[0]))
```

```
100%|██████████████████████████████████████████████████
████████████████████████████████████| 33500/33500 [00:01<00:
00, 18039.42it/s]

33500
300
```

In [28]:

```
# For test data
tfidf_w2v_test_title = [] # the avg-w2v for each sentence/review is stored in this list
for sentence_test in tqdm(X_test['preprocessed_titles']): # for each review/sentence
    vector5 = np.zeros(300) # as word vectors are of zero length
    #tf_idf_weight5 =0; # num of words with a valid vector in the sentence/review
    for word5 in str(sentence_test).split(): # for each word in a review/sentence
        if (word5 in glove_words) and (word5 in tfidf_words_titles):
            vec6 = model[word5]  # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sen
tence.count(word)/len(sentence.split())))
            tf_idf5 = dictionary_title[word5]*(sentence_test.count(word5)/len(str(sente
nce_test).split())) # getting the tfidf value for each word
            vector5 += (vec6 * tf_idf5) # calculating tfidf weighted w2v
            tf_idf_weight5 += tf_idf5
    if tf_idf_weight5 != 0:
        vector5 /= tf_idf_weight5
        tfidf_w2v_test_title.append(vector5)
print(len(tfidf_w2v_test_title))
print(len(tfidf_w2v_test_title[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████
██████████████████████████████████████████████████| 16500/16500 [00:00<00:
00, 18648.59it/s]

16500
300
```

1.4 Make Data Model Ready: Response coding of categorical features

In [35]:

```
#https://www.geeksforgeeks.org/python-pandas-dataframe-mask/
#https://github.com/AnveshAeturi/Random-Forest---GBDT-on-Decision-Trees

def mask(df, key, value):
    return df[df[key] == value]

def class_prob(Xi,yi):

    """

    This function creates 2 dictionaries containing probability values for each subcate
gory
    belonging to positive & negative classes respectively.

    """

    f=Xi.values.tolist()
    o=yi.values.tolist()
    uni=Xi.unique() #corresponds to unique values in the column
    df=pd.DataFrame({'feature':f , 'label': o}) # creating a dataframe with column as f
eature & approval status as label
    pd.DataFrame.mask = mask

    count_accept = {};count_reject={};
    class_0_prob = {};class_1_prob={};

    for i in uni:
        count_0 = len(df.mask('feature', i).mask('label', 0))
        count_1 = len(df.mask('feature', i).mask('label', 1))
        total   =   count_0 + count_1
        prob_0 = count_0/total
        prob_1 = count_1/total
        count_accept[i] = count_1
        count_reject[i] = count_0
        class_0_prob[i] = prob_0
        class_1_prob[i] = prob_1

    return class_0_prob,class_1_prob
```

Train data

1.4.1 Response coding of School State

In [36]:

```
state_0_train, state_1_train = class_prob(X_train['school_state'],y_train)
```

In [37]:

```
state_neg_train = []
state_pos_train = []
for i in X_train['school_state']:
    state_neg_train.append(state_0_train[i])
    state_pos_train.append(state_1_train[i])
X_train['state_0'] = state_neg_train
X_train['state_1'] = state_pos_train
```

In [38]:

```
X_train.head(2)
```

Out[38]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state
17584	53097	p094559	03d781cdbaec2f81e5554b7a932a5f58	Mrs.	DE
35316	109795	p160679	51f1bde8f3739c46d6ed4204dd9cd367	Ms.	TN

2 rows × 30 columns

1.4.2 Response coding of teacher_prefix

In [39]:

```
prefix_0_train, prefix_1_train = class_prob(X_train['teacher_prefix'],y_train)
```

In [40]:

```
prefix_neg_train = []
prefix_pos_train = []
for i in X_train['teacher_prefix']:
    prefix_neg_train.append(prefix_0_train[i])
    prefix_pos_train.append(prefix_1_train[i])
X_train['prefix_0'] = prefix_neg_train
X_train['prefix_1'] = prefix_pos_train
```

In [41]:

```
X_train.head(2)
```

Out[41]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state
17584	53097	p094559	03d781cdbaec2f81e5554b7a932a5f58	Mrs.	DE
35316	109795	p160679	51f1bde8f3739c46d6ed4204dd9cd367	Ms.	TN

2 rows × 32 columns

In [40]:

```
X_train.columns
```

Out[40]:

```
Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',
       'project_submitted_datetime', 'project_grade_category', 'project_ti
tle',
       'project_essay_1', 'project_essay_2', 'project_essay_3',
       'project_essay_4', 'project_resource_summary',
       'teacher_number_of_previously_posted_projects', 'clean_categories',
       'clean_subcategories', 'essay', 'price', 'quantity',
       'Numerical digits in summary', 'titles_sw', 'essays_sw',
       'preprocessed_project_grade_category', 'preprocessed_essays',
       'preprocessed_titles', 'sentimental_score',
       'preprocessed_essay_word_count', 'preprocessed_title_word_count'],
      dtype='object')
```

1.4.3 Response coding of project_grade_category

In [42]:

```
pgc_0_train, pgc_1_train = class_prob(X_train['preprocessed_project_grade_category'],y_
train)
```

In [43]:

```
pgc_neg_train = []
pgc_pos_train = []
for i in X_train['preprocessed_project_grade_category']:
    pgc_neg_train.append(pgc_0_train[i])
    pgc_pos_train.append(pgc_1_train[i])
X_train['pgc_0'] = pgc_neg_train
X_train['pgc_1'] = pgc_pos_train
```

In [44]:

```
X_train.head(2)
```

Out[44]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state
17584	53097	p094559	03d781cdbaec2f81e5554b7a932a5f58	Mrs.	DE
35316	109795	p160679	51f1bde8f3739c46d6ed4204dd9cd367	Ms.	TN

2 rows × 34 columns

1.4.4 Response coding of clean_categories

In [45]:

```
cat_0_train, cat_1_train = class_prob(X_train['clean_categories'],y_train)
```

In [46]:

```
cat_neg_train = []
cat_pos_train = []
for i in X_train['clean_categories']:
    cat_neg_train.append(cat_0_train[i])
    cat_pos_train.append(cat_1_train[i])
X_train['cat_0'] = cat_neg_train
X_train['cat_1'] = cat_pos_train
```

In [47]:

```
X_train.head(2)
```

Out[47]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state
17584	53097	p094559	03d781cdbaec2f81e5554b7a932a5f58	Mrs.	DE
35316	109795	p160679	51f1bde8f3739c46d6ed4204dd9cd367	Ms.	TN

2 rows × 36 columns

1.4.5 Response coding of clean_subcategories

In [48]:

```
subcat_0_train, subcat_1_train = class_prob(X_train['clean_subcategories'],y_train)
```

In [49]:

```
subcat_neg_train = []
subcat_pos_train = []
for i in X_train['clean_subcategories']:
    subcat_neg_train.append(subcat_0_train[i])
    subcat_pos_train.append(subcat_1_train[i])
X_train['subcat_0'] = subcat_neg_train
X_train['subcat_1'] = subcat_pos_train
```

In [50]:

```
X_train.head(2)
```

Out[50]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state
17584	53097	p094559	03d781cdbaec2f81e5554b7a932a5f58	Mrs.	DE
35316	109795	p160679	51f1bde8f3739c46d6ed4204dd9cd367	Ms.	TN

2 rows × 38 columns

Test data

1.4.6 Response coding of School State

In [51]:

```
state_0_test, state_1_test = class_prob(X_test['school_state'],y_test)
```

In [52]:

```
state_neg_test = []
state_pos_test = []
for i in X_test['school_state']:
    state_neg_test.append(state_0_test[i])
    state_pos_test.append(state_1_test[i])
X_test['state_0'] = state_neg_test
X_test['state_1'] = state_pos_test
```

In [53]:

```
X_test.head(2)
```

Out[53]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state
45339	142677	p033686	17819cf8323ac4622b50d21275568ca1	Mrs.	OK
12659	164850	p010512	ace0ef76af891ab9b4cfd63a31e76c68	Ms.	MO

2 rows × 30 columns

1.4.7 Response coding of teacher_prefix

In [54]:

```
prefix_0_test, prefix_1_test = class_prob(X_test['teacher_prefix'],y_test)
```

In [55]:

```
prefix_neg_test = []
prefix_pos_test = []
for i in X_test['teacher_prefix']:
    prefix_neg_test.append(prefix_0_test[i])
    prefix_pos_test.append(prefix_1_test[i])
X_test['prefix_0'] = prefix_neg_test
X_test['prefix_1'] = prefix_pos_test
```

In [56]:

```
X_test.head(2)
```

Out[56]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state
45339	142677	p033686	17819cf8323ac4622b50d21275568ca1	Mrs.	OK
12659	164850	p010512	ace0ef76af891ab9b4cfd63a31e76c68	Ms.	MO

2 rows × 32 columns

1.4.8 Response coding of project_grade_category

In [57]:

```
pgc_0_test, pgc_1_test = class_prob(X_test['preprocessed_project_grade_category'],y_tes
t)
```

In [58]:

```
pgc_neg_test = []
pgc_pos_test = []
for i in X_test['preprocessed_project_grade_category']:
    pgc_neg_test.append(pgc_0_test[i])
    pgc_pos_test.append(pgc_1_test[i])
X_test['pgc_0'] = pgc_neg_test
X_test['pgc_1'] = pgc_pos_test
```

In [59]:

```
X_test.head(2)
```

Out[59]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state
45339	142677	p033686	17819cf8323ac4622b50d21275568ca1	Mrs.	OK
12659	164850	p010512	ace0ef76af891ab9b4cfd63a31e76c68	Ms.	MO

2 rows × 34 columns

1.4.9 Response coding of clean_categories

In [60]:

```
cat_0_test, cat_1_test = class_prob(X_test['clean_categories'],y_test)
```

In [61]:

```
cat_neg_test = []
cat_pos_test = []
for i in X_test['clean_categories']:
    cat_neg_test.append(cat_0_test[i])
    cat_pos_test.append(cat_1_test[i])
X_test['cat_0'] = cat_neg_test
X_test['cat_1'] = cat_pos_test
```

In [62]:

```
X_test.head(2)
```

Out[62]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state
45339	142677	p033686	17819cf8323ac4622b50d21275568ca1	Mrs.	OK
12659	164850	p010512	ace0ef76af891ab9b4cfd63a31e76c68	Ms.	MO

2 rows × 36 columns

1.4.10 Response coding of clean_subcategories

In [63]:

```
subcat_0_test, subcat_1_test = class_prob(X_test['clean_subcategories'],y_test)
```

In [64]:

```
subcat_neg_test = []
subcat_pos_test = []
for i in X_test['clean_subcategories']:
    subcat_neg_test.append(subcat_0_test[i])
    subcat_pos_test.append(subcat_1_test[i])
X_test['subcat_0'] = subcat_neg_test
X_test['subcat_1'] = subcat_pos_test
```

In [65]:

```
X_test.head(2)
```

Out[65]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_state
45339	142677	p033686	17819cf8323ac4622b50d21275568ca1	Mrs.	OK
12659	164850	p010512	ace0ef76af891ab9b4cfd63a31e76c68	Ms.	MO

2 rows × 38 columns

1.5 Make Data Model Ready: Encoding of numerical features

1.5.1 Encoding numerical features: Price

In [66]:

```
from sklearn.preprocessing import Normalizer
normalizer1 = Normalizer()
# normalizer.fit(X_train['price'].values)
#this will rise an error Expected 2D array, got 1D array instead:
normalizer1.fit(X_train['price'].values.reshape(-1,1))

X_train_price_norm = normalizer1.transform(X_train['price'].values.reshape(-1,1))
X_test_price_norm = normalizer1.transform(X_test['price'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)
print(X_test_price_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
==========================================================================
==========================
```

1.5.2 Encoding numerical features: Quantity

In [68]:

```
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_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_test_quantity_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
==========================================================================
==========================
```

1.5.3 Encoding numerical features: teacher_number_of_previously_posted_projects

In [69]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

X_train_projects_norm = normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))
X_test_projects_norm = normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_projects_norm.shape, y_train.shape)
print(X_test_projects_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
====================================================================================================
```

1.5.4 Encoding numerical features: sentimental_score

In [70]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
normalizer.fit(X_train['sentimental_score'].values.reshape(-1,1))

X_train_senti_norm = normalizer.transform(X_train['sentimental_score'].values.reshape(-1,1))
X_test_senti_norm = normalizer.transform(X_test['sentimental_score'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_senti_norm.shape, y_train.shape)
print(X_test_senti_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
====================================================================================================
```

1.5.5 Encoding numerical features: preprocessed_essay_word_count

In [71]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
normalizer.fit(X_train['preprocessed_essay_word_count'].values.reshape(-1,1))

X_train_ewc_norm = normalizer.transform(X_train['preprocessed_essay_word_count'].values
.reshape(-1,1))
X_test_ewc_norm = normalizer.transform(X_test['preprocessed_essay_word_count'].values.r
eshape(-1,1))

print("After vectorization")
print(X_train_ewc_norm.shape, y_train.shape)
print(X_test_ewc_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorization
(33500, 1) (33500,)
(16500, 1) (16500,)
==========================================================================
==========================
```

1.5.6 Encoding numerical features: preprocessed_title_word_count

In [72]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
normalizer.fit(X_train['preprocessed_title_word_count'].values.reshape(-1,1))

X_train_twc_norm = normalizer.transform(X_train['preprocessed_title_word_count'].values
.reshape(-1,1))
X_test_twc_norm = normalizer.transform(X_test['preprocessed_title_word_count'].values.r
eshape(-1,1))

print("After vectorization")
print(X_train_twc_norm.shape, y_train.shape)
print(X_test_twc_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorization
(33500, 1) (33500,)
(16500, 1) (16500,)
==========================================================================
==========================
```

1.5.7 Encoding numerical features: clean_categories

In [73]:

```
normalizer = Normalizer()

normalizer.fit(X_train["cat_0"].values.reshape(-1,1))  #fit has to be done only on Trai
n data

cat_0_train_normalized = normalizer.transform(X_train["cat_0"].values.reshape(-1,1))
cat_0_test_normalized = normalizer.transform(X_test["cat_0"].values.reshape(-1,1))

print("After vectorizations")
print(cat_0_train_normalized.shape, y_train.shape)
print(cat_0_test_normalized.shape, y_test.shape)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
```

In [74]:

```
normalizer = Normalizer()

normalizer.fit(X_train["cat_1"].values.reshape(-1,1))  #fit has to be done only on Trai
n data

cat_1_train_normalized = normalizer.transform(X_train["cat_1"].values.reshape(-1,1))
cat_1_test_normalized = normalizer.transform(X_test["cat_1"].values.reshape(-1,1))

print("After vectorizations")
print(cat_1_train_normalized.shape, y_train.shape)
print(cat_1_test_normalized.shape, y_test.shape)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
```

1.5.8 Encoding numerical features: clean_subcategories

In [75]:

```
normalizer = Normalizer()

normalizer.fit(X_train["subcat_0"].values.reshape(-1,1))  #fit has to be done only on T
rain data

subcat_0_train_normalized = normalizer.transform(X_train["subcat_0"].values.reshape(-1,
1))
subcat_0_test_normalized = normalizer.transform(X_test["subcat_0"].values.reshape(-1,1
))

print("After vectorizations")
print(subcat_0_train_normalized.shape, y_train.shape)
print(subcat_0_test_normalized.shape, y_test.shape)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
```

In [76]:

```
normalizer = Normalizer()

normalizer.fit(X_train["subcat_1"].values.reshape(-1,1))  #fit has to be done only on T
rain data

subcat_1_train_normalized = normalizer.transform(X_train["subcat_1"].values.reshape(-1,
1))
subcat_1_test_normalized = normalizer.transform(X_test["subcat_1"].values.reshape(-1,1
))

print("After vectorizations")
print(subcat_1_train_normalized.shape, y_train.shape)
print(subcat_1_test_normalized.shape, y_test.shape)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
```

1.5.9 Encoding numerical features: school_state

In [77]:

```
normalizer = Normalizer()

normalizer.fit(X_train["state_0"].values.reshape(-1,1))  #fit has to be done only on Tr
ain data

state_0_train_normalized = normalizer.transform(X_train["state_0"].values.reshape(-1,1
))
state_0_test_normalized = normalizer.transform(X_test["state_0"].values.reshape(-1,1))

print("After vectorizations")
print(state_0_train_normalized.shape, y_train.shape)
print(state_0_test_normalized.shape, y_test.shape)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
```

In [78]:

```
normalizer = Normalizer()

normalizer.fit(X_train["state_1"].values.reshape(-1,1))  #fit has to be done only on Tr
ain data

state_1_train_normalized = normalizer.transform(X_train["state_1"].values.reshape(-1,1
))
state_1_test_normalized = normalizer.transform(X_test["state_1"].values.reshape(-1,1))

print("After vectorizations")
print(state_1_train_normalized.shape, y_train.shape)
print(state_1_test_normalized.shape, y_test.shape)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
```

1.5.10 Encoding numerical features: teacher_prefix

In [79]:

```
normalizer = Normalizer()

normalizer.fit(X_train["prefix_0"].values.reshape(-1,1))  #fit has to be done only on T
rain data

prefix_0_train_normalized = normalizer.transform(X_train["prefix_0"].values.reshape(-1,
1))
prefix_0_test_normalized = normalizer.transform(X_test["prefix_0"].values.reshape(-1,1
))

print("After vectorizations")
print(prefix_0_train_normalized.shape, y_train.shape)
print(prefix_0_test_normalized.shape, y_test.shape)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
```

In [80]:

```
normalizer = Normalizer()

normalizer.fit(X_train["prefix_1"].values.reshape(-1,1))  #fit has to be done only on T
rain data

prefix_1_train_normalized = normalizer.transform(X_train["prefix_1"].values.reshape(-1,
1))
prefix_1_test_normalized = normalizer.transform(X_test["prefix_1"].values.reshape(-1,1
))

print("After vectorizations")
print(prefix_1_train_normalized.shape, y_train.shape)
print(prefix_1_test_normalized.shape, y_test.shape)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
```

1.5.11 Encoding numerical features: project_grade_category

In [81]:

```
normalizer = Normalizer()

normalizer.fit(X_train["pgc_0"].values.reshape(-1,1))  #fit has to be done only on Trai
n data

pgc_0_train_normalized = normalizer.transform(X_train["pgc_0"].values.reshape(-1,1))
pgc_0_test_normalized = normalizer.transform(X_test["pgc_0"].values.reshape(-1,1))

print("After vectorizations")
print(pgc_0_train_normalized.shape, y_train.shape)
print(pgc_0_test_normalized.shape, y_test.shape)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
```

In [82]:

```
normalizer = Normalizer()

normalizer.fit(X_train["pgc_1"].values.reshape(-1,1))  #fit has to be done only on Trai
n data

pgc_1_train_normalized = normalizer.transform(X_train["pgc_1"].values.reshape(-1,1))
pgc_1_test_normalized = normalizer.transform(X_test["pgc_1"].values.reshape(-1,1))

print("After vectorizations")
print(pgc_1_train_normalized.shape, y_train.shape)
print(pgc_1_test_normalized.shape, y_test.shape)
```

```
After vectorizations
(33500, 1) (33500,)
(16500, 1) (16500,)
```

1.6 Concatinating all the features

1.6.1 Set 1: Using categorical features + numerical features + preprocessed_titles(BOW) + preprocessed_essays(BOW)

In [96]:

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

X_tr_bow = hstack((X_train_essay_bow, X_train_title_bow, X_train_price_norm, X_train_qu
antity_norm, X_train_projects_norm, X_train_senti_norm, X_train_ewc_norm, X_train_twc_n
orm, cat_0_train_normalized, cat_1_train_normalized, subcat_0_train_normalized, subcat_
1_train_normalized, state_0_train_normalized, state_1_train_normalized, prefix_0_train_
normalized, prefix_1_train_normalized, pgc_0_train_normalized, pgc_1_train_normalized
)).tocsr()


X_test_bow = hstack((X_test_essay_bow, X_test_title_bow, X_test_price_norm, X_test_quan
tity_norm, X_test_projects_norm, X_test_senti_norm, X_test_ewc_norm, X_test_twc_norm, c
at_0_test_normalized, cat_1_test_normalized, subcat_0_test_normalized, subcat_1_test_no
rmalized, state_0_test_normalized, state_1_test_normalized, prefix_0_test_normalized, p
refix_1_test_normalized, pgc_0_test_normalized, pgc_1_test_normalized )).tocsr()

print("Final Data Matrix")
print(X_tr_bow.shape, y_train.shape)
print(X_test_bow.shape, y_test.shape)
```

```
Final Data Matrix
(33500, 7356) (33500,)
(16500, 7356) (16500,)
```

In [86]:

```
type(X_test_bow)
```

Out[86]:

```
scipy.sparse.csr.csr_matrix
```

In [107]:

```
type(y_train)
```

Out[107]:

```
pandas.core.series.Series
```

In [110]:

```
a=y_train.values
a
```

Out[110]:

```
array([1, 0, 1, ..., 1, 1, 1], dtype=int64)
```

In [111]:

```
type(a)
```

Out[111]:

```
numpy.ndarray
```

In [117]:

```
b=y_test.values
b
```

Out[117]:

```
array([1, 1, 1, ..., 1, 1, 1], dtype=int64)
```

In [118]:

```
#https://www.geeksforgeeks.org/numpy-save/
np.save('y_train', a)
np.save('y_test', b)
```

In [ ]:

```
#b = np.load('geekfile.npy')
```

In [112]:

```
# https://stackoverflow.com/questions/8955448/save-load-scipy-sparse-csr-matrix-in-port
able-data-format
from scipy import sparse

sparse.save_npz("X_tr_bow.npz", X_tr_bow)
```

In [113]:

```
sparse.save_npz("X_test_bow.npz", X_test_bow)
```

1.4.5.2 Set 2: Using categorical features + numerical features + preprocessed_titles(TFIDF) + preprocessed_essays(TFIDF)

In [87]:

```
X_tr_tfidf = hstack((X_train_essay_tfidf, X_train_titles_tfidf, X_train_price_norm, X_train_quantity_norm, X_train_projects_norm, X_train_senti_norm, X_train_ewc_norm, X_train_twc_norm, cat_0_train_normalized, cat_1_train_normalized, subcat_0_train_normalized, subcat_1_train_normalized, state_0_train_normalized, state_1_train_normalized, prefix_0_train_normalized, prefix_1_train_normalized, pgc_0_train_normalized, pgc_1_train_normalized )).tocsr()


X_test_tfidf = hstack((X_test_essay_tfidf, X_test_titles_tfidf, X_test_price_norm, X_test_quantity_norm, X_test_projects_norm, X_test_senti_norm, X_test_ewc_norm, X_test_twc_norm, cat_0_test_normalized, cat_1_test_normalized, subcat_0_test_normalized, subcat_1_test_normalized, state_0_test_normalized, state_1_test_normalized, prefix_0_test_normalized, prefix_1_test_normalized, pgc_0_test_normalized, pgc_1_test_normalized )).tocsr()

print("Final Data Matrix")
print(X_tr_tfidf.shape, y_train.shape)
print(X_test_tfidf.shape, y_test.shape)
```

```
Final Data Matrix
(33500, 7356) (33500,)
(16500, 7356) (16500,)
```

In [114]:

```
sparse.save_npz("X_tr_tfidf.npz", X_tr_tfidf)
sparse.save_npz("X_test_tfidf.npz", X_test_tfidf)
```

- Note: W2V vectorization creates dense vectors. For h-stack to work the vector must be sparse. If not it will throw an error saying "could not broadcast input array from shape (33500,1) into shape (33500)"
- Hence I used coomatrix to convert dense features of set3 & 4 to a sparse one.
- Ref: https://blog.csdn.net/w55100/article/details/90369779 (https://blog.csdn.net/w55100/article/details/90369779)

1.4.5.3 Set 3: Using categorical features + numerical features + preprocessed_titles(Avg W2V) + preprocessed_essays(Avg W2V)

In [105]:

```
X_tr_avgw2v = hstack((coo_matrix(sent_vectors_train), coo_matrix(avg_w2v_essay_train), coo_matrix(X_train_price_norm), coo_matrix(X_train_quantity_norm), coo_matrix(X_train_projects_norm), coo_matrix(X_train_senti_norm), coo_matrix(X_train_ewc_norm), coo_matrix(X_train_twc_norm), coo_matrix(cat_0_train_normalized), coo_matrix(cat_1_train_normalized), coo_matrix(subcat_0_train_normalized), coo_matrix(subcat_1_train_normalized), coo_matrix(state_0_train_normalized), coo_matrix(state_1_train_normalized), coo_matrix(prefix_0_train_normalized), coo_matrix(prefix_1_train_normalized), coo_matrix(pgc_0_train_normalized), coo_matrix(pgc_1_train_normalized))).tocsr()


X_test_avgw2v = hstack((coo_matrix(sent_vectors_test), coo_matrix(avg_w2v_essay_test), coo_matrix(X_test_price_norm), coo_matrix(X_test_quantity_norm), coo_matrix(X_test_projects_norm), coo_matrix(X_test_senti_norm), coo_matrix(X_test_ewc_norm), coo_matrix(X_test_twc_norm), coo_matrix(cat_0_test_normalized), coo_matrix(cat_1_test_normalized), coo_matrix(subcat_0_test_normalized), coo_matrix(subcat_1_test_normalized), coo_matrix(state_0_test_normalized), coo_matrix(state_1_test_normalized), coo_matrix(prefix_0_test_normalized), coo_matrix(prefix_1_test_normalized), coo_matrix(pgc_0_test_normalized), coo_matrix(pgc_1_test_normalized))).tocsr()

print("Final Data Matrix")
print(X_tr_avgw2v.shape, y_train.shape)
print(X_test_avgw2v.shape, y_test.shape)
```

```
Final Data Matrix
(33500, 366) (33500,)
(16500, 366) (16500,)
```

In [115]:

```
sparse.save_npz("X_tr_avgw2v.npz", X_tr_avgw2v)
sparse.save_npz("X_test_avgw2v.npz", X_test_avgw2v)
```

1.4.5.4 Set 4: Using categorical features + numerical features + preprocessed_titles(TFIDF W2V) + preprocessed_essays(TFIDF W2V)

In [106]:

```
X_tr_tfidf_w2v = hstack((coo_matrix(tfidf_w2v_train_essay), coo_matrix(tfidf_w2v_train_
title), coo_matrix(X_train_price_norm), coo_matrix(X_train_quantity_norm), coo_matrix(X
_train_projects_norm), coo_matrix(X_train_senti_norm), coo_matrix(X_train_ewc_norm), co
o_matrix(X_train_twc_norm), coo_matrix(cat_0_train_normalized), coo_matrix(cat_1_train_
normalized), coo_matrix(subcat_0_train_normalized), coo_matrix(subcat_1_train_normalize
d), coo_matrix(state_0_train_normalized), coo_matrix(state_1_train_normalized), coo_mat
rix(prefix_0_train_normalized), coo_matrix(prefix_1_train_normalized), coo_matrix(pgc_0
_train_normalized), coo_matrix(pgc_1_train_normalized))).tocsr()

X_test_tfidf_w2v = hstack((coo_matrix(tfidf_w2v_test_essay), coo_matrix(tfidf_w2v_test_
title), coo_matrix(X_test_price_norm), coo_matrix(X_test_quantity_norm), coo_matrix(X_t
est_projects_norm), coo_matrix(X_test_senti_norm), coo_matrix(X_test_ewc_norm), coo_mat
rix(X_test_twc_norm), coo_matrix(cat_0_test_normalized), coo_matrix(cat_1_test_normaliz
ed), coo_matrix(subcat_0_test_normalized), coo_matrix(subcat_1_test_normalized), coo_ma
trix(state_0_test_normalized), coo_matrix(state_1_test_normalized), coo_matrix(prefix_0
_test_normalized), coo_matrix(prefix_1_test_normalized), coo_matrix(pgc_0_test_normaliz
ed), coo_matrix(pgc_1_test_normalized))).tocsr()

print("Final Data Matrix")
print(X_tr_tfidf_w2v.shape, y_train.shape)
print(X_test_tfidf_w2v.shape, y_test.shape)
```

```
Final Data Matrix
(33500, 616) (33500,)
(16500, 616) (16500,)
```

In [116]:

```
sparse.save_npz("X_tr_tfidf_w2v.npz", X_tr_tfidf_w2v)
sparse.save_npz("X_test_tfidf_w2v.npz", X_test_tfidf_w2v)
```

Loading the concatenated features as I'm running the models in GCP

In [2]:

```
from scipy import sparse
import numpy as np
X_tr_bow = sparse.load_npz("X_tr_bow.npz")
X_test_bow= sparse.load_npz("X_test_bow.npz")
X_tr_tfidf= sparse.load_npz("X_tr_tfidf.npz")
X_test_tfidf = sparse.load_npz("X_test_tfidf.npz")
X_tr_avgw2v= sparse.load_npz("X_tr_avgw2v.npz")
X_test_avgw2v= sparse.load_npz("X_test_avgw2v.npz")
X_tr_tfidf_w2v= sparse.load_npz("X_tr_tfidf_w2v.npz")
X_test_tfidf_w2v= sparse.load_npz("X_test_tfidf_w2v.npz")
y_train = np.load('y_train.npy')
y_test = np.load('y_test.npy')
```

2. Applying RF

2.1 Set 1: BOW featurization

2.1.1 Hyper parameter tuning

In [6]:

```
%%time
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification

rf_bow = RandomForestClassifier(criterion='gini',class_weight = 'balanced')
#https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassif
ier.html
parameters = {'n_estimators': [4, 8, 16, 32, 64, 100],'max_depth': [4, 6, 8, 10, 20, 25
]}  #https://medium.com/all-things-ai/in-depth-parameter-tuning-for-random-forest-d67bb
7e920d
clf1 = RandomizedSearchCV(rf_bow, parameters, cv=10, scoring='roc_auc',return_train_sco
re=True,n_jobs=-1)
rs1 = clf1.fit(X_tr_bow, y_train)
```

```
CPU times: user 13.6 s, sys: 248 ms, total: 13.9 s
Wall time: 1min 27s
```

In [7]:

```
df=pd.DataFrame(clf1.cv_results_)
df.head(2)
```

Out[7]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_max_depth
0	11.505452	0.225072	0.683051	0.959581	20
1	0.421158	0.030518	0.608860	0.641260	4

2 rows × 32 columns

2.1.2 3D-Plot

In [3]:

```
%matplotlib inline
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np
```

In [4]:

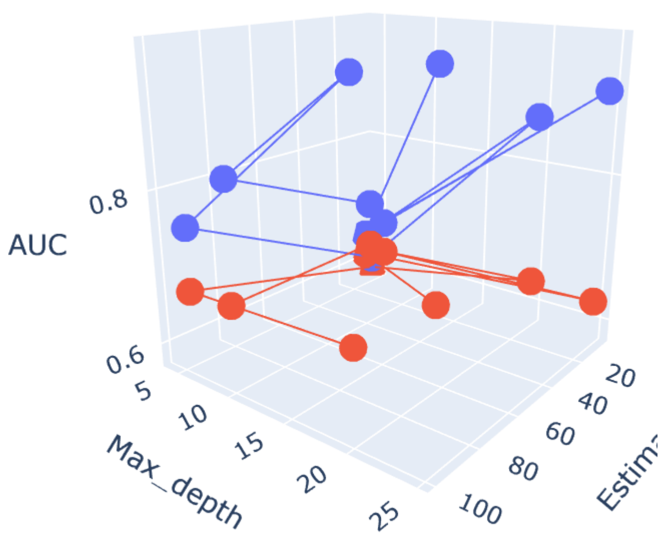
```
def enable_plotly_in_cell():
    import IPython
    from plotly.offline import init_notebook_mode
    display(IPython.core.display.HTML('''<script src="/static/components/requirejs/requ
ire.js"></script>'''))
    init_notebook_mode(connected=False)
```

In [10]:

```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=df['param_n_estimators'],y=df['param_max_depth'],z=df['mean_train_score'], name = 'train')
trace2 = go.Scatter3d(x=df['param_n_estimators'],y=df['param_max_depth'],z=df['mean_test_score'], name = 'Cross validation')
data = [trace1, trace2]
enable_plotly_in_cell()

layout = go.Layout(scene = dict(
        xaxis = dict(title='Estimators'),
        yaxis = dict(title='Max_depth'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```



2.1.3 Best Hyperparameters

In [11]:

```
print(clf1.best_estimator_)
print('CV score on train data:', {clf1.score(X_tr_bow,y_train)})
print('Mean cross-validated score of the best_estimator :', {clf1.best_score_})
```

```
RandomForestClassifier(bootstrap=True, class_weight='balanced',
                       criterion='gini', max_depth=20, max_features='aut
o',
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       n_estimators=100, n_jobs=None, oob_score=False,
                       random_state=None, verbose=0, warm_start=False)
CV score on train data: {0.9645644963969895}
Mean cross-validated score of the best_estimator : {0.6876762481536259}
```

In [12]:

```
best_parameters_bow = {'n_estimators': [100],'max_depth': [6]}
```

2.1.4 Applying Best Hyperparameters on train & test data & plotting ROC curve

In [5]:

```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates
 of the positive class
    # not the predicted outputs
    y_data_pred = []
    pred_labels=[]
    tr_loop = data.shape[0] - data.shape[0]%1000;
    # consider you X_tr shape is 49041, then your tr_loop will be 49041 - 49041%1000 =
 49000
    # in this for loop we will iterate until the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1]) # we will be predict
ing for the last data points
        pred_labels.extend(clf.predict(data[i:i+1000]))
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])
        pred_labels.extend(clf.predict(data[tr_loop:]))

    return y_data_pred,pred_labels
```

In [14]:

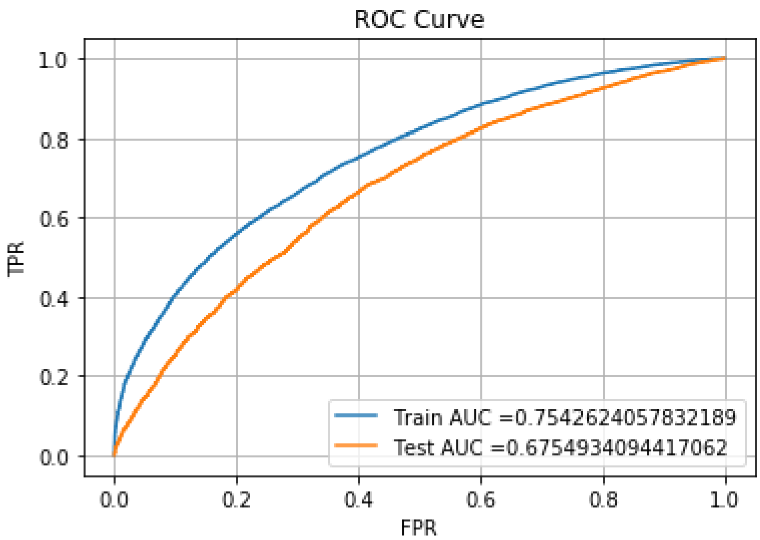
```
rf_best= RandomForestClassifier(n_estimators= 100 , criterion='gini', max_depth= 6, cla
ss_weight = 'balanced')

rf_best.fit(X_tr_bow, y_train)

y_train_pred_bow_best,pred_labels_train = batch_predict(rf_best, X_tr_bow)
y_test_pred_bow_best,pred_labels_test = batch_predict(rf_best, X_test_bow)

train_tpr_bow, train_fpr_bow, tr_thresholds_bow = roc_curve(y_train, y_train_pred_bow_b
est)
test_tpr_bow, test_fpr_bow, te_thresholds_bow = roc_curve(y_test, y_test_pred_bow_best)

plt.plot(train_tpr_bow, train_fpr_bow,label="Train AUC ="+str(auc(train_tpr_bow, train_
fpr_bow)))
plt.plot(test_tpr_bow, test_fpr_bow, label="Test AUC ="+str(auc(test_tpr_bow, test_fpr_
bow)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve")
plt.grid()
plt.show()
```



2.1.5 Plot confusion matrix

In [6]:

```
## we will pick a threshold that will give the least fpr

def find_best_threshold(threshold, fpr, tpr):
    t = threshold[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("The maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)),"for threshold", np.roun
d(t,3))
    return t

def predict_with_best_t(proba, threshold):
    predictions = []
    for i in proba:
        if i>=threshold:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions

print("="*100)
```

```
===========================================================================
==========================
```

In [7]:

```
#function to get heatmap of confusion matrix
# Reference: https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-ma
trix

def cm_heatmap(cm):
    #y_pred = clf.predict(X_te)
    df_cm = pd.DataFrame(cm, range(2),range(2))
    df_cm.columns = ['Predicted NO','Predicted YES']
    df_cm = df_cm.rename({0: 'Actual NO', 1: 'Actual YES'})
    sns.set(font_scale=1.4)#for label size
    sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='d')
```

In [17]:

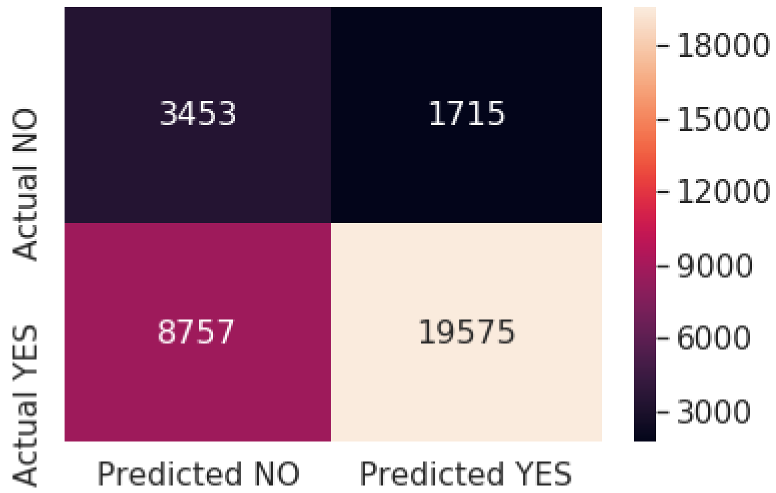
```
from sklearn.metrics import confusion_matrix
best_t_bow = find_best_threshold(tr_thresholds_bow, train_fpr_bow, train_tpr_bow)
print("Train confusion matrix")
cm_train_bow=confusion_matrix(y_train, predict_with_best_t(y_train_pred_bow_best, best_
t_bow))
print(cm_train_bow)
print("Test confusion matrix")
cm_test_bow=confusion_matrix(y_test, predict_with_best_t(y_test_pred_bow_best, best_t_b
ow))
print(cm_test_bow)
```

```
The maximum value of tpr*(1-fpr) 0.10256985367878176 for threshold 0.496
Train confusion matrix
[[ 3453  1715]
 [ 8757 19575]]
Test confusion matrix
[[1474 1072]
 [4393 9561]]
```

In [18]:

```
# confusion matrix heatmap for train data
print("Train confusion matrix heatmap")
cm_heatmap(cm_train_bow)
```

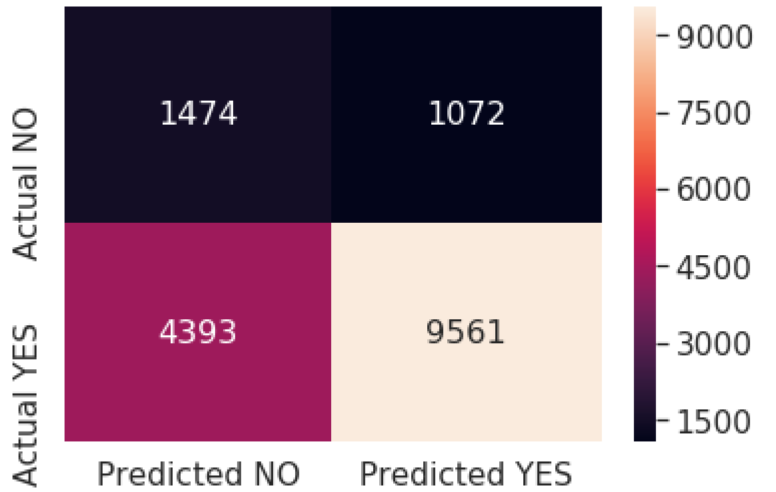
```
Train confusion matrix heatmap
```



In [19]:

```
# confusion matrix heatmap for test data
print("Test confusion matrix heatmap")
cm_heatmap(cm_test_bow)
```

```
Test confusion matrix heatmap
```



2.2 Set 2: TFIDF featurization

2.2.1 Hyper parameter tuning

In [20]:

```
%%time
rf_tfidf = RandomForestClassifier(criterion='gini',class_weight = 'balanced')
parameters = {'n_estimators': [4, 8, 16, 32, 64, 100],'max_depth': [4, 6, 8, 10, 20, 25
]}
clf2 = RandomizedSearchCV(rf_tfidf, parameters, cv=9, scoring='roc_auc',return_train_sc
ore=True,n_jobs=-1)
rs2 = clf2.fit(X_tr_tfidf, y_train)
```

```
CPU times: user 4.34 s, sys: 340 ms, total: 4.68 s
Wall time: 52.5 s
```

In [21]:

```
df1=pd.DataFrame(clf2.cv_results_)
df1.head(2)
```

Out[21]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_max_depth	p
0	11.245168	0.130586	0.653550	0.988432	25	
1	0.643692	0.020888	0.593261	0.694117	10	

2 rows × 30 columns

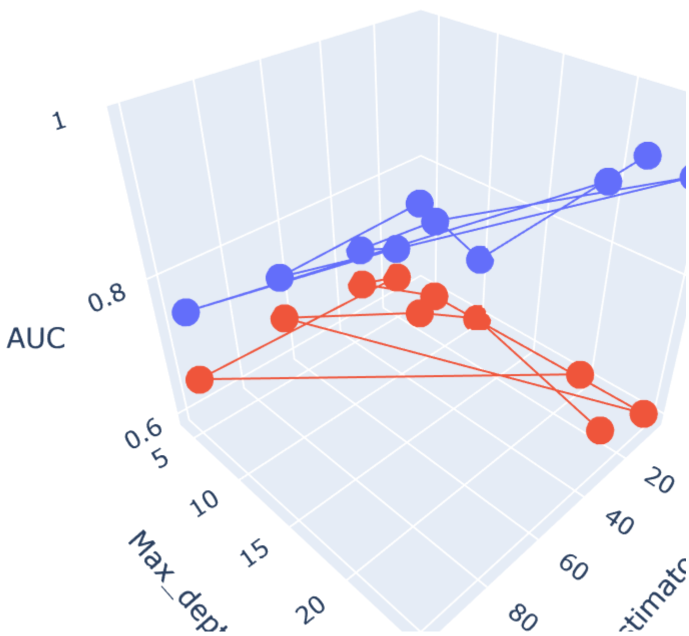
2.2.2 3D-Plot

In [22]:

```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=df1['param_n_estimators'],y=df1['param_max_depth'],z=df1['mean_
train_score'], name = 'train')
trace2 = go.Scatter3d(x=df1['param_n_estimators'],y=df1['param_max_depth'],z=df1['mean_
test_score'], name = 'Cross validation')
data = [trace1, trace2]
enable_plotly_in_cell()

layout = go.Layout(scene = dict(
        xaxis = dict(title='Estimators'),
        yaxis = dict(title='Max_depth'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```



2.2.3 Best Hyperparameters

In [23]:

```
print(clf2.best_estimator_)
print('CV score on train data:', {clf2.score(X_tr_tfidf,y_train)})
print('Mean cross-validated score of the best_estimator :', {clf2.best_score_})
```

```
RandomForestClassifier(bootstrap=True, class_weight='balanced',
                       criterion='gini', max_depth=6, max_features='auto',
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       n_estimators=100, n_jobs=None, oob_score=False,
                       random_state=None, verbose=0, warm_start=False)
CV score on train data: {0.7641645722774497}
Mean cross-validated score of the best_estimator : {0.6793788521640505}
```

In [24]:

```
best_parameters_tfidf = {'n_estimators': [100],'max_depth': [6]}
```

2.2.4 Applying Best Hyperparameters on train & test data & plotting ROC curve

In [25]:

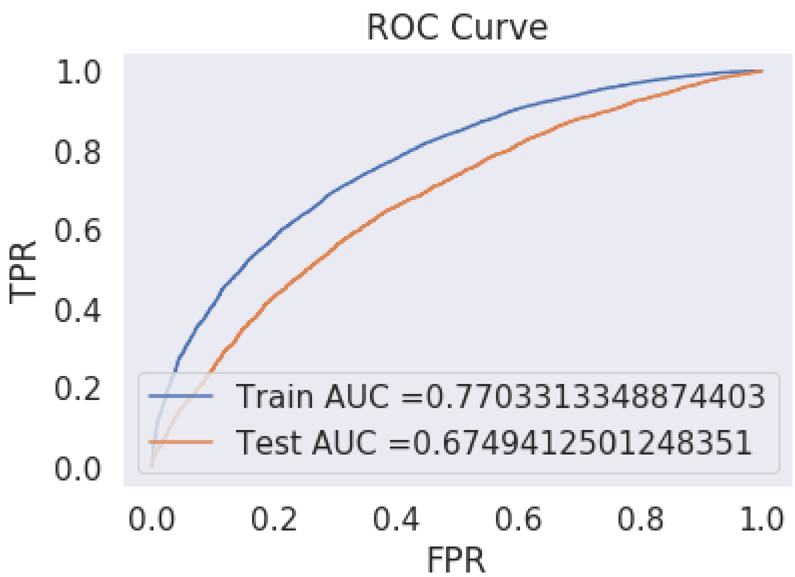
```
rf_best_tfidf= RandomForestClassifier(n_estimators= 100 , criterion='gini', max_depth=
6, class_weight = 'balanced')

rf_best_tfidf.fit(X_tr_tfidf, y_train)

y_train_pred_tfidf_best,pred_labels_train = batch_predict(rf_best_tfidf, X_tr_tfidf)
y_test_pred_tfidf_best,pred_labels_test = batch_predict(rf_best_tfidf, X_test_tfidf)

train_tpr_tfidf, train_fpr_tfidf, tr_thresholds_tfidf = roc_curve(y_train, y_train_pred
_tfidf_best)
test_tpr_tfidf, test_fpr_tfidf, te_thresholds_tfidf = roc_curve(y_test, y_test_pred_tfi
df_best)

plt.plot(train_tpr_tfidf, train_fpr_tfidf,label="Train AUC ="+str(auc(train_tpr_tfidf,
train_fpr_tfidf)))
plt.plot(test_tpr_tfidf, test_fpr_tfidf, label="Test AUC ="+str(auc(test_tpr_tfidf, tes
t_fpr_tfidf)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve")
plt.grid()
plt.show()
```



2.2.5 Plot confusion matrix

In [26]:

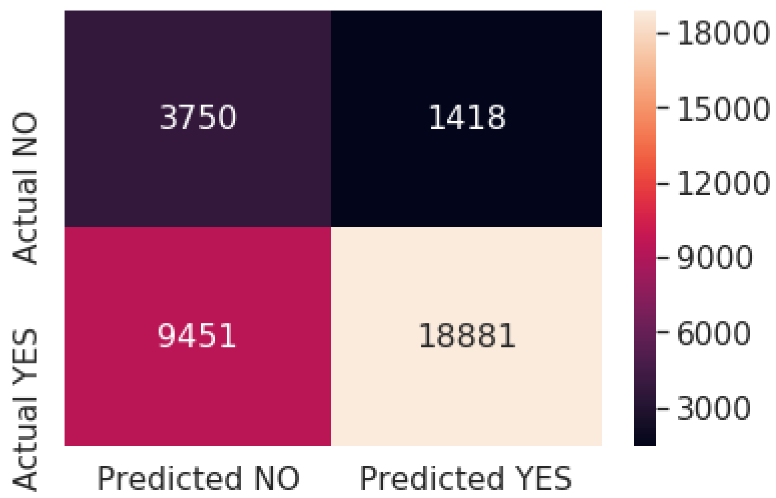
```
best_t_tfidf = find_best_threshold(tr_thresholds_tfidf, train_fpr_tfidf, train_tpr_tfid
f)
print("Train confusion matrix")
cm_train_tfidf=confusion_matrix(y_train, predict_with_best_t(y_train_pred_tfidf_best, b
est_t_tfidf))
print(cm_train_tfidf)
print("Test confusion matrix")
cm_test_tfidf=confusion_matrix(y_test, predict_with_best_t(y_test_pred_tfidf_best, best
_t_tfidf))
print(cm_test_tfidf)
```

```
The maximum value of tpr*(1-fpr) 0.09152805970690735 for threshold 0.502
Train confusion matrix
[[ 3750  1418]
 [ 9451 18881]]
Test confusion matrix
[[1567  979]
 [4913 9041]]
```

In [27]:

```
# confusion matrix heatmap for train data
print("Train confusion matrix heatmap")
cm_heatmap(cm_train_tfidf)
```

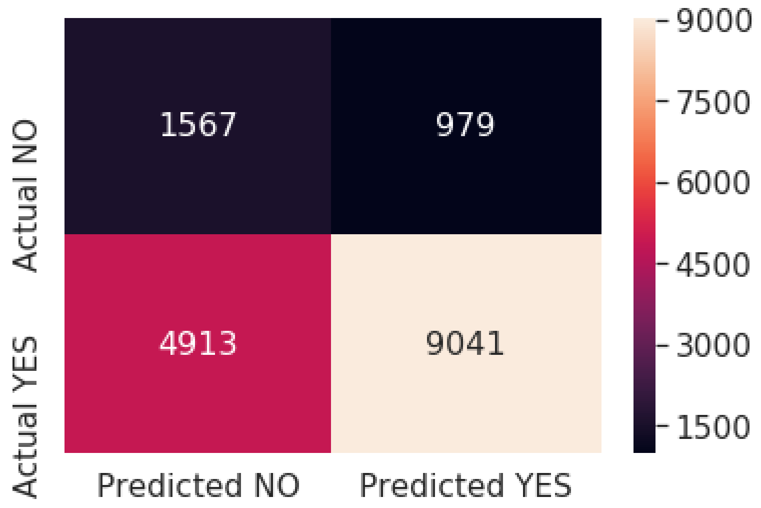
```
Train confusion matrix heatmap
```



In [28]:

```
# confusion matrix heatmap for test data
print("Test confusion matrix heatmap")
cm_heatmap(cm_test_tfidf)
```

```
Test confusion matrix heatmap
```



2.3 Set 3: AvgW2V featurization

2.3.1 Hyper parameter tuning

In [29]:

```
rf_avg = RandomForestClassifier(criterion='gini',class_weight = 'balanced')
parameters = {'n_estimators': [4, 8, 16, 32, 64, 100],'max_depth': [4, 6, 8, 10, 20, 25
]}
clf3 = RandomizedSearchCV(rf_avg, parameters, cv=10, scoring='roc_auc',return_train_sco
re=True,n_jobs=-1)
rs3 = clf3.fit(X_tr_avgw2v, y_train)
```

In [30]:

```
df2=pd.DataFrame(clf3.cv_results_)
df2.head(2)
```

Out[30]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_max_depth
0	4.672281	0.028756	0.609083	0.778914	8
1	8.380357	0.038161	0.624012	0.827517	8

2 rows × 32 columns

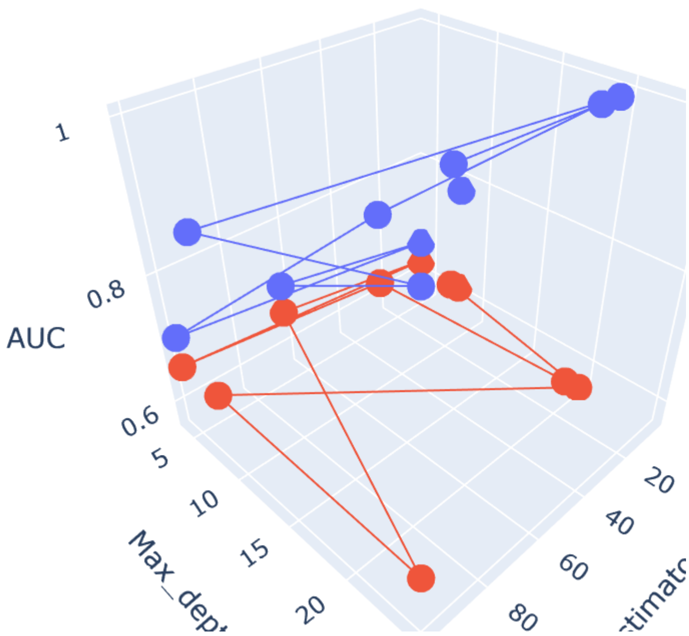
2.3.2 3D-Plot

In [31]:

```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=df2['param_n_estimators'],y=df2['param_max_depth'],z=df2['mean_train_score'], name = 'train')
trace2 = go.Scatter3d(x=df2['param_n_estimators'],y=df2['param_max_depth'],z=df2['mean_test_score'], name = 'Cross validation')
data = [trace1, trace2]
enable_plotly_in_cell()

layout = go.Layout(scene = dict(
        xaxis = dict(title='Estimators'),
        yaxis = dict(title='Max_depth'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```



2.3.3 Best Hyperparameters

In [84]:

```
print(clf3.best_estimator_)
print('Score on train data :', {clf3.score(X_tr_avgw2v,y_train)})
print('Mean cross-validated score of the best_estimator :', {clf3.best_score_})
```

```
RandomForestClassifier(bootstrap=True, class_weight='balanced',
                       criterion='gini', max_depth=6, max_features='auto',
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       n_estimators=32, n_jobs=None, oob_score=False,
                       random_state=None, verbose=0, warm_start=False)
Score on train data : {0.7668597853885528}
Mean cross-validated score of the best_estimator : {0.6587822671723945}
```

In [87]:

```
best_parameters_tfidf = {'n_estimators': [32],'max_depth': [6]}
```

2.3.4 Applying Best Hyperparameters on train & test data & plotting ROC curve

In [88]:

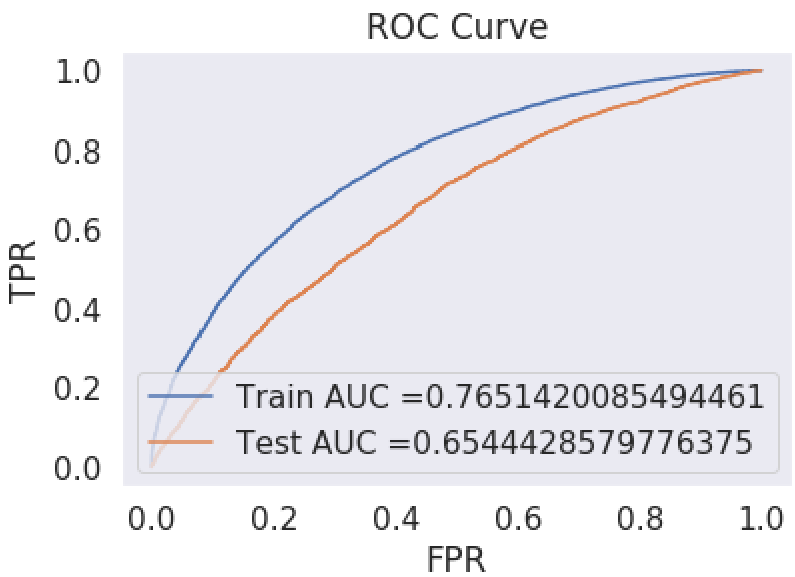
```
rf_best_avg= RandomForestClassifier(n_estimators= 32 , criterion='gini', max_depth= 6,
class_weight = 'balanced')

rf_best_avg.fit(X_tr_avgw2v, y_train)

y_train_pred_avg_best,pred_labels_train = batch_predict(rf_best_avg, X_tr_avgw2v)
y_test_pred_avg_best,pred_labels_test = batch_predict(rf_best_avg, X_test_avgw2v)

train_tpr_avg, train_fpr_avg, tr_thresholds_avg = roc_curve(y_train, y_train_pred_avg_b
est)
test_tpr_avg, test_fpr_avg, te_thresholds_avg = roc_curve(y_test, y_test_pred_avg_best)

plt.plot(train_tpr_avg, train_fpr_avg,label="Train AUC ="+str(auc(train_tpr_avg, train_
fpr_avg)))
plt.plot(test_tpr_avg, test_fpr_avg, label="Test AUC ="+str(auc(test_tpr_avg, test_fpr_
avg)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve")
plt.grid()
plt.show()
```



2.3.5 Plot confusion matrix

In [89]:

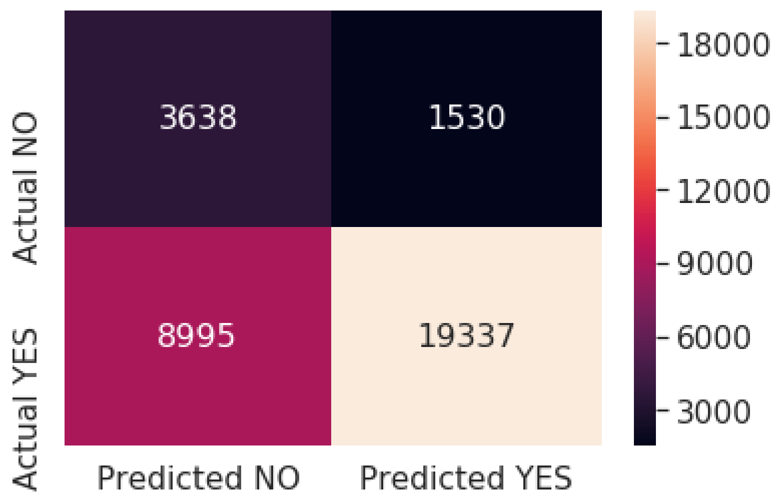
```
from sklearn.metrics import confusion_matrix
best_t_avg = find_best_threshold(tr_thresholds_avg, train_fpr_avg, train_tpr_avg)
print("Train confusion matrix")
cm_train_avg=confusion_matrix(y_train, predict_with_best_t(y_train_pred_avg_best, best_
t_avg))
print(cm_train_avg)
print("Test confusion matrix")
cm_test_avg=confusion_matrix(y_test, predict_with_best_t(y_test_pred_avg_best, best_t_a
vg))
print(cm_test_avg)
```

```
The maximum value of tpr*(1-fpr) 0.09399242626897614 for threshold 0.512
Train confusion matrix
[[ 3638  1530]
 [ 8995 19337]]
Test confusion matrix
[[1374 1172]
 [4399 9555]]
```

In [90]:

```
# confusion matrix heatmap for train data
print("Train confusion matrix heatmap")
cm_heatmap(cm_train_avg)
```

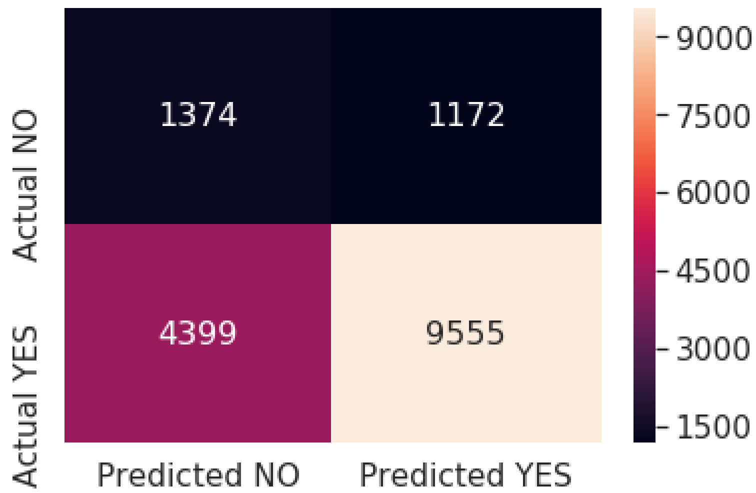
```
Train confusion matrix heatmap
```



In [91]:

```
# confusion matrix heatmap for test data
print("Test confusion matrix heatmap")
cm_heatmap(cm_test_avg)
```

```
Test confusion matrix heatmap
```



2.4 Set 4: TFIDFW2V featurization

2.4.1 Hyper parameter tuning

In [32]:

```
rf_tw = RandomForestClassifier(criterion='gini',class_weight = 'balanced')
parameters = {'n_estimators': [4, 8, 16, 32, 64, 100],'max_depth': [4, 6, 8, 10, 20, 25
]}
clf4 = RandomizedSearchCV(rf_tw, parameters, cv=10, scoring='roc_auc',return_train_scor
e=True,n_jobs=-1)
rs4 = clf4.fit(X_tr_tfidf_w2v, y_train)
```

In [33]:

```
df3=pd.DataFrame(clf4.cv_results_)
df3.head(2)
```

Out[33]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_max_depth
0	4.180500	0.044120	0.627240	0.709001	6
1	97.276872	0.188452	0.626161	0.999992	20

2 rows × 32 columns

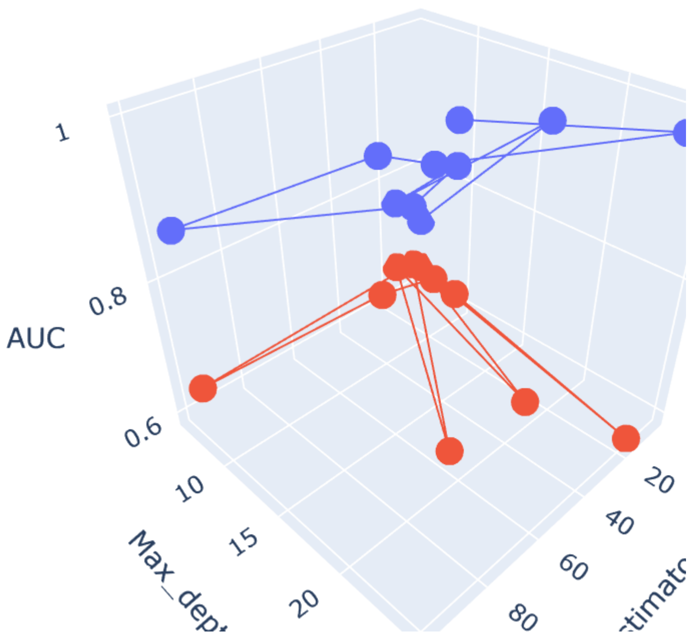
2.4.2 3D-Plot

In [34]:

```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=df3['param_n_estimators'],y=df3['param_max_depth'],z=df3['mean_
train_score'], name = 'train')
trace2 = go.Scatter3d(x=df3['param_n_estimators'],y=df3['param_max_depth'],z=df3['mean_
test_score'], name = 'Cross validation')
data = [trace1, trace2]
enable_plotly_in_cell()

layout = go.Layout(scene = dict(
        xaxis = dict(title='Estimators'),
        yaxis = dict(title='Max_depth'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```



2.4.3 Best Hyperparameters

In [35]:

```
print(clf4.best_estimator_)
print('Score on train data :', {clf4.score(X_tr_tfidf_w2v,y_train)})
print('Mean cross-validated score of the best_estimator :', {clf4.best_score_})
```

```
RandomForestClassifier(bootstrap=True, class_weight='balanced',
                       criterion='gini', max_depth=8, max_features='auto',
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       n_estimators=100, n_jobs=None, oob_score=False,
                       random_state=None, verbose=0, warm_start=False)
Score on train data : {0.8795813551852448}
Mean cross-validated score of the best_estimator : {0.6687675620797752}
```

In [97]:

```
best_parameters_tfidf = {'n_estimators': [8],'max_depth': [6]}
```

2.4.4 Applying Best Hyperparameters on train & test data & plotting ROC curve

In [36]:

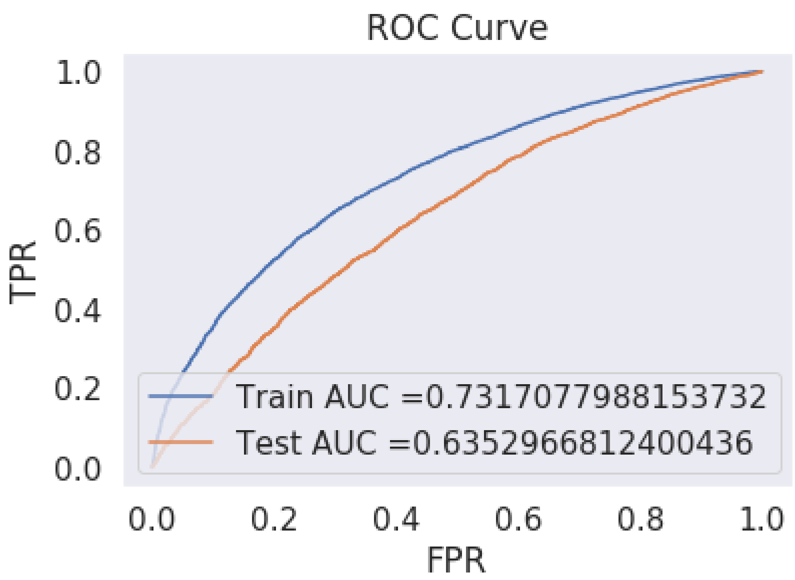
```
rf_best_tw= RandomForestClassifier(n_estimators= 8 , criterion='gini', max_depth= 6, cl
ass_weight = 'balanced')

rf_best_tw.fit(X_tr_tfidf_w2v, y_train)

y_train_pred_tw_best,pred_labels_train = batch_predict(rf_best_tw, X_tr_tfidf_w2v)
y_test_pred_tw_best,pred_labels_test = batch_predict(rf_best_tw, X_test_tfidf_w2v)

train_tpr_tw, train_fpr_tw, tr_thresholds_tw = roc_curve(y_train, y_train_pred_tw_best)
test_tpr_tw, test_fpr_tw, te_thresholds_tw = roc_curve(y_test, y_test_pred_tw_best)

plt.plot(train_tpr_tw, train_fpr_tw,label="Train AUC ="+str(auc(train_tpr_tw, train_fpr
_tw)))
plt.plot(test_tpr_tw, test_fpr_tw, label="Test AUC ="+str(auc(test_tpr_tw, test_fpr_tw
)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve")
plt.grid()
plt.show()
```



2.4.5 Plot confusion matrix

In [37]:

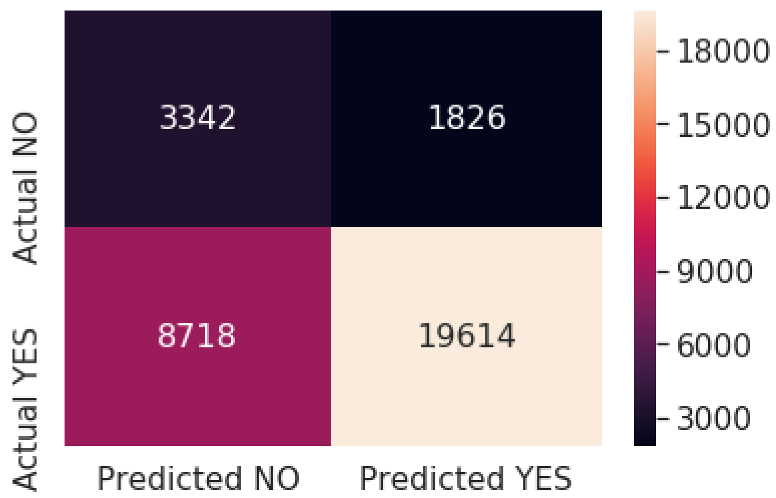
```
from sklearn.metrics import confusion_matrix
best_t_tw = find_best_threshold(tr_thresholds_tw, train_fpr_tw, train_tpr_tw)
print("Train confusion matrix")
cm_train_tw=confusion_matrix(y_train, predict_with_best_t(y_train_pred_tw_best, best_t_
tw))
print(cm_train_tw)
print("Test confusion matrix")
cm_test_tw=confusion_matrix(y_test, predict_with_best_t(y_test_pred_tw_best, best_t_tw
))
print(cm_test_tw)
```

```
The maximum value of tpr*(1-fpr) 0.10872211688126066 for threshold 0.493
Train confusion matrix
[[ 3342  1826]
 [ 8718 19614]]
Test confusion matrix
[[1311 1235]
 [4488 9466]]
```

In [38]:

```
# confusion matrix heatmap for train data
print("Train confusion matrix heatmap")
cm_heatmap(cm_train_tw)
```

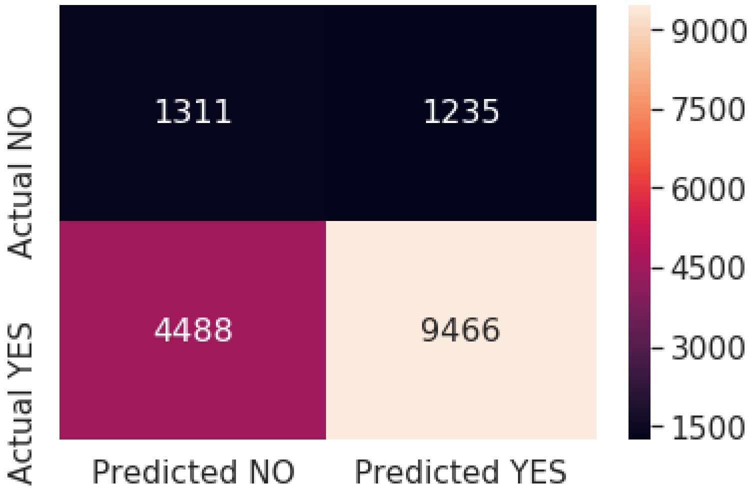
```
Train confusion matrix heatmap
```



In [39]:

```
# confusion matrix heatmap for test data
print("Test confusion matrix heatmap")
cm_heatmap(cm_test_tw)
```

```
Test confusion matrix heatmap
```



3. Applying XGBooost

3.1 Set 1: BOW featurization

3.1.1 Hyper parameter tuning

In [3]:

```
#https://dask-ml.readthedocs.io/en/stable/modules/generated/dask_ml.xgboost.XGBClassifi
er.html
#https://machinelearningmastery.com/develop-first-xgboost-model-python-scikit-learn/

from sklearn.metrics import roc_auc_score
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import cross_val_score
from xgboost import XGBClassifier

xg_bow = XGBClassifier()
parameters = {'n_estimators': [4, 8, 16, 32, 64, 100],'max_depth': [4, 6, 8, 10, 20, 25
]}
clf1 = RandomizedSearchCV(xg_bow, parameters, cv=10, scoring='roc_auc',return_train_sco
re=True,n_jobs=-1)
rs1 = clf1.fit(X_tr_bow, y_train)
```

In [12]:

```
df=pd.DataFrame(clf1.cv_results_)
df.head(2)
```

Out[12]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_max_depth	p
0	77.689885	0.384683	0.668469	0.999142	20	
1	37.674505	0.346404	0.671309	0.921477	10	

2 rows × 32 columns

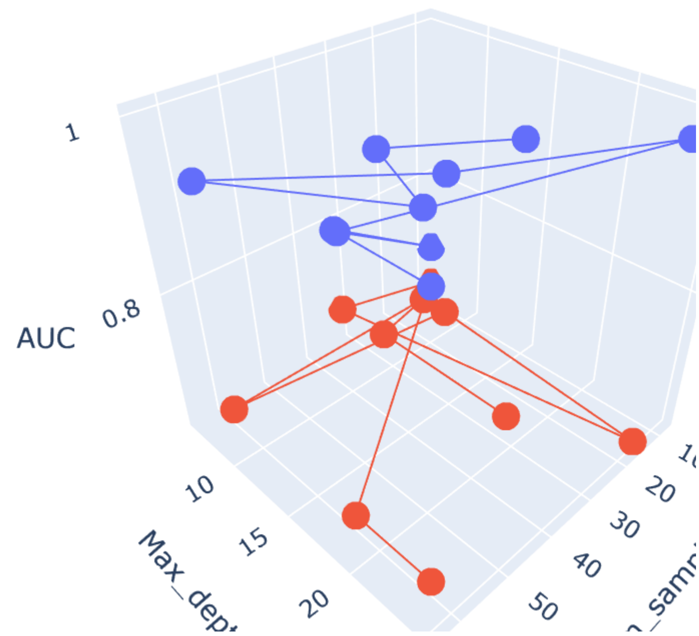
3.1.2 3D-Plot

In [13]:

```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=df['param_n_estimators'],y=df['param_max_depth'],z=df['mean_tra
in_score'], name = 'train')
trace2 = go.Scatter3d(x=df['param_n_estimators'],y=df['param_max_depth'],z=df['mean_tes
t_score'], name = 'Cross validation')
data = [trace1, trace2]
enable_plotly_in_cell()

layout = go.Layout(scene = dict(
        xaxis = dict(title='Estimators'),
        yaxis = dict(title='Max_depth'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```



3.1.3 Best Hyperparameters

In [14]:

```
print(clf1.best_estimator_)
print('CV score on train data:', {clf1.score(X_tr_bow,y_train)})
print('Mean cross-validated score of the best_estimator :', {clf1.best_score_})
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=10,
              min_child_weight=1, missing=None, n_estimators=64, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
CV score on train data: {0.9574053166151545}
Mean cross-validated score of the best_estimator : {0.6902191750430758}
```

- From the above result when I considered max_depth=10 & n_estimators=64, the model was overfitting the data
- Hence from the graph I chose 8 & 6 for estimators & depth respectively

In [15]:

```
best_parameters_bow = {'n_estimators': [8],'max_depth': [6]}
```

3.1.4 Applying Best Hyperparameters on train & test data & plotting ROC curve

In [31]:

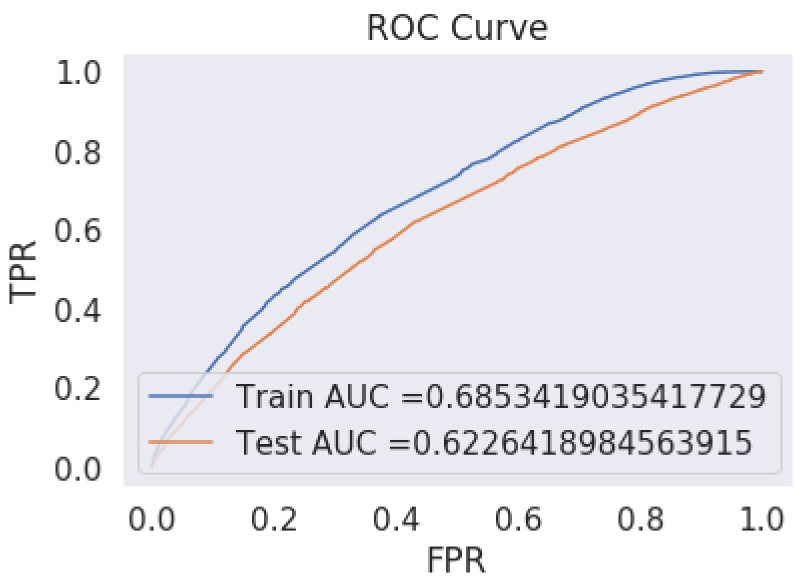
```
xg_best= XGBClassifier(n_estimators= 8 , max_depth= 6)

xg_best.fit(X_tr_bow, y_train)

y_train_pred_bow_best,pred_labels_train = batch_predict(xg_best, X_tr_bow)
y_test_pred_bow_best,pred_labels_test = batch_predict(xg_best, X_test_bow)

train_tpr_bow, train_fpr_bow, tr_thresholds_bow = roc_curve(y_train, y_train_pred_bow_b
est)
test_tpr_bow, test_fpr_bow, te_thresholds_bow = roc_curve(y_test, y_test_pred_bow_best)

plt.plot(train_tpr_bow, train_fpr_bow,label="Train AUC ="+str(auc(train_tpr_bow, train_
fpr_bow)))
plt.plot(test_tpr_bow, test_fpr_bow, label="Test AUC ="+str(auc(test_tpr_bow, test_fpr_
bow)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve")
plt.grid()
plt.show()
```



3.1.5 Plot confusion matrix

In [32]:

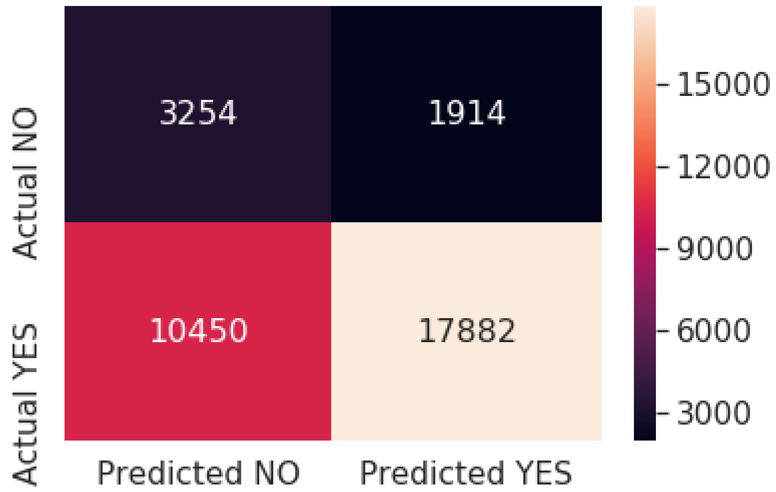
```
from sklearn.metrics import confusion_matrix
best_t_bow = find_best_threshold(tr_thresholds_bow, train_fpr_bow, train_tpr_bow)
print("Train confusion matrix")
cm_train_bow=confusion_matrix(y_train, predict_with_best_t(y_train_pred_bow_best, best_
t_bow))
print(cm_train_bow)
print("Test confusion matrix")
cm_test_bow=confusion_matrix(y_test, predict_with_best_t(y_test_pred_bow_best, best_t_b
ow))
print(cm_test_bow)
```

```
The maximum value of tpr*(1-fpr) 0.13660244911179212 for threshold 0.692
Train confusion matrix
[[ 3254  1914]
 [10450 17882]]
Test confusion matrix
[[1453 1093]
 [5325 8629]]
```

In [33]:

```
# confusion matrix heatmap for train data
print("Train confusion matrix heatmap")
cm_heatmap(cm_train_bow)
```

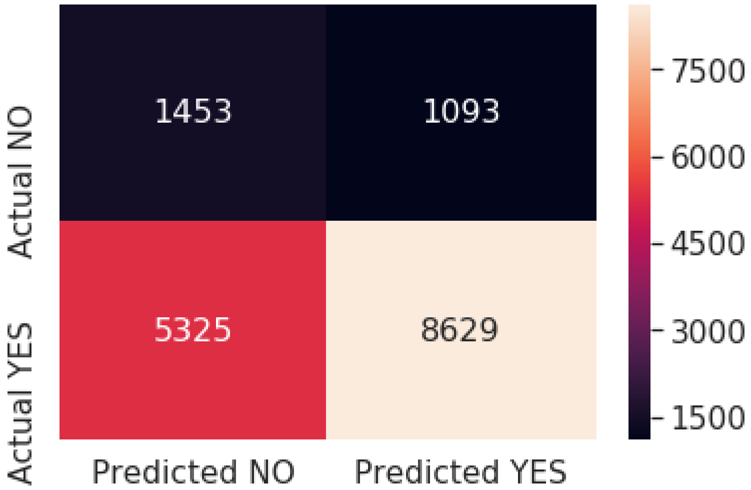
```
Train confusion matrix heatmap
```



In [34]:

```
# confusion matrix heatmap for test data
print("Test confusion matrix heatmap")
cm_heatmap(cm_test_bow)
```

```
Test confusion matrix heatmap
```



3.2 Set 2: TFIDF featurization

3.2.1 Hyper parameter tuning

In [20]:

```
%%time
xg_tfidf = XGBClassifier()
parameters = {'n_estimators': [4, 8, 16, 32, 64, 100],'max_depth': [4, 6, 8, 10, 20, 25
]}
clf2 = RandomizedSearchCV(xg_tfidf, parameters, cv=10, scoring='roc_auc',return_train_s
core=True,n_jobs=-1)
rs2 = clf2.fit(X_tr_tfidf, y_train)
```

```
CPU times: user 7min 35s, sys: 472 ms, total: 7min 35s
Wall time: 51min 3s
```

In [21]:

```
df1=pd.DataFrame(clf2.cv_results_)
df1.head(2)
```

Out[21]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_max_depth
0	598.782416	0.476588	0.696119	0.999997	20
1	63.765750	0.347809	0.616969	0.961846	25

2 rows × 32 columns

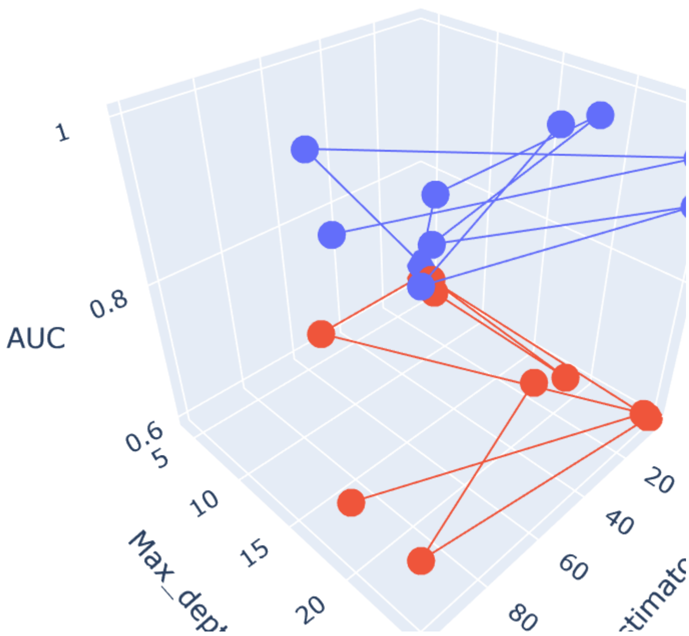
3.2.2 3D-Plot

In [26]:

```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=df1['param_n_estimators'],y=df1['param_max_depth'],z=df1['mean_
train_score'], name = 'train')
trace2 = go.Scatter3d(x=df1['param_n_estimators'],y=df1['param_max_depth'],z=df1['mean_
test_score'], name = 'Cross validation')
data = [trace1, trace2]
enable_plotly_in_cell()

layout = go.Layout(scene = dict(
        xaxis = dict(title='Estimators'),
        yaxis = dict(title='Max_depth'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```



3.2.3 Best Hyperparameters

In [23]:

```
print(clf2.best_estimator_)
print('CV score on train data:', {clf2.score(X_tr_tfidf,y_train)})
print('Mean cross-validated score of the best_estimator :', {clf2.best_score_})
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=25,
              min_child_weight=1, missing=None, n_estimators=100, n_jobs=
1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
CV score on train data: {0.9999998873103043}
Mean cross-validated score of the best_estimator : {0.6975534821057656}
```

- From the above result when I considered max_depth=25 & n_estimators=100, the model was overfitting the data
- Hence from the graph I chose 8 & 6 for estimators & depth respectively

In [25]:

```
best_parameters_tfidf = {'n_estimators': [8],'max_depth': [6]}
```

3.2.4 Applying Best Hyperparameters on train & test data & plotting ROC curve

In [27]:

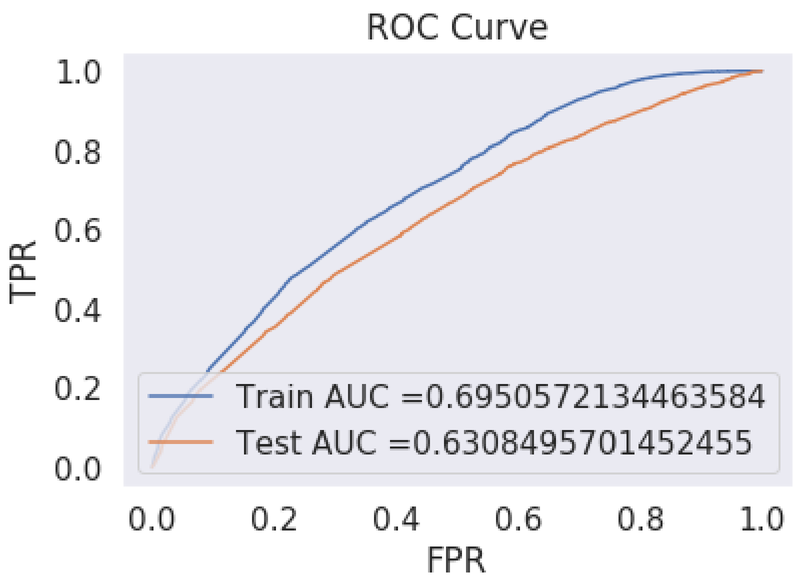
```
xg_best_tfidf= XGBClassifier(n_estimators= 8 , max_depth=6)

xg_best_tfidf.fit(X_tr_tfidf, y_train)

y_train_pred_tfidf_best,pred_labels_train = batch_predict(xg_best_tfidf, X_tr_tfidf)
y_test_pred_tfidf_best,pred_labels_test = batch_predict(xg_best_tfidf, X_test_tfidf)

train_tpr_tfidf, train_fpr_tfidf, tr_thresholds_tfidf = roc_curve(y_train, y_train_pred
_tfidf_best)
test_tpr_tfidf, test_fpr_tfidf, te_thresholds_tfidf = roc_curve(y_test, y_test_pred_tfi
df_best)

plt.plot(train_tpr_tfidf, train_fpr_tfidf,label="Train AUC ="+str(auc(train_tpr_tfidf,
train_fpr_tfidf)))
plt.plot(test_tpr_tfidf, test_fpr_tfidf, label="Test AUC ="+str(auc(test_tpr_tfidf, tes
t_fpr_tfidf)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve")
plt.grid()
plt.show()
```



3.2.5 Plot confusion matrix

In [28]:

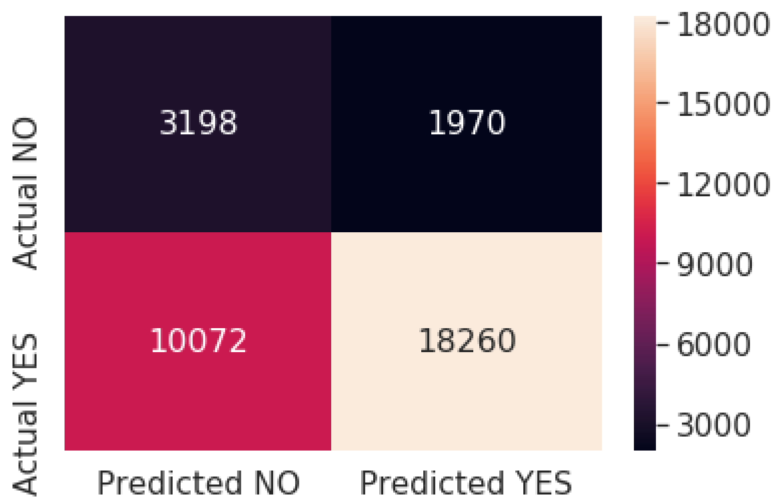
```
best_t_tfidf = find_best_threshold(tr_thresholds_tfidf, train_fpr_tfidf, train_tpr_tfid
f)
print("Train confusion matrix")
cm_train_tfidf=confusion_matrix(y_train, predict_with_best_t(y_train_pred_tfidf_best, b
est_t_tfidf))
print(cm_train_tfidf)
print("Test confusion matrix")
cm_test_tfidf=confusion_matrix(y_test, predict_with_best_t(y_test_pred_tfidf_best, best
_t_tfidf))
print(cm_test_tfidf)
```

```
The maximum value of tpr*(1-fpr) 0.13551338857395878 for threshold 0.699
Train confusion matrix
[[ 3198  1970]
 [10072 18260]]
Test confusion matrix
[[1399 1147]
 [5126 8828]]
```

In [29]:

```
# confusion matrix heatmap for train data
print("Train confusion matrix heatmap")
cm_heatmap(cm_train_tfidf)
```

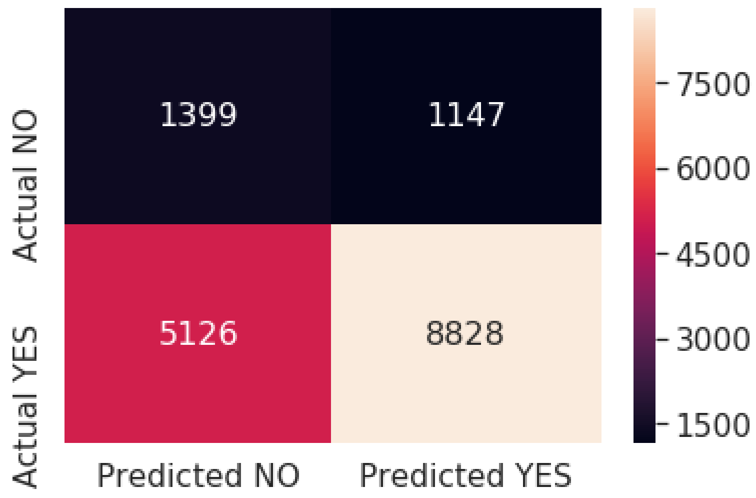
```
Train confusion matrix heatmap
```



In [30]:

```
# confusion matrix heatmap for test data
print("Test confusion matrix heatmap")
cm_heatmap(cm_test_tfidf)
```

```
Test confusion matrix heatmap
```



3.3 Set 3: AvgW2V featurization

3.3.1 Hyper parameter tuning

In [35]:

```
xg_avg = XGBClassifier()
parameters = {'n_estimators': [4, 8, 16, 32, 64, 100],'max_depth': [4, 6, 8, 10, 20, 25
]}
clf3 = RandomizedSearchCV(xg_avg, parameters, cv=10, scoring='roc_auc',return_train_sco
re=True,n_jobs=-1)
rs3 = clf3.fit(X_tr_avgw2v, y_train)
```

In [36]:

```
df2=pd.DataFrame(clf3.cv_results_)
df2.head(2)
```

Out[36]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_max_depth
0	202.450500	0.911339	0.684710	0.831717	4
1	54.451288	0.894091	0.655966	0.810541	6

2 rows × 32 columns

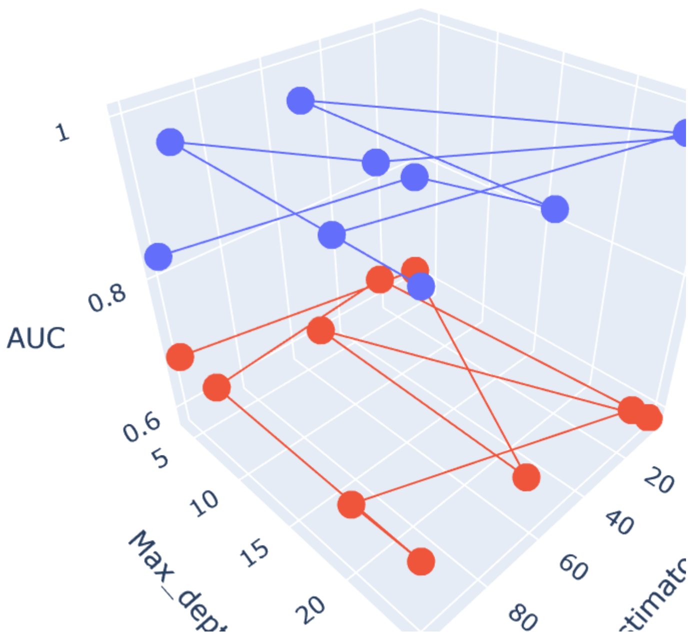
3.3.2 3D-Plot

In [37]:

```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=df2['param_n_estimators'],y=df2['param_max_depth'],z=df2['mean_train_score'], name = 'train')
trace2 = go.Scatter3d(x=df2['param_n_estimators'],y=df2['param_max_depth'],z=df2['mean_test_score'], name = 'Cross validation')
data = [trace1, trace2]
enable_plotly_in_cell()

layout = go.Layout(scene = dict(
        xaxis = dict(title='Estimators'),
        yaxis = dict(title='Max_depth'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```



3.3.3 Best Hyperparameters

In [38]:

```
print(clf3.best_estimator_)
print('Score on train data :', {clf3.score(X_tr_avgw2v,y_train)})
print('Mean cross-validated score of the best_estimator :', {clf3.best_score_})
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=8,
              min_child_weight=1, missing=None, n_estimators=100, n_jobs=
1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
Score on train data : {0.9987744688258504}
Mean cross-validated score of the best_estimator : {0.6856626294924051}
```

In [39]:

```
best_parameters_tfidf = {'n_estimators': [16],'max_depth': [6]}
```

3.3.4 Applying Best Hyperparameters on train & test data & plotting ROC curve

In [40]:

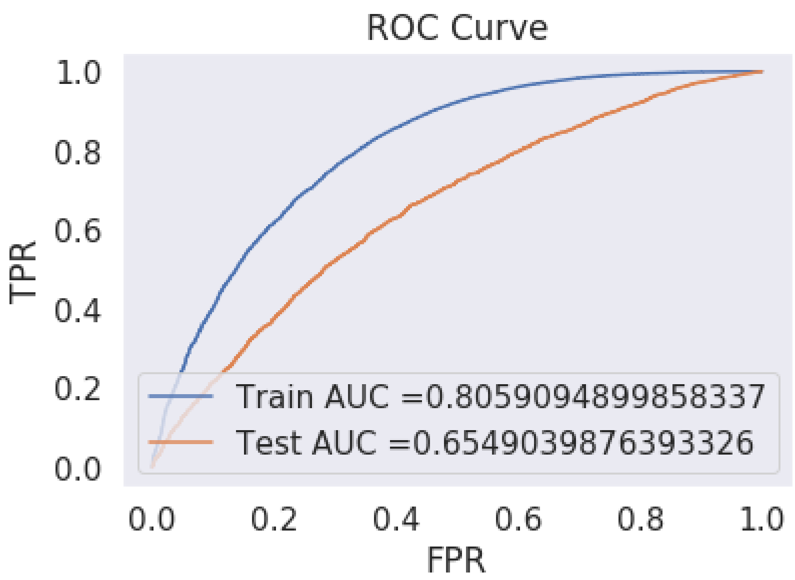
```
xg_best_avg= XGBClassifier(n_estimators= 16 , max_depth= 6)

xg_best_avg.fit(X_tr_avgw2v, y_train)

y_train_pred_avg_best,pred_labels_train = batch_predict(xg_best_avg, X_tr_avgw2v)
y_test_pred_avg_best,pred_labels_test = batch_predict(xg_best_avg, X_test_avgw2v)

train_tpr_avg, train_fpr_avg, tr_thresholds_avg = roc_curve(y_train, y_train_pred_avg_b
est)
test_tpr_avg, test_fpr_avg, te_thresholds_avg = roc_curve(y_test, y_test_pred_avg_best)

plt.plot(train_tpr_avg, train_fpr_avg,label="Train AUC ="+str(auc(train_tpr_avg, train_
fpr_avg)))
plt.plot(test_tpr_avg, test_fpr_avg, label="Test AUC ="+str(auc(test_tpr_avg, test_fpr_
avg)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve")
plt.grid()
plt.show()
```



3.3.5 Plot confusion matrix

In [41]:

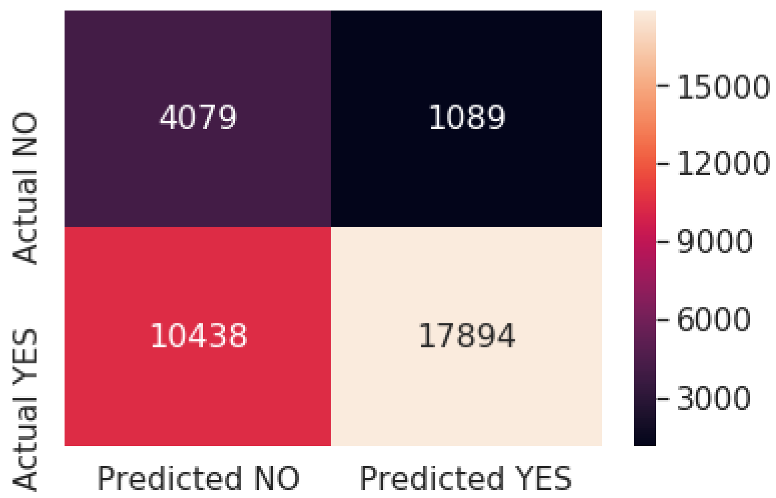
```
from sklearn.metrics import confusion_matrix
best_t_avg = find_best_threshold(tr_thresholds_avg, train_fpr_avg, train_tpr_avg)
print("Train confusion matrix")
cm_train_avg=confusion_matrix(y_train, predict_with_best_t(y_train_pred_avg_best, best_
t_avg))
print(cm_train_avg)
print("Test confusion matrix")
cm_test_avg=confusion_matrix(y_test, predict_with_best_t(y_test_pred_avg_best, best_t_a
vg))
print(cm_test_avg)
```

```
The maximum value of tpr*(1-fpr) 0.07763283287634588 for threshold 0.787
Train confusion matrix
[[ 4079  1089]
 [10438 17894]]
Test confusion matrix
[[1758  788]
 [6513 7441]]
```

In [42]:

```
# confusion matrix heatmap for train data
print("Train confusion matrix heatmap")
cm_heatmap(cm_train_avg)
```

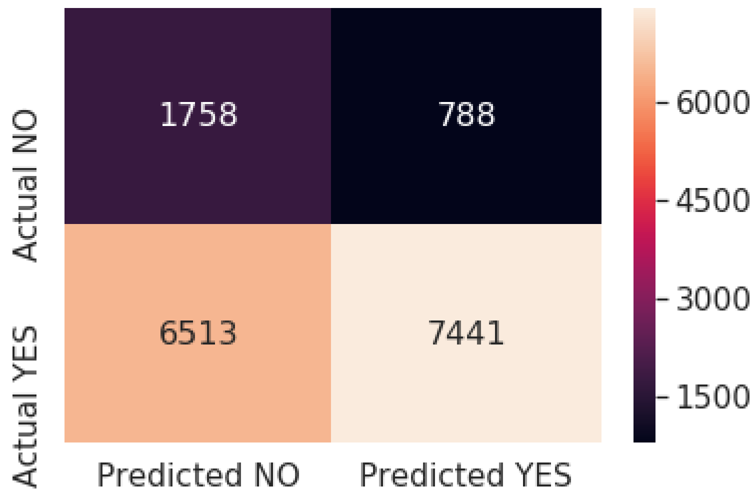
```
Train confusion matrix heatmap
```



In [43]:

```
# confusion matrix heatmap for test data
print("Test confusion matrix heatmap")
cm_heatmap(cm_test_avg)
```

```
Test confusion matrix heatmap
```



3.4 Set 4: TFIDFW2V featurization

3.4.1 Hyper parameter tuning

In [11]:

```
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import cross_val_score
from xgboost import XGBClassifier

xg_tw = XGBClassifier()
parameters = {'n_estimators': [4, 8, 16, 32, 64, 100],'max_depth': [4, 6, 8, 10, 20, 25
]}
clf4 = RandomizedSearchCV(xg_tw, parameters, cv=10, scoring='roc_auc',return_train_scor
e=True,n_jobs=-1)
rs4 = clf4.fit(X_tr_tfidf_w2v, y_train)
```

In [12]:

```
df3=pd.DataFrame(clf4.cv_results_)
df3.head(2)
```

Out[12]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_max_depth	p
0	650.620977	1.549350	0.647025	1.000000	20	
1	51.779221	1.462077	0.629482	0.808335	8	

2 rows × 32 columns

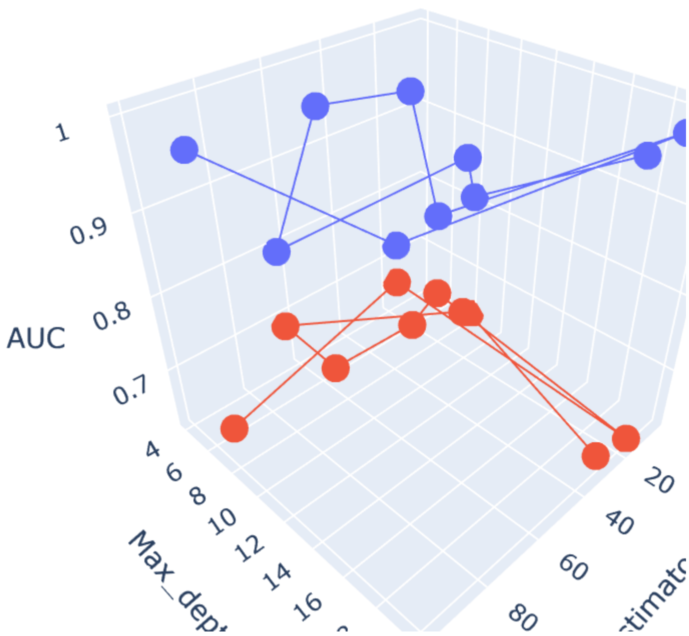
3.4.2 3D-Plot

In [13]:

```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=df3['param_n_estimators'],y=df3['param_max_depth'],z=df3['mean_
train_score'], name = 'train')
trace2 = go.Scatter3d(x=df3['param_n_estimators'],y=df3['param_max_depth'],z=df3['mean_
test_score'], name = 'Cross validation')
data = [trace1, trace2]
enable_plotly_in_cell()

layout = go.Layout(scene = dict(
        xaxis = dict(title='Estimators'),
        yaxis = dict(title='Max_depth'),
        zaxis = dict(title='AUC'),))

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```



3.4.3 Best Hyperparameters

In [14]:

```
print(clf4.best_estimator_)
print('Score on train data :', {clf4.score(X_tr_tfidf_w2v,y_train)})
print('Mean cross-validated score of the best_estimator :', {clf4.best_score_})
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=4,
              min_child_weight=1, missing=None, n_estimators=64, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
Score on train data : {0.782515153554121}
Mean cross-validated score of the best_estimator : {0.6853332894692152}
```

In [15]:

```
best_parameters_tfidf = {'n_estimators': [64],'max_depth': [4]}
```

3.4.4 Applying Best Hyperparameters on train & test data & plotting ROC curve

In [16]:

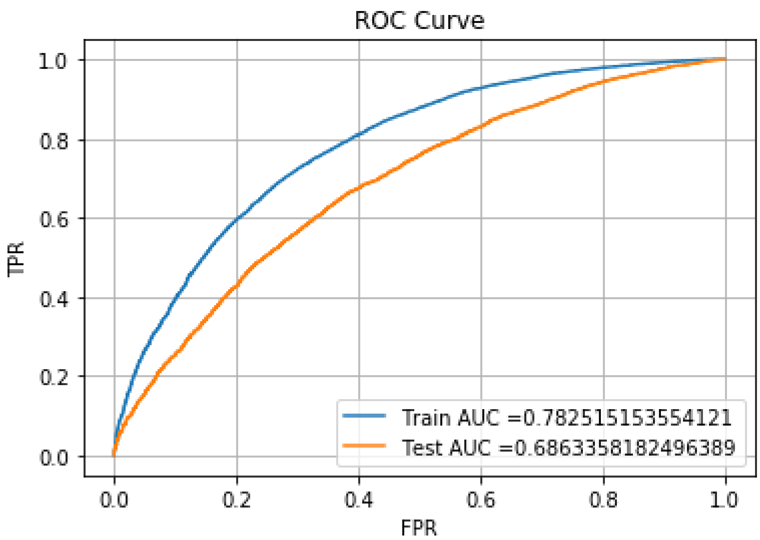
```
xg_best_tw= XGBClassifier(n_estimators= 64 , max_depth= 4)

xg_best_tw.fit(X_tr_tfidf_w2v, y_train)

y_train_pred_tw_best,pred_labels_train = batch_predict(xg_best_tw, X_tr_tfidf_w2v)
y_test_pred_tw_best,pred_labels_test = batch_predict(xg_best_tw, X_test_tfidf_w2v)

train_tpr_tw, train_fpr_tw, tr_thresholds_tw = roc_curve(y_train, y_train_pred_tw_best)
test_tpr_tw, test_fpr_tw, te_thresholds_tw = roc_curve(y_test, y_test_pred_tw_best)

plt.plot(train_tpr_tw, train_fpr_tw,label="Train AUC ="+str(auc(train_tpr_tw, train_fpr
_tw)))
plt.plot(test_tpr_tw, test_fpr_tw, label="Test AUC ="+str(auc(test_tpr_tw, test_fpr_tw
)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve")
plt.grid()
plt.show()
```



3.4.5 Plot confusion matrix

In [17]:

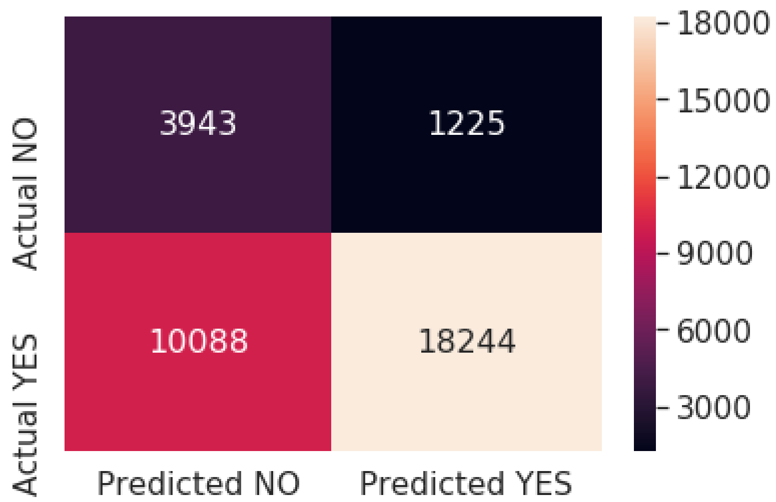
```
from sklearn.metrics import confusion_matrix
best_t_tw = find_best_threshold(tr_thresholds_tw, train_fpr_tw, train_tpr_tw)
print("Train confusion matrix")
cm_train_tw=confusion_matrix(y_train, predict_with_best_t(y_train_pred_tw_best, best_t_
tw))
print(cm_train_tw)
print("Test confusion matrix")
cm_test_tw=confusion_matrix(y_test, predict_with_best_t(y_test_pred_tw_best, best_t_tw
))
print(cm_test_tw)
```

```
The maximum value of tpr*(1-fpr) 0.08439980129460106 for threshold 0.849
Train confusion matrix
[[ 3943  1225]
 [10088 18244]]
Test confusion matrix
[[1673  873]
 [5390 8564]]
```

In [18]:

```
# confusion matrix heatmap for train data
print("Train confusion matrix heatmap")
cm_heatmap(cm_train_tw)
```

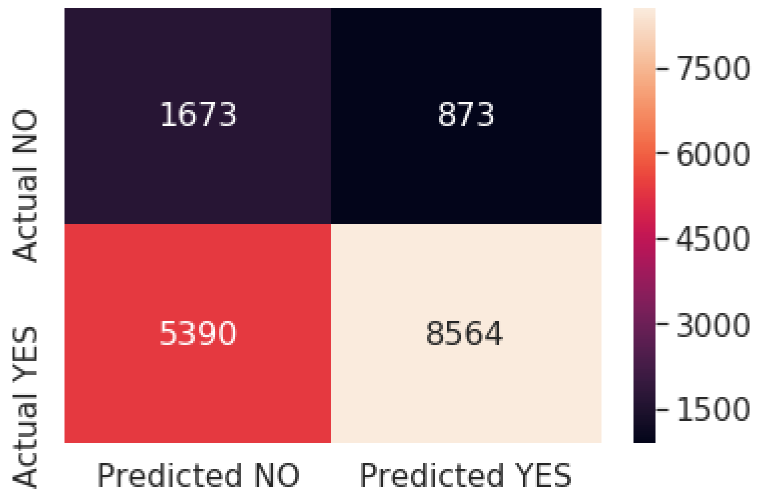
```
Train confusion matrix heatmap
```



In [19]:

```
# confusion matrix heatmap for test data
print("Test confusion matrix heatmap")
cm_heatmap(cm_test_tw)
```

```
Test confusion matrix heatmap
```



4.0 Summary

In [20]:

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

x = PrettyTable()
print('RF Models summary')
x.field_names = ["Vectorizer","max_depth","n_estimators" ,"Test AUC"]
x.add_row(["BOW", 6, 100, 0.68])
x.add_row(["TFIDF", 6, 100, 0.67])
x.add_row(["Avg W2V", 6, 32, 0.65])
x.add_row(["TFIDF W2V", 6, 8, 0.64])

print(x)
```

```
RF Models summary
+------------+-----------+--------------+----------+
| Vectorizer | max_depth | n_estimators | Test AUC |
+------------+-----------+--------------+----------+
|    BOW     |     6     |     100      |   0.68   |
|   TFIDF    |     6     |     100      |   0.67   |
|  Avg W2V   |     6     |      32      |   0.65   |
| TFIDF W2V  |     6     |      8       |   0.64   |
+------------+-----------+--------------+----------+
```

In [21]:

```
x = PrettyTable()
print('XGBoost Models summary')
x.field_names = ["Vectorizer","max_depth","n_estimators" ,"Test AUC"]
x.add_row(["BOW", 6, 8, 0.62])
x.add_row(["TFIDF", 6, 8, 0.63])
x.add_row(["Avg W2V", 6, 16, 0.66])
x.add_row(["TFIDF W2V", 4, 64, 0.69])

print(x)
```

```
XGBoost Models summary
+------------+-----------+--------------+----------+
| Vectorizer | max_depth | n_estimators | Test AUC |
+------------+-----------+--------------+----------+
|    BOW     |     6     |      8       |   0.62   |
|   TFIDF    |     6     |      8       |   0.63   |
|  Avg W2V   |     6     |      16      |   0.66   |
| TFIDF W2V  |     4     |      64      |   0.69   |
+------------+-----------+--------------+----------+
```