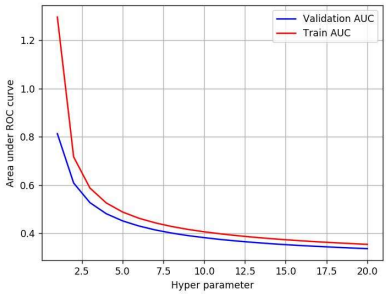
Assignment 6: Apply NB

1. Minimum data points need to be considered for people having 4GB RAM is **50k** and for 8GB RAM is **100k**
2. When you are using ramdomsearchcv or gridsearchcv you need not split the data into X_train,X_cv,X_test. As the above methods use kfold. The model will learn better if train data is more so splitting to X_train,X_test will suffice.
3. If you are writing for loops to tune your model then you need split the data into X_train,X_cv,X_test.
4. While splitting the data explore stratify parameter.
5. **Apply Multinomial NB on these feature sets**

    - Features that need to be considered
      essay
      while encoding essay, try to experiment with the max_features and n_grams parameter of vectorizers and see if it increases AUC score.
      categorical features
      - teacher_prefix
      - project_grade_category
      - school_state
      - clean_categories
      - clean_subcategories
      numerical features
      - price
      - teacher_number_of_previously_posted_projects
      while encoding the numerical features check this (https://imgur.com/ldZA1zg) and this (https://ac-classroom-production.s3.amazonaws.com/public/COMMENT/Annotation_2020-05-21_225912_0IyZzN8.jpg)

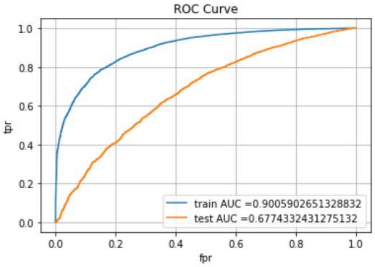
    - Set 1: categorical, numerical features + preprocessed_eassay (BOW)
    - Set 2: categorical, numerical features + preprocessed_eassay (TFIDF)
6. **The hyper paramter tuning(find best alpha:smoothing parameter)**

    - Consider alpha values in range: 10^-5 to 10^2 like [0.00001,0.0005, 0.0001,0.005,0.001,0.05,0.01,0.1,0.5,1,5,10,50,100]
    - Explore class_prior = [0.5, 0.5] parameter which can be present in MultinomialNB function(go through this (https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html)) then check how results might change.
    - 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
    - For hyper parameter tuning using k-fold cross validation(use GridsearchCV or RandomsearchCV)/simple cross validation data (write for loop to iterate over hyper parameter values)
    - 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



    -while plotting take log(alpha) on your X-axis so that it will be more readable

    - 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 = ??

-plot the confusion matrix in heatmaps, while plotting the confusion matrix go through the link (https://stackoverflow.com/questions/61748441/how-to-fix-the-values-displayed-in-a-confusion-matrix-in-exponential-form-to-nor)

7. find the top 20 features from either from feature Set 1 or feature Set 2 using values of `feature_log_prob_ ` parameter of `MultinomialNB` (https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html) and print **BOTH** positive as well as negative corresponding feature names.
   - go through the link (https://imgur.com/mWvE7gj)
8. 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

In [1]:
```
%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
```

2. Naive Bayes

1.1 Loading Data

In [2]:
```
#make sure you are loading atleast 50k datapoints
#you can work with features of preprocessed_data.csv for the assignment.
# If you want to add more features, you can add. (This is purely optional, not mandatory)

import pandas
data = pandas.read_csv('preprocessed_data.csv',nrows=50000)

print(len(data))
data.head(3)
```

```
50000
```

Out[2]:

	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	specialneed
2	ca	mrs	grades_prek_2	10	1	literacy_language	literac

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

In [3]:
```
# write your code in following steps for task 1
# 1. Split your data.
# 2. Perform Bag of Words Vectorization of text data.
# 3. Perform tfidf vectorization of text data.
# 4. perform one-hot encoding of categorical features.
# 5. perform normalization of numerical features
# 6. For set 1 stack up all the features using hstack()
# 7. For set 2 stack up all the features using hstack()
# 8. Perform hyperparameter tuning and represent the training and cross-validation AUC scores for different 'alpha' values, using
# 9. Find the best hyperparameter 'alpha' and fit the model. Plot ROC-AUC curve(by obtaining the probabilities using 'predict pro
# 10. Plot confusion matrix based on the best threshold value
# 11. Either for the model in set 1 or in set 2, print the top 20 features(you have to print the names, not the indexes) associat
# 12. Summarize your observations and compare both the models(ie., from set 1 and set 2) in terms of optimal hyperparameter value
# 13. You can use Prettytable or any other tabular format for comparison.


# 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
```

In [4]:
```
# drop project_approved column and make it new array Y
y = data['project_is_approved'].values
X = data.drop(['project_is_approved'], axis=1)
X.head(1)
```

Out[4]:

	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

In [5]:
```
# 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)
#X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33, stratify=y_train)
print(X_train.shape, y_train.shape)
#print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)
```

```
(33500, 8) (33500,)
(16500, 8) (16500,)
```

In [6]:
```
# Split the dataset
# 1) If you want to apply simple cross-validation, split the dataset into 3 parts (ie., train, CV and test sets)
# 2) If you want to apply K-fold CV (or) GridSearch Cross Validation (or) Randomized Search Cross Validation, just split the data
```

1.3 Make Data Model Ready: encoding essay, and project_title

In [7]:
```
# Apply Bag of Words (BOW) vectorization on 'Preprocessed_Essay'

# Encoding of Essay using Bag of words
vectorizer = CountVectorizer(min_df=50)
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_essay_bow = vectorizer.transform(X_train['essay'].values)
#X_cv_essay_bow = vectorizer.transform(X_cv['essay'].values)
X_test_essay_bow = vectorizer.transform(X_test['essay'].values)

print("After vectorizations")
print(X_train_essay_bow.shape, y_train.shape)
#print(X_cv_essay_bow.shape, y_cv.shape)
print(X_test_essay_bow.shape, y_test.shape)
```

```
After vectorizations
(33500, 4906) (33500,)
(16500, 4906) (16500,)
```

In [8]:
```
# Apply TF-IDF vectorization on 'Preprocessed_Essay'

# Encoding of Essay using TF-IDF

from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=50)
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_essay_tfidf = vectorizer.transform(X_train['essay'].values)
#X_cv_essay_tfidf = vectorizer.transform(X_cv['essay'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['essay'].values)

print("After vectorizations")
print(X_train_essay_tfidf.shape, y_train.shape)
#print(X_cv_essay_tfidf.shape, y_cv.shape)
print(X_test_essay_tfidf.shape, y_test.shape)
```

```
After vectorizations
(33500, 4906) (33500,)
(16500, 4906) (16500,)
```

1.4 Make Data Model Ready: encoding numerical, categorical features

In [9]:
```
# 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_cv_cat_feat = vectorizer.transform(X_cv[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_cv_cat_feat.shape, y_cv.shape)
print(X_test_cat_feat.shape, y_test.shape)

from sklearn.preprocessing import Normalizer
scaler = Normalizer()

numeric = ['price','teacher_number_of_previously_posted_projects']

scaler.fit(X_train[numeric])
X_train_numeric = scaler.transform(X_train[numeric])
#X_cv_numeric = scaler.transform(X_cv[numeric])
X_test_numeric = scaler.transform(X_test[numeric])
```

```
After vectorizations
(33500, 440) (33500,)
(16500, 440) (16500,)
```

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

Apply NB 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

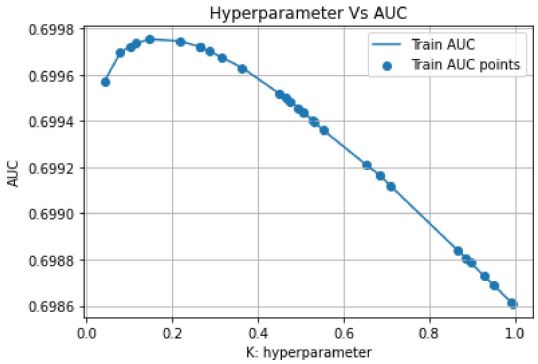
Set 1

In [10]:
```
# Perform Hyperparameter Tuning.

from scipy.sparse import hstack
X_train1 = hstack((X_train_essay_bow,X_train_cat_feat,X_train_numeric)).tocsr()
#X_cv1 = hstack((X_cv_essay_bow,X_cv_cat_feat,X_cv_numeric)).tocsr()
X_test1 = hstack((X_test_essay_bow,X_test_cat_feat,X_test_numeric)).tocsr()

print("Shape final repesentation in set 1:")
print(X_train1.shape,y_train.shape)
#print(X_cv1.shape,y_cv.shape)
print(X_test1.shape,y_test.shape)

from  sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import GridSearchCV

mnb = MultinomialNB()
mnb.get_params().keys()
mnb.fit(X_train1.toarray(),y_train)
param_grid = {"alpha":sorted(list(np.random.random(30)))}
gridSearch = GridSearchCV(mnb,param_grid,scoring='roc_auc',cv=4,n_jobs=-1)
gridSearch.fit(X_train1.toarray(),y_train)
```

```
Shape final repesentation in set 1:
(33500, 5348) (33500,)
(16500, 5348) (16500,)
```

Out[10]:
```
GridSearchCV(cv=4, estimator=MultinomialNB(), n_jobs=-1,
             param_grid={'alpha': [0.04147089099732948, 0.07865657556640504,
                                   0.1008932356439316, 0.1149804319320904,
                                   0.14613629912400505, 0.2193268201727958,
                                   0.2651406776207802, 0.2652056523624359,
                                   0.2875816413143344, 0.31438333829540177,
                                   0.36354505169956486, 0.450597211292845,
                                   0.46600295930261604, 0.4762027252118176,
                                   0.49484382621123957, 0.5067196950875067,
                                   0.507758182973228, 0.529161664685517,
                                   0.5311838344426396, 0.552240712034293,
                                   0.6542133898050556, 0.6842134341524992,
                                   0.7109943596989398, 0.8657688849697988,
                                   0.8864649045906177, 0.8969841804396943,
                                   0.9293841672589291, 0.9495342738485781,
                                   0.9921671329654522, 0.9945492537573607]},
             scoring='roc_auc')
```

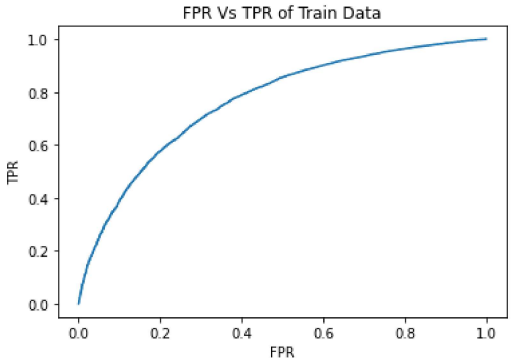
In [11]:
```
#print(gridSearch.cv_results_)

# Plot the training and the CV AUC scores, for different values of 'alpha', using a 2D line plot
x = list(param_grid["alpha"])
y = list(gridSearch.cv_results_['mean_test_score'])

plt.plot(x, y, label='Train AUC')
#plt.xticks(x)
plt.scatter(x, y, label='Train AUC points')

plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyperparameter Vs AUC")
plt.grid()
plt.show()

print("Best alpha is : ",x[y.index(max(y))])
print("Maximum AUC with training data:",max(y))
```



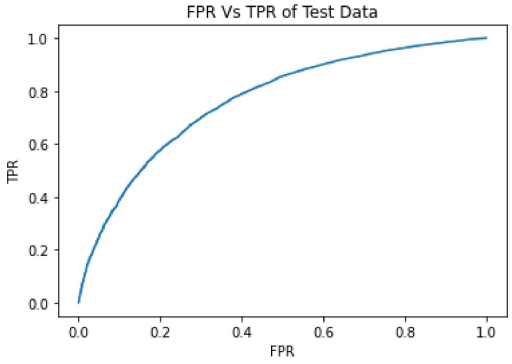

```
Best alpha is :  0.14613629912400505
Maximum AUC with training data: 0.6997541115322528
```

In [12]:
```
# Obtain the optimal value for 'alpha' and using the obtained optimal 'alpha' value, fit a multinomial naive bayes model, on the
# Note: If you have split the datase into 3 parts (ie., train, cv and test sets) in the beginning, then the training data for thi
# Make class label and probability predictions on the train and test data.

from  sklearn.naive_bayes import MultinomialNB
mnb = MultinomialNB(alpha=x[y.index(max(y))])
mnb.fit(X_train1.toarray(),y_train)


# To extract particular column in numpy
# https://stackoverflow.com/questions/8386675/extracting-specific-columns-in-numpy-array

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

In [13]:
```
# 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]))
```




```
0.7648362152191595
```




```
0.6997340085864968
```

In [14]:
```
# 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("="*200)
# (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)

print("Train confusion matrix : ")
print(confusion_matrix(y_train, y_pred))
print("\n")

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

print("Test confusion matrix : ")
print(confusion_matrix(y_test, y_pred))
```

```
========================================================================================================================================================
========================================================================
the maximum value of tpr*(1-fpr) 0.49037464985008544 for threshold 0.773

Train confusion matrix :
[[ 3682  1683]
 [ 8033 20102]]


Test confusion matrix :
[[1581 1061]
 [4181 9677]]
```

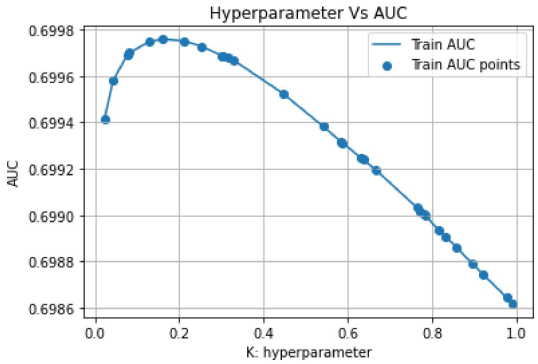
Set 2

In [15]:
```
# Perform Hyperparameter Tuning.

from scipy.sparse import hstack
X_train2 = hstack((X_train_essay_tfidf,X_train_cat_feat,X_train_numeric)).tocsr()
#X_cv2 = hstack((X_cv_essay_tfidf,X_cv_cat_feat,X_cv_numeric)).tocsr()
X_test2 = hstack((X_test_essay_tfidf,X_test_cat_feat,X_test_numeric)).tocsr()

print("Shape final repesentation in set 1:")
print(X_train2.shape,y_train.shape)
#print(X_cv2.shape,y_cv.shape)
print(X_test2.shape,y_test.shape)


from  sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import GridSearchCV

mnb = MultinomialNB()
mnb.fit(X_train1.toarray(),y_train)
param_grid = {"alpha":sorted(list(np.random.random(30)))}
gridSearch = GridSearchCV(mnb,param_grid,scoring='roc_auc',cv=4,n_jobs=-1)
gridSearch.fit(X_train1.toarray(),y_train)
```

```
Shape final repesentation in set 1:
(33500, 5348) (33500,)
(16500, 5348) (16500,)
```

Out[15]:
```
GridSearchCV(cv=4, estimator=MultinomialNB(), n_jobs=-1,
             param_grid={'alpha': [0.022341730009598493, 0.04358718459048383,
                                   0.07699998856877932, 0.08138967261651886,
                                   0.12916344226051157, 0.16028419352610979,
                                   0.210835003101281, 0.2540923070215414,
                                   0.3001945421476264, 0.30386971724520717,
                                   0.316626828469897, 0.3286867107274982,
                                   0.44559833952906636, 0.5401698730323756,
                                   0.5820224038084079, 0.5876350066035392,
                                   0.6292584947150821, 0.6360157509714683,
                                   0.665040125232417, 0.7636209029531866,
                                   0.7709926575920331, 0.7789434855857431,
                                   0.7819158825827188, 0.8150379251956059,
                                   0.8305126049601764, 0.8559672562195834,
                                   0.8955505186625443, 0.9188879476311976,
                                   0.9755090987091007, 0.9894621173839626]},
             scoring='roc_auc')
```

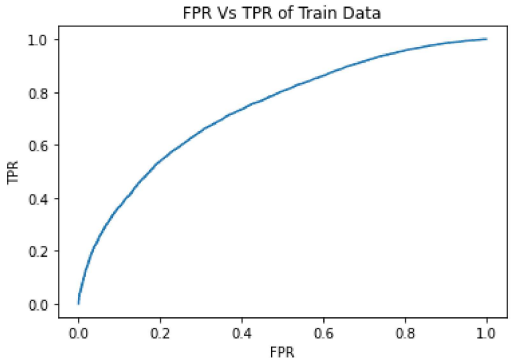
In [16]:
```
#print(gridSearch.cv_results_)

# Plot the training and the CV AUC scores, for different values of 'alpha', using a 2D line plot

x = list(param_grid["alpha"])
y = list(gridSearch.cv_results_['mean_test_score'])

plt.plot(x, y, label='Train AUC')
#plt.xticks(x)
plt.scatter(x, y, label='Train AUC points')

plt.legend()
plt.xlabel("K: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyperparameter Vs AUC")
plt.grid()
plt.show()

print("Best alpha is : ",x[y.index(max(y))])
print("Maximum AUC with training data:",max(y))
```



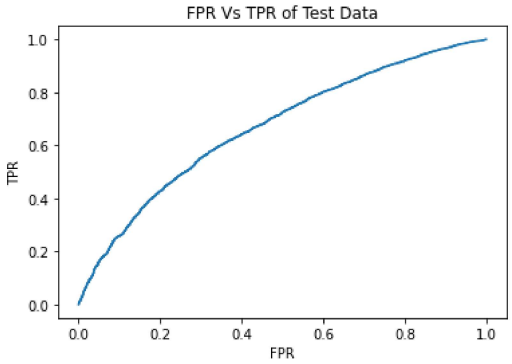

```
Best alpha is :  0.16028419352610979
Maximum AUC with training data: 0.6997585767346723
```

In [17]:
```
# Obtain the optimal value for 'alpha' and using the obtained optimal 'alpha' value, fit a multinomial naive bayes model, on the
# Note: If you have split the datase into 3 parts (ie., train, cv and test sets) in the beginning, then the training datafor this
# Make class label and probability predictions on the train and test data.
from  sklearn.naive_bayes import MultinomialNB
mnb = MultinomialNB(alpha=x[y.index(max(y))])
mnb.fit(X_train2.toarray(),y_train)

# To extract particular column in numpy
# https://stackoverflow.com/questions/8386675/extracting-specific-columns-in-numpy-array

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

In [18]:
```
# 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_test[0], k_test[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]))
```




```
0.7375615073840992
```




```
0.6680194208391832
```

In [19]:
```
# https://stackoverflow.com/questions/61034101/scikit-learn-change-threshold-in-confusion-matrix

# 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("="*200)
# (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)

print("Train confusion matrix : ")
print(confusion_matrix(y_train, y_pred))
print("\n")

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

print("Test confusion matrix : ")
print(confusion_matrix(y_test, y_pred))
```

```
========================================================================================================================================================
========================================================================
the maximum value of tpr*(1-fpr) 0.4579507834927824 for threshold 0.847

Train confusion matrix :
[[ 3717  1648]
 [ 9539 18596]]


Test confusion matrix :
[[1563 1079]
 [4880 8978]]
```

In [20]:
```
# Either from set 1 (or) set 2, print the names of the top 20 features associated with the positive and negative classes each. (Y
# https://stackoverflow.com/questions/50526898/how-to-get-feature-importance-in-naive-bayes

# print(np.take(vectorizer.get_feature_names(), neg_class_prob_sorted[:10]))
# print(np.take(vectorizer.get_feature_names(), pos_class_prob_sorted[:10]))
# X_train2 = hstack((X_train_essay_tfidf,X_train_cat_feat,X_train_numeric)).tocsr()

neg_class_prob_sorted = mnb.feature_log_prob_[0, :].argsort()[::-1]
pos_class_prob_sorted = mnb.feature_log_prob_[1, :].argsort()[::-1]

neg_class_prob_sorted = neg_class_prob_sorted[:20]
pos_class_prob_sorted = pos_class_prob_sorted[:20]

essay_lst = []
cat_lst = []
numeric_lst = []
for k in pos_class_prob_sorted:
    if k < X_train_essay_tfidf.shape[1]:
        essay_lst.append(k)
    elif k < (X_train_essay_tfidf.shape[1]+X_train_cat_feat.shape[1]):
        cat_lst.append(k-X_train_essay_tfidf.shape[1])
    else:
        numeric_lst.append(k-(X_train_essay_tfidf.shape[1]+X_train_cat_feat.shape[1]))

# print(X_train_essay_tfidf.shape[1])
# print(X_train_cat_feat.shape[1])
# print(X_train_numeric.shape[1])

# print(essay_lst)
# print(cat_lst)
# print(numeric_lst)

print("Names of the top 20 features associated with the positive class : ")
print(list(np.take(vectorizer.get_feature_names(),essay_lst))+list(np.take(vectorizer1.get_feature_names(), cat_lst))+list(np.tak
print("\n")

essay_lst = []
cat_lst = []
numeric_lst = []
for k in neg_class_prob_sorted:
    if k < X_train_essay_tfidf.shape[1]:
        essay_lst.append(k)
    elif k < (X_train_essay_tfidf.shape[1]+X_train_cat_feat.shape[1]):
        cat_lst.append(k-X_train_essay_tfidf.shape[1])
    else:
        numeric_lst.append(k-(X_train_essay_tfidf.shape[1]+X_train_cat_feat.shape[1]))


print("Names of the top 20 features associated with the negative class : ")
print(list(np.take(vectorizer.get_feature_names(),essay_lst))+list(np.take(vectorizer1.get_feature_names(), cat_lst))+list(np.tak
print("\n")
```

```
Names of the top 20 features associated with the positive class :
['students', 'x0_mrs', 'x1_grades_prek_2', 'x0_ms', 'x1_grades_3_5', 'x3_literacy_language', 'x1_grades_6_8', 'x2_ca', 'x3_math
_science', 'x3_health_sports', 'x3_literacy_language math_science', 'x1_grades_9_12', 'x0_mr', 'x4_literacy', 'x4_literacy math
ematics', 'x2_tx', 'x2_fl', 'x2_ny', 'price', 'teacher_number_of_previously_posted_projects']


Names of the top 20 features associated with the negative class :
['students', 'school', 'x0_mrs', 'x1_grades_prek_2', 'x0_ms', 'x1_grades_3_5', 'x3_literacy_language', 'x3_math_science', 'x1_g
rades_6_8', 'x2_ca', 'x3_health_sports', 'x3_literacy_language math_science', 'x0_mr', 'x1_grades_9_12', 'x2_tx', 'x4_literac
y', 'x4_literacy mathematics', 'x2_fl', 'x4_mathematics', 'price']
```

3. Summary

- In 1st model we used Bag of Words and 2nd model we used TF-IDF to represent essay in given dataset.
- Both models worked almost same, but BOW gave slightly better result.
- Mainly beacuase as we can see from Top 20 features of each class are not from essay ,but mostly catogorical and numeric features of dataset.

In [21]:
```
from prettytable import PrettyTable

x = PrettyTable()
# x.field_names = ["Percentile", "Approved Projects", "Not Approved Projects"]

# for i in range(0,101,5):
#     x.add_row([i,np.round(np.percentile(approved_price,i), 3), np.round(np.percentile(rejected_price,i), 3)])
# print(x)

# ======Set 1=======
# best alpha : .2233
# max AUC : 0.694
# train AUC : 0.7613
# test AUC : 0.6984
# Max trp*(1-frp) : 0.486
# for threshold : 0.792


# =====Set-2========
# best alpha : 0.2552
# max AUC : 0.694
# train AUC : 0.731
# test AUC = 0.659
# max trp*(1-frp) : 0.449
# for threshold : 0.848

x.field_names = ['Model','Best alpha','Max AUC','Train ROC-AUC','Test ROC-AUC','Max trp*(1-fpr)','Threshold']

x.add_row(['BOW',0.2233,0.694,0.7613,0.6984,0.486,0.792])
x.add_row(['TF-IDF',0.2552,0.694,0.731,0.659,0.449,0.848])

print(x)
```

```
+--------+------------+---------+---------------+--------------+-----------------+-----------+
| Model  | Best alpha | Max AUC | Train ROC-AUC | Test ROC-AUC | Max trp*(1-fpr) | Threshold |
+--------+------------+---------+---------------+--------------+-----------------+-----------+
|  BOW   |   0.2233   |  0.694  |     0.7613    |    0.6984    |      0.486      |   0.792   |
| TF-IDF |   0.2552   |  0.694  |     0.731     |    0.659     |      0.449      |   0.848   |
+--------+------------+---------+---------------+--------------+-----------------+-----------+
```

In [ ]:
```

```