Assignment-4 Apply Naive Bayes on Donors Choose dataset.

In [1]:

```
import warnings
warnings.filterwarnings("ignore")
import sqlite3
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 TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer
import re
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle
from tqdm import tqdm
import os
from plotly import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from collections import Counter
```

1.1 Loading Data

In [2]:

```
data = pd.read_csv('preprocessed_data.csv', nrows=50000)
data.head(2)
```

Out[2]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_s
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.	FL

2 rows × 24 columns

In [3]:

```
# To ensure we are dealing with an imbalanced data set.
data['project_is_approved'].value_counts()
```

Out[3]:

```
1    42286
0     7714
Name: project_is_approved, dtype: int64
```

In [4]:

```
y = data['project_is_approved'].values
X = data.drop(['project_is_approved'], axis=1)
X.head(1)
```

Out[4]:

	Unnamed: 0	id	teacher_id	teacher_prefix	school_sta
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.	IN

1 rows × 23 columns

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

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, strat
ify=y_train)
```

1.3 Make Data Model Ready: encoding essay, and project_title

1.3.1 Vectorizing preprocessed essays & project_title using BOW

In [96]:

```
# preprocessed essays
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)


vectorizer = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['preprocessed_essays'].values)  # fit has to happen only on trai
n data

# we use the fit CountVectorizer to convert the text to vector
X_train_essay_bow = vectorizer.transform(X_train['preprocessed_essays'].values)
X_cv_essay_bow = vectorizer.transform(X_cv['preprocessed_essays'].values)
X_test_essay_bow = vectorizer.transform(X_test['preprocessed_essays'].values)
```

```
(22445, 23) (22445,)
(11055, 23) (11055,)
(16500, 23) (16500,)
==========================================================================
==========================
```

In [97]:

```
print("After vectorization")
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)
print("="*100)
```

```
After vectorization
(22445, 5000) (22445,)
(11055, 5000) (11055,)
(16500, 5000) (16500,)
==========================================================================
==========================
```

In [98]:

```
#project_title
vectorizer1 = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer1.fit(X_train['preprocessed_titles'].values.astype('U')) #https://stackoverfl
ow.com/questions/39303912/tfidfvectorizer-in-scikit-learn-valueerror-np-nan-is-an-inval
id-document

X_train_title_bow = vectorizer1.transform(X_train['preprocessed_titles'].values.astype(
'U'))
X_cv_title_bow = vectorizer1.transform(X_cv['preprocessed_titles'].values.astype('U'))
X_test_title_bow = vectorizer1.transform(X_test['preprocessed_titles'].values.astype(
'U'))
```

In [99]:

```
print("After vectorization")
print(X_train_title_bow.shape, y_train.shape)
print(X_cv_title_bow.shape, y_cv.shape)
print(X_test_title_bow.shape, y_test.shape)
print("="*100)
```

```
After vectorization
(22445, 1687) (22445,)
(11055, 1687) (11055,)
(16500, 1687) (16500,)
==========================================================================
==========================
```

1.3.2 Vectorizing preprocessed essays & project_title using TFIDF

In [16]:

```
#TFIDF for preprocessed_essays
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['preprocessed_essays'].values)

X_train_essay_tfidf = vectorizer.transform(X_train['preprocessed_essays'].values)
X_cv_essay_tfidf = vectorizer.transform(X_cv['preprocessed_essays'].values)
X_test_essay_tfidf = vectorizer.transform(X_test['preprocessed_essays'].values)
```

In [18]:

```
print("After vectorization")
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)
print("="*100)
```

```
After vectorization
(22445, 5000) (22445,)
(11055, 5000) (11055,)
(16500, 5000) (16500,)
==========================================================================
==========================
```

In [20]:

```
#TFIDF for preprocessed_titles
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train['preprocessed_titles'].values.astype('U'))

X_train_titles_tfidf = vectorizer.transform(X_train['preprocessed_titles'].values.astyp
e('U'))
X_cv_titles_tfidf = vectorizer.transform(X_cv['preprocessed_titles'].values.astype('U'
))
X_test_titles_tfidf = vectorizer.transform(X_test['preprocessed_titles'].values.astype(
'U'))
```

In [21]:

```
print("After vectorization")
print(X_train_titles_tfidf.shape, y_train.shape)
print(X_cv_titles_tfidf.shape, y_cv.shape)
print(X_test_titles_tfidf.shape, y_test.shape)
print("="*100)
```

```
After vectorization
(22445, 1687) (22445,)
(11055, 1687) (11055,)
(16500, 1687) (16500,)
==========================================================================
==========================
```

1.4 Make Data Model Ready: encoding numerical, categorical features

1.4.1 Encoding categorical features: School State

In [100]:

```
vectorizer2 = CountVectorizer()
vectorizer2.fit(X_train['school_state'].values) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_state = vectorizer2.transform(X_train['school_state'].values)
X_cv_state = vectorizer2.transform(X_cv['school_state'].values)
X_test_state = vectorizer2.transform(X_test['school_state'].values)

print("After vectorizations")
print(X_train_state.shape, y_train.shape)
print(X_cv_state.shape, y_cv.shape)
print(X_test_state.shape, y_test.shape)
print(vectorizer2.get_feature_names())
print("="*100)
```

```
After vectorizations
(22445, 51) (22445,)
(11055, 51) (11055,)
(16500, 51) (16500,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi',
'ia', 'id', 'il', 'in', 'ks', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'm
o', 'ms', 'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', 'nv', 'ny', 'oh', 'o
k', 'or', 'pa', 'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'w
i', 'wv', 'wy']
==========================================================================
==========================
```

1.4.2 Encoding categorical features: teacher_prefix

In [101]:

```
vectorizer3 = CountVectorizer()
vectorizer3.fit(X_train['teacher_prefix'].values)

X_train_teacher = vectorizer3.transform(X_train['teacher_prefix'].values)
X_cv_teacher = vectorizer3.transform(X_cv['teacher_prefix'].values)
X_test_teacher = vectorizer3.transform(X_test['teacher_prefix'].values)

print("After vectorizations")
print(X_train_teacher.shape, y_train.shape)
print(X_cv_teacher.shape, y_cv.shape)
print(X_test_teacher.shape, y_test.shape)
print(vectorizer3.get_feature_names())
print("="*100)
```

```
After vectorizations
(22445, 6) (22445,)
(11055, 6) (11055,)
(16500, 6) (16500,)
['dr', 'mr', 'mrs', 'ms', 'none', 'teacher']
===========================================================================
=========================
```

1.4.3 Encoding categorical features: project_grade_category

In [102]:

```
#This step is to intialize a vectorizer with vocab from train data
#Ref: https://www.kaggle.com/shashank49/donors-choose-knn#Concatinating-all-features-(T
FIDF)
from collections import Counter
my_counter = Counter()
for word in X_train['preprocessed_project_grade_category'].values:
    my_counter.update([word[i:i+14] for i in range(0, len(word),14)]) #https://www.geek
sforgeeks.org/python-string-split/

# dict sort by value python: https://stackoverflow.com/a/613218/4084039
project_grade_category_dict = dict(my_counter)
sorted_project_grade_category_dict = dict(sorted(project_grade_category_dict.items(), k
ey=lambda kv: kv[1]))
```

In [103]:

```
vectorizer4 = CountVectorizer(vocabulary=list(sorted_project_grade_category_dict.keys
()), lowercase=False, binary=True,max_features=4)
vectorizer4.fit(X_train['preprocessed_project_grade_category'].values) # fit has to hap
pen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_grade = vectorizer4.transform(X_train['preprocessed_project_grade_category'].va
lues)
X_cv_grade = vectorizer4.transform(X_cv['preprocessed_project_grade_category'].values)
X_test_grade = vectorizer4.transform(X_test['preprocessed_project_grade_category'].valu
es)

print("After vectorizations")
print(X_train_grade.shape, y_train.shape)
print(X_cv_grade.shape, y_cv.shape)
print(X_test_grade.shape, y_test.shape)
print(vectorizer4.get_feature_names())
```

```
After vectorizations
(22445, 4) (22445,)
(11055, 4) (11055,)
(16500, 4) (16500,)
['grades_9_12', 'grades_6_8', 'grades_3_5', 'grades_prek_2']
```

1.4.4 Encoding categorical features: clean_categories

In [104]:

```
vectorizer5 = CountVectorizer()
vectorizer5.fit(X_train['clean_categories'].values) # fit has to happen only on train d
ata

# we use the fitted CountVectorizer to convert the text to vector
X_train_cat = vectorizer5.transform(X_train['clean_categories'].values)
X_cv_cat = vectorizer5.transform(X_cv['clean_categories'].values)
X_test_cat = vectorizer5.transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_cat.shape, y_train.shape)
print(X_cv_cat.shape, y_cv.shape)
print(X_test_cat.shape, y_test.shape)
print(vectorizer5.get_feature_names())
print("="*100)
```

```
After vectorizations
(22445, 9) (22445,)
(11055, 9) (11055,)
(16500, 9) (16500,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics', 'lit
eracy_language', 'math_science', 'music_arts', 'specialneeds', 'warmth']
==========================================================================
==========================
```

1.4.5 Encoding categorical features: clean_subcategories

In [105]:

```
vectorizer6 = CountVectorizer()
vectorizer6.fit(X_train['clean_subcategories'].values) # fit has to happen only on trai
n data

# we use the fitted CountVectorizer to convert the text to vector
X_train_subcat = vectorizer6.transform(X_train['clean_subcategories'].values)
X_cv_subcat = vectorizer6.transform(X_cv['clean_subcategories'].values)
X_test_subcat = vectorizer6.transform(X_test['clean_subcategories'].values)

print("After vectorizations")
print(X_train_subcat.shape, y_train.shape)
print(X_cv_subcat.shape, y_cv.shape)
print(X_test_subcat.shape, y_test.shape)
print(vectorizer6.get_feature_names())
print("="*100)
```

```
After vectorizations
(22445, 30) (22445,)
(11055, 30) (11055,)
(16500, 30) (16500,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_governmen
t', 'college_careerprep', 'communityservice', 'earlydevelopment', 'economi
cs', 'environmentalscience', 'esl', 'extracurricular', 'financialliterac
y', 'foreignlanguages', 'gym_fitness', 'health_lifescience', 'health_welln
ess', 'history_geography', 'literacy', 'literature_writing', 'mathematic
s', 'music', 'nutritioneducation', 'other', 'parentinvolvement', 'performi
ngarts', 'socialsciences', 'specialneeds', 'teamsports', 'visualarts', 'wa
rmth']
==========================================================================
==========================
```

1.4.6 Encoding numerical features: Price

In [34]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
#this will rise an error Expected 2D array, got 1D array instead:
normalizer.fit(X_train['price'].values.reshape(1,-1))

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

print("After vectorizations")
print((X_train_price_norm.reshape(-1,1)).shape, y_train.shape)
print((X_cv_price_norm.reshape(-1,1)).shape, y_cv.shape)
print((X_test_price_norm.reshape(-1,1)).shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
==========================================================================
==========================
```

1.4.6 Encoding numerical features: Quantity

In [35]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
#this will rise an error Expected 2D array, got 1D array instead:
normalizer.fit(X_train['quantity'].values.reshape(1,-1))

X_train_quantity_norm = normalizer.transform(X_train['quantity'].values.reshape(1,-1))
X_cv_quantity_norm = normalizer.transform(X_cv['quantity'].values.reshape(1,-1))
X_test_quantity_norm = normalizer.transform(X_test['quantity'].values.reshape(1,-1))

print("After vectorizations")
print((X_train_quantity_norm.reshape(-1,1)).shape, y_train.shape)
print((X_cv_quantity_norm.reshape(-1,1)).shape, y_cv.shape)
print((X_test_quantity_norm.reshape(-1,1)).shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
===========================================================================
=========================
```

1.4.8 Encoding numerical features: teacher_number_of_previously_posted_projects

In [36]:

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
# normalizer.fit(X_train['price'].values)
#this will rise an error Expected 2D array, got 1D array instead:
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.reshape(1
,-1))

X_train_projects_norm = normalizer.transform(X_train['teacher_number_of_previously_post
ed_projects'].values.reshape(1,-1))
X_cv_projects_norm = normalizer.transform(X_cv['teacher_number_of_previously_posted_pro
jects'].values.reshape(1,-1))
X_test_projects_norm = normalizer.transform(X_test['teacher_number_of_previously_posted
_projects'].values.reshape(1,-1))

print("After vectorizations")
print((X_train_projects_norm.reshape(-1,1)).shape, y_train.shape)
print((X_cv_projects_norm.reshape(-1,1)).shape, y_cv.shape)
print((X_test_projects_norm.reshape(-1,1)).shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(22445, 1) (22445,)
(11055, 1) (11055,)
(16500, 1) (16500,)
==========================================================================
==========================
```

1.5 Concatinating all the features

Set 1: Using categorical features + numerical features + preprocessed_titles(BOW) + preprocessed_essays(BOW)

In [39]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_tr_bow = hstack((X_train_essay_bow, X_train_title_bow, X_train_state, X_train_teacher
, X_train_grade, X_train_cat, X_train_subcat, X_train_price_norm, X_train_quantity_norm
, X_train_projects_norm )).tocsr()

X_cv_bow = hstack((X_cv_essay_bow, X_cv_title_bow, X_cv_state, X_cv_teacher, X_cv_grade
, X_cv_cat, X_cv_subcat, X_cv_price_norm, X_cv_quantity_norm, X_cv_projects_norm )).toc
sr()

X_test_bow = hstack((X_test_essay_bow, X_test_title_bow, X_test_state, X_test_teacher,
X_test_grade, X_test_cat, X_test_subcat, X_test_price_norm, X_test_quantity_norm, X_tes
t_projects_norm )).tocsr()

print("Final Data Matrix")
print(X_tr_bow.shape, y_train.shape)
print(X_cv_bow.shape, y_train.shape)
print(X_test_bow.shape, y_train.shape)
```

```
Final Data Matrix
(22445, 6790) (22445,)
(11055, 6790) (22445,)
(16500, 6790) (22445,)
```

Set 2: Using categorical features + numerical features + preprocessed_titles(TFIDF) + preprocessed_essays(TFIDF)

In [40]:

```
# merge two sparse matrices: https://stackoverflow.com/a/19710648/4084039
from scipy.sparse import hstack

X_tr_tfidf = hstack((X_train_essay_tfidf, X_train_titles_tfidf, X_train_state, X_train_
teacher, X_train_grade, X_train_cat, X_train_subcat, X_train_price_norm, X_train_quanti
ty_norm, X_train_projects_norm )).tocsr()

X_cv_tfidf = hstack((X_cv_essay_tfidf, X_cv_titles_tfidf, X_cv_state, X_cv_teacher, X_c
v_grade, X_cv_cat, X_cv_subcat, X_cv_price_norm, X_cv_quantity_norm, X_cv_projects_norm
)).tocsr()

X_test_tfidf = hstack((X_test_essay_tfidf, X_test_titles_tfidf, X_test_state, X_test_te
acher, X_test_grade, X_test_cat, X_test_subcat, X_test_price_norm, X_test_quantity_norm
, X_test_projects_norm )).tocsr()

print("Final Data Matrix")
print(X_tr_tfidf.shape, y_train.shape)
print(X_cv_tfidf.shape, y_train.shape)
print(X_test_tfidf.shape, y_train.shape)
```

```
Final Data Matrix
(22445, 6790) (22445,)
(11055, 6790) (22445,)
(16500, 6790) (22445,)
```

1.6 Applying Naive Bayes(Multinomial)

1.6.1 Applying Naive Bayes: BOW featurization

1.6.1.1 Hyper parameter tuning

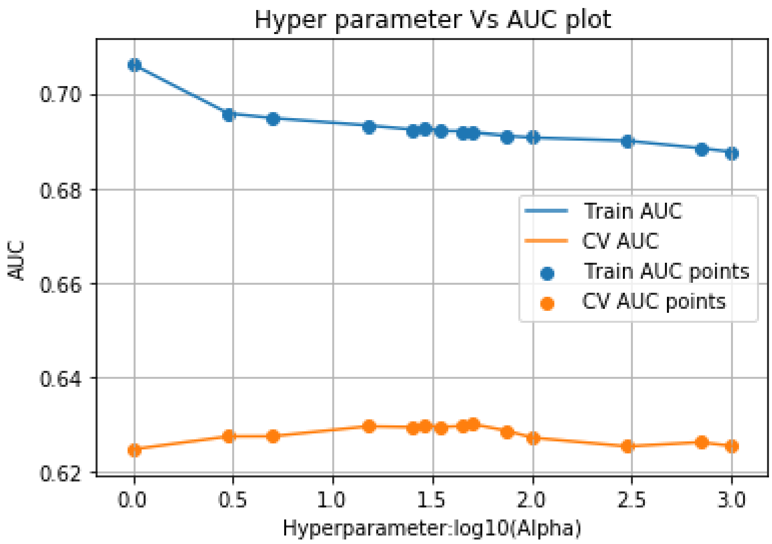
In [41]:

```
import matplotlib.pyplot as plt
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
```

In [43]:

```
import warnings
warnings.filterwarnings("ignore")

# Simple CV using for loops.
train_auc = []
cv_auc = []
alpha = np.log10([1, 3, 5, 15, 25, 29, 35, 45, 51, 75, 101, 301,701,1001])
for i in tqdm(alpha):
    clf=MultinomialNB(alpha=i, fit_prior=True, class_prior=[0.5,0.5])
    clf.fit(X_tr_bow, y_train)
    y_train_pred = clf.predict(X_tr_bow)
    y_cv_pred = clf.predict(X_cv_bow)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he positive class
# not the predicted outputs
    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.plot(alpha, train_auc, label='Train AUC')
plt.plot(alpha, cv_auc, label='CV AUC')

plt.scatter(alpha, train_auc, label='Train AUC points')
plt.scatter(alpha, cv_auc, label='CV AUC points')
plt.legend()
plt.xlabel("Hyperparameter:log10(Alpha)")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
```

```
100%|██████████████████████████████████████████████| 14/14 [00:00<00:00, 16.86it/s]
```



As a result of simple CV and from the Hyper parameter Vs AUC plot , I shall be testing with values of log10(alpha) in the range 1.0-1.75.

1.6.1.2 Testing the performance of the model on test data & plotting ROC Curves for train & test data

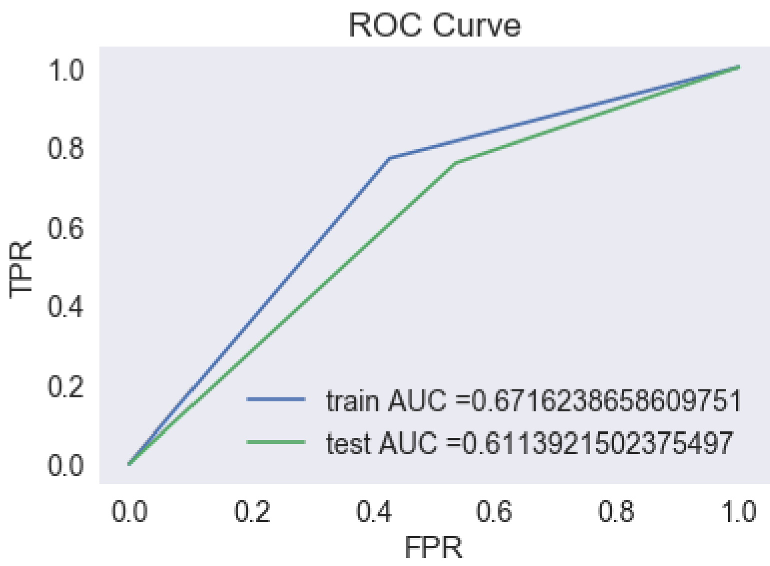
In [47]:

```
a=(10**0.98)
print(a)
```

```
9.549925860214358
```

In [133]:

```
#Case 1: alpha=9

best_alpha = 9
clf1= MultinomialNB(alpha=best_alpha, fit_prior=True, class_prior=[0.5,0.5])
clf1.fit(X_tr_bow, y_train)
y_train_pred_bow = clf1.predict(X_tr_bow)
y_test_pred_bow = clf1.predict(X_test_bow)

train_tpr, train_fpr, tr_thresholds = roc_curve(y_train, y_train_pred_bow)
test_tpr, test_fpr, te_thresholds = roc_curve(y_test, y_test_pred_bow)

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



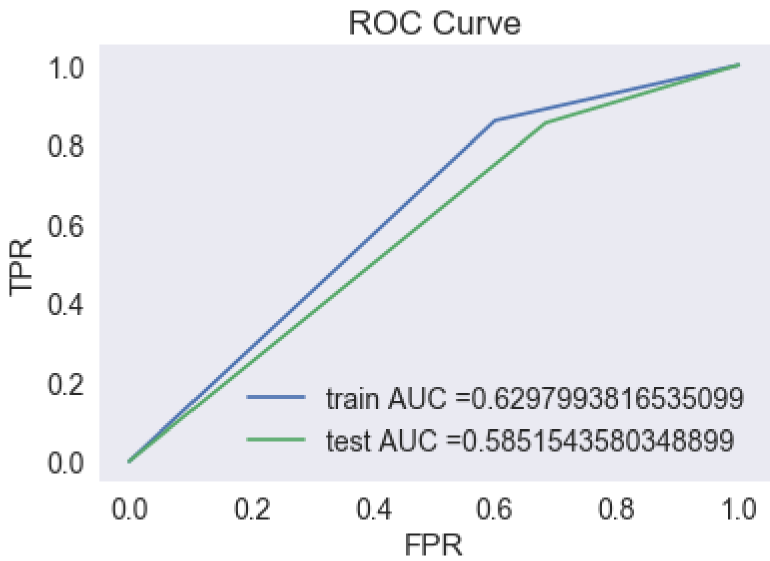
In [49]:

```
a=(10**1.2)
print(a)
```

15.848931924611133

In [126]:

```
#Case 2: alpha=15

best_alpha = 15
clf1= MultinomialNB(alpha=best_alpha, fit_prior=True, class_prior=[0.5,0.5])
clf1.fit(X_tr_bow, y_train)
y_train_pred_bow = clf1.predict(X_tr_bow)
y_test_pred_bow = clf1.predict(X_test_bow)

train_tpr, train_fpr, tr_thresholds = roc_curve(y_train, y_train_pred_bow)
test_tpr, test_fpr, te_thresholds = roc_curve(y_test, y_test_pred_bow)

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



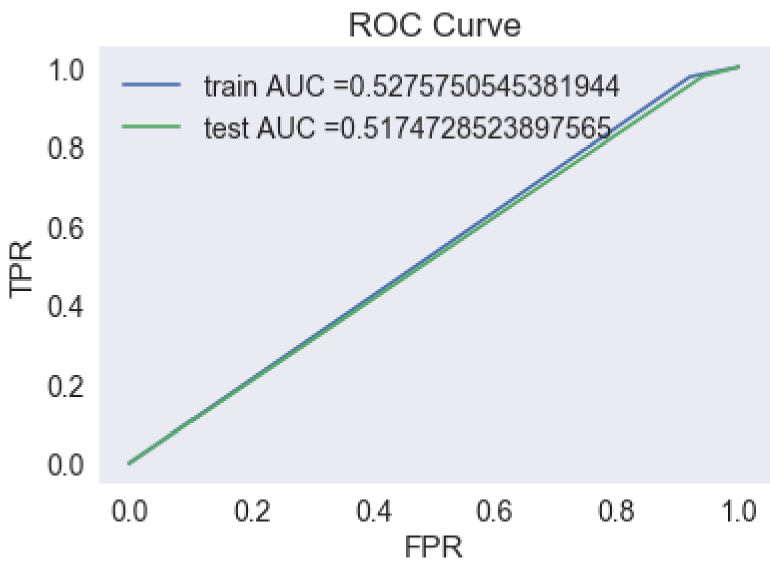
In [51]:

```
a=(10**1.5)
print(a)
```

```
31.622776601683793
```

In [128]:

```
#Case 3: alpha=31

best_alpha = 31
clf4= MultinomialNB(alpha=best_alpha, fit_prior=True, class_prior=[0.5,0.5])
clf4.fit(X_tr_bow, y_train)
y_train_pred_bow = clf4.predict(X_tr_bow)
y_test_pred_bow = clf4.predict(X_test_bow)

train_tpr, train_fpr, tr_thresholds = roc_curve(y_train, y_train_pred_bow)
test_tpr, test_fpr, te_thresholds = roc_curve(y_test, y_test_pred_bow)

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



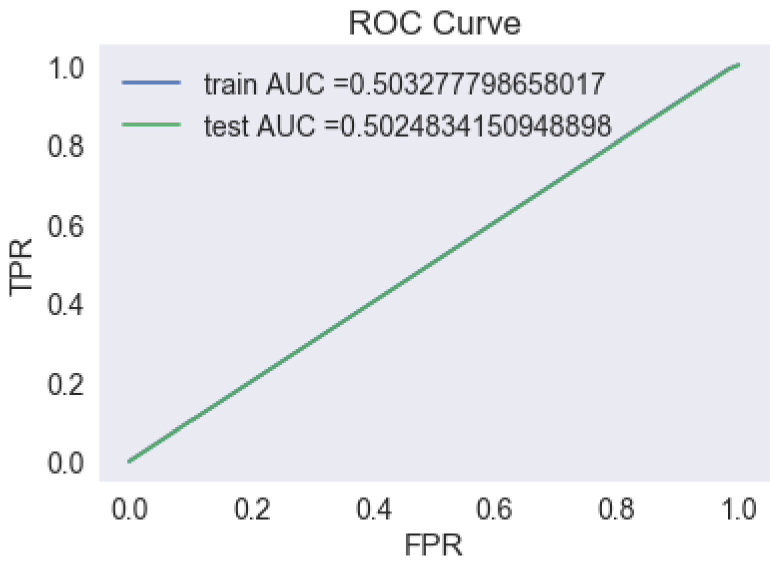
In [53]:

```
a=(10**1.7)
print(a)
```

```
50.11872336272722
```

In [130]:

```
#Case 4: alpha=49

best_alpha = 49
clf1= MultinomialNB(alpha=best_alpha, fit_prior=True, class_prior=[0.5,0.5])
clf1.fit(X_tr_bow, y_train)
y_train_pred_bow = clf1.predict(X_tr_bow)
y_test_pred_bow = clf1.predict(X_test_bow)

train_tpr, train_fpr, tr_thresholds = roc_curve(y_train, y_train_pred_bow)
test_tpr, test_fpr, te_thresholds = roc_curve(y_test, y_test_pred_bow)

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



- After experimenting with all odd values within the above mentioned range i.e. 9-49, maximum AUC of 0.61 was obtained for alpha=9 and minimum AUC of 0.50 for alpha=49.
- Therefore I found the value of alpha=9 as the best or optimal value.

In [55]:

```
## we will pick a threshold that will give the least fpr

def find_best_threshold(threshold, fpr, tpr):
    t = threshold[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("The maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)),"for threshold", np.roun
d(t,3))
    return t

def predict_with_best_t(proba, threshold):
    predictions = []
    for i in proba:
        if i>=threshold:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions

print("="*100)
```

```
===========================================================================
==========================
```

In [61]:

```
#function to get heatmap of confusion matrix
# Reference: https://stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-ma
trix

def cm_heatmap(cm):
    #y_pred = clf.predict(X_te)
    df_cm = pd.DataFrame(cm, range(2),range(2))
    df_cm.columns = ['Predicted NO','Predicted YES']
    df_cm = df_cm.rename({0: 'Actual NO', 1: 'Actual YES'})
    sns.set(font_scale=1.4)#for label size
    sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='d')
```

1.6.1.3 Confusion matrices: For best alpha=9

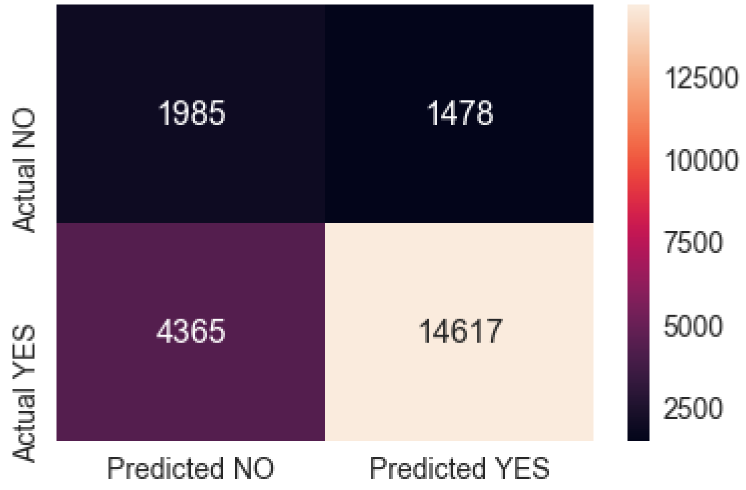
In [62]:

```
from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
print("Train confusion matrix")
cm_train=confusion_matrix(y_train, predict_with_best_t(y_train_pred_bow, best_t))
print(cm_train)
print("Test confusion matrix")
cm_test=confusion_matrix(y_test, predict_with_best_t(y_test_pred_bow, best_t))
print(cm_test)
```

```
The maximum value of tpr*(1-fpr) 0.09814410557741331 for threshold 1
Train confusion matrix
[[ 1985  1478]
 [ 4365 14617]]
Test confusion matrix
[[ 1183  1363]
 [ 3375 10579]]
```

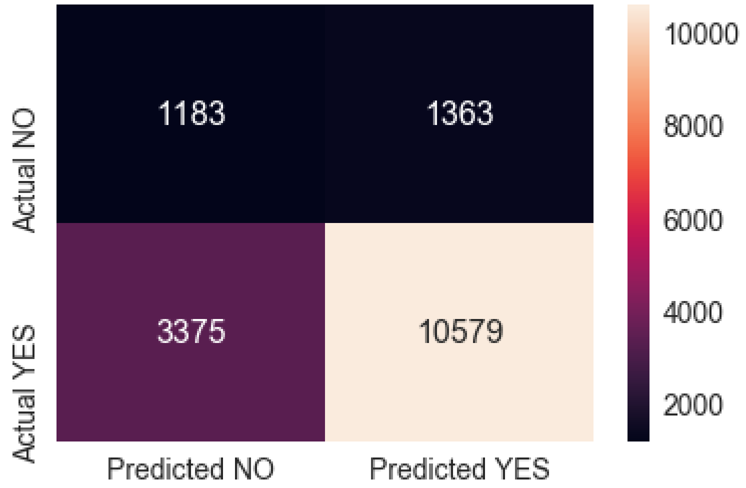
In [63]:

```
# confusion matrix heatmap for train data
cm_heatmap(cm_train)
```



In [64]:

```
# confusion matrix heatmap for test data
cm_heatmap(cm_test)
```



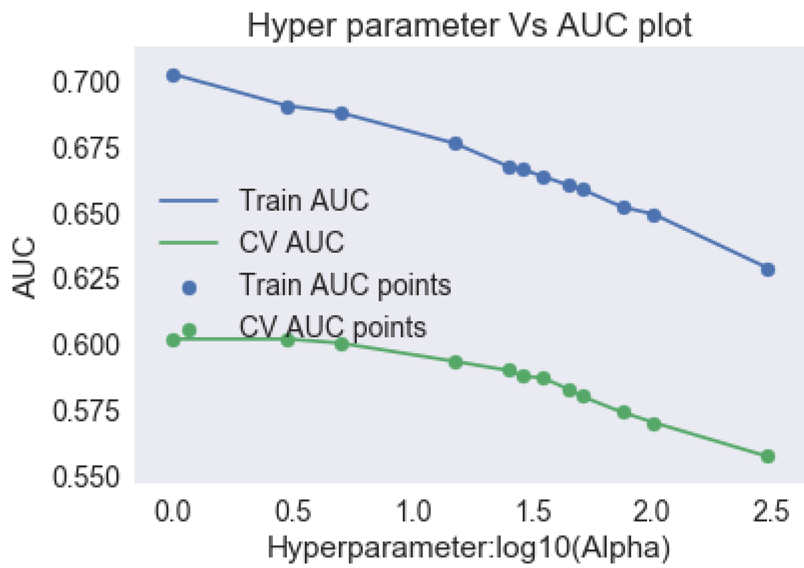
1.6.2 Applying Naive Bayes: TFIDF featurization

1.6.2.1 Hyper parameter tuning

In [65]:

```
# Simple CV using for loops.
train_auc_tfidf = []
cv_auc_tfidf = []
alpha = np.log10([1, 3, 5, 15, 25, 29, 35, 45, 51, 75, 101, 301])
for i in tqdm(alpha):
    clf2=MultinomialNB(alpha=i, fit_prior=True, class_prior=[0.5,0.5])
    clf2.fit(X_tr_tfidf, y_train)
    y_train_pred_tfidf = clf2.predict(X_tr_tfidf)
    y_cv_pred_tfidf = clf2.predict(X_cv_tfidf)

# roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
he positive class
# not the predicted outputs
    train_auc_tfidf.append(roc_auc_score(y_train,y_train_pred_tfidf))
    cv_auc_tfidf.append(roc_auc_score(y_cv, y_cv_pred_tfidf))

plt.plot(alpha, train_auc_tfidf, label='Train AUC')
plt.plot(alpha, cv_auc_tfidf, label='CV AUC')

plt.scatter(alpha, train_auc_tfidf, label='Train AUC points')
plt.scatter(alpha, cv_auc_tfidf, label='CV AUC points')
plt.legend()
plt.xlabel("Hyperparameter:log10(Alpha)")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
```

```
100%|████████████████████████████████████████████████████████████| 12/12 [00:00<00:00, 16.22it/s]
```



As a result of simple CV and from the Hyper parameter Vs AUC plot , I shall be testing with values of log10(alpha) in the range 0.5-1.0.

1.6.2.2 Testing the performance of the model on test data & plotting ROC Curves for train & test data

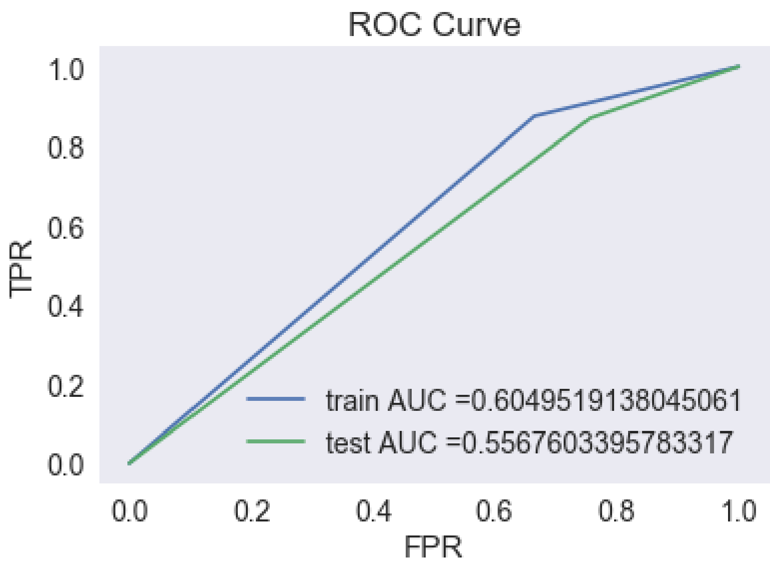
In [66]:

```
b=(10**0.5)
print(b)
```

3.1622776601683795

In [74]:

```
#Case 1: alpha=3

best_alpha = 3
clf= MultinomialNB(alpha=best_alpha, fit_prior=True, class_prior=[0.5,0.5])
clf.fit(X_tr_tfidf, y_train)
y_train_pred_tr = clf.predict(X_tr_tfidf)
y_test_pred_ts = clf.predict(X_test_tfidf)

train_tpr_1, train_fpr_1, tr_thresholds_1 = roc_curve(y_train, y_train_pred_tr)
test_tpr_1, test_fpr_1, te_thresholds_1 = roc_curve(y_test, y_test_pred_ts)

plt.plot(train_tpr_1, train_fpr_1,label="train AUC ="+str(auc(train_tpr_1, train_fpr_1
)))
plt.plot(test_tpr_1, test_fpr_1, label="test AUC ="+str(auc(test_tpr_1, test_fpr_1)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve")
plt.grid()
plt.show()
```



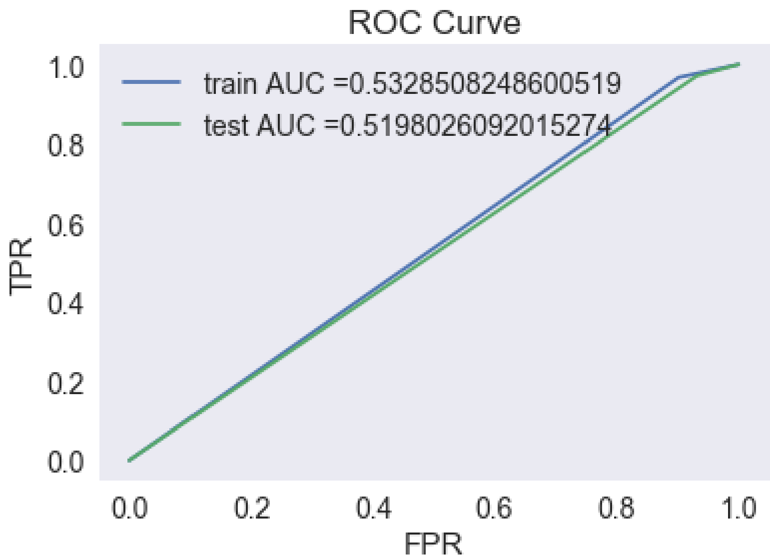
In [68]:

```
b=(10**0.75)
print(b)
```

5.623413251903491

In [69]:

```
#Case 2: alpha=5

best_alpha = 5
clf3= MultinomialNB(alpha=best_alpha, fit_prior=True, class_prior=[0.5,0.5])
clf3.fit(X_tr_tfidf, y_train)
y_train_pred_tr = clf3.predict(X_tr_tfidf)
y_test_pred_ts = clf3.predict(X_test_tfidf)

train_tpr_1, train_fpr_1, tr_thresholds_1 = roc_curve(y_train, y_train_pred_tr)
test_tpr_1, test_fpr_1, te_thresholds_1 = roc_curve(y_test, y_test_pred_ts)

plt.plot(train_tpr_1, train_fpr_1,label="train AUC ="+str(auc(train_tpr_1, train_fpr_1
)))
plt.plot(test_tpr_1, test_fpr_1, label="test AUC ="+str(auc(test_tpr_1, test_fpr_1)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve")
plt.grid()
plt.show()
```



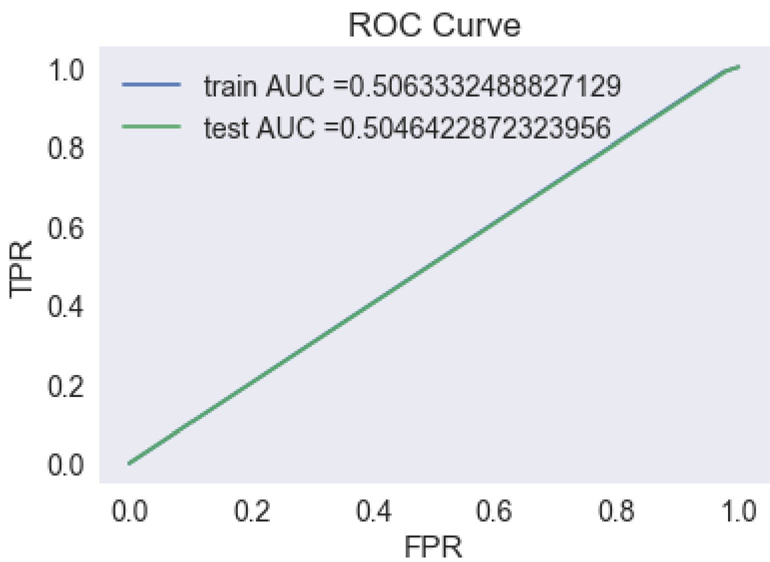
In [70]:

```
b=(10**0.85)
print(b)
```

7.079457843841379

In [71]:

```
#Case 3: alpha=7

best_alpha = 7
clf3= MultinomialNB(alpha=best_alpha, fit_prior=True, class_prior=[0.5,0.5])
clf3.fit(X_tr_tfidf, y_train)
y_train_pred_tr = clf3.predict(X_tr_tfidf)
y_test_pred_ts = clf3.predict(X_test_tfidf)

train_tpr_1, train_fpr_1, tr_thresholds_1 = roc_curve(y_train, y_train_pred_tr)
test_tpr_1, test_fpr_1, te_thresholds_1 = roc_curve(y_test, y_test_pred_ts)

plt.plot(train_tpr_1, train_fpr_1,label="train AUC ="+str(auc(train_tpr_1, train_fpr_1
)))
plt.plot(test_tpr_1, test_fpr_1, label="test AUC ="+str(auc(test_tpr_1, test_fpr_1)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve")
plt.grid()
plt.show()
```



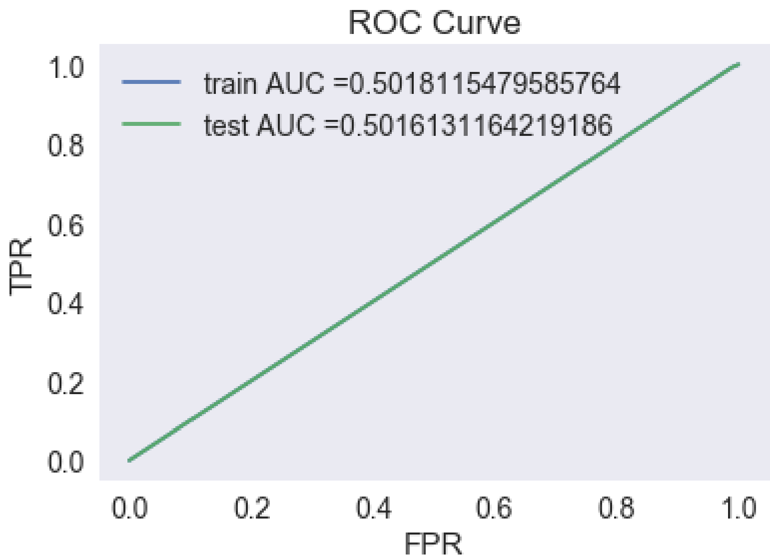
In [72]:

```
b=(10**0.95)
print(b)
```

```
8.912509381337454
```

In [73]:

```
#Case 4: alpha=9

best_alpha = 9
clf3= MultinomialNB(alpha=best_alpha, fit_prior=True, class_prior=[0.5,0.5])
clf3.fit(X_tr_tfidf, y_train)
y_train_pred_tr = clf3.predict(X_tr_tfidf)
y_test_pred_ts = clf3.predict(X_test_tfidf)

train_tpr_1, train_fpr_1, tr_thresholds_1 = roc_curve(y_train, y_train_pred_tr)
test_tpr_1, test_fpr_1, te_thresholds_1 = roc_curve(y_test, y_test_pred_ts)

plt.plot(train_tpr_1, train_fpr_1,label="train AUC ="+str(auc(train_tpr_1, train_fpr_1
)))
plt.plot(test_tpr_1, test_fpr_1, label="test AUC ="+str(auc(test_tpr_1, test_fpr_1)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve")
plt.grid()
plt.show()
```



- After experimenting with all odd values within the above mentioned range i.e. 3-9, maximum AUC of 0.56 was obtained for alpha=3 and minimum AUC of 0.50 for alpha=9.
- Therefore I found the value of alpha=3 as the best or optimal value.

1.6.2.3 Confusion matrices

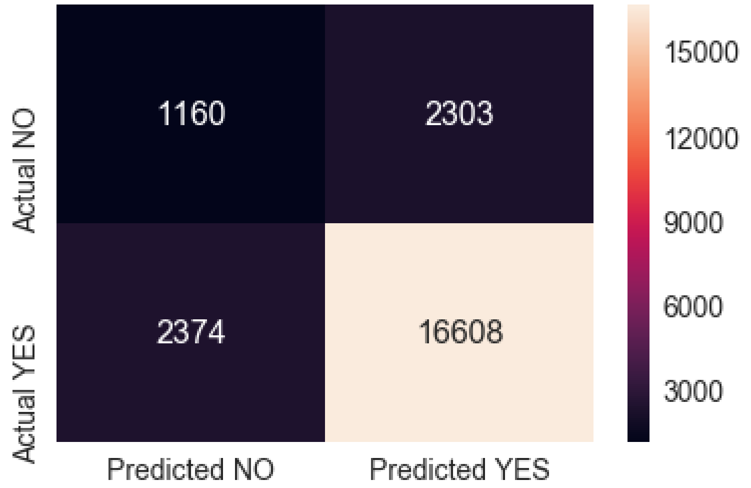
In [75]:

```
from sklearn.metrics import confusion_matrix
best_t_1 = find_best_threshold(tr_thresholds_1, train_fpr_1, train_tpr_1)
print("Train confusion matrix")
cm_train_1=confusion_matrix(y_train, predict_with_best_t(y_train_pred_tr, best_t_1))
print(cm_train_1)
print("Test confusion matrix")
cm_test_1=confusion_matrix(y_test, predict_with_best_t(y_test_pred_ts, best_t_1))
print(cm_test_1)
```

```
The maximum value of tpr*(1-fpr) 0.08317258354975134 for threshold 1
Train confusion matrix
[[ 1160  2303]
 [ 2374 16608]]
Test confusion matrix
[[  620  1926]
 [ 1814 12140]]
```

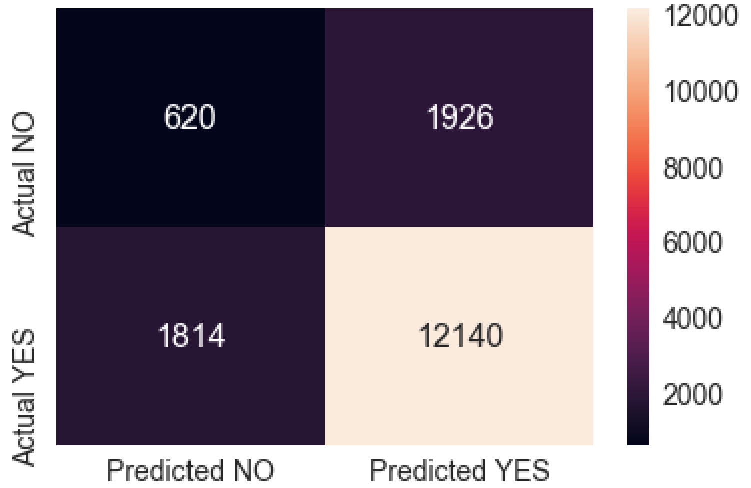
In [76]:

```
# confusion matrix heatmap for train data
cm_heatmap(cm_train_1)
```



In [77]:

```
# confusion matrix heatmap for test data
cm_heatmap(cm_test_1)
```



1.7 Top 20 features from Set-1 using feature_log_prob

In [152]:

```
# Storing feature names of project title, essay & all categorical features in a list ca
lled "features=[]"

lst=[vectorizer,vectorizer1,vectorizer2,vectorizer3,vectorizer4,vectorizer5,vectorizer6
] #vectorizers used in the features mentioned above
features=[]
for i in tqdm(lst):
    features.extend(i.get_feature_names())
```

```
100%|██████████████████████████████████████████████████████████████████████
██████████████████████████████████████████████████████████████| 8/8 [00:00<
00:00, 39.22it/s]
```

In [157]:

```
pos_class_prob_sorted =np.argsort((clf1.feature_log_prob_)[1])[::-1]
pos_class_features = np.take(features, pos_class_prob_sorted[-10:])
print("Top 10 informative features for positive class:")
print(pd.DataFrame(data=pos_class_features ))
```

```
Top 10 informative features for positive class:
                0
0         animals
1  learning games
2        outdoors
3       readiness
4      love learn
5            path
6  flexible minds
7             nan
8            none
9              dr
```

In [156]:

```
neg_class_prob_sorted =np.argsort((clf1.feature_log_prob_)[0])[::-1] #calculating proba
bility values for each feature & then sorting them
neg_class_features = np.take(features, neg_class_prob_sorted[-10:]) #top 10 features
print("Top 10 informative features for negative class:")
print(pd.DataFrame(data=neg_class_features ))
```

```
Top 10 informative features for negative class:
               0
0        strings
1  learning part
2             10
3             16
4           leap
5     volleyball
6       memories
7        protect
8            7th
9          spark
```

2.0 Summary

In [146]:

```
from prettytable import PrettyTable

x = PrettyTable(["Hyperparameter", "Train AUC", "Test AUC"])

x.add_row([9,0.67,0.61])
x.add_row([15,0.63,0.59])
x.add_row([31,0.52,0.52])
x.add_row([49,0.5,0.5])

print("Summary of scores using BOW Vectorization")
print(x.get_string(title="BOW Vectorization"))
```

```
Summary of scores using BOW Vectorization
+----------------+-----------+----------+
| Hyperparameter | Train AUC | Test AUC |
+----------------+-----------+----------+
|       9        |    0.67   |   0.61   |
|       15       |    0.63   |   0.59   |
|       31       |    0.52   |   0.52   |
|       49       |    0.5    |   0.5    |
+----------------+-----------+----------+
```

In [147]:

```
x = PrettyTable(["Hyperparameter", "Train AUC", "Test AUC"])

x.add_row([3,0.60,0.56])
x.add_row([5,0.53,0.52])
x.add_row([7,0.51,0.50])
x.add_row([9,0.5,0.5])

print("Summary of scores using TFIDF Vectorization")
print(x.get_string(title="TFIDF Vectorization"))
```

```
Summary of scores using TFIDF Vectorization
+----------------+-----------+----------+
| Hyperparameter | Train AUC | Test AUC |
+----------------+-----------+----------+
|       3        |    0.6    |   0.56   |
|       5        |    0.53   |   0.52   |
|       7        |    0.51   |   0.5    |
|       9        |    0.5    |   0.5    |
+----------------+-----------+----------+
```