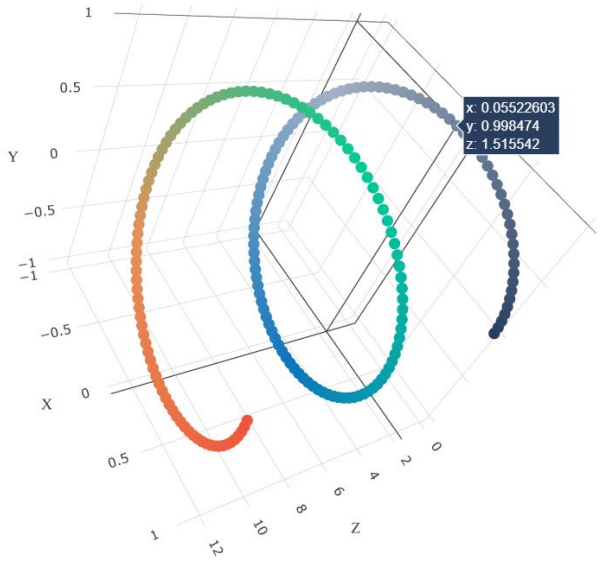
Assignment 8: DT

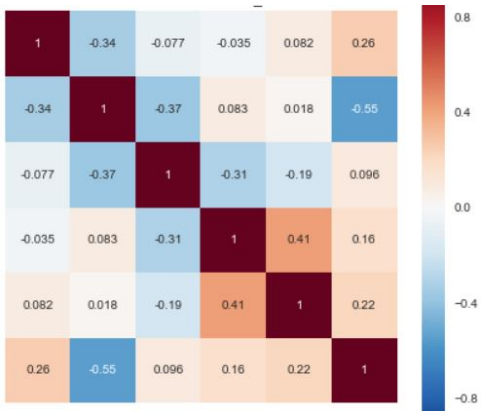
1. **Apply Decision Tree Classifier(DecisionTreeClassifier) on these feature sets**
    - Set 1: categorical, numerical features + preprocessed_eassay (TFIDF)
    - Set 2: categorical, numerical features + preprocessed_eassay (TFIDF W2V)
2. **The hyper paramter tuning (best `depth` in range [1, 5, 10, 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*

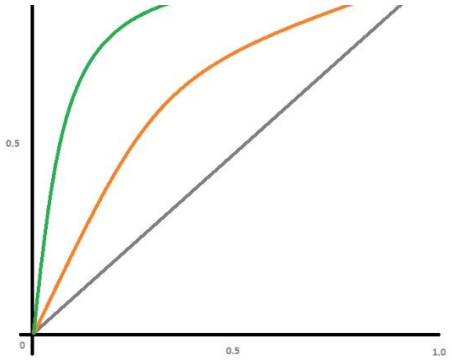
or

- 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](https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points

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

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

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

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

Vectorizer	Model	Hyper parameter	AUC
BOW	Brute	7	0.78
TFIDF	Brute	12	0.79
W2V	Brute	10	0.78
TFIDFW2V	Brute	6	0.78

1. Decision Tree

1.1 Loading Data

In [1]:

```
import pandas
data = pandas.read_csv(r'C:\Users\hoj5kor\Desktop\GIT\preprocessed_data.csv')
```

In [2]:

```
data.head(2)
```

Out[2]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved	clean_cate
0	ca	mrs	grades_prek_2	53	1	math_s

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved	clean_cate
1	ut	ms	grades_3_5	4	1	special

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

In [3]:

```
# 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
from sklearn.model_selection import train_test_split
X = data.drop('project_is_approved',axis=1)
y = data['project_is_approved']
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.15,random_state=42,stratify=y)
X_train,X_cv,y_train,y_cv =
train_test_split(X_train,y_train,test_size=0.15,random_state=42,stratify=y_train)
print(X_train.shape,y_train.shape)
print(X_test.shape,y_train.shape)
print(X_cv.shape,y_cv.shape)
```

```
(78931, 8) (78931,)
(16388, 8) (78931,)
(13929, 8) (13929,)
```

1.3 Make Data Model Ready: encoding eassay, and project_title

In [4]:

```
import pickle
with open(r'C:\Users\hoj5kor\Desktop\GIT\glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

In [5]:

```
from tqdm import tqdm
def tfidf_w2v(vectorizer, data):
    dictionary = dict(zip(vectorizer.get_feature_names(), list(vectorizer.idf_)))
    tfidf_words = set(vectorizer.get_feature_names())
    tfidf_w2v_vectors = []; # the avg-w2v for each sentence/review is stored in this list
    for sentence in tqdm(data): # 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 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)

    return np.array(tfidf_w2v_vectors)
```

In [6]:

In [6]:

```
# 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
# make sure you featurize train and test data separatly

# 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 numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer

feature_names = []

TfidfVec = TfidfVectorizer(ngram_range=(1,4),min_df=10,max_features=5000)
TfidfVec.fit(X_train['essay'].values)

X_train_TfidfVec = TfidfVec.transform(X_train['essay'].values)
X_train_TfidfW2V = tfidf_w2v(TfidfVec, X_train['essay'].values)
X_test_TfidfVec = TfidfVec.transform(X_test['essay'].values)
X_test_TfidfW2V = tfidf_w2v(TfidfVec, X_test['essay'].values)
X_cv_TfidfVec = TfidfVec.transform(X_cv['essay'].values)
X_cv_TfidfW2V = tfidf_w2v(TfidfVec, X_cv['essay'].values)

print('After Vectorization')
print('='*50)
print(X_train_TfidfVec.shape, y_train.shape)
print(X_test_TfidfVec.shape, y_test.shape)
print(X_cv_TfidfVec.shape, y_cv.shape)
print('='*50)
print(X_train_TfidfW2V.shape, y_train.shape)
print(X_test_TfidfW2V.shape, y_test.shape)
print(X_cv_TfidfW2V.shape, y_cv.shape)
print('='*50)
print(TfidfVec.get_feature_names()[:10])
feature_names.extend(TfidfVec.get_feature_names())
```

```
100%|██████████████████████████████████████████████████████████████████████████| 78931/78931 [02:
29<00:00, 528.73it/s]
100%|██████████████████████████████████████████████████████████████████████████| 16388/16388 [00:
31<00:00, 521.71it/s]
100%|██████████████████████████████████████████████████████████████████████████| 13929/13929 [00:
26<00:00, 520.02it/s]
```

```
After Vectorization
==================================================
(78931, 5000) (78931,)
(16388, 5000) (16388,)
(13929, 5000) (13929,)
==================================================
(78931, 300) (78931,)
(16388, 300) (16388,)
(13929, 300) (13929,)
==================================================
['000', '10', '100', '100 free', '100 percent', '100 students', '100 students receive', '100 stude
nts receive free', '11', '12']
```

1.4 Make Data Model Ready: encoding numerical, categorical features

In [7]:

```
# 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
# make sure you featurize train and test data separatly

# 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

# encoding categorical features using tfidf vectorizer

# encoding school_state
vectorizer = TfidfVectorizer()
vectorizer.fit(X_train['school_state'].values)

X_train_state_tfidf = vectorizer.transform(X_train['school_state'].values)
X_test_state_tfidf = vectorizer.transform(X_test['school_state'].values)
X_cv_state_tfidf = vectorizer.transform(X_cv['school_state'].values)


print(vectorizer.get_feature_names()[:5])
feature_names.extend(vectorizer.get_feature_names())

# encoding teacher_prefix
vectorizer = TfidfVectorizer()
vectorizer.fit(X_train['teacher_prefix'].values)

X_train_teacher_prefix_tfidf = vectorizer.transform(X_train['teacher_prefix'].values)
X_test_teacher_prefix_tfidf = vectorizer.transform(X_test['teacher_prefix'].values)
X_cv_teacher_prefix_tfidf = vectorizer.transform(X_cv['teacher_prefix'].values)

print(vectorizer.get_feature_names()[:5])
feature_names.extend(vectorizer.get_feature_names())

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

X_train_project_grade_category_tfidf =
vectorizer.transform(X_train['project_grade_category'].values)
X_test_project_grade_category_tfidf = vectorizer.transform(X_test['project_grade_category'].values
)
X_cv_project_grade_category_tfidf = vectorizer.transform(X_cv['project_grade_category'].values)

print(vectorizer.get_feature_names()[:5])
feature_names.extend(vectorizer.get_feature_names())

# encoding clean_categories
vectorizer = TfidfVectorizer()
vectorizer.fit(X_train['clean_categories'].values)

X_train_clean_categories_tfidf = vectorizer.transform(X_train['clean_categories'].values)
X_test_clean_categories_tfidf = vectorizer.transform(X_test['clean_categories'].values)
X_cv_clean_categories_tfidf = vectorizer.transform(X_cv['clean_categories'].values)

print(vectorizer.get_feature_names()[:5])
feature_names.extend(vectorizer.get_feature_names())


# encoding clean_subcategories
vectorizer = TfidfVectorizer()
vectorizer.fit(X_train['clean_subcategories'].values)

X_train_clean_subcategories_tfidf = vectorizer.transform(X_train['clean_subcategories'].values)
X_test_clean_subcategories_tfidf = vectorizer.transform(X_test['clean_subcategories'].values)
X_cv_clean_subcategories_tfidf = vectorizer.transform(X_cv['clean_subcategories'].values)

print(vectorizer.get_feature_names()[:5])
feature_names.extend(vectorizer.get_feature_names())
```

```
['ak', 'al', 'ar', 'az', 'ca']
['dr', 'mr', 'mrs', 'ms', 'teacher']
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'literacy_language']
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government',
'college_careerprep']
```

In [8]:

```
# encoding numerical features
# encoding price
```

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

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

X_train_price_norm = normalizer.transform(X_train['price'].values.reshape(1,-1)).reshape(-1,1)
X_cv_price_norm = normalizer.transform(X_cv['price'].values.reshape(1,-1)).reshape(-1,1)
X_test_price_norm = normalizer.transform(X_test['price'].values.reshape(1,-1)).reshape(-1,1)

print(X_cv_price_norm[:10])

# encoding teacher_number_of_previously_posted_projects

from sklearn.preprocessing import Normalizer
normalizer = Normalizer()

normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))

X_train_teacher_number_of_previously_posted_projects_norm =
normalizer.transform(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1,-1))
.reshape(-1,1)
X_cv_teacher_number_of_previously_posted_projects_norm =
normalizer.transform(X_cv['teacher_number_of_previously_posted_projects'].values.reshape(1,-1)).res
hape(-1,1)
X_test_teacher_number_of_previously_posted_projects_norm =
normalizer.transform(X_test['teacher_number_of_previously_posted_projects'].values.reshape(1,-1)).r
eshape(-1,1)

print('='*25)
print(X_train_teacher_number_of_previously_posted_projects_norm[:10])
```

```
[[0.00279308]
 [0.00579594]
 [0.00704402]
 [0.01440459]
 [0.00138981]
 [0.00613287]
 [0.00139355]
 [0.00130866]
 [0.00570264]
 [0.0009347 ]]
=========================
[[0.        ]
 [0.00155586]
 [0.00682187]
 [0.00059841]
 [0.00155586]
 [0.02752684]
 [0.        ]
 [0.00071809]
 [0.00251332]
 [0.00167555]]
```

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

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

In [9]:

```
# 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
# concatenating all features
```

```
from scipy.sparse import hstack

X_tr_tfidf =
hstack((X_train_TfidfVec,X_train_state_tfidf,X_train_teacher_prefix_tfidf,X_train_project_grade_cat
egory_tfidf,X_train_clean_categories_tfidf,X_train_clean_subcategories_tfidf,X_train_price_norm,X_
train_teacher_number_of_previously_posted_projects_norm)).tocsr()
X_cv_tfidf =
hstack((X_cv_TfidfVec,X_cv_state_tfidf,X_cv_teacher_prefix_tfidf,X_cv_project_grade_category_tfidf
,X_cv_clean_categories_tfidf,X_cv_clean_subcategories_tfidf,X_cv_price_norm,X_cv_teacher_number_of_
previously_posted_projects_norm)).tocsr()
X_te_tfidf =
hstack((X_test_TfidfVec,X_test_state_tfidf,X_test_teacher_prefix_tfidf,X_test_project_grade_categor
y_tfidf,X_test_clean_categories_tfidf,X_test_clean_subcategories_tfidf,X_test_price_norm,X_test_tea
cher_number_of_previously_posted_projects_norm)).tocsr()
print(X_train_state_tfidf.shape)
X_tr_tfidfw2v =
hstack((X_train_TfidfW2V,X_train_state_tfidf,X_train_teacher_prefix_tfidf,X_train_project_grade_cat
egory_tfidf,X_train_clean_categories_tfidf,X_train_clean_subcategories_tfidf,X_train_price_norm,X_
train_teacher_number_of_previously_posted_projects_norm)).tocsr()
X_cv_tfidfw2v =
hstack((X_cv_TfidfW2V,X_cv_state_tfidf,X_cv_teacher_prefix_tfidf,X_cv_project_grade_category_tfidf
,X_cv_clean_categories_tfidf,X_cv_clean_subcategories_tfidf,X_cv_price_norm,X_cv_teacher_number_of_
previously_posted_projects_norm)).tocsr()
X_te_tfidfw2v =
hstack((X_test_TfidfW2V,X_test_state_tfidf,X_test_teacher_prefix_tfidf,X_test_project_grade_categor
y_tfidf,X_test_clean_categories_tfidf,X_test_clean_subcategories_tfidf,X_test_price_norm,X_test_tea
cher_number_of_previously_posted_projects_norm)).tocsr()

print("Final Data matrix")
print('='*50)
print(X_tr_tfidf.shape, y_train.shape)
print(X_cv_tfidf.shape, y_cv.shape)
print(X_te_tfidf.shape, y_test.shape)
print('='*50)
print(X_tr_tfidfw2v.shape, y_train.shape)
print(X_cv_tfidfw2v.shape, y_cv.shape)
print(X_te_tfidfw2v.shape, y_test.shape)
```

```
(78931, 51)
Final Data matrix
==================================================
(78931, 5101) (78931,)
(13929, 5101) (13929,)
(16388, 5101) (16388,)
==================================================
(78931, 401) (78931,)
(13929, 401) (13929,)
(16388, 401) (16388,)
```

In [10]:

```
# 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
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the posi
tive class
    # not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your tr_loop will be 49041 - 49041%1000 = 49000
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

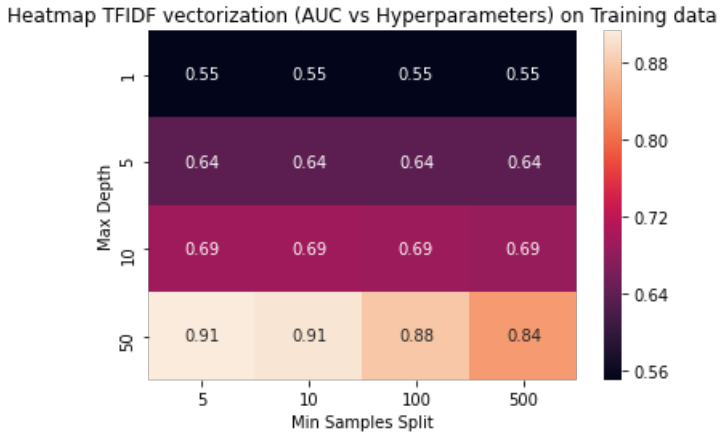
```
    return y_data_pred
```

In [11]:

```
from sklearn.tree import DecisionTreeClassifier
from tqdm import tqdm
from sklearn.metrics import roc_auc_score
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
#parameters = {'max_depth':[1, 5, 10, 50],'min_samples_split':[5, 10, 100, 500]}
depth=[1, 5, 10, 50]
min_samples_split=[5, 10, 100, 500]
train_auc = [[0 for samples in min_samples_split]for max_depth in depth]
cv_auc = [[0 for samples in min_samples_split]for max_depth in depth]
i=0
for max_depth in depth:
    j = 0
    for samples in min_samples_split:
        clf_tfidf = DecisionTreeClassifier(max_depth = max_depth,min_samples_split=samples)
        clf_tfidf.fit(X_tr_tfidf, y_train)

        y_train_pred = batch_predict(clf_tfidf, X_tr_tfidf)
        y_cv_pred = batch_predict(clf_tfidf, X_cv_tfidf)

        train_auc[i][j] = roc_auc_score(y_train,y_train_pred)
        cv_auc[i][j] = roc_auc_score(y_cv, y_cv_pred)
        j += 1
    i += 1
```

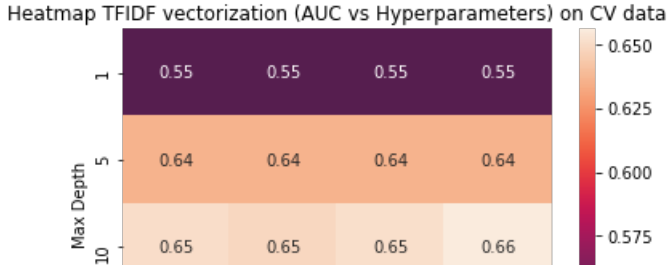
In [12]:

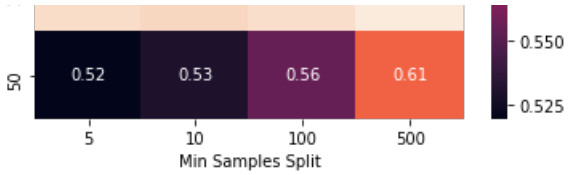
```
sns.heatmap(train_auc,annot=True,xticklabels=min_samples_split,yticklabels=depth)
plt.title('Heatmap TFIDF vectorization (AUC vs Hyperparameters) on Training data')
plt.ylabel('Max Depth')
plt.xlabel('Min Samples Split')
plt.show()
```



In [13]:

```
sns.heatmap(cv_auc,annot=True,xticklabels=min_samples_split,yticklabels=depth)
plt.title('Heatmap TFIDF vectorization (AUC vs Hyperparameters) on CV data')
plt.ylabel('Max Depth')
plt.xlabel('Min Samples Split')
plt.show()
```





From the above plots, we can see that Max_Depth=10 and Min_Samples_Split=500 leads to the best performance.

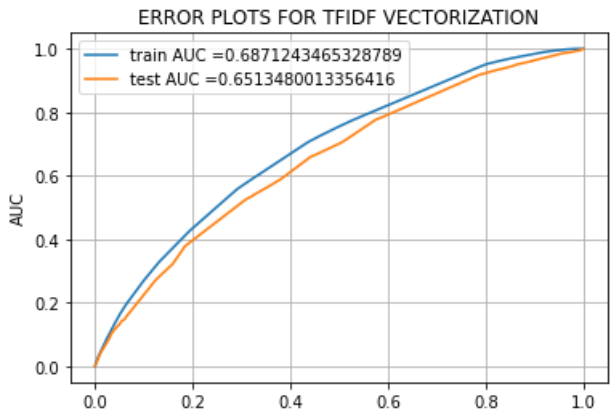
In [14]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc, confusion_matrix, accuracy_score

clf1 = DecisionTreeClassifier(max_depth = 10,min_samples_split=500)
clf1.fit(X_tr_tfidf, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = batch_predict(clf1, X_tr_tfidf)
y_test_pred = batch_predict(clf1, X_te_tfidf)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

y_test_class = list(map(lambda x:1 if x>=0.5 else 0, y_test_pred))
tn, fp, fn, tp = confusion_matrix(y_test, y_test_class).ravel()
print('Confusion Matrix')
print('================')
print(np.array([[tn, tp],[fn, fp]]))
print('Accuracy Score: ',accuracy_score(y_test, y_test_class))
```

```
Confusion Matrix
================
[[  105 13711]
 [  196  2376]]
Accuracy Score:  0.8430558945569929
```

In [15]:

```
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("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS FOR TFIDF VECTORIZATION")
plt.grid()
plt.show()
```

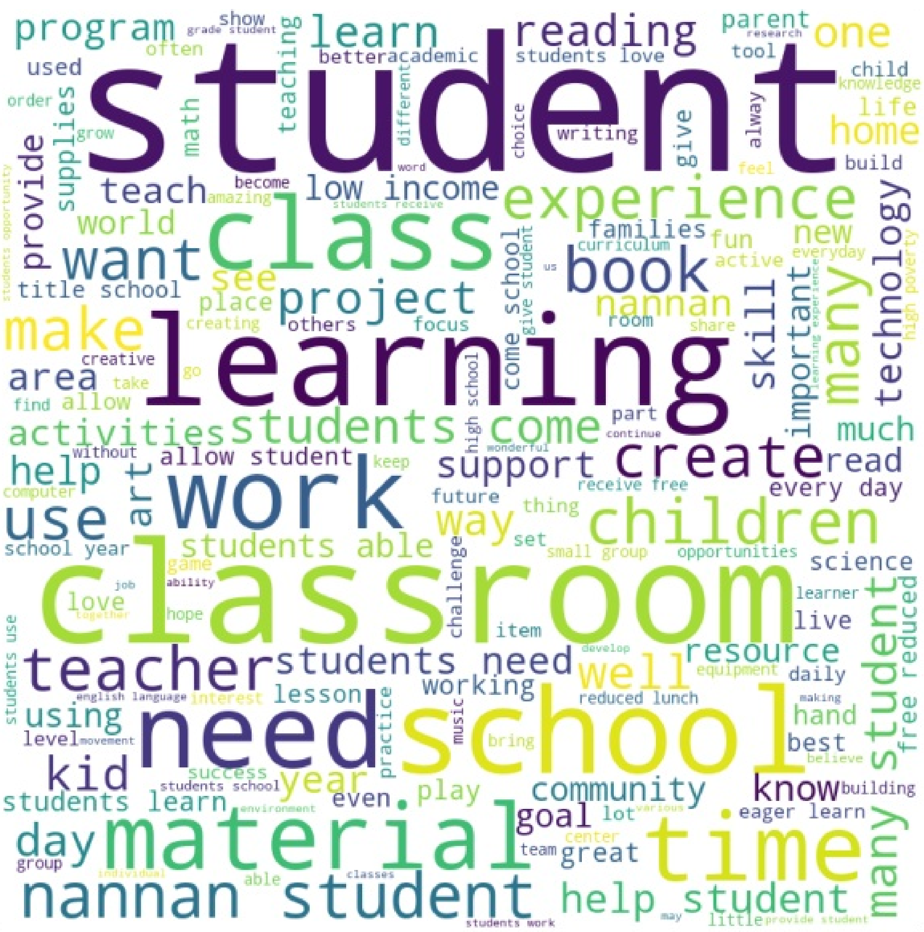


In [16]:

```
from wordcloud import WordCloud, STOPWORDS
```

```
fp = []
for i in range(len(y_test)):
    if y_test.values[i]==0 and y_test_class[i]==1:
        fp.append(X_test.values[i][-2])

print(len(fp))

comment_words = ' '
stopwords = set(STOPWORDS)

# iterate through the csv file
for val in fp:
    # typecaste each val to string
    val = str(val)
    # split the value
    tokens = val.split()
    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()
    for words in tokens:
        comment_words = comment_words + words + ' '


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

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

plt.show()
```

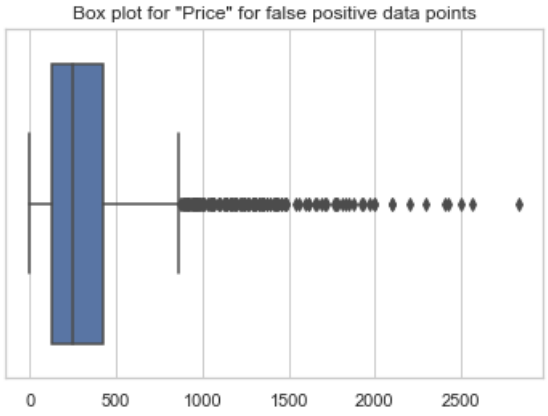
```
2376
```



In [17]:

```
#Plot the box plot with the `price` of these `false positive data points`
import seaborn as sns
fp_price = []
for i in range(len(y_test)):
    if y_test.values[i]==0 and y_test_class[i]==1:
        fp_price.append(X_test.values[i][-1])

print(len(fp_price))

sns.set(style="whitegrid")
sns.boxplot(fp_price)
plt.title('Box plot for "Price" for false positive data points')
plt.show()
```

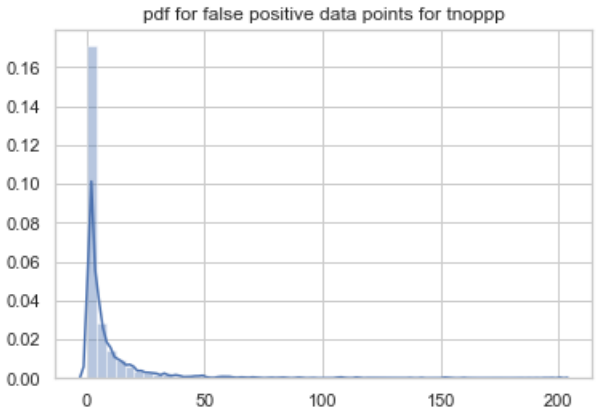
2376



In [18]:

```
#Plot the pdf with the `teacher_number_of_previously_posted_projects` of these `false positive dat
a points`
fp_tnoppp = []
for i in range(len(y_test)):
    if y_test.values[i]==0 and y_test_class[i]==1:
        fp_tnoppp.append(X_test.values[i][3])

print(len(fp_tnoppp))

sns.distplot(fp_tnoppp)
plt.title('pdf for false positive data points for tnoppp')
plt.show()
```

2376



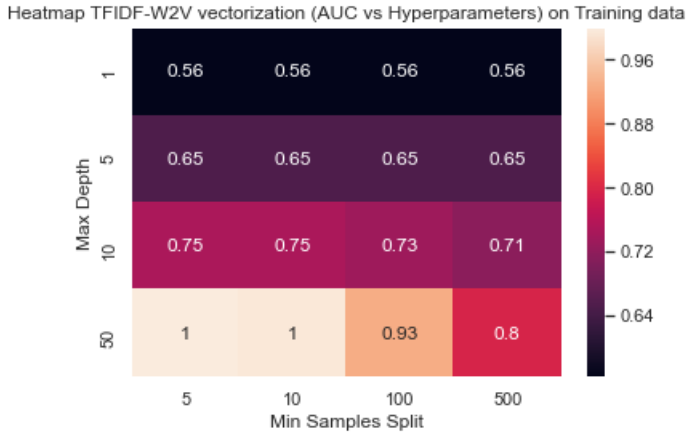
In [19]:

```
from sklearn.tree import DecisionTreeClassifier
from tqdm import tqdm
from sklearn.metrics import roc_auc_score
import numpy as np
```

```
import seaborn as sns
import matplotlib.pyplot as plt
#parameters = {'max_depth':[1, 5, 10, 50],'min_samples_split':[5, 10, 100, 500]}
depth=[1, 5, 10, 50]
min_samples_split=[5, 10, 100, 500]
train_auc = [[0 for samples in min_samples_split]for max_depth in depth]
cv_auc = [[0 for samples in min_samples_split]for max_depth in depth]
i=0
for max_depth in depth:
    j = 0
    for samples in min_samples_split:
        clf_tfidfw2v = DecisionTreeClassifier(max_depth = max_depth,min_samples_split=samples)
        clf_tfidfw2v.fit(X_tr_tfidfw2v, y_train)

        y_train_pred = batch_predict(clf_tfidfw2v, X_tr_tfidfw2v)
        y_cv_pred = batch_predict(clf_tfidfw2v, X_cv_tfidfw2v)

        train_auc[i][j] = roc_auc_score(y_train,y_train_pred)
        cv_auc[i][j] = roc_auc_score(y_cv, y_cv_pred)
        j += 1
    i += 1
```

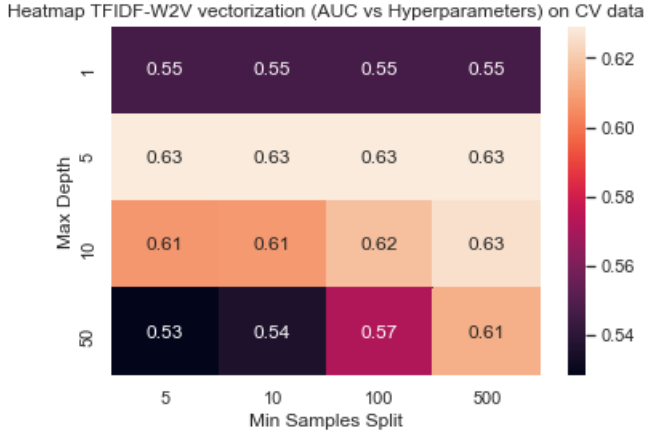
In [20]:

```
sns.heatmap(train_auc,annot=True,xticklabels=min_samples_split,yticklabels=depth)
plt.title('Heatmap TFIDF-W2V vectorization (AUC vs Hyperparameters) on Training data')
plt.ylabel('Max Depth')
plt.xlabel('Min Samples Split')
plt.show()
```



In [21]:

```
sns.heatmap(cv_auc,annot=True,xticklabels=min_samples_split,yticklabels=depth)
plt.title('Heatmap TFIDF-W2V vectorization (AUC vs Hyperparameters) on CV data')
plt.ylabel('Max Depth')
plt.xlabel('Min Samples Split')
plt.show()
```

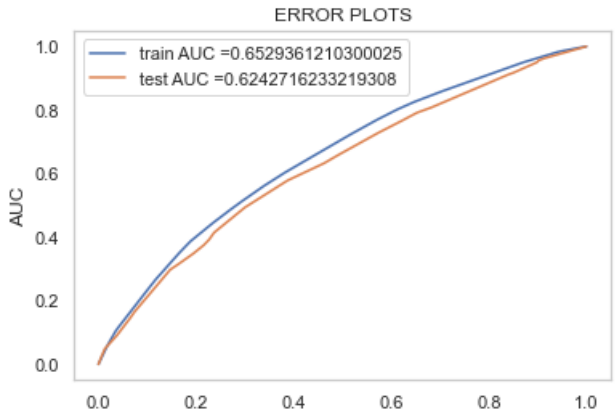


From the above plots, we can see that Max_Depth=5 and Min_Samples_Split=500 leads to the best performance.

In [22]:

```
# https://scikit-
learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
from sklearn.metrics import roc_curve, auc

clf2 = DecisionTreeClassifier(max_depth = 5,min_samples_split=500)
clf2.fit(X_tr_tfidfw2v, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive
class
# not the predicted outputs

y_train_pred = batch_predict(clf2, X_tr_tfidfw2v)
y_test_pred = batch_predict(clf2, X_te_tfidfw2v)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

y_test_class = list(map(lambda x:1 if x>=0.5 else 0, y_test_pred))
tn, fp, fn, tp = confusion_matrix(y_test, y_test_class).ravel()
print('Confusion Matrix')
print('================')
print(np.array([[tn, tp],[fn, fp]]))
print('Accuracy Score: ',accuracy_score(y_test, y_test_class))

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("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

```
Confusion Matrix
================
[[    0 13907]
 [    0  2481]]
Accuracy Score:  0.8486087381010495
```

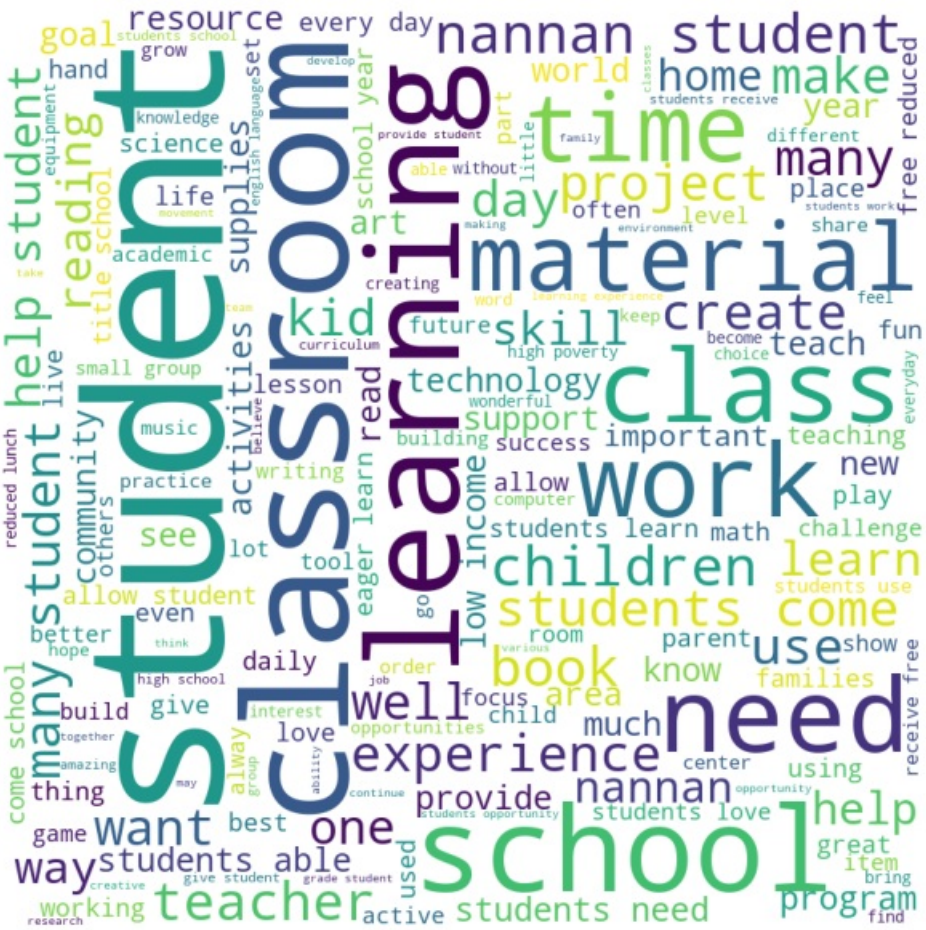


In [23]:

```
fp = []
for i in range(len(y_test)):
    if y_test.values[i]==0 and y_test_class[i]==1:
        fp.append(X_test.values[i][-2])

print(len(fp))

comment_words = ' '
stopwords = set(STOPWORDS)

# iterate through the csv file
for val in fp:
    # typecaste each val to string
```

```
    val = str(val)
    # split the value
    tokens = val.split()
    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()
    for words in tokens:
        comment_words = comment_words + words + ' '


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

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

plt.show()
```

2481

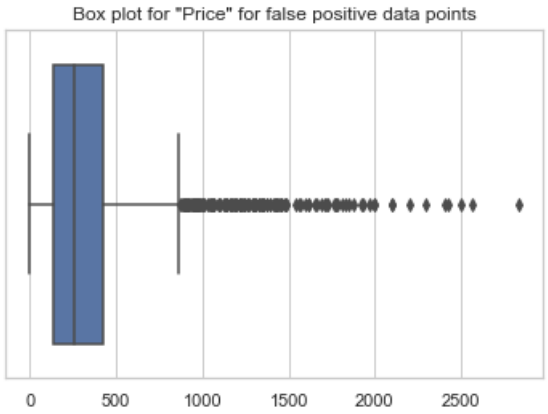


In [24]:

```
fp_price = []
for i in range(len(y_test)):
    if y_test.values[i]==0 and y_test_class[i]==1:
        fp_price.append(X_test.values[i][-1])

print(len(fp_price))

sns.boxplot(fp_price)
plt.title('Box plot for "Price" for false positive data points')
plt.show()
```

2481

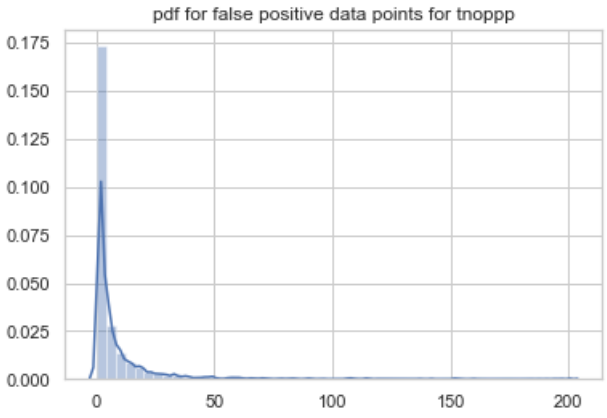
2481



In [25]:

```
#Plot the pdf with the `teacher_number_of_previously_posted_projects` of these `false positive dat
a points`
fp_tnoppp = []
for i in range(len(y_test)):
    if y_test.values[i]==0 and y_test_class[i]==1:
        fp_tnoppp.append(X_test.values[i][3])

print(len(fp_tnoppp))

sns.distplot(fp_tnoppp)
plt.title('pdf for false positive data points for tnoppp')
plt.show()
```

2481



1.6 Getting top features using `feature_importances_`

In [26]:

```
'''# 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
#Top 20 features for Tfidf
min_indices = np.argsort(clf_tfidf.feature_importances_)
max_indices = min_indices[::-1][:20]
np.take(feature_names,max_indices)[:20]

Task 2: For this task consider set-1 features. Select all the features which are having non-zero f
eature 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 kee
p it None.'''

import pandas as pd
clf1.feature_importances_[clf1.feature_importances_>0]
X_tr_tfidf_new = X_tr_tfidf.todense()[:,clf1.feature_importances_>0]
X_cv_tfidf_new = X_cv_tfidf.todense()[:,clf1.feature_importances_>0]
X_te_tfidf_new = X_te_tfidf.todense()[:,clf1.feature_importances_>0]
print(X_tr_tfidf_new.shape)
print(X_cv_tfidf_new.shape)
print(X_te_tfidf_new.shape)
```

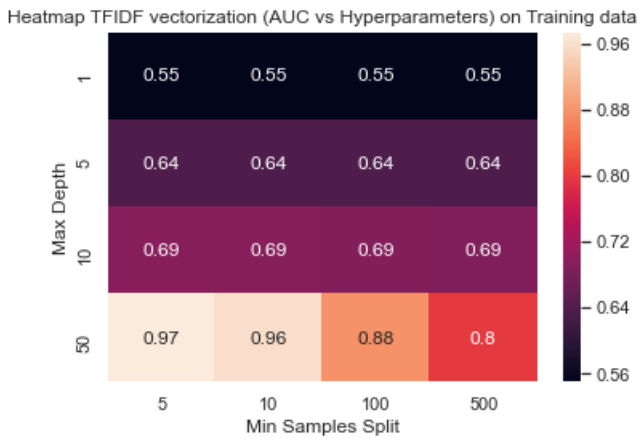
```
(78931, 96)
(13929, 96)
(16388, 96)
```

In [27]:

```
depth=[1, 5, 10, 50]
min_samples_split=[5, 10, 100, 500]
train_auc = [[0 for samples in min_samples_split]for max_depth in depth]
cv_auc = [[0 for samples in min_samples_split]for max_depth in depth]
i=0
for max_depth in depth:
    j = 0
    for samples in min_samples_split:
        clf_tfidf = DecisionTreeClassifier(max_depth = max_depth,min_samples_split=samples)
        clf_tfidf.fit(X_tr_tfidf_new, y_train)

        y_train_pred = batch_predict(clf_tfidf, X_tr_tfidf_new)
        y_cv_pred = batch_predict(clf_tfidf, X_cv_tfidf_new)

        train_auc[i][j] = roc_auc_score(y_train,y_train_pred)
        cv_auc[i][j] = roc_auc_score(y_cv, y_cv_pred)
        j += 1
    i += 1
```

In [31]:

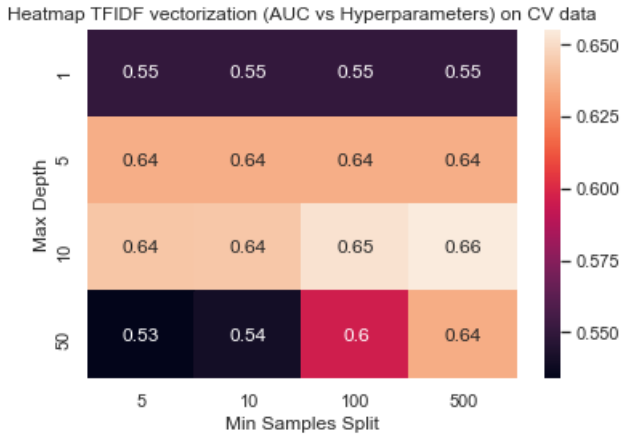
```
sns.heatmap(train_auc,annot=True,xticklabels=min_samples_split,yticklabels=depth)
plt.title('Heatmap TFIDF vectorization (AUC vs Hyperparameters) on Training data')
plt.ylabel('Max Depth')
plt.xlabel('Min Samples Split')
plt.show()
```



In [32]:

```
sns.heatmap(cv_auc,annot=True,xticklabels=min_samples_split,yticklabels=depth)
plt.title('Heatmap TFIDF vectorization (AUC vs Hyperparameters) on CV data')
```

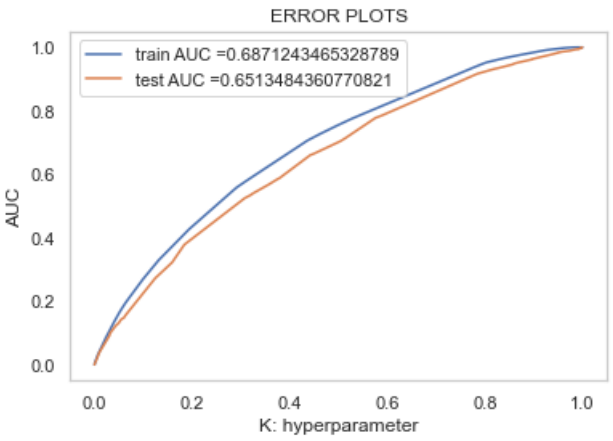
```
plt.title('Heatmap TFIDF vectorization (AUC vs Hyperparameters) on CV data')
plt.ylabel('Max Depth')
plt.xlabel('Min Samples Split')
plt.show()
```



In [30]:

```
from sklearn.metrics import roc_curve, auc

clf2 = DecisionTreeClassifier(max_depth = 10,min_samples_split=500)
clf2.fit(X_tr_tfidf_new, y_train)

y_train_pred = batch_predict(clf2, X_tr_tfidf_new)
y_test_pred = batch_predict(clf2, X_te_tfidf_new)

train_fpr, train_tpr, tr_thresholds = roc_curve(y_train, y_train_pred)
test_fpr, test_tpr, te_thresholds = roc_curve(y_test, y_test_pred)

y_test_class = list(map(lambda x:1 if x>=0.5 else 0, y_test_pred))
tn, fp, fn, tp = confusion_matrix(y_test, y_test_class).ravel()
print('Confusion Matrix')
print('================')
print(np.array([[tn, tp],[fn, fp]]))
print('Accuracy Score: ',accuracy_score(y_test, y_test_class))

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("K: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

```
Confusion Matrix
================
[[  105 13711]
 [  196  2376]]
Accuracy Score:  0.8430558945569929
```



2. Summary

In [33]:

```
print('+--------------------------+-------+------------------------+')
print('|       Vectorizer         | Model | Hyper Parameter |  AUC |')
print('+--------------------------+-------+------------------------+')
print('|           TFIDF          | Brute |    (10,500)     | 0.65 |')
print('+--------------------------+-------+------------------------+')
print('|       TFIDF-W2V          | Brute |    (05,500)     | 0.62 |')
print('+--------------------------+-------+------------------------+')
print('| TFIDF(Non-Zero Features) | Brute |    (10,500)     | 0.65 |')
print('+--------------------------+-------+------------------------+')
```

```
+--------------------------+-------+------------------------+
|       Vectorizer         | Model | Hyper Parameter |  AUC |
+--------------------------+-------+------------------------+
|           TFIDF          | Brute |    (10,500)     | 0.65 |
+--------------------------+-------+------------------------+
|       TFIDF-W2V          | Brute |    (05,500)     | 0.62 |
+--------------------------+-------+------------------------+
| TFIDF(Non-Zero Features) | Brute |    (10,500)     | 0.65 |
+--------------------------+-------+------------------------+
```

In [ ]:

```

```