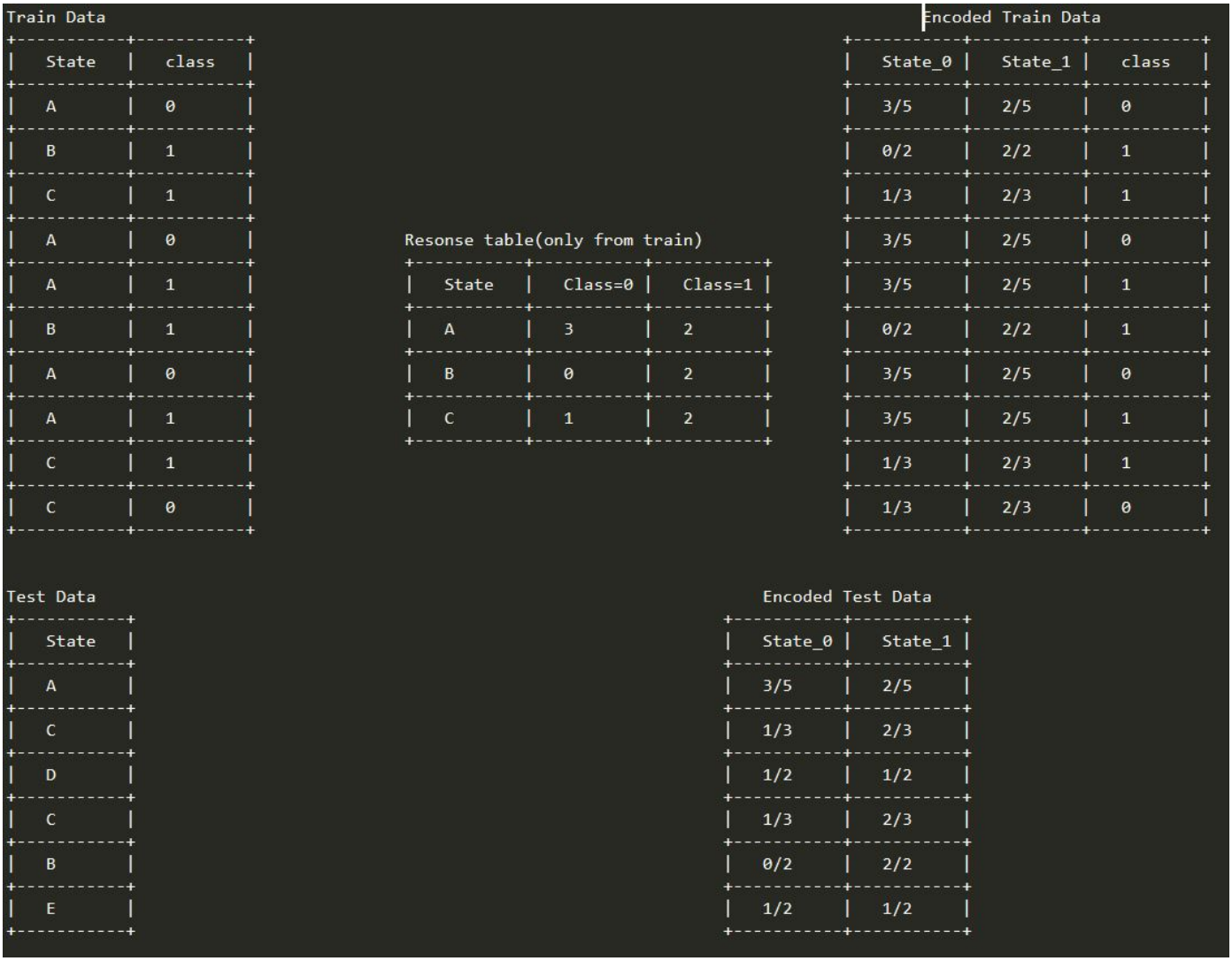
Assignment 9: GBDT

Response Coding: Example



> 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]

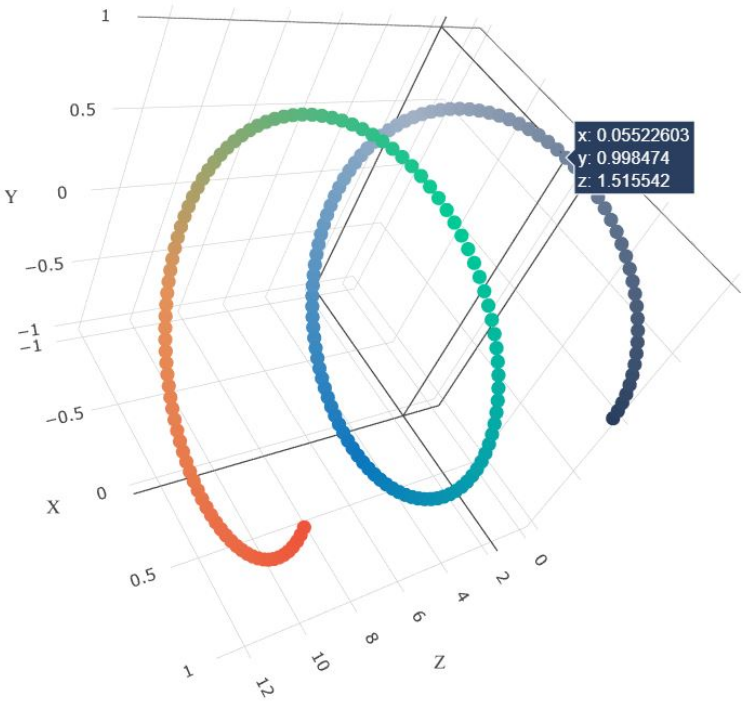
1. **Apply GBDT on these feature sets**
   - Set 1: categorical(instead of one hot encoding, try response coding (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/): use probability values), numerical features + project_title(TFIDF)+ 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 (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/): 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 (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) 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**

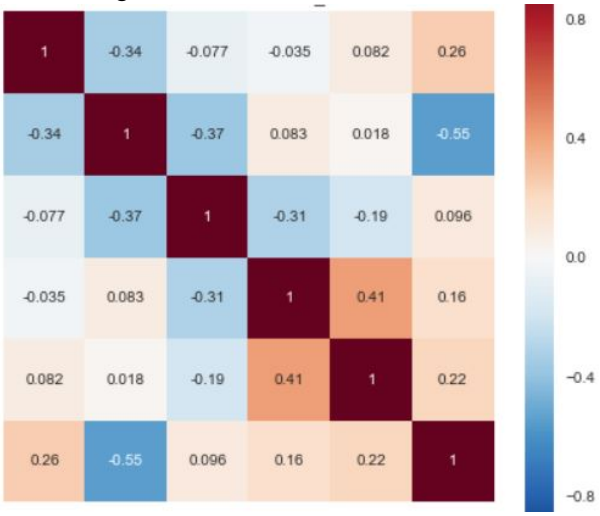
- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure



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*

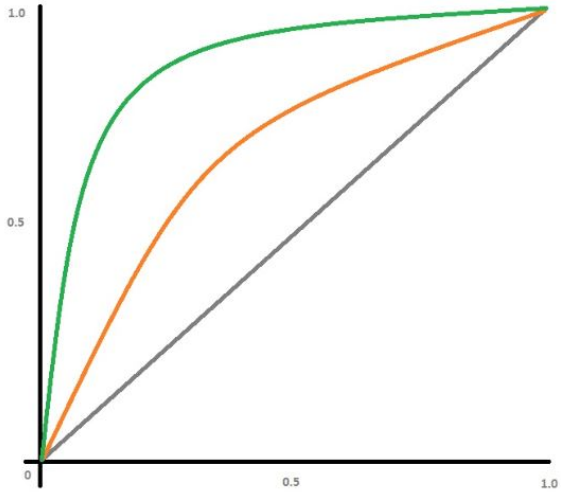
or

- 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 (https://seaborn.pydata.org/generated/seaborn.heatmap.html) 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 (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) 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

1. GBDT (xgboost/lightgbm)

1.1 Loading Data

In [1]:

```
import pandas
data = pandas.read_csv('preprocessed_data.csv', nrows = 50000)
```

In [2]:

```
data.shape
```

Out[2]:

```
(50000, 9)
```

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

In [3]:

```
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 = 0.4, stratify = Y)

X_train.reset_index(inplace = True, drop = True)
X_test.reset_index(inplace = True, drop = True)
X.shape,X_train.shape,X_test.shape
```

Out[3]:

```
((50000, 8), (30000, 8), (20000, 8))
```

1.3 Make Data Model Ready: encoding eassay, and project_title

Sentiment Score of essay

In [4]:

```
import numpy as np
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from tqdm import tqdm

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

```
C:\Anaconda3\lib\site-packages\nltk\twitter\__init__.py:20: UserWarning: T
he twython library has not been installed. Some functionality from the twi
tter package will not be available.
  warnings.warn("The twython library has not been installed. "
```

In [5]:

```
X_train_essay_ss = sentiment_features(X_train['essay'].values)
X_test_essay_ss = sentiment_features(X_test['essay'].values)

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

```
100%|████████████████████████████████████████████████████████████████
██████| 30000/30000 [01:33<00:00, 322.46it/s]
100%|████████████████████████████████████████████████████████████████
██████| 20000/20000 [00:57<00:00, 348.91it/s]
```

Set 1: TFIDF Vectorizer

In [6]:

```
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
(30000, 9956)
(20000, 9956)
```

Set 2: TFIDF W2V Vectorizer

In [7]:

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

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

In [8]:

```
import numpy as np
from tqdm import tqdm

def calculate_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 [9]:

```
X_train_essay_avgtfidfw2v = calculate_Tfidf_w2v(X_train['essay'].values)
X_test_essay_avgtfidfw2v = calculate_Tfidf_w2v(X_test['essay'].values)

print('After Average W2V TfIdf')
print('(',len(X_train_essay_avgtfidfw2v),',',len(X_train_essay_avgtfidfw2v[0]),')')
print('(',len(X_test_essay_avgtfidfw2v),',',len(X_test_essay_avgtfidfw2v[0]),')')
```

```
100%|████████████████████████████████████████████████████████████████████████
███████| 30000/30000 [49:04<00:00, 10.19it/s]
100%|████████████████████████████████████████████████████████████████████████
███████| 20000/20000 [27:36<00:00, 12.07it/s]

After Average W2V TfIdf
( 30000 , 300 )
( 20000 , 300 )
```

1.4 Make Data Model Ready: encoding numerical, categorical features

In [10]:

```
import numpy as np
import warnings
warnings.filterwarnings("ignore")

# calculating train unique data probability of positive class
def response_value(X,Y,feature):
    prob_value = {}
    for value in X[feature].value_counts().index:
        total_count = X[X[feature] == value].shape[0]
        positive_count = X[(X[feature] == value) & (Y[:] == 1)].shape[0]
        negative_count = X[(X[feature] == value) & (Y[:] == 0)].shape[0]
        prob_value[value] = positive_count/total_count
    return prob_value

# response coding function
def encoded_value(df,feature,prob_value):
    X_feature = np.zeros((df.shape[0],2))

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

In [11]:

```
# encoding categorical features: School State

school_response_value = response_value(X_train,Y_train,'school_state')

X_train_state_rc = encoded_value(X_train,'school_state',school_response_value)
X_test_state_rc = encoded_value(X_test,'school_state',school_response_value)
X_state_features = ['school_state_0','school_state_1']
```

In [12]:

```
# encoding categorical features: teacher_prefix

teacher_response_value = response_value(X_train,Y_train,'teacher_prefix')

X_train_teacher_rc = encoded_value(X_train,'teacher_prefix',teacher_response_value)
X_test_teacher_rc = encoded_value(X_test,'teacher_prefix',teacher_response_value)
X_teacher_features = ['teacher_prefix_0','teacher_prefix_1']
```

In [13]:

```
# encoding categorical features: clean_categories

category_response_value = response_value(X_train,Y_train,'clean_categories')

X_train_category_rc = encoded_value(X_train,'clean_categories',category_response_value)
X_test_category_rc = encoded_value(X_test,'clean_categories',category_response_value)
X_category_features = ['clean_categories_0','clean_categories_1']
```

In [14]:

```
# encoding categorical features: clean_subcategories

subcategory_response_value = response_value(X_train,Y_train,'clean_subcategories')

X_train_subcategory_rc = encoded_value(X_train,'clean_subcategories',subcategory_respons
X_test_subcategory_rc = encoded_value(X_test,'clean_subcategories',subcategory_response_
X_subcategory_features = ['clean_subcategories_0','clean_subcategories_1']
```

In [15]:

```
# encoding categorical features: project_grade_category

grade_response_value = response_value(X_train,Y_train,'project_grade_category')

X_train_grade_rc = encoded_value(X_train,'project_grade_category',grade_response_value)
X_test_grade_rc = encoded_value(X_test,'project_grade_category',grade_response_value)
X_grade_features = ['project_grade_category_0','project_grade_category_1']
```

In [16]:

```
# encoding numerical features: Price

from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
# this will rise an error Expected 2D array, got 1D array instead:
# array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
# Reshape your data either using
# array.reshape(-1, 1) if your data has a single feature
# array.reshape(1, -1)  if it contains a single sample.
normalizer.fit(X_train['price'].values.reshape(1,-1))

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

1.4.1 Concatinating all the features

Set 1

In [17]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr_tfidf = hstack((X_train_essay_ss, X_train_essay_tfidf, X_train_state_rc, X_train_t
X_te_tfidf = hstack((X_test_essay_ss, X_test_essay_tfidf, X_test_state_rc, X_test_teach
X_feature = X_essay_ss_features + X_essay_tfidf_features + X_state_features + X_teacher_

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
(30000, 9971) (30000,)
(20000, 9971) (20000,)
Feature size: 9971
====================================================================================
==========================
```

Set 2

In [18]:

```
from scipy import sparse

X_tr_avgtfidfw2v = sparse.csr_matrix(np.hstack((X_train_essay_ss, X_train_essay_avgtfid
X_te_avgtfidfw2v = sparse.csr_matrix(np.hstack((X_test_essay_ss, X_test_essay_avgtfidfw

print("Final Data matrix")
print(X_tr_avgtfidfw2v.shape, Y_train.shape)
print(X_te_avgtfidfw2v.shape, Y_test.shape)
print("="*100)
```

```
Final Data matrix
(30000, 315) (30000,)
(20000, 315) (20000,)
====================================================================================
==========================
```

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 [20]:

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

model_tfidf = GradientBoostingClassifier()
parameters = {'max_depth':[1,5,10,50],'min_samples_split': [5,10,100,500]}

clf_tfidf = GridSearchCV(model_tfidf,parameters,n_jobs = -1,scoring = 'roc_auc')
%time clf_tfidf.fit(X_tr_tfidf,Y_train)

results = pd.DataFrame.from_dict(clf_tfidf.cv_results_)

max_depth = results['param_max_depth']
min_samples_split = results['param_min_samples_split']
train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
```

```
Wall time: 6h 21min 41s
```

1.5.2 Representation of TFIDF results

In [21]:

```
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()

# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=min_samples_split,y=max_depth,z=train_auc, name = 'train')
trace2 = go.Scatter3d(x=min_samples_split,y=max_depth,z=cv_auc, name = 'Cross validatio
data2 = [trace1, trace2]

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

fig = go.Figure(data=data2, layout=layout)
%time offline.iplot(fig, filename='3d-scatter-colorscale')
```

Wall time: 197 ms

1.5.3 Training TFIDF model with best parameter

In [22]:

```
clf_tfidf.best_estimator_
```

Out[22]:

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=5,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=500,
              min_weight_fraction_leaf=0.0, n_estimators=100,
              presort='auto', random_state=None, subsample=1.0, verbose=0,
              warm_start=False)
```

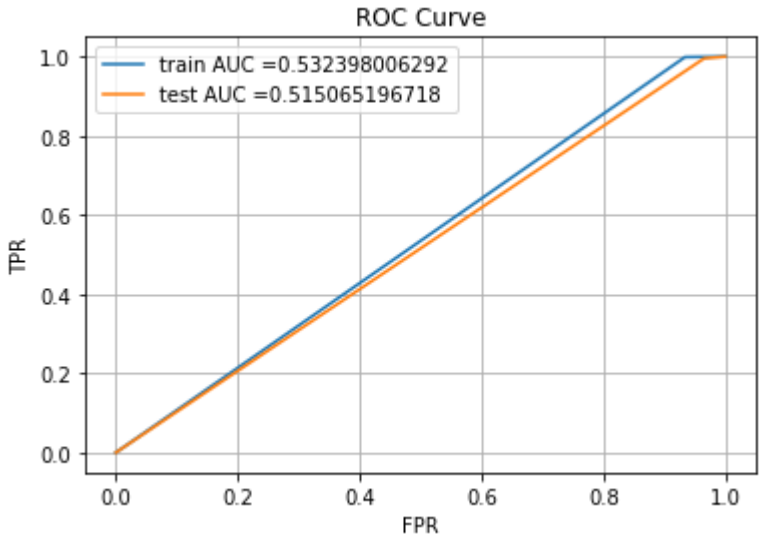
In [23]:

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

In [24]:

```
%matplotlib inline
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt


best_model = GradientBoostingClassifier(max_depth = best_max_depth_tfidf, min_samples_s
%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, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve")
plt.grid()
plt.show()
```

```
Wall time: 4min 42s
```



1.5.4 TFIDF Confusion Matrix

In [25]:

```
# we are writing our own function for predict, with defined thresould
# 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.rou
    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 [26]:

```
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) 0.0659081097286 for threshold 1
Train confusion matrix
[[  317  4487]
 [   30 25166]]
Test confusion matrix
[[  111  3092]
 [   76 16721]]
```

1.5.5 Hyper-Paramter Tuning : Average TFIDF W2V

In [28]:

```
model_avgtfidfw2v = GradientBoostingClassifier()
parameters = {'max_depth':[1,5,10,50],'min_samples_split': [5,10,100,500]}

clf_avgtfidfw2v = GridSearchCV(model_tfidf,parameters,n_jobs = -1,scoring = 'roc_auc')
%time clf_avgtfidfw2v.fit(X_tr_avgtfidfw2v,Y_train)

results = pd.DataFrame.from_dict(clf_avgtfidfw2v.cv_results_)

max_depth_avgtfidfw2v  = results['param_max_depth']
min_samples_split_avgtfidfw2v = results['param_min_samples_split']
train_auc_avgtfidfw2v = results['mean_train_score']
train_auc_std_avgtfidfw2v = results['std_train_score']
cv_auc_avgtfidfw2v  = results['mean_test_score']
cv_auc_std_avgtfidfw2v = results['std_test_score']
```

```
Wall time: 14h 13min 36s
```

1.5.6 Representation of Average TFIDF W2V results

In [29]:

```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=min_samples_split_avgtfidfw2v,y=max_depth_avgtfidfw2v,z=train_au
trace2 = go.Scatter3d(x=min_samples_split_avgtfidfw2v,y=max_depth_avgtfidfw2v,z=cv_auc_a
data2 = [trace1, trace2]

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

fig = go.Figure(data=data2, layout=layout)
%time offline.iplot(fig, filename='3d-scatter-colorscale')
```

Wall time: 247 ms

In [30]:

```
clf_avgtfidfw2v.best_estimator_
```

Out[30]:

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=5,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=500,
              min_weight_fraction_leaf=0.0, n_estimators=100,
              presort='auto', random_state=None, subsample=1.0, verbose=0,
              warm_start=False)
```

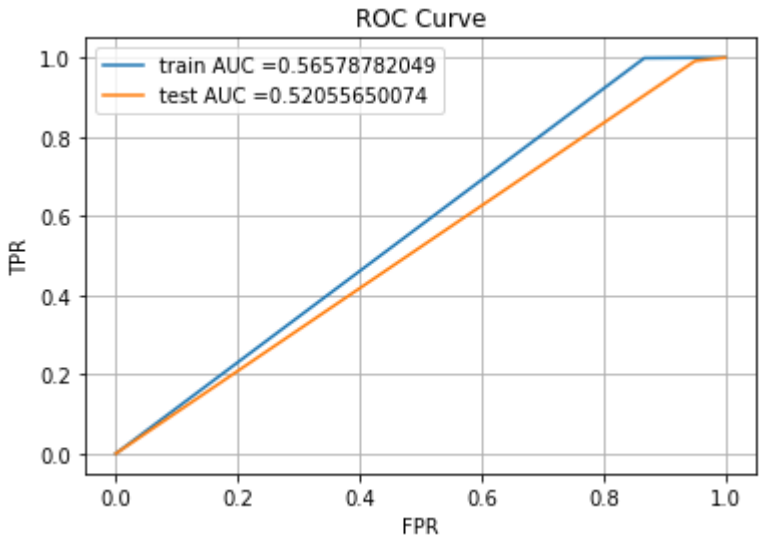
In [31]:

```
best_max_depth_avgtfidfw2v = clf_avgtfidfw2v.best_params_['max_depth']
best_min_samples_split_avgtfidfw2v = clf_avgtfidfw2v.best_params_['min_samples_split']
```

In [32]:

```
best_model_avgtfidfw2v = GradientBoostingClassifier(max_depth = best_max_depth_avgtfidfw
%time best_model_avgtfidfw2v.fit(X_tr_avgtfidfw2v,Y_train)

y_train_pred = best_model_avgtfidfw2v.predict(X_tr_avgtfidfw2v)
y_test_pred = best_model_avgtfidfw2v.predict(X_te_avgtfidfw2v)

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)

avgtfidfw2v_auc_score = auc(train_fpr, train_tpr)

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

```
Wall time: 13min 53s
```

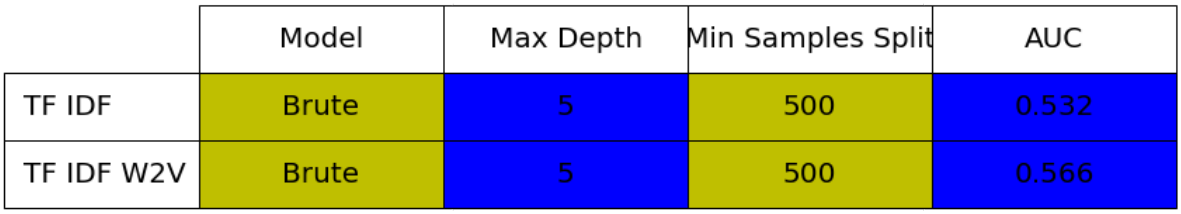


3. Summary

as mentioned in the step 4 of instructions

In [33]:

```
# https://pythonmatplotlibtips.blogspot.com/2018/11/matplotlib-only-table.html

fig = plt.figure()
ax = fig.add_subplot(111)
ax.grid(False)

col_labels = ['Model','Max Depth','Min Samples Split','AUC']
row_labels = ['TF IDF','TF IDF W2V']
table_vals = [['Brute', best_max_depth_tfidf, best_min_samples_split_tfidf, np.round(tf
# Draw table
the_table = plt.table(cellText=table_vals,
                      cellLoc = 'center',
                      cellColours = [['y','b','y','b'],['y','b','y','b']],
                      colWidths=[0.13] * 10,
                      rowLabels=row_labels,
                      colLabels=col_labels,
                      loc='center')
the_table.auto_set_font_size(False)
the_table.set_fontsize(20)
the_table.scale(4, 4)

# Removing ticks and spines enables you to get the figure only with table
plt.tick_params(axis='x', which='both', bottom=False, top=False, labelbottom=False)
plt.tick_params(axis='y', which='both', right=False, left=False, labelleft=False)
for pos in ['right','top','bottom','left']:
    plt.gca().spines[pos].set_visible(False)
```



	Model	Max Depth	Min Samples Split	AUC
TF IDF	Brute	5	500	0.532
TF IDF W2V	Brute	5	500	0.566