Assignment_DT_Instructions

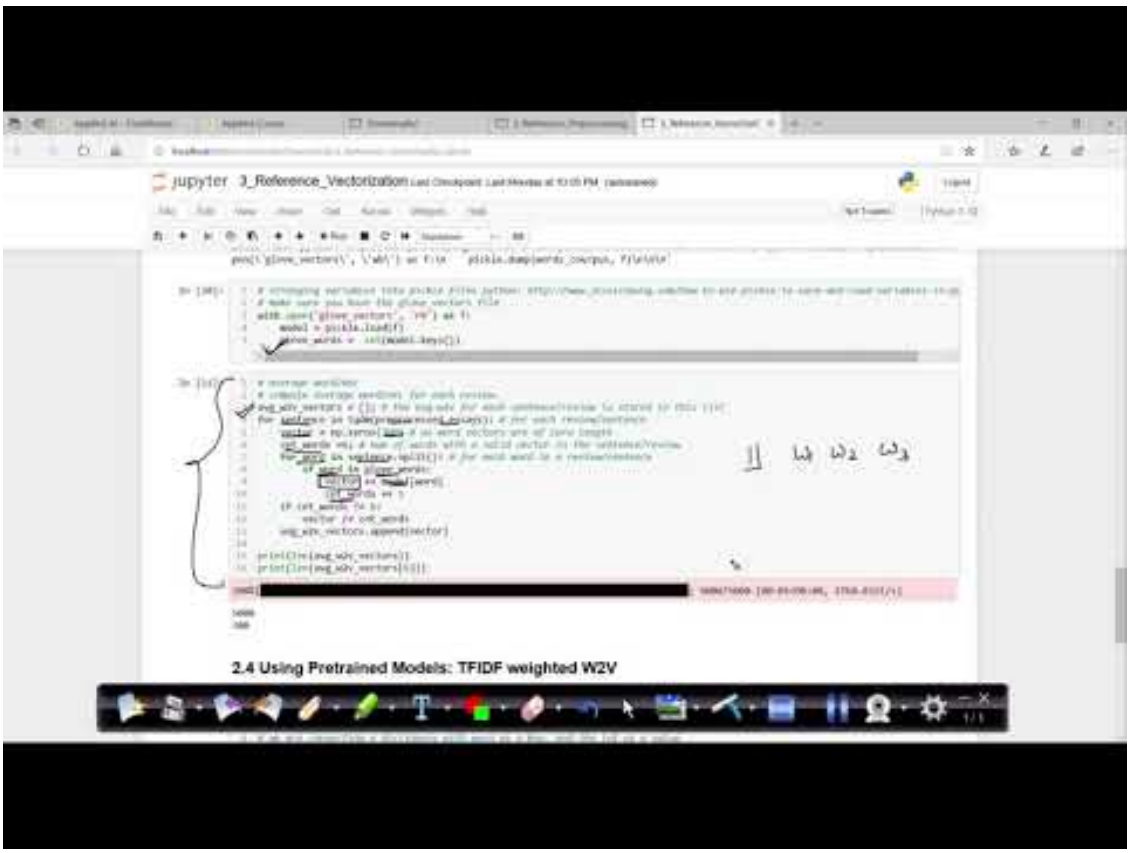
December 13, 2020

1 Assignment : DT

Please check below video before attempting this assignment

[ ]:
```
from IPython.display import YouTubeVideo
YouTubeVideo('ZhLXULFjIjQ', width="1000",height="500")
```

[ ]:



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-IDFW2Vfor more details.

Glove vectors

In this assignment you will be working with glove vectors , please check this and this for more details.

Download glove vectors from this link

[ ]:

[ ]:

or else , you can use below code

[ ]:
```
'''
# 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-use-pickle-to-save-and-load-variables-in-python/

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


'''
```

[ ]:
```
'\n# Reading glove vectors in python:
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",       le
n(inter_words),"(",np.round(len(inter_words)/len(words)*100,3),"%)")\n\nwords_co
urpus = {}\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# stronging variables into pickle files python:
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'
```

2 Task - 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)

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)

```
</ul>
</li>
<li>
<strong>Representation of results</strong>
    <ul>
<li>You need to plot the performance of model both on train data and cross validation data for
<img src='https://i.imgur.com/Gp2DQmh.jpg' width=500px> with X-axis as <strong>min_sample_split
        <p style="text-align:center;font-size:30px;color:red;"><strong>or</strong></p> <br>
<li>You need to plot the performance of model both on train data and cross validation data for
<img src='https://i.imgur.com/fgN9aUP.jpg' width=300px> <a href='https://seaborn.pydata.org/ge
<li>You choose either of the plotting techniques out of 3d plot or heat map</li>
<li>Once after you found the best hyper parameter, you need to train your model with it, and f
<img src='https://i.imgur.com/wMQDTFe.jpg' width=300px></li>
<li>Along with plotting ROC curve, you need to print the <a href='https://www.appliedaicourse.
<img src='https://i.imgur.com/IdN5Ctv.png' width=300px></li>
<li>Once after you plot the confusion matrix with the test data, get all the `false positive da
    <ul>
        <li> Plot the WordCloud(https://www.geeksforgeeks.org/generating-word-cloud-python/) w
        <li> Plot the box plot with the `price` of these `false positive data points`</li>
        <li> Plot the pdf with the `teacher_number_of_previously_posted_projects` of these `fa
    </ul>
    </ul>
</li>
```

[ ]:

2.1 SET-1

categorical, numerical features + preprocessed_essay (TFIDF) + Sentiment scores(preprocessed_essay)

[ ]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
import nltk
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
from sklearn.preprocessing import StandardScaler,MinMaxScaler
import pickle
from tqdm import tqdm
import os
'''
from plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
'''
import nltk
nltk.download('vader_lexicon')
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from collections import Counter
from scipy.sparse import hstack
from scipy.sparse import csr_matrix
from sklearn.tree import DecisionTreeClassifier
```

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

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[ ]:
```
import pandas as pd
data = pd.read_csv('/content/drive/MyDrive/AAIC/Assignments/11.Apply Decision␣
 ↪Trees on Donors Choose dataset/practice/preprocessed_data.csv',nrows=50000)
```

[ ]:

[ ]:
```
# train test split
y = data['project_is_approved'].values
x = data.drop(['project_is_approved'], axis=1)

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.33,␣
 ↪stratify=y)
```

```
#X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.
↪33, stratify=y_train)
```

[ ]:
```
x_train.columns
```

[ ]:
```
Index(['school_state', 'teacher_prefix', 'project_grade_category',
       'teacher_number_of_previously_posted_projects', 'clean_categories',
       'clean_subcategories', 'essay', 'price'],
      dtype='object')
```

[ ]:
```
def preprocess_cat(col,data):
  '''
  This function encodes the categorical features to one-hot-encoding.
  '''
  ohe_vectorizer = CountVectorizer(binary=True)
  a  = ohe_vectorizer.fit(data[col].values)
  aa = a.transform(data[col].values)
  b  = pd.DataFrame(aa.toarray(),columns=ohe_vectorizer.
↪get_feature_names(),index=list(data.index))
  return a,b


def preprocess_num(col,data):
  ''' It normalize's the numerical features.
  '''
  scaler = StandardScaler() #MinMaxScaler
  scaler.fit(data[col].values.reshape(-1, 1))
  a = pd.DataFrame(scaler.transform(data[col].values.reshape(-1,␣
↪1)),columns=[col],index=list(data.index))
  return scaler,a
```

[ ]:
```
def pre_process_num_cat(data,cat_feat,num_feat):
  '''
  This function takes categorical features, numerical features and encodes them␣
↪using above two functions defined.
  '''
  preprocessed_cat = list() # This list is used to store all the encodings of␣
↪categorical features.
  preprocessed_num = list() # This list is used to store all the encodings of␣
↪numerical features.
  cat_dict = dict() # This dict is used to store the vectorizer's of all the␣
↪categorical features.
  num_dict = dict() # This dict is used to store the vectorizer's of all the␣
↪numerical features.

  for i in (cat_feat): # Here,each categorical feature is encoded and its␣
↪corresponding vectorizer is stored.
    cat_dict[i], cat = preprocess_cat(i,data)
```

```
    preprocessed_cat.append(cat)

  for i in (num_feat):# Here,each numerical feature is encoded and its␣
 ↪corresponding vectorizer is stored.
    num_dict[i],num = preprocess_num(i,data)
    preprocessed_num.append(num)

  cat_feat = pd.concat(preprocessed_cat,axis=1) # concatening all the␣
 ↪cat-feat's
  num_feat = pd.concat(preprocessed_num,axis=1) # concatening all the␣
 ↪num-feat's

  # https://stackoverflow.com/questions/20459536/
 ↪convert-pandas-dataframe-to-sparse-numpy-matrix-directly
  cat_feat_csr = csr_matrix(cat_feat.values) # storing data in csr-format
  num_feat_csr = csr_matrix(num_feat.values)
  return cat_feat_csr,num_feat_csr,cat_dict,num_dict,cat_feat,num_feat

def senti_score(data,col): # computing sentiment-scores and storing in␣
 ↪csr-format.
  sid = SentimentIntensityAnalyzer()
  l = list()
  for i in data[col].values:
    l.append(sid.polarity_scores(i))
  senti_score_feat_csr = csr_matrix(pd.DataFrame(l).values)
  return senti_score_feat_csr,pd.DataFrame(l)
```

[ ]:
```
def pre_process_text(data): # text feature is preprocessed
  from sklearn.feature_extraction.text import TfidfVectorizer
  tfidf_vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4),␣
 ↪max_features=5000)
  tfidf_vectorizer.fit(data['essay'].values)
  text_feat = tfidf_vectorizer.transform(data['essay'].values)
  return text_feat,tfidf_vectorizer
```

[ ]:
```
cat_feat = ['school_state', 'teacher_prefix',␣
 ↪'project_grade_category','clean_categories', 'clean_subcategories']
num_feat = ['teacher_number_of_previously_posted_projects', 'price']

cat_feat_csr_tr,num_feat_csr_tr,cat_dict_vec_tr,num_dict_vec_tr,cat_feat_col,num_feat_col␣
 ↪= pre_process_num_cat(x_train,cat_feat,num_feat)
senti_score_feat_csr_tr,senti_score_feat = senti_score(x_train,'essay')
text_feat_tr,tfidf_vectorizer_tr = pre_process_text(x_train)

x_tr =␣
 ↪hstack((cat_feat_csr_tr,num_feat_csr_tr,text_feat_tr,senti_score_feat_csr_tr))
```

[ ]:
```
'''#pd.DataFrame(text_feat_tr.toarray(),columns=tfidf_vectorizer_tr.
 ↪get_feature_names())
features = list()
features.extend(cat_feat_col.columns)
features.extend(num_feat_col)
features.extend(senti_score_feat.columns)
features.extend(tfidf_vectorizer_tr.get_feature_names())
'''
```

[ ]:
```
def pre_process_test(data,cat_feat,num_feat,cat_dict_vec_tr,num_dict_vec_tr):
  '''
  Encoding the categorical and numerical features for test data points.
  '''
  preprocessed_cat = list() #This list is used to store the encoded categorical␣
 ↪features.
  preprocessed_num = list() #This list is used to store the encoded numerical␣
 ↪features.

  for i in (cat_feat):# using cat_dict_vec_tr, encoding/transforming the␣
 ↪categorical test data points.
    a = (cat_dict_vec_tr[i].transform(data[i].values)).toarray()
    aa = pd.DataFrame(a,columns=cat_dict_vec_tr[i].
 ↪get_feature_names(),index=list(data.index))
    preprocessed_cat.append(aa)

  for i in (num_feat):# using cat_dict_vec_tr, encoding/transforming the␣
 ↪numerical test data points.
    b = num_dict_vec_tr[i].transform(data[i].values.reshape(-1,1))
    bb = pd.DataFrame(b,columns=[i],index=list(data.index))
    preprocessed_num.append(bb)

  cat_feat = pd.concat(preprocessed_cat,axis=1) #concatenating all the␣
 ↪categorical features into single df.
  num_feat = pd.concat(preprocessed_num,axis=1) #concatenating all the␣
 ↪numerical features into single df.
  cat_feat_csr = csr_matrix(cat_feat.values)
  num_feat_csr = csr_matrix(num_feat.values)
  return cat_feat_csr,num_feat_csr
```

[ ]:
```
cat_feat_csr_ts,num_feat_csr_ts =␣
 ↪pre_process_test(x_test,cat_feat,num_feat,cat_dict_vec_tr,num_dict_vec_tr)
text_feat_ts = tfidf_vectorizer_tr.transform(x_test['essay'].values)
senti_score_feat_csr_ts,senti_score_feat_ts = senti_score(x_test,'essay')
x_ts =␣
 ↪hstack((cat_feat_csr_ts,num_feat_csr_ts,text_feat_ts,senti_score_feat_csr_ts))
```

[ ]:
```

```

[ ]:
```
clf = DecisionTreeClassifier()
param = {'max_depth':[1, 5, 10, 50],
         'min_samples_split':[5, 10, 100, 500]}
from sklearn.model_selection import GridSearchCV
model_ =␣
 ↪GridSearchCV(estimator=clf,param_grid=param,cv=10,return_train_score=True,verbose=2,n_jobs=
```

[ ]:
```
model_.fit(x_tr,y_train)
```

```
Fitting 10 folds for each of 16 candidates, totalling 160 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:   21.2s
[Parallel(n_jobs=-1)]: Done 160 out of 160 | elapsed: 23.0min finished
```

[ ]:
```
GridSearchCV(cv=10, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort='deprecated',
                                              random_state=None,
                                              splitter='best'),
             iid='deprecated', n_jobs=-1,
             param_grid={'max_depth': [1, 5, 10, 50],
                         'min_samples_split': [5, 10, 100, 500]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=2)
```

[ ]:
```
model_.best_params_
```

[ ]:
```
{'max_depth': 10, 'min_samples_split': 500}
```

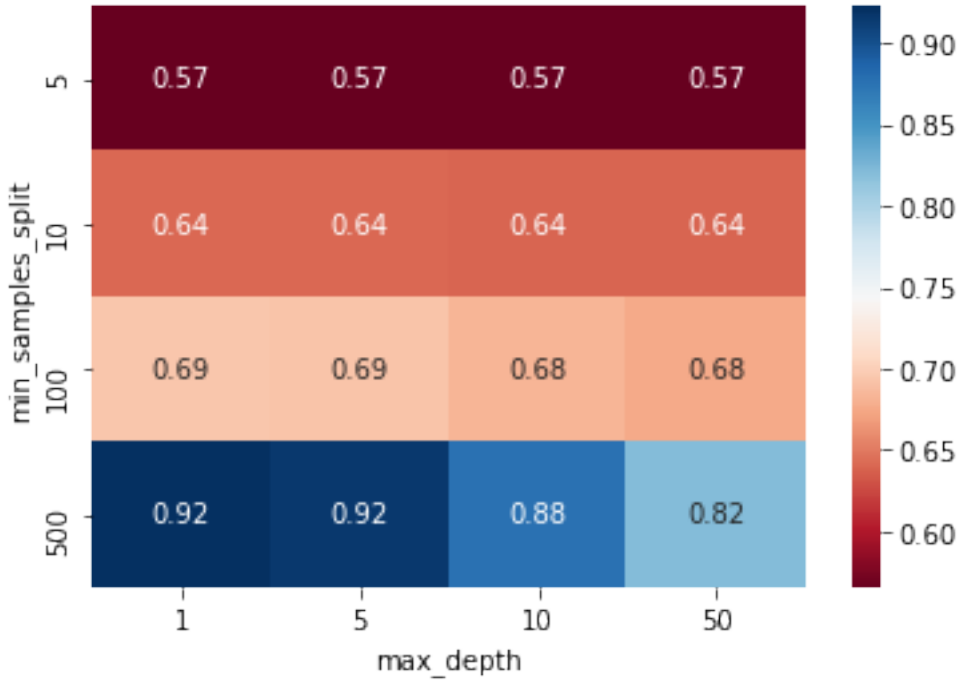
[ ]:
```
final_clf_ = DecisionTreeClassifier(max_depth=model_.best_params_['max_depth'],
                                    min_samples_split=model_.
 ↪best_params_['min_samples_split'])
final_clf_.fit(x_tr,y_train)
```

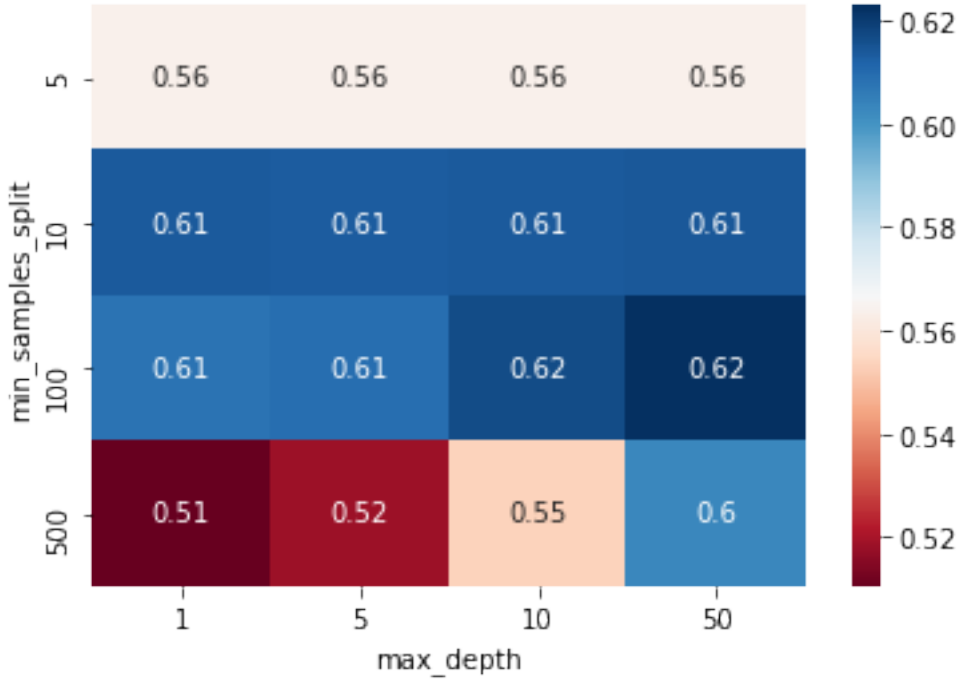
[ ]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=10, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=500,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

[ ]:
```
'''import plotly.express as px

fig = px.scatter_3d(plot_data,x='min_samples_split', y = 'max_depth', z = 'AUC_Train',color='tr_cl')
#fig = px.scatter_3d(plot_data,x='min_samples_split', y = 'max_depth', z = 'AUC_Test',color='ts_cl')
#fig = px.scatter(df, x="sepal_length", y="sepal_width", color="species",
#                 title="Automatic Labels Based on Data Frame Column Names")

fig.show()
'''
```

[ ]:
```

```

[ ]:
```
x,y,z = param['min_samples_split'],param['max_depth'],model_.cv_results_['mean_train_score']
import itertools
import seaborn as sns
plot_data = pd.DataFrame(list(itertools.product(x,y)),columns=['min_samples_split','max_depth'])
plot_data['AUC_Train'] = z
plot_data['AUC_Test'] = model_.cv_results_['mean_test_score']

#plot_data['tr_cl']= '#EF553B'
#plot_data['ts_cl']= '#FF6692'

# https://stackoverflow.com/questions/45470882/x-y-z-array-data-to-heatmap/45660022

pivotted_tr= plot_data.pivot('min_samples_split','max_depth','AUC_Train')
sns.heatmap(pivotted_tr,cmap='RdBu',annot=True)
plt.show()

pivotted_ts= plot_data.pivot('min_samples_split','max_depth','AUC_Test')
sns.heatmap(pivotted_ts,cmap='RdBu',annot=True)
plt.show()
```

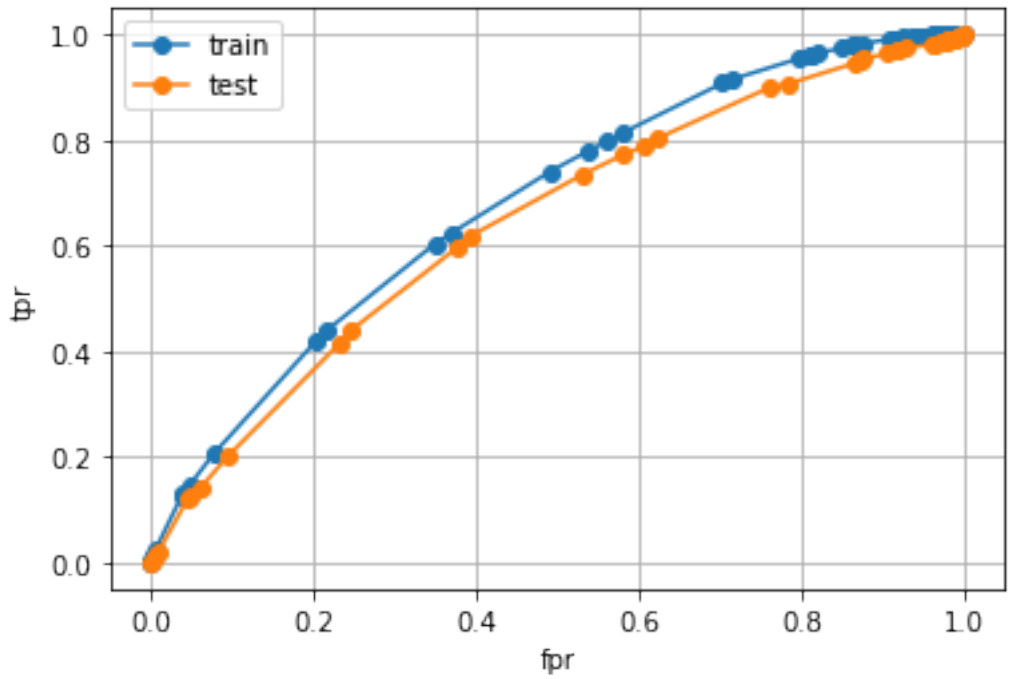




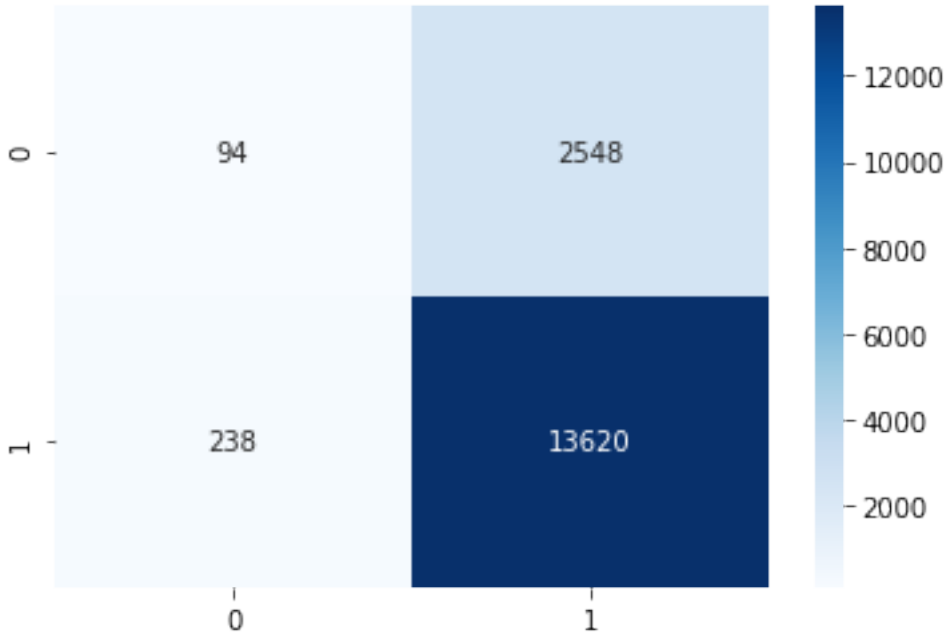
[79]:
```
import seaborn as sns
def plot_roc(model,x_train,x_test,y_train,y_test):
```

```
    tr_fpr, tr_tpr, tr_thresholds = roc_curve(y_train,model.
↪predict_proba(x_train)[:,1])
    ts_fpr, ts_tpr, ts_thresholds = roc_curve(y_test,model.predict_proba(x_test)[:
↪,1])
    plt.plot(tr_fpr,tr_tpr,'-o',label='train')
    plt.plot(ts_fpr,ts_tpr,'-o',label='test')
    plt.xlabel('fpr')
    plt.ylabel('tpr')
    plt.legend()
    plt.grid()
    plt.show()

    # https://stackoverflow.com/questions/61748441/
↪how-to-fix-the-values-displayed-in-a-confusion-matrix-in-exponential-form-to-nor
    cm = confusion_matrix(y_test,model.predict(x_test))
    sns.heatmap(cm, annot=True,fmt="d",cmap='Blues')
    return 0
```

[80]:
```
plot_roc(model=final_clf_,x_train=x_tr,x_test=x_ts,y_train=y_train,y_test=y_test)
```



[80]: 0



[81]:
```
x = pd.DataFrame(y_test,columns=['y_test'],index=list(x_test.index))
x['y_pred_0'] = final_clf_.predict_proba(x_ts)[:,0]
x['y_pred_1'] = final_clf_.predict_proba(x_ts)[:,1]
x['cl'] = final_clf_.predict(x_ts)

a=x[x['y_test']==1]
b = a[a['cl']==0]

fp_points = x_test[x_test.index.isin(b.index)]
```

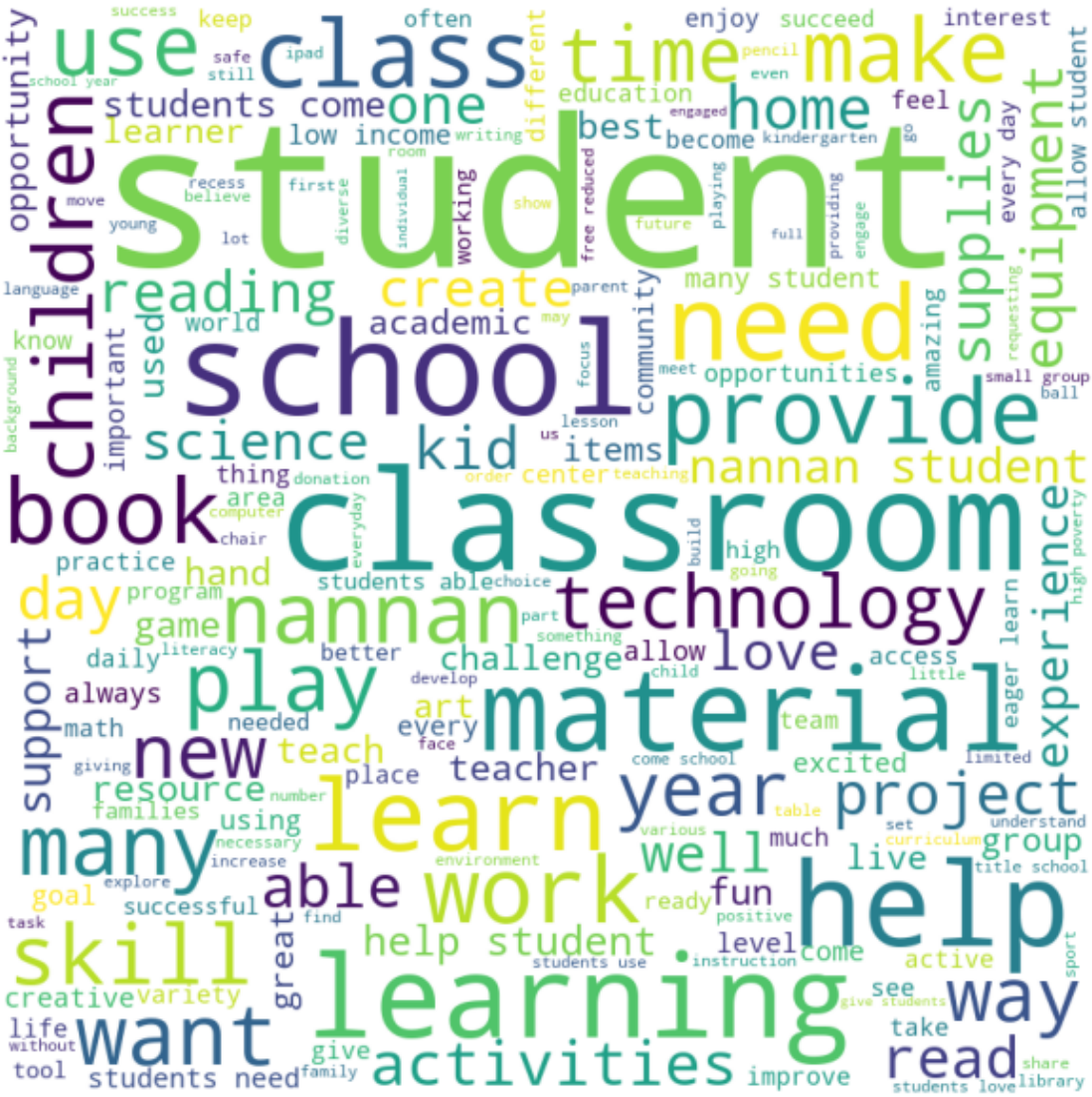
[ ]:
```
from sklearn.metrics import roc_auc_score,roc_curve
print('AUC_test',roc_auc_score(y_test,final_clf_.predict_proba(x_ts)[:,1]))
print('AUC_train',roc_auc_score(y_train,final_clf_.predict_proba(x_tr)[:,1]))
```

[ ]:
```
auc_test_tfidf = roc_auc_score(y_test,final_clf_.predict_proba(x_ts)[:,1])
```

[ ]:
```
y_pred_ts = final_clf_.predict(x_ts)
```

[82]:
```
# Python program to generate WordCloud

# importing all necessery modules
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
import pandas as pd

# Reads 'Youtube04-Eminem.csv' file
#df = pd.read_csv(r"Youtube04-Eminem.csv", encoding ="latin-1")
df = fp_points['essay']
```

```
comment_words = ''
stopwords = set(STOPWORDS)

# iterate through the csv file
for val in df.values:

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

        # split the value
        tokens = val.split()

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

        comment_words += " ".join(tokens)+" "

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

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

plt.show()
```



[ ]:

[ ]:

2.2 SET-2

[ ]:
```
'''
import pandas as pd
data = pd.read_csv('/content/drive/MyDrive/AAIC/Assignments/11.Apply Decision␣
↪Trees on Donors Choose dataset/practice/preprocessed_data.csv',nrows=50000)
# train test split
y = data['project_is_approved'].values
x = data.drop(['project_is_approved'], axis=1)
```

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.33,␣
 ↪stratify=y)
#X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.
 ↪33, stratify=y_train)
'''
```

[ ]:
```
"\nimport pandas as pd\ndata =
pd.read_csv('/content/drive/MyDrive/AAIC/Assignments/11.Apply Decision Trees on
Donors Choose dataset/practice/preprocessed_data.csv',nrows=50000)\n# train test
split\ny = data['project_is_approved'].values\nx =
data.drop(['project_is_approved'], axis=1)\n\nfrom sklearn.model_selection
import train_test_split\nx_train, x_test, y_train, y_test = train_test_split(x,
y, test_size=0.33, stratify=y)\n#X_train, X_cv, y_train, y_cv =
train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)\n"
```

[ ]:
```
#please use below code to load glove vectors
with open('/content/drive/MyDrive/AAIC/Assignments/11.Apply Decision Trees on␣
 ↪Donors Choose dataset/practice/glove_vectors', 'rb') as f:
    model_glove = pickle.load(f)
    glove_words =  set(model_glove.keys())
```

[ ]:
```
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())
```

[ ]:
```
def␣
 ↪tfidf_w2v(preprocessed_essays,glove_words,tfidf_words,dictionary,model_glove):
 ↪
  # average Word2Vec
  # compute average word2vec for each review.
  tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in␣
 ↪this list
  for sentence in tqdm(preprocessed_essays): # 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_glove[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 tfidf 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_vectors.append(vector)

  #print(len(tfidf_w2v_vectors))
  #print(len(tfidf_w2v_vectors[0]))
  return csr_matrix(tfidf_w2v_vectors)
```

[ ]:

[ ]:
```
tfidf_w2v_feat_tr = ␣
 ↪tfidf_w2v(preprocessed_essays,glove_words,tfidf_words,dictionary,model_glove)
x_tr_tfidf = ␣
 ↪hstack((cat_feat_csr_tr,num_feat_csr_tr,tfidf_w2v_feat_tr,senti_score_feat_csr_tr))
```

```
100%|| 33500/33500 [01:09<00:00, 482.00it/s]
```

[ ]:
```
preprocessed_essays = x_test['essay'].values
tfidf_w2v_feat_ts = ␣
 ↪tfidf_w2v(preprocessed_essays,glove_words,tfidf_words,dictionary,model_glove)
x_ts_tfidf = ␣
 ↪hstack((cat_feat_csr_ts,num_feat_csr_ts,text_feat_ts,senti_score_feat_csr_ts))
```

```
100%|| 16500/16500 [00:33<00:00, 489.18it/s]
```

[ ]:
```
clf = DecisionTreeClassifier()
param = {'max_depth':[1, 5, 10, 50],
         'min_samples_split':[5, 10, 100, 500]}
from sklearn.model_selection import GridSearchCV
model = ␣
 ↪GridSearchCV(estimator=clf,param_grid=param,cv=10,return_train_score=True,verbose=2,n_jobs=
model.fit(x_tr_tfidf,y_train)
```

```
Fitting 10 folds for each of 16 candidates, totalling 160 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:   51.0s
[Parallel(n_jobs=-1)]: Done 160 out of 160 | elapsed: 42.2min finished
```

[ ]:
```
GridSearchCV(cv=10, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
```

```
                                                min_impurity_split=None,
                                                min_samples_leaf=1,
                                                min_samples_split=2,
                                                min_weight_fraction_leaf=0.0,
                                                presort='deprecated',
                                                random_state=None,
                                                splitter='best'),
             iid='deprecated', n_jobs=-1,
             param_grid={'max_depth': [1, 5, 10, 50],
                         'min_samples_split': [5, 10, 100, 500]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=2)
```

[57]:
```
final_clf = DecisionTreeClassifier(max_depth=model.best_params_['max_depth'],
                                   min_samples_split=model.
 ↪best_params_['min_samples_split'])
final_clf.fit(x_tr,y_train)
```

[57]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, 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=5,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

[58]:
```
from sklearn.metrics import roc_auc_score,roc_curve
print('AUC_test',roc_auc_score(y_test,final_clf.predict_proba(x_ts)[:,1]))
print('AUC_train',roc_auc_score(y_train,final_clf.predict_proba(x_tr)[:,1]))
```

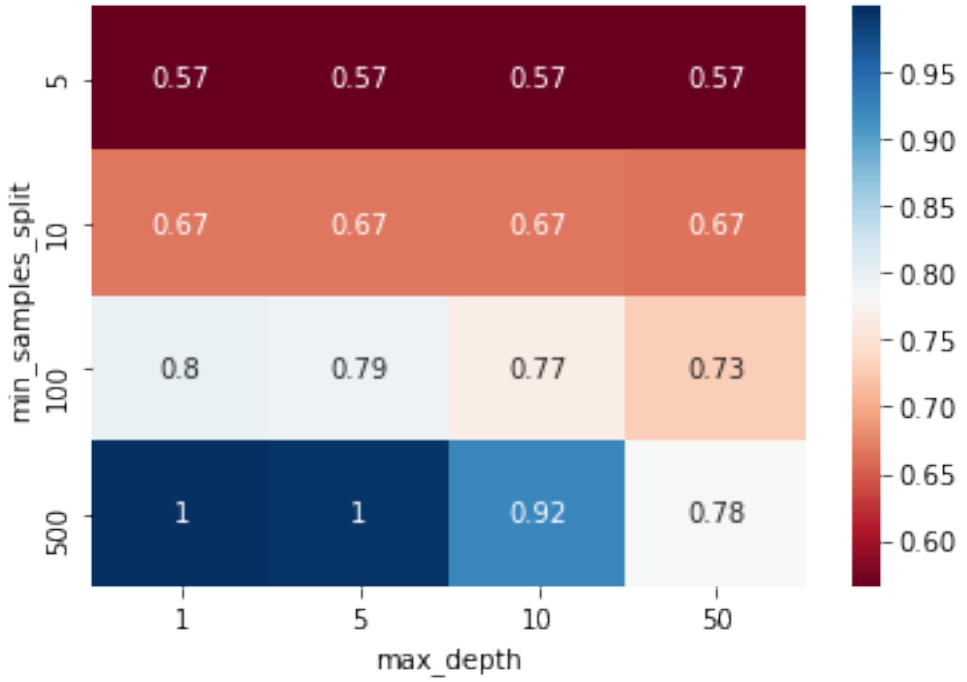
```
AUC_test 0.625197690230825
AUC_train 0.6406833813339392
```

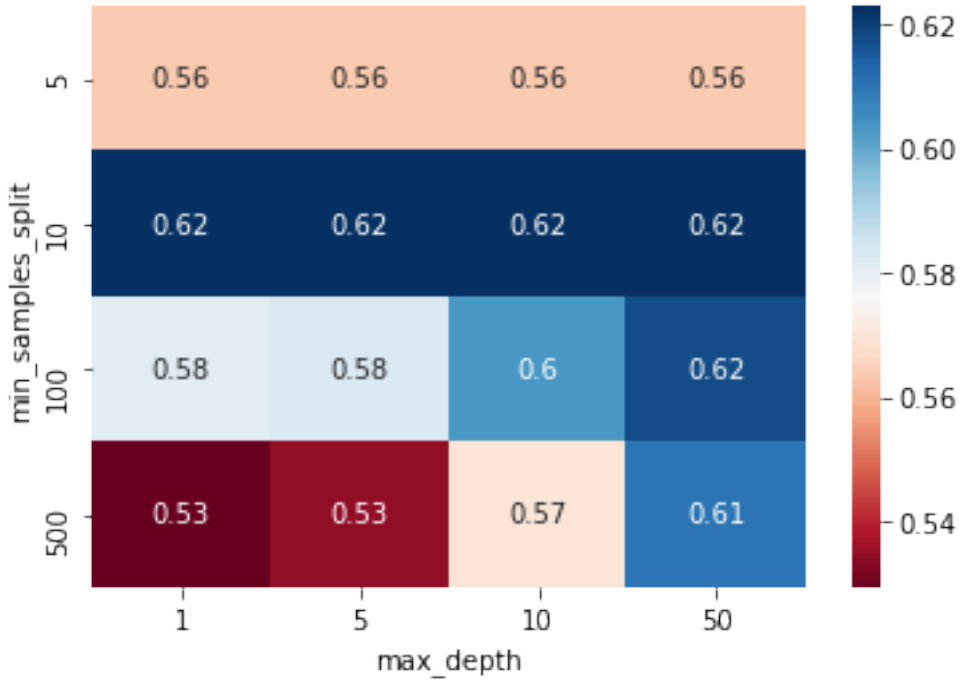
[59]:
```
auc_test_tfidf_w2v = roc_auc_score(y_test,final_clf.predict_proba(x_ts)[:,1])
```

[ ]:
```

```

[63]:
```
import numpy as np
import pandas as pd

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

x,y,z = param['min_samples_split'],param['max_depth'],model.
 ↪cv_results_['mean_train_score']
import itertools
plot_data = pd.DataFrame(list(itertools.
 ↪product(x,y)),columns=['min_samples_split','max_depth'])
plot_data['AUC_Train'] = z
plot_data['AUC_Test'] = model.cv_results_['mean_test_score']
```

```
#plot_data['tr_cl']= '#EF553B'
#plot_data['ts_cl']= '#FF6692'

# https://stackoverflow.com/questions/45470882/x-y-z-array-data-to-heatmap/
 ↪45660022

pivotted= plot_data.pivot('min_samples_split','max_depth','AUC_Train')
import seaborn as sns
sns.heatmap(pivotted,cmap='RdBu',annot=True,)
plt.show()
print('\n')
pivotted_ts= plot_data.pivot('min_samples_split','max_depth','AUC_Test')
import seaborn as sns
sns.heatmap(pivotted_ts,cmap='RdBu',annot=True,)
plt.show()
```

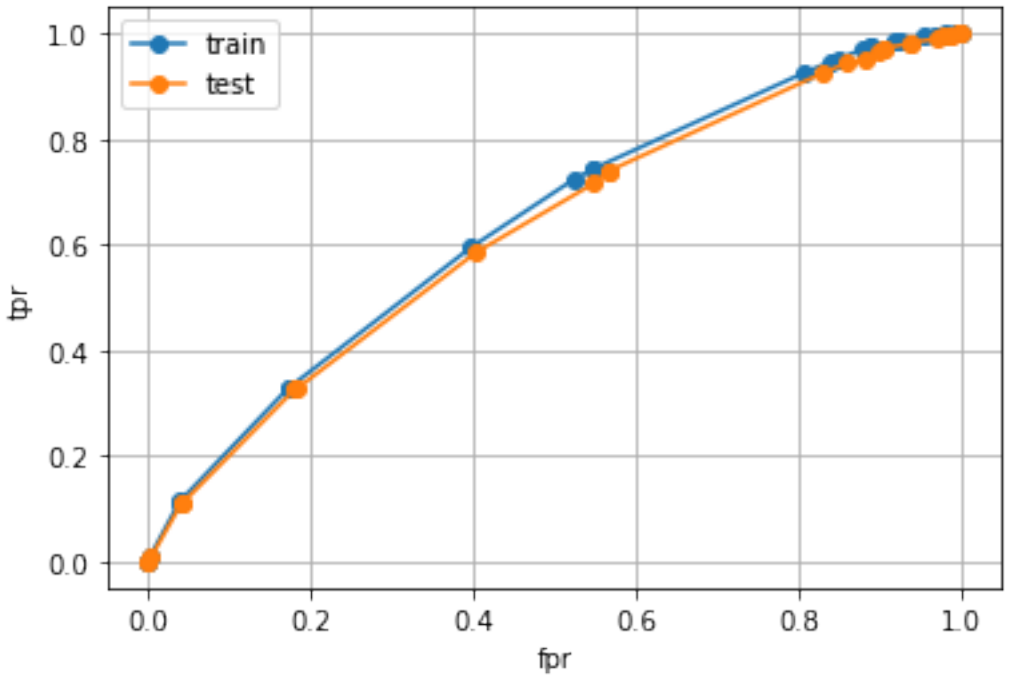




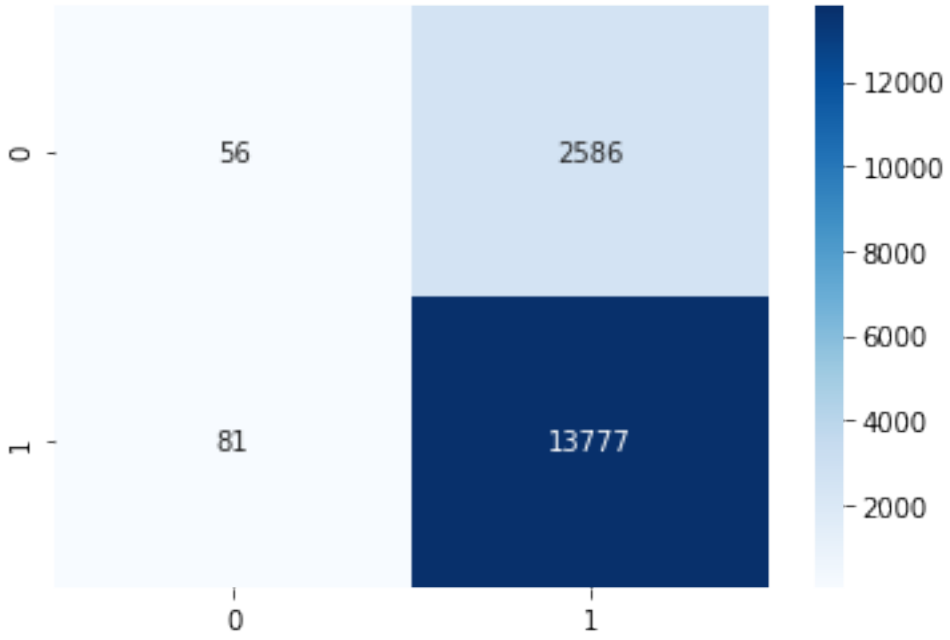
[ ]:

[ ]:

[64]:
```
plot_roc(model=final_clf,x_train=x_tr,x_test=x_ts,y_train=y_train,y_test=y_test)
```



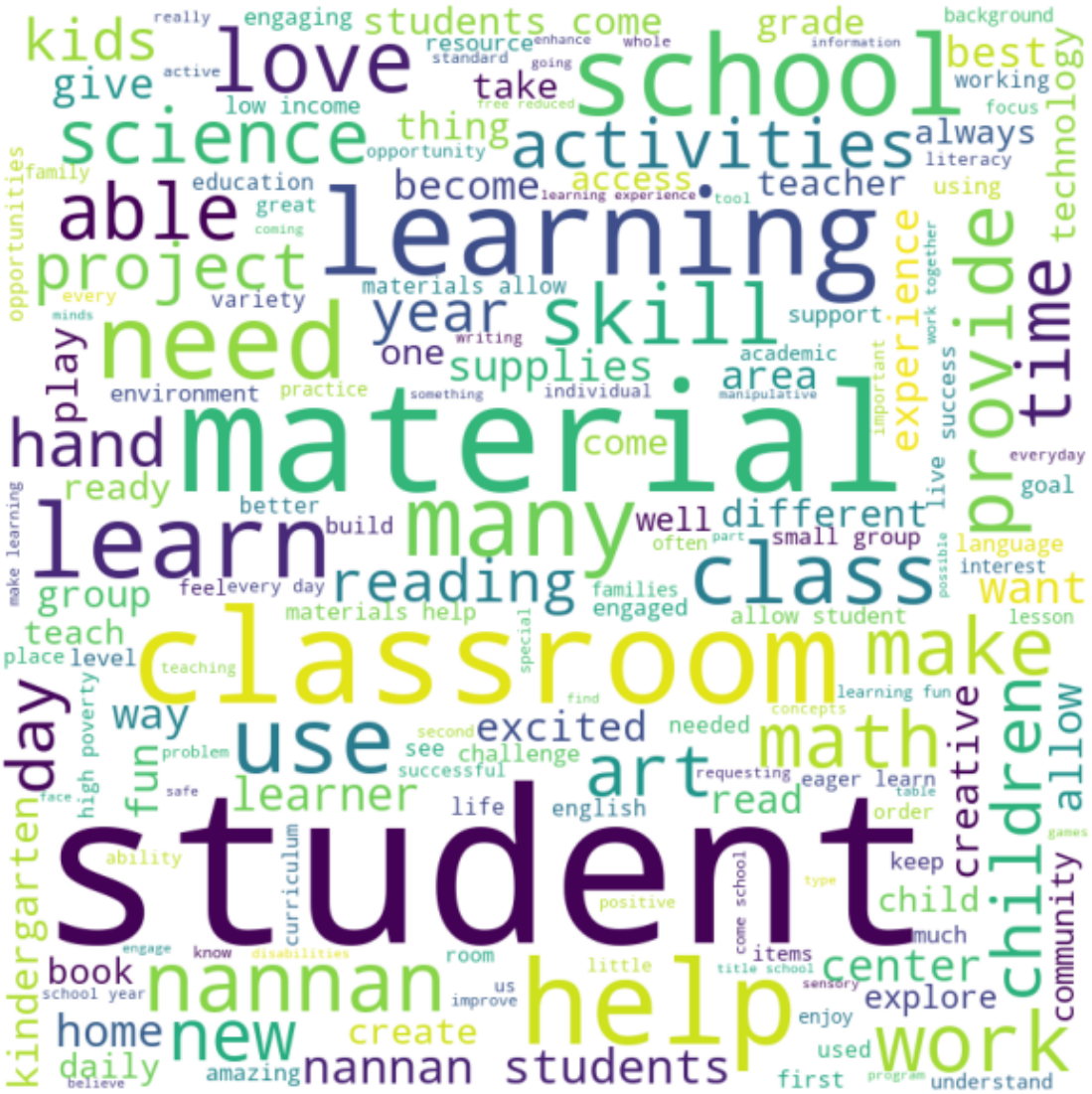
[64]: 0



[65]:
```
x = pd.DataFrame(y_test,columns=['y_test'],index=list(x_test.index))
x['y_pred_0'] = final_clf.predict_proba(x_ts)[:,0]
x['y_pred_1'] = final_clf.predict_proba(x_ts)[:,1]
x['cl'] = final_clf.predict(x_ts)

a=x[x['y_test']==1]
b = a[a['cl']==0]

fp_points = x_test[x_test.index.isin(b.index)]
```

[66]:
```
# Python program to generate WordCloud

# importing all necessery modules
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt
import pandas as pd

# Reads 'Youtube04-Eminem.csv' file
#df = pd.read_csv(r"Youtube04-Eminem.csv", encoding ="latin-1")
df = fp_points['essay']

comment_words = ''
stopwords = set(STOPWORDS)
```

```
# iterate through the csv file
for val in df.values:

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

        # split the value
        tokens = val.split()

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

        comment_words += " ".join(tokens)+" "

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

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

plt.show()
```



[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

3 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.
  You need to summarize the results at the end of the notebook, summarize it in the table format

[68]:
```
#pd.DataFrame(text_feat_tr.toarray(),columns=tfidf_vectorizer_tr.
↪get_feature_names())
features = list()
features.extend(cat_feat_col.columns)
features.extend(num_feat_col.columns)
features.extend(senti_score_feat.columns)
features.extend(tfidf_vectorizer_tr.get_feature_names())
```

[69]:
```
fi = pd.DataFrame(final_clf_.feature_importances_)
features_index = list(fi[fi>0].dropna().index)
top_feat = list()
for i in features_index:
  #print(features[i])
  top_feat.append(features[i])
```

[70]:
```
final_df = pd.DataFrame(x_tr.toarray(),columns=features)[top_feat]
```

[71]:
```
clf_2 = DecisionTreeClassifier()
param = {'max_depth':[1, 5, 10, 50],
         'min_samples_split':[5, 10, 100, 500]}
from sklearn.model_selection import GridSearchCV
model_2 =␣
↪GridSearchCV(estimator=clf_2,param_grid=param,cv=10,return_train_score=True,verbose=2,n_job
```

[72]:
```
model_2.fit(csr_matrix(final_df),y_train)
```

```
Fitting 10 folds for each of 16 candidates, totalling 160 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  48 tasks      | elapsed:    4.6s
[Parallel(n_jobs=-1)]: Done 160 out of 160 | elapsed:  1.4min finished
```

[72]:
```
GridSearchCV(cv=10, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
```

```
                                                min_impurity_split=None,
                                                min_samples_leaf=1,
                                                min_samples_split=2,
                                                min_weight_fraction_leaf=0.0,
                                                presort='deprecated',
                                                random_state=None,
                                                splitter='best'),
             iid='deprecated', n_jobs=-1,
             param_grid={'max_depth': [1, 5, 10, 50],
                         'min_samples_split': [5, 10, 100, 500]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='roc_auc', verbose=2)
```

[73]:
```
model_2.best_params_
```

[73]:
```
{'max_depth': 50, 'min_samples_split': 500}
```

[75]:
```
final_clf_2 = DecisionTreeClassifier(max_depth=model_2.
↪best_params_['max_depth'],
                                     min_samples_split=model_2.
↪best_params_['min_samples_split'])
final_clf_2.fit(x_tr,y_train)
```

[75]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=50, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=500,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

[76]:
```
auc_test_tfidf_2 = roc_auc_score(y_test,final_clf_2.predict_proba(x_ts)[:,1])
```

[77]:
```
#https://stackoverflow.com/questions/9535954/printing-lists-as-tabular-data
from prettytable import PrettyTable
t = PrettyTable(['Vectorizer', 'Model' , 'Hyper parameter' , 'AUC'])
#print(clf.best_estimator_,'\n',auc_test)
t.add_row(['TFIDF', 'DT',(model_.best_params_),auc_test_tfidf])
t.add_row(['TFIDF_W2V', 'DT',(model.best_params_),auc_test_tfidf_w2v])
t.add_row(['TFIDF', 'DT',(model_2.best_params_),auc_test_tfidf_2])
print(t)
```

```
+------------+-------+-----------------------------------------------+------------
-------+
| Vectorizer | Model |               Hyper parameter                 |        AUC
|
+------------+-------+-----------------------------------------------+------------
-------+
|   TFIDF    |   DT  | {'max_depth': 10, 'min_samples_split': 500} |
0.645068412619006 |
| TFIDF_W2V  |   DT  |   {'max_depth': 5, 'min_samples_split': 5}  |
```

```
0.625197690230825 |
|   TFIDF    |   DT  | {'max_depth': 50, 'min_samples_split': 500} |
0.623210982618227 |
+------------+-------+---------------------------------------------+------------
-------+
```

[ ]: