Assignment : DT

TF-IDFW2V

Tfidf w2v (w1,w2..) = (tfidf(w1) * w2v(w1) + tfidf(w2) * w2v(w2) + ...) / (tfidf(w1) + tfidf(w2) + ...)

(Optional) Please check course video on AVgw2V and TF-IDFW2V (https://www.appliedaicourse.com/lecture/11/applied-machine-learning-online-course/2916/avg-word2vec-tf-idf-weighted-word2vec/3/module-3-foundations-of-natural-language-processing-and-machine-learning)for more details.

Glove vectors

In this assignment you will be working with glove vectors , please check this (https://en.wikipedia.org/wiki/GloVe_(machine_learning)) and this (https://en.wikipedia.org/wiki/GloVe_(machine_learning)) for more details.

Download glove vectors from this link (https://drive.google.com/file/d/1lDca_ge-GYO0iQ6_XDLWePQFMdAA2b8f/view?usp=sharing)

In [1]:
```
#please use below code to load glove vectors
import pickle
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

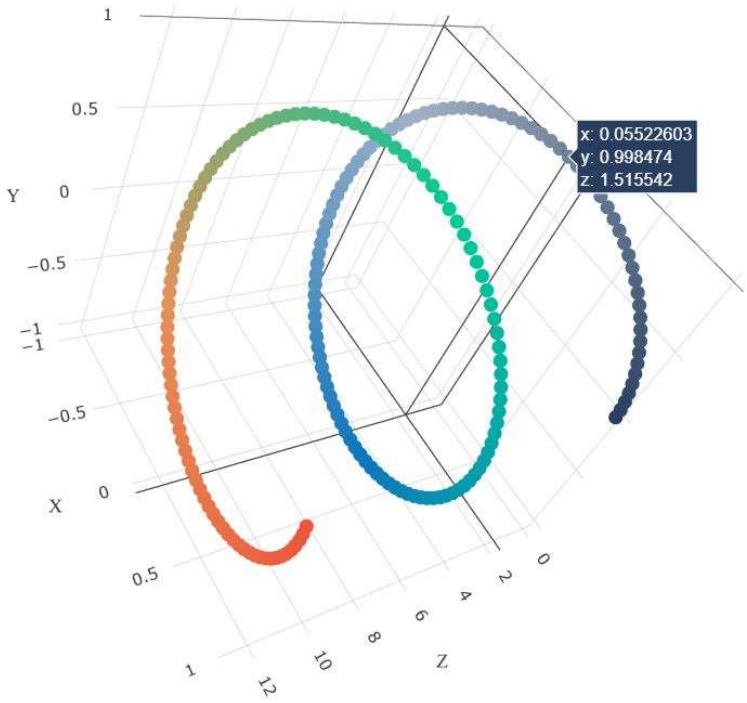
or else , you can use below code

In [2]:
```
'''
# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039
def loadGloveModel(gloveFile):
    print ("Loading Glove Model")
    f = open(gloveFile,'r', encoding="utf8")
    model = {}
    for line in tqdm(f):
        splitLine = line.split()
        word = splitLine[0]
        embedding = np.array([float(val) for val in splitLine[1:]])
        model[word] = embedding
    print ("Done.",len(model)," words loaded!")
    return model
model = loadGloveModel('glove.42B.300d.txt')

# ============================
Output:

Loading Glove Model
1917495it [06:32, 4879.69it/s]
Done. 1917495  words loaded!

# ============================

words = []
for i in preproced_texts:
    words.extend(i.split(' '))

for i in preproced_titles:
    words.extend(i.split(' '))
print("all the words in the coupus", len(words))
words = set(words)
print("the unique words in the coupus", len(words))

inter_words = set(model.keys()).intersection(words)
print("The number of words that are present in both glove vectors and our coupus"
      len(inter_words),"(",np.round(len(inter_words)/len(words)*100,3),"%)")

words_courpus = {}
words_glove = set(model.keys())
for i in words:
    if i in words_glove:
        words_courpus[i] = model[i]
print("word 2 vec length", len(words_courpus))


# stronging variables into pickle files python: http://www.jessicayung.com/how-to

import pickle
with open('glove_vectors', 'wb') as f:
    pickle.dump(words_courpus, f)


'''
```

Out[2]:
```
'\n# Reading glove vectors in python: https://stackoverflow.com/a/38230349/4084039\ndef (https://stackoverflow.com/a/38230349/4084039\ndef) loadGloveModel(gloveFile):\n    print ("Loading Glove Model")\n    f = open(gloveFile,\'r\', encoding="utf8")\n    model = {}\n    for line in tqdm(f):\n        splitLine = line.split()\n        word = splitLine[0]\n        embedding = np.array([float(val) for val in splitLine[1:]])\n        model[word] = embedding\n    print ("Done.",len(model)," words loaded!")\n    return model\nmodel = loadGloveModel(\'glove.42B.300d.txt\')\n\n# ============================\nOutput:\n\nLoading Glove Model\n1917495it [06:32, 4879.69it/s]\nDone. 1917495  words loaded!\n\n# ============================\n\nwords = []\nfor i in preproced_texts:\n    words.extend(i.split(\' \'))\n\nfor i in preproced_titles:\n    words.extend(i.split(\' \'))\nprint("all the words in the coupus", len(words))\nwords = set(words)\nprint("the unique words in the coupus", len(words))\n\ninter_words = set(model.keys()).intersection(words)\nprint("The number of words that are present in both glove vectors and our coupus",       len(inter_words),"(",np.round(len(inter_words)/len(words)*100,3),"%)")\n\nwords_courpus = {}\nwords_glove = set(model.keys())\nfor i in words:\n    if i in words_glove:\n        words_courpus[i] = model[i]\nprint("word 2 vec length", len(words_courpus))\n\n\n\n# stronging variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-python/\n\nimport (http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-python/\n\nimport) pickle\nwith open(\'glove_vectors\', \'wb\') as f:\n    pickle.dump(words_courpus, f)\n\n\n\n'
```

Task - 1

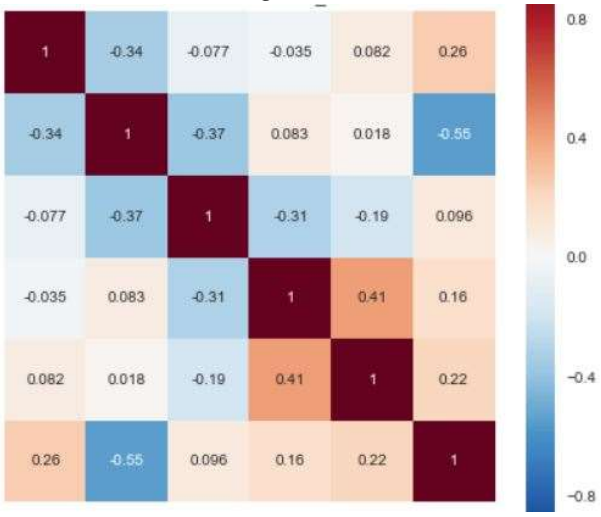
1. **Apply Decision Tree Classifier(DecisionTreeClassifier) on these feature sets**
    - Set 1: categorical, numerical features + preprocessed_essay (TFIDF) + Sentiment scores(preprocessed_essay)
    - Set 2: categorical, numerical features + preprocessed_essay (TFIDF W2V) + Sentiment scores(preprocessed_essay)
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/) 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*

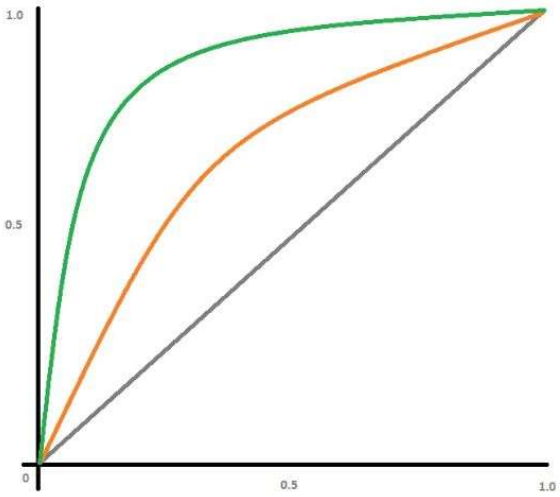
or

- 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) 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`

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 (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.

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

Importing relevant libraries

In [3]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

from sklearn.feature_extraction.text import TfidfVectorizer
from tqdm import tqdm
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import Normalizer
from chart_studio.plotly import plotly
import matplotlib.pyplot as plt
import plotly.offline as offline
import plotly.graph_objs as go
import seaborn as sns
from itertools import chain
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_curve, auc
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn.feature_selection import SelectFromModel
from sklearn.linear_model import LogisticRegression
from wordcloud import WordCloud
from prettytable import PrettyTable
```

Loading data

In [4]:
```
data=pd.read_csv('preprocessed_data.csv',nrows=50000)
```

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

Out[5]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projec
0	ca	mrs	grades_prek_2	
1	ut	ms	grades_3_5	
2	ca	mrs	grades_prek_2	
3	ga	mrs	grades_prek_2	
4	wa	mrs	grades_3_5	

Finding sentiment scores

In [6]:
```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

neg = []
pos = []
neu = []
compound = []

sid = SentimentIntensityAnalyzer()

for for_sentiment  in tqdm(data['essay']):

    neg.append(sid.polarity_scores(for_sentiment)['neg']) #Negative Sentiment sc
    pos.append(sid.polarity_scores(for_sentiment)['pos']) #Positive Sentiment sc
    neu.append(sid.polarity_scores(for_sentiment)['neu']) #Neutral Sentiment scor
    compound.append(sid.polarity_scores(for_sentiment)['compound']) #Compound Ser

# Creating new features
data['Essay_neg_ss']      = neg
data['Essay_pos_ss']      = pos
data['Essay_neu_ss']      = neu
data['Essay_compound_ss'] = compound
```

```
100%|██████████| 50000/50000 [16:52<00:00, 49.36it/s]
```

In [7]:
```
data.head(3)
```

Out[7]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projec
0	ca	mrs	grades_prek_2	
1	ut	ms	grades_3_5	
2	ca	mrs	grades_prek_2	

Splitting data into train, cv and test

In [8]:
```
y=data['project_is_approved']
X=data.drop(['project_is_approved'],axis=1)

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=0)
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.2,s
X_train.shape,X_test.shape,X_cv.shape,X_train.shape,y_cv.shape,y_test.shape
```

Out[8]:
```
((32000, 12), (10000, 12), (8000, 12), (32000, 12), (8000,), (10000,))
```

Vectorizing text features (essay)

In [9]:
```
## tfidf vectorizer for essay


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

essay_tfidf_train = vectorizer_essay_tfidf.transform(X_train['essay'])
essay_tfidf_test = vectorizer_essay_tfidf.transform(X_test['essay'])
essay_tfidf_cv = vectorizer_essay_tfidf.transform(X_cv['essay'])

print("Shape of train, test and CV are : ")
print(essay_tfidf_train.shape)
print(essay_tfidf_test.shape)
print(essay_tfidf_cv.shape)
```

```
Shape of train, test and CV are :
(32000, 10133)
(10000, 10133)
(8000, 10133)
```

In [10]:
```
## tfidf-w2vec
preprocessed_essays = X_train['essay'].values


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


## defining a function to calulate tfidf-w2v for train,test and cv

def tf_idf_weight_func(sentence): # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf valu
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    return vector

## calculating tfidf-w2v for each data

essay_tfidf_w2v_train = []
essay_tfidf_w2v_test  = []
essay_tfidf_w2v_cv    = []

for sentence in tqdm(X_train['essay']):
    essay_tfidf_w2v_train.append(tf_idf_weight_func(sentence)) #  TFIDF weighted


for sentence in tqdm(X_test['essay']):
    essay_tfidf_w2v_test.append(tf_idf_weight_func(sentence)) #  TFIDF weighted W



for sentence in tqdm(X_cv['essay']):
    essay_tfidf_w2v_cv.append(tf_idf_weight_func(sentence)) #  TFIDF weighted W2V


print("shape of essay_tfidf_w2v_train : ",len(essay_tfidf_w2v_train),'X',len(essa
print("shape of essay_tfidf_w2v_cv : ",len(essay_tfidf_w2v_cv),'X',len(essay_tfic
print("shape of essay_tfidf_w2v_test : ",len(essay_tfidf_w2v_test),'X',len(essay_
```

```
100%|██████████| 32000/32000 [02:45<00:00, 193.09it/s]
100%|██████████| 10000/10000 [00:52<00:00, 188.99it/s]
100%|██████████| 8000/8000 [00:45<00:00, 176.18it/s]

shape of essay_tfidf_w2v_train :  32000 X 300
shape of essay_tfidf_w2v_cv :  8000 X 300
```

```
shape of essay_tfidf_w2v_test :  10000 X 300
```

vectorizing categorical features

In [11]:

```
## vectorizing school_state
vectorizer_school_state= CountVectorizer(binary=True)
vectorizer_school_state.fit(X_train['school_state'].values)

X_train_state_ohe = vectorizer_school_state.transform(X_train['school_state'].va
X_cv_state_ohe = vectorizer_school_state.transform(X_cv['school_state'].values)
X_test_state_ohe = vectorizer_school_state.transform(X_test['school_state'].value

## vectorizing teachers prefix
vectorizer_teacher_prefix_cat= CountVectorizer(binary=True)
vectorizer_teacher_prefix_cat.fit(X_train['teacher_prefix'])

X_train_teacher_prefix_ohe=vectorizer_teacher_prefix_cat.transform(X_train['teach
X_cv_teacher_prefix_ohe=vectorizer_teacher_prefix_cat.transform(X_cv['teacher_pre
X_test_teacher_prefix_ohe=vectorizer_teacher_prefix_cat.transform(X_test['teacher

## vectorizing project_grade_category

vectorizer_project_grade_cat= CountVectorizer(binary=True)
vectorizer_project_grade_cat.fit(X_train['project_grade_category'])

X_train_project_grade_category_ohe=vectorizer_project_grade_cat.transform(X_train
X_cv_project_grade_category_ohe=vectorizer_project_grade_cat.transform(X_cv['proj
X_test_project_grade_category_ohe=vectorizer_project_grade_cat.transform(X_test['


## vectorizing clean_categories
vectorizer_categories= CountVectorizer(binary=True)
vectorizer_categories.fit(X_train['clean_categories'])

X_train_clean_categories_ohe=vectorizer_categories.transform(X_train['clean_categ
X_cv_clean_categories_ohe=vectorizer_categories.transform(X_cv['clean_categories'
X_test_clean_categories_ohe=vectorizer_categories.transform(X_test['clean_categor

## vectorizing clean_subcategories
vectorizer_sub_cat= CountVectorizer(binary=True)
vectorizer_sub_cat.fit(X_train['clean_subcategories'])

X_train_clean_subcategories_ohe=vectorizer_sub_cat.transform(X_train['clean_subca
X_cv_clean_subcategories_ohe=vectorizer_sub_cat.transform(X_cv['clean_subcategori
X_test_clean_subcategories_ohe=vectorizer_sub_cat.transform(X_test['clean_subcate
```

Normalizing numerical features

In [12]:

```
normalizer = Normalizer()

# normalizing price feature
normalizer.fit(X_train['price'].values.reshape(1,-1))

## after normalizing using reshape (1,-1) again reshaping back to (-1,1) for fixi
X_train_price_norm = normalizer.transform(X_train['price'].values.reshape(-1,1))
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(-1,1))
X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_price_norm.shape, y_train.shape)
print(X_test_price_norm.shape, y_test.shape)
print(X_cv_price_norm.shape, y_cv.shape)



## normalizing privously_submitted_project feature

normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.res

## after normalizing using reshape (1,-1) again reshaping back to (-1,1) for fixi
prev_no_projects_train = normalizer.transform(X_train['teacher_number_of_previous
prev_no_projects_cv = normalizer.transform(X_cv['teacher_number_of_previously_pos
prev_no_projects_test = normalizer.transform(X_test['teacher_number_of_previously

print("After vectorizations")
print(prev_no_projects_train.shape, y_train.shape)
print(prev_no_projects_cv.shape, y_cv.shape)
print(prev_no_projects_test.shape, y_test.shape)
```

```
After vectorizations
(32000, 1) (32000,)
(10000, 1) (10000,)
(8000, 1) (8000,)
After vectorizations
(32000, 1) (32000,)
(8000, 1) (8000,)
(10000, 1) (10000,)
```

Normalizing sentiment scores

In [13]:
```
## Vectorizing negative Sentiment Score

normalizer = Normalizer()
normalizer.fit(X_train['Essay_neg_ss'].values.reshape(1,-1))

essay_neg_train = normalizer.fit_transform(X_train['Essay_neg_ss'].values.reshape
essay_neg_test = normalizer.transform(X_test['Essay_neg_ss'].values.reshape(-1,1)
essay_neg_train = normalizer.fit_transform(X_train['Essay_neg_ss'].values.reshape
essay_neg_cv = normalizer.transform(X_cv['Essay_neg_ss'].values.reshape(-1,1))

print("Shape of train, test and CV of negative sentiment score vector : ")
print(essay_neg_train.shape)
print(essay_neg_test.shape)
print(essay_neg_cv.shape)
print("*"*100)



## Vectorizing positive Sentiment Score

normalizer = Normalizer()
normalizer.fit(X_train['Essay_pos_ss'].values.reshape(1,-1))

essay_pos_train = normalizer.fit_transform(X_train['Essay_pos_ss'].values.reshape
essay_pos_test = normalizer.transform(X_test['Essay_pos_ss'].values.reshape(-1,1)
essay_pos_train = normalizer.fit_transform(X_train['Essay_pos_ss'].values.reshape
essay_pos_cv = normalizer.transform(X_cv['Essay_pos_ss'].values.reshape(-1,1))

print("Shape of train, test and CV of positive sentiment score vector : ")
print(essay_pos_train.shape, y_train.shape)
print(essay_pos_test.shape, y_test.shape)
print(essay_pos_cv.shape,y_cv.shape)
print("*"*100)



## Vectorizing neutral Sentiment Score

normalizer = Normalizer()
normalizer.fit(X_train['Essay_neu_ss'].values.reshape(1,-1))

essay_neu_train = normalizer.fit_transform(X_train['Essay_neu_ss'].values.reshape

essay_neu_test = normalizer.transform(X_test['Essay_neu_ss'].values.reshape(-1,1)
essay_neu_train = normalizer.fit_transform(X_train['Essay_neu_ss'].values.reshape
essay_neu_cv = normalizer.transform(X_cv['Essay_neu_ss'].values.reshape(-1,1))

print("Shape of train, test and CV of neutral sentiment score vector : ")
print(essay_neu_train.shape)
print(essay_neu_test.shape)
print(essay_neu_cv.shape)
print("*"*100)



## Vectorizing compound Sentiment Score
```

```
normalizer = Normalizer()
normalizer.fit(X_train['Essay_compound_ss'].values.reshape(1,-1))

essay_compound_train = normalizer.fit_transform(X_train['Essay_compound_ss'].valu
essay_compound_test = normalizer.transform(X_test['Essay_compound_ss'].values.res
essay_compound_train = normalizer.fit_transform(X_train['Essay_compound_ss'].valu
essay_compound_cv = normalizer.transform(X_cv['Essay_compound_ss'].values.reshape

print("Shape of train, test and CV of compound sentiment score vector : ")
print(essay_compound_train.shape, y_train.shape)
print(essay_compound_test.shape, y_test.shape)
print(essay_compound_cv.shape,y_cv.shape)
print("*"*100)
```

```
Shape of train, test and CV of negative sentiment score vector :
(32000, 1)
(10000, 1)
(8000, 1)
****************************************************************************
********************
Shape of train, test and CV of positive sentiment score vector :
(32000, 1) (32000,)
(10000, 1) (10000,)
(8000, 1) (8000,)
****************************************************************************
********************
Shape of train, test and CV of neutral sentiment score vector :
(32000, 1)
(10000, 1)
(8000, 1)
****************************************************************************
********************
Shape of train, test and CV of compound sentiment score vector :
(32000, 1) (32000,)
(10000, 1) (10000,)
(8000, 1) (8000,)
****************************************************************************
********************
```

Forming SET 1 and SET 2

SET 1 : categorical, numerical features + preprocessed_essay (TFIDF) + Sentiment scores(preprocessed_essay)

In [14]:
```
from scipy.sparse import hstack

X_tr_1 = hstack((essay_tfidf_train,prev_no_projects_train,essay_neg_train,essay_p
                 X_train_state_ohe,X_train_teacher_prefix_ohe,X_train_project_gra
                 X_train_clean_categories_ohe,X_train_clean_subcategories_ohe,X_t


X_cv_1= hstack((essay_tfidf_cv,prev_no_projects_cv,essay_neg_cv,essay_pos_cv,essa
                X_cv_state_ohe,X_cv_teacher_prefix_ohe,\
                X_cv_project_grade_category_ohe,X_cv_clean_categories_ohe,X_cv_cl

X_te_1= hstack((essay_tfidf_test,prev_no_projects_test,essay_neg_test,essay_pos_t
                X_test_state_ohe,X_test_teacher_prefix_ohe,\
                X_test_project_grade_category_ohe,X_test_clean_categories_ohe,\
                X_test_clean_subcategories_ohe,X_test_price_norm)).tocsr()

print("Final Data matrix for set 1")
print(X_tr_1.shape, y_train.shape)
print(X_cv_1.shape, y_cv.shape)
print(X_te_1.shape, y_test.shape)
print("*"*100)
```

```
Final Data matrix for set 1
(32000, 10234) (32000,)
(8000, 10234) (8000,)
(10000, 10234) (10000,)
****************************************************************************
********************
```

SET 2 : categorical, numerical features + preprocessed_essay (TFIDF W2V) + Sentiment scores(preprocessed_essay)

In [15]:
```
X_tr_2 = hstack((essay_tfidf_w2v_train,prev_no_projects_train,essay_neg_train,ess
                 X_train_state_ohe,X_train_teacher_prefix_ohe,X_train_project_gra
                 X_train_clean_categories_ohe,X_train_clean_subcategories_ohe,X_t


X_cv_2= hstack((essay_tfidf_w2v_cv,prev_no_projects_cv,essay_neg_cv,essay_pos_cv,
                X_cv_state_ohe,X_cv_teacher_prefix_ohe,\
                X_cv_project_grade_category_ohe,X_cv_clean_categories_ohe,X_cv_cl

X_te_2= hstack((essay_tfidf_w2v_test,prev_no_projects_test,essay_neg_test,essay_p
                X_test_state_ohe,X_test_teacher_prefix_ohe,\
                X_test_project_grade_category_ohe,X_test_clean_categories_ohe,\
                X_test_clean_subcategories_ohe,X_test_price_norm)).tocsr()

print("Final Data matrix for set 2")
print(X_tr_1.shape, y_train.shape)
print(X_cv_1.shape, y_cv.shape)
print(X_te_1.shape, y_test.shape)
print("*"*100)
```

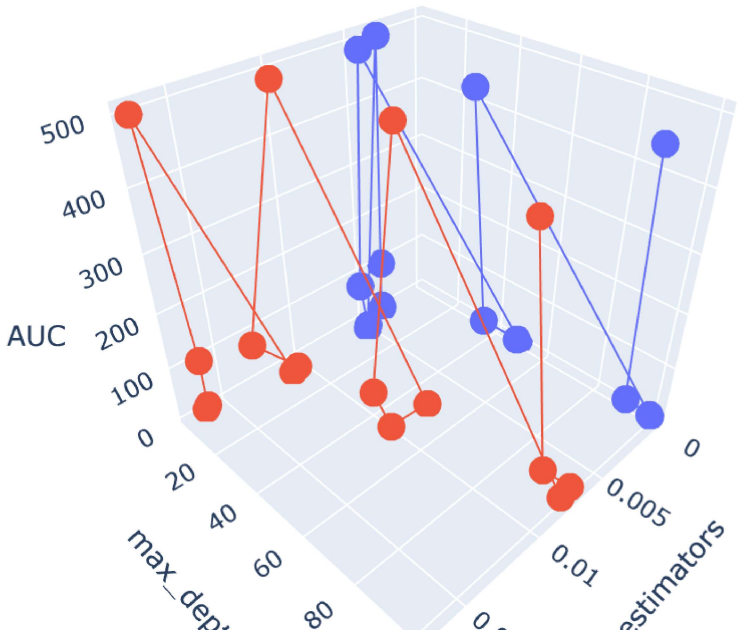
```
Final Data matrix for set 2
(32000, 10234) (32000,)
(8000, 10234) (8000,)
(10000, 10234) (10000,)
****************************************************************************************
********************
```

Applying Decision Tree on SET 1

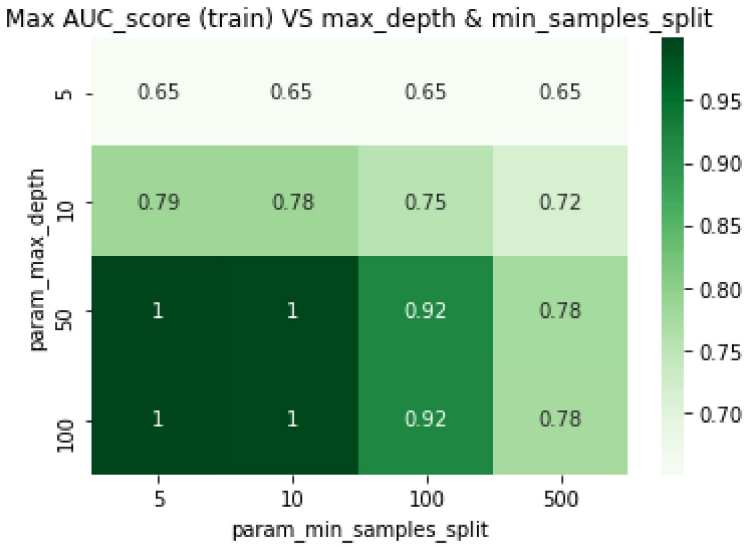
In [16]:
```
## defining batch_predict to precict values of y
## we will use this for AUC and confusion matrix

def batch_predict(clf, data):
    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])

    y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [17]:
```
## training our model
dt = DecisionTreeClassifier()
parameters = {'max_depth':[5, 10, 50, 100], 'min_samples_split': [5, 10, 100, 500

clf = GridSearchCV(dt, parameters, cv= 10, scoring='roc_auc',return_train_score=T
clf.fit(X_tr_2,y_train)

## obtaining values
train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
parm_max_depth = clf.cv_results_['param_max_depth']
param_min_samples_split = clf.cv_results_['param_min_samples_split']
```

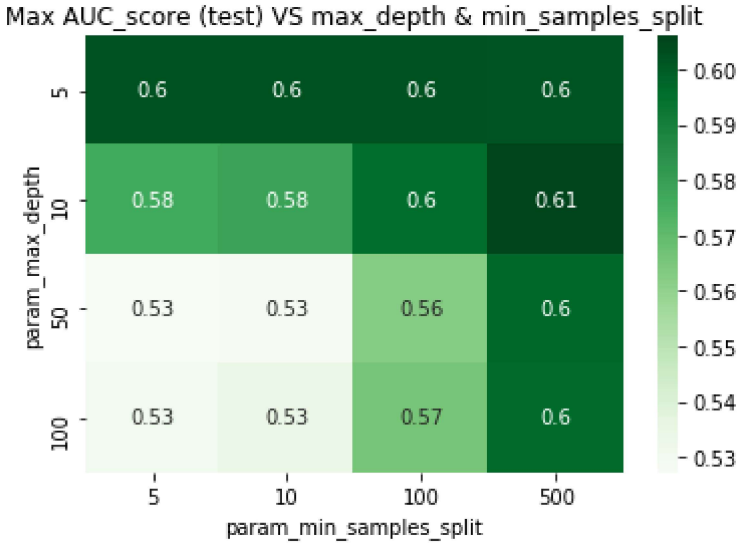
In [18]:
```
## Plotting 3D plot for AUC, maximum depth and minimum sample split

x1=train_auc_std
y1=parm_max_depth
z1=param_min_samples_split
x2=cv_auc_std
trace1 = go.Scatter3d(x=x1,y=y1,z=z1, name = 'train')
trace2 = go.Scatter3d(x=x2,y=y1,z=z1, name = 'Cross validation')
data = [trace1,trace2]

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

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



In [19]:
```
## plotting heatmap for AUC, maximum depth and minimum sample split for train dat
df_gridsearch = pd.DataFrame(clf.cv_results_)
max_scores = df_gridsearch.groupby(['param_max_depth','param_min_samples_split'])

sns.heatmap(max_scores.mean_train_score, annot=True,cmap="Greens")
plt.title('Max AUC_score (train) VS max_depth & min_samples_split')
plt.show()
```



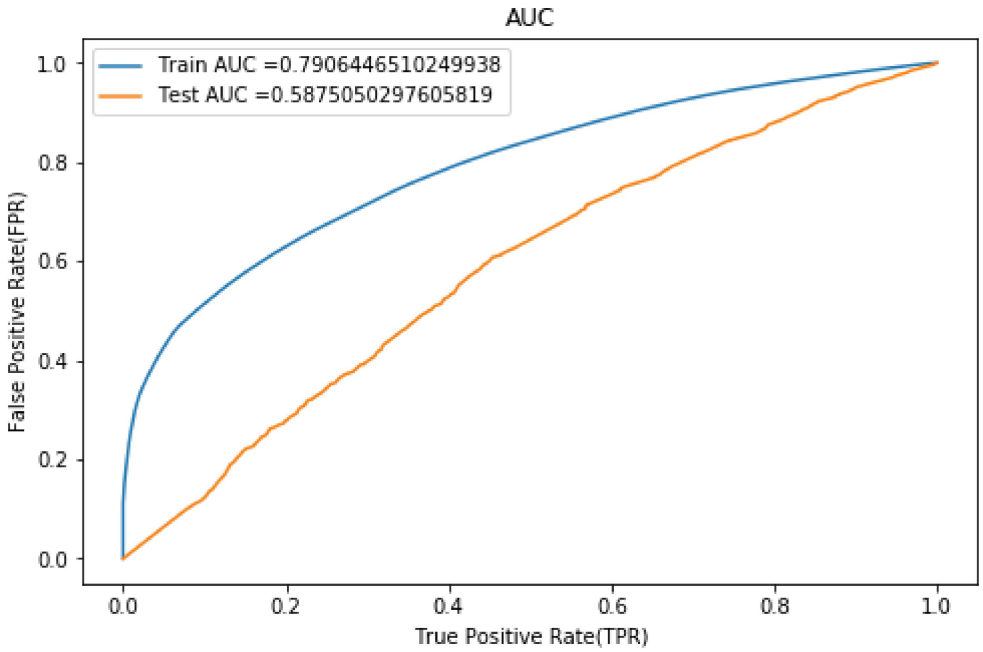
In [20]:
```
## plotting heatmap for AUC, maximum depth and minimum sample split for test data
sns.heatmap(max_scores.mean_test_score, annot=True,cmap="Greens")
plt.title('Max AUC_score (test) VS max_depth & min_samples_split')
plt.show()
```



Observation

1. From the above plots we can see that optimal value for depth of tree is 10
2. Also the minimum sample split is 100

In [21]:
```
## training model on best value of parameters and finding ROC and AUC

dt = DecisionTreeClassifier(max_depth = 10, min_samples_split = 100,class_weight=

clf = dt.fit(X_tr_2, y_train)

y_train_pred = dt.predict_proba(X_tr_2)[:,1]
y_test_pred = dt.predict_proba(X_te_2)[:,1]

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.figure(figsize=(8,5))
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("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.show()
```



In [22]:
```
## Finding Confusion Matrix
def predict(proba, threshould, fpr, tpr):

    t = threshould[np.argmax(tpr*(1-fpr))]

    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold",
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions

conf_mat_train = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_
sns.set(font_scale=1.4)
sns.heatmap(conf_mat_train, annot=True,annot_kws={"size": 16}, fmt='g',cmap="Gree
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
plt.show()
```

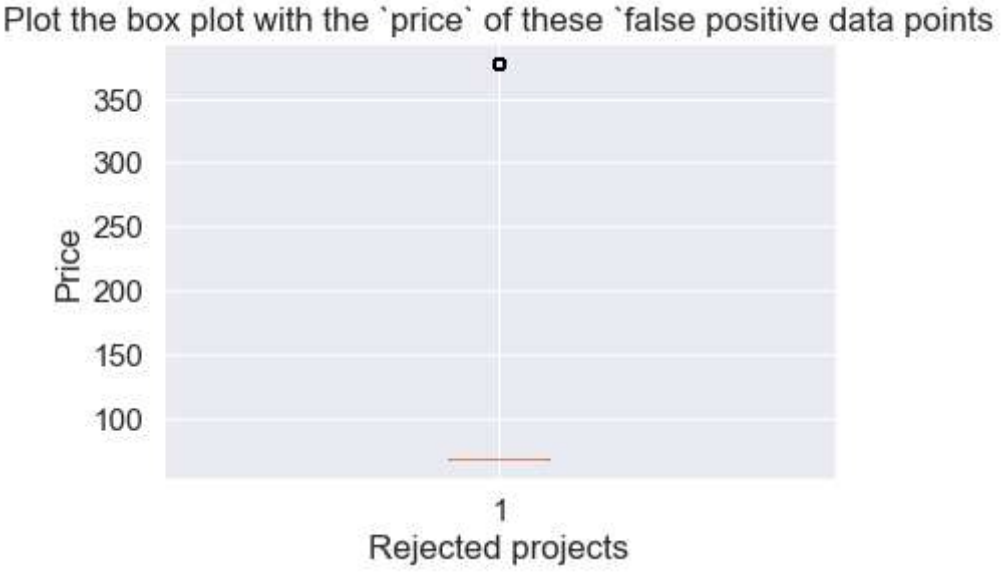
```
the maximum value of tpr*(1-fpr) 0.5064686659279597 for threshold 0.463
```

In [23]:
```
## obtaining all feature names to plot word cloud
a = vectorizer_categories.get_feature_names()
b = vectorizer_sub_cat.get_feature_names()
c = vectorizer_school_state.get_feature_names()
d = vectorizer_project_grade_cat.get_feature_names()
e = vectorizer_teacher_prefix_cat.get_feature_names()
g = vectorizer_essay_tfidf.get_feature_names()

feature_names_ = list(chain(a,b,c,d,e,['Price','Prec_no_projts','Essay_neg_ss','E
```

In [24]:
```
## finding false positive rates
fp_rows = []
y_train_label = []


for i in range(len(y_train)):
    if (y_train_pred[i] >= 0.562):

        y_train_label.append(1)
    else:
        y_train_label.append(0)
for i,j in zip(y_train,y_train_label):
    if i==j:
        fp_rows.append(i)

tp_freq = {}

df_ = pd.DataFrame(X_tr_1.todense())
df_fp = df_.iloc[fp_rows,:]
df_fp.columns = feature_names_
tp_freq = (df_fp.sum()).to_dict()
```

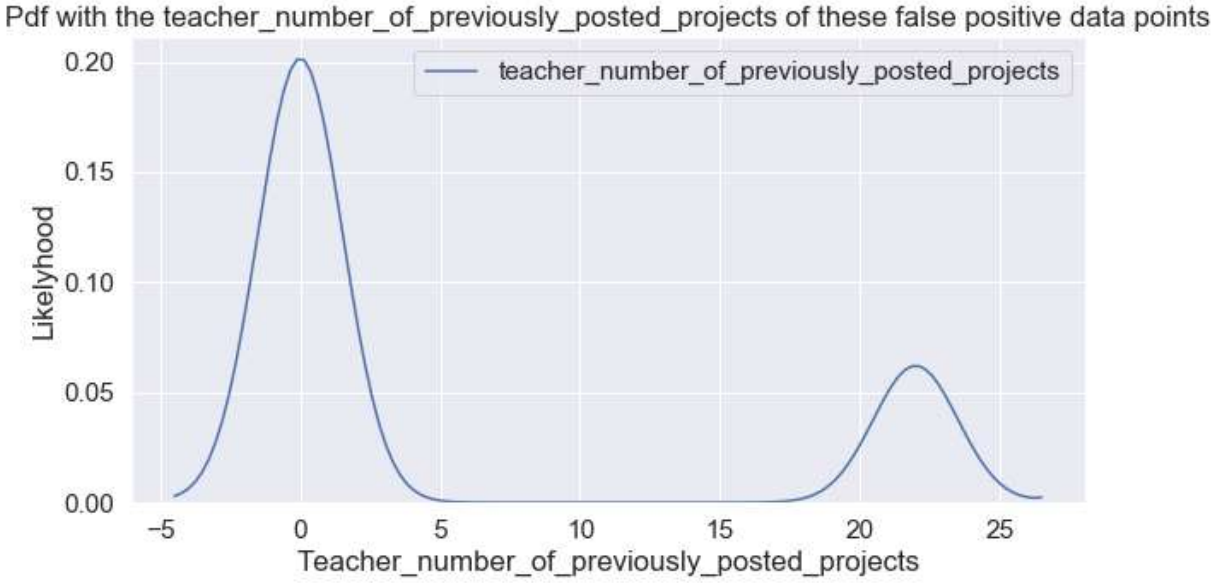
In [25]:
```
## Ploting word cloud
wordcloud = WordCloud(width = 1000, height = 500, background_color ='white').gene
plt.figure(figsize=(25,10))
plt.imshow(wordcloud)
plt.axis("off")
plt.show()
plt.close()
```

In [26]:
```
## Box plot for price and FPR
plt.boxplot(X_train['price'].iloc[fp_rows])
plt.title('Plot the box plot with the `price` of these `false positive data point
plt.xlabel('Rejected projects')
plt.ylabel('Price')
plt.grid(True)
plt.show()
```



In [27]:
```
ser_test=X_train['teacher_number_of_previously_posted_projects'].iloc[fp_rows]
```

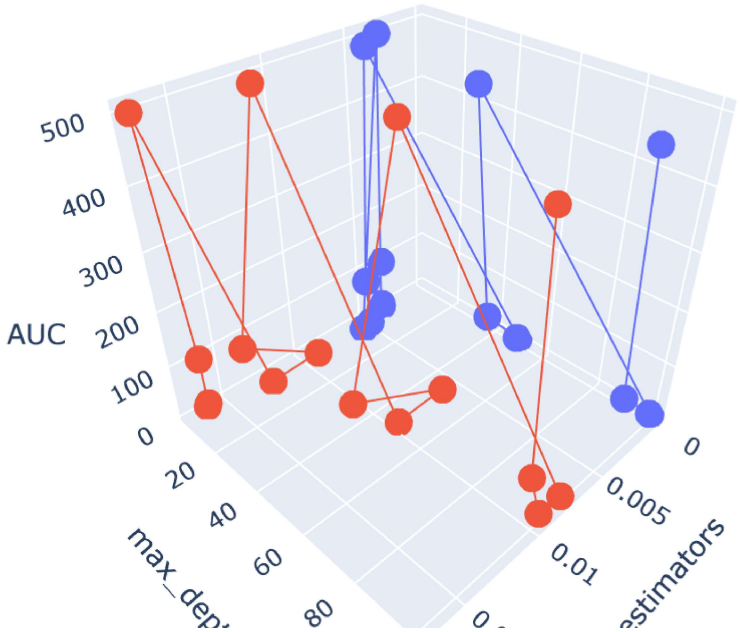
In [28]:
```
## plotting pdf for teacher previously submitted number of projects and FPR
plt.figure(figsize=(10,5))
sns.kdeplot(ser_test, cumulative=False, bw=1.5)
plt.title('Pdf with the teacher_number_of_previously_posted_projects of these fal
plt.xlabel('Teacher_number_of_previously_posted_projects')
plt.ylabel('Likelyhood')
plt.legend()
plt.show()
```



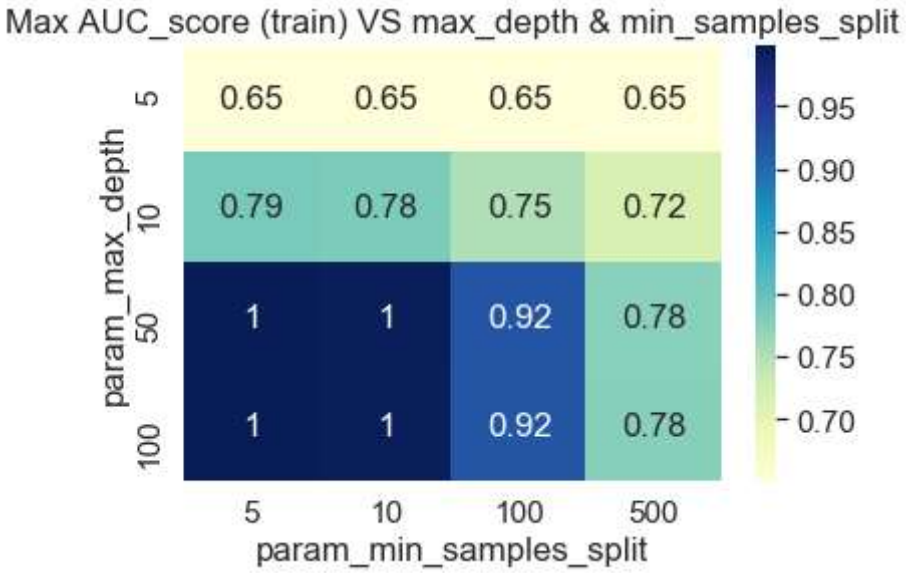
Applying Decision Tree on SET 2 dataset

In [29]:
```
## training data and fitting model
dt = DecisionTreeClassifier()
parameters = {'max_depth':[5, 10, 50, 100], 'min_samples_split': [5, 10, 100, 500


clf = GridSearchCV(dt, parameters, cv= 10, scoring='roc_auc',return_train_score=T
clf.fit(X_tr_2,y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
parm_max_depth = clf.cv_results_['param_max_depth']
param_min_samples_split = clf.cv_results_['param_min_samples_split']
```

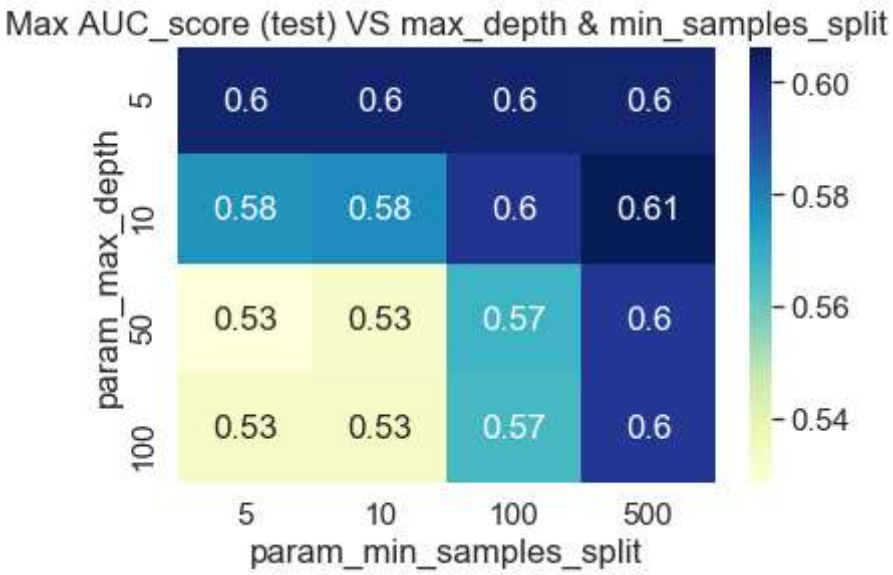
In [30]:
```
## plotting 3d plot as in case of set 1
x1=train_auc_std
y1=parm_max_depth
z1=param_min_samples_split
x2=cv_auc_std
trace1 = go.Scatter3d(x=x1,y=y1,z=z1, name = 'train')
trace2 = go.Scatter3d(x=x2,y=y1,z=z1, name = 'Cross validation')
data = [trace1,trace2]

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

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



In [31]:
```
## Plotting heatmap for train
df_gridsearch = pd.DataFrame(clf.cv_results_)
max_scores = df_gridsearch.groupby(['param_max_depth','param_min_samples_split'])

sns.heatmap(max_scores.mean_train_score, annot=True, cmap="YlGnBu")
plt.title('Max AUC_score (train) VS max_depth & min_samples_split')
plt.show()
```



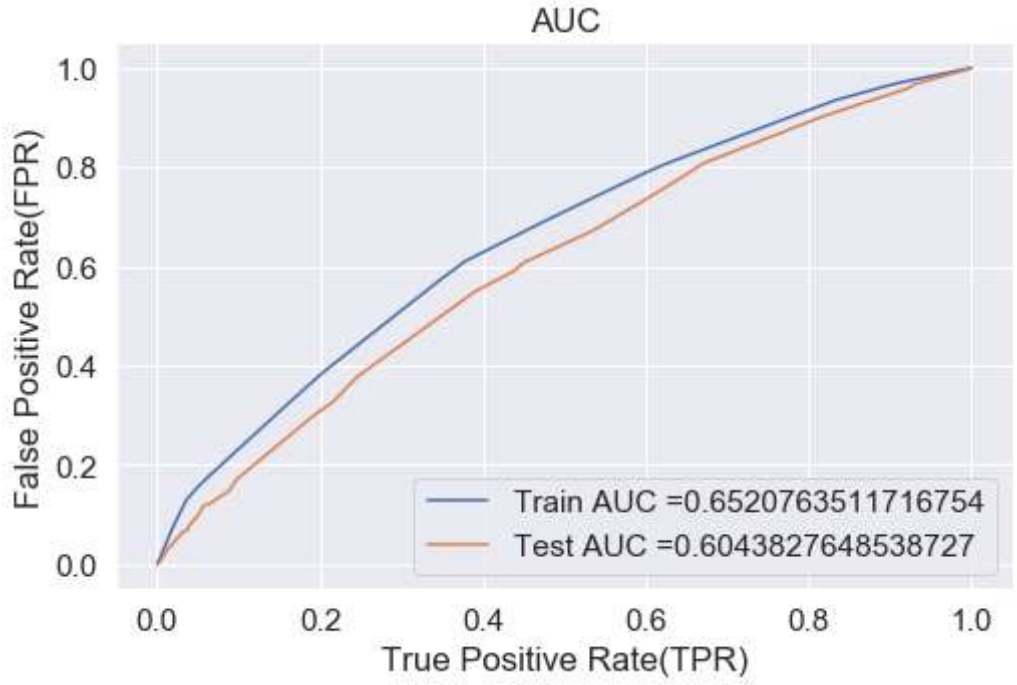
In [32]:
```
## plotting heatmap for test
sns.heatmap(max_scores.mean_test_score, annot=True,cmap="YlGnBu")
plt.title('Max AUC_score (test) VS max_depth & min_samples_split')
plt.show()
```



Observation

1. From the above plots we can see that optimal value for depth of tree is 5
2. Also the minimum sample split is 100

In [33]:
```
## training model on best value of parameters and finding ROC and AUC

dt = DecisionTreeClassifier(max_depth = 5, min_samples_split = 100,class_weight='

clf = dt.fit(X_tr_2, y_train)

y_train_pred = dt.predict_proba(X_tr_2)[:,1]
y_test_pred = dt.predict_proba(X_te_2)[:,1]

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.figure(figsize=(8,5))
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("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.show()
```



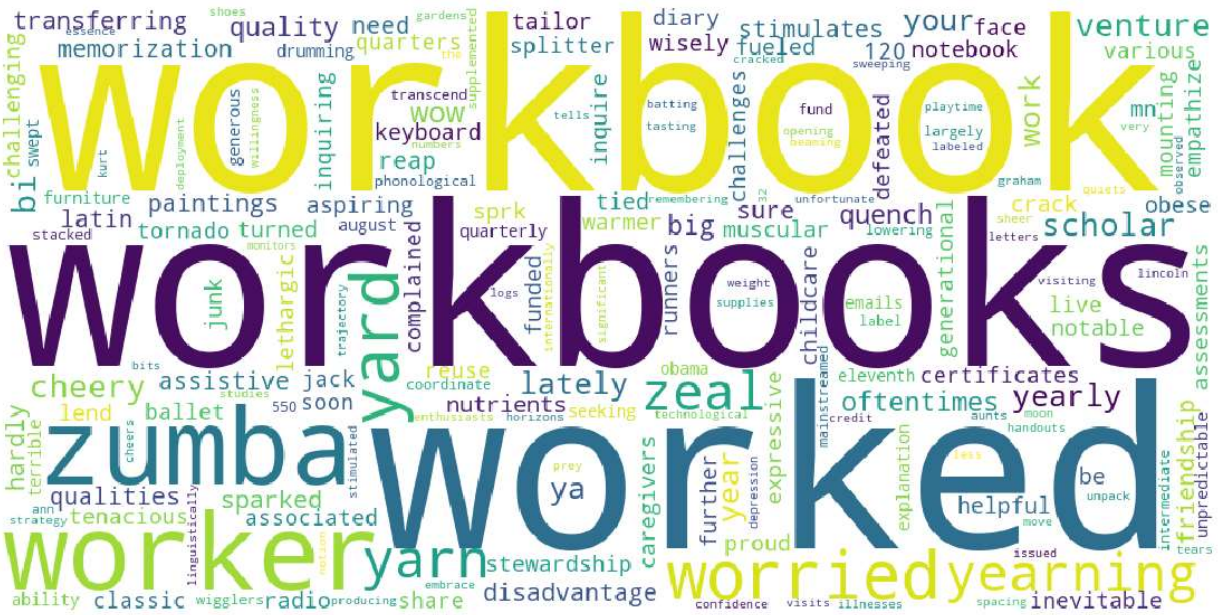
In [34]:
```
## Finding Confusion Matrix
def predict(proba, threshould, fpr, tpr):

    t = threshould[np.argmax(tpr*(1-fpr))]

    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold",
    predictions = []
    for i in proba:
        if i>=t:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions

conf_mat_train = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_
sns.set(font_scale=1.4)
sns.heatmap(conf_mat_train, annot=True,annot_kws={"size": 16}, fmt='g',cmap="YlGr
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.37961190304883796 for threshold 0.551
```

In [35]:
```
## getting name of all features

a = vectorizer_categories.get_feature_names()
b = vectorizer_sub_cat.get_feature_names()
c = vectorizer_school_state.get_feature_names()
d = vectorizer_project_grade_cat.get_feature_names()
e = vectorizer_teacher_prefix_cat.get_feature_names()
g = vectorizer_essay_tfidf.get_feature_names()

from itertools import chain

feature_names_ = list(chain(a,b,c,d,e,['Price','Prec_no_projts','Essay_neg_ss','E
```

In [36]:
```
## finding false positive rate
fp_rows = []
y_train_label = []


for i in range(len(y_train)):
    if (y_train_pred[i] >= 0.562):

        y_train_label.append(1)
    else:
        y_train_label.append(0)
for i,j in zip(y_train,y_train_label):
    if i==j:
        fp_rows.append(i)

tp_freq = {}

df_ = pd.DataFrame(X_tr_1.todense())
df_fp = df_.iloc[fp_rows,:]
df_fp.columns = feature_names_
tp_freq = (df_fp.sum()).to_dict()
```

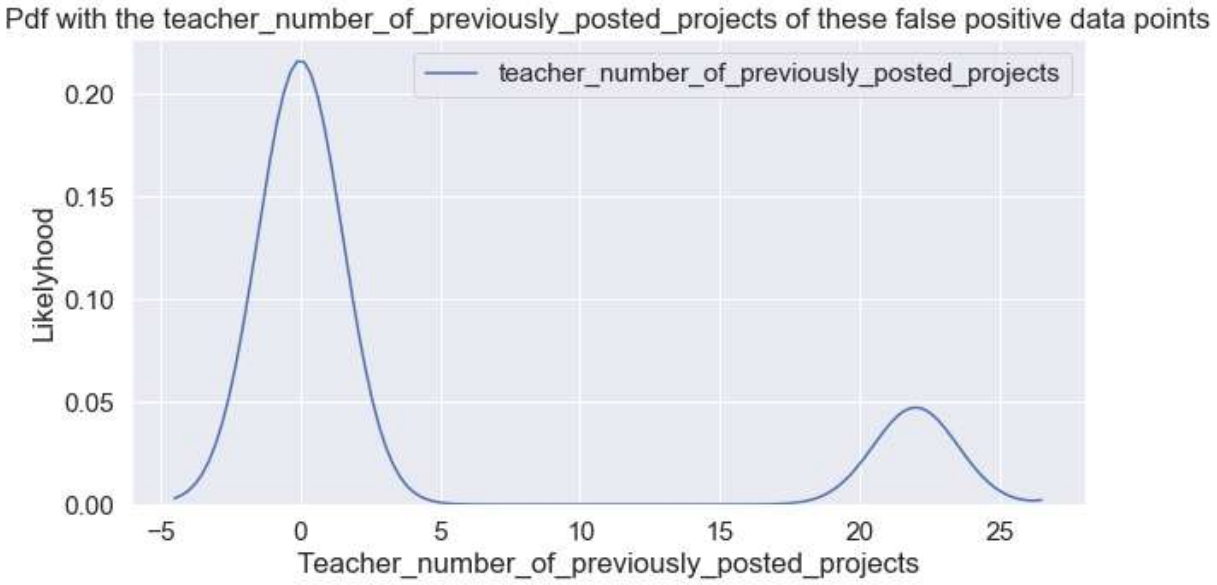
In [37]:
```
wordcloud = WordCloud(width = 1000, height = 500, background_color ='white').gene
plt.figure(figsize=(25,10))
plt.imshow(wordcloud)
plt.axis("off")
plt.show()
plt.close()
```



In [38]:
```
## Box plot
plt.boxplot(X_train['price'].iloc[fp_rows])
plt.title('Plot the box plot with the `price` of these `false positive data point
plt.xlabel('Rejected projects')
plt.ylabel('Price')
plt.grid(True)
plt.show()
```

In [39]:
```
ser_test=X_train['teacher_number_of_previously_posted_projects'].iloc[fp_rows]
```

In [40]:
```
plt.figure(figsize=(10,5))
sns.kdeplot(ser_test, cumulative=False, bw=1.5)
plt.title('Pdf with the teacher_number_of_previously_posted_projects of these fal
plt.xlabel('Teacher_number_of_previously_posted_projects')
plt.ylabel('Likelyhood')
plt.legend()
plt.show()
```



Selecting features which have non zero feature importance

In [41]:
```
# using Decision Tree to train data with selected features and also for selecting

# using features of set 1

print('Original number of features in dataset ')
X_tr_1.shape,X_te_1.shape,X_cv_1.shape
```

```
Original number of features in dataset
```

Out[41]:
```
((32000, 10234), (10000, 10234), (8000, 10234))
```

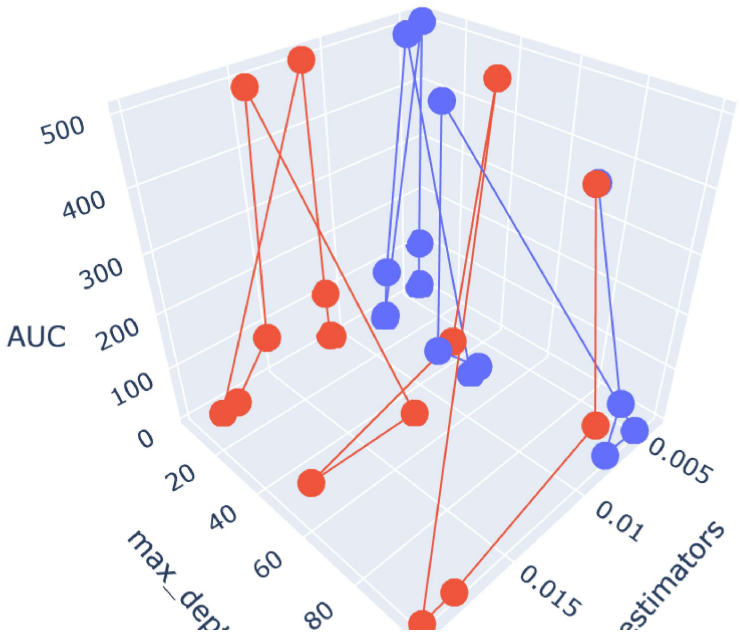
In [42]:
```
## selecting features

sel_ = SelectFromModel(DecisionTreeClassifier())

sel_.fit(X_tr_1,y_train)

X_train_selected = sel_.transform(X_tr_1)
X_test_selected = sel_.transform(X_te_1)
X_cv_selected=sel_.transform(X_cv_1)
print('After removing features with non zero importance shape of dataset ')
X_train_selected.shape, X_test_selected.shape,X_cv_selected.shape
```

```
After removing features with non zero importance shape of dataset
```

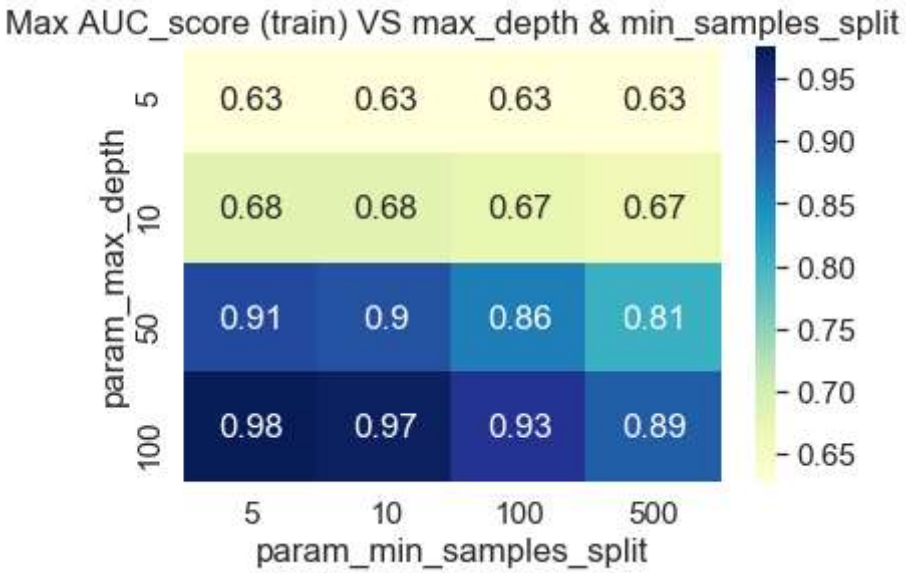
Out[42]:
```
((32000, 1883), (10000, 1883), (8000, 1883))
```

In [43]:
```
## training model
dt = DecisionTreeClassifier()
parameters = {'max_depth':[ 5, 10, 50, 100], 'min_samples_split': [5, 10, 100, 50


clf = GridSearchCV(dt, parameters, cv= 10, scoring='roc_auc',return_train_score=T
clf.fit(X_train_selected,y_train)

train_auc= clf.cv_results_['mean_train_score']
train_auc_std= clf.cv_results_['std_train_score']
cv_auc = clf.cv_results_['mean_test_score']
cv_auc_std= clf.cv_results_['std_test_score']
parm_max_depth = clf.cv_results_['param_max_depth']
param_min_samples_split = clf.cv_results_['param_min_samples_split']
```

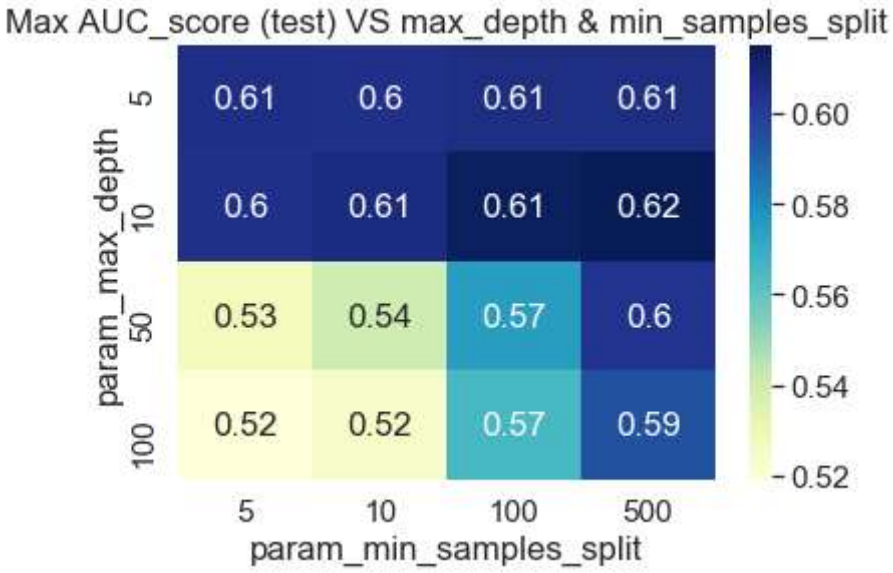
In [44]:
```
x1=train_auc_std
y1=parm_max_depth
z1=param_min_samples_split
x2=cv_auc_std
trace1 = go.Scatter3d(x=x1,y=y1,z=z1, name = 'train')
trace2 = go.Scatter3d(x=x2,y=y1,z=z1, name = 'Cross validation')
data = [trace1,trace2]

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

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



In [45]:
```
df_gridsearch = pd.DataFrame(clf.cv_results_)
max_scores = df_gridsearch.groupby(['param_max_depth','param_min_samples_split'])

sns.heatmap(max_scores.mean_train_score, annot=True, cmap="YlGnBu")
plt.title('Max AUC_score (train) VS max_depth & min_samples_split')
plt.show()
```



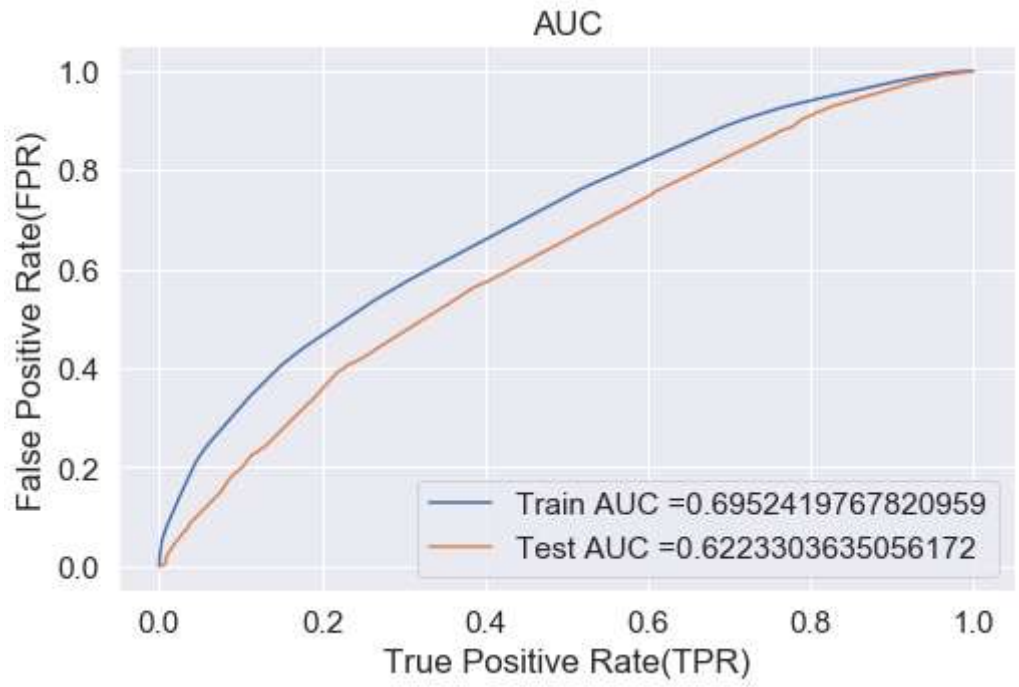
In [46]:
```
sns.heatmap(max_scores.mean_test_score, annot=True,cmap="YlGnBu")
plt.title('Max AUC_score (test) VS max_depth & min_samples_split')
plt.show()
```



Observation

1. Optimal depth of tree is 10
2. Optimal value for sample split is 100

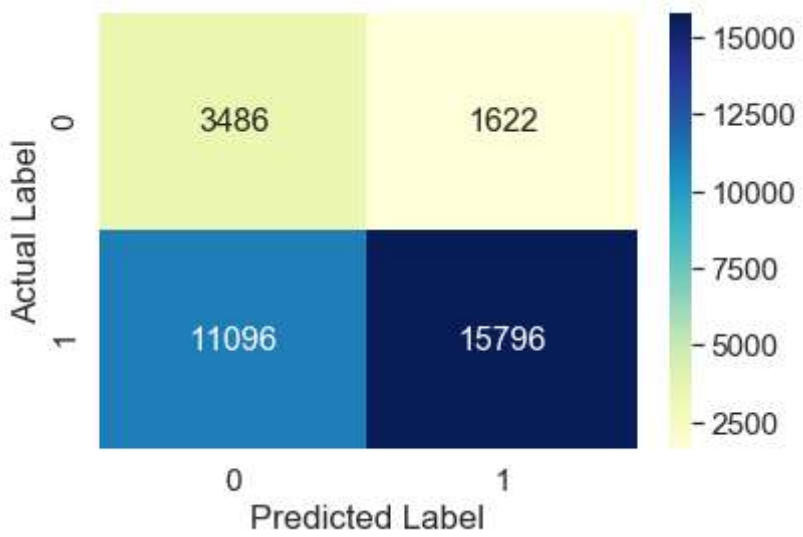
In [47]:
```
## training model on best value of parameters and finding ROC and AUC

dt = DecisionTreeClassifier(max_depth = 10, min_samples_split = 500,class_weight=

clf = dt.fit(X_train_selected, y_train)

y_train_pred = dt.predict_proba(X_train_selected)[:,1]
y_test_pred = dt.predict_proba(X_test_selected)[:,1]

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.figure(figsize=(8,5))
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("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.show()
```



In [48]:
```
## Confusion matrix
conf_mat_train = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_
```

```
the maximum value of tpr*(1-fpr) 0.4008671945242031 for threshold 0.478
```

In [49]:
```
sns.set(font_scale=1.4)
sns.heatmap(conf_mat_train, annot=True,annot_kws={"size": 16}, fmt='g',cmap="YlGr
plt.xlabel("Predicted Label")
plt.ylabel("Actual Label")
plt.show()
```



Summary

In [51]:
```
x_pretty_table = PrettyTable()
x_pretty_table.field_names = ["Model Type","Vectorizer","max_depth","min_sample_s

x_pretty_table.add_row(["Decision Tree","TFIDF",10,500,0.79,0.58])
x_pretty_table.add_row([ "Decision Tree","TFIDF-W2V",50,500,0.60, 0.65])
x_pretty_table.add_row([ "Decision Tree : Non zero Features","TFIDF",50,500,0.69,

print(x_pretty_table)
```

```
+-----------------------------------+------------+-----------+-----------------
-+-----------+-----------+
|             Model Type            | Vectorizer | max_depth | min_sample_split
| Train-AUC | Test-AUC |
+-----------------------------------+------------+-----------+-----------------
-+-----------+-----------+
|           Decision Tree           |   TFIDF    |     10    |       500
|    0.79   |   0.58    |
|           Decision Tree           | TFIDF-W2V  |     50    |       500
|    0.6    |   0.65    |
| Decision Tree : Non zero Features |   TFIDF    |     50    |       500
|    0.69   |   0.62    |
+-----------------------------------+------------+-----------+-----------------
-+-----------+-----------+
```