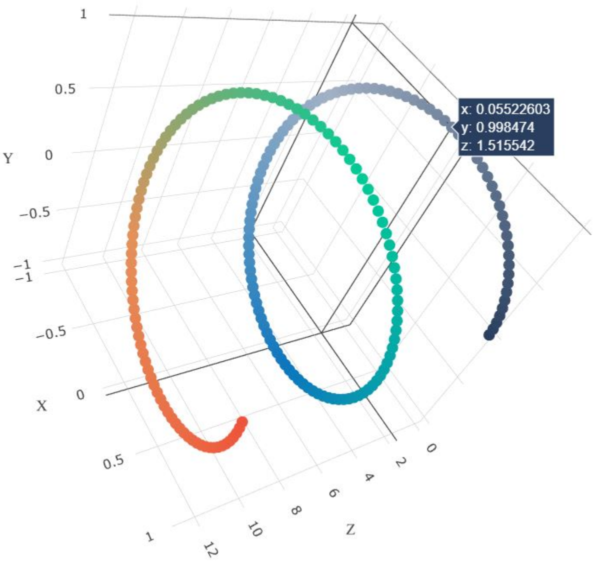
Assignment 8: DT

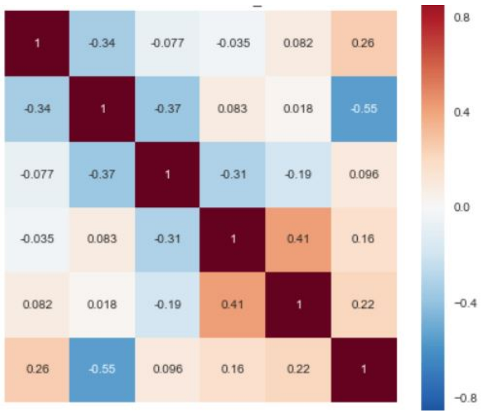
1. **Apply Decision Tree Classifier(DecisionTreeClassifier) on these feature sets**
    - Set 1: categorical, numerical features + preprocessed_eassay (TFIDF)
    - Set 2: categorical, numerical features + preprocessed_eassay (TFIDF W2V)
2. **The hyper paramter tuning (best `depth` in range [1, 5, 10, 50], and the best `min_samples_split` in range [5, 10, 100, 500])**
    - Find the best hyper parameter which will give the maximum AUC 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*

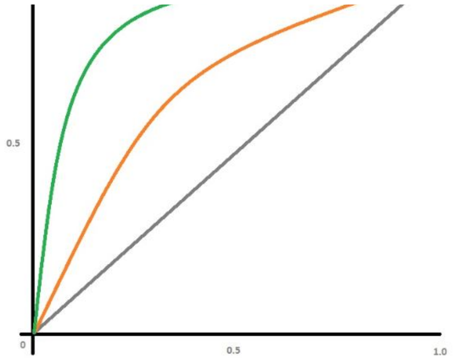
or

- 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

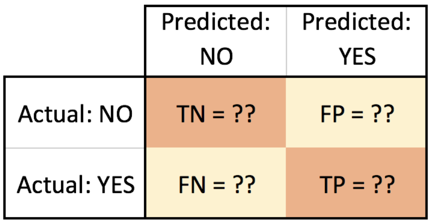


seaborn heat maps with rows as **n_estimators**, columns as **max_depth**, and values inside the cell representing **AUC Score**

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



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

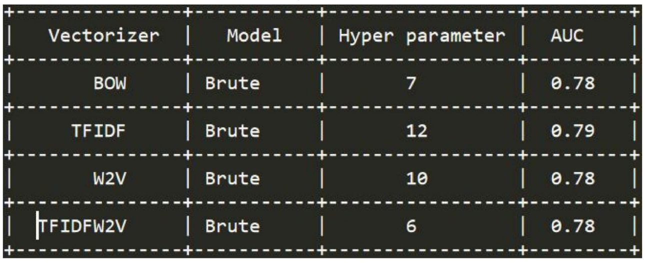


	Predicted: NO	Predicted: YES
Actual: NO	TN = ??	FP = ??
Actual: YES	FN = ??	TP = ??

- 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  |
+--------------+----------+------------------+--------+
|    TFIDF     | Brute    |        12        |  0.79  |
+--------------+----------+------------------+--------+
|     W2V      | Brute    |        10        |  0.78  |
+--------------+----------+------------------+--------+
|  TFIDFW2V    | Brute    |        6         |  0.78  |
+--------------+----------+------------------+--------+
```

1. Decision Tree

1.1 Loading Data

In [1]:

```
##from google.colab import drive
##drive.mount('/content/gdrive')
```

Import Important Libraries & Read Input Data

In [2]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import re
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
from tqdm import tqdm
import pickle
from sklearn.preprocessing import Normalizer
from tqdm import tqdm
import nltk
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('words')
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from tqdm.notebook import tqdm
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning:
pandas.util.testing is deprecated. Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
[nltk_data] Downloading package words to /root/nltk_data...
[nltk_data]   Unzipping corpora/words.zip.
```

In [3]:

```
data_df = pd.read_csv('/content/gdrive/My Drive/9_Donors_choose_DT/preprocessed_data.csv')
print(data_df.shape)
data_df.head()
```

```
(109248, 9)
```

Out[3]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved	clean_categ
0	ca	mrs	grades_prek_2	53	1	math_s
1	ut	ms	grades_3_5	4	1	special
2	ca	mrs	grades_prek_2	10	1	literacy_lan
3	ga	mrs	grades_prek_2	2	1	appliedle
4	wa	mrs	grades_3_5	2	1	literacy_lan

StopWord removal and Lemmatizion of the Essay Features.

In [0]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",
\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under'
, 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "do
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [0]:

```
# https://stackoverflow.com/a/47091490/4084039
def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

In [6]:

```
# Combining all the above stundents
preprocessed_essays = []
# tqdm is for printing the status bar
for sentance in tqdm(data_df['essay'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_essays.append(sent.lower().strip())
```

In [0]:

```
data_df['essay'] = preprocessed_essays
```

In [8]:

```
%%time
### Essay Preprocessing
def lem_stop_word(essay_str):

    from textblob import TextBlob

    lemmatizer = WordNetLemmatizer()
    tokens = word_tokenize(essay_str)
    new_token = []
    for word in tokens:
          word_x = lemmatizer.lemmatize(word,pos='v')
          new_token.append(lemmatizer.lemmatize(word_x,pos='a'))

    words = " ".join(new_token)
    blob = TextBlob(words)
    sin_token_word  = [word.singularize() for word in blob.words]
    return (" ".join(sin_token_word))


data_df['essay'] = data_df['essay'].apply(lem_stop_word)
```

```
CPU times: user 15min 39s, sys: 358 ms, total: 15min 40s
Wall time: 15min 40s
```

In [0]:

```
def number_of_word_per_sentence(data,col='essay'):

  total_word = []
  total_sent = []
  for text_corpus in data['essay']:
    total_word_cnt = 0
    total_sent_cnt = 0
    for sent in text_corpus.split('.'):
      total_sent_cnt+=1
      total_word_cnt +=len(sent.split(' '))

    total_word.append(total_word_cnt)
    total_sent.append(total_sent_cnt)

  data['Total_Word'] = total_word
  return data
```

In [0]:

```
data_df = number_of_word_per_sentence(data_df,'essay')
```

Add new feature respect to polarity and subjectivity

In [11]:

```
nltk.download('vader_lexicon')
from nltk.sentiment.vader import SentimentIntensityAnalyzer
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
```

```
/usr/local/lib/python3.6/dist-packages/nltk/twitter/__init__.py:20: UserWarning:

The twython library has not been installed. Some functionality from the twitter package will not b
e available.
```

In [0]:

```
negative = []
positive = []
neutral = []
compound = []

def sentimental_analyzer(_analyzer):
    negative.append(_analyzer["neg"])
    positive.append(_analyzer["pos"])
    neutral.append(_analyzer["neu"])
    compound.append(_analyzer["compound"])
```

In [13]:

```
analyzer = SentimentIntensityAnalyzer()
for essay in tqdm(data_df['essay']):
  sentimental_analyzer(analyzer.polarity_scores(essay))
```

In [14]:

```
from textblob import TextBlob
polarity = []
subjectivity = []
for essay in tqdm(data_df['essay']):
  testimonial = TextBlob(essay)
  polarity.append(testimonial.polarity)
  subjectivity.append(testimonial.subjectivity)
```

In [0]:

```
data_df['negative'] = negative
data_df["positive"] = positive
data_df['neutral'] = neutral
data_df['compound'] = compound
data_df['polarity'] = polarity
data_df['subjectivity'] = subjectivity
```

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

In [16]:

```
# please write all the code with proper documentation, and proper titles for each subsection
# go through documentations and blogs before you start coding
# first figure out what to do, and then think about how to do.
# reading and understanding error messages will be very much helpfull in debugging your code
# when you plot any graph make sure you use
    # a. Title, that describes your plot, this will be very helpful to the reader
    # b. Legends if needed
    # c. X-axis label
    # d. Y-axis label
## Import required library
from sklearn.model_selection import train_test_split

## create dependent and independent varaible
y_df = data_df['project_is_approved'].values
X_df = data_df.drop('project_is_approved',axis=1)

## create train,test split
X_tr,X_test,y_tr,y_test = train_test_split(X_df, y_df, test_size=0.33, random_state=42,stratify=y_d
f)
print(X_tr.shape, y_tr.shape)
print(X_test.shape, y_test.shape)
```

```
(73196, 15) (73196,)
(36052, 15) (36052,)
```

1.4 Make Data Model Ready: encoding eassay, and project_title

TFIDF Featurization

In [0]:

```
def TFIDF_essays_sub(X_train, y_train, X_test, y_test, col_name = 'essay'):

    vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4),max_features=300)
    vectorizer.fit(X_train[col_name].values) # fit has to happen only on train data

    # we use the fitted CountVectorizer to convert the text to vector
    X_train_ohe =   vectorizer.transform(X_train[col_name].values)
    X_test_ohe = vectorizer.transform(X_test[col_name].values)

    print("After vectorizations {}".format(col_name))
    print(X_train_ohe.shape, y_train.shape)
    print(X_test_ohe.shape, y_test.shape)
    #print(vectorizer.get_feature_names())
    print("="*100)

    return X_train_ohe, X_test_ohe, vectorizer.get_feature_names()
```

In [18]:

```
%%time
X_tr_essay_tfidf, X_test_essay_tfidf,TFIDF_feature_name = TFIDF_essays_sub(X_tr, y_tr, X_test, y_te
st, col_name = 'essay')
```

```
After vectorizations essay
(73196, 300) (73196,)
(36052, 300) (36052,)
===================================================================================================

CPU times: user 2min 59s, sys: 4.94 s, total: 3min 3s
Wall time: 3min 4s
```

One Hot Encoding Of Categorical Features

In [0]:

```
def one_hot_encoding(X_train, y_train, X_test, y_test,col_name):
    vectorizer = CountVectorizer()
    vectorizer.fit(X_train[col_name].values) # fit has to happen only on train data

    # we use the fitted CountVectorizer to convert the text to vector
    X_tr_ohe = vectorizer.transform(X_train[col_name].values)
    X_test_ohe = vectorizer.transform(X_test[col_name].values)

    print("After vectorizations {}".format(col_name))
    print(X_tr_ohe.shape, y_train.shape)
    print(X_test_ohe.shape, y_test.shape)
    print(vectorizer.get_feature_names())
    print("="*100)

    return X_tr_ohe, X_test_ohe, vectorizer.get_feature_names()
```

In [20]:

```
X_tr_state_ohe, X_test_state_ohe, state_feature_name = one_hot_encoding(X_tr, y_tr, X_test, y_test,
'school_state')
```

```
After vectorizations school_state
(73196, 51) (73196,)
(36052, 51) (36052,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'k
s', '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 [21]:

```
X_tr_pgc_ohe, X_test_pgc_ohe, pgc_feature_name = one_hot_encoding(X_tr, y_tr, X_test, y_test,'project_grade_category')
```

```
After vectorizations project_grade_category
(73196, 4) (73196,)
(36052, 4) (36052,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
==========================================================================================
```

In [22]:

```
X_tr_tpr_ohe, X_test_tpr_ohe, tp_feature_name = one_hot_encoding(X_tr, y_tr, X_test, y_test,'teacher_prefix')
print(len(tp_feature_name))
```

```
After vectorizations teacher_prefix
(73196, 5) (73196,)
(36052, 5) (36052,)
['dr', 'mr', 'mrs', 'ms', 'teacher']
==========================================================================================

5
```

In [23]:

```
X_tr_cln_catg_ohe, X_test_cln_catg_ohe, cln_cat_feature_name = one_hot_encoding(X_tr, y_tr, X_test, y_test,'clean_categories')
```

```
After vectorizations clean_categories
(73196, 9) (73196,)
(36052, 9) (36052,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language',
'math_science', 'music_arts', 'specialneeds', 'warmth']
==========================================================================================
```

In [24]:

```
X_tr_cln_sub_catg_ohe, X_test_cln_sub_catg_ohe, cln_sub_catg_feature_name = one_hot_encoding(X_tr, y_tr, X_test, y_test,'clean_subcategories')
```

```
After vectorizations clean_subcategories
(73196, 30) (73196,)
(36052, 30) (36052,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government',
'college_careerprep', 'communityservice', 'earlydevelopment', 'economics', 'environmentalscience',
'esl', 'extracurricular', 'financialliteracy', 'foreignlanguages', 'gym_fitness',
'health_lifescience', 'health_wellness', 'history_geography', 'literacy', 'literature_writing', 'm
athematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'socia
lsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
==========================================================================================
```

Normalize the numerical features

In [0]:

```
### Normalize your data
def norm_data(X_tr,y_tr, X_test, y_test, col_name = 'price'):

    normalizer = Normalizer()
    normalizer.fit(X_tr[col_name].values.reshape(1,-1))

    X_tr_norm = normalizer.transform(X_tr[col_name].values.reshape(1,-1))
    X_test_norm = normalizer.transform(X_test[col_name].values.reshape(1,-1))

    X_tr_norm = X_tr_norm.reshape(-1,1)
    X_test_norm = X_test_norm.reshape(-1,1)
    print("After vectorizations")
    print(X_tr_norm.shape, y_tr.shape)
    print(X_test_norm.shape, y_test.shape)
    print("="*100)
    return X_tr_norm,X_test_norm
```

In [26]:

```
X_tr_price_norm, X_test_price_norm, = norm_data(X_tr, y_tr, X_test, y_test,'price')
```

```
After vectorizations
(73196, 1) (73196,)
(36052, 1) (36052,)
====================================================================================================
```

In [27]:

```
X_tr_nopp_norm, X_test_nopp_norm = norm_data(X_tr, y_tr, X_test,
y_test,'teacher_number_of_previously_posted_projects')
```

```
After vectorizations
(73196, 1) (73196,)
(36052, 1) (36052,)
====================================================================================================
```

In [28]:

```
print(X_tr_essay_tfidf.shape)
print(X_tr_state_ohe.shape)
print(X_tr_pgc_ohe.shape)
print(X_tr_tpr_ohe.shape)
print(X_tr_cln_catg_ohe.shape)
print(X_tr_cln_sub_catg_ohe.shape)
print(X_tr_price_norm.shape)
print(X_tr_nopp_norm.shape)
```

```
(73196, 300)
(73196, 51)
(73196, 4)
(73196, 5)
(73196, 9)
(73196, 30)
(73196, 1)
(73196, 1)
```

Stack the features

In [29]:

```
## prepare your data set to train your model
from scipy.sparse import hstack
X_tr_tfidf_csr = hstack((X_tr_essay_tfidf,X_tr_state_ohe, X_tr_pgc_ohe, X_tr_tpr_ohe, X_tr_cln_catg
_ohe,X_tr_cln_sub_catg_ohe,X_tr_nopp_norm,X_tr_price_norm)).tocsr()
X_test_tfidf_csr = hstack((X_test_essay_tfidf,X_test_state_ohe, X_test_pgc_ohe, X_test_tpr_ohe, X_t
est_cln_catg_ohe,X_test_cln_sub_catg_ohe,X_test_nopp_norm,X_test_price_norm)).tocsr()

print("Final Data matrix")
print(X_tr_tfidf_csr.shape, y_tr.shape)
print(X_test_tfidf_csr.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(73196, 401) (73196,)
(36052, 401) (36052,)
============================================================================================
```

Apply Decision Train Classifier on TFIDF featurized Vector

Tune DT Classifier -- GridSearch CV

In [30]:

```
%%time
### Train DT Classifier

hyper_parameter = {'max_depth': [1, 5, 10, 50, 100 ], \
                  'min_samples_split': [5, 10, 100, 500]}

dt_clf = DecisionTreeClassifier(class_weight='balanced')
clf = GridSearchCV(dt_clf, hyper_parameter, cv=10, scoring='roc_auc', return_train_score=True, n_jo
bs=-1)
clf.fit(X_tr_tfidf_csr,y_tr)
```

```
CPU times: user 10.5 s, sys: 383 ms, total: 10.9 s
Wall time: 52min 44s
```

In [31]:

```
train_auc= clf.cv_results_['mean_train_score']
train_auc_std = clf.cv_results_['std_train_score']
test_auc = clf.cv_results_['mean_test_score']
test_auc_std = clf.cv_results_['std_test_score']

#Output of GridSearchCV
print('Best score: ',clf.best_score_)
print('Best Hyper parameters: ',clf.best_params_)
print('='*75)
print('Train AUC scores')
print(clf.cv_results_['mean_train_score'])
print('CV AUC scores')
print(clf.cv_results_['mean_test_score'])
```

```
Best score:  0.6476799178602873
Best Hyper parameters:  {'max_depth': 10, 'min_samples_split': 500}
===========================================================================
Train AUC scores
[0.54904956 0.54904956 0.54904956 0.54904956 0.64669083 0.64669083
 0.64619247 0.64577897 0.74564067 0.74452143 0.72554103 0.70518882
 0.99357554 0.98842954 0.90701177 0.79169026 0.99964381 0.99764109
 0.92341838 0.79318915]
CV AUC scores
[0.54799432 0.54799432 0.54799432 0.54799432 0.63283665 0.63283665
 0.63266045 0.63271777 0.63417463 0.63438783 0.6380736  0.64767992
 0.54335347 0.54628818 0.57711108 0.61073712 0.53575203 0.53822175
 0.56770762 0.60890594]
```

Heat Plot to Compare CV and Test AUC**

In [0]:

```
from itertools import repeat
x1 = []
y1 = []
max_depth = [1, 5, 10, 50, 100 ]
min_samples_split =  [5, 10, 100, 500]
train_auc_scores = clf.cv_results_['mean_train_score']
cv_auc_scores = clf.cv_results_['mean_test_score']
```

```
x1 = [x for item in max_depth for x in repeat(item, 4)]
for _ in max_depth:
    for item in min_samples_split:
        y1.append(item)
```

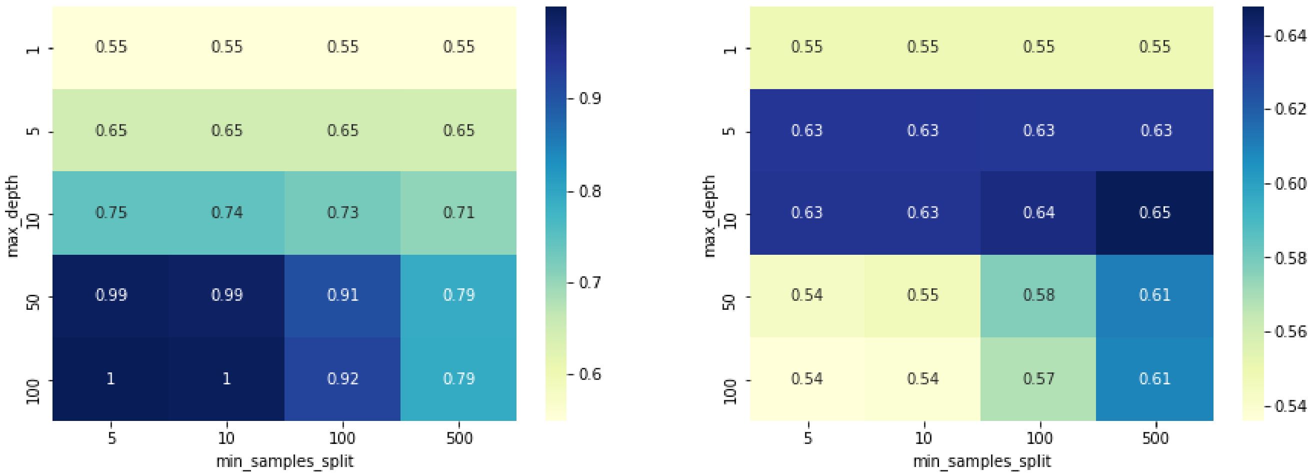
In [33]:

```
df1 = pd.DataFrame(list(zip(x1,y1,train_auc))).rename(columns = {0 : 'max_depth', 1:
'min_samples_split', 2: 'mean_train_score'}).pivot('max_depth','min_samples_split','mean_train_scor
e')
df2 = pd.DataFrame(list(zip(x1,y1,test_auc))).rename(columns = {0 : 'max_depth', 1:
'min_samples_split', 2: 'mean_CV_score'}).pivot('max_depth','min_samples_split','mean_CV_score')
plt.figure(figsize=(50,20))
f, axes = plt.subplots(1, 2,figsize=(15,5))
sns.heatmap(df1,annot=True,ax=axes[0],cbar = True ,cmap= "YlGnBu")
sns.heatmap(df2,annot=True,ax=axes[1],cbar = True ,cmap= "YlGnBu")
```

Out[33]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f13e0830898>
```

```
<Figure size 3600x1440 with 0 Axes>
```



Retrain DT Model after estimating the best hyper parameter

Best Parameter seems to be 10 as max_dept and 500 as min_sample_split

In [34]:

```
dt_clf = DecisionTreeClassifier(class_weight='balanced', max_depth = 10,
                                   min_samples_split = 500)
dt_clf.fit(X_tr_tfidf_csr,y_tr)
```

Out[34]:

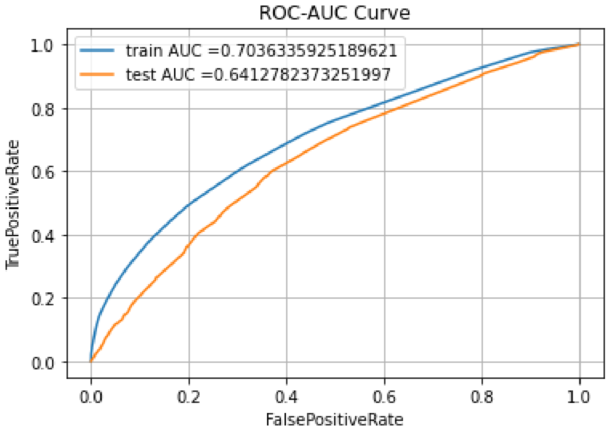
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight='balanced', criterion='gini',
                       max_depth=10, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=500,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

ROC-AUC Curve TFIDF Model

In [35]:

```
from sklearn.metrics import roc_curve, auc
y_train_pred = dt_clf.predict_proba(X_tr_tfidf_csr)[::,1]
y_test_pred =  dt_clf.predict_proba(X_test_tfidf_csr)[::,1]
```

```
train_fpr, train_tpr, tr_thresholds = roc_curve(y_tr, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)
auc_tfidf_train = auc(train_fpr, train_tpr)
auc_tfidf_test = auc(test_fpr, test_tpr)
### feature importance


plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc_tfidf_train))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc_tfidf_test))
plt.legend()
plt.xlabel("FalsePositiveRate")
plt.ylabel("TruePositiveRate")
plt.title("ROC-AUC Curve")
plt.grid()
plt.show()
```



Confusion Matrix TFIDF Implementation

In [0]:

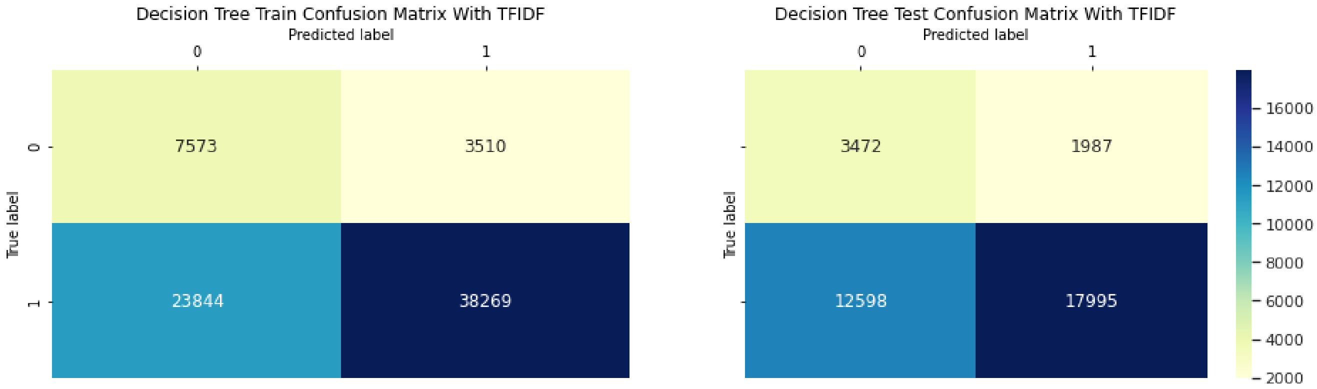
```
### find best threshold
def find_best_threshold(threshold, fpr, tpr):
    t = threshold[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    return t

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

In [37]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
y_train_pred_class = predict_with_best_t(y_train_pred, best_t)
y_test_pred_class = predict_with_best_t(y_test_pred, best_t)
train_conf_mat = pd.DataFrame(confusion_matrix(y_tr, y_train_pred_class))
test_conf_mat = pd.DataFrame(confusion_matrix(y_test, y_test_pred_class))

fig, axs = plt.subplots(1,2,figsize=(16, 4),sharey=True)
sns.set(font_scale=1)
f1 = sns.heatmap(train_conf_mat, annot=True,fmt="d", ax=axs[0], cbar = False ,cmap= "YlGnBu")

f1.set_title('Decision Tree Train Confusion Matrix With TFIDF')
f1.set_ylabel('True label')
f1.set_xlabel('Predicted label')
f1.xaxis.set_ticks_position('top')
f1.xaxis.set_label_position('top')

f2 = sns.heatmap(test_conf_mat, annot=True, fmt = 'd', ax=axs[1] , cmap = "YlGnBu")
f2.set_title('Decision Tree Test Confusion Matrix With TFIDF')
```

```
f2.set_title('Decision Tree Test Confusion Matrix With TFIDF')
f2.set_ylabel('True label')
f2.set_xlabel('Predicted label')
f2.xaxis.set_ticks_position('top')
f2.xaxis.set_label_position('top')
```

```
the maximum value of tpr*(1-fpr) 0.42099334606364613 for threshold 0.476
```



Obtain the False Positive Data - TFIDF Essay

In [0]:

```
### Get all the indices of False Postive Data from test and training data set
false_pos_tr_indices = []
for i in range(len(y_tr)):
  if y_tr[i] == 0 and y_train_pred_class[i] == 1:
    false_pos_tr_indices.append(i)

false_pos_test_indices = []
for i in range(len(y_test)):
  if y_test[i] == 0 and y_test_pred_class[i] == 1:
    false_pos_test_indices.append(i)
```

Fetching the Essay features for thos indices..

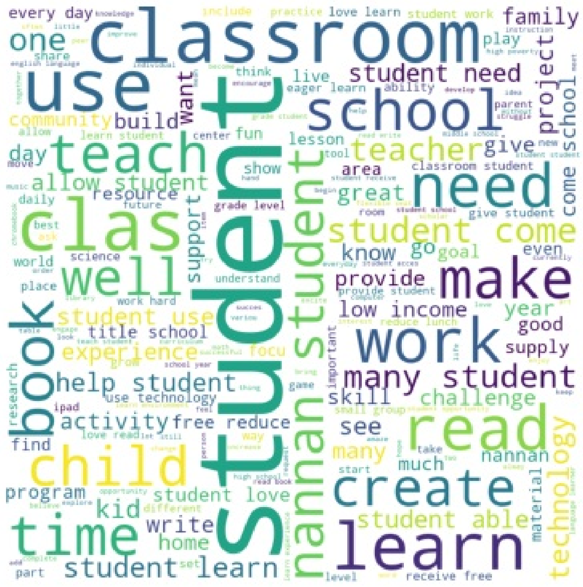
In [0]:

```
false_pos_tr_essay = []
for i in false_pos_tr_indices :
  false_pos_tr_essay.append(X_tr['essay'].values[i])

false_pos_test_essay = []
for i in false_pos_test_indices :
  false_pos_test_essay.append(X_test['essay'].values[i])
```

Plot word cloud for train and test essay for false positive data

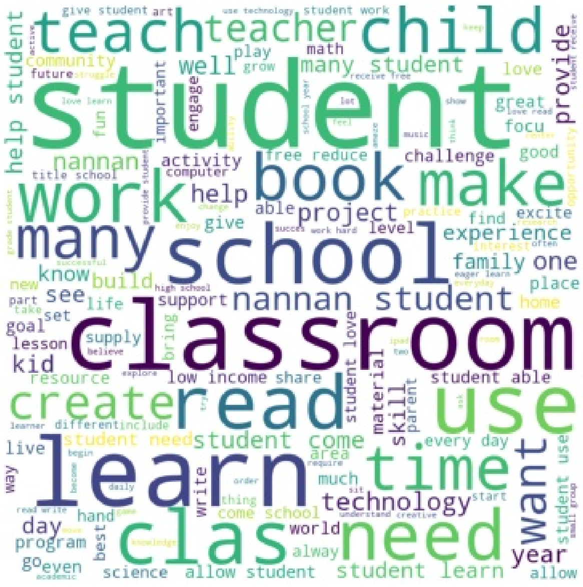
In [40]:

```
#https://www.geeksforgeeks.org/generating-word-cloud-python/
from wordcloud import WordCloud, STOPWORDS
stopwords = set(STOPWORDS)
false_positive_words = ' '
for essay in false_pos_tr_essay:
  token = str(essay).lower().split()
  false_positive_words += " ".join(token)+" "

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

# plot the WordCloud image
plt.figure(figsize = (5, 15), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```



In [41]:

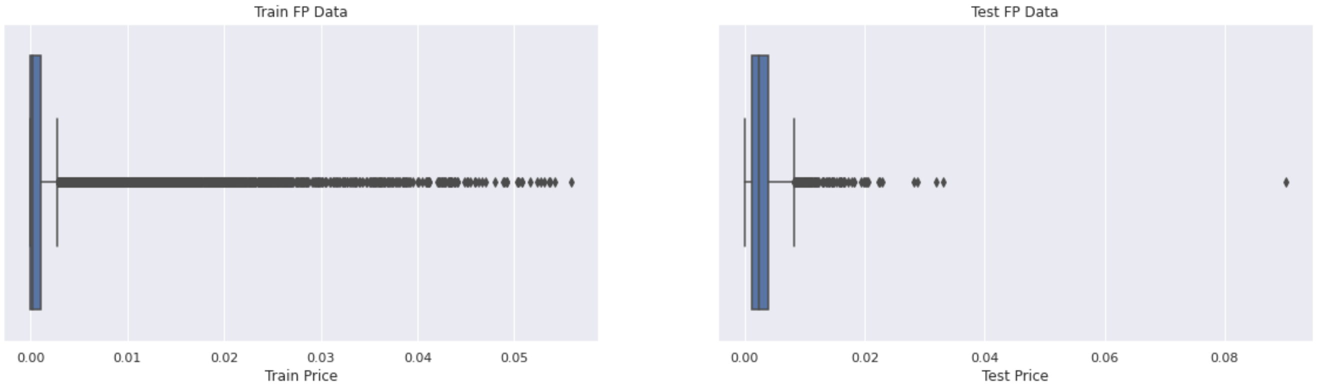
```
#https://www.geeksforgeeks.org/generating-word-cloud-python/
false_positive_words = ' '
for essay in false_pos_test_essay:
  token = str(essay).lower().split()
  false_positive_words += " ".join(token)+" "

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

# plot the WordCloud image
plt.figure(figsize = (5, 15), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```



Plot Box Plot for Price and Numeber of project previously posted by teacher

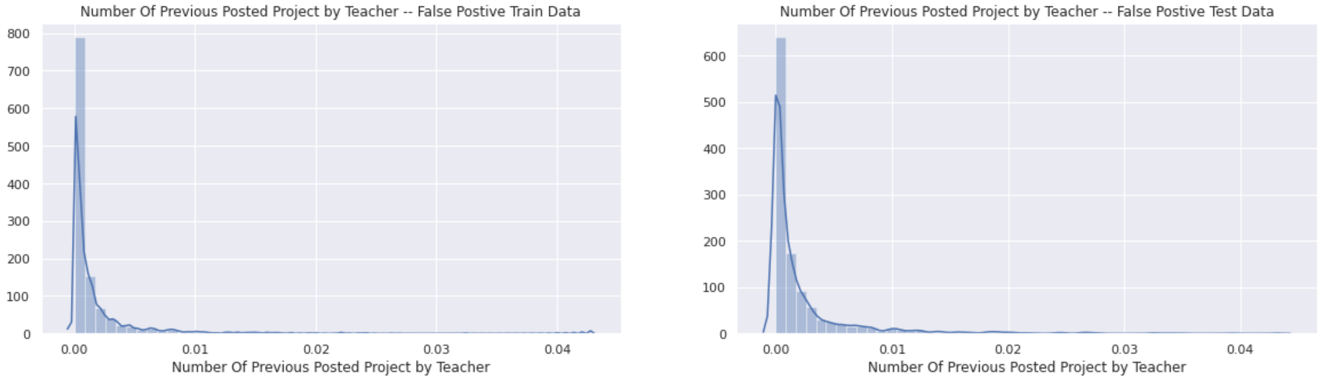
In [42]:

```
false_pos_tr_price = []
for i in false_pos_tr_indices :
  false_pos_tr_price.append(X_tr_price_norm[i])

false_pos_test_price = []
for i in false_pos_test_indices :
  false_pos_test_price.append(X_test_price_norm[i])

fig,ax = plt.subplots(1,2,figsize=(20,5))
sns.boxplot(X_tr_nopp_norm, ax= ax[0])
ax[0].set_xlabel('Train Price')
ax[0].set_title('Train FP Data')
sns.boxplot(false_pos_test_price, ax= ax[1])
ax[1].set_xlabel('Test Price')
ax[1].set_title('Test FP Data')
plt.show()
```



In [43]:

```
false_pos_tr_nopp = []
for i in false_pos_tr_indices :
  false_pos_tr_nopp.append(X_tr_nopp_norm[i])

false_pos_test_nopp = []
for i in false_pos_test_indices :
  false_pos_test_nopp.append(X_test_nopp_norm[i])

fig,ax = plt.subplots(1,2,figsize=(20,5))
sns.distplot(false_pos_tr_nopp, ax= ax[0])
ax[0].set_xlabel('Number Of Previous Posted Project by Teacher')
ax[0].set_title('Number Of Previous Posted Project by Teacher -- False Postive Train Data')
sns.distplot(false_pos_test_nopp, ax= ax[1])
ax[1].set_xlabel('Number Of Previous Posted Project by Teacher')
ax[1].set_title('Number Of Previous Posted Project by Teacher -- False Postive Test Data')
plt.show()
```



Feature selection for TFIDF

In [44]:

```
## Create list of TFIDF features
from itertools import chain
tfidf_features = (list(chain(TFIDF_feature_name
tfidf_features = (list(chain(tfidf_feature_name,
                          state_feature_name,
                          pgc_feature_name,
                          tp_feature_name,
                          cln_cat_feature_name,
                          cln_sub_catg_feature_name,
                          ['Number Of Previously Posted Project',
                          'Price']
                          )))

print(tfidf_features[300:])
```

```
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'k
s', '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', 'grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2', 'dr', 'mr', 'mrs', 'ms',
'teacher', 'appliedlearning', 'care_hunger', 'health_sports', 'history_civics',
'literacy_language', 'math_science', 'music_arts', 'specialneeds', 'warmth', 'appliedsciences',
'care_hunger', 'charactereducation', 'civics_government', 'college_careerprep',
'communityservice', 'earlydevelopment', 'economics', 'environmentalscience', 'esl',
'extracurricular', 'financialliteracy', 'foreignlanguages', 'gym_fitness', 'health_lifescience', '
health_wellness', 'history_geography', 'literacy', 'literature_writing', 'mathematics', 'music', '
nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'socialsciences',
'specialneeds', 'teamsports', 'visualarts', 'warmth', 'Number Of Previously Posted Project',
'Price']
```

In [45]:

```
def fetch_important_feature(clf,features):

    class_feature =  clf.feature_importances_
    class_imp_feature = pd.DataFrame(zip(features,class_feature),columns=['Feature_Name','Feature_I
mportance']).sort_values(by='Feature_Importance',ascending=False).reset_index(drop=True)

    return class_imp_feature,class_feature

tfidf_feature_imp,class_feature = fetch_important_feature(dt_clf, tfidf_features)

print("Top 20 Features")
tfidf_feature_imp.head(20)
```

```
Top 20 Features
```

Out[45]:

	Feature_Name	Feature_Importance
0	Price	0.186500
1	Number Of Previously Posted Project	0.113386
2	material	0.104998
3	book	0.068828
4	chromebook	0.060642
5	use	0.048735
6	ipad	0.045419
7	chair	0.038350
8	table	0.020769
9	supply	0.020531
10	want	0.012647
11	allow student	0.011055
12	need	0.008582
13	school student	0.007590
14	good	0.007110
15	student	0.005764
16	interest	0.005356
17	live	0.005312

	Feature_Name	Feature_Importance
17	live	0.005312
18	curriculum	0.005222
19	basic	0.005129

Train DT with positive features.

In [46]:

```python
tfidf_imprtnt_feature = tfidf_feature_imp[tfidf_feature_imp['Feature_Importance'] > 0 ]
print(tfidf_imprtnt_feature.shape)
tfidf_imprtnt_feature.head(10)
```

```
(95, 2)
```

Out[46]:

	Feature_Name	Feature_Importance
0	Price	0.186500
1	Number Of Previously Posted Project	0.113386
2	material	0.104998
3	book	0.068828
4	chromebook	0.060642
5	use	0.048735
6	ipad	0.045419
7	chair	0.038350
8	table	0.020769
9	supply	0.020531

In [47]:

```python
temp_tr = X_tr_tfidf_csr.T
temp_test = X_test_tfidf_csr.T
print(temp_tr.shape)
print(temp_test.shape)
print(class_feature.shape)
only_pos_tfidf_tr_feature = temp_tr[class_feature > 0].T
only_pos_tfidf_test_feature = temp_test[class_feature > 0].T
print(only_pos_tfidf_tr_feature.shape)
print(only_pos_tfidf_test_feature.shape)
```

```
(401, 73196)
(401, 36052)
(401,)
(73196, 95)
(36052, 95)
```

In [48]:

```python
%%time
### Train DT Classifier on positive feature

hyper_parameter = {'max_depth': [1, 5, 10, 50, 100 ], \
                  'min_samples_split': [5, 10, 100, 500]}

dt_clf = DecisionTreeClassifier(class_weight='balanced')
clf = GridSearchCV(dt_clf, hyper_parameter, cv=10, scoring='roc_auc', return_train_score=True, n_jo
bs=-1)
clf.fit(only_pos_tfidf_tr_feature,y_tr)
```

```
CPU times: user 5.06 s, sys: 7.46 s, total: 12.5 s
Wall time: 16min 48s
```

In [49]:

```
train_auc= clf.cv_results_['mean_train_score']
train_auc_std = clf.cv_results_['std_train_score']
test_auc = clf.cv_results_['mean_test_score']
test_auc_std = clf.cv_results_['std_test_score']

#Output of GridSearchCV
print('Best score: ',clf.best_score_)
print('Best Hyper parameters: ',clf.best_params_)
print('='*75)
print('Train AUC scores')
print(clf.cv_results_['mean_train_score'])
print('CV AUC scores')
print(clf.cv_results_['mean_test_score'])
```

```
Best score:  0.6531380226765998
Best Hyper parameters:  {'max_depth': 10, 'min_samples_split': 500}
===========================================================================
Train AUC scores
[0.54904956 0.54904956 0.54904956 0.54904956 0.64602291 0.64601754
 0.64561916 0.64520075 0.74074485 0.73946062 0.72174004 0.703048
 0.99732707 0.99175257 0.88497957 0.76940632 0.9996888  0.9960073
 0.88785155 0.76958979]
CV AUC scores
[0.54799432 0.54799432 0.54799432 0.54799432 0.63317872 0.63328742
 0.63313639 0.63308437 0.64172211 0.64172193 0.64438762 0.65313802
 0.54436234 0.54840366 0.58907169 0.62812706 0.53872437 0.54146196
 0.58732878 0.6283091 ]
```

In [0]:

```
from itertools import repeat
x1 = []
y1 = []
max_depth = [1, 5, 10, 50, 100 ]
min_samples_split =  [5, 10, 100, 500]
train_auc_scores = clf.cv_results_['mean_train_score']
cv_auc_scores = clf.cv_results_['mean_test_score']

x1 = [x for item in max_depth for x in repeat(item, 4)]
for _ in max_depth:
    for item in min_samples_split:
        y1.append(item)
```

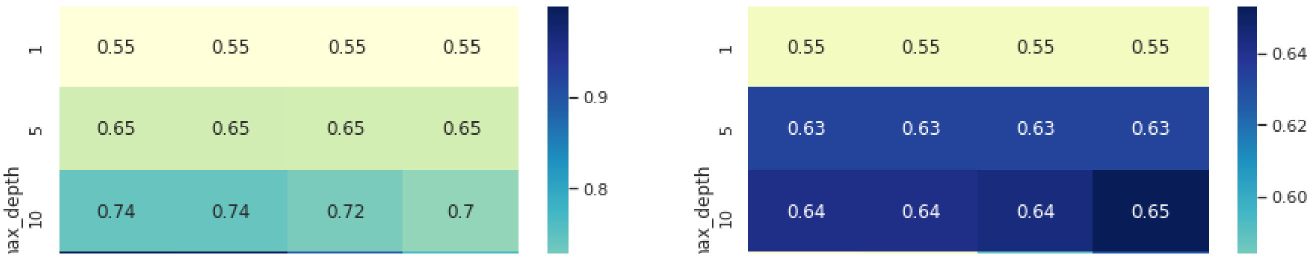
In [51]:

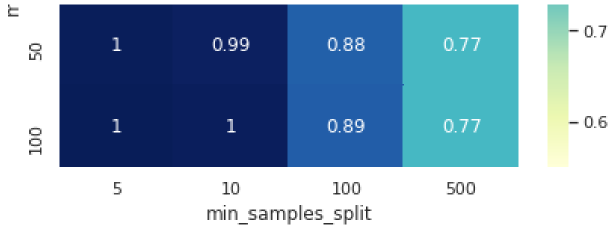
```
df1 = pd.DataFrame(list(zip(x1,y1,train_auc))).rename(columns = {0 : 'max_depth', 1:
'min_samples_split', 2: 'mean_train_score'}).pivot('max_depth','min_samples_split','mean_train_scor
e')
df2 = pd.DataFrame(list(zip(x1,y1,test_auc))).rename(columns = {0 : 'max_depth', 1:
'min_samples_split', 2: 'mean_CV_score'}).pivot('max_depth','min_samples_split','mean_CV_score')
plt.figure(figsize=(50,20))
f, axes = plt.subplots(1, 2,figsize=(15,5))
sns.heatmap(df1,annot=True,ax=axes[0],cbar = True ,cmap= "YlGnBu")
sns.heatmap(df2,annot=True,ax=axes[1],cbar = True ,cmap= "YlGnBu")
```

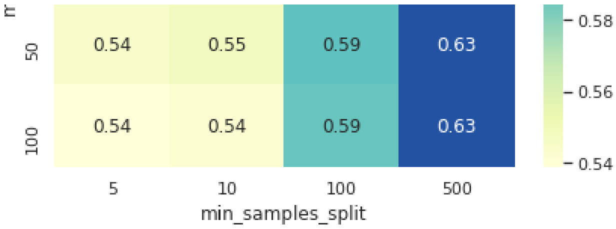
Out[51]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f13d3edadd8>
```

```
<Figure size 3600x1440 with 0 Axes>
```







Retrain DT Model after estimating the best hyper parameter

Best Parameter seems to be 10 as max_dept and 500 as min_sample_split

In [52]:

```
dt_clf = DecisionTreeClassifier(class_weight='balanced', max_depth = 10,
                                   min_samples_split = 500)
dt_clf.fit(only_pos_tfidf_tr_feature,y_tr)
```

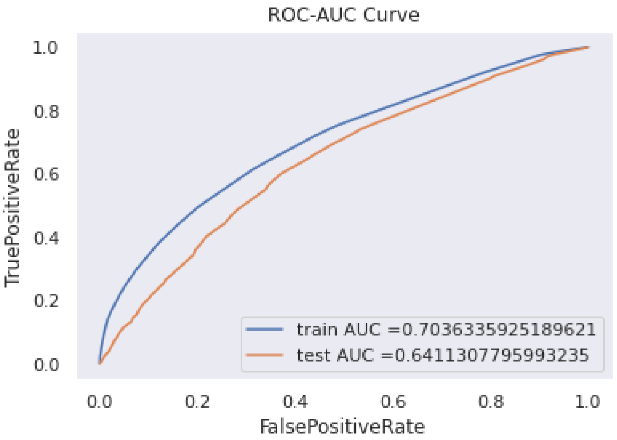
Out[52]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight='balanced', criterion='gini',
                       max_depth=10, 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='deprecated',
                       random_state=None, splitter='best')
```

ROC-AUC Curve With Positive Feature -- TFIDF Impelmentation

In [53]:

```
from sklearn.metrics import roc_curve, auc
y_train_pred = dt_clf.predict_proba(only_pos_tfidf_tr_feature)[::,1]
y_test_pred =  dt_clf.predict_proba(only_pos_tfidf_test_feature)[::,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_tr, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)
auc_tfidf_train = auc(train_fpr, train_tpr)
auc_tfidf_test = auc(test_fpr, test_tpr)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc_tfidf_train))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc_tfidf_test))
plt.legend()
plt.xlabel("FalsePositiveRate")
plt.ylabel("TruePositiveRate")
plt.title("ROC-AUC Curve")
plt.grid()
plt.show()
```

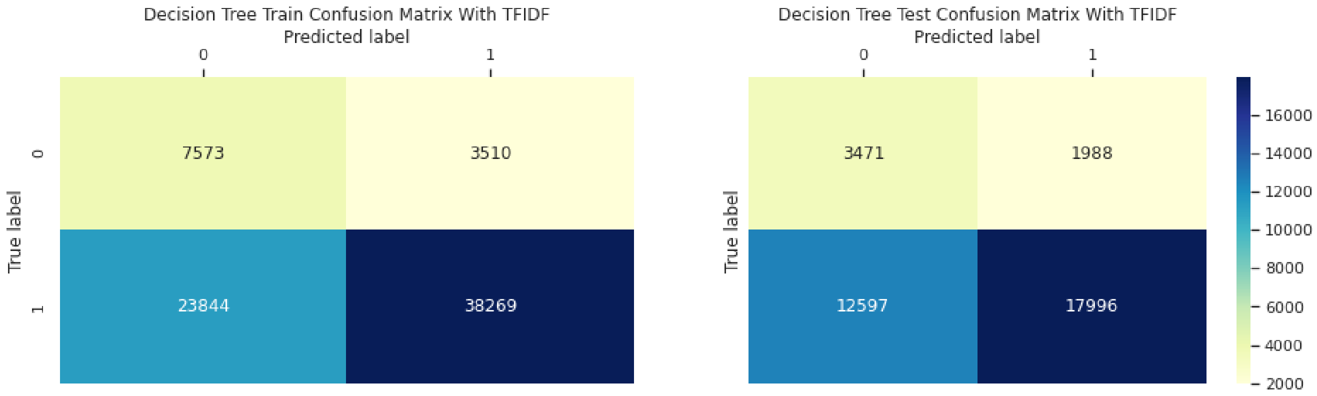


Confusion Matrix TFIDF Implementation-- Postive Features Only

In [54]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
y_train_pred_class = predict_with_best_t(y_train_pred, best_t)
y_test_pred_class = predict_with_best_t(y_test_pred, best_t)
train_conf_mat = pd.DataFrame(confusion_matrix(y_tr, y_train_pred_class))
test_conf_mat = pd.DataFrame(confusion_matrix(y_test, y_test_pred_class))

fig, axs = plt.subplots(1,2,figsize=(16, 4),sharey=True)
sns.set(font_scale=1)
f1 = sns.heatmap(train_conf_mat, annot=True,fmt="d", ax=axs[0], cbar = False ,cmap= "YlGnBu")

f1.set_title('Decision Tree Train Confusion Matrix With TFIDF')
f1.set_ylabel('True label')
f1.set_xlabel('Predicted label')
f1.xaxis.set_ticks_position('top')
f1.xaxis.set_label_position('top')

f2 = sns.heatmap(test_conf_mat, annot=True, fmt = 'd', ax=axs[1] , cmap = "YlGnBu")
f2.set_title('Decision Tree Test Confusion Matrix With TFIDF')
f2.set_ylabel('True label')
f2.set_xlabel('Predicted label')
f2.xaxis.set_ticks_position('top')
f2.xaxis.set_label_position('top')
```

```
the maximum value of tpr*(1-fpr) 0.42099334606364613 for threshold 0.476
```



Conclusion ROC-AUC is sames as that of earlir but there is huge change in train time. Train time was reduced from 53 minutes to 18 minutes approximately 1/3 of the earlier time as the train time complexity of DT is O(*nlogn*d)

TFIDF Implementation with new features i.e. Subjectivity, Polarity,etc...

In [0]:

```
positive_tr   = np.array(X_tr['positive']).reshape(-1,1)
positive_test = np.array(X_test['positive']).reshape(-1,1)
negative_tr   = np.array(X_tr['negative']).reshape(-1,1)
negative_test = np.array(X_test['negative']).reshape(-1,1)
neutral_tr    = np.array(X_tr['neutral']).reshape(-1,1)
neutral_test  = np.array(X_test['neutral']).reshape(-1,1)
compund_tr    = np.array(X_tr['compound']).reshape(-1,1)
compound_test = np.array(X_test['compound']).reshape(-1,1)
polarity_tr   = np.array(X_tr['polarity']).reshape(-1,1)
polarity_test = np.array(X_test['polarity']).reshape(-1,1)
subjectivity_tr = np.array(X_tr['subjectivity']).reshape(-1,1)
subjectivity_test = np.array(X_test['subjectivity']).reshape(-1,1)
```

In [56]:

```
## prepare your data set to train your model
from scipy.sparse import hstack
X_tr_tfidf_with_new_ftr = hstack((X_tr_tfidf_csr,positive_tr, negative_tr, neutral_tr, compund_tr,
polarity_tr, subjectivity_tr)).tocsr()
X_test_tfidf_with_new_ftr = hstack((X_test_tfidf_csr,positive_test, negative_test, neutral_test,
compound_test, polarity_test, subjectivity_test)).tocsr()
```

```
print("Final Data matrix")
print(X_tr_tfidf_with_new_ftr.shape, y_tr.shape)
print(X_test_tfidf_with_new_ftr.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(73196, 407) (73196,)
(36052, 407) (36052,)
====================================================================================================
```

In [57]:

```
%%time
### Train DT Classifier

hyper_parameter = {'max_depth': [1, 5, 10, 50, 100 ], \
                   'min_samples_split': [5, 10, 100, 500]}

dt_clf = DecisionTreeClassifier(class_weight='balanced')
clf = GridSearchCV(dt_clf, hyper_parameter, cv=10, scoring='roc_auc', return_train_score=True, n_jo
bs=-1)
clf.fit(X_tr_tfidf_with_new_ftr,y_tr)
```

```
CPU times: user 11.3 s, sys: 7.33 s, total: 18.6 s
Wall time: 55min 48s
```

In [58]:

```
train_auc= clf.cv_results_['mean_train_score']
train_auc_std = clf.cv_results_['std_train_score']
test_auc = clf.cv_results_['mean_test_score']
test_auc_std = clf.cv_results_['std_test_score']

#Output of GridSearchCV
print('Best score: ',clf.best_score_)
print('Best Hyper parameters: ',clf.best_params_)
print('='*75)
print('Train AUC scores')
print(clf.cv_results_['mean_train_score'])
print('CV AUC scores')
print(clf.cv_results_['mean_test_score'])
```

```
Best score:  0.6477149686300451
Best Hyper parameters:  {'max_depth': 10, 'min_samples_split': 500}
===========================================================================
Train AUC scores
[0.54904956 0.54904956 0.54904956 0.54904956 0.64675126 0.64675126
 0.6462395  0.64582005 0.7470407  0.74593854 0.72688679 0.70622502
 0.99396785 0.98900791 0.90966897 0.79031809 0.99972406 0.9977851
 0.92506441 0.7911859 ]
CV AUC scores
[0.54799432 0.54799432 0.54799432 0.54799432 0.63264778 0.63264778
 0.63245828 0.63250517 0.63515844 0.63460261 0.63837797 0.64771497
 0.54811887 0.55116159 0.57526381 0.61177398 0.53779641 0.5374681
 0.56385397 0.61124579]
```

In [0]:

```
from itertools import repeat
x1 = []
y1 = []
max_depth = [1, 5, 10, 50, 100 ]
min_samples_split =  [5, 10, 100, 500]
train_auc_scores = clf.cv_results_['mean_train_score']
cv_auc_scores = clf.cv_results_['mean_test_score']

x1 = [x for item in max_depth for x in repeat(item, 4)]
for _ in max_depth:
    for item in min_samples_split:
        y1.append(item)
```

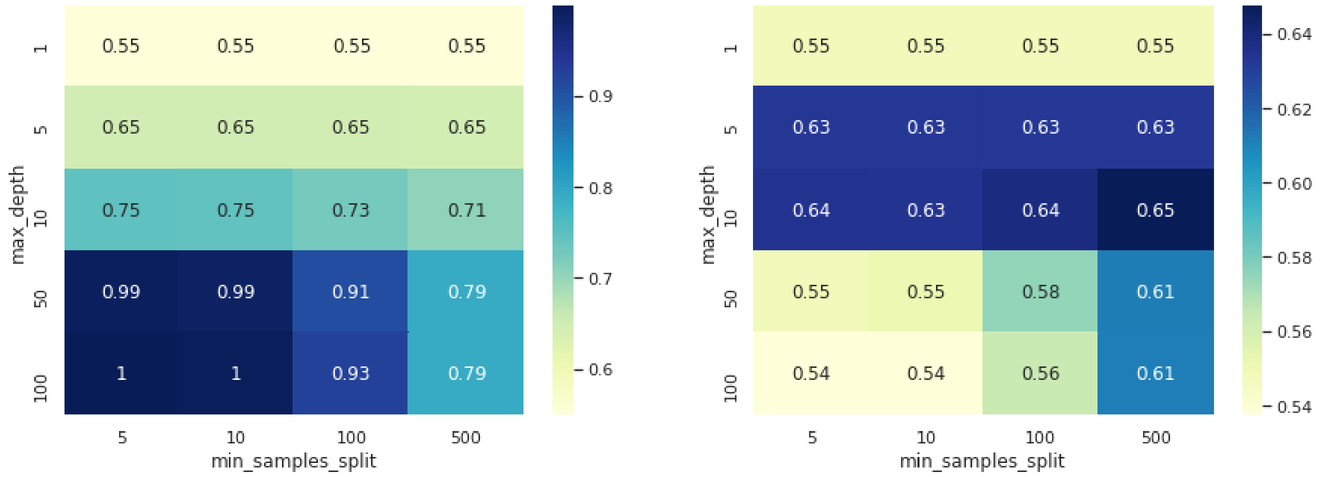
In [60]:

```
df1 = pd.DataFrame(list(zip(x1,y1,train_auc))).rename(columns = {0 : 'max_depth', 1:
'min_samples_split', 2: 'mean_train_score'}).pivot('max_depth','min_samples_split','mean_train_scor
e')
df2 = pd.DataFrame(list(zip(x1,y1,test_auc))).rename(columns = {0 : 'max_depth', 1:
'min_samples_split', 2: 'mean_CV_score'}).pivot('max_depth','min_samples_split','mean_CV_score')
plt.figure(figsize=(50,20))
f, axes = plt.subplots(1, 2,figsize=(15,5))
sns.heatmap(df1,annot=True,ax=axes[0],cbar = True ,cmap= "YlGnBu")
sns.heatmap(df2,annot=True,ax=axes[1],cbar = True ,cmap= "YlGnBu")
```

Out[60]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f13d209f7b8>
```

```
<Figure size 3600x1440 with 0 Axes>
```



In [61]:

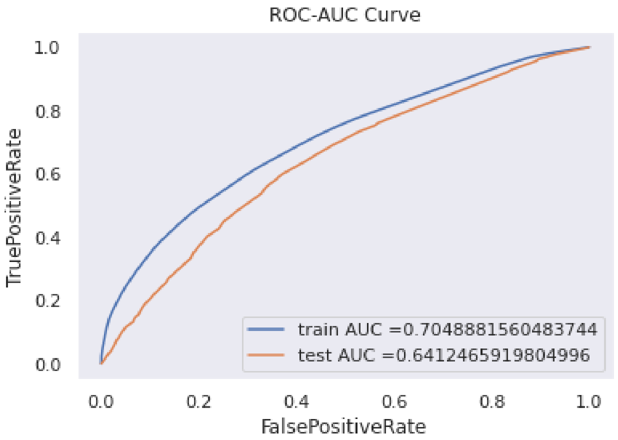
```
dt_clf = DecisionTreeClassifier(class_weight='balanced', max_depth = 10,
                                min_samples_split = 500)
dt_clf.fit(X_tr_tfidf_with_new_ftr,y_tr)
```

Out[61]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight='balanced', criterion='gini',
                       max_depth=10, 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='deprecated',
                       random_state=None, splitter='best')
```

In [62]:

```
from sklearn.metrics import roc_curve, auc
y_train_pred = dt_clf.predict_proba(X_tr_tfidf_with_new_ftr)[::,1]
y_test_pred =  dt_clf.predict_proba(X_test_tfidf_with_new_ftr)[::,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_tr, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)
auc_tfidf_train = auc(train_fpr, train_tpr)
auc_tfidf_test = auc(test_fpr, test_tpr)
### feature importance


plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc_tfidf_train))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc_tfidf_test))
plt.legend()
plt.xlabel("FalsePositiveRate")
plt.ylabel("TruePositiveRate")
plt.title("ROC-AUC Curve")
plt.grid()
plt.grid()
plt.show()
```



DT -- TFIDF Word2Vec Implementation

Weighted TFIDF Word2Vec Implementation

In [0]:

```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_tr['essay'])
# 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())
```

In [0]:

```
with open('/content/gdrive/My Drive/9_Donors_choose_DT/glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [65]:

```
tfidf_w2v_tr_vector = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_tr['essay']): # 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
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_tr_vector.append(vector)

print(len(tfidf_w2v_tr_vector))
print(len(tfidf_w2v_tr_vector[0]))
```

```
73196
300
```

In [66]:

```
tfidf_w2v_test_vector = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in tqdm(X_test['essay']): # 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
    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
value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tf
idf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    tfidf_w2v_test_vector.append(vector)

print(len(tfidf_w2v_test_vector))
print(len(tfidf_w2v_test_vector[0]))
```

```
36052
300
```

In [67]:

```
## prepare your data set to train your model
from scipy.sparse import hstack
X_tr_w2v_csr = hstack((tfidf_w2v_tr_vector,X_tr_state_ohe, X_tr_pgc_ohe, X_tr_tpr_ohe,
X_tr_cln_catg_ohe,X_tr_cln_sub_catg_ohe,X_tr_nopp_norm,X_tr_price_norm)).tocsr()
X_test_w2v_csr = hstack((tfidf_w2v_test_vector,X_test_state_ohe, X_test_pgc_ohe, X_test_tpr_ohe, X_
test_cln_catg_ohe,X_test_cln_sub_catg_ohe,X_test_nopp_norm,X_test_price_norm)).tocsr()

print("Final Data matrix")
print(X_tr_w2v_csr.shape, y_tr.shape)
print(X_test_w2v_csr.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(73196, 401) (73196,)
(36052, 401) (36052,)
====================================================================================================
```

HyperParameter Tunning DT-- GridSearch Classifier

In [68]:

```
%%time
### Train DT Classifier

hyper_parameter = {'max_depth': [1, 5, 10, 50, 100 ], \
                  'min_samples_split': [5, 10, 100, 500]}

dt_clf = DecisionTreeClassifier(class_weight='balanced')
clf = GridSearchCV(dt_clf, hyper_parameter, cv=10, scoring='roc_auc', return_train_score=True, n_jo
bs=-1)
clf.fit(X_tr_w2v_csr,y_tr)
```

```
CPU times: user 25 s, sys: 1.08 s, total: 26.1 s
Wall time: 2h 17min 38s
CPU times: user 25 s, sys: 1.08 s, total: 26.1 s
Wall time: 2h 17min 38s
```

In [69]:

```
train_auc= clf.cv_results_['mean_train_score']
train_auc_std = clf.cv_results_['std_train_score']
test_auc = clf.cv_results_['mean_test_score']
test_auc_std = clf.cv_results_['std_test_score']

#Output of GridSearchCV
print('Best score: ',clf.best_score_)
print('Best Hyper parameters: ',clf.best_params_)
print('='*75)
print('Train AUC scores')
print(clf.cv_results_['mean_train_score'])
print('CV AUC scores')
print(clf.cv_results_['mean_test_score'])
```

```
Best score:  0.6382868559454432
Best Hyper parameters:  {'max_depth': 5, 'min_samples_split': 500}
===========================================================================
Train AUC scores
[0.54904956 0.54904956 0.54904956 0.54904956 0.66006402 0.66006402
 0.66001924 0.65936087 0.80018316 0.79914268 0.77844185 0.73560541
 0.99986641 0.99896475 0.90467948 0.76169806 0.9999464  0.99918136
 0.90503426 0.76173578]
CV AUC scores
[0.54799432 0.54799432 0.54799432 0.54799432 0.63796837 0.63796837
 0.63793302 0.63828686 0.61610701 0.61596045 0.62048555 0.63778324
 0.53215089 0.53380895 0.57045618 0.62462892 0.53206452 0.53318243
 0.57014149 0.62441309]
Best score:  0.6382868559454432
Best Hyper parameters:  {'max_depth': 5, 'min_samples_split': 500}
===========================================================================
Train AUC scores
[0.54904956 0.54904956 0.54904956 0.54904956 0.66006402 0.66006402
 0.66001924 0.65936087 0.80018316 0.79914268 0.77844185 0.73560541
 0.99986641 0.99896475 0.90467948 0.76169806 0.9999464  0.99918136
 0.90503426 0.76173578]
CV AUC scores
[0.54799432 0.54799432 0.54799432 0.54799432 0.63796837 0.63796837
 0.63793302 0.63828686 0.61610701 0.61596045 0.62048555 0.63778324
 0.53215089 0.53380895 0.57045618 0.62462892 0.53206452 0.53318243
 0.57014149 0.62441309]
```

Scatter Plot (3D) - Compute Best HyperParameter

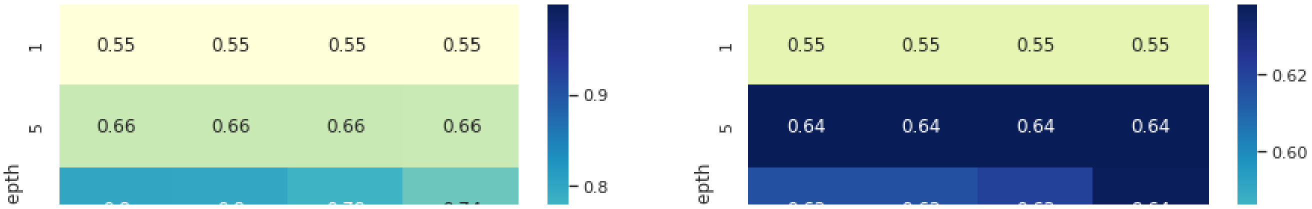
In [0]:

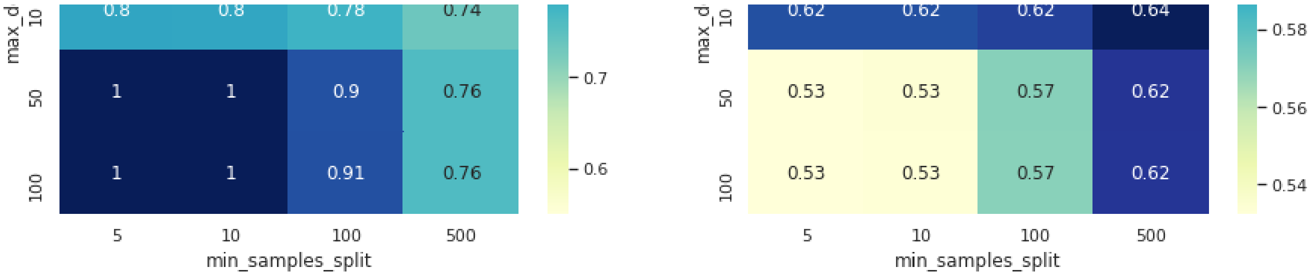
```
from itertools import repeat
x1 = []
y1 = []
max_depth = [1, 5, 10, 50, 100 ]
min_samples_split =  [5, 10, 100, 500]
train_auc_scores = clf.cv_results_['mean_train_score']
cv_auc_scores = clf.cv_results_['mean_test_score']

x1 = [x for item in max_depth for x in repeat(item, 4)]
for _ in max_depth:
    for item in min_samples_split:
        y1.append(item)
```

In [72]:

```
df1 = pd.DataFrame(list(zip(x1,y1,train_auc))).rename(columns = {0 : 'max_depth', 1:
'min_samples_split', 2: 'mean_train_score'}).pivot('max_depth','min_samples_split','mean_train_scor
e')
df2 = pd.DataFrame(list(zip(x1,y1,test_auc))).rename(columns = {0 : 'max_depth', 1:
'min_samples_split', 2: 'mean_CV_score'}).pivot('max_depth','min_samples_split','mean_CV_score')
plt.figure(figsize=(50,20))
f, axes = plt.subplots(1, 2,figsize=(15,5))
sns.heatmap(df1,annot=True,ax=axes[0],cbar = True ,cmap= "YlGnBu")
sns.heatmap(df2,annot=True,ax=axes[1],cbar = True ,cmap= "YlGnBu")
plt.show()
```

```
<Figure size 3600x1440 with 0 Axes>
```





Retrain DT with Word2Vec with best parameter

In [84]:

```
dt_clf = DecisionTreeClassifier(class_weight='balanced', max_depth = 5,
                                    min_samples_split = 500)
dt_clf.fit(X_tr_w2v_csr,y_tr)
```

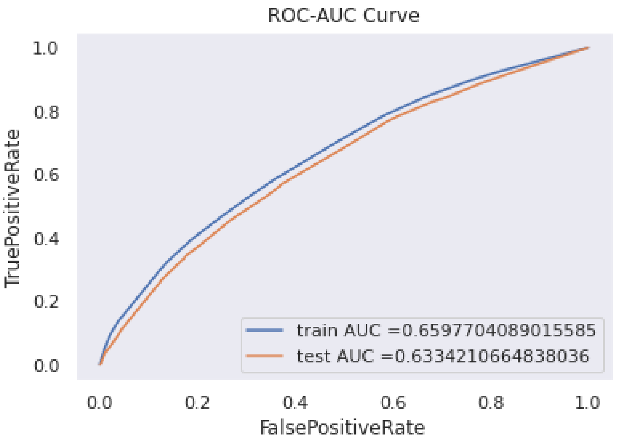
Out[84]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight='balanced', criterion='gini',
                       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, presort='deprecated',
                       random_state=None, splitter='best')
```

ROC-AUC Curve for Word2Vec Implementation

In [85]:

```
from sklearn.metrics import roc_curve, auc
y_train_pred = dt_clf.predict_proba(X_tr_w2v_csr)[::,1]
y_test_pred =  dt_clf.predict_proba(X_test_w2v_csr)[::,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_tr, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)
auc_w2v_train = auc(train_fpr, train_tpr)
auc_w2v_test = auc(test_fpr, test_tpr)
### feature importance


plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc_w2v_train))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc_w2v_test))
plt.legend()
plt.xlabel("FalsePositiveRate")
plt.ylabel("TruePositiveRate")
plt.title("ROC-AUC Curve")
plt.grid()
plt.show()
```

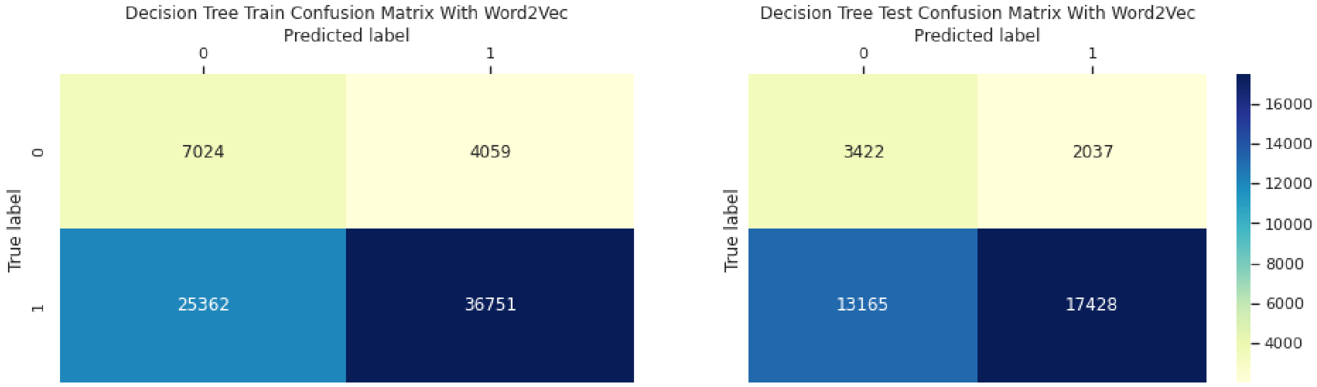


Plot Confusion Matrix for Word2Vec Implementation

In [86]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
y_train_pred_class = predict_with_best_t(y_train_pred, best_t)
y_test_pred_class = predict_with_best_t(y_test_pred, best_t)
train_conf_mat = pd.DataFrame(confusion_matrix(y_tr, y_train_pred_class))
test_conf_mat = pd.DataFrame(confusion_matrix(y_test, y_test_pred_class))

fig, axs = plt.subplots(1,2,figsize=(16, 4),sharey=True)
sns.set(font_scale=1)
f1 = sns.heatmap(train_conf_mat, annot=True,fmt="d", ax=axs[0], cbar = False ,cmap= "YlGnBu")

f1.set_title('Decision Tree Train Confusion Matrix With Word2Vec')
f1.set_ylabel('True label')
f1.set_xlabel('Predicted label')
f1.xaxis.set_ticks_position('top')
f1.xaxis.set_label_position('top')

f2 = sns.heatmap(test_conf_mat, annot=True, fmt = 'd', ax=axs[1] , cmap = "YlGnBu")
f2.set_title('Decision Tree Test Confusion Matrix With Word2Vec')
f2.set_ylabel('True label')
f2.set_xlabel('Predicted label')
f2.xaxis.set_ticks_position('top')
f2.xaxis.set_label_position('top')
```

```
the maximum value of tpr*(1-fpr) 0.37498493877191424 for threshold 0.482
```



Obtain the False Positive Data - Word2Vec Essay

In [0]:

```
### Get all the indices of False Postive Data from test and training data set
false_pos_tr_indices = []
for i in range(len(y_tr)):
  if y_tr[i] == 0 and y_train_pred_class[i] == 1:
    false_pos_tr_indices.append(i)

false_pos_test_indices = []
for i in range(len(y_test)):
  if y_test[i] == 0 and y_test_pred_class[i] == 1:
    false_pos_test_indices.append(i)
```

Fetching Esssay feature for those indices

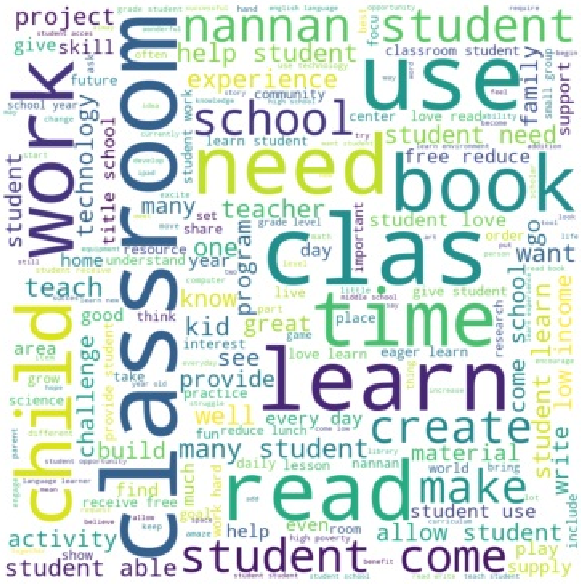
In [0]:

```
false_pos_tr_essay = []
for i in false_pos_tr_indices :
  false_pos_tr_essay.append(X_tr['essay'].values[i])

false_pos_test_essay = []
for i in false_pos_test_indices :
  false_pos_test_essay.append(X_test['essay'].values[i])
```

Plot Word Cloud for train and test data for false postive data points

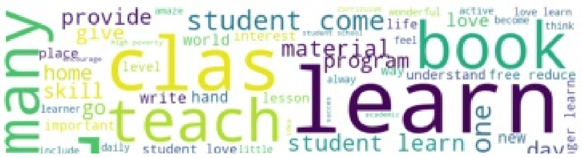
In [89]:

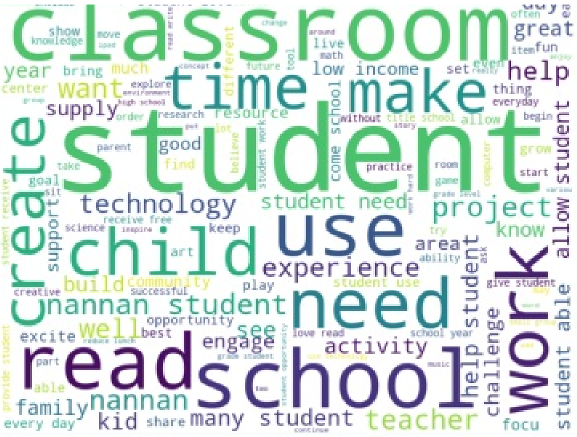
```
#https://www.geeksforgeeks.org/generating-word-cloud-python/
from wordcloud import WordCloud, STOPWORDS
stopwords = set(STOPWORDS)
false_positive_words = ' '
for essay in false_pos_tr_essay:
  token = str(essay).lower().split()
  false_positive_words += " ".join(token)+" "

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

# plot the WordCloud image
plt.figure(figsize = (5, 15), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```



In [90]:

```
#https://www.geeksforgeeks.org/generating-word-cloud-python/
false_positive_words = ' '
for essay in false_pos_test_essay:
  token = str(essay).lower().split()
  false_positive_words += " ".join(token)+" "

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

# plot the WordCloud image
plt.figure(figsize = (5, 15), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```

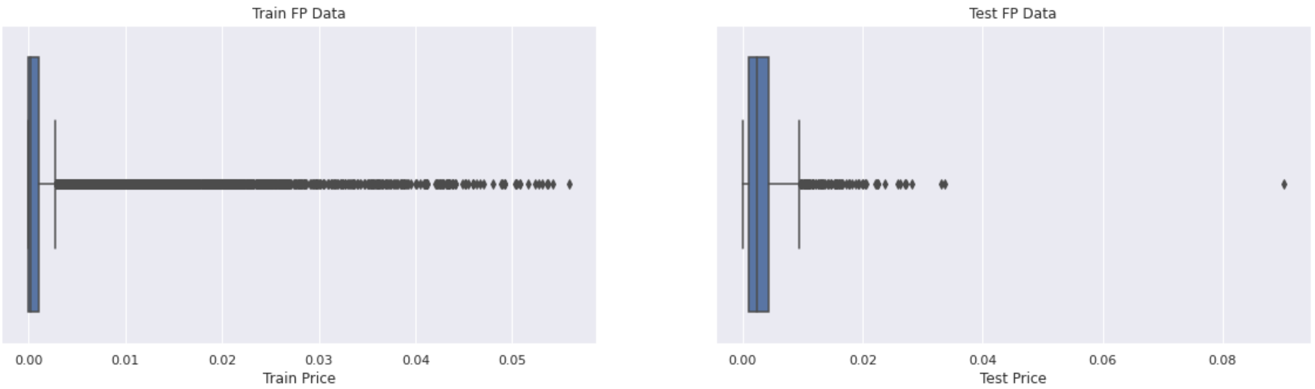




Plot Box Plot for price and distrubtion for Number of previously postd project

In [91]:

```
false_pos_tr_price = []
for i in false_pos_tr_indices :
  false_pos_tr_price.append(X_tr_price_norm[i])

false_pos_test_price = []
for i in false_pos_test_indices :
  false_pos_test_price.append(X_test_price_norm[i])

fig,ax = plt.subplots(1,2,figsize=(20,5))
sns.boxplot(X_tr_nopp_norm, ax= ax[0])
ax[0].set_xlabel('Train Price')
ax[0].set_title('Train FP Data')
sns.boxplot(false_pos_test_price, ax= ax[1])
ax[1].set_xlabel('Test Price')
ax[1].set_title('Test FP Data')
plt.show()
```

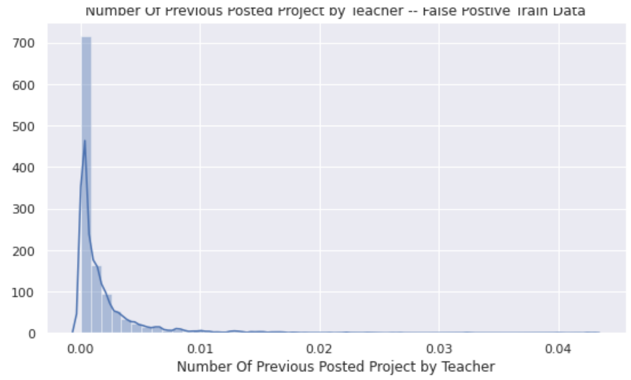


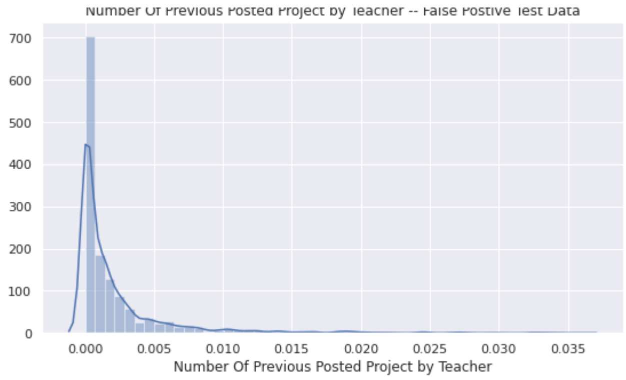
In [92]:

```
false_pos_tr_nopp = []
for i in false_pos_tr_indices :
  false_pos_tr_nopp.append(X_tr_nopp_norm[i])

false_pos_test_nopp = []
for i in false_pos_test_indices :
  false_pos_test_nopp.append(X_test_nopp_norm[i])

fig,ax = plt.subplots(1,2,figsize=(20,5))
sns.distplot(false_pos_tr_nopp, ax= ax[0])
ax[0].set_xlabel('Number Of Previous Posted Project by Teacher')
ax[0].set_title('Number Of Previous Posted Project by Teacher -- False Postive Train Data')
sns.distplot(false_pos_test_nopp, ax= ax[1])
ax[1].set_xlabel('Number Of Previous Posted Project by Teacher')
ax[1].set_title('Number Of Previous Posted Project by Teacher -- False Postive Test Data')
plt.show()
```

Number Of Previous Posted Project by Teacher -- False Postive Train Data

Number Of Previous Posted Project by Teacher -- False Postive Test Data


Number Of Previous Posted Project by Teacher -- False Postive Train Data


Number Of Previous Posted Project by Teacher -- False Postive Test Data

Train DT with only positive important Features

In [0]:

```
class_feature = dt_clf.feature_importances_
```

In [97]:

```
temp_tr = X_tr_w2v_csr.T
temp_test = X_test_w2v_csr.T
print(temp_tr.shape)
print(temp_test.shape)
print(class_feature.shape)
only_pos_tfw2v_tr_feature = temp_tr[class_feature > 0].T
only_pos_tfw2v_test_feature = temp_test[class_feature > 0].T
print(only_pos_tfw2v_tr_feature.shape)
print(only_pos_tfw2v_test_feature.shape)
```

```
(401, 73196)
(401, 36052)
(401,)
(73196, 21)
(36052, 21)
```

In [98]:

```
%%time
### Train DT Classifier on positive feature

hyper_parameter = {'max_depth': [1, 5, 10, 50, 100 ], \
                   'min_samples_split': [5, 10, 100, 500]}

dt_clf = DecisionTreeClassifier(class_weight='balanced')
clf = GridSearchCV(dt_clf, hyper_parameter, cv=10, scoring='roc_auc', return_train_score=True, n_jo
bs=-1)
clf.fit(only_pos_tfw2v_tr_feature,y_tr)
```

```
CPU times: user 5.08 s, sys: 286 ms, total: 5.37 s
Wall time: 9min 20s
```

In [99]:

```
train_auc= clf.cv_results_['mean_train_score']
train_auc_std = clf.cv_results_['std_train_score']
test_auc = clf.cv_results_['mean_test_score']
test_auc_std = clf.cv_results_['std_test_score']

#Output of GridSearchCV
print('Best score: ',clf.best_score_)
print('Best Hyper parameters: ',clf.best_params_)
print('='*75)
print('Train AUC scores')
print(clf.cv_results_['mean_train_score'])
print('CV AUC scores')
print(clf.cv_results_['mean_test_score'])
```

```
Best score:  0.6465565303853363
Best Hyper parameters:  {'max_depth': 10, 'min_samples_split': 500}
===========================================================================
Train AUC scores
[0.54904956 0.54904956 0.54904956 0.54904956 0.65943786 0.65943786
 0.65939235 0.65885927 0.77078083 0.76943698 0.75088776 0.71816412
 0.99950357 0.99354987 0.85550851 0.73765895 0.9995064  0.99353273
 0.85550416 0.73764683]
CV AUC scores
[0.54799432 0.54799432 0.54799432 0.54799432 0.64218184 0.64218184
 0.64215162 0.64262584 0.629447   0.62960097 0.63434159 0.64655653
 0.53643607 0.54365376 0.59936611 0.63925461 0.53749094 0.544134
 0.59906406 0.63930766]
```

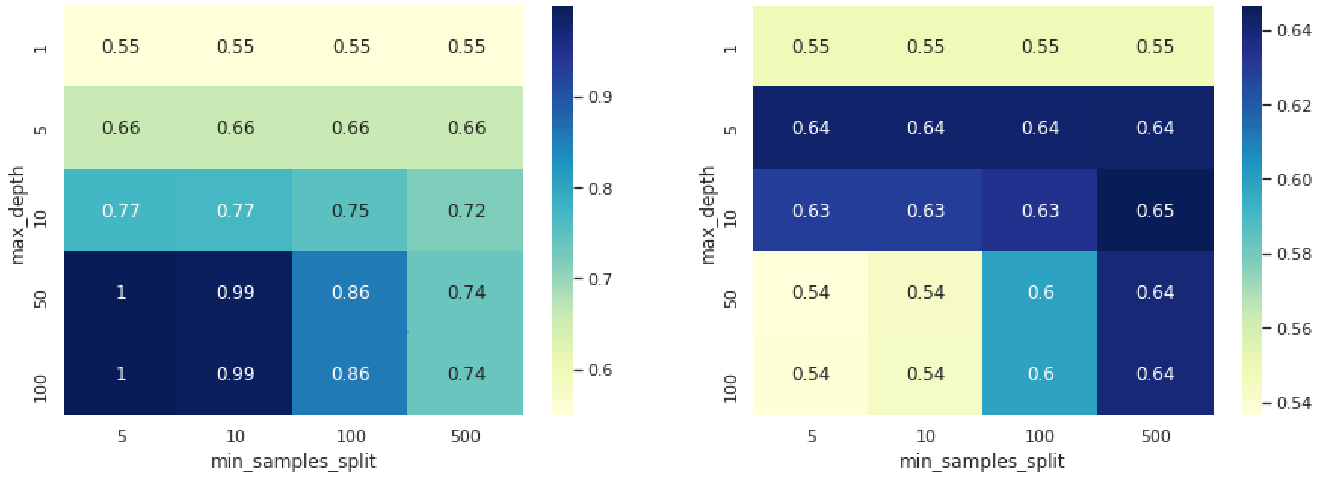
In [0]:

```
from itertools import repeat
x1 = []
y1 = []
max_depth = [1, 5, 10, 50, 100 ]
min_samples_split =  [5, 10, 100, 500]
train_auc_scores = clf.cv_results_['mean_train_score']
cv_auc_scores = clf.cv_results_['mean_test_score']

x1 = [x for item in max_depth for x in repeat(item, 4)]
for _ in max_depth:
    for item in min_samples_split:
        y1.append(item)
```

In [101]:

```
df1 = pd.DataFrame(list(zip(x1,y1,train_auc))).rename(columns = {0 : 'max_depth', 1:
'min_samples_split', 2: 'mean_train_score'}).pivot('max_depth','min_samples_split','mean_train_scor
e')
df2 = pd.DataFrame(list(zip(x1,y1,test_auc))).rename(columns = {0 : 'max_depth', 1:
'min_samples_split', 2: 'mean_CV_score'}).pivot('max_depth','min_samples_split','mean_CV_score')
plt.figure(figsize=(50,20))
f, axes = plt.subplots(1, 2,figsize=(15,5))
sns.heatmap(df1,annot=True,ax=axes[0],cbar = True ,cmap= "YlGnBu")
sns.heatmap(df2,annot=True,ax=axes[1],cbar = True ,cmap= "YlGnBu")
plt.show()
```

```
<Figure size 3600x1440 with 0 Axes>
```



In [102]:

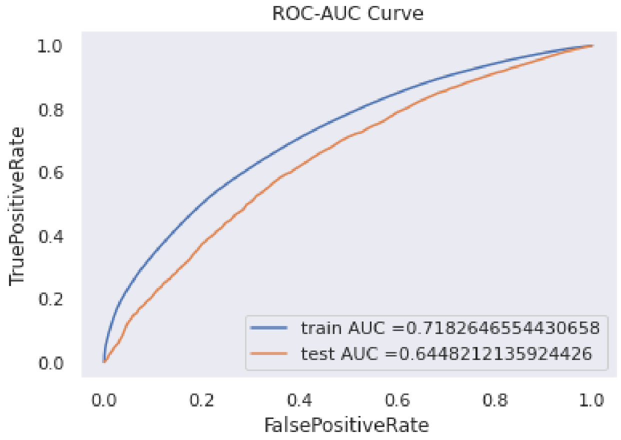
```
dt_clf = DecisionTreeClassifier(class_weight='balanced', max_depth = 10,
                                   min_samples_split = 500)
dt_clf.fit(only_pos_tfw2v_tr_feature,y_tr)
```

Out[102]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight='balanced', criterion='gini',
                                    max_depth=10, 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='deprecated',
                                    random_state=None, splitter='best')
```

In [103]:

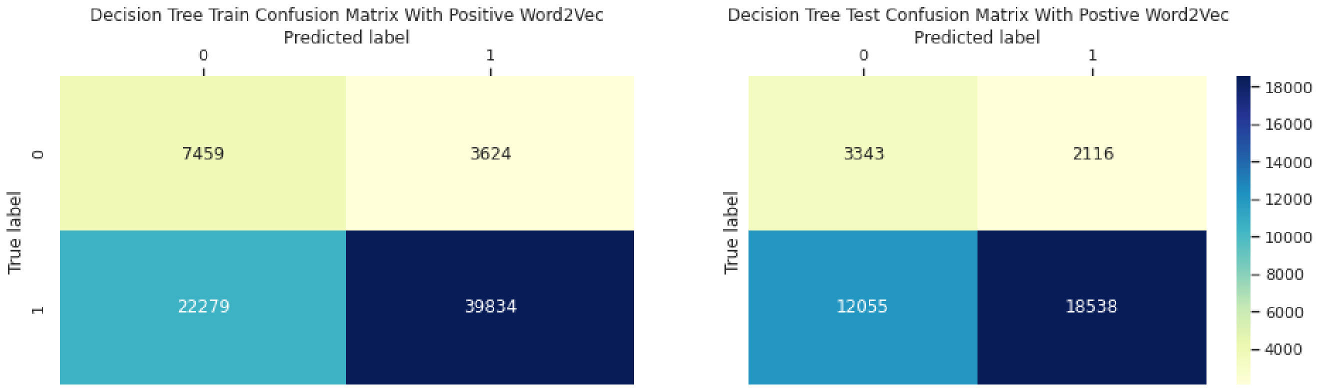
```
from sklearn.metrics import roc_curve, auc
y_train_pred = dt_clf.predict_proba(only_pos_tfw2v_tr_feature)[::,1]
y_test_pred =  dt_clf.predict_proba(only_pos_tfw2v_test_feature)[::,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_tr, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)
auc_w2v_train = auc(train_fpr, train_tpr)
auc_w2v_test = auc(test_fpr, test_tpr)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc_w2v_train))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc_w2v_test))
plt.legend()
plt.xlabel("FalsePositiveRate")
plt.ylabel("TruePositiveRate")

plt.title("ROC-AUC Curve")
plt.grid()
plt.show()
```



In [104]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
y_train_pred_class = predict_with_best_t(y_train_pred, best_t)
y_test_pred_class = predict_with_best_t(y_test_pred, best_t)
train_conf_mat = pd.DataFrame(confusion_matrix(y_tr, y_train_pred_class))
test_conf_mat = pd.DataFrame(confusion_matrix(y_test, y_test_pred_class))

fig, axs = plt.subplots(1,2,figsize=(16, 4),sharey=True)
sns.set(font_scale=1)
f1 = sns.heatmap(train_conf_mat, annot=True,fmt="d", ax=axs[0], cbar = False ,cmap= "YlGnBu")

f1.set_title('Decision Tree Train Confusion Matrix With Positive Word2Vec')
f1.set_ylabel('True label')
f1.set_xlabel('Predicted label')
f1.xaxis.set_ticks_position('top')
f1.xaxis.set_label_position('top')

f2 = sns.heatmap(test_conf_mat, annot=True, fmt = 'd', ax=axs[1] , cmap = "YlGnBu")
f2.set_title('Decision Tree Test Confusion Matrix With Postive Word2Vec')
f2.set_ylabel('True label')
f2.set_xlabel('Predicted label')
f2.xaxis.set_ticks_position('top')
f2.xaxis.set_label_position('top')
plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.43161316915303194 for threshold 0.489
```



In [105]:

```
## prepare your data set to train your model
from scipy.sparse import hstack
only_pos_tr_w2v_with_new_ftr = hstack((only_pos_tfw2v_tr_feature,positive_tr, negative_tr,
neutral_tr, compund_tr, polarity_tr, subjectivity_tr)).tocsr()
only_pos_test_with_new_ftr = hstack((only_pos_tfw2v_test_feature,positive_test, negative_test,
neutral_test, compound_test, polarity_test, subjectivity_test)).tocsr()

print("Final Data matrix")
print(only_pos_tr_w2v_with_new_ftr.shape, y_tr.shape)
print(only_pos_test_with_new_ftr.shape, y_test.shape)
print("="*100)
```

```
Final Data matrix
(73196, 27) (73196,)
(36052, 27) (36052,)
====================================================================================================
```

In [106]:

```
%%time
### Train DT Classifier on positive feature

hyper_parameter = {'max_depth': [1, 5, 10, 50, 100 ], \
                   'min_samples_split': [5, 10, 100, 500]}

dt_clf = DecisionTreeClassifier(class_weight='balanced')
clf = GridSearchCV(dt_clf, hyper_parameter, cv=10, scoring='roc_auc', return_train_score=True, n_jo
bs=-1)
clf.fit(only_pos_tr_w2v_with_new_ftr,y_tr)
```

```
CPU times: user 5.92 s, sys: 346 ms, total: 6.27 s
Wall time: 11min 35s
```

In [107]:

```
train_auc= clf.cv_results_['mean_train_score']
train_auc_std = clf.cv_results_['std_train_score']
test_auc = clf.cv_results_['mean_test_score']
test_auc_std = clf.cv_results_['std_test_score']

#Output of GridSearchCV
print('Best score: ',clf.best_score_)
print('Best Hyper parameters: ',clf.best_params_)
print('='*75)
print('Train AUC scores')
print(clf.cv_results_['mean_train_score'])
print('CV AUC scores')
print(clf.cv_results_['mean_test_score'])
```

```
Best score:  0.6433128262311189
Best Hyper parameters:  {'max_depth': 10, 'min_samples_split': 500}
===========================================================================
Train AUC scores
```

```
[0.54904956 0.54904956 0.54904956 0.54904956 0.65944805 0.65944805
 0.65940254 0.65888791 0.77321901 0.77197741 0.75290072 0.71924853
 0.99962122 0.99471008 0.86083704 0.73944201 0.99962322 0.99474507
 0.8608338  0.73944201]
CV AUC scores
[0.54799432 0.54799432 0.54799432 0.54799432 0.64214957 0.64214957
 0.64211935 0.64256492 0.62517849 0.62557267 0.62941862 0.64331283
 0.53379982 0.54039708 0.59096164 0.63689826 0.53517906 0.5411029
 0.59068953 0.63690016]
```

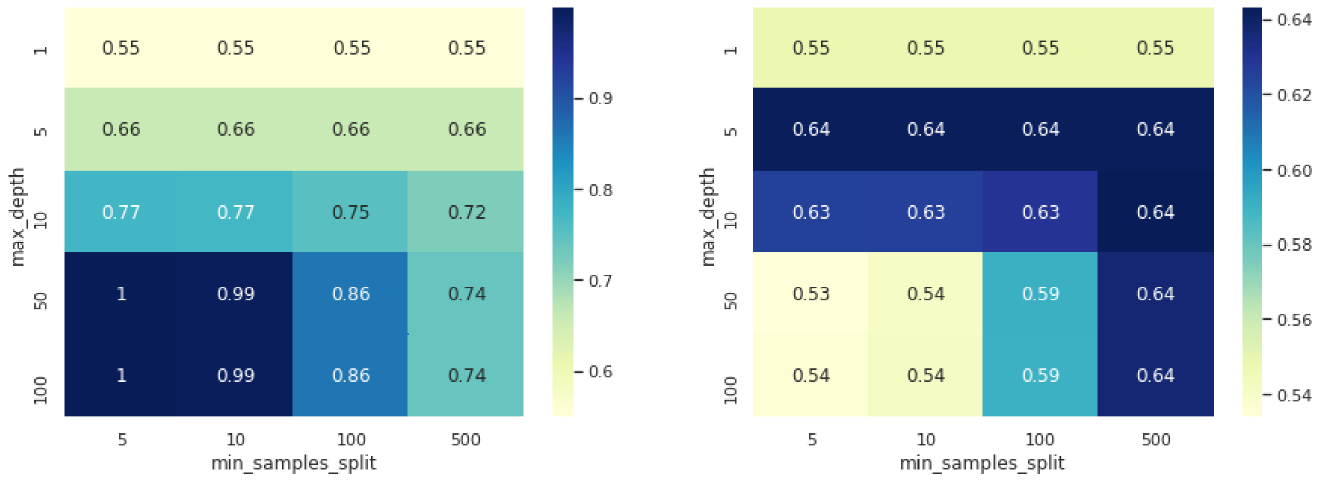
In [0]:

```python
from itertools import repeat
x1 = []
y1 = []
max_depth = [1, 5, 10, 50, 100 ]
min_samples_split =  [5, 10, 100, 500]
train_auc_scores = clf.cv_results_['mean_train_score']
cv_auc_scores = clf.cv_results_['mean_test_score']

x1 = [x for item in max_depth for x in repeat(item, 4)]
for _ in max_depth:
    for item in min_samples_split:
        y1.append(item)
```

In [109]:

```python
df1 = pd.DataFrame(list(zip(x1,y1,train_auc))).rename(columns = {0 : 'max_depth', 1:
'min_samples_split', 2: 'mean_train_score'}).pivot('max_depth','min_samples_split','mean_train_scor
e')
df2 = pd.DataFrame(list(zip(x1,y1,test_auc))).rename(columns = {0 : 'max_depth', 1:
'min_samples_split', 2: 'mean_CV_score'}).pivot('max_depth','min_samples_split','mean_CV_score')
plt.figure(figsize=(50,20))
f, axes = plt.subplots(1, 2,figsize=(15,5))
sns.heatmap(df1,annot=True,ax=axes[0],cbar = True ,cmap= "YlGnBu")
sns.heatmap(df2,annot=True,ax=axes[1],cbar = True ,cmap= "YlGnBu")
plt.show()
```

```
<Figure size 3600x1440 with 0 Axes>
```



In [113]:

```python
dt_clf = DecisionTreeClassifier(class_weight='balanced', max_depth = 10,
                               min_samples_split = 500)
dt_clf.fit(only_pos_tr_w2v_with_new_ftr,y_tr)
```

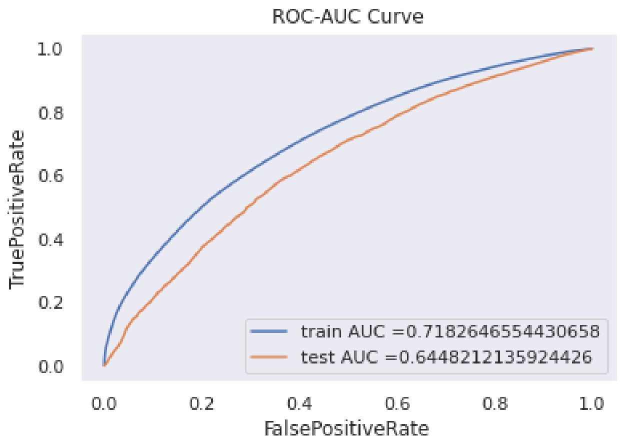
Out[113]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight='balanced', criterion='gini',
                       max_depth=10, 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='deprecated',
                       random_state=None, splitter='best')
```

```
                            random_state=None, splitter='best')
```

In [111]:

```
from sklearn.metrics import roc_curve, auc
y_train_pred = dt_clf.predict_proba(only_pos_tr_w2v_with_new_ftr)[::,1]
y_test_pred =  dt_clf.predict_proba(only_pos_test_with_new_ftr)[::,1]

train_fpr, train_tpr, tr_thresholds = roc_curve(y_tr, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)
auc_w2v_train = auc(train_fpr, train_tpr)
auc_w2v_test = auc(test_fpr, test_tpr)

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc_w2v_train))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc_w2v_test))
plt.legend()
plt.xlabel("FalsePositiveRate")

plt.ylabel("TruePositiveRate")

plt.title("ROC-AUC Curve")
plt.grid()
plt.show()
```



In [0]:

```
from prettytable import PrettyTable
```

In [116]:

```
model = ['TFIDF','TFIDF with Positive Features','TFIDF with new features','TFIDF-Word2Vec','Word2Vec with Positive Features','Word2Vec with new Features']
max_depth = [10,10,5,10,10,10]
min_split = [500,500,500,500,500,500]
train_auc = [0.70,0.70,0.70,0.66,0.71,0.71]
test_auc = [0.64,0.64,0.64,0.63,0.65,0.65]


x = PrettyTable()
x.field_names = ['Model','Max_Depth','Min_Split','Train_AUC',"Test_AUC"]
for i in range(6):
  x.add_row([model[i],max_depth[i],min_split[i],train_auc[i],test_auc[i]])

print(x)
```

```
+----------------------------------+-----------+-----------+-----------+----------+
|              Model               | Max_Depth | Min_Split | Train_AUC | Test_AUC |
+----------------------------------+-----------+-----------+-----------+----------+
|              TFIDF               |     10    |    500    |    0.7    |   0.64   |
|   TFIDF with Positive Features   |     10    |    500    |    0.7    |   0.64   |
|     TFIDF with new features      |     5     |    500    |    0.7    |   0.64   |
|          TFIDF-Word2Vec          |     10    |    500    |    0.66   |   0.63   |
| Word2Vec with Positive Features  |     10    |    500    |    0.71   |   0.65   |
|    Word2Vec with new Features    |     10    |    500    |    0.71   |   0.65   |
+----------------------------------+-----------+-----------+-----------+----------+
```

In [0]:

```

```