Assignment : DT

Please check below video before attempting this assignment

In [1]:
```
# 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-IDFW2V (https://www.appliedaicourse.com/lecture/11/applied-machine-learning-online-course/2916/avg-word2vec-tf-idf-weighted-word2vec/3/module-3-foundations-of-natural-language-processing-and-machine-learning)for more details.

Glove vectors

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

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

In [2]:
```
#please use below code to load glove vectors
import pickle
from tqdm import tqdm
import os
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())
```

or else , you can use below code

In [3]:
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")


import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer

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


from chart_studio.plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
```

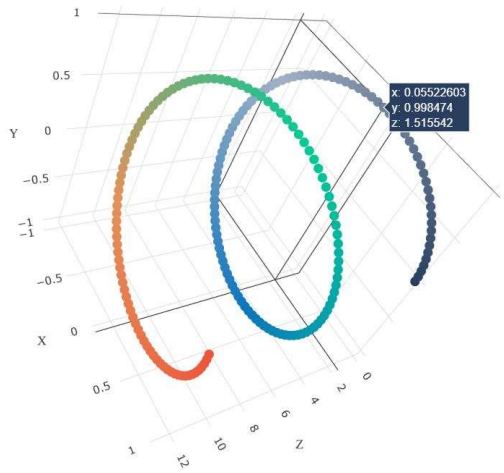
Task - 1

1. **Apply Decision Tree Classifier(DecisionTreeClassifier) on these feature sets**
   - Set 1: categorical, numerical features + preprocessed_essay (TFIDF) + Sentiment scores(preprocessed_essay)
   - Set 2: categorical, numerical features + preprocessed_essay (TFIDF W2V) + Sentiment scores(preprocessed_essay)
2. **The hyper paramter tuning (best `depth` in range [1, 3, 10, 30], and the best `min_samples_split` in range [5, 10, 100, 500])**
   - Find the best hyper parameter which will give the maximum AUC (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) value

- find the best hyper paramter using k-fold cross validation(use gridsearch cv or randomsearch cv)/simple cross validation data(you can write your own for loops refer sample solution)

3. **Representation of results**

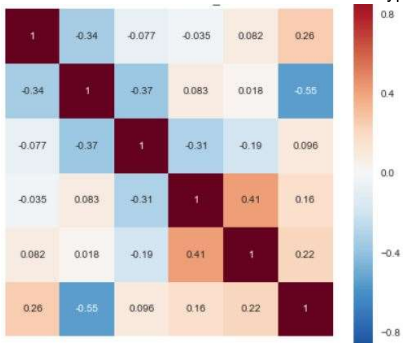
- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure



with X-axis as **min_sample_split**, Y-axis as **max_depth**, and Z-axis as **AUC Score** , we have given the notebook which explains how to plot this 3d plot, you can find it in the same drive *3d_scatter_plot.ipynb*

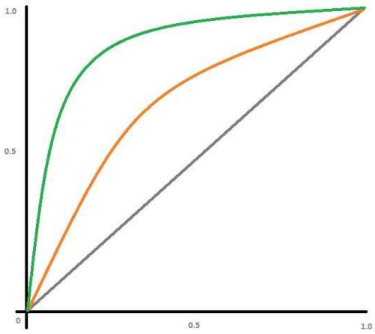
or

- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure



seaborn heat maps (https://seaborn.pydata.org/generated/seaborn.heatmap.html) with rows as **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. Make sure that you are using predict_proba method to calculate AUC curves, because AUC is calcualted on class probabilities and not on class labels.



- 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`

In [4]:
```
#pip install wordcloud
```

Task - 2

For this task consider **set-1** features.

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

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

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

Hint for calculating Sentiment scores

In [5]:
```
import nltk
nltk.download('vader_lexicon')
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\gurus\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

Out[5]: True

In [6]:
```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
# import nltk
# nltk.download('vader_lexicon')

sid = SentimentIntensityAnalyzer()

sample_sentence_1='I am happy.'
ss_1 = sid.polarity_scores(sample_sentence_1)
print('sentiment score for sentence 1',ss_1)

sample_sentence_2='I am sad.'
ss_2 = sid.polarity_scores(sample_sentence_2)
print('sentiment score for sentence 2',ss_2)

sample_sentence_3='I am going to New Delhi tommorow.'
ss_3 = sid.polarity_scores(sample_sentence_3)
print('sentiment score for sentence 3',ss_3)
```

```
sentiment score for sentence 1 {'neg': 0.0, 'neu': 0.213, 'pos': 0.787, 'compound': 0.5719}
sentiment score for sentence 2 {'neg': 0.756, 'neu': 0.244, 'pos': 0.0, 'compound': -0.4767}
sentiment score for sentence 3 {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}
```

Decision Tree

Task - 1

1.1 Loading Data

In [7]:
```
#make sure you are loading atleast 50k datapoints
#you can work with features of preprocessed_data.csv for the assignment.
import pandas
data = pandas.read_csv('preprocessed_data.csv',nrows=50000)
data.head()
```

Out[7]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved	clean_categories	clean_subcategorie
0	ca	mrs	grades_prek_2	53	1	math_science	appliedscience health_lifescienc
1	ut	ms	grades_3_5	4	1	specialneeds	specialneec
2	ca	mrs	grades_prek_2	10	1	literacy_language	literac
3	ga	mrs	grades_prek_2	2	1	appliedlearning	earlydevelopme
4	wa	mrs	grades_3_5	2	1	literacy_language	literac

In [8]:
```
# write your code in following steps for task 1
# 1. calculate sentiment scores for the essay feature
# 2. Split your data.
# 3. perform tfidf vectorization of text data.
# 4. perform tfidf w2v vectorization of text data.
# 5. perform encoding of categorical features.
# 6. perform encoding of numerical features
# 7. For task 1 set 1 stack up all the features
# 8. For task 1 set 2 stack up all the features (for stacking dense features you can use np.stack)
# 9. Perform hyperparameter tuning and plot either heatmap or 3d plot.
# 10. Find the best parameters and fit the model. Plot ROC-AUC curve(using predict proba method)
# 11. Plot confusion matrix based on best threshold value
# 12. Find all the false positive data points and plot wordcloud of essay text and pdf of teacher_number_of_previously_posted_prc
# 13. Write your observations about the wordcloud and pdf.
```

1. calculate sentiment scores for the essay feature

In [9]:
```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
# import nltk
# nltk.download('vader_lexicon')
Neg = []
Pos = []
Nue = []

sia = SentimentIntensityAnalyzer()
for essay in data['essay'].values:
    k = sia.polarity_scores(essay)
    Neg.append(k['neg'])
    Nue.append(k['neu'])
    Pos.append(k['pos'])

data['Positive'] = Pos
data['Negetive'] = Neg
data['Nuetral'] = Nue
```

2. Split your data.

In [10]:
```
# drop project_approved column and make it new array Y
y = data['project_is_approved'].values
X = data.drop(['project_is_approved'], axis=1)

# train test split
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)

print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
X.head(1)
```

```
(33500, 11) (33500,)
(16500, 11) (16500,)
```

Out[10]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	clean_subcategories	essay	price
0	ca	mrs	grades_prek_2	53	math_science	appliedsciences health_lifescience	i fortunate enough use fairy tale stem kits cl...	725.05

3. perform tfidf vectorization of text data.

In [11]:
```
# Apply TF-IDF vectorization on 'Preprocessed_Essay'

# Encoding of Essay using TF-IDF
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=100)
vectorizer.fit(X_train['essay'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_tfidf = vectorizer.transform(X_train['essay'].values)

X_test_tfidf = vectorizer.transform(X_test['essay'].values)

print("After vectorizations")
print(X_train_tfidf.shape, y_train.shape)
print(X_test_tfidf.shape, y_test.shape)
```

```
After vectorizations
(33500, 3405) (33500,)
(16500, 3405) (16500,)
```

4. perform tfidf w2v vectorization of text data.

In [12]:
```
# Apply TF-IDF W2V vectorization on 'Preprocessed_Essay'
tfidf_model = TfidfVectorizer(min_df=50)
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())

# Encoding of Essay using TF-IDF W2V
X_train_tfidf_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in X_train['essay']: # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the 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
    X_train_tfidf_w2v.append(vector)

X_train_tfidf_w2v = np.array(X_train_tfidf_w2v)
print(X_train_tfidf_w2v.shape)

X_test_tfidf_w2v = []; # the avg-w2v for each sentence/review is stored in this list
for sentence in X_test['essay']: # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    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
    X_test_tfidf_w2v.append(vector)

X_test_tfidf_w2v = np.array(X_test_tfidf_w2v)
print(X_test_tfidf_w2v.shape)
```

```
(33500, 300)
(16500, 300)
```

5. perform encoding of categorical features.

In [13]:
```
# Apply One-Hot Encoding on the categorical features either using OneHotEncoder() (or) CountVectorizer(binary=True)
from sklearn.preprocessing import OneHotEncoder
cat_feat = ['teacher_prefix','project_grade_category','school_state','clean_categories','clean_subcategories']
vectorizer1 = OneHotEncoder(handle_unknown='ignore')
# fit has to happen only on train data
vectorizer1.fit(X_train[cat_feat])

# we use the fitted CountVectorizer to convert the text to vector
X_train_cat_feat = vectorizer1.transform(X_train[cat_feat])
X_test_cat_feat = vectorizer1.transform(X_test[cat_feat])

print("After vectorizations")
print(X_train_cat_feat.shape, y_train.shape)
print(X_test_cat_feat.shape, y_test.shape)
```

```
After vectorizations
(33500, 444) (33500,)
(16500, 444) (16500,)
```

6. perform encoding of numerical features

In [14]:
```
from sklearn.preprocessing import Normalizer
scaler = Normalizer()

numeric = ['price','teacher_number_of_previously_posted_projects','Positive','Nuetral','Negetive']

scaler.fit(X_train[numeric])

X_train_numeric = scaler.transform(X_train[numeric])
X_test_numeric = scaler.transform(X_test[numeric])
```

7. For task 1 set 1 stack up all the features

In [15]:
```
from scipy.sparse import hstack
X_train1 = hstack((X_train_tfidf,X_train_cat_feat,X_train_numeric)).tocsr()

X_test1 = hstack((X_test_tfidf,X_test_cat_feat,X_test_numeric)).tocsr()

print("Shape final repesentation in set 1:")
print(X_train1.shape,y_train.shape)
print(X_test1.shape,y_test.shape)
```

```
Shape final repesentation in set 1:
(33500, 3854) (33500,)
(16500, 3854) (16500,)
```

8. For task 1 set 2 stack up all the features (for stacking dense features you can use np.stack)

In [16]:
```
X_train2 = hstack((X_train_tfidf_w2v,X_train_cat_feat,X_train_numeric)).tocsr()

X_test2 = hstack((X_test_tfidf_w2v,X_test_cat_feat,X_test_numeric)).tocsr()

print("Shape final repesentation in set 2:")
print(X_train2.shape,y_train.shape)
print(X_test2.shape,y_test.shape)
```

```
Shape final repesentation in set 2:
(33500, 749) (33500,)
(16500, 749) (16500,)
```

9. Perform hyperparameter tuning and plot heatmap .

In [17]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
dt = DecisionTreeClassifier()
#The hyper paramter tuning (best `depth` in range [1, 3, 10, 30],
# and the best `min_samples_split` in range [5, 10, 100, 500])
param_grid = {'max_depth':[1, 3, 10, 30],'min_samples_split':[5, 10, 100, 500]}
cv1 = GridSearchCV(estimator=dt,param_grid=param_grid,scoring='roc_auc',cv=5)
cv1.fit(X_train1,y_train)
```

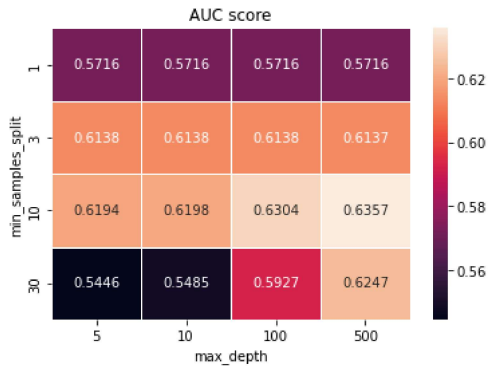
Out[17]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
             param_grid={'max_depth': [1, 3, 10, 30],
                         'min_samples_split': [5, 10, 100, 500]},
             scoring='roc_auc')
```

In [18]:
```
param_grid = {'max_depth':[1, 3, 10, 30],'min_samples_split':[5, 10, 100, 500]}
cv2 = GridSearchCV(estimator=dt,param_grid=param_grid,scoring='roc_auc',cv=5)
cv2.fit(X_train2,y_train)
```

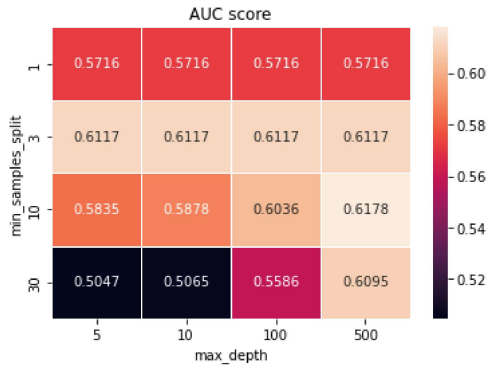
Out[18]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
             param_grid={'max_depth': [1, 3, 10, 30],
                         'min_samples_split': [5, 10, 100, 500]},
             scoring='roc_auc')
```

10. Find the best parameters and fit the model.

In [19]:
```
res = np.array(cv1.cv_results_['mean_test_score']).reshape(4,4)
grid = []
for i,depth in enumerate([1, 3, 10, 30]):
    for j,samples_split in enumerate([5, 10, 100, 500]):
        grid.append([depth,samples_split,res[i][j]])

df = pd.DataFrame(grid,columns=['max_depth','min_samples_split','mean_test_score'])
grid = df.pivot('max_depth','min_samples_split','mean_test_score')
ax = sns.heatmap(grid,annot=True,fmt="0.4f",linewidths=0.5)

plt.xlabel("max_depth")
plt.ylabel("min_samples_split")
plt.title("AUC score")
plt.show()
```

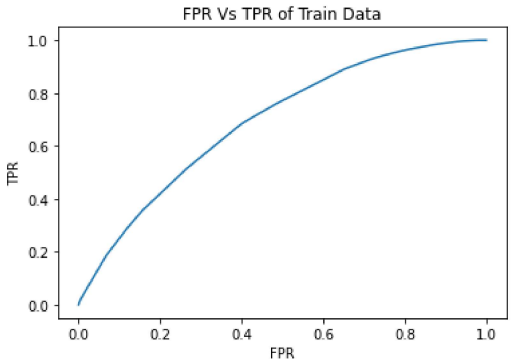


In [20]:
```
res = np.array(cv2.cv_results_['mean_test_score']).reshape(4,4)
grid = []
for i,depth in enumerate([1, 3, 10, 30]):
    for j,samples_split in enumerate([5, 10, 100, 500]):
        grid.append([depth,samples_split,res[i][j]])

df = pd.DataFrame(grid,columns=['max_depth','min_samples_split','mean_test_score'])
grid = df.pivot('max_depth','min_samples_split','mean_test_score')
ax = sns.heatmap(grid,annot=True,fmt="0.4f",linewidths=0.5)

plt.xlabel("max_depth")
plt.ylabel("min_samples_split")
plt.title("AUC score")
plt.show()
```

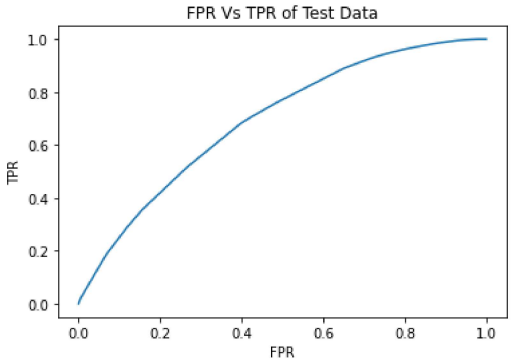


Plot ROC-AUC curve(using predict proba method)

In [21]:
```
dt = DecisionTreeClassifier(max_depth=cv1.best_params_['max_depth'],min_samples_split=cv1.best_params_['min_samples_split'])
dt.fit(X_train1,y_train)

pro_score_train = dt.predict_proba(X_train1)[:,1]
pro_score_test = dt.predict_proba(X_test1)[:,1]

# Plot the ROC-AUC curves using the probability predictions made on train and test data.
k_train = roc_curve(y_train,pro_score_train)
plt.plot(k_train[0], k_train[1])
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("FPR Vs TPR of Train Data")
plt.show()
print(auc(k_train[0],k_train[1]))


k_test = roc_curve(y_test,pro_score_test)
plt.plot(k_train[0], k_train[1])
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("FPR Vs TPR of Test Data")
plt.show()
print(auc(k_test[0],k_test[1]))
```




```
0.6929203111545635
```

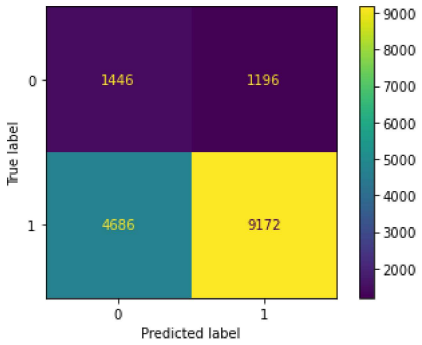



```
0.6370704662157282
```

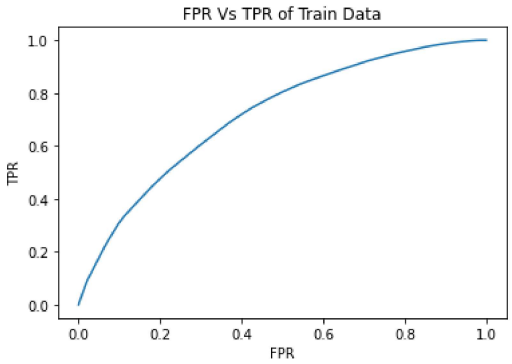
In [22]:
```
# Pick the best threshold among the probability estimates, such that it has to yield maximum value for TPR*(1-FPR)
best_thr_index = np.argmax((k_train[1]*(1-k_train[0])))
threshold = k_train[2][best_thr_index]
print("="*100)
# (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
print("the maximum value of tpr*(1-fpr)", max((k_train[1]*(1-k_train[0]))), "for threshold", np.round(threshold,3),"\n")

# Plot the confusion matrices(each for train and test data) afer encoding the predicted class labels, on the basis of the best th

# y_pred = []
# for prob in pro_score_train:
#     if prob > threshold:
#         y_pred.append(1)
#     else:
#         y_pred.append(0)


# ConfusionMatrixDisplay(confusion_matrix(y_train, y_pred)).plot()


y_pred = []
for prob in pro_score_test:
    if prob > threshold:
        y_pred.append(1)
    else:
        y_pred.append(0)


ConfusionMatrixDisplay(confusion_matrix(y_test, y_pred)).plot()
```

```
====================================================================================================
the maximum value of tpr*(1-fpr) 0.41022330923117156 for threshold 0.865
```

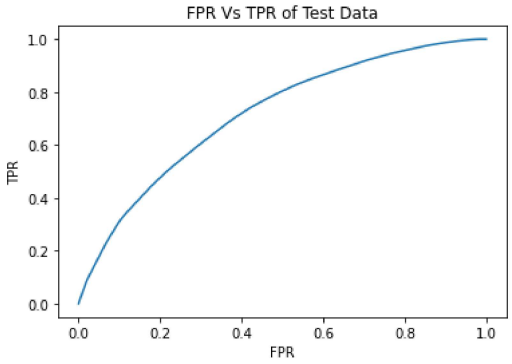
Out[22]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21a43f5af40>



In [23]:
```
dt = DecisionTreeClassifier(max_depth=cv2.best_params_['max_depth'],min_samples_split=cv2.best_params_['min_samples_split'])
dt.fit(X_train2,y_train)

pro_score_train = dt.predict_proba(X_train2)[:,1]
pro_score_test = dt.predict_proba(X_test2)[:,1]

# Plot the ROC-AUC curves using the probability predictions made on train and test data.
k_train = roc_curve(y_train,pro_score_train)
plt.plot(k_train[0], k_train[1])
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("FPR Vs TPR of Train Data")
plt.show()
print(auc(k_train[0],k_train[1]))


k_test = roc_curve(y_test,pro_score_test)
plt.plot(k_train[0], k_train[1])
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("FPR Vs TPR of Test Data")
plt.show()
print(auc(k_test[0],k_test[1]))
```




```
0.7174211642011598
```



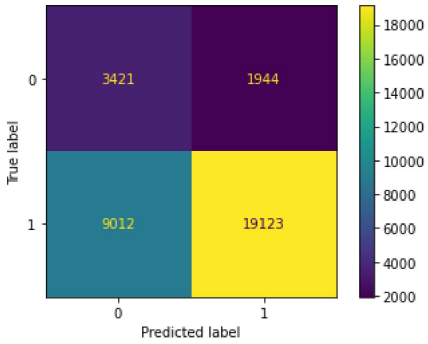

```
0.6256952889418345
```

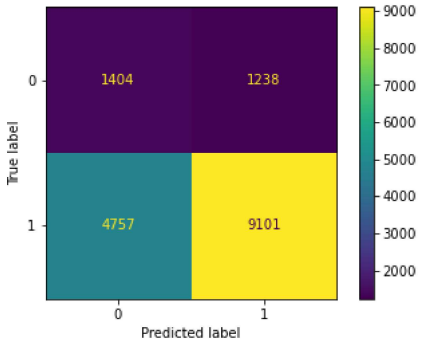
11. Plot confusion matrix based on best threshold value

In [24]:
```
# Pick the best threshold among the probability estimates, such that it has to yield maximum value for TPR*(1-FPR)
best_thr_index = np.argmax((k_train[1]*(1-k_train[0])))
threshold = k_train[2][best_thr_index]
print("="*100)
# (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((k_train[1]*(1-k_train[0]))), "for threshold", np.round(threshold,3),"\n")

# Plot the confusion matrices(each for train and test data) afer encoding the predicted class labels, on the basis of the best th

y_pred = []
FP_indices = [] #Indices of train datapoints which are False Positive
for i,prob in enumerate(pro_score_train):
    if prob > threshold:
        y_pred.append(1)
        if y_train[i] == 0:
            FP_indices.append(i)
    else:
        y_pred.append(0)

ConfusionMatrixDisplay(confusion_matrix(y_train, y_pred)).plot()

y_pred = []
for prob in pro_score_test:
    if prob > threshold:
        y_pred.append(1)
    else:
        y_pred.append(0)


ConfusionMatrixDisplay(confusion_matrix(y_test, y_pred)).plot()
```

```
==========================================================================================
the maximum value of tpr*(1-fpr) 0.4337113149869381 for threshold 0.855
```

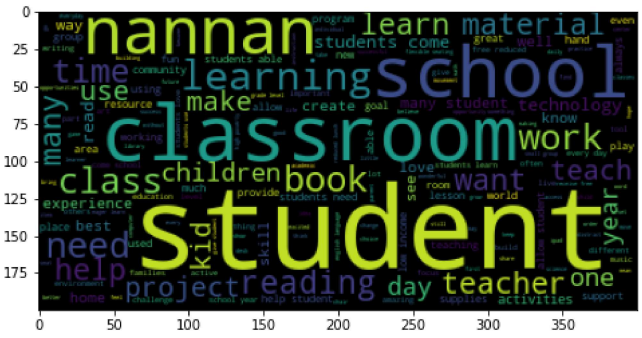
Out[24]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x21a44bbb610>





12. Find all the false positive data points and plot wordcloud of essay text and pdf of teacher_number_of_previously_posted_projects.

In [25]:
```
words = ""
pre_posts = []
price = []
for i in FP_indices:
    words+="".join((X_train['essay'].values[i]).lower())+" "
    price.append(X_train['price'].values[i])
    pre_posts.append(X_train['teacher_number_of_previously_posted_projects'].values[i])
# print(words)

from wordcloud import WordCloud
wordcloud = WordCloud().generate(words)
plt.figure(figsize = (8, 8))
plt.imshow(wordcloud)
plt.show()
```

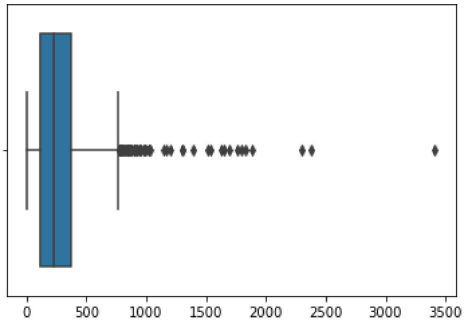


Plot the box plot with the `price` of these `false positive data points`

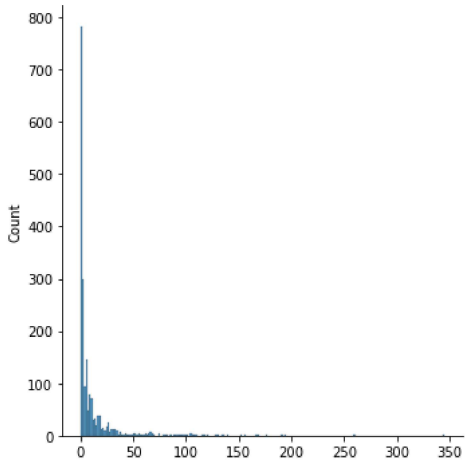
Plot the pdf with the `teacher_number_of_previously_posted_projects` of these `false positive data points`

In [26]:
```
price_df = pd.DataFrame(price)
print(price_df.describe())
sns.boxplot(price)
plt.show()
```

```
                0
count  1944.000000
mean    277.031564
std     247.006652
min       2.550000
25%     113.827500
50%     225.185000
75%     375.820000
max    3412.010000
```



In [27]:
```
sns.displot(pre_posts)
plt.show()
```



Type *Markdown* and LaTeX: α^2

In [28]:
```
# 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
```

Task - 2

1. select all non zero features
2. Update your dataset i.e. X_train,X_test and X_cv so that it contains all rows and only non zero features

In [29]:
```
print(len(dt.feature_importances_))
X_train_new = pd.DataFrame(X_train2.toarray())
X_test_new = pd.DataFrame(X_test2.toarray())

indices = []
for i,feature in enumerate(dt.feature_importances_):
    if feature == 0:
        indices.append(i)

X_train_new.drop(X_train_new.columns[indices], axis=1, inplace=True)
X_test_new.drop(X_test_new.columns[indices],axis=1,inplace=True)

print(X_train_new.shape)
print(X_test_new.shape)
```

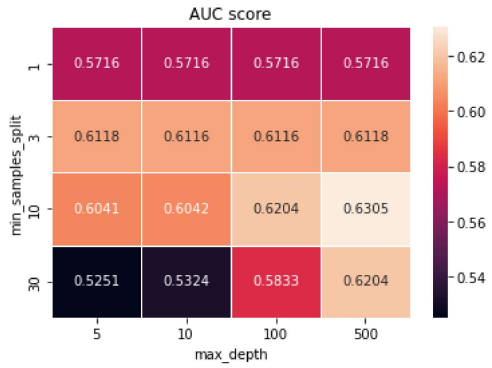
```
749
(33500, 92)
(16500, 92)
```

3. perform hyperparameter tuning and plot either heatmap or 3d plot.

In [30]:
```
param_grid = {'max_depth':[1, 3, 10, 30],'min_samples_split':[5, 10, 100, 500]}
cv2 = GridSearchCV(estimator=dt,param_grid=param_grid,scoring='roc_auc',cv=5)
cv2.fit(X_train_new,y_train)
```

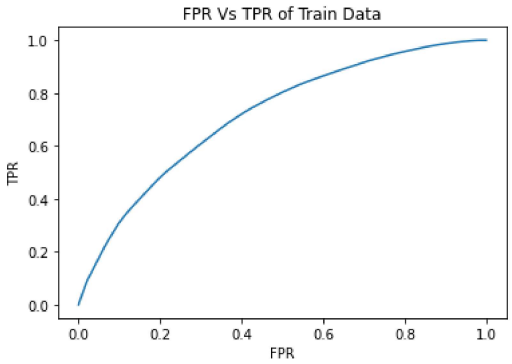
Out[30]:
```
GridSearchCV(cv=5,
             estimator=DecisionTreeClassifier(max_depth=10,
                                              min_samples_split=500),
             param_grid={'max_depth': [1, 3, 10, 30],
                         'min_samples_split': [5, 10, 100, 500]},
             scoring='roc_auc')
```

In [31]:
```
res = np.array(cv2.cv_results_['mean_test_score']).reshape(4,4)
grid = []
for i,depth in enumerate([1, 3, 10, 30]):
    for j,samples_split in enumerate([5, 10, 100, 500]):
        grid.append([depth,samples_split,res[i][j]])

df = pd.DataFrame(grid,columns=['max_depth','min_samples_split','mean_test_score'])
grid = df.pivot('max_depth','min_samples_split','mean_test_score')
ax = sns.heatmap(grid,annot=True,fmt="0.4f",linewidths=0.5)

plt.xlabel("max_depth")
plt.ylabel("min_samples_split")
plt.title("AUC score")
plt.show()
```

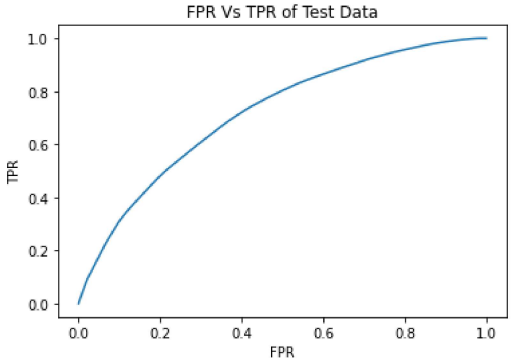


4. Fit the best model. Plot ROC AUC curve and confusion matrix similar to model 1.

In [32]:
```
dt = DecisionTreeClassifier(max_depth=cv2.best_params_['max_depth'],min_samples_split=cv2.best_params_['min_samples_split'])
dt.fit(X_train_new,y_train)

pro_score_train = dt.predict_proba(X_train_new)[:,1]
pro_score_test = dt.predict_proba(X_test_new)[:,1]

# Plot the ROC-AUC curves using the probability predictions made on train and test data.
k_train = roc_curve(y_train,pro_score_train)
plt.plot(k_train[0], k_train[1])
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("FPR Vs TPR of Train Data")
plt.show()
print(auc(k_train[0],k_train[1]))


k_test = roc_curve(y_test,pro_score_test)
plt.plot(k_train[0], k_train[1])
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("FPR Vs TPR of Test Data")
plt.show()
print(auc(k_test[0],k_test[1]))
```




```
0.7179662395277993
```




```
0.6262289269260648
```

In [33]:
```
from prettytable import PrettyTable
x = PrettyTable()

x.field_names = ['Model',"max_depth","min_samples_split","Max ROC-AUC",'Train ROC-AUC','Test ROC-AUC']

x.add_row(["tf_idf",500,10,0.6330,0.6949,0.6421])
x.add_row(["tfidf_w2v",500,10,0.6182,0.7118,0.6215])
x.add_row(["Non_Zero_features",500,10,0.6191,0.7118,0.6215])
print(x)
```

```
+-------------------+-----------+-------------------+-------------+---------------+--------------+
|       Model       | max_depth | min_samples_split | Max ROC-AUC | Train ROC-AUC | Test ROC-AUC |
+-------------------+-----------+-------------------+-------------+---------------+--------------+
|       tf_idf      |    500    |         10        |    0.633    |     0.6949    |    0.6421    |
|     tfidf_w2v     |    500    |         10        |    0.6182   |     0.7118    |    0.6215    |
| Non_Zero_features |    500    |         10        |    0.6191   |     0.7118    |    0.6215    |
+-------------------+-----------+-------------------+-------------+---------------+--------------+
```

- As we can see tfidf_w2v with 743 feature giving Test ROC-AUC 0.6215,But we are getting same Test ROC-AUC 0.6215 with just 81 feature which are having non-zero importance.

Type *Markdown* and LaTeX: α^2