# DBDT with Response coding featurization

In [115]:

```
import pandas as pd
import pickle
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
import math
import pandas as pd
import numpy as np
import nltk
import matplotlib.pyplot as plt
import seaborn as sns
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
import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import pickle
from tqdm import tqdm
import os
from collections import Counter
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
import numpy as np
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from sklearn.preprocessing import MinMaxScaler
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix
from tqdm import tqdm
from wordcloud import WordCloud
from sklearn.metrics import roc_curve, auc
from  sklearn.linear_model import LogisticRegression
```

# Task-1

# Featurising data

In [116]:

```
#please use below code to load glove vectors
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

### Loading Data

In [117]:

```
#make sure you are loading atleast 50k datapoints
#you can work with features of preprocessed_data.csv for the assignment.
import pandas
data = pandas.read_csv('preprocessed_data.csv',nrows=50000)
```

In [118]:

```
df_data=data.copy()      #copying original dataframe
```

In [119]:

```
df_data.head(1)     #copy of original dataframe
```

Out[119]:

| | school_state | teacher_prefix | project_grade_category | teacher_number_of_previously_posted_projects | project_is_approved | clean_categories | clean_subcat |
|---|---|---|---|---|---|---|---|
| 0 | ca | mrs | grades_prek_2 | 53 | 1 | math_science | applieds health_life |

In [120]:

```
# find the sentiment score for eassy using sentiment intensity analyzer

df_polarity=pd.DataFrame(columns=["negative","neutral","positive","compound"])    #how to enter values rowwise in dataframe: #https://stac
sid = SentimentIntensityAnalyzer()  #initialising sentiment intensity analyzer

for idx,row in tqdm(enumerate(data["essay"])):# for  essay in each project
    ss_1 = sid.polarity_scores(row)     #finding polarity score
    polarity_features=list(ss_1.values())
    df_polarity.loc[idx] = (polarity_features[0] , polarity_features[1] , polarity_features[2],polarity_features[3]) #making new dataframe
```

```
50000it [02:40, 311.00it/s]
```

In [121]:

```
df_polarity.head(5)     #dataframe with polarity score
```

Out[121]:

| | negative | neutral | positive | compound |
|---|---|---|---|---|
| 0 | 0.013 | 0.783 | 0.205 | 0.9867 |
| 1 | 0.072 | 0.680 | 0.248 | 0.9897 |
| 2 | 0.017 | 0.721 | 0.262 | 0.9860 |
| 3 | 0.030 | 0.783 | 0.187 | 0.9524 |
| 4 | 0.029 | 0.683 | 0.288 | 0.9873 |

In [122]:

```
#merging polarity df with data_df
data = pd.merge(df_data,df_polarity,left_index=True, right_index=True, how='left')
```

In [123]:

```
data.head(2)      #after merging
```

Out[123]:

| | school_state | teacher_prefix | project_grade_category | teacher_number_of_previously_posted_projects | project_is_approved | clean_categories | clean_subcat |
|---|---|---|---|---|---|---|---|
| 0 | ca | mrs | grades_prek_2 | 53 | 1 | math_science | applieds health_life |
| 1 | ut | ms | grades_3_5 | 4 | 1 | specialneeds | speci |

In [124]:

```
len(data.columns)    #after two dataframe get merged we get 13 columns
```

Out[124]:

```
13
```

### Splitting data

In [125]:

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

Out[125]:

| | school_state | teacher_prefix | project_grade_category | teacher_number_of_previously_posted_projects | clean_categories | clean_subcategories | essay | p |
|---|---|---|---|---|---|---|---|---|
| 0 | ca | mrs | grades_prek_2 | 53 | math_science | appliedsciences health_lifescience | i fortunate enough use fairy tale stem kits cl... | 72 |

In [126]:

```
# train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y)
```

In [127]:

```
'''with open('X_train.pickle', 'wb') as f:
    pickle.dump(X_train, f)

with open('X_test.pickle', 'wb') as f:
    pickle.dump(X_test, f)

with open('y_train.pickle', 'wb') as f:
    pickle.dump(y_train, f)

with open('y_test.pickle', 'wb') as f:
    pickle.dump(y_test, f)'''
```

Out[127]:

```
"with open('X_train.pickle', 'wb') as f:\n    pickle.dump(X_train, f)\n    \nwith open('X_test.pickle', 'wb') as f:\n    pi
ckle.dump(X_test, f)\n\nwith open('y_train.pickle', 'wb') as f:\n    pickle.dump(y_train, f)\n\nwith open('y_test.pickle',
'wb') as f:\n    pickle.dump(y_test, f)"
```

### Vectorizing categorical featues

In [128]:

```
#Function to do Response coding
#to create this function reference taken from 2 places which is mentioned below

def response_coding(X_train, y_train,X_test,y_test,feature):

    X_train = X_train.reset_index().drop("index",axis=1) #reset the index in the dataframe
    X_test = X_test.reset_index().drop("index",axis=1)

    category_dict = dict()         #dictionary to store the categories and their number of times appeared count

    unique_cat_labels_train = X_train[feature].unique()      #store unique categories for a given feature in train data
    unique_cat_labels_test = X_test[feature].unique()        #store unique categories for a given feature in test data

    unique_on_test= set(unique_cat_labels_test)-set(unique_cat_labels_train)
    unique_on_test=list(unique_on_test)           #set of categories which is find only on test data(these categories needs laplace smoot


    #1)how to count number of rows with conditions- reference taken from : https://stackoverflow.com/a/66122578/17345549

    for i in tqdm(range(len(unique_cat_labels_train))):      #for each unique categories
        total_count = X_train.loc[:,feature][(X_train[feature] == unique_cat_labels_train[i])].count()            #count total number of 
        p_0 = X_train.loc[:, feature][((X_train[feature] == unique_cat_labels_train[i]) & (y_train==0))].count()  ##count total number of
        p_1 = X_train.loc[:, feature][((X_train[feature] == unique_cat_labels_train[i]) & (y_train==1))].count()  ##count total number of

        category_dict[unique_cat_labels_train[i]] = [p_1/total_count, p_0/total_count]     #calculating and storing the probability of oc

    if unique_on_test:              #if any category not present in train but present in test, put (0.5,0.5) as probability value
        for i in range(len(unique_on_test)):
            category_dict[unique_on_test[i]] =[0.5 , 0.5]

    value_count = dict(X_train[feature].value_counts())


    # 2)for creating response coded df and merge with train dataframe-reference taken from : https://gist.github.com/sukanta-27/1fb75d974

    res_fea_train=[]                       #response coding of features in train data
    for index, row in X_train.iterrows():  #in each row of train data
        res_fea_train.append(category_dict[row[feature]])  #getting values form dictionary for each row's given feature's category of the

    df_response_train = pd.DataFrame(np.array(res_fea_train), columns=[feature+"_0", feature+"_1"]) #make a dataframe with response coded
    X_train = pd.concat([X_train, df_response_train], axis=1)                                     #merge the response coded df with X_tr



    res_fea_test=[]                        #response coding of features in test data
    for index, row in X_test.iterrows():    #in each row of test data
        res_fea_test.append(category_dict[row[feature]])    #getting values form dictionary for each row's given feature's category of the

    df_response_test= pd.DataFrame(np.array(res_fea_test), columns=[feature+"_0", feature+"_1"])  #make a dataframe with response coded v
    X_test = pd.concat([X_test, df_response_test], axis=1)                                       #merge the response coded df with X_test

    X_train = X_train.drop(feature, axis=1)      #drop the categorical features column on both train and test df
    X_test = X_test.drop(feature, axis=1)
    return X_train,X_test                   #return updated train and test df
```

In [ ]:

In [129]:

```
#featurising school_state categorical feature

print("Before vectorizations")
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
print("="*100)

X_train,X_test=response_coding(X_train,y_train,X_test,y_test,'school_state')

print("After vectorizations")  #one original feature column dropped (so,12-1=11) and two response coded column added(so,11+2=13).
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
print("="*100)
```

```
Before vectorizations
(33500, 12) (33500,)
(16500, 12) (16500,)
====================================================================================================

100%|██████████████████████████████████████████████████████████████████████████████| 51/51 [00:00<00:00, 194.32it/s]

After vectorizations
(33500, 13) (33500,)
(16500, 13) (16500,)
====================================================================================================
```

In [130]:

```
#featurising teacher_prefix categorical feature

print("Before vectorizations")
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
print("="*100)

X_train,X_test=response_coding(X_train,y_train,X_test,y_test,'teacher_prefix')

print("After vectorizations")
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
print("="*100)
```

```
Before vectorizations
(33500, 13) (33500,)
(16500, 13) (16500,)
====================================================================================================

100%|████████████████████████████████████████████████████████████████████████████████| 5/5 [00:00<00:00, 151.51it/s]

After vectorizations
(33500, 14) (33500,)
(16500, 14) (16500,)
====================================================================================================
```

In [131]:

```
#featurising project_grade_category categorical features

print("Before vectorizations")
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
print("="*100)

X_train,X_test=response_coding(X_train,y_train,X_test,y_test,'project_grade_category')

print("After vectorizations")
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
print("="*100)
```

```
Before vectorizations
(33500, 14) (33500,)
(16500, 14) (16500,)
====================================================================================================

100%|████████████████████████████████████████████████████████████████████████████████| 4/4 [00:00<00:00, 153.82it/s]

After vectorizations
(33500, 15) (33500,)
(16500, 15) (16500,)
====================================================================================================
```

In [132]:

```
#featurising clean_categories categorical features

print("Before vectorizations")
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
print("="*100)

X_train,X_test=response_coding(X_train,y_train,X_test,y_test,'clean_categories')

print("After vectorizations")
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
print("="*100)
```

```
Before vectorizations
(33500, 15) (33500,)
(16500, 15) (16500,)
====================================================================================================

100%|██████████████████████████████████████████████████████████████████████████████████| 44/44 [00:00<00:00, 192.83it/s]

After vectorizations
(33500, 16) (33500,)
(16500, 16) (16500,)
====================================================================================================
```

In [133]:

```
#featurising clean_subcategories categorical features

print("Before vectorizations")
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
print("="*100)

X_train,X_test=response_coding(X_train,y_train,X_test,y_test,'clean_subcategories')

print("After vectorizations")
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
print("="*100)
```

```
Before vectorizations
(33500, 16) (33500,)
(16500, 16) (16500,)
====================================================================================================

100%|████████████████████████████████████████████████████████████████████████████████| 335/335 [00:01<00:00, 206.50it/s]

After vectorizations
(33500, 17) (33500,)
(16500, 17) (16500,)
====================================================================================================
```

### Vectorizing Numberical featues

In [134]:

```
#featurising price - numerical features


scaler = MinMaxScaler()       #doing minmaxscaling to numerical feature(price)
#reshape(-1, 1)--so that minmaxscaling applied on price column(feature column)
X_train_price_norm=scaler.fit_transform(X_train['price'].values.reshape(-1, 1)) #fitting train data
X_test_price_norm = scaler.transform(X_test['price'].values.reshape(-1, 1))     #converting test data using fitted minmaxscaler


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,)
====================================================================================================
```

In [138]:

```
X_train
```

Out[138]:

| | teacher_number_of_previously_posted_projects | essay | price | negative | neutral | positive | compound | school_state_0 | school_state_1 | teacher_prefix_0 | tea |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | hi i title i teacher works esl english second ... | 1372.34 | 0.047 | 0.705 | 0.248 | 0.9896 | 0.792904 | 0.207096 | 0.847212 | |
| **1** | 29 | i teach rural extremely diverse middle school ... | 1019.64 | 0.069 | 0.695 | 0.236 | 0.9838 | 0.850589 | 0.149411 | 0.847212 | |
| **2** | 0 | my first grade students attend title | 257.76 | 0.065 | 0.731 | 0.205 | 0.9771 | 0.853981 | 0.146019 | 0.847212 | |

In [146]:

```
#featurising teacher_number_of_previously_posted_projects - numerical features

scaler = MinMaxScaler()
#reshape(-1, 1)--so that minmaxscaling applied on teacher_number_of_previously_posted_projects column(feature column)
X_train_previous_projects_norm = scaler.fit_transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1))
X_test_previous_projects_norm = scaler.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1, 1))


print("After vectorizations")
print(X_train_previous_projects_norm.shape, y_train.shape)
print(X_test_previous_projects_norm.shape, y_test.shape)
print("="*100)
```

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

In [147]:

```
boolean=X_train.isnull().values.any()   #check if any Null value present in the dataframe
print(boolean)
```

```
False
```

# Tfidf-vectorization

In [148]:

```
#tfidf vectorization

vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=8000)                 #TFIDF vectorizer

X_train_essay_tfidf = vectorizer.fit_transform(X_train['essay'].values) #fitted only the train data
X_test_essay_tfidf = vectorizer.transform(X_test['essay'].values) # we use the fitted TfidfVectorizer to convert the test text to vector


print("After vectorizations")
print(X_train_essay_tfidf.shape, y_train.shape)  #size of train and test vector
print(X_test_essay_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(33500, 8000) (33500,)
(16500, 8000) (16500,)
====================================================================================================
```

### Combining all the featuers of set1

In [168]:

```
from scipy.sparse import hstack
#all the necessary feature for set1 is stacked together horizontally
#stacked train features
X_tr_tfidf = hstack((X_train['school_state_0'].to_numpy().reshape(-1,1),X_train['school_state_1'].to_numpy().reshape(-1,1), X_train['teac
#stacked test features
X_te_tfidf = hstack((X_test['school_state_0'].to_numpy().reshape(-1,1),X_test['school_state_1'].to_numpy().reshape(-1,1), X_test['teacher_


print("Final Data matrix")
print(X_tr_tfidf.shape, y_train.shape)
print(X_te_tfidf.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(33500, 8016) (33500,)
(16500, 8016) (16500,)
====================================================================================================
```

# Tfidf weighted w2v

### calculate idf value with only train data

In [150]:

```
preprocessed_essays_train = X_train['essay'].values    #using X_train essay to find idf_value of words

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

#### creating tfidf wighted w2v vectors for train data

In [151]:

```
# average Word2Vec
# compute average word2vec for each essay in train data.

tfidf_w2v_vectors_train = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm(preprocessed_essays_train): # for  each essay
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the each essay
    for word in sentence.split(): # for each word in a essay
        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_train.append(vector)

print(len(tfidf_w2v_vectors_train))
print(len(tfidf_w2v_vectors_train[0]))
```

```
100%|████████████████████████████████████████████████████████████████████████████| 33500/33500 [01:00<00:00, 556.02it/s]

33500
300
```

#### Creating tfidf wighted w2v vectors for test data

In [152]:

```
# average Word2Vec
# compute average word2vec for each essay in test data.
preprocessed_essays_test = X_test['essay'].values

tfidf_w2v_vectors_test = []; # the avg-w2v for each essay is stored in this list
for sentence in tqdm(preprocessed_essays_test): # for each essay
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the essay
    for word in sentence.split(): # for each word in a essay
        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_test.append(vector)

print(len(tfidf_w2v_vectors_test))
print(len(tfidf_w2v_vectors_test[0]))
```

```
100%|██████████████████████████████████████████████████████████████████████████| 16500/16500 [00:28<00:00, 570.10it/s]

16500
300
```

## Combining all the features of set2

In [194]:

```
from scipy import sparse
```

In [195]:

```
tfidf_w2v_vectors_train=np.array(tfidf_w2v_vectors_train)          #converting dense matrix into spare matrix
tfidf_w2v_vectors_train=sparse.csr_matrix(tfidf_w2v_vectors_train)

tfidf_w2v_vectors_test=np.array(tfidf_w2v_vectors_test)
tfidf_w2v_vectors_test=sparse.csr_matrix(tfidf_w2v_vectors_test)
```

In [198]:

```
from scipy.sparse import hstack
#all the necessary feature for set2 is stacked together horizontally
#stacked train features
X_tr_tfidf_w2v = hstack((X_train['school_state_0'].to_numpy().reshape(-1,1),X_train['school_state_1'].to_numpy().reshape(-1,1), X_train['
#stacked test features
X_te_tfidf_w2v = hstack((X_test['school_state_0'].to_numpy().reshape(-1,1),X_test['school_state_1'].to_numpy().reshape(-1,1), X_test['tea

print("Final Data matrix")
print(X_tr_tfidf_w2v.shape, y_train.shape)
print(X_te_tfidf_w2v.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(33500, 316) (33500,)
(16500, 316) (16500,)
====================================================================================================
```

In [70]:

```
'''with open('X_tr_tfidf_w2v.pickle', 'wb') as f:
    pickle.dump(X_tr_tfidf_w2v, f)

with open('X_te_tfidf_w2v.pickle', 'wb') as f:
    pickle.dump(X_te_tfidf_w2v, f)

with open('X_tr_tfidf.pickle', 'wb') as f:
    pickle.dump(X_tr_tfidf, f)

with open('X_te_tfidf.pickle', 'wb') as f:
    pickle.dump(X_te_tfidf, f)'''
```

In [71]:

```
'''with open('X_tr_tfidf_w2v.pickle', 'rb') as f:
    X_tr_tfidf_w2v=pickle.load(f)

with open('X_te_tfidf_w2v.pickle', 'rb') as f:
    X_te_tfidf_w2v=pickle.load(f)

with open('X_tr_tfidf.pickle', 'rb') as f:
    X_tr_tfidf=pickle.load(f)

with open('X_te_tfidf.pickle', 'rb') as f:
    X_te_tfidf=pickle.load(f)'''
```

Out[71]:

```
"with open('X_tr_tfidf_w2v.pickle', 'rb') as f:\n    X_tr_tfidf_w2v=pickle.load(f)\n    \nwith open('X_te_tfidf_w2v.pickl
e', 'rb') as f:\n    X_te_tfidf_w2v=pickle.load(f)\n\nwith open('X_tr_tfidf.pickle', 'rb') as f:\n    X_tr_tfidf=pickle.loa
d(f)\n\nwith open('X_te_tfidf.pickle', 'rb') as f:\n    X_te_tfidf=pickle.load(f)"
```

# Set1-Hyperparameter tuning

In [199]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import GradientBoostingClassifier


GBDT1 = GradientBoostingClassifier(learning_rate=0.1, n_estimators=60,random_state=42)  #Gradient boosted decision tree classifier
parameters = {'max_depth': [1, 3, 10, 30],'min_samples_split':[5, 10, 100, 500]}      #hyper parameter list for gridsearch

clf = GridSearchCV(GBDT1, parameters, cv=3, scoring='roc_auc',return_train_score=True,n_jobs=-1)           #applying gridsearch to find
clf.fit(X_tr_tfidf, y_train)                                      #fitting the GBDT model with train data

results = pd.DataFrame.from_dict(clf.cv_results_)          #storing Gridsearch results
print(results)


train_auc= results['mean_train_score']#storing required Gridsearch results in required variable
cv_auc = results['mean_test_score']
param_max_depth  =  results['param_max_depth'].tolist()
param_min_samples_split  = results['param_min_samples_split'].tolist()
```

```
    mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
0       51.626296      1.320836         0.308349        0.340055
1       51.023612      1.732617         0.065337        0.014271
2       49.495960      0.375699         0.070010        0.004543
3       51.206964      0.149836         0.063667        0.021476
4       96.427136      0.236846         0.043336        0.001706
5       94.613790      0.242305         0.045008        0.002169
6       95.807463      0.348406         0.046670        0.004996
7       94.994138      0.399234         0.049691        0.004495
8      354.084053      2.725889         0.072669        0.002626
9      338.762097      1.442760         0.070666        0.002624
10     314.587378      0.780597         0.068000        0.002163
11     297.038390      0.264275         0.071674        0.004496
12    1240.997900     12.419568         0.126677        0.001697
13    1146.388267      3.781063         0.125327        0.003298
14     993.044985      4.309970         0.123012        0.004234
15     867.886440      3.816465         0.120335        0.003298

   param_max_depth param_min_samples_split  \
```

In [200]:

```
'''with open('clf1.pickle', 'wb') as f:
    pickle.dump(clf, f)'''
```

In [202]:

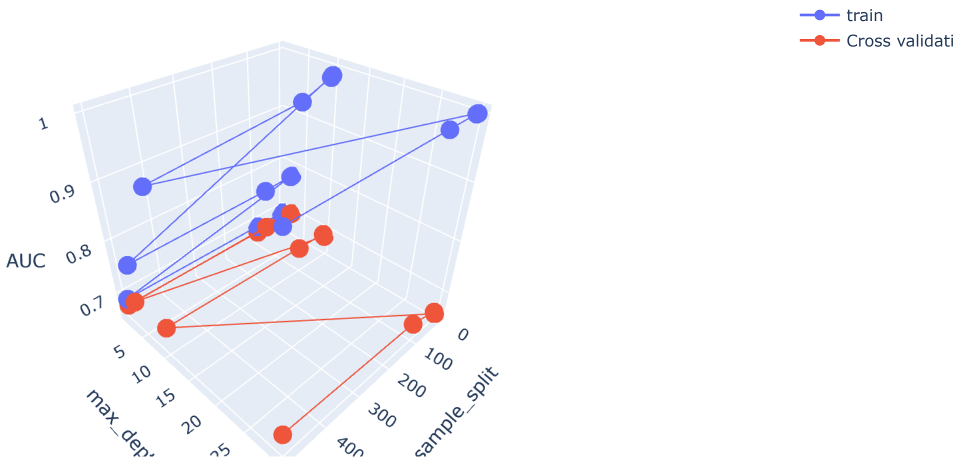
```
'''with open('results.pickle', 'wb') as f:
    pickle.dump(results, f)'''
```

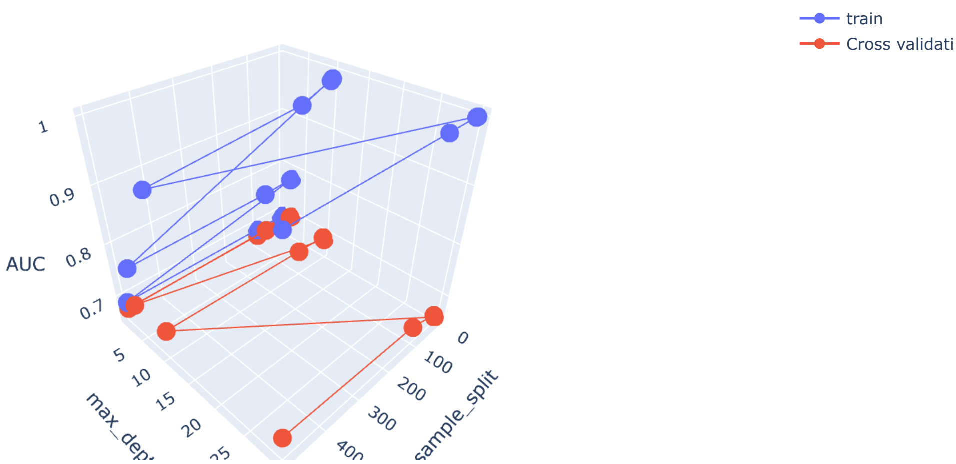
# Set1-Representation of results

# Set1 - plotting auc score vs hyperparameters

In [203]:

```
#plotting auc score vs hyperparameter using plotly library

# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=param_min_samples_split,y=param_max_depth,z=train_auc, name = 'train')          #train data
trace2 = go.Scatter3d(x=param_min_samples_split,y=param_max_depth,z=cv_auc, name = 'Cross validation')  #cv data
data = [trace1, trace2]

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

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




## Set1 - Heatmap

### For train data

In [204]:

```
#how to draw three variable heatmap: https://stackoverflow.com/a/39042065/17345549
df_train_hyperparameter = pd.DataFrame({'param_min_samples_split': param_min_samples_split, 'param_max_depth': param_max_depth, 'train_au
df_train_hyper_pivoted = df_train_hyperparameter.pivot("param_min_samples_split", "param_max_depth", "train_auc") #pivoting based on our r
```

In [205]:

```
df_train_hyperparameter.head(5)
```

Out[205]:

| | param_min_samples_split | param_max_depth | train_auc |
|---|---|---|---|
| **0** | 5 | 1 | 0.687365 |
| **1** | 10 | 1 | 0.687365 |
| **2** | 100 | 1 | 0.687365 |
| **3** | 500 | 1 | 0.687365 |
| **4** | 5 | 3 | 0.778643 |

In [206]:

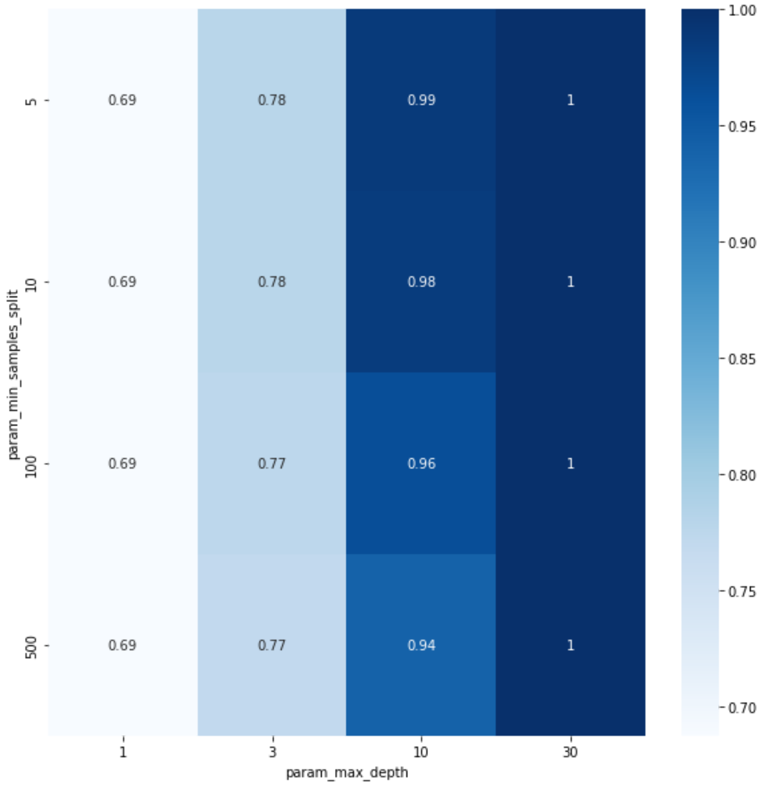
```
df_train_hyper_pivoted
```

Out[206]:

| param_max_depth | 1 | 3 | 10 | 30 |
|---|---|---|---|---|
| **param_min_samples_split** | | | | |
| **5** | 0.687365 | 0.778643 | 0.987283 | 1.000000 |
| **10** | 0.687365 | 0.778315 | 0.984718 | 1.000000 |
| **100** | 0.687365 | 0.774816 | 0.964078 | 0.999994 |
| **500** | 0.687365 | 0.769647 | 0.939233 | 0.999713 |

In [207]:

```
#plotting heatmap with x axis as parameter max depth , y axis as param min number of smaples splits and has auc value in inside cell

plt.figure(figsize = (10,10))  #how to draw correlation heatmap(3 values): https://www.geeksforgeeks.org/how-to-create-a-seaborn-correlati
dataplot = sns.heatmap(df_train_hyper_pivoted, cmap="Blues", annot=True)
```



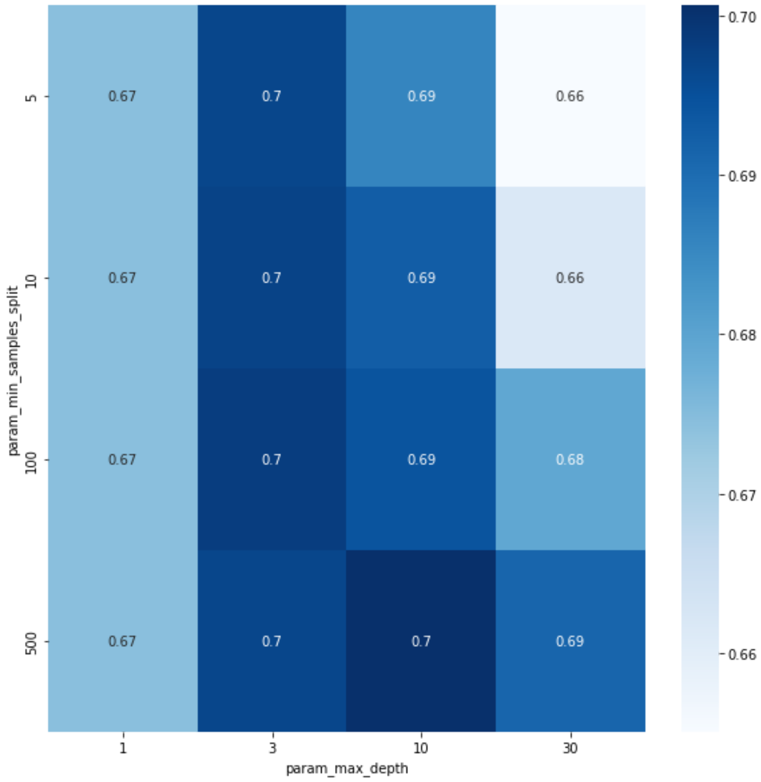
### For test data

In [208]:

```
#how to draw three variable heatmap: https://stackoverflow.com/a/39042065/17345549
df_cv_hyper = pd.DataFrame({'param_min_samples_split': param_min_samples_split, 'param_max_depth': param_max_depth, 'cv_auc': cv_auc})
df_cv_hyper_pivoted = df_cv_hyper.pivot("param_min_samples_split", "param_max_depth", "cv_auc")
```

In [209]:

```
plt.figure(figsize = (10,10))  #how to draw correlation heatmap: https://www.geeksforgeeks.org/how-to-create-a-seaborn-correlation-heatmap
dataplot = sns.heatmap(df_cv_hyper_pivoted, cmap="Blues", annot=True)
```

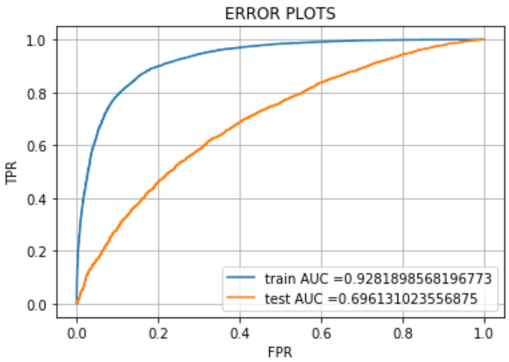



## set 1 - Fitting data with Best model

**From the above 3d plot and heatmap we can see that the mean test auc is maximum and the gap between mean train auc and mean test auc is minimum when min sample split=500 and max_depth=10**

In [210]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

GBDT1 = GradientBoostingClassifier(learning_rate=0.1, n_estimators=60,random_state=42,min_samples_split=500,max_depth=10)
GBDT1.fit(X_tr_tfidf, y_train)      #fitting the GBDT model
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class,not the predicted value

y_train_pred_proba = GBDT1.predict_proba(X_tr_tfidf)
y_test_pred_proba = GBDT1.predict_proba(X_te_tfidf)

#how to get a perticular column in nd array:https://stackoverflow.com/a/8386737/17345549
#[:,[1]].reshape(1,-1)[0]---to take probability values for the positive class

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred_proba[:,[1]].reshape(1,-1)[0]) #find FPR and TPR for plotting roc cu
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred_proba[:,[1]].reshape(1,-1)[0])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr))) #finding area under the ROC curve using FPR and 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("ERROR PLOTS")
plt.grid()
plt.show()
```




In [211]:

```
'''with open('GBDT1.pickle', 'wb') as f:
    pickle.dump(GBDT1, f)'''
```

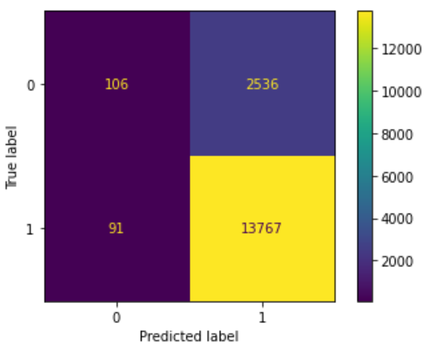
# Set1-plotting confusion matrix

In [212]:

```
plot_confusion_matrix(GBDT1, X_te_tfidf, y_test)
```

Out[212]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x298e3350700>
```




# Set2-Hyperparameter tuning

**Since most probabily we will get the best hyperparamet as max depth=10 and min_samples_split=500, we used gridsearch with very less number of parameter values in each hyperparameter for this set2 datapoints.Because hyperparameter tuning takes lots of time with limited computer resources even given with n_jobs=-1.**

In [213]:

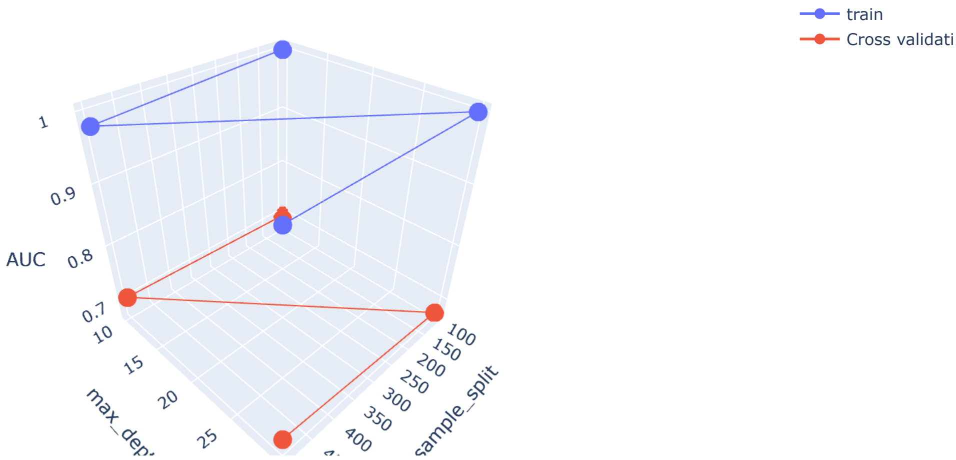
```
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

GBDT2 = GradientBoostingClassifier(learning_rate=0.1, n_estimators=60,random_state=52)  #Gradient boosted decision tree classifier

parameters = {'max_depth': [10, 30],'min_samples_split':[100, 500]}      #hyper parameter list for gridsearch
clf = GridSearchCV(GBDT2, parameters, cv=3, scoring='roc_auc',return_train_score=True,n_jobs=-1)            #applying gridsearch to find
clf.fit(X_tr_tfidf_w2v, y_train)                                 #fitting the DT model with train data

results = pd.DataFrame.from_dict(clf.cv_results_)          #storing Gridsearch results
print(results)


train_auc= results['mean_train_score']#storing required Gridsearch results in required variable
cv_auc = results['mean_test_score']
param_max_depth  =  results['param_max_depth'].tolist()
param_min_samples_split  = results['param_min_samples_split'].tolist()
```

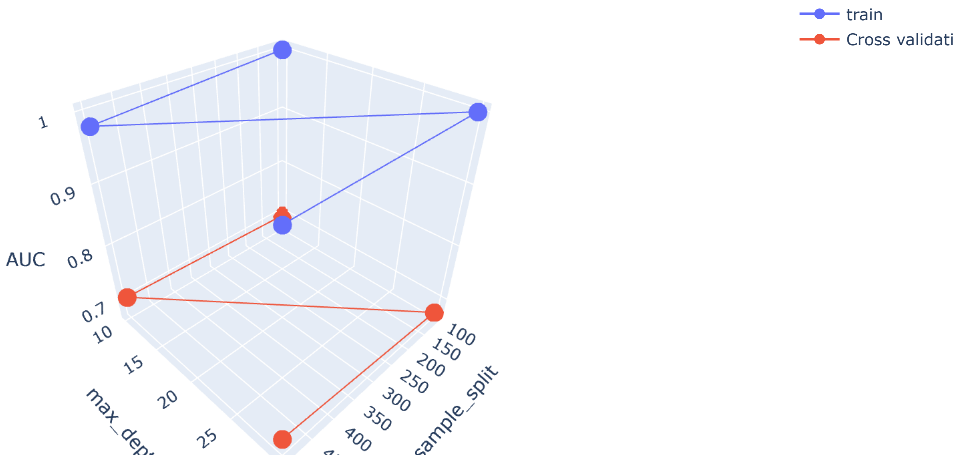
```
   mean_fit_time  std_fit_time  mean_score_time  std_score_time  \
0     758.774916      1.499788         0.130664        0.014394
1     608.224738      2.258559         0.164002        0.024777
2    2365.738817     12.503173         0.290545        0.027350
3     947.580245     10.843745         0.138669        0.002493

  param_max_depth param_min_samples_split  \
0              10                     100
1              10                     500
2              30                     100
3              30                     500

                                      params  split0_test_score  \
0  {'max_depth': 10, 'min_samples_split': 100}           0.678905
1  {'max_depth': 10, 'min_samples_split': 500}           0.691728
2  {'max_depth': 30, 'min_samples_split': 100}           0.663100
3  {'max_depth': 30, 'min_samples_split': 500}           0.684187

   split1_test_score  split2_test_score  mean_test_score  std_test_score  \
```

## Set2 - plotting auc score vs hyperparameters

In [214]:

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

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

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




## Set2 - Heatmap

### for train data

In [215]:

```
#how to draw three variable heatmap: https://stackoverflow.com/a/39042065/17345549
df_train_hyper = pd.DataFrame({'param_min_samples_split': param_min_samples_split, 'param_max_depth': param_max_depth, 'train_auc': train_
df_train_hyper_pivoted = df_train_hyper.pivot("param_min_samples_split", "param_max_depth", "train_auc")
```

In [216]:

```
df_train_hyper
```

Out[216]:

| | param_min_samples_split | param_max_depth | train_auc |
|---|---|---|---|
| **0** | 100 | 10 | 0.999572 |
| **1** | 500 | 10 | 0.981304 |
| **2** | 100 | 30 | 1.000000 |
| **3** | 500 | 30 | 0.999932 |

In [217]:

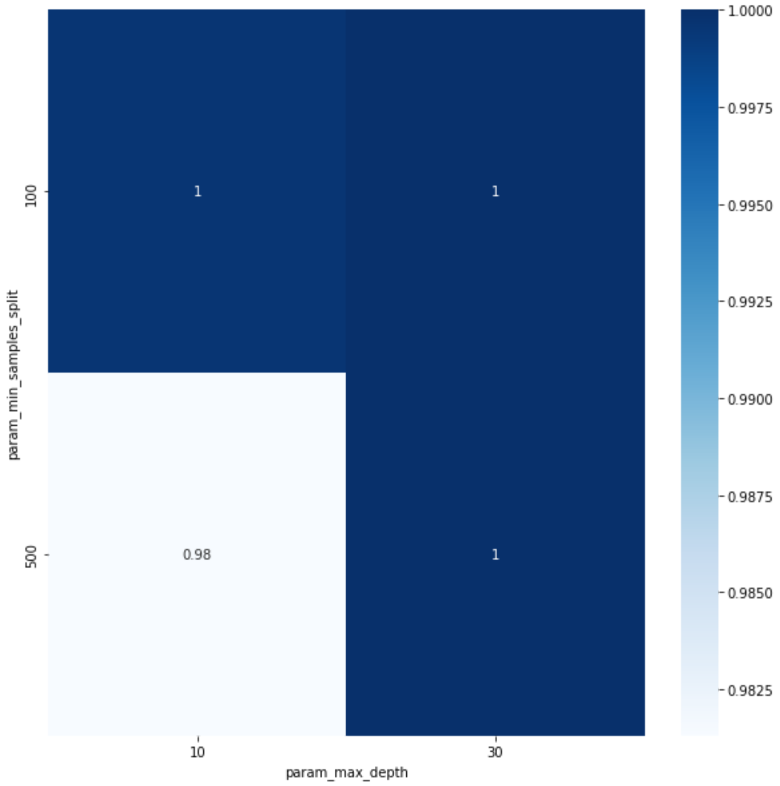
```
df_train_hyper_pivoted
```

Out[217]:

| param_max_depth | 10 | 30 |
|---|---|---|
| **param_min_samples_split** | | |
| **100** | 0.999572 | 1.000000 |
| **500** | 0.981304 | 0.999932 |

In [218]:

```
plt.figure(figsize = (10,10))  #how to draw correlation heatmap: https://www.geeksforgeeks.org/how-to-create-a-seaborn-correlation-heatmap
dataplot = sns.heatmap(df_train_hyper_pivoted, cmap="Blues", annot=True)
```



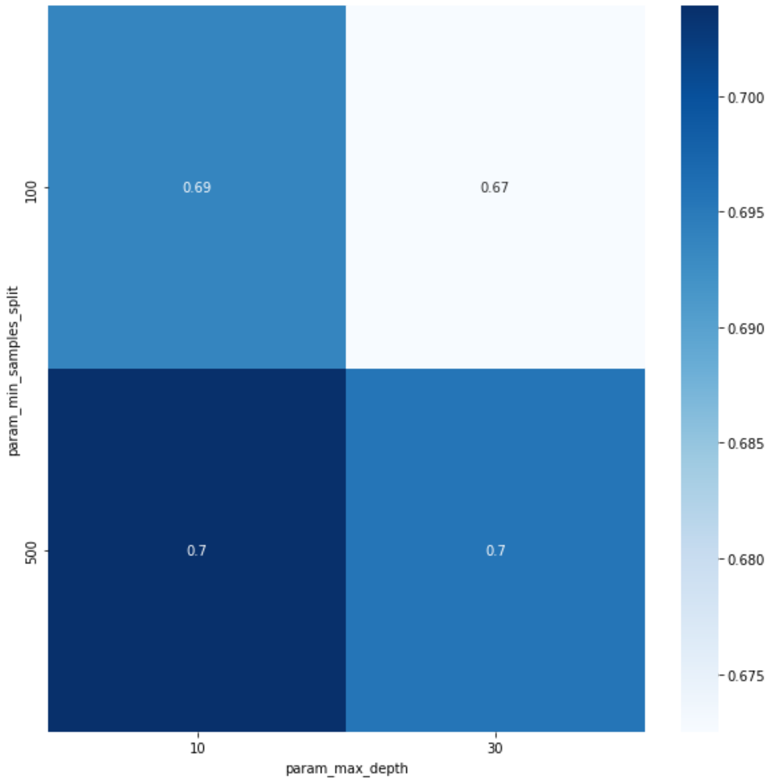
#### for cross validation data

In [219]:

```
#how to draw three variable heatmap: https://stackoverflow.com/a/39042065/17345549
df_cv_hyper = pd.DataFrame({'param_min_samples_split': param_min_samples_split, 'param_max_depth': param_max_depth, 'cv_auc': cv_auc})
df_cv_hyper_pivoted = df_cv_hyper.pivot("param_min_samples_split", "param_max_depth", "cv_auc")
```

In [220]:

```
plt.figure(figsize = (10,10))  #how to draw correlation heatmap: https://www.geeksforgeeks.org/how-to-create-a-seaborn-correlation-heatmap
dataplot = sns.heatmap(df_cv_hyper_pivoted, cmap="Blues", annot=True)
```

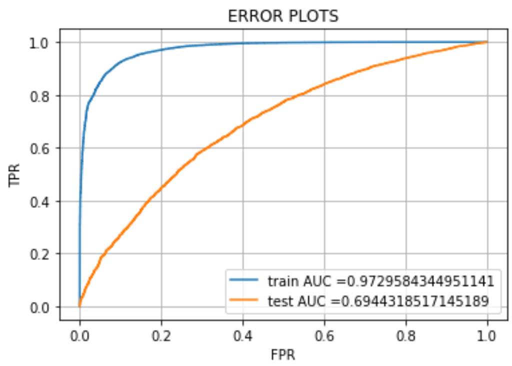



## set 2 - Best model

- From the above 3d plot and heatmap we can see that the mean test auc is maximum and the gap between mean train auc and mean test auc is minimum when min sample split=500 and max_depth=10

In [221]:

```
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

GBDT2 = GradientBoostingClassifier(learning_rate=0.1, n_estimators=60,random_state=52,min_samples_split=500,max_depth=10)
GBDT2.fit(X_tr_tfidf_w2v, y_train)     #fitting the GBDT model with parameter
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class,not the predicted value

y_train_pred_proba = GBDT2.predict_proba(X_tr_tfidf_w2v)
y_test_pred_proba = GBDT2.predict_proba(X_te_tfidf_w2v)

#how to get a perticular column in nd array:https://stackoverflow.com/a/8386737/17345549
#[:,[1]].reshape(1,-1)[0]---to take probability values for the positive class

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred_proba[:,[1]].reshape(1,-1)[0]) #find FPR and TPR for plotting roc cu
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred_proba[:,[1]].reshape(1,-1)[0])

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr))) #finding area under the ROC curve using FPR and 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("ERROR PLOTS")
plt.grid()
plt.show()
```




In [222]:

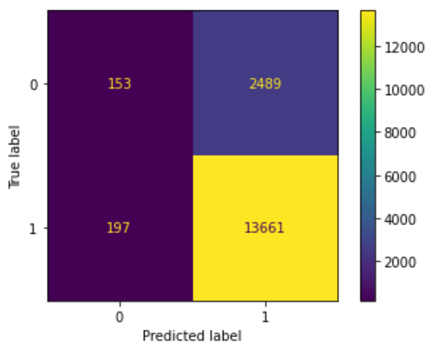
```
'''with open('GBDT2.pickle', 'wb') as f:
    pickle.dump(GBDT2, f)'''
```

In [223]:

```
plot_confusion_matrix(GBDT2, X_te_tfidf_w2v, y_test)
```

Out[223]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x298ddf1adc0>
```




# Summary

In [225]:

```
from prettytable import PrettyTable          # Reference Link for Pretty table:  https://pypi.org/project/prettytable/
x = PrettyTable()
```

In [227]:

```
x.field_names = ["Vectorizer","Model", "hyperparamerter", "train_AUC","test_AUC"]
x.add_row(["TFIDF","GBDT", "MSS=500 and MD=10",0.928189,0.696131])
x.add_row(["TFIDF weighted W2V","GBDT", "MSS=500 and MD=10", 0.972958,0.694431])
```

In [228]:

```
print(x)
```

```
+--------------------+-------+-------------------+-----------+----------+
|     Vectorizer     | Model |  hyperparamerter  | train_AUC | test_AUC |
+--------------------+-------+-------------------+-----------+----------+
|       TFIDF        |  GBDT | MSS=500 and MD=10 |  0.928189 | 0.696131 |
| TFIDF weighted W2V |  GBDT | MSS=500 and MD=10 |  0.972958 | 0.694431 |
+--------------------+-------+-------------------+-----------+----------+
```

Here in Hyperparameter column - MSS means "min sample split" and MD means "Max depth"

- Since GBDT is Tree based model it tend to **overfit with the train data**.That is the reason we have large train AUC.But it also **performed well on test data**.
- Since GBDT has **lots of hyperparameters** and it takes huge amount of time to find best hyperparamets by fitting the data with combinations of all hyperparameters, we just simply tuned 2 hyperparameter only which are **"min sample split" and "Max depth"**.
- If we have huge computer resource, we can finetune all hyperparameter to have best results.