Assignment : DT

TF-IDFW2V

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

(Optional) Please check course video on AVgw2V and TF-IDFW2V for more details.

Glove vectors

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

Download glove vectors from this link

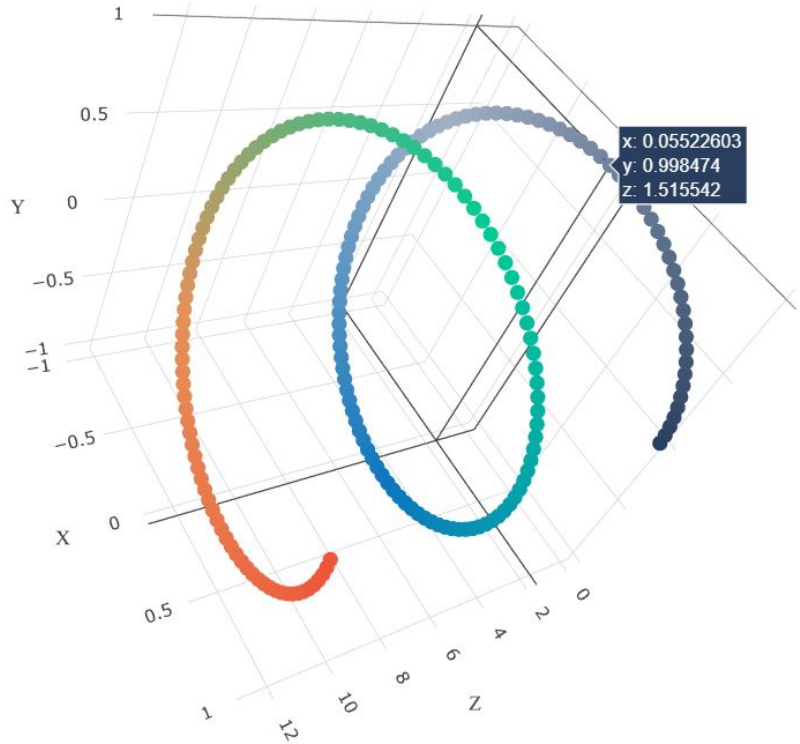
In [116…

```
#please use below code to load glove vectors
# with open('glove_vectors', 'rb') as f:
#     model = pickle.load(f)
#     glove_words =  set(model.keys())
import pickle
store = None

def pickleLoad():
    return pickle.load(open("glove_vectors","rb" ) )

store = pickleLoad()
glove_words =  set(store.keys())
```

Task - 1

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

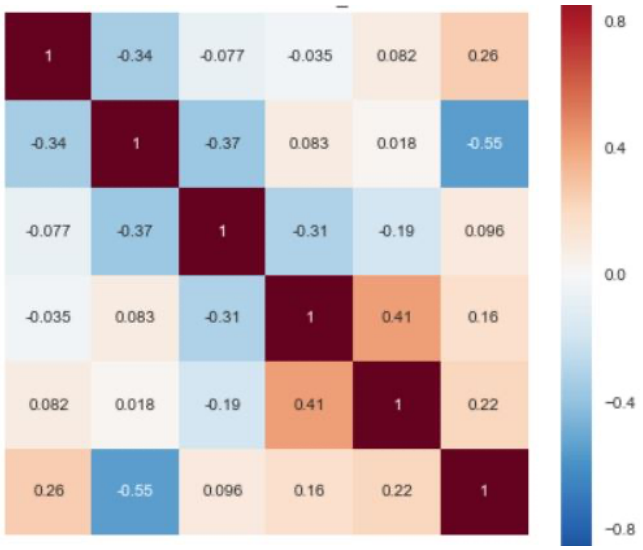
- **Set 1: categorical, numerical features + preprocessed_essay (TFIDF) + Sentiment scores(preprocessed_essay)**
- **Set 2: categorical, numerical features + preprocessed_essay (TFIDF W2V) + Sentiment scores(preprocessed_essay)**
  ** **
- **The hyper paramter tuning (best $depth$ in range [1, 5, 10, 50], and the best $\min_{s} amp \leq s_s plit$ 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)**
    ** **
  - **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*

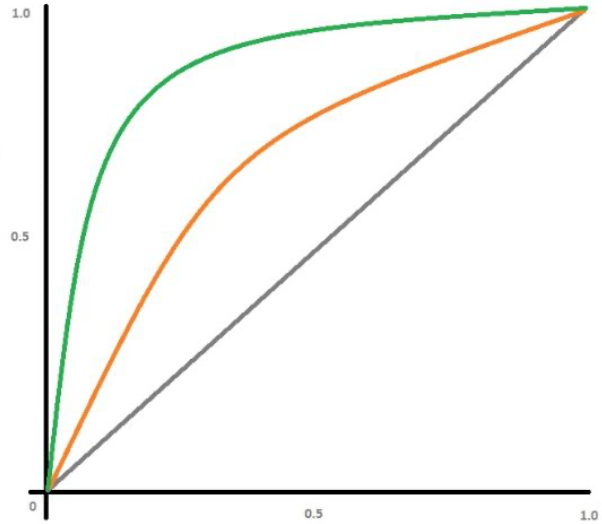
or

- **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 min_sample_split, 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**

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

data points

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

** **

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
  <img src='http://i.imgur.com/YVpIGGE.jpg' width=400px>
  ** **

Hint for calculating Sentiment scores

In [2]:
```
import nltk
nltk.download('vader_lexicon')
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     /Users/poojashah/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

Out[2]: True

In [3]:
```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer

# import nltk
# nltk.download('vader_lexicon')

sid = SentimentIntensityAnalyzer()

for_sentiment = 'a person is a person no matter how small dr seuss i teach the smallest students with the biggest
for learning my students learn in many different ways using all of our senses and multiple intelligences i use a w
of techniques to help all my students succeed students in my class come from a variety of different backgrounds wh
for wonderful sharing of experiences and cultures including native americans our school is a caring community of s
learners which can be seen through collaborative student project based learning in and out of the classroom kinder
in my class love to work with hands on materials and have many different opportunities to practice a skill before
mastered having the social skills to work cooperatively with friends is a crucial aspect of the kindergarten curri
montana is the perfect place to learn about agriculture and nutrition my students love to role play in our pretend
in the early childhood classroom i have had several kids ask me can we try cooking with real food i will take thei
and create common core cooking lessons where we learn important math and writing concepts while cooking delicious
food for snack time my students will have a grounded appreciation for the work that went into making the food and
of where the ingredients came from as well as how it is healthy for their bodies this project would expand our lea
nutrition and agricultural cooking recipes by having us peel our own apples to make homemade applesauce make our o
and mix up healthy plants from our classroom garden in the spring we will also create our own cookbooks to be prin
shared with families students will gain math and literature skills as well as a life long enjoyment for healthy co
nannan'
ss = sid.polarity_scores(for_sentiment)

for k in ss:
    print('{0}: {1}, '.format(k, ss[k]), end='')

# we can use these 4 things as features/attributes (neg, neu, pos, compound)
# neg: 0.0, neu: 0.753, pos: 0.247, compound: 0.93
```

```
neg: 0.01, neu: 0.745, pos: 0.245, compound: 0.9975,
```

1. Decision Tree

1.1 Loading Data

In [4]:
```
import pandas as pd
import numpy as np
from tqdm import tqdm
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import StandardScaler
from scipy.sparse import hstack
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
import warnings
warnings.filterwarnings("ignore")
```

In [5]:
```
data = pd.read_csv('preprocessed_data.csv')
Y = data['project_is_approved'].values
X = data.drop(['project_is_approved'], axis=1)
X.head(1)
```

Out[5]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clean_subcatego
0	ca	mrs	grades_prek_2	53	math_science	appliedscien health_lifescie

In [6]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state = 0)
X_train, X_cv, Y_train, Y_cv = train_test_split(X_train, Y_train, test_size=0.3)
```

In [7]:
```
print(len(X_train))
print(len(X_cv))
print(len(X_test))
```

```
53531
22942
32775
```

In [8]:
```
data.columns
```

Out[8]:
```
Index(['school_state', 'teacher_prefix', 'project_grade_category',
       'teacher_number_of_previously_posted_projects', 'project_is_approved',
       'clean_categories', 'clean_subcategories', 'essay', 'price'],
      dtype='object')
```

Encoding Categorical features

In [9]:
```
vectorizer2 = CountVectorizer()
vectorizer2.fit(X_train['school_state'].values)
train_state = vectorizer2.transform(X_train['school_state'].values)
cv_state = vectorizer2.transform(X_cv['school_state'].values)
test_state = vectorizer2.transform(X_test['school_state'].values)
print(train_state.shape)
print(cv_state.shape)
print(test_state.shape)
print(vectorizer2.get_feature_names())
```

```
(53531, 51)
(22942, 51)
(32775, 51)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia', 'id', 'il', 'in', 'ks', 'ky', 'la',
'ma', 'md', 'me', 'mi', 'mn', 'mo', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', 'nv', 'ny', 'oh', 'ok', 'or',
'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv', 'wy']
```

In [10]:
```
vectorizer3 = CountVectorizer()
vectorizer3.fit(X_train['teacher_prefix'].values)
train_teacher_prefix = vectorizer3.transform(X_train['teacher_prefix'].values)
cv_teacher_prefix = vectorizer3.transform(X_cv['teacher_prefix'].values)
test_teacher_prefix = vectorizer3.transform(X_test['teacher_prefix'].values)
print(train_teacher_prefix.shape)
print(cv_teacher_prefix.shape)
print(test_teacher_prefix.shape)
print(vectorizer3.get_feature_names())
```

```
(53531, 5)
(22942, 5)
(32775, 5)
['dr', 'mr', 'mrs', 'ms', 'teacher']
```

In [11]:
```
vectorizer4 = CountVectorizer()
vectorizer4.fit(X_train['clean_categories'].values)
train_cat = vectorizer4.transform(X_train['clean_categories'].values)
cv_cat = vectorizer4.transform(X_cv['clean_categories'].values)
test_cat = vectorizer4.transform(X_test['clean_categories'].values)
print(train_cat.shape)
print(cv_cat.shape)
print(test_cat.shape)
print(vectorizer4.get_feature_names())
```

```
(53531, 9)
(22942, 9)
(32775, 9)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language', 'math_science', 'music_
arts', 'specialneeds', 'warmth']
```

In [12]:
```
vectorizer5 = CountVectorizer()
vectorizer5.fit(X_train['clean_subcategories'].values)
train_subcat = vectorizer5.transform(X_train['clean_subcategories'].values)
cv_subcat = vectorizer5.transform(X_cv['clean_subcategories'].values)
test_subcat = vectorizer5.transform(X_test['clean_subcategories'].values)
print(train_subcat.shape)
print(cv_subcat.shape)
print(test_subcat.shape)
print(vectorizer5.get_feature_names())
```

```
(53531, 30)
(22942, 30)
(32775, 30)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government', 'college_careerprep', 'communityserv
ice', 'earlydevelopment', 'economics', 'environmentalscience', 'esl', 'extracurricular', 'financialliteracy', 'for
eignlanguages', 'gym_fitness', 'health_lifescience', 'health_wellness', 'history_geography', 'literacy', 'literatu
re_writing', 'mathematics', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performingarts', 'social
sciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
```

In [13]:
```
vectorizer6 = CountVectorizer()
vectorizer6.fit(X_train['project_grade_category'].values)
train_grade = vectorizer6.transform(X_train['project_grade_category'].values)
cv_grade = vectorizer6.transform(X_cv['project_grade_category'].values)
test_grade = vectorizer6.transform(X_test['project_grade_category'].values)
print(train_grade.shape)
print(cv_grade.shape)
print(test_grade.shape)
print(vectorizer6.get_feature_names())
```

```
(53531, 4)
(22942, 4)
(32775, 4)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
```

Encoding Numerical features

In [14]:
```
previous_project_scalar = StandardScaler()
previous_project_scalar.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1)) # findin
print(f"Mean : {previous_project_scalar.mean_[0]}, Standard deviation : {np.sqrt(previous_project_scalar.var_[0])}

train_prPos_norm = previous_project_scalar.transform(X_train['teacher_number_of_previously_posted_projects'].value
cv_prPos_norm = previous_project_scalar.transform(X_cv['teacher_number_of_previously_posted_projects'].values.resh
test_prPos_norm = previous_project_scalar.transform(X_test['teacher_number_of_previously_posted_projects'].values.

print("After vectorizations")
print(train_prPos_norm.shape, Y_train.shape)
print(cv_prPos_norm.shape, Y_cv.shape)
print(test_prPos_norm.shape, Y_test.shape)
```

```
Mean : 11.106592441762716, Standard deviation : 27.516634107287434
After vectorizations
(53531, 1) (53531,)
(22942, 1) (22942,)
(32775, 1) (32775,)
```

In [15]:
```
from sklearn.preprocessing import StandardScaler

price_scalar = StandardScaler()
price_scalar.fit(X_train['price'].values.reshape(-1,1)) # finding the mean and standard deviation of this data
print(f"Mean : {price_scalar.mean_[0]}, Standard deviation : {np.sqrt(price_scalar.var_[0])}")

train_scaler_price = price_scalar.transform(X_train['price'].values.reshape(-1, 1))
cv_scaler_price = price_scalar.transform(X_cv['price'].values.reshape(-1, 1))
test_scaler_price = price_scalar.transform(X_test['price'].values.reshape(-1, 1))

print("After vectorizations")
print(train_scaler_price.shape, Y_train.shape)
print(cv_scaler_price.shape, Y_train.shape)
print(test_scaler_price.shape, Y_test.shape)
```

```
Mean : 297.82742803235504, Standard deviation : 360.23740722009467
After vectorizations
(53531, 1) (53531,)
(22942, 1) (53531,)
(32775, 1) (32775,)
```

Encoding essay

In [16]:
```
X_train_essay = X_train['essay']
X_cv_essay = X_cv['essay']
X_test_essay = X_test['essay']
```

TF-IDF Featurization

In [17]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer1 = TfidfVectorizer(min_df=10, ngram_range=(1,3))
vectorizer1.fit(X_train_essay)
train_essay_tfidf = vectorizer1.transform(X_train_essay)
cv_essay_tfidf = vectorizer1.transform(X_cv_essay)
test_essay_tfidf = vectorizer1.transform(X_test_essay)
print(train_essay_tfidf.shape)
print(cv_essay_tfidf.shape)
print(test_essay_tfidf.shape)
print(Y_train.shape, Y_cv.shape, Y_test.shape)
```

```
(53531, 165313)
(22942, 165313)
(32775, 165313)
(53531,) (22942,) (32775,)
```

Using TF-IDF W2V

In [18]:
```
tfidf_model = TfidfVectorizer()
tfidf_model.fit(X_train_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 [19]:
```
def tfidf_w2v(words):
    tfidf_w2v_vectors = []
    for sentence in tqdm(words): # 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 = store[word] # getting the vector for each word
                # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(s
                tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value f
                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 tfidf_w2v_vectors
```

In [20]:
```
train_essay_tfidf_w2v = tfidf_w2v(X_train_essay)
cv_essay_tfidf_w2v = tfidf_w2v(X_cv_essay)
test_essay_tfidf_w2v = tfidf_w2v(X_test_essay)
```

```
100%|██████████| 53531/53531 [01:16<00:00, 702.87it/s]
  1%|          | 135/22942 [00:00<00:35, 651.21it/s]
53531
300
100%|██████████| 22942/22942 [00:32<00:00, 710.44it/s]
  0%|          | 137/32775 [00:00<00:48, 671.51it/s]
22942
300
100%|██████████| 32775/32775 [00:46<00:00, 704.79it/s]
32775
300
```

In [21]:
```
import nltk
nltk.download('vader_lexicon')
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     /Users/poojashah/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

In [22]:
```
sid = SentimentIntensityAnalyzer()

sentiment_pos=[]
sentiment_neg=[]
sentiment_neu=[]
sentiment_com=[]

for i in X_train['essay']:
    ss = sid.polarity_scores(i)
    sentiment_pos.append(ss['pos'])
    sentiment_neg.append(ss['neg'])
    sentiment_neu.append(ss['neu'])
    sentiment_com.append(ss['compound'])
```

In [23]:
```
X_train['sentiment_train_pos'] = sentiment_pos
sentiment_pos = X_train['sentiment_train_pos'].values.reshape(-1,1)
X_train['sentiment_train_neg'] = sentiment_neg
sentiment_neg = X_train['sentiment_train_neg'].values.reshape(-1,1)
X_train['sentiment_train_neu'] = sentiment_neu
sentiment_neu = X_train['sentiment_train_neu'].values.reshape(-1,1)
X_train['sentiment_train_compound'] = sentiment_com
sentiment_com = X_train['sentiment_train_compound'].values.reshape(-1,1)
```

In [24]:
```
# sid = SentimentIntensityAnalyzer()

sentiment_pos_cv=[]
sentiment_neg_cv=[]
sentiment_neu_cv=[]
sentiment_com_cv=[]

for i in X_cv['essay']:
    ss = sid.polarity_scores(i)
    sentiment_pos_cv.append(ss['pos'])
    sentiment_neg_cv.append(ss['neg'])
    sentiment_neu_cv.append(ss['neu'])
    sentiment_com_cv.append(ss['compound'])
```

In [25]:
```
X_cv['sentiment_pos_cv'] = sentiment_pos_cv
sentiment_pos_cv = X_cv['sentiment_pos_cv'].values.reshape(-1,1)
X_cv['sentiment_neg_cv'] = sentiment_neg_cv
sentiment_neg_cv = X_cv['sentiment_neg_cv'].values.reshape(-1,1)
X_cv['sentiment_neu_cv'] = sentiment_neu_cv
sentiment_neu_cv = X_cv['sentiment_neu_cv'].values.reshape(-1,1)
X_cv['sentiment_com_cv'] = sentiment_com_cv
sentiment_com_cv = X_cv['sentiment_com_cv'].values.reshape(-1,1)
```

In [26]:
```
# sid = SentimentIntensityAnalyzer()

sentiment_pos_test=[]
sentiment_neg_test=[]
sentiment_neu_test=[]
sentiment_com_test=[]

for i in X_test['essay']:
    ss = sid.polarity_scores(i)
    sentiment_pos_test.append(ss['pos'])
    sentiment_neg_test.append(ss['neg'])
    sentiment_neu_test.append(ss['neu'])
    sentiment_com_test.append(ss['compound'])
```

In [27]:
```
X_test['sentiment_pos_test'] = sentiment_pos_test
sentiment_pos_test = X_test['sentiment_pos_test'].values.reshape(-1,1)
X_test['sentiment_neg_test'] = sentiment_neg_test
sentiment_neg_test = X_test['sentiment_neg_test'].values.reshape(-1,1)
X_test['sentiment_neu_test'] = sentiment_neu_test
sentiment_neu_test = X_test['sentiment_neu_test'].values.reshape(-1,1)
X_test['sentiment_com_test'] = sentiment_com_test
sentiment_com_test = X_test['sentiment_com_test'].values.reshape(-1,1)
```

Stacking all vectors

Set 1

In [28]:
```
X_set1_train = []
X_set1_train = hstack((train_essay_tfidf, train_teacher_prefix, train_cat, train_subcat,
                       train_grade, train_state, train_scaler_price, train_prPos_norm,
                       sentiment_pos, sentiment_neg, sentiment_neu, sentiment_com))
print(X_set1_train.shape, Y_train.shape)
```

```
(53531, 165418) (53531,)
```

In [29]:
```
X_set1_cv = []
X_set1_cv = hstack((cv_essay_tfidf, cv_teacher_prefix, cv_cat, cv_subcat,
                    cv_grade, cv_state, cv_scaler_price, cv_prPos_norm,
                    sentiment_pos_cv, sentiment_neg_cv, sentiment_neu_cv, sentiment_com_cv))
print(X_set1_cv.shape, Y_cv.shape)
```

```
(22942, 165418) (22942,)
```

In [30]:
```
X_set1_test = []
X_set1_test = hstack((test_essay_tfidf, test_teacher_prefix, test_cat, test_subcat,
                    test_grade, test_state, test_scaler_price, test_prPos_norm,
                    sentiment_pos_test, sentiment_neg_test, sentiment_neu_test, sentiment_com_test))
print(X_set1_test.shape, Y_test.shape)
```

```
(32775, 165418) (32775,)
```

Set 2

In [31]:
```
X_set2_train = []
X_set2_train = hstack((train_essay_tfidf_w2v, train_teacher_prefix, train_cat, train_subcat,
                    train_grade, train_state, train_scaler_price, train_prPos_norm,
                    sentiment_pos, sentiment_neg, sentiment_neu, sentiment_com))
print(X_set2_train.shape, Y_train.shape)
```

```
(53531, 405) (53531,)
```

In [32]:
```
X_set2_cv = hstack((cv_essay_tfidf_w2v, cv_teacher_prefix, cv_cat, cv_subcat,
                    cv_grade, cv_state, cv_scaler_price, cv_prPos_norm,
                    sentiment_pos_cv, sentiment_neg_cv, sentiment_neu_cv, sentiment_com_cv))
print(X_set2_cv.shape, Y_cv.shape)
```

```
(22942, 405) (22942,)
```

In [33]:
```
X_set2_test = hstack((test_essay_tfidf_w2v, test_teacher_prefix, test_cat, test_subcat,
                    test_grade, test_state, test_scaler_price, test_prPos_norm,
                    sentiment_pos_test, sentiment_neg_test, sentiment_neu_test, sentiment_com_test))
print(X_set2_test.shape, Y_test.shape)
```

```
(32775, 405) (32775,)
```

Decision Tree on TF-IDF using set 1

In [34]:
```
dt1 = DecisionTreeClassifier(class_weight = 'balanced')
parameters = {'max_depth': [1, 5, 10, 50], 'min_samples_split': [5, 10, 100, 500]}
clf1 = GridSearchCV(dt1, parameters, cv=3, scoring='roc_auc',return_train_score=True)
clf1 = clf1.fit(X_set1_train, Y_train)
```

In [35]:
```
clf_v1 = DecisionTreeClassifier (class_weight = 'balanced',max_depth=None,min_samples_split=500)
clf_v1.fit(X_set1_train, Y_train)
```

Out[35]:
```
DecisionTreeClassifier(class_weight='balanced', min_samples_split=500)
```

In [50]:
```
print(clf1.best_estimator_)
#Mean cross-validated score of the best_estimator
print(clf1.score(X_set1_train,Y_train))
print(clf1.score(X_set1_test,Y_test))
pd.DataFrame(clf1.cv_results_).head(2)
test_results = clf1.cv_results_
```

```
DecisionTreeClassifier(class_weight='balanced', max_depth=10,
                       min_samples_split=500)
0.7128266022957498
0.6514689638995421
```

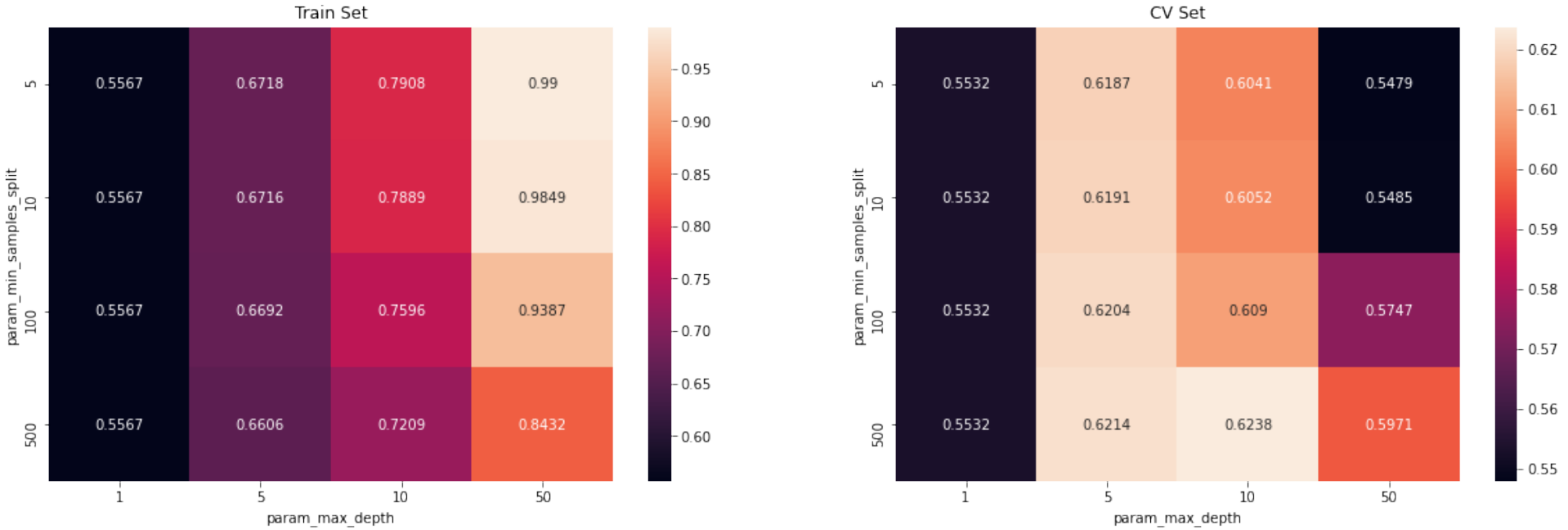
In [51]:
```
clf1.fit(X_set1_cv, Y_cv)
print(clf1.best_estimator_)
#Mean cross-validated score of the best_estimator
print(clf1.score(X_set1_cv,Y_cv))
print(clf1.score(X_set1_test,Y_test))
pd.DataFrame(clf1.cv_results_).head(2)
cv_results = clf1.cv_results_
```

```
DecisionTreeClassifier(class_weight='balanced', max_depth=10,
                       min_samples_split=500)
0.7168915226371841
0.6220331601525269
```

Heatmap on test and cv data, based on max depth, min sample split and AUC scores

In [119…

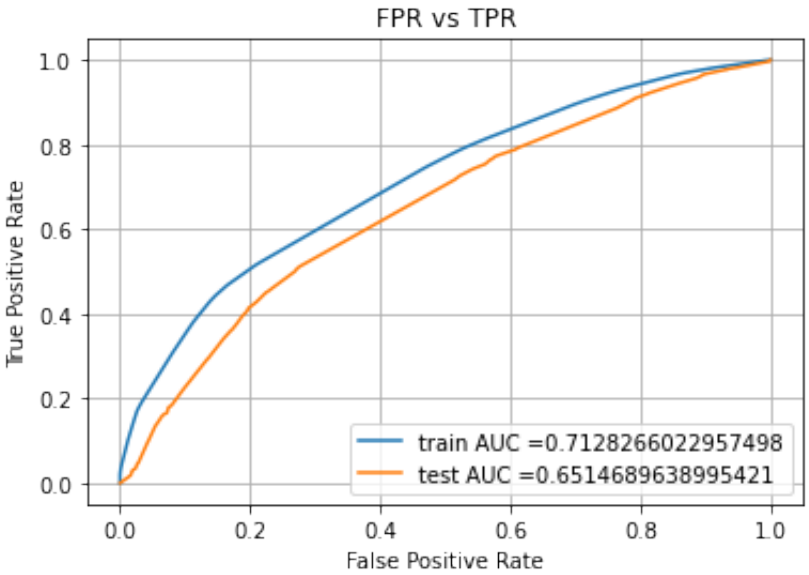
```
import seaborn as sns
max_scores1 = pd.DataFrame(clf1.cv_results_).groupby(['param_min_samples_split', 'param_max_depth']).max().unstack
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores1.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores1.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```



ROC Curve Of Test and Train Data

In [53]:
```
hyper_parameters=[{'max_depth':[10], 'min_samples_split':[500] }]
```

In [54]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

train_auc = []
test_auc = []

clf_s1 = GridSearchCV( DecisionTreeClassifier(class_weight = 'balanced'),hyper_parameters)
clf_s1.fit(X_set1_train, Y_train)
# clf_v1 = DecisionTreeClassifier (class_weight = 'balanced',max_depth=10,min_samples_split=500)

Y_train_pred1 = clf_s1.predict_proba(X_set1_train)[:, 1]
Y_test_pred1 = clf_s1.predict_proba(X_set1_test)[:, 1]

train_fpr, train_tpr, tr_thresholds = roc_curve(Y_train, Y_train_pred1)
test_fpr, test_tpr, te_thresholds = roc_curve(Y_test, Y_test_pred1)

auc1=auc(test_fpr, test_tpr)

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



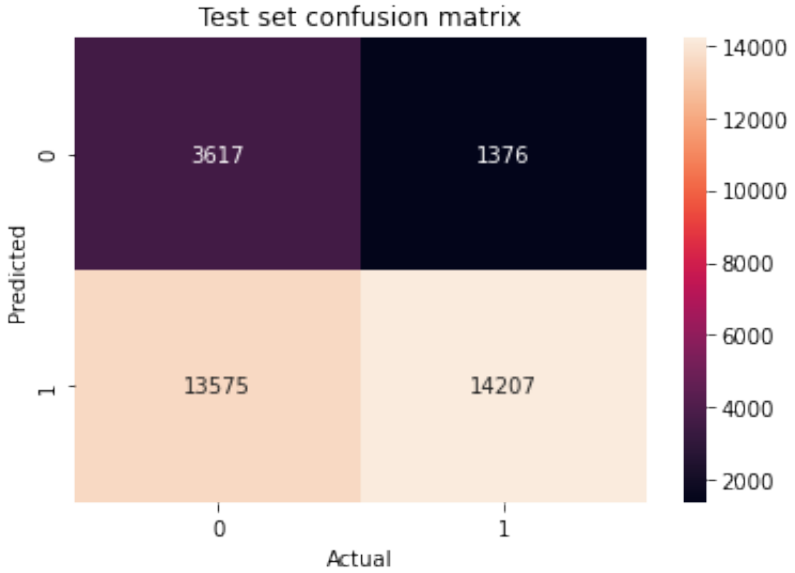
Confusion Matrix On Test Data

In [55]:
```
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[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, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

In [56]:
```
import numpy as np
print("="*100)
from sklearn.metrics import confusion_matrix
best_t1 = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(Y_train, predict_with_best_t(Y_train_pred1, best_t1)))
```

```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.4096054533158911 for threshold 0.479
Train confusion matrix
[[ 6261  1733]
 [21722 23815]]
```

In [57]:
```
import seaborn as sns
heatmap_train = sns.heatmap(confusion_matrix(Y_test, predict_with_best_t(Y_test_pred1, best_t1)), annot=True, fmt=
plt.title("Test set confusion matrix")
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.show()
```



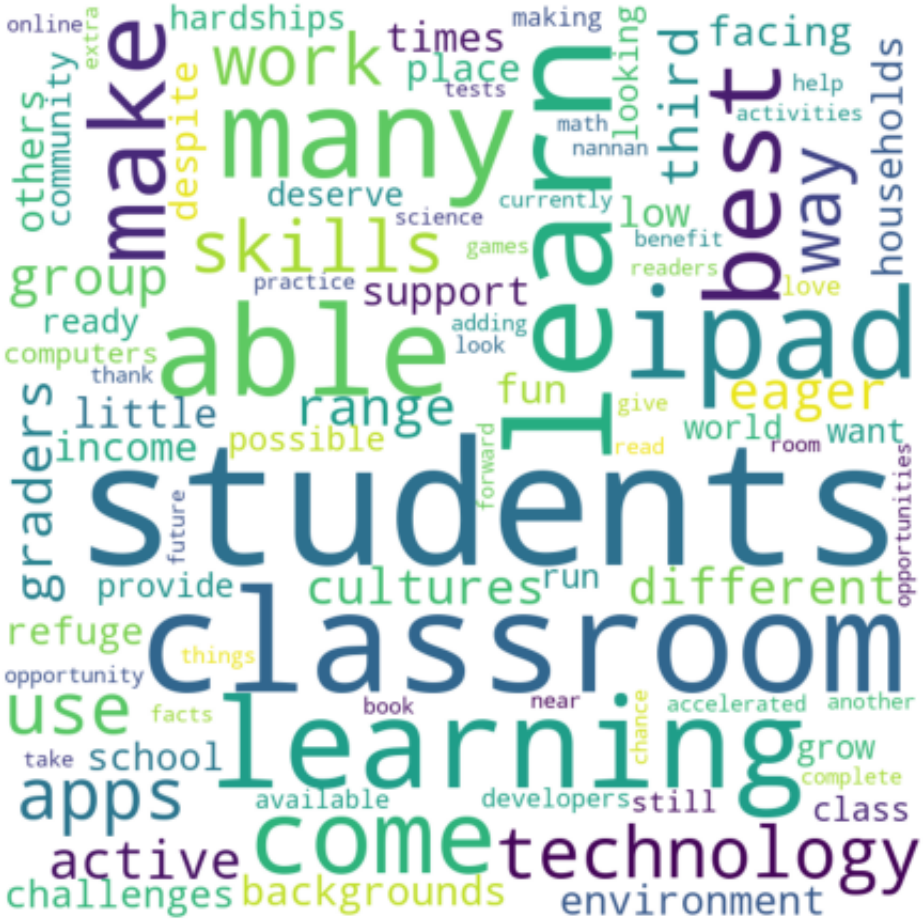
Word Cloud on False Positives

In [58]:
```
pred1 = predict_with_best_t(Y_test_pred1, best_t1)
print(Y_test.shape)
print(len(pred1))
```

```
(32775,)
32775
```

In [59]:
```
fp = []
for i in range(len(Y_test)) :
  if (Y_test[i] == 0) & (pred1[i] == 1) :
    fp.append(i)

fp_essay1 = []
for i in fp :
  fp_essay1.append(X_test['essay'].values[i])
```

In [60]:
```
from wordcloud import WordCloud, STOPWORDS

comment_words = ' '
stopwords = set(STOPWORDS)
for val in fp_essay1 :
  val = str(val)
  tokens = val.split()
for i in range(len(tokens)):
  tokens[i] = tokens[i].lower()
for words in tokens :
  comment_words = comment_words + words + ' '
wordcloud = WordCloud(width = 800, height = 800, background_color ='white', stopwords = stopwords,min_font_size =

plt.figure(figsize = (6, 6), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```



Decision Tree on TF-IDF W2V using set 2

In [61]:
```
dt2 = DecisionTreeClassifier(class_weight = 'balanced')
parameters = {'max_depth': [1, 5, 10, 50], 'min_samples_split': [5, 10, 100, 500]}
clf2 = GridSearchCV(dt2, parameters, cv=3, scoring='roc_auc',return_train_score=True)
se2 = clf2.fit(X_set2_train, Y_train)
```

In [62]:
```
print(clf2.best_estimator_)
#Mean cross-validated score of the best_estimator
print(clf2.score(X_set2_train,Y_train))
print(clf2.score(X_set2_test,Y_test))
pd.DataFrame(clf2.cv_results_).head(2)
test_results = clf2.cv_results_
```

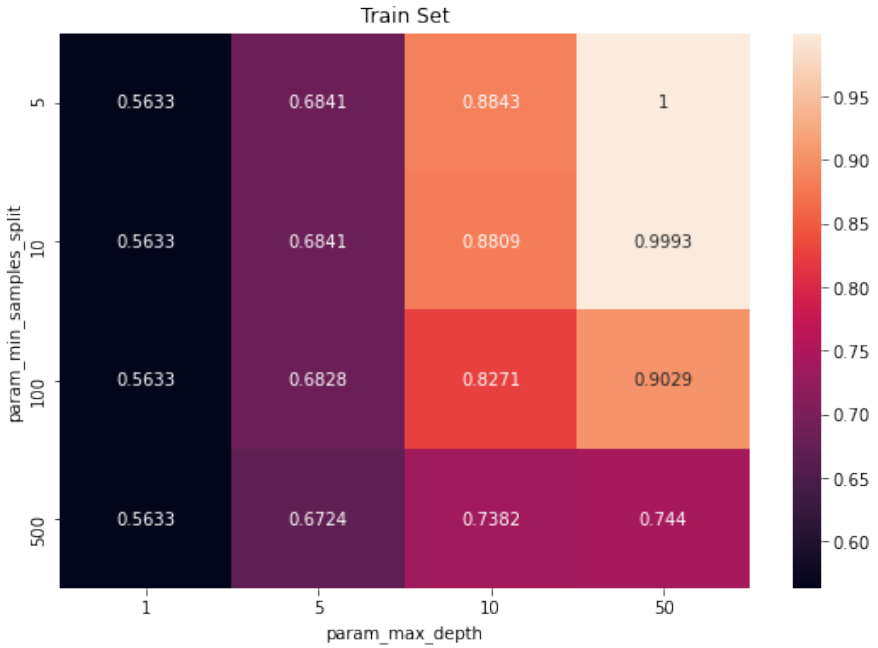
```
DecisionTreeClassifier(class_weight='balanced', max_depth=5,
                       min_samples_split=500)
0.6553870689377576
0.6303019677840533
```

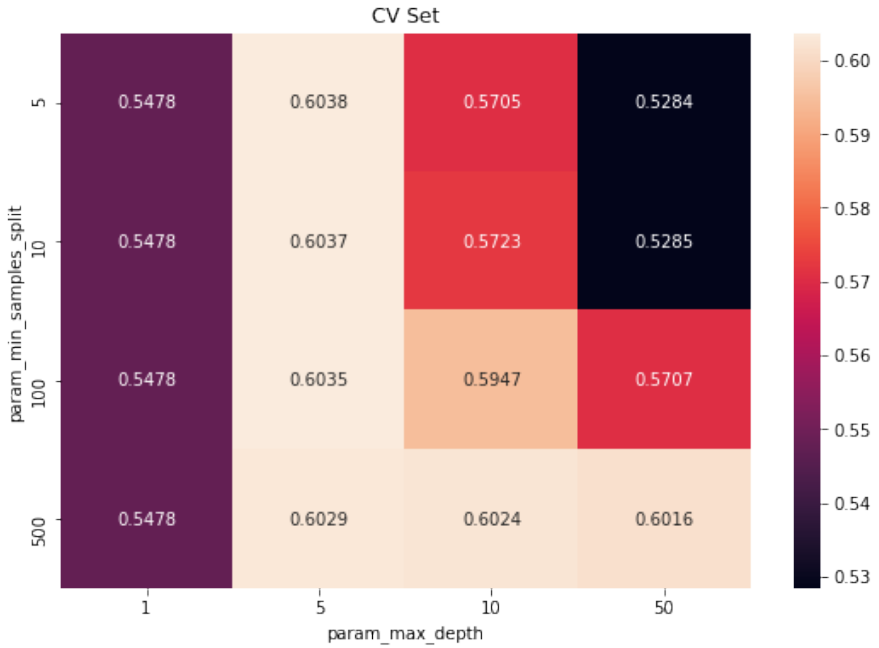
In [63]:
```
clf2.fit(X_set2_cv, Y_cv)
print(clf2.best_estimator_)
#Mean cross-validated score of the best_estimator
print(clf2.score(X_set2_cv,Y_cv))
print(clf2.score(X_set2_test,Y_test))
pd.DataFrame(clf2.cv_results_).head(2)
cv_results = clf2.cv_results_
```

```
DecisionTreeClassifier(class_weight='balanced', max_depth=5,
                       min_samples_split=5)
0.6694840968511894
0.6160437332732314
```

Heatmap on test and cv data, based on max depth, min sample split and AUC scores

In [64]:
```
import seaborn as sns
max_scores1 = pd.DataFrame(clf2.cv_results_).groupby(['param_min_samples_split', 'param_max_depth']).max().unstack
fig, ax = plt.subplots(1,2, figsize=(20,6))
sns.heatmap(max_scores1.mean_train_score, annot = True, fmt='.4g', ax=ax[0])
sns.heatmap(max_scores1.mean_test_score, annot = True, fmt='.4g', ax=ax[1])
ax[0].set_title('Train Set')
ax[1].set_title('CV Set')
plt.show()
```

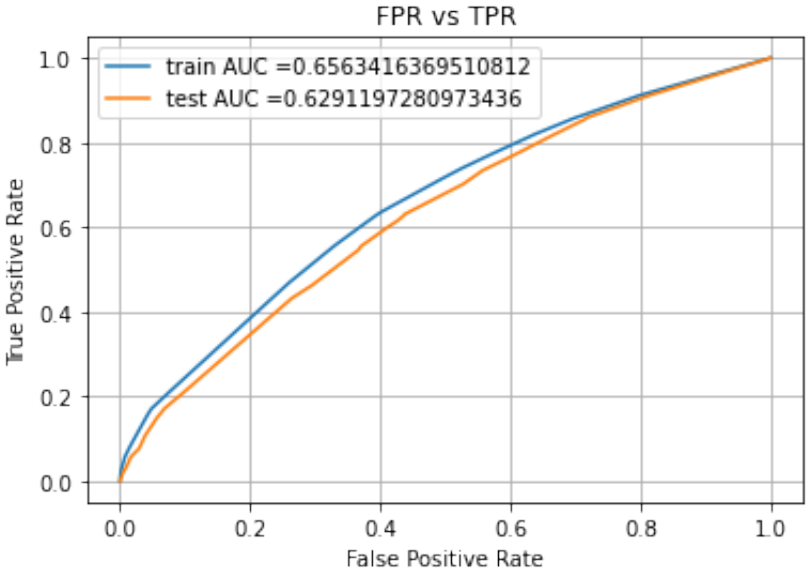




ROC Curve of Train and Test Data

In [65]:
```
hyper_parameters=[{'max_depth':[5], 'min_samples_split':[5] }]
```

In [66]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

train_auc = []
test_auc = []

clf_s1 = GridSearchCV( DecisionTreeClassifier(class_weight = 'balanced'),hyper_parameters)
clf_s1.fit(X_set2_train, Y_train)
# clf_v1 = DecisionTreeClassifier (class_weight = 'balanced',max_depth=10,min_samples_split=500)

Y_train_pred1 = clf_s1.predict_proba(X_set2_train)[:, 1]
Y_test_pred1 = clf_s1.predict_proba(X_set2_test)[:, 1]

train_fpr, train_tpr, tr_thresholds = roc_curve(Y_train, Y_train_pred1)
test_fpr, test_tpr, te_thresholds = roc_curve(Y_test, Y_test_pred1)

auc2 = auc(test_fpr, test_tpr)

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

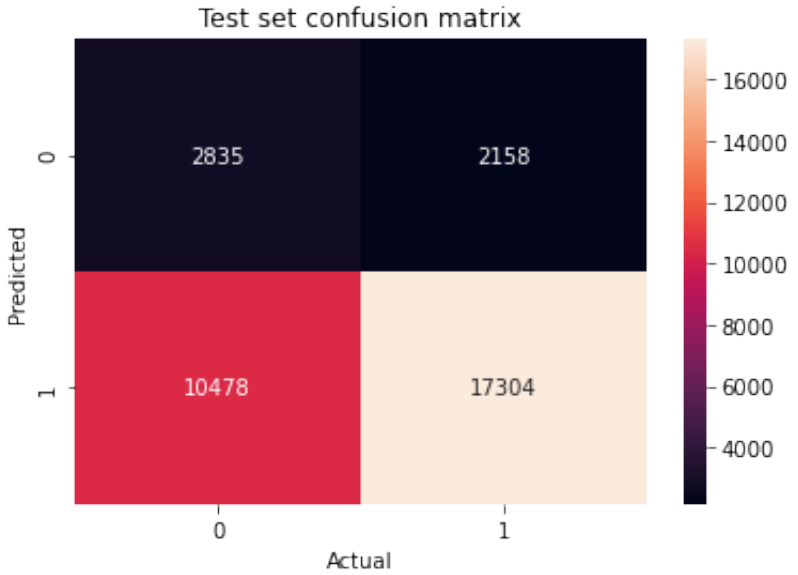


Confusion Matrix

In [67]:
```
import numpy as np
print("="*100)
from sklearn.metrics import confusion_matrix
best_t1 = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
print(confusion_matrix(Y_train, predict_with_best_t(Y_train_pred1, best_t1)))
```

```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.3808595461023596 for threshold 0.518
Train confusion matrix
[[ 4825  3169]
 [16803 28734]]
```

In [68]:
```
import seaborn as sns
heatmap_train = sns.heatmap(confusion_matrix(Y_test, predict_with_best_t(Y_test_pred1, best_t1)), annot=True, fmt=
plt.title("Test set confusion matrix")
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.show()
```



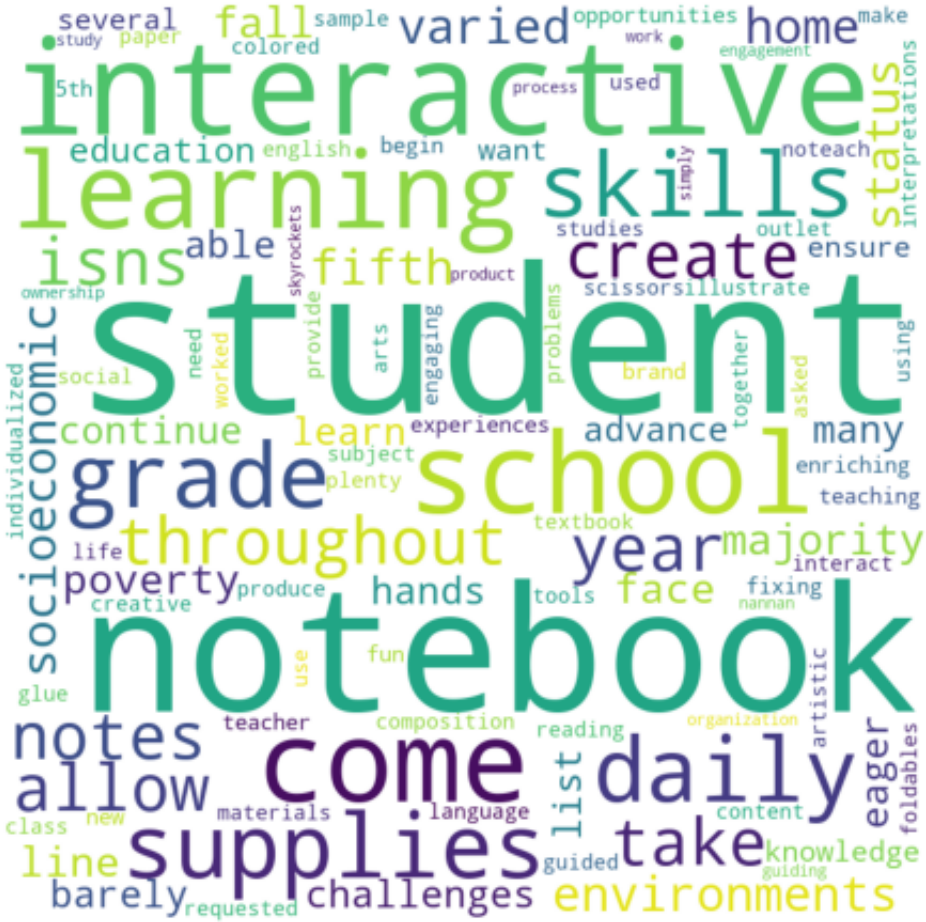
Word Cloud on False Positives

In [69]:
```
pred1 = predict_with_best_t(Y_test_pred1, best_t1)
print(Y_test.shape)
print(len(pred1))
```

```
(32775,)
32775
```

In [70]:
```
fp = []
for i in range(len(Y_test)) :
  if (Y_test[i] == 0) & (pred1[i] == 1) :
    fp.append(i)

fp_essay1 = []
for i in fp :
  fp_essay1.append(X_test['essay'].values[i])
```

In [71]:
```
from wordcloud import WordCloud, STOPWORDS

comment_words = ' '
stopwords = set(STOPWORDS)
for val in fp_essay1 :
  val = str(val)
  tokens = val.split()
for i in range(len(tokens)):
  tokens[i] = tokens[i].lower()
for words in tokens :
  comment_words = comment_words + words + ' '
wordcloud = WordCloud(width = 800, height = 800, background_color ='white', stopwords = stopwords,min_font_size =

plt.figure(figsize = (6, 6), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)
plt.show()
```



Task 2:

In [117…

```
features = clf_v1.feature_importances_
print(features)
non_zero_f = np.where(clf_v1.feature_importances_ > 0)[0]
```

```
[0.         0.         0.         ... 0.         0.00595292 0.00327126]
```

In [77]:
```
features_names = []
for a in vectorizer1.get_feature_names(): # clean_sub_categories
    features_names.append(a)
for a in vectorizer3.get_feature_names(): # clean_sub_categories
    features_names.append(a)
for a in vectorizer4.get_feature_names(): # school state
    features_names.append(a)
for a in vectorizer5.get_feature_names(): # teacher_prefix
    features_names.append(a)
for a in vectorizer6.get_feature_names(): # Grades
    features_names.append(a)
for a in vectorizer2.get_feature_names(): # bow_essay
    features_names.append(a)

features_names.append("price")
features_names.append("teacher_number_of_previously_posted_projects")
features_names.append("pos")
features_names.append("neg")
features_names.append("neu")
features_names.append("com")

print(len(features_names))
```

```
165418
```

In [118…
```
non0_features = []
n=0
for i in features:
    if(i > 0.0):
        non0_features.append(features_names[n])
        n+=1

print(len(non0_features))
```

```
917
```

In [101…
```
X_set1_train
```

Out[101…
```
<53531x165418 sparse matrix of type '<class 'numpy.float64'>'
        with 12813350 stored elements in COOrdinate format>
```

In [102…
```
X_train_new = scipy.sparse.csr_matrix(X_set1_train)
```

In [103…
```
X_train_new
```

Out[103…
```
<53531x165418 sparse matrix of type '<class 'numpy.float64'>'
        with 12813350 stored elements in Compressed Sparse Row format>
```

In [104…
```
X_train_new.tocsc()[:,non_zero_f]
```

Out[104…
```
<53531x917 sparse matrix of type '<class 'numpy.float64'>'
        with 2291383 stored elements in Compressed Sparse Column format>
```

In [110…
```
X_test_new = scipy.sparse.csr_matrix(X_set1_test)
```

In [111…
```
X_test_new
```

Out[111…
```
<32775x165418 sparse matrix of type '<class 'numpy.float64'>'
        with 7725026 stored elements in Compressed Sparse Row format>
```

In [112…
```
X_test_new.tocsc()[:,non_zero_f]
```

Out[112…
```
<32775x917 sparse matrix of type '<class 'numpy.float64'>'
        with 1399889 stored elements in Compressed Sparse Column format>
```

In [106…
```
dt = DecisionTreeClassifier(class_weight = 'balanced')
parameters = {'max_depth': [1, 5, 10, 50], 'min_samples_split': [5, 10, 100, 500]}
m1 = GridSearchCV(dt, parameters, cv=3, scoring='roc_auc',return_train_score=True)
m1 = m1.fit(X_train_new, Y_train)
```

In [113…
```
print(m1.best_estimator_)
#Mean cross-validated score of the best_estimator
print(m1.score(X_train_new,Y_train))
print(m1.score(X_test_new,Y_test))
pd.DataFrame(m1.cv_results_).head(2)
test_results = m1.cv_results_
```

```
DecisionTreeClassifier(class_weight='balanced', max_depth=10,
                       min_samples_split=500)
0.7128266022957498
0.6514689638995421
```

Summary

In [115…
```
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Max Depth", "Min Sample Split", "Test AUC"]

x.add_row(["TFIDF", "Decision Tree", 10, 500, auc1])
x.add_row([" ", " ", " ", " ", " "])
x.add_row(["TFIDF W2V", "Decision Tree", 5, 5, auc2])
x.add_row([" ", " ", " ", " ", " "])
x.add_row(["TFIDF with non zero features", "Decision Tree", 10, 500, 0.6514689638995421])

print(x)
```

```
+------------------------------+---------------+-----------+------------------+--------------------+
|          Vectorizer          |     Model     | Max Depth | Min Sample Split |      Test AUC      |
+------------------------------+---------------+-----------+------------------+--------------------+
|            TFIDF             | Decision Tree |     10    |       500        | 0.6514689638995421 |
|                              |               |           |                  |                    |
|          TFIDF W2V           | Decision Tree |     5     |        5         | 0.6291197280973436 |
|                              |               |           |                  |                    |
| TFIDF with non zero features | Decision Tree |     10    |       500        | 0.6514689638995421 |
+------------------------------+---------------+-----------+------------------+--------------------+
```