Assignment 9: GBDT

Response Coding: Example

Train Data

State	class
A	0
B	1
C	1
A	0
A	1
B	1
A	0
A	1
C	1
C	0

Resonse table(only from train)

State	Class=0	Class=1
A	3	2
B	0	2
C	1	2

Encoded Train Data

State_0	State_1	class
3/5	2/5	0
0/2	2/2	1
1/3	2/3	1
3/5	2/5	0
3/5	2/5	1
0/2	2/2	1
3/5	2/5	0
3/5	2/5	1
1/3	2/3	1
1/3	2/3	0

Test Data

State
A
C
D
C
B
E

Encoded Test Data

State_0	State_1
3/5	2/5
1/3	2/3
1/2	1/2
1/3	2/3
0/2	2/2
1/2	1/2

> The response tabel is built only on train dataset. For a category which is not there in train data and present in test data, we will encode them with default values Ex: in our test data if have State: D then we encode it as [0.5, 0.05]

1. **Apply GBDT on these feature sets**

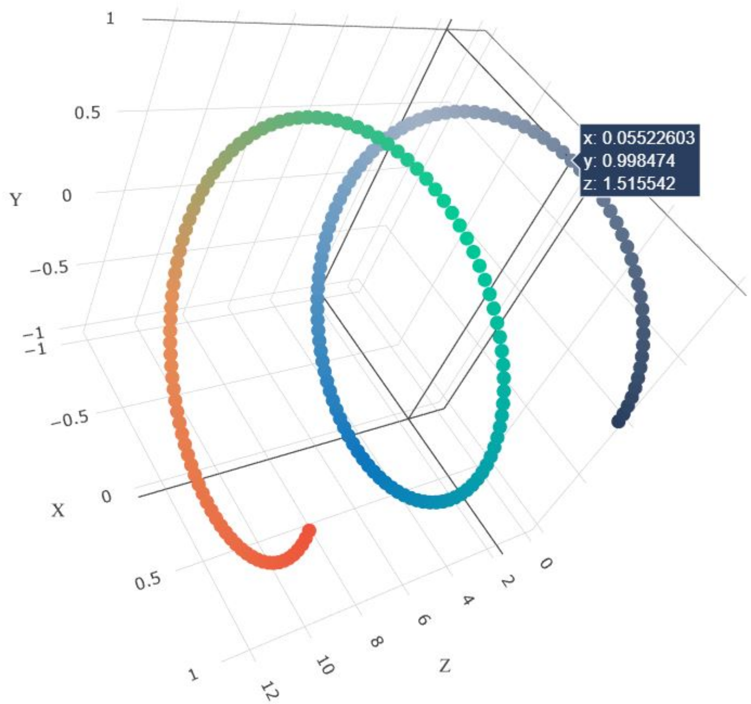
    - Set 1: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)+sentiment Score of eassay(check the bellow example, include all 4 values as 4 features)
    - Set 2: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V)

2. **The hyper paramter tuning (Consider any two hyper parameters)**

    - Find the best hyper parameter which will give the maximum AUC value
    - find the best hyper paramter using k-fold cross validation/simple cross validation data
    - use gridsearch cv or randomsearch cv or you can write your own for loops to do this task

3. **Representation of results**

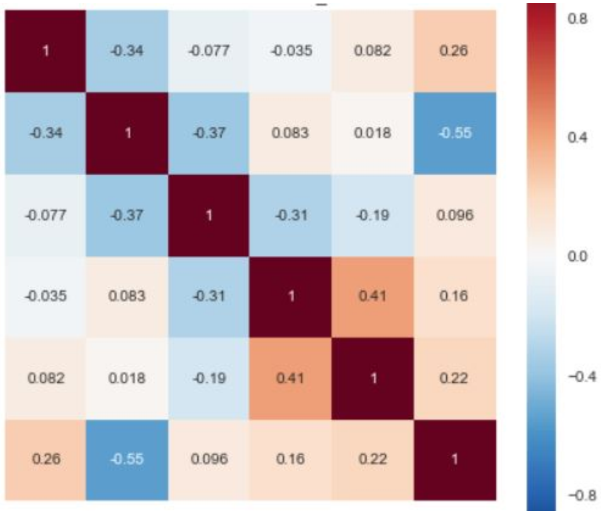
    - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure



with X-axis as **n_estimators**, Y-axis as **max_depth**, and Z-axis as **AUC Score** , we have given the notebook which explains how to plot this 3d plot, you can find it in the same drive *3d_scatter_plot.ipynb*

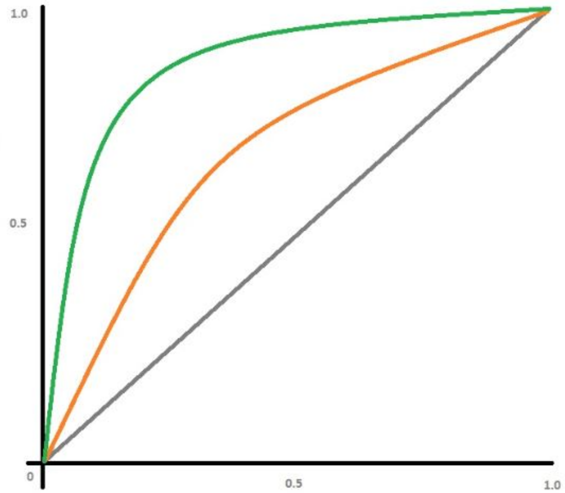
or

- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure



[seaborn heat maps](#) with rows as **n_estimators**, columns as **max_depth**, and values inside the cell representing **AUC Score**

- You choose either of the plotting techniques out of 3d plot or heat map
- Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.



- Along with plotting ROC curve, you need to print the [confusion matrix](#) with predicted and original labels of test data points

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

4. You need to summarize the results at the end of the notebook, summarize it in the table format

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

In [1]:
```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# import nltk
# nltk.download('vader_lexicon')

sid = SentimentIntensityAnalyzer()

for_sentiment = 'a person is a person no matter how small dr seuss i te
ach the smallest students with the biggest enthusiasm \
for learning my students learn in many different ways using all of our
 senses and multiple intelligences i use a wide range\
of techniques to help all my students succeed students in my class come
 from a variety of different backgrounds which makes\
for wonderful sharing of experiences and cultures including native amer
```

```
icans our school is a caring community of successful \
learners which can be seen through collaborative student project based
 learning in and out of the classroom kindergarteners \
in my class love to work with hands on materials and have many differen
t opportunities to practice a skill before it is\
mastered having the social skills to work cooperatively with friends is
 a crucial aspect of the kindergarten curriculum\
montana is the perfect place to learn about agriculture and nutrition m
y students love to role play in our pretend kitchen\
in the early childhood classroom i have had several kids ask me can we
 try cooking with real food i will take their idea \
and create common core cooking lessons where we learn important math an
d writing concepts while cooking delicious healthy \
food for snack time my students will have a grounded appreciation for t
he work that went into making the food and knowledge \
of where the ingredients came from as well as how it is healthy for the
ir bodies this project would expand our learning of \
nutrition and agricultural cooking recipes by having us peel our own ap
ples to make homemade applesauce make our own bread \
and mix up healthy plants from our classroom garden in the spring we wi
ll also create our own cookbooks to be printed and \
shared with families students will gain math and literature skills as w
ell as a life long enjoyment for healthy cooking \
nannan'
ss = sid.polarity_scores(for_sentiment)

for k in ss:
    print('{0}: {1}, '.format(k, ss[k]), end='')

# we can use these 4 things as features/attributes (neg, neu, pos, comp
ound)
# neg: 0.0, neu: 0.753, pos: 0.247, compound: 0.93
```

```
neg: 0.01, neu: 0.745, pos: 0.245, compound: 0.9975,
```

In [2]:

```
import numpy as np
from tqdm import tqdm

def sentiment(data):
```

```
    X_feat = np.zeros((data.shape[0],4))
    sid = SentimentIntensityAnalyzer()
    for idx in tqdm(range(data.shape[0])):
        senti = sid.polarity_scores(data[idx])
        X_feat[idx,0] = senti['neg']
        X_feat[idx,1] = senti['neu']
        X_feat[idx,2] = senti['pos']
        X_feat[idx,3] = senti['compound']
    return X_feat
```

1. GBDT (xgboost/lightgbm)

1.1 Loading Data

In [3]:
```
import pandas
data = pandas.read_csv('preprocessed_data.csv',nrows = 50000)
```

In [4]:
```
data.shape
```

Out[4]:
```
(50000, 9)
```

In [5]:
```
data.head(2)
```

Out[5]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_proj
0	ca	mrs	grades_prek_2	

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_proj
1	ut	ms	grades_3_5	

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

In [6]:

```
from sklearn.model_selection import train_test_split
X = data.drop('project_is_approved',axis=1)
y = data['project_is_approved']
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.20,ran
dom_state=100,stratify=y)

print(X_train.shape,y_train.shape)
print(X_test.shape,y_train.shape)
```

```
(40000, 8) (40000,)
(10000, 8) (40000,)
```

In [7]:

```
X_tr_essay_senti = sentiment(X_train['essay'].values)
X_te_essay_senti = sentiment(X_test['essay'].values)

X_essay_senti_features = ['neg','neu','pos','compound']
```

```
100%|██████████████████████████████████████████████████████████
████████| 40000/40000 [01:04<00:00, 621.47it/s]
100%|██████████████████████████████████████████████████████████
████████| 10000/10000 [00:16<00:00, 595.43it/s]
```

1.3 Make Data Model Ready: encoding eassay, and

project_title

In [8]:

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

In [9]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(min_df = 10)
vectorizer.fit(X_train['essay'].values)

X_train_essay_tfidf = vectorizer.transform(X_train['essay'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['essay'].values)
X_essay_tfidf_features = vectorizer.get_feature_names()

print('After TfIdf Vectorizer')
print(X_train_essay_tfidf.shape)
print(X_test_essay_tfidf.shape)
```

```
After TfIdf Vectorizer
(40000, 11109)
(10000, 11109)
```

In [10]:

```
def Tfidf_w2v(data):
    vectorizer = TfidfVectorizer()
    vectorizer.fit(data)

    idf_value = dict(zip(vectorizer.get_feature_names(),vectorizer.idf_
))
    feature = vectorizer.get_feature_names()

    tfidf_w2v = []
    for sentence in tqdm(data):
        w2v = np.zeros(300)
        tf_idf_values = 0
        for word in sentence.split():
            if (word in feature) and (word in glove_words):
```

```
                tfidf_value = idf_value[word]* sentence.count(word)/len
(sentence.split())
                w2v += model[word] * tfidf_value
                tf_idf_values += tfidf_value
        if tf_idf_values != 0:
            w2v /= tf_idf_values
        tfidf_w2v.append(w2v)
    return tfidf_w2v
```

In [11]:
```
X_train_essay_tfidfw2v = Tfidf_w2v(X_train['essay'].values)
X_test_essay_tfidfw2v = Tfidf_w2v(X_test['essay'].values)

print(' TfIdf W2V')
print('(',len(X_train_essay_tfidfw2v),',',len(X_train_essay_tfidfw2v[0
]),')')
print('(',len(X_test_essay_tfidfw2v),',',len(X_test_essay_tfidfw2v[0]),
')')
```

```
100%|██████████████████████████████████████████████████████████████████
██████████| 40000/40000 [37:07<00:00, 17.96it/s]
100%|██████████████████████████████████████████████████████████████████
██████████| 10000/10000 [04:59<00:00, 33.44it/s]
```

```
 TfIdf W2V
( 40000 , 300 )
( 10000 , 300 )
```

In [12]:
```
X_test=X_test.reset_index(drop=True)
```

In [13]:
```
X_train=X_train.reset_index(drop=True)
```

1.4 Make Data Model Ready: encoding numerical, categorical features

In [14]:
```
import numpy as np
```

```
import warnings
warnings.filterwarnings("ignore")


def response_value(X,Y,feature):
    probability = {}
    for value in X[feature].value_counts().index:
        total = X[X[feature] == value].shape[0]
        positive = X[(X[feature] == value) & (Y[:] == 1)].shape[0]
        negative = X[(X[feature] == value) & (Y[:] == 0)].shape[0]
        probability[value] = positive/total
    return probability


def encoded_value(k,feature,probability):
    Xfeature = np.zeros((k.shape[0],2))

    for index,row in k.iterrows():
        if row[feature] in probability.keys():
            pos_prob = probability[row[feature]]
            Xfeature[index,1] = pos_prob
            Xfeature[index,0] = 1 - pos_prob
        else:
            Xfeature[index,1] = 0.5
            Xfeature[index,0] = 0.5
    return Xfeature
```

In [15]:
```
school_state_response = response_value(X_train,y_train,'school_state')

X_train_school_state = encoded_value(X_train,'school_state',school_stat
e_response)
X_test_school_state = encoded_value(X_test,'school_state',school_state_
response)
X_state_features = ['school_state_0','school_state_1']
```

In [16]:
```
teacher_response = response_value(X_train,y_train,'teacher_prefix')
```

```
X_train_teacher_prefix = encoded_value(X_train,'teacher_prefix',teacher
_response)
X_test_teacher_prefix = encoded_value(X_test,'teacher_prefix',teacher_r
esponse)
X_teacher_features = ['teacher_prefix_0','teacher_prefix_1']
```

In [17]:

```
category_response = response_value(X_train,y_train,'clean_categories')

X_train_clean_category = encoded_value(X_train,'clean_categories',categ
ory_response)
X_test_clean_category = encoded_value(X_test,'clean_categories',categor
y_response)
X_category_features = ['clean_categories_0','clean_categories_1']
```

In [18]:

```
subcategory_response = response_value(X_train,y_train,'clean_subcategor
ies')

X_train_clean_subcategory = encoded_value(X_train,'clean_subcategories'
,subcategory_response)
X_test_clean_subcategory = encoded_value(X_test,'clean_subcategories',s
ubcategory_response)
X_subcategory_features = ['clean_subcategories_0','clean_subcategories_
1']
```

In [19]:

```
grade_response = response_value(X_train,y_train,'project_grade_categor
y')

X_train_grade_category = encoded_value(X_train,'project_grade_category'
,grade_response)
X_test_grade_category = encoded_value(X_test,'project_grade_category',g
rade_response)
X_grade_features = ['project_grade_category_0','project_grade_category_
1']
```

In [20]:

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

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

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

1.4.1 Concatinating all the features

Set 1

In [21]:

```
from scipy.sparse import hstack
X_tr_tfidf = hstack((X_tr_essay_senti,X_train_essay_tfidf,X_train_school_state,X_train_teacher_prefix,X_train_clean_category, X_train_clean_subcategory,X_train_grade_category,X_train_price_norm)).tocsr()
X_te_tfidf = hstack((X_te_essay_senti,X_test_essay_tfidf,X_test_school_state,X_test_teacher_prefix,X_test_clean_category, X_test_clean_subcategory,X_test_grade_category,X_test_price_norm)).tocsr()
X_feature = X_essay_senti_features + X_essay_tfidf_features + X_state_features + X_teacher_features + X_grade_features + X_category_features + X_subcategory_features + X_price_features

print("Final Data matrix")
print(X_tr_tfidf.shape, y_train.shape)
print(X_te_tfidf.shape, y_test.shape)
print('Feature size:',len(X_feature))
print("="*100)
```

```
Final Data matrix
(40000, 11124) (40000,)
(10000, 11124) (10000,)
Feature size: 11124
====================================================================================================
```

Set 2

In [22]:
```
from scipy import sparse

X_tr_tfidfw2v = sparse.csr_matrix(np.hstack((X_tr_essay_senti,X_train_e
ssay_tfidfw2v,X_train_school_state,X_train_teacher_prefix,X_train_clean
_category, X_train_clean_subcategory,X_train_grade_category,X_train_pri
ce_norm)))
X_te_tfidfw2v = sparse.csr_matrix(np.hstack((X_te_essay_senti,X_test_es
say_tfidfw2v,X_test_school_state,X_test_teacher_prefix,X_test_clean_cat
egory, X_test_clean_subcategory,X_test_grade_category,X_test_price_norm
)))

print("Final Data matrix")
print(X_tr_tfidfw2v.shape, y_train.shape)
print(X_te_tfidfw2v.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(40000, 315) (40000,)
(10000, 315) (10000,)
========================================================================
============================
```

1.5 Appling Models on different kind of featurization as mentioned in the instructions

Apply GBDT on different kind of featurization as mentioned in the instructions
For Every model that you work on make sure you do the step 2 and step 3 of instrucations

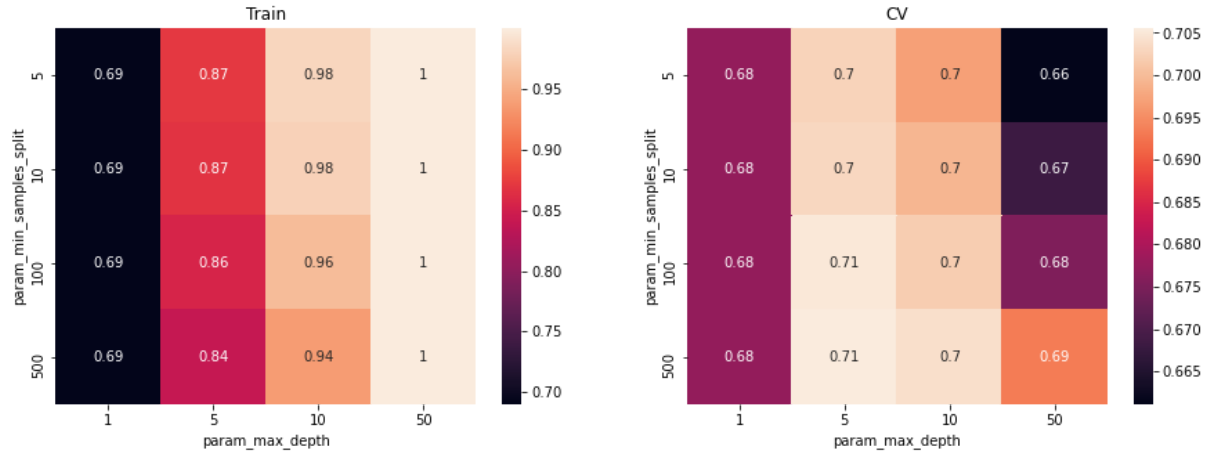
1.5.1 Hyper-Paramter Tuning : TFIDF

In [23]:
```
from sklearn.model_selection import GridSearchCV
```

```
from sklearn.ensemble import GradientBoostingClassifier
import pandas as pd
import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_auc_score

tree = GradientBoostingClassifier()
parameters = {'max_depth':[1,5,10,50],'min_samples_split': [5,10,100,50
0]}

clf_tfidf = GridSearchCV(tree,parameters,n_jobs = -1,scoring = 'roc_au
c', return_train_score=True)
final=clf_tfidf.fit(X_tr_tfidf,y_train)
```

1.5.2 Representation of TFIDF results

In [24]:
```
import matplotlib.pyplot as plt
import seaborn as sns


maximum_score = pd.DataFrame(clf_tfidf.cv_results_).groupby(['param_min
_samples_split', 'param_max_depth']).max().unstack()[['mean_test_score'
,'mean_train_score']]

fig, ax = plt.subplots(1,2, figsize=(15,5))
sns.heatmap(maximum_score.mean_train_score, annot = True, ax=ax[0])
sns.heatmap(maximum_score.mean_test_score, annot = True, ax=ax[1])
ax[0].set_title('Train')
ax[1].set_title('CV')
plt.show()
```



1.5.3 Training TFIDF model with best parameter

In [25]:
```
print(clf_tfidf.best_estimator_)

print(clf_tfidf.score(X_tr_tfidf,y_train))
print(clf_tfidf.score(X_te_tfidf,y_test))
```

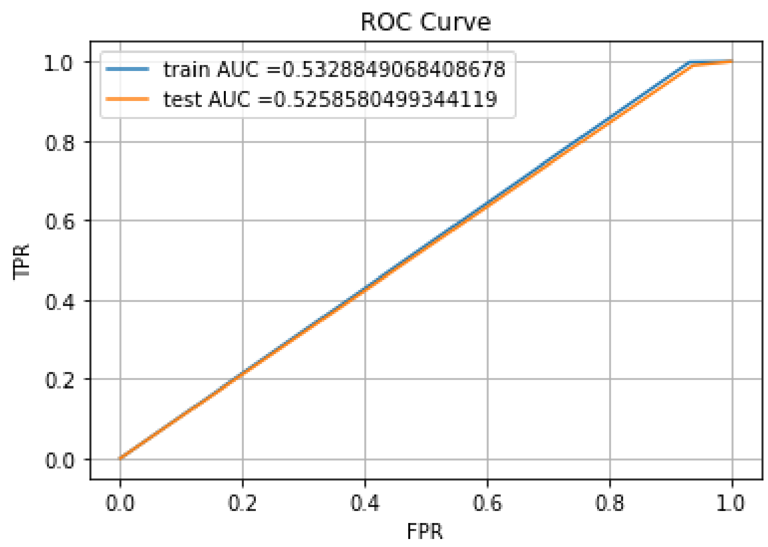
```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', ini
t=None,
                           learning_rate=0.1, loss='deviance', max_dept
h=5,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_spli
t=None,
                           min_samples_leaf=1, min_samples_split=500,
                           min_weight_fraction_leaf=0.0, n_estimators=1
00,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=None, subsample=1.0, tol=0.000
1,
```

```
                              validation_fraction=0.1, verbose=0,
                              warm_start=False)
0.8323697438760707
0.7025054066770835
```

In [26]:
```
best_max_depth_tfidf=clf_tfidf.best_params_['max_depth']
best_min_samples_split_tfidf=clf_tfidf.best_params_['min_samples_split'
]
```

In [27]:
```
%matplotlib inline
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt


best_model = GradientBoostingClassifier(max_depth = best_max_depth_tfid
f, min_samples_split = best_min_samples_split_tfidf)
%time best_model.fit(X_tr_tfidf,y_train)

y_train_pred = best_model.predict(X_tr_tfidf)
y_test_pred = best_model.predict(X_te_tfidf)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

tfidf_auc_score = auc(train_fpr, train_tpr)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, t
rain_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_
tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve")
plt.grid()
plt.show()
```

```
Wall time: 3min 44s
```



1.5.4 TFIDF Confusion Matrix

In [28]:
```
def find_best_threshold(threshold, fpr, tpr):
    t = threshold[np.argmax(tpr*(1-fpr))]

    print("The maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for th
reshold", np.round(t,3))
    return t

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

In [29]:
```
from sklearn.metrics import confusion_matrix

best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_
t)))
print("Test confusion matrix")
print(confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t
)))
```

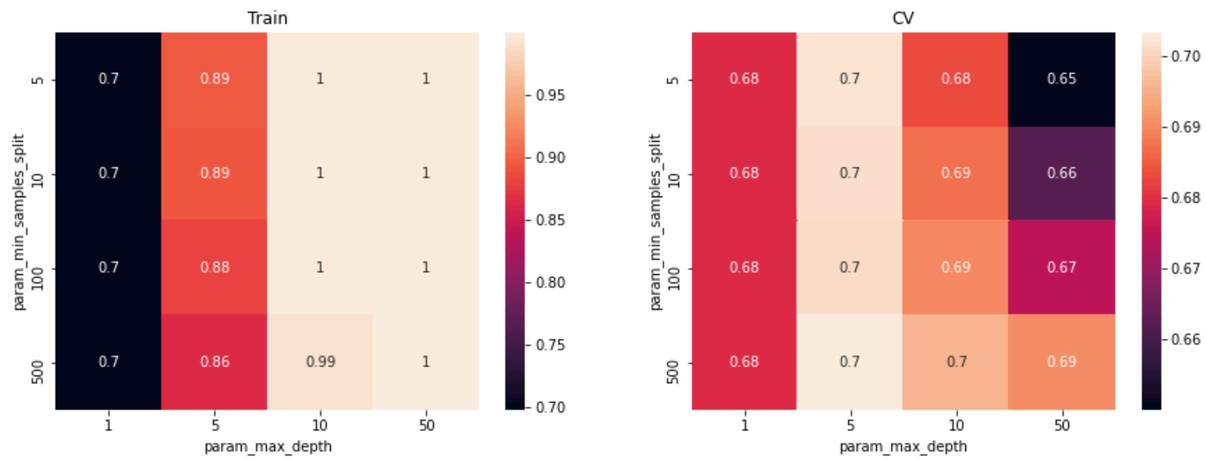
```
The maximum value of tpr*(1-fpr) 0.06732436331651702 for threshold 1
Train confusion matrix
[[  432  5974]
 [   56 33538]]
Test confusion matrix
[[  99 1502]
 [  85 8314]]
```

1.5.5 Hyper-Paramter Tuning : TFIDF W2V

In [30]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import GradientBoostingClassifier
import pandas as pd
import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import cross_val_score
from sklearn.metrics import roc_auc_score



tree_w2v = GradientBoostingClassifier()
parameters = {'max_depth':[1,5,10,50],'min_samples_split': [5,10,100,50
0]}

clf_tfidfw2v = GridSearchCV(tree_w2v,parameters,n_jobs = -1,scoring =
'roc_auc', return_train_score=True)
final=clf_tfidfw2v.fit(X_tr_tfidfw2v,y_train)
```

1.5.6 Representation of Average TFIDF W2V results

In [31]:
```
import seaborn as sns

maximum_score = pd.DataFrame(clf_tfidfw2v.cv_results_).groupby(['param_
min_samples_split', 'param_max_depth']).max().unstack()[['mean_test_sco
re','mean_train_score']]

fig, ax = plt.subplots(1,2, figsize=(15,5))
sns.heatmap(maximum_score.mean_train_score, annot = True, ax=ax[0])
sns.heatmap(maximum_score.mean_test_score, annot = True, ax=ax[1])
ax[0].set_title('Train')
ax[1].set_title('CV')
plt.show()
```



In [32]:
```
print(clf_tfidfw2v.best_estimator_)

print(clf_tfidfw2v.score(X_tr_tfidfw2v,y_train))
print(clf_tfidfw2v.score(X_te_tfidfw2v,y_test))
```

```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', ini
```

```
t=None,
                           learning_rate=0.1, loss='deviance', max_dept
h=5,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_spli
t=None,
                           min_samples_leaf=1, min_samples_split=500,
                           min_weight_fraction_leaf=0.0, n_estimators=1
00,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=None, subsample=1.0, tol=0.000
1,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
0.8481191359249718
0.7034597602001784
```

In [33]:

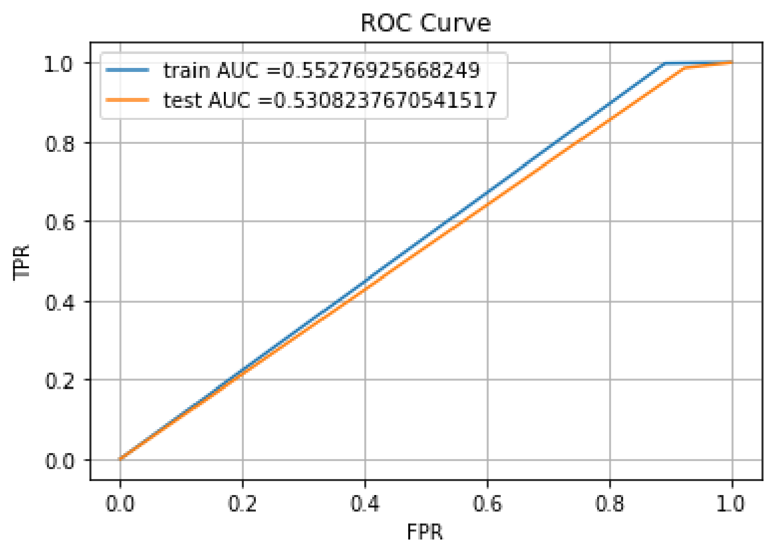
```
best_max_depth_tfidfw2v = clf_tfidfw2v.best_params_['max_depth']
best_min_samples_split_tfidfw2v = clf_tfidfw2v.best_params_['min_sample
s_split']
```

In [34]:

```
best_model_tfidfw2v = GradientBoostingClassifier(max_depth = best_max_d
epth_tfidfw2v, min_samples_split = best_min_samples_split_tfidfw2v)
%time best_model_tfidfw2v.fit(X_tr_tfidfw2v,y_train)

y_train_pred = best_model_tfidfw2v.predict(X_tr_tfidfw2v)
y_test_pred = best_model_tfidfw2v.predict(X_te_tfidfw2v)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

tfidfw2v_auc_score = auc(train_fpr, train_tpr)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, t
rain_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_
tpr)))
plt.legend()
```

```
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve")
plt.grid()
plt.show()
```

```
Wall time: 11min 11s
```



3. Summary

as mentioned in the step 4 of instructions

In [2]:
```
from prettytable import PrettyTable
tb = PrettyTable()
tb.field_names= (" Vectorizer ", " Max_depth ", " Min_sample_split ","
 Test -AUC ")
tb.add_row([" TfIdf", 10 , 500, 0.528 ])
tb.add_row(["TfIdf_w2v", 10 , 500, 0.530 ])
```

```python
print(tb.get_string(titles = "Observations"))
```

```
+--------------+------------+-------------------+------------+
|  Vectorizer  | Max_depth  |  Min_sample_split |  Test -AUC |
+--------------+------------+-------------------+------------+
|    TfIdf     |     10     |        500        |   0.528    |
|  TfIdf_w2v   |     10     |        500        |    0.53    |
+--------------+------------+-------------------+------------+
```

In [ ]: