Assignment 8: DT

1. **Apply Decision Tree Classifier(DecisionTreeClassifier) on these feature sets**

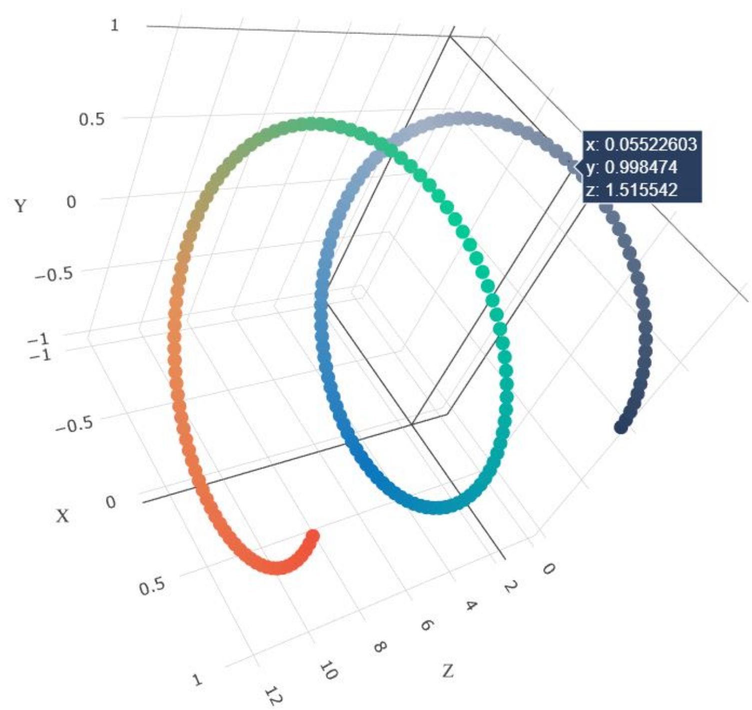
    - Set 1: categorical, numerical features + preprocessed_eassay (TFIDF)
    - Set 2: categorical, numerical features + preprocessed_eassay (TFIDF W2V)

2. **The hyper paramter tuning (best `depth` in range [1, 5, 10, 50], and the best `min_samples_split` in range [5, 10, 100, 500])**

    - 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/)](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(use gridsearch cv or randomsearch cv)/simple cross validation data(you can write your own for loops refer sample solution)

3. **Representation of results**

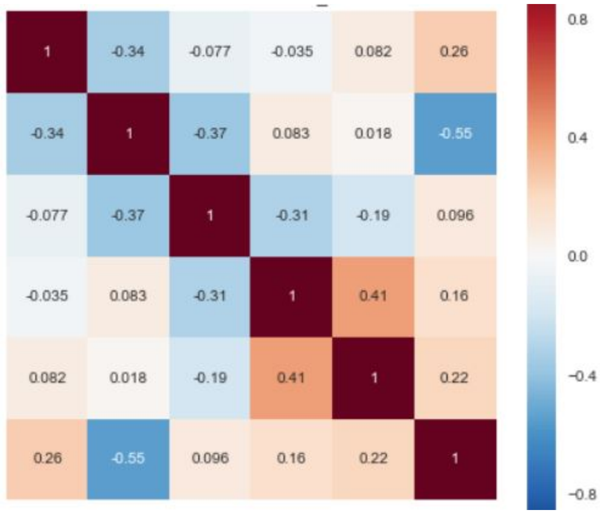
    - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure



    with X-axis as **min_sample_split**, 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*

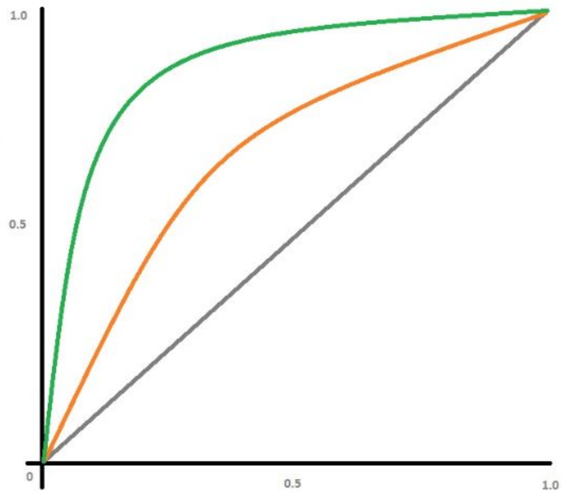
or

- 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)](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/)](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 = ??

- Once after you plot the confusion matrix with the test data, get all the `false positive data points`
  - Plot the WordCloud(https://www.geeksforgeeks.org/generating-word-cloud-python/) with the words of essay text of these `false positive data points`
  - Plot the box plot with the `price` of these `false positive data points`
  - Plot the pdf with the `teacher_number_of_previously_posted_projects` of these `false positive data points`

4. **Task 2:** For this task consider set-1 features. Select all the features which are having non-zero feature importance.You can get the feature importance using 'feature_importances_` (https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html), discard the all other remaining features and then apply any of the model of you choice i.e. (Dession tree, Logistic Regression, Linear SVM), you need to do hyperparameter tuning corresponding to the model you selected and procedure in step 2 and step 3
Note: when you want to find the feature importance make sure you don't use max_depth parameter keep it None.

5. 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   |
+--------------+----------+----------------+---------+
```

1. Decision Tree

1.1 Loading Data

In [1]:
```
import pandas
data = pandas.read_csv('preprocessed_data.csv', nrows=5000)
```

In [2]:
```
data.shape
```

Out[2]: (5000, 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_rem,Y_train,Y_rem = train_test_split(X,Y, test_size = 0.4, stratify
= Y)
X_cv,X_test,Y_cv,Y_test = train_test_split(X_rem,Y_rem, test_size = 0.5, strat
ify = Y_rem)

X.shape,X_train.shape,X_cv.shape,X_test.shape
```

Out[3]: ((5000, 8), (3000, 8), (1000, 8), (1000, 8))

1.3 Make Data Model Ready: encoding eassay, and project_title

Set 1: TFIDF Vectorizer

In [4]:
```
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_cv_essay_tfidf = vectorizer.transform(X_cv['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)
print(X_cv_essay_tfidf.shape)
```

```
After TfIdf Vectorizer
(3000, 3462)
(1000, 3462)
(1000, 3462)
```

Set 2: TFIDF W2V Vectorizer

In [5]:
```
# stronging variables into pickle files python: http://www.jessicayung.com/how
-to-use-pickle-to-save-and-load-variables-in-python/
# make sure you have the glove_vectors file
import pickle

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

In [6]:

```
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(senten
ce.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 [7]:

```
X_train_essay_avgtfidfw2v = calculate_Tfidf_w2v(X_train['essay'].values)
X_cv_essay_avgtfidfw2v = calculate_Tfidf_w2v(X_cv['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_cv_essay_avgtfidfw2v),',',len(X_cv_essay_avgtfidfw2v[0]),')')
print('(',len(X_test_essay_avgtfidfw2v),',',len(X_test_essay_avgtfidfw2v[0]),
')')
```

```
100%|██████████| 3000/3000 [02:00<00:00, 24.90it/s]
100%|██████████| 1000/1000 [00:26<00:00, 38.25it/s]
100%|██████████| 1000/1000 [00:26<00:00, 38.34it/s]

After Average W2V TfIdf
( 3000 , 300 )
( 1000 , 300 )
( 1000 , 300 )
```

1.4 Make Data Model Ready: encoding numerical, categorical features

In [8]:
```
from sklearn.feature_extraction.text import CountVectorizer

# encoding categorical features: School State

vectorizer = CountVectorizer()
vectorizer.fit(X_train['school_state'].values) # fit has to happen only on tra
in data

# we use the fitted CountVectorizer to convert the text to vector
X_train_state_ohe = vectorizer.transform(X_train['school_state'].values)
X_cv_state_ohe = vectorizer.transform(X_cv['school_state'].values)
X_test_state_ohe = vectorizer.transform(X_test['school_state'].values)
X_state_features = vectorizer.get_feature_names()

print("After vectorizations")
print(X_train_state_ohe.shape, Y_train.shape)
print(X_cv_state_ohe.shape, Y_cv.shape)
print(X_test_state_ohe.shape, Y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(3000, 51) (3000,)
(1000, 51) (1000,)
(1000, 51) (1000,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'i
a', 'id', 'il', 'in', 'ks', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'mo',
'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', 'nv', 'ny', 'oh', 'ok', 'or',
'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv', 'wy']
=============================================================================
=======================
```

In [9]:

```
# encoding categorical features: teacher_prefix

vectorizer = CountVectorizer()
vectorizer.fit(X_train['teacher_prefix'].values) # fit has to happen only on t
rain data

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_ohe = vectorizer.transform(X_train['teacher_prefix'].values)
X_cv_teacher_ohe = vectorizer.transform(X_cv['teacher_prefix'].values)
X_test_teacher_ohe = vectorizer.transform(X_test['teacher_prefix'].values)
X_teacher_features = vectorizer.get_feature_names()

print("After vectorizations")
print(X_train_teacher_ohe.shape, Y_train.shape)
print(X_cv_teacher_ohe.shape, Y_cv.shape)
print(X_test_teacher_ohe.shape, Y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(3000, 4) (3000,)
(1000, 4) (1000,)
(1000, 4) (1000,)
['mr', 'mrs', 'ms', 'teacher']
==============================================================================
======================
```

In [10]:

```
# encoding categorical features: clean_categories

vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_categories'].values) # fit has to happen only on
train data
# print(vectorizer.get_feature_names())
# we use the fitted CountVectorizer to convert the text to vector
X_train_category_ohe = vectorizer.transform(X_train['clean_categories'].values
)
X_cv_category_ohe = vectorizer.transform(X_cv['clean_categories'].values)
X_test_category_ohe = vectorizer.transform(X_test['clean_categories'].values)
X_category_features = vectorizer.get_feature_names()

print("After vectorizations")
print(X_train_category_ohe.shape, Y_train.shape)
print(X_cv_category_ohe.shape, Y_cv.shape)
print(X_test_category_ohe.shape, Y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(3000, 7) (3000,)
(1000, 7) (1000,)
(1000, 7) (1000,)
['appliedlearning', 'health_sports', 'history_civics', 'literacy_language',
'math_science', 'music_arts', 'specialneeds']
==============================================================================
======================
```

In [11]:
```
# encoding categorical features: clean_subcategories

vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_subcategories'].values) # fit has to happen only
on train data
# print(vectorizer.get_feature_names())
# we use the fitted CountVectorizer to convert the text to vector
X_train_subcategory_ohe = vectorizer.transform(X_train['clean_subcategories'].
values)
X_cv_subcategory_ohe = vectorizer.transform(X_cv['clean_subcategories'].values
)
X_test_subcategory_ohe = vectorizer.transform(X_test['clean_subcategories'].va
lues)
X_subcategory_features = vectorizer.get_feature_names()

print("After vectorizations")
print(X_train_subcategory_ohe.shape, Y_train.shape)
print(X_cv_subcategory_ohe.shape, Y_cv.shape)
print(X_test_subcategory_ohe.shape, Y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(3000, 28) (3000,)
(1000, 28) (1000,)
(1000, 28) (1000,)
['appliedsciences', 'charactereducation', 'civics_government', 'college_caree
rprep', 'communityservice', 'earlydevelopment', 'economics', 'environmentalsc
ience', 'esl', 'extracurricular', 'financialliteracy', 'foreignlanguages', 'g
ym_fitness', 'health_lifescience', 'health_wellness', 'history_geography', 'l
iteracy', 'literature_writing', 'mathematics', 'music', 'nutritioneducation',
'other', 'parentinvolvement', 'performingarts', 'socialsciences', 'specialnee
ds', 'teamsports', 'visualarts']
=============================================================================
=======================
```

In [12]:
```
# encoding categorical features: project_grade_category

vectorizer = CountVectorizer()

vectorizer.fit(X_train['project_grade_category'].values) #

vectorizer = CountVectorizer()
vectorizer.fit(X_train['project_grade_category'].values) # fit has to happen o
nly on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_grade_ohe = vectorizer.transform(X_train['project_grade_category'].val
ues)
X_cv_grade_ohe = vectorizer.transform(X_cv['project_grade_category'].values)
X_test_grade_ohe = vectorizer.transform(X_test['project_grade_category'].value
s)
X_grade_features = vectorizer.get_feature_names()

print("After vectorizations")
print(X_train_grade_ohe.shape, Y_train.shape)
print(X_cv_grade_ohe.shape, Y_cv.shape)
print(X_test_grade_ohe.shape, Y_test.shape)
print(vectorizer.get_feature_names())
print("="*100)
```

```
After vectorizations
(3000, 4) (3000,)
(1000, 4) (1000,)
(1000, 4) (1000,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
=============================================================================
=======================
```

In [13]:
```
# 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(-1,1)
X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(1,-1)).res
hape(-1,1)
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(1,-1))
.reshape(-1,1)
X_price_features = ['price']


print("After vectorizations")
print(X_train_price_norm.shape, Y_train.shape)
print(X_cv_price_norm.shape, Y_cv.shape)
print(X_test_price_norm.shape, Y_test.shape)
print("="*100)
```

```
After vectorizations
(3000, 1) (3000,)
(1000, 1) (1000,)
(1000, 1) (1000,)
==============================================================================
======================
```

1.5.1 Concatinating all the features

Set 1

In [14]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack
X_tr_tfidf = hstack((X_train_essay_tfidf, X_train_state_ohe, X_train_teacher_o
he, X_train_grade_ohe,X_train_category_ohe,X_train_subcategory_ohe, X_train_pr
ice_norm)).tocsr()
X_cr_tfidf = hstack((X_cv_essay_tfidf, X_cv_state_ohe, X_cv_teacher_ohe, X_cv_
grade_ohe,X_cv_category_ohe,X_cv_subcategory_ohe, X_cv_price_norm)).tocsr()
X_te_tfidf = hstack((X_test_essay_tfidf, X_test_state_ohe, X_test_teacher_ohe,
X_test_grade_ohe,X_test_category_ohe,X_test_subcategory_ohe, X_test_price_norm
)).tocsr()
X_feature = X_essay__tfidf_features + X_state_features + X_teacher_features +
X_grade_features + X_category_features + X_subcategory_features + X_price_feat
ures

print("Final Data matrix")
print(X_tr_tfidf.shape, Y_train.shape)
print(X_cr_tfidf.shape, Y_cv.shape)
print(X_te_tfidf.shape, Y_test.shape)
print('Feature size:',len(X_feature))
print("="*100)
```

```
Final Data matrix
(3000, 3557) (3000,)
(1000, 3557) (1000,)
(1000, 3557) (1000,)
Feature size: 3557
=============================================================================
=======================
```

Set 2

In [15]:
```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
X_tr_avgtfidfw2v = hstack((X_train_essay_avgtfidfw2v, X_train_state_ohe, X_tra
in_teacher_ohe, X_train_grade_ohe,X_train_category_ohe,X_train_subcategory_ohe
, X_train_price_norm)).tocsr()
X_cr_avgtfidfw2v = hstack((X_cv_essay_avgtfidfw2v, X_cv_state_ohe, X_cv_teache
r_ohe, X_cv_grade_ohe,X_cv_category_ohe,X_cv_subcategory_ohe, X_cv_price_norm
)).tocsr()
X_te_avgtfidfw2v = hstack((X_test_essay_avgtfidfw2v, X_test_state_ohe, X_test_
teacher_ohe, X_test_grade_ohe,X_test_category_ohe,X_test_subcategory_ohe, X_te
st_price_norm)).tocsr()
# X_feature = X_essay__tfidf_features + X_state_features + X_teacher_features
 + X_grade_features + X_category_features + X_subcategory_features + X_price_f
eatures

print("Final Data matrix")
print(X_tr_avgtfidfw2v.shape, Y_train.shape)
print(X_cr_avgtfidfw2v.shape, Y_cv.shape)
print(X_te_avgtfidfw2v.shape, Y_test.shape)
# print('Feature size:',len(X_feature))
print("="*100)
```

```
Final Data matrix
(3000, 395) (3000,)
(1000, 395) (1000,)
(1000, 395) (1000,)
==============================================================================
======================
```

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

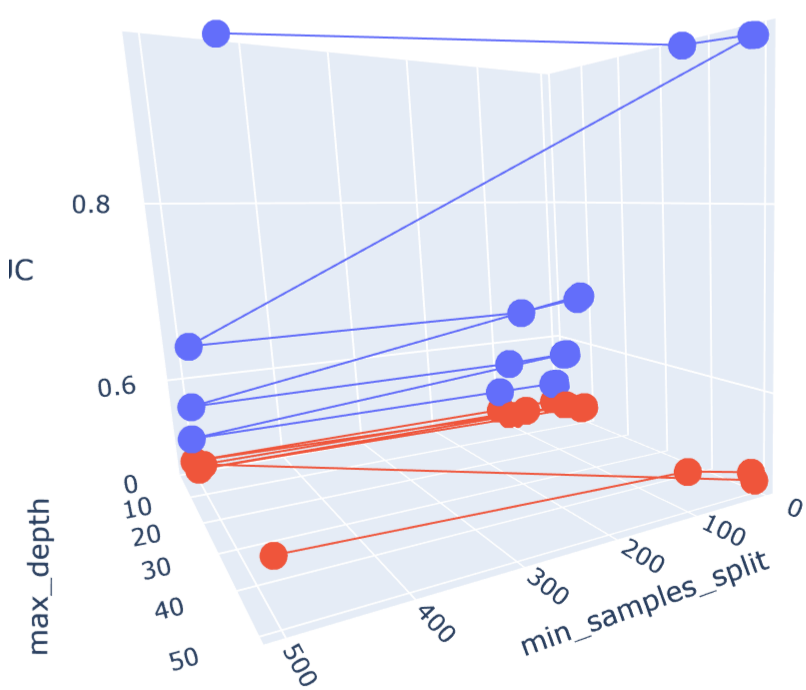
Apply Decision Tree on different kind of featurization as mentioned in the instructions
For Every model that you work on make sure you do the step 2 and step 3 of instrucations

1.5.1 Hyper-Paramter Tuning : TFIDF

In [19]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
import pandas as pd
import warnings
warnings.filterwarnings("ignore")

model_tfidf = DecisionTreeClassifier()
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_au
c')
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']
```

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 = 'tra
in')
trace2 = go.Scatter3d(x=min_samples_split,y=max_depth,z=cv_auc, name = 'Cross
 validation')
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)
offline.iplot(fig, filename='3d-scatter-colorscale')
```



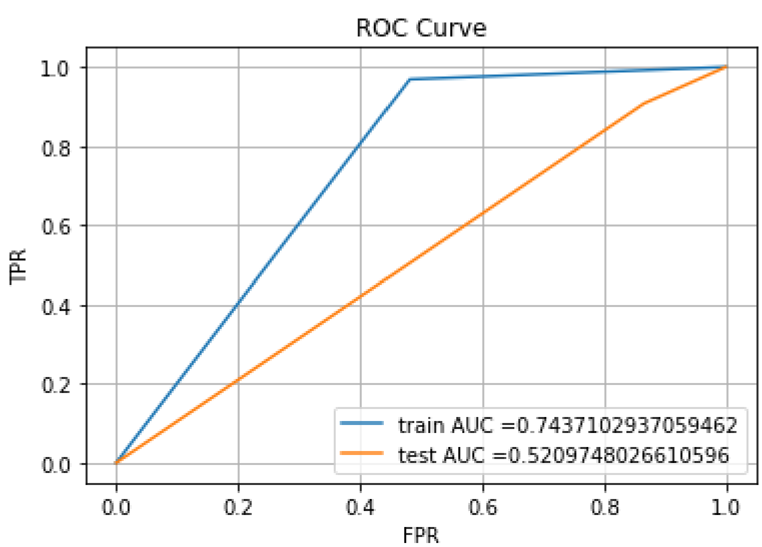
1.5.3 Training TFIDF model with best parameter

In [18]:
```
clf_tfidf.best_estimator_
```

Out[18]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=50,
                       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, presort=False, random_state=None,
                       splitter='best')
```

In [22]:
```
best_max_depth=50
best_min_samples_split=500
```

In [24]:
```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

best_model = DecisionTreeClassifier(max_depth = best_max_depth, min_samples_sp
lit = best_min_samples_split)
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)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tp
r)))
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()
```



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 hi
gh
    print("The maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshol
d", 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 [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) 0.5024350486484214 for threshold 1
Train confusion matrix
[[ 209  194]
 [  81 2516]]
Test confusion matrix
[[ 18 116]
 [ 80 786]]
```

1.5.5 Plot the WordCloud with the words of essay text of these `false positive data points`

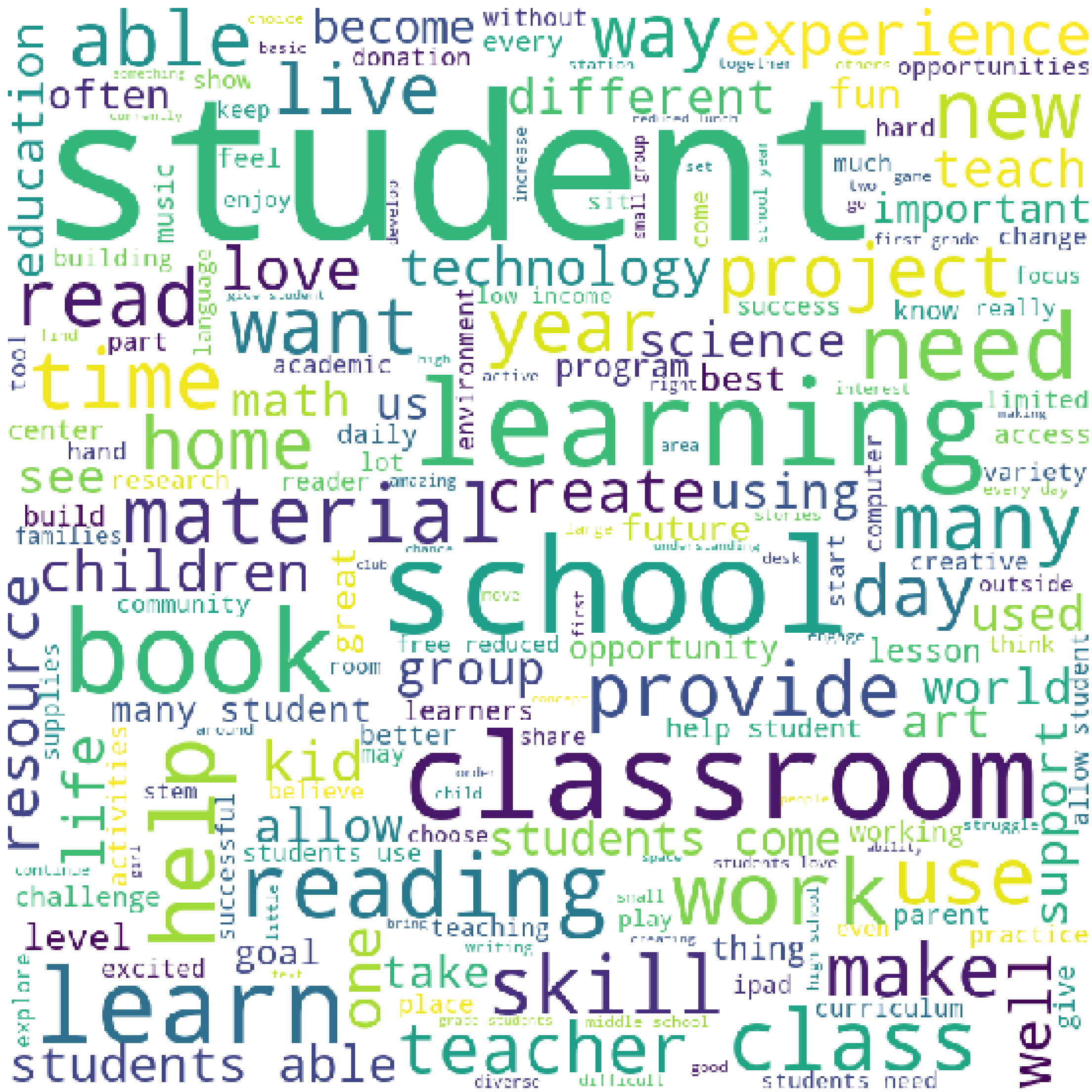
In [27]:

```
idx = []
for i in range(len(Y_train)):
    if (Y_train.iloc[i] == 0) and (y_train_pred[i] == 1):
        idx.append(i)
```

In [28]:

```
fpr_data = data.iloc[idx]
```

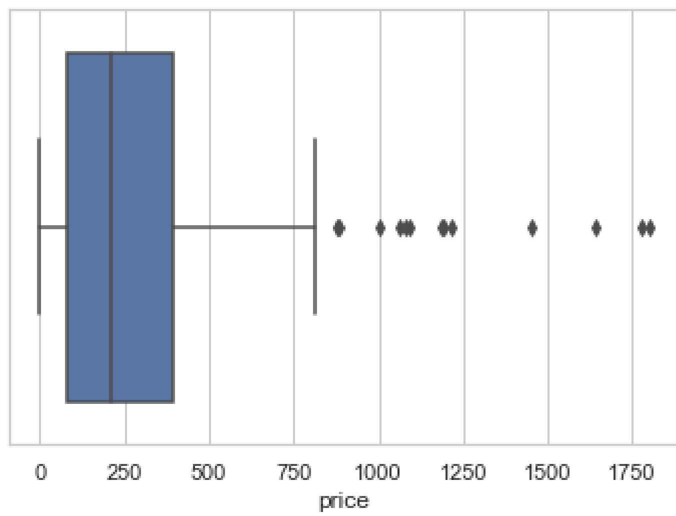
In [29]:
```
from wordcloud import WordCloud, STOPWORDS

comment_words = ' '
stopwords = set(STOPWORDS)

# iterate through the csv file
for val in fpr_data['essay']:

    # typecaste each val to string
    val = str(val)

    # split the value
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    for words in tokens:
        comment_words = comment_words + words + ' '


wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



1.5.6 Plot the box plot with the `price` of these `false positive data points`

In [30]:
```
import seaborn as sns

sns.set(style="whitegrid")
sns.boxplot(fpr_data['price'])
```

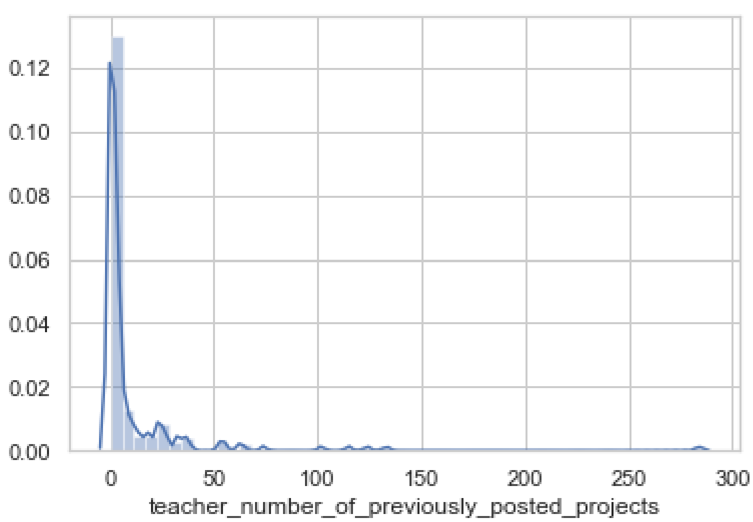
Out[30]: <matplotlib.axes._subplots.AxesSubplot at 0x1dbf29e8>



1.5.7 Plot the pdf with the `teacher_number_of_previously_posted_projects` of `these false positive data points`

In [31]:
```
sns.distplot(fpr_data['teacher_number_of_previously_posted_projects'])
```

Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0x1d7d8b00>

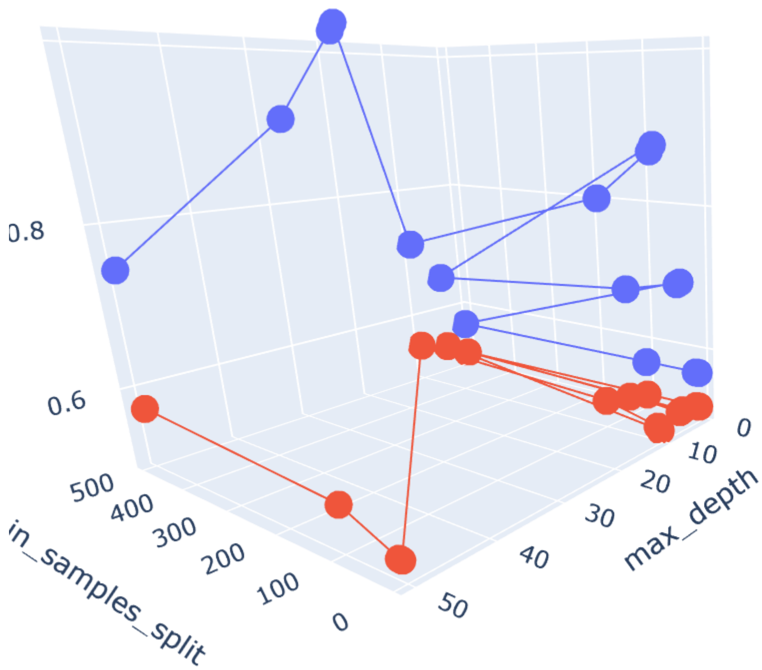


1.5.8 Hyper-Paramter Tuning : Average TFIDF W2V

In [32]:
```
model_avgtfidfw2v = DecisionTreeClassifier()
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')
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']
```

1.5.9 Representation of Average TFIDF W2V results

In [33]:
```
# https://plot.ly/python/3d-axes/
trace1 = go.Scatter3d(x=min_samples_split_avgtfidfw2v,y=max_depth_avgtfidfw2v,
z=train_auc_avgtfidfw2v, name = 'train')
trace2 = go.Scatter3d(x=min_samples_split_avgtfidfw2v,y=max_depth_avgtfidfw2v,
z=cv_auc_avgtfidfw2v, name = 'Cross validation')
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)
offline.iplot(fig, filename='3d-scatter-colorscale')
```



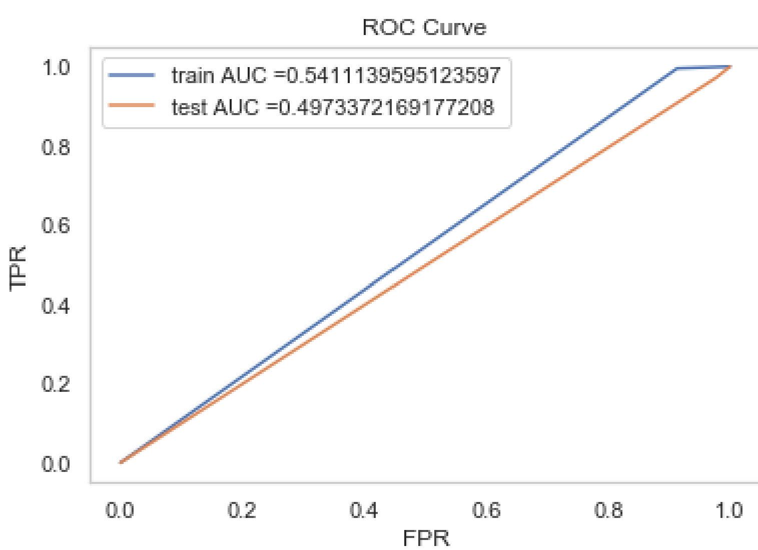
1.5.10 Training Average TFIDF W2V model with best parameter

In [34]:
```
clf_avgtfidfw2v.best_estimator_
```

Out[34]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=50,
            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, presort=False, random_state=None,
            splitter='best')
```

In [35]:
```
best_max_depth_avgtfidfw2v=50
best_min_samples_split_avgtfidfw2v=500
```

In [36]:
```
best_model_avgtfidfw2v = DecisionTreeClassifier(max_depth = best_max_depth, mi
n_samples_split = best_min_samples_split)
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)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tp
r)))
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()
```



1.6 Getting top features using `feature_importances_`

In [37]:
```
tfidf_data = pd.DataFrame(data=X_tr_tfidf.toarray()[:,:], index= range(len(X_t
r_tfidf.toarray())), columns=X_feature)
tfidf_data.head()
```

Out[37]:

	000	10	100	10th	11	12	12th	13	14	15	...	music	nutritioneducation	other	parent
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0	
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	

5 rows × 3557 columns

In [38]:
```
# Extracting best feature index

best_feature_idx = []
for idx in range(len(X_feature)):
    if best_model.feature_importances_[idx]>0:
        best_feature_idx.append(idx)
len(best_feature_idx)
```

Out[38]: 79

In [39]:
```
feat_imp_data = tfidf_data.iloc[:,best_feature_idx]
feat_imp_data.head()
```

Out[39]:

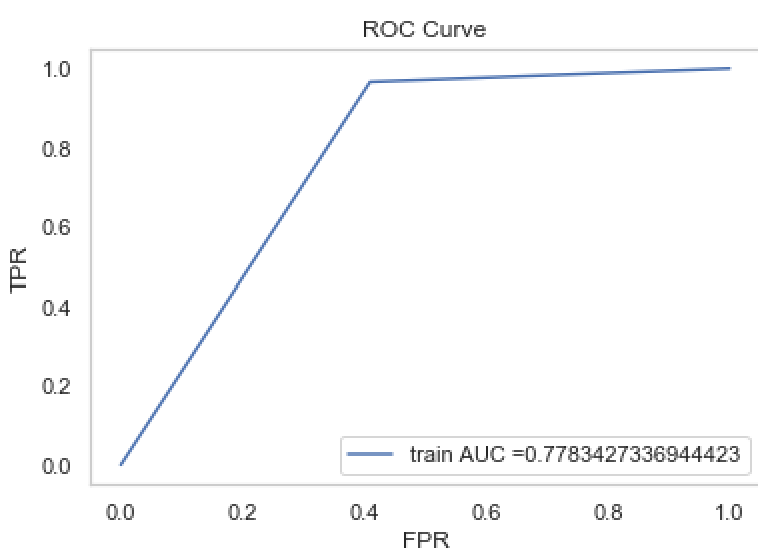
	2015	academy	accomplish	act	action	afford	anxious	assigned	becomes	cause	...
0	0.128325	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	...
1	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.241542	0.0	0.0	...
2	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	...
3	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	...
4	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	...

5 rows × 79 columns

In [40]:
```
model = DecisionTreeClassifier()
parameters = {'min_samples_split': [5,10,100,500]}

feat_imp_model = GridSearchCV(model_tfidf,parameters,n_jobs = -1)
feat_imp_model.fit(feat_imp_data,Y_train)

clf_tfidf.best_estimator_
```

Out[40]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=50,
                       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, presort=False, random_state=None,
                       splitter='best')
```

In [41]:
```
feat_imp_best_model = DecisionTreeClassifier(class_weight=None, criterion='gin
i',
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=100,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
feat_imp_best_model.fit(feat_imp_data,Y_train)

y_train_pred = feat_imp_best_model.predict(feat_imp_data)

train_fpr, train_tpr, tr_thresholds = roc_curve(Y_train, y_train_pred)

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



2. Summary

In [43]:

```
# 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', 50, 100, 0.7437],['Brute', 50, 500, 0.5411]]
# 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=F
alse)
plt.tick_params(axis='y', which='both', right=False, left=False, labelleft=Fal
se)
for pos in ['right','top','bottom','left']:
    plt.gca().spines[pos].set_visible(False)
```

	Model	Max Depth	Min Samples Split	AUC
TF IDF	Brute	50	100	0.7437
TF IDF W2V	Brute	50	500	0.5411