6_Assignment_NB_Instructions

February 26, 2021

1 Assignment 6: Apply NB

Minimum data points need to be considered for people having 4GB RAM is 50k and for 8GB RAM is 100k

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.

If you are writing for loops to tune your model then you need split the data into X_train,X_cv,X_test.

While splitting the data explore stratify parameter.

Apply Multinomial NB on these feature sets

```
<ul>
        <li>Features that need to be considered</li>
            <dl>
              <dt>essay</dt>
                <dd>while encoding essay, try to experiment with the max_features and n_grams p
              <dt>categorical features</dt>
              <dd> - teacher_prefix</dd>
              <dd> - project_grade_category</dd>
              <dd> - school_state</dd>
              <dd> - clean_categories</dd>
              <dd> - clean_subcategories</dd>
              <dt>numerical features</dt>
              <dd> - price</dd>
              <dd> - teacher_number_of_previously_posted_projects</dd>
              <dd>while encoding the numerical features check <a href='https://imgur.com/ldZA1z
            </dl>
        <li><font color='red'>Set 1</font>: categorical, numerical features + preprocessed_eass
        <li><font color='red'>Set 2</font>: categorical, numerical features + preprocessed_eass
    </ul>
<li><strong>The hyper paramter tuning(find best alpha:smoothing parameter)</strong>
    <ul>
<li>Consider alpha values in range: 10^-5 to 10^2 like [0.00001,0.0005, 0.0001,0.005,0.001,0.05
<li>Explore class_prior = [0.5, 0.5] parameter which can be present in MultinomialNB function(g
<li>Find the best hyper parameter which will give the maximum <a href='https://www.appliedaicou
```

For hyper parameter tuning using k-fold cross validation(use GridsearchCV or RandomsearchC
You need to plot the performance of model both on train data and cross validation data for
<img src='https://i.imgur.com/hUv6aEy.jpg' width=300px><dd>-while plotting take log(alpha) on
Once after you found the best hyper parameter, you need to train your model with it, and f
<img src='https://imgur.com/q2P65L5.jpg' width=300px>
Along with plotting ROC curve, you need to print the <a href='https://www.appliedaicourse.
    <img src='https://i.imgur.com/IdN5Ctv.png' width=300px><dd>-plot the confusion matrix in h

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

2. Naive Bayes

1.0.1 Importing necessary packages

[1]:
```
import numpy as np
import pandas as pd
import re
from nltk.corpus import stopwords
import pickle
from tqdm import tqdm
import os
import nltk
from tqdm import tqdm

from sklearn.model_selection import train_test_split
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 collections import Counter
from sklearn.preprocessing import Normalizer
from scipy.sparse import hstack
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import GridSearchCV
import math

%matplotlib inline
import matplotlib.pyplot as plt
```

```
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
from chart_studio import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from prettytable import PrettyTable
```

1.1 1.1 Loading Data

[2]:
```
data = pd.read_csv('preprocessed_data.csv')
```

[3]:
```
y = data['project_is_approved'].values
X = data.drop(['project_is_approved'], axis=1)
X.head(1)
```

[3]:
```
  school_state teacher_prefix project_grade_category  \
0           ca            mrs          grades_prek_2

   teacher_number_of_previously_posted_projects clean_categories  \
0                                            53     math_science

                  clean_subcategories  \
0  appliedsciences health_lifescience

                                               essay   price
0  i fortunate enough use fairy tale stem kits cl…  725.05
```

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

[4]:
```
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)
```

1.3 Encoding Data

1.2 Encoding essays using Bag of Words

[5]:
```
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)

vectorizer_bow = CountVectorizer(min_df=10,ngram_range=(1,4), max_features=5000)
vectorizer_bow.fit(X_train['essay'].values)
```

```
X_train_essay_bow = vectorizer_bow.transform(X_train['essay'].values)
X_cv_essay_bow = vectorizer_bow.transform(X_cv['essay'].values)
X_test_essay_bow = vectorizer_bow.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)
print("="*100)
```

```
(49041, 8) (49041,)
(24155, 8) (24155,)
(36052, 8) (36052,)
================================================================================
====================
After vectorizations
(49041, 5000) (49041,)
(24155, 5000) (24155,)
(36052, 5000) (36052,)
================================================================================
====================
```

1.3 Encoding essays using TFIDF

[6]:
```
print(X_train.shape, y_train.shape)
print(X_cv.shape, y_cv.shape)
print(X_test.shape, y_test.shape)

print("="*100)

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

X_train_essay_tfidf = vectorizer_tfidf.transform(X_train['essay'].values)
X_cv_essay_tfidf = vectorizer_tfidf.transform(X_cv['essay'].values)
X_test_essay_tfidf = vectorizer_tfidf.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)
print("="*100)
```

```
(49041, 8) (49041,)
(24155, 8) (24155,)
(36052, 8) (36052,)
```

```
================================================================================
====================
After vectorizations
(49041, 5000) (49041,)
(24155, 5000) (24155,)
(36052, 5000) (36052,)
================================================================================
====================
```

1.4 Encoding numerical and categorical features

1.4 Encoding the State of the Project

[7]:
```
vectorizer_state = CountVectorizer()
vectorizer_state.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_ohe = vectorizer_state.transform(X_train['school_state'].values)
X_cv_state_ohe = vectorizer_state.transform(X_cv['school_state'].values)
X_test_state_ohe = vectorizer_state.transform(X_test['school_state'].values)

print("After vectorizations")
print(X_train_state_ohe.shape, y_train.shape)
print(X_cv_state_ohe.shape, y_cv.shape)
print(X_test_state_ohe.shape, y_test.shape)
print(vectorizer_state.get_feature_names())
print("="*100)
```

```
After vectorizations
(49041, 51) (49041,)
(24155, 51) (24155,)
(36052, 51) (36052,)
['ak', 'al', 'ar', 'az', 'ca', 'co', 'ct', 'dc', 'de', 'fl', 'ga', 'hi', 'ia',
'id', 'il', 'in', 'ks', 'ky', 'la', 'ma', 'md', 'me', 'mi', 'mn', 'mo', 'ms',
'mt', 'nc', 'nd', 'ne', 'nh', 'nj', 'nm', 'nv', 'ny', 'oh', 'ok', 'or', 'pa',
'ri', 'sc', 'sd', 'tn', 'tx', 'ut', 'va', 'vt', 'wa', 'wi', 'wv', 'wy']
================================================================================
====================
```

1.5 Encoding the Grade of the Project

[8]:
```
vectorizer_grade = CountVectorizer()
vectorizer_grade.fit(X_train['project_grade_category'].values) # fit has to␣
 ↪happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
```

```
X_train_project_category_ohe = vectorizer_grade.
 ↪transform(X_train['project_grade_category'].values)
X_cv_project_category_ohe = vectorizer_grade.
 ↪transform(X_cv['project_grade_category'].values)
X_test_project_category_ohe = vectorizer_grade.
 ↪transform(X_test['project_grade_category'].values)

print("After vectorizations")
print(X_train_project_category_ohe.shape, y_train.shape)
print(X_cv_project_category_ohe.shape, y_cv.shape)
print(X_test_project_category_ohe.shape, y_test.shape)
print(vectorizer_grade.get_feature_names())
print("="*100)
```

```
After vectorizations
(49041, 4) (49041,)
(24155, 4) (24155,)
(36052, 4) (36052,)
['grades_3_5', 'grades_6_8', 'grades_9_12', 'grades_prek_2']
================================================================================
====================
```

1.6 Encoding the Prefix/Surname of the Teacher

[9]:
```
vectorizer_teacher_prefix = CountVectorizer()
vectorizer_teacher_prefix.fit(X_train['teacher_prefix'].values) # fit has to␣
 ↪happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_teacher_prefix_ohe = vectorizer_teacher_prefix.
 ↪transform(X_train['teacher_prefix'].values)
X_cv_teacher_prefix_ohe = vectorizer_teacher_prefix.
 ↪transform(X_cv['teacher_prefix'].values)
X_test_teacher_prefix_ohe = vectorizer_teacher_prefix.
 ↪transform(X_test['teacher_prefix'].values)

print("After vectorizations")
print(X_train_teacher_prefix_ohe.shape, y_train.shape)
print(X_cv_teacher_prefix_ohe.shape, y_cv.shape)
print(X_test_teacher_prefix_ohe.shape, y_test.shape)
print(vectorizer_teacher_prefix.get_feature_names())
print("="*100)
```

```
After vectorizations
(49041, 5) (49041,)
(24155, 5) (24155,)
(36052, 5) (36052,)
```

```
['dr', 'mr', 'mrs', 'ms', 'teacher']
================================================================================
====================
```

1.7 Encoding the Categories of the Project

[10]:
```
vectorizer_cat = CountVectorizer()
vectorizer_cat.fit(X_train['clean_categories'].values) # fit has to happen only␣
↪on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_categories_ohe = vectorizer_cat.
↪transform(X_train['clean_categories'].values)
X_cv_clean_categories_ohe = vectorizer_cat.transform(X_cv['clean_categories'].
↪values)
X_test_clean_categories_ohe = vectorizer_cat.
↪transform(X_test['clean_categories'].values)

print("After vectorizations")
print(X_train_clean_categories_ohe.shape, y_train.shape)
print(X_cv_clean_categories_ohe.shape, y_cv.shape)
print(X_test_clean_categories_ohe.shape, y_test.shape)
print(vectorizer_cat.get_feature_names())
print("="*100)
```

```
After vectorizations
(49041, 9) (49041,)
(24155, 9) (24155,)
(36052, 9) (36052,)
['appliedlearning', 'care_hunger', 'health_sports', 'history_civics',
'literacy_language', 'math_science', 'music_arts', 'specialneeds', 'warmth']
================================================================================
====================
```

1.8 Encoding the Subcategories of the Project

[11]:
```
vectorizer_subcat = CountVectorizer()
vectorizer_subcat.fit(X_train['clean_subcategories'].values) # fit has to␣
↪happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_clean_subcategories_ohe = vectorizer_subcat.
↪transform(X_train['clean_subcategories'].values)
X_cv_clean_subcategories_ohe = vectorizer_subcat.
↪transform(X_cv['clean_subcategories'].values)
X_test_clean_subcategories_ohe = vectorizer_subcat.
↪transform(X_test['clean_subcategories'].values)
```

```
print("After vectorizations")
print(X_train_clean_subcategories_ohe.shape, y_train.shape)
print(X_cv_clean_subcategories_ohe.shape, y_cv.shape)
print(X_test_clean_subcategories_ohe.shape, y_test.shape)
print(vectorizer_subcat.get_feature_names())
print("="*100)
```

```
After vectorizations
(49041, 30) (49041,)
(24155, 30) (24155,)
(36052, 30) (36052,)
['appliedsciences', 'care_hunger', 'charactereducation', 'civics_government',
'college_careerprep', 'communityservice', 'earlydevelopment', 'economics',
'environmentalscience', 'esl', 'extracurricular', 'financialliteracy',
'foreignlanguages', 'gym_fitness', 'health_lifescience', 'health_wellness',
'history_geography', 'literacy', 'literature_writing', 'mathematics', 'music',
'nutritioneducation', 'other', 'parentinvolvement', 'performingarts',
'socialsciences', 'specialneeds', 'teamsports', 'visualarts', 'warmth']
================================================================================
====================
```

1.9 Encoding the Price required for the project

[12]:
```
normalizer = Normalizer()
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.shape, y_train.shape)
print(X_cv_price_norm.shape, y_cv.shape)
print(X_test_price_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
================================================================================
====================
```

1.10 Encoding the Number of Previous Projects trained by the Teacher

[13]:
```
normalizer = Normalizer()
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.
↪reshape(1,-1))

X_train_prev_projects_norm = normalizer.
↪transform(X_train['teacher_number_of_previously_posted_projects'].values.
↪reshape(-1,1))
X_cv_prev_projects_norm = normalizer.
↪transform(X_cv['teacher_number_of_previously_posted_projects'].values.
↪reshape(-1,1))
X_test_prev_projects_norm = normalizer.
↪transform(X_test['teacher_number_of_previously_posted_projects'].values.
↪reshape(-1,1))


print("After vectorizations")
print(X_train_prev_projects_norm.shape, y_train.shape)
print(X_cv_prev_projects_norm.shape, y_cv.shape)
print(X_test_prev_projects_norm.shape, y_test.shape)
print("="*100)
```

```
After vectorizations
(49041, 1) (49041,)
(24155, 1) (24155,)
(36052, 1) (36052,)
================================================================================
====================
```

1.11 Stacking all features into two different datasets.

1.12 1. Bag of Words

1.13 2. TFIDF

[14]:
```
X_tr_bow = hstack((X_train_state_ohe, X_train_project_category_ohe,␣
↪X_train_teacher_prefix_ohe, X_train_clean_categories_ohe,␣
↪X_train_clean_subcategories_ohe, X_train_price_norm,␣
↪X_train_prev_projects_norm, X_train_essay_bow)).tocsr()
X_cr_bow = hstack((X_cv_state_ohe, X_cv_project_category_ohe,␣
↪X_cv_teacher_prefix_ohe, X_cv_clean_categories_ohe,␣
↪X_cv_clean_subcategories_ohe, X_cv_price_norm, X_cv_prev_projects_norm,␣
↪X_cv_essay_bow)).tocsr()
X_te_bow = hstack((X_test_state_ohe, X_test_project_category_ohe,␣
↪X_test_teacher_prefix_ohe, X_test_clean_categories_ohe,␣
↪X_test_clean_subcategories_ohe, X_test_price_norm,␣
↪X_test_prev_projects_norm, X_test_essay_bow)).tocsr()
```

```
X_tr_tfidf = hstack((X_train_state_ohe, X_train_project_category_ohe,␣
 ↪X_train_teacher_prefix_ohe, X_train_clean_categories_ohe,␣
 ↪X_train_clean_subcategories_ohe, X_train_price_norm,␣
 ↪X_train_prev_projects_norm, X_train_essay_tfidf)).tocsr()
X_cr_tfidf = hstack((X_cv_state_ohe, X_cv_project_category_ohe,␣
 ↪X_cv_teacher_prefix_ohe, X_cv_clean_categories_ohe,␣
 ↪X_cv_clean_subcategories_ohe, X_cv_price_norm, X_cv_prev_projects_norm,␣
 ↪X_cv_essay_tfidf)).tocsr()
X_te_tfidf = hstack((X_test_state_ohe, X_test_project_category_ohe,␣
 ↪X_test_teacher_prefix_ohe, X_test_clean_categories_ohe,␣
 ↪X_test_clean_subcategories_ohe, X_test_price_norm,␣
 ↪X_test_prev_projects_norm, X_test_essay_tfidf)).tocsr()

print("Final Data matrix: BoW")
print(X_tr_bow.shape, y_train.shape)
print(X_cr_bow.shape, y_cv.shape)
print(X_te_bow.shape, y_test.shape)

print("="*100)

print("Final Data matrix: TFIDF")
print(X_tr_tfidf.shape, y_train.shape)
print(X_cr_tfidf.shape, y_cv.shape)
print(X_te_tfidf.shape, y_test.shape)
```

```
Final Data matrix: BoW
(49041, 5101) (49041,)
(24155, 5101) (24155,)
(36052, 5101) (36052,)
================================================================================
====================
Final Data matrix: TFIDF
(49041, 5101) (49041,)
(24155, 5101) (24155,)
(36052, 5101) (36052,)
```

1.5 Appling NB on both these datasets

1.14 Bag of Words

[15]:
```
def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability␣
 ↪estimates of the positive class
    # not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
```

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

    return y_data_pred
```

1.14.1 Using various alpha values to calculate the AUC Scores to find the best Alpha value

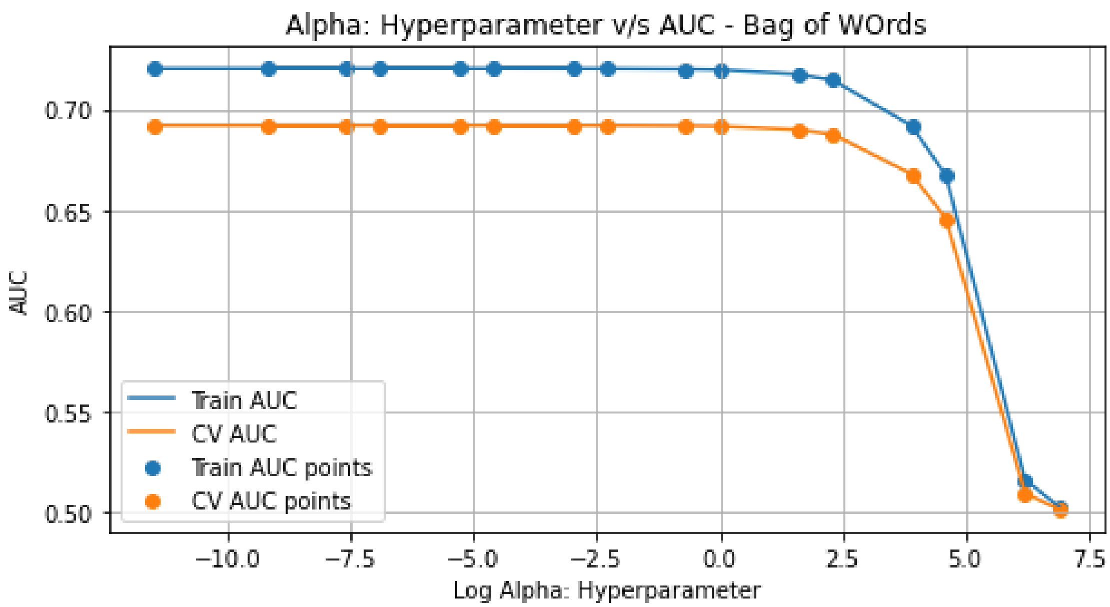
[16]:
```
train_aucs_bow = []
cv_aucs_bow = []
log_alphas_bow = []
alphas = [0.00001,0.0005, 0.0001,0.005,0.001,0.05,0.01,0.1,0.
↪5,1,5,10,50,100,500,1000]

for alpha in tqdm(alphas):
    nb = MultinomialNB(alpha = alpha,class_prior=[0.5,0.5])
    nb.fit(X_tr_bow, y_train)
    y_train_pred_bow = batch_predict(nb, X_tr_bow)
    y_cv_pred_bow = batch_predict(nb, X_cr_bow)

    train_aucs_bow.append(roc_auc_score(y_train,y_train_pred_bow))
    cv_aucs_bow.append(roc_auc_score(y_cv, y_cv_pred_bow))

for a in tqdm(alphas):
    b = math.log(a)
    log_alphas_bow.append(b)
```

```
100%|      | 16/16 [00:02<00:00,  6.59it/s]
100%|      | 16/16 [00:00<00:00, 387912.51it/s]
```

1.14.2 Plotting the log of Alphas vs AUC Scores with that Alpha

[17]:
```
plt.figure(figsize=(8,4))
plt.plot(log_alphas_bow, train_aucs_bow, label='Train AUC')
plt.plot(log_alphas_bow, cv_aucs_bow, label='CV AUC')

plt.scatter(log_alphas_bow, train_aucs_bow, label='Train AUC points')
plt.scatter(log_alphas_bow, cv_aucs_bow, label='CV AUC points')

plt.legend()
plt.xlabel("Log Alpha: Hyperparameter")
```

```
plt.ylabel("AUC")
plt.title("Alpha: Hyperparameter v/s AUC - Bag of WOrds")
plt.grid()
plt.show()
```



1.14.3 The curves fall steeply and join after Alpha = 50 (Fourth dot from the right = 50)

1.14.4 Using GridSearchCV to get the best parameter possible

[18]:
```
nb_bow = MultinomialNB(class_prior=[0.5,0.5])

parameters = {'alpha':[0.00001,0.0005, 0.0001,0.005,0.001,0.05,0.01,0.1,0.
↪5,1,5,10,50,100,500,1000]}

clf_bow = GridSearchCV(nb_bow, parameters, cv= 10,␣
↪scoring='roc_auc',return_train_score=True,verbose=2)

clf_bow.fit(X_tr_bow, y_train)

train_auc_bow = clf_bow.cv_results_['mean_train_score']
train_auc_std_bow = clf_bow.cv_results_['std_train_score']
cv_auc_bow = clf_bow.cv_results_['mean_test_score']
cv_auc_std_bow = clf_bow.cv_results_['std_test_score']
```

```
Fitting 10 folds for each of 16 candidates, totalling 160 fits
[CV] alpha=1e-05 …
```

```
[CV] … alpha=1e-05, total=   0.1s
[CV] alpha=1e-05 …
[CV] … alpha=1e-05, total=   0.0s
[CV] alpha=1e-05 …
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
```

```
[CV] … alpha=1e-05, total=   0.0s
[CV] alpha=1e-05 …
[CV] … alpha=1e-05, total=   0.0s
[CV] alpha=1e-05 …
[CV] … alpha=1e-05, total=   0.0s
[CV] alpha=1e-05 …
[CV] … alpha=1e-05, total=   0.0s
[CV] alpha=1e-05 …
[CV] … alpha=1e-05, total=   0.0s
[CV] alpha=1e-05 …
[CV] … alpha=1e-05, total=   0.0s
[CV] alpha=1e-05 …
[CV] … alpha=1e-05, total=   0.0s
[CV] alpha=1e-05 …
[CV] … alpha=1e-05, total=   0.0s
[CV] alpha=0.0005 …
[CV] … alpha=0.0005, total=   0.0s
[CV] alpha=0.0005 …
[CV] … alpha=0.0005, total=   0.0s
[CV] alpha=0.0005 …
[CV] … alpha=0.0005, total=   0.0s
[CV] alpha=0.0005 …
[CV] … alpha=0.0005, total=   0.0s
[CV] alpha=0.0005 …
[CV] … alpha=0.0005, total=   0.0s
[CV] alpha=0.0005 …
[CV] … alpha=0.0005, total=   0.0s
[CV] alpha=0.0005 …
[CV] … alpha=0.0005, total=   0.0s
[CV] alpha=0.0005 …
[CV] … alpha=0.0005, total=   0.1s
[CV] alpha=0.0005 …
[CV] … alpha=0.0005, total=   0.0s
[CV] alpha=0.0005 …
[CV] … alpha=0.0005, total=   0.0s
[CV] alpha=0.0001 …
[CV] … alpha=0.0001, total=   0.0s
[CV] alpha=0.0001 …
[CV] … alpha=0.0001, total=   0.0s
[CV] alpha=0.0001 …
[CV] … alpha=0.0001, total=   0.0s
```

```
[CV] alpha=0.0001 …
[CV] … alpha=0.0001, total=   0.0s
[CV] alpha=0.0001 …
[CV] … alpha=0.0001, total=   0.0s
[CV] alpha=0.0001 …
[CV] … alpha=0.0001, total=   0.0s
[CV] alpha=0.0001 …
[CV] … alpha=0.0001, total=   0.0s
[CV] alpha=0.0001 …
[CV] … alpha=0.0001, total=   0.0s
[CV] alpha=0.0001 …
[CV] … alpha=0.0001, total=   0.0s
[CV] alpha=0.0001 …
[CV] … alpha=0.0001, total=   0.0s
[CV] alpha=0.005 …
[CV] … alpha=0.005, total=   0.0s
[CV] alpha=0.005 …
[CV] … alpha=0.005, total=   0.0s
[CV] alpha=0.005 …
[CV] … alpha=0.005, total=   0.1s
[CV] alpha=0.005 …
[CV] … alpha=0.005, total=   0.0s
[CV] alpha=0.005 …
[CV] … alpha=0.005, total=   0.0s
[CV] alpha=0.005 …
[CV] … alpha=0.005, total=   0.0s
[CV] alpha=0.005 …
[CV] … alpha=0.005, total=   0.0s
[CV] alpha=0.005 …
[CV] … alpha=0.005, total=   0.0s
[CV] alpha=0.005 …
[CV] … alpha=0.005, total=   0.0s
[CV] alpha=0.005 …
[CV] … alpha=0.005, total=   0.0s
[CV] alpha=0.001 …
[CV] … alpha=0.001, total=   0.0s
[CV] alpha=0.001 …
[CV] … alpha=0.001, total=   0.0s
[CV] alpha=0.001 …
[CV] … alpha=0.001, total=   0.0s
[CV] alpha=0.001 …
[CV] … alpha=0.001, total=   0.0s
[CV] alpha=0.001 …
[CV] … alpha=0.001, total=   0.0s
[CV] alpha=0.001 …
[CV] … alpha=0.001, total=   0.0s
[CV] alpha=0.001 …
[CV] … alpha=0.001, total=   0.0s
```

```
[CV] alpha=0.001 …
[CV] … alpha=0.001, total=   0.0s
[CV] alpha=0.001 …
[CV] … alpha=0.001, total=   0.0s
[CV] alpha=0.001 …
[CV] … alpha=0.001, total=   0.0s
[CV] alpha=0.05 …
[CV] … alpha=0.05, total=   0.0s
[CV] alpha=0.05 …
[CV] … alpha=0.05, total=   0.0s
[CV] alpha=0.05 …
[CV] … alpha=0.05, total=   0.0s
[CV] alpha=0.05 …
[CV] … alpha=0.05, total=   0.0s
[CV] alpha=0.05 …
[CV] … alpha=0.05, total=   0.0s
[CV] alpha=0.05 …
[CV] … alpha=0.05, total=   0.0s
[CV] alpha=0.05 …
[CV] … alpha=0.05, total=   0.0s
[CV] alpha=0.05 …
[CV] … alpha=0.05, total=   0.0s
[CV] alpha=0.05 …
[CV] … alpha=0.05, total=   0.0s
[CV] alpha=0.05 …
[CV] … alpha=0.05, total=   0.0s
[CV] alpha=0.01 …
[CV] … alpha=0.01, total=   0.1s
[CV] alpha=0.01 …
[CV] … alpha=0.01, total=   0.0s
[CV] alpha=0.01 …
[CV] … alpha=0.01, total=   0.0s
[CV] alpha=0.01 …
[CV] … alpha=0.01, total=   0.0s
[CV] alpha=0.01 …
[CV] … alpha=0.01, total=   0.1s
[CV] alpha=0.01 …
[CV] … alpha=0.01, total=   0.0s
[CV] alpha=0.01 …
[CV] … alpha=0.01, total=   0.0s
[CV] alpha=0.01 …
[CV] … alpha=0.01, total=   0.0s
[CV] alpha=0.01 …
[CV] … alpha=0.01, total=   0.0s
[CV] alpha=0.01 …
[CV] … alpha=0.01, total=   0.0s
[CV] alpha=0.1 …
[CV] … alpha=0.1, total=   0.0s
```

```
[CV] alpha=0.1 …
[CV] … alpha=0.1, total=   0.0s
[CV] alpha=0.1 …
[CV] … alpha=0.1, total=   0.0s
[CV] alpha=0.1 …
[CV] … alpha=0.1, total=   0.0s
[CV] alpha=0.1 …
[CV] … alpha=0.1, total=   0.0s
[CV] alpha=0.1 …
[CV] … alpha=0.1, total=   0.0s
[CV] alpha=0.1 …
[CV] … alpha=0.1, total=   0.0s
[CV] alpha=0.1 …
[CV] … alpha=0.1, total=   0.0s
[CV] alpha=0.1 …
[CV] … alpha=0.1, total=   0.0s
[CV] alpha=0.1 …
[CV] … alpha=0.1, total=   0.0s
[CV] alpha=0.5 …
[CV] … alpha=0.5, total=   0.1s
[CV] alpha=0.5 …
[CV] … alpha=0.5, total=   0.0s
[CV] alpha=0.5 …
[CV] … alpha=0.5, total=   0.0s
[CV] alpha=0.5 …
[CV] … alpha=0.5, total=   0.0s
[CV] alpha=0.5 …
[CV] … alpha=0.5, total=   0.0s
[CV] alpha=0.5 …
[CV] … alpha=0.5, total=   0.0s
[CV] alpha=0.5 …
[CV] … alpha=0.5, total=   0.0s
[CV] alpha=0.5 …
[CV] … alpha=0.5, total=   0.0s
[CV] alpha=0.5 …
[CV] … alpha=0.5, total=   0.0s
[CV] alpha=0.5 …
[CV] … alpha=0.5, total=   0.0s
[CV] alpha=1 …
[CV] … alpha=1, total=   0.0s
[CV] alpha=1 …
[CV] … alpha=1, total=   0.0s
[CV] alpha=1 …
[CV] … alpha=1, total=   0.0s
[CV] alpha=1 …
[CV] … alpha=1, total=   0.0s
[CV] alpha=1 …
[CV] … alpha=1, total=   0.0s
```

```
[CV] alpha=1 …
[CV] … alpha=1, total=   0.0s
[CV] alpha=1 …
[CV] … alpha=1, total=   0.0s
[CV] alpha=1 …
[CV] … alpha=1, total=   0.0s
[CV] alpha=1 …
[CV] … alpha=1, total=   0.0s
[CV] alpha=1 …
[CV] … alpha=1, total=   0.0s
[CV] alpha=5 …
[CV] … alpha=5, total=   0.0s
[CV] alpha=5 …
[CV] … alpha=5, total=   0.0s
[CV] alpha=5 …
[CV] … alpha=5, total=   0.0s
[CV] alpha=5 …
[CV] … alpha=5, total=   0.0s
[CV] alpha=5 …
[CV] … alpha=5, total=   0.0s
[CV] alpha=5 …
[CV] … alpha=5, total=   0.0s
[CV] alpha=5 …
[CV] … alpha=5, total=   0.0s
[CV] alpha=5 …
[CV] … alpha=5, total=   0.0s
[CV] alpha=5 …
[CV] … alpha=5, total=   0.0s
[CV] alpha=5 …
[CV] … alpha=5, total=   0.1s
[CV] alpha=10 …
[CV] … alpha=10, total=   0.0s
[CV] alpha=10 …
[CV] … alpha=10, total=   0.0s
[CV] alpha=10 …
[CV] … alpha=10, total=   0.0s
[CV] alpha=10 …
[CV] … alpha=10, total=   0.1s
[CV] alpha=10 …
[CV] … alpha=10, total=   0.0s
[CV] alpha=10 …
[CV] … alpha=10, total=   0.0s
[CV] alpha=10 …
[CV] … alpha=10, total=   0.0s
[CV] alpha=10 …
[CV] … alpha=10, total=   0.0s
[CV] alpha=10 …
[CV] … alpha=10, total=   0.0s
```

```
[CV] alpha=10 …
[CV] … alpha=10, total=   0.0s
[CV] alpha=50 …
[CV] … alpha=50, total=   0.0s
[CV] alpha=50 …
[CV] … alpha=50, total=   0.0s
[CV] alpha=50 …
[CV] … alpha=50, total=   0.0s
[CV] alpha=50 …
[CV] … alpha=50, total=   0.0s
[CV] alpha=50 …
[CV] … alpha=50, total=   0.0s
[CV] alpha=50 …
[CV] … alpha=50, total=   0.0s
[CV] alpha=50 …
[CV] … alpha=50, total=   0.0s
[CV] alpha=50 …
[CV] … alpha=50, total=   0.0s
[CV] alpha=50 …
[CV] … alpha=50, total=   0.0s
[CV] alpha=50 …
[CV] … alpha=50, total=   0.0s
[CV] alpha=100 …
[CV] … alpha=100, total=   0.0s
[CV] alpha=100 …
[CV] … alpha=100, total=   0.0s
[CV] alpha=100 …
[CV] … alpha=100, total=   0.0s
[CV] alpha=100 …
[CV] … alpha=100, total=   0.0s
[CV] alpha=100 …
[CV] … alpha=100, total=   0.0s
[CV] alpha=100 …
[CV] … alpha=100, total=   0.0s
[CV] alpha=100 …
[CV] … alpha=100, total=   0.0s
[CV] alpha=100 …
[CV] … alpha=100, total=   0.0s
[CV] alpha=100 …
[CV] … alpha=100, total=   0.0s
[CV] alpha=100 …
[CV] … alpha=100, total=   0.0s
[CV] alpha=500 …
[CV] … alpha=500, total=   0.0s
[CV] alpha=500 …
[CV] … alpha=500, total=   0.0s
[CV] alpha=500 …
[CV] … alpha=500, total=   0.0s
```

```
[CV] alpha=500 …
[CV] … alpha=500, total=   0.0s
[CV] alpha=500 …
[CV] … alpha=500, total=   0.0s
[CV] alpha=500 …
[CV] … alpha=500, total=   0.0s
[CV] alpha=500 …
[CV] … alpha=500, total=   0.0s
[CV] alpha=500 …
[CV] … alpha=500, total=   0.0s
[CV] alpha=500 …
[CV] … alpha=500, total=   0.0s
[CV] alpha=500 …
[CV] … alpha=500, total=   0.0s
[CV] alpha=1000 …
[CV] … alpha=1000, total=   0.0s
[CV] alpha=1000 …
[CV] … alpha=1000, total=   0.0s
[CV] alpha=1000 …
[CV] … alpha=1000, total=   0.1s
[CV] alpha=1000 …
[CV] … alpha=1000, total=   0.0s
[CV] alpha=1000 …
[CV] … alpha=1000, total=   0.0s
[CV] alpha=1000 …
[CV] … alpha=1000, total=   0.0s
[CV] alpha=1000 …
[CV] … alpha=1000, total=   0.0s
[CV] alpha=1000 …
[CV] … alpha=1000, total=   0.0s
[CV] alpha=1000 …
[CV] … alpha=1000, total=   0.0s
[CV] alpha=1000 …
[CV] … alpha=1000, total=   0.0s

[Parallel(n_jobs=1)]: Done 160 out of 160 | elapsed:   13.3s finished
```

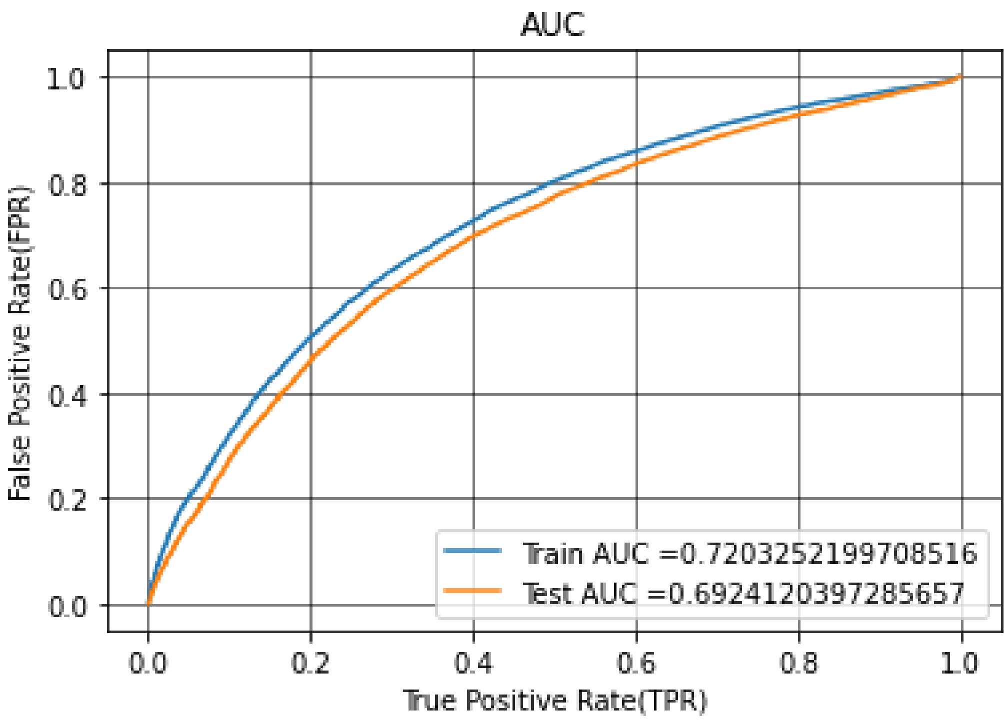
1.14.5 GridSearchCV gives the best parameter as 0.0001

[19]:
```
clf_bow.best_params_
```

[19]:
```
{'alpha': 0.0001}
```

1.14.6 Plotting the AUC Graph with the obtained best hyperparameter

[43]:
```
nb_bow = MultinomialNB(alpha = 0.0001,class_prior=[0.5,0.5])

nb_bow.fit(X_tr_bow, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability␣
↪estimates of the positive class
# not the predicted outputs

y_train_pred_bow = batch_predict(nb_bow, X_tr_bow)
y_test_pred_bow = batch_predict(nb_bow, X_te_bow)

train_fpr_bow, train_tpr_bow, tr_thresholds_bow = roc_curve(y_train,␣
↪y_train_pred_bow)
test_fpr_bow, test_tpr_bow, te_thresholds_bow = roc_curve(y_test,␣
↪y_test_pred_bow)

plt.plot(train_fpr_bow, train_tpr_bow, label="Train AUC␣
↪="+str(auc(train_fpr_bow, train_tpr_bow)))
plt.plot(test_fpr_bow, test_tpr_bow, label="Test AUC ="+str(auc(test_fpr_bow,␣
↪test_tpr_bow)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid(color='black', linestyle='-', linewidth=0.5)
plt.show()
```



[44]:
```
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for␣
↪threshold", np.round(t,3))
    return t

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

1.14.7 Calculating the Confusion Matrix on Train Data

[45]:
```
best_threshold = find_best_threshold(tr_thresholds_bow, train_fpr_bow,␣
↪train_tpr_bow)

conf_mat_train_bow = confusion_matrix(y_train,␣
↪predict_with_best_t(y_train_pred_bow, best_threshold))
```

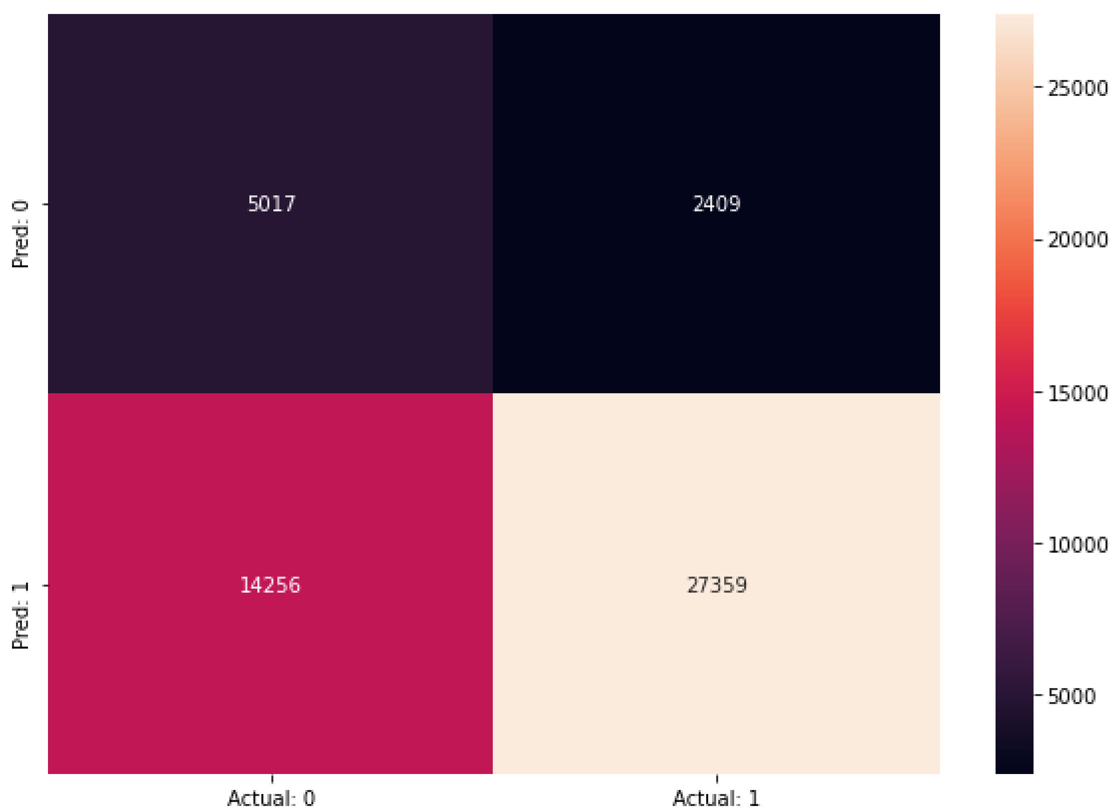
```
print("Train confusion matrix")
print(conf_mat_train_bow)
```

```
the maximum value of tpr*(1-fpr) 0.44416003288192635 for threshold 0.538
Train confusion matrix
[[ 5017  2409]
 [14256 27359]]
```

1.14.8 Displaying the Confusion Matrix

[53]:
```
df_cm = pd.DataFrame(conf_mat_train_bow, index = ["Pred: 0", "Pred: 1"],␣
 ↪columns = ["Actual: 0", "Actual: 1"])
plt.figure(figsize = (10,7))
sns.heatmap(df_cm, annot=True, fmt='g')
```

[53]: <AxesSubplot:>



True Positives : 27539

True Negatives : 5017

False Positives : 14256

False Negatives : 2409

1.14.9 Calculating the Confusion Matrix on Test Data

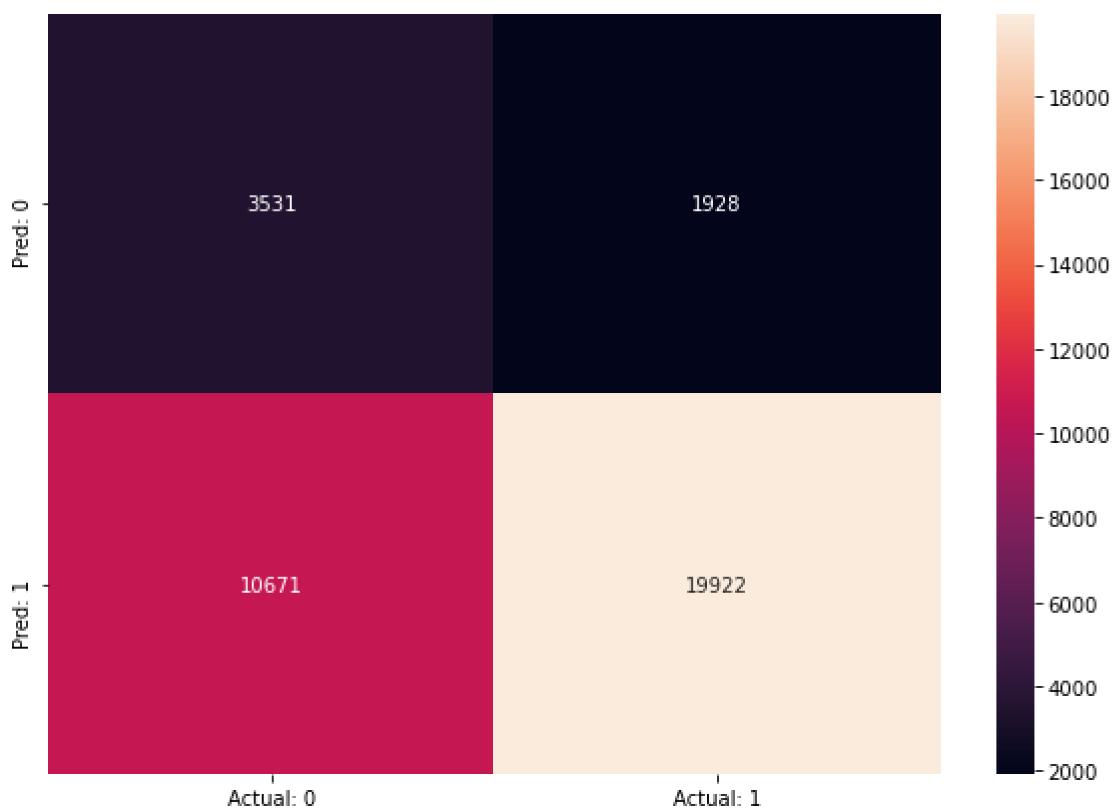
[47]:
```
conf_mat_test_bow = confusion_matrix(y_test,␣
 ↪predict_with_best_t(y_test_pred_bow, best_threshold))
print("Test confusion matrix")
print(conf_mat_test_bow)
```

```
Test confusion matrix
[[ 3531  1928]
 [10671 19922]]
```

1.14.10 Displaying the Confusion Matrix

[54]:
```
df_cm = pd.DataFrame(conf_mat_test_bow, index = ["Pred: 0", "Pred: 1"], columns␣
 ↪= ["Actual: 0", "Actual: 1"])
plt.figure(figsize = (10,7))
sns.heatmap(df_cm, annot=True, fmt='g')
```

[54]: <AxesSubplot:>



True Positives : 19922

True Negatives : 3531

False Positives : 10671

False Negatives : 1928

1.15 TFIDF

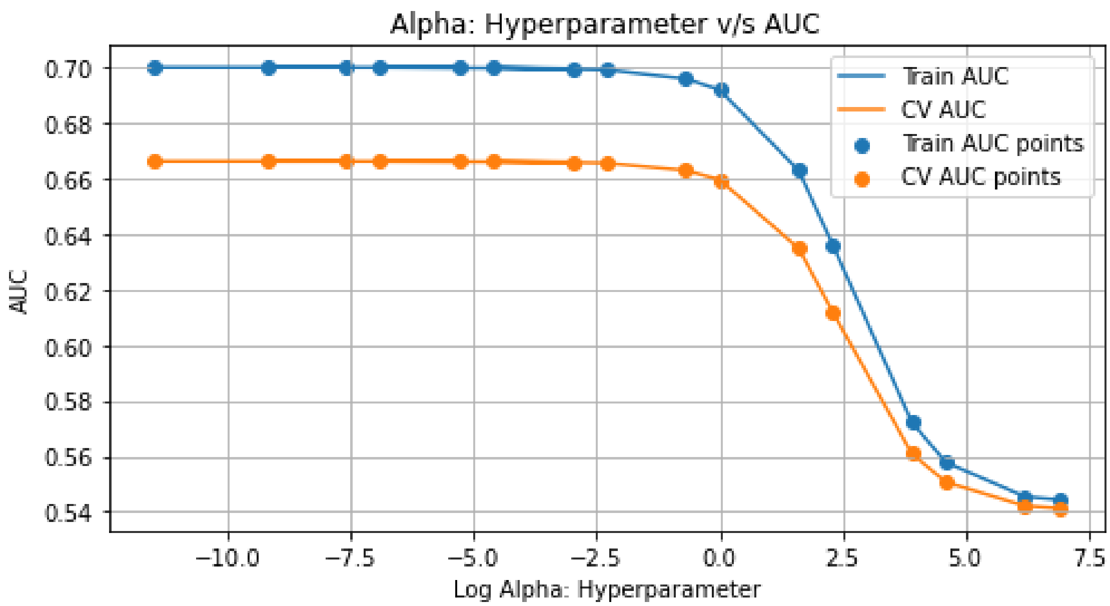
1.15.1 Using various alpha values to calculate the AUC Scores to find the best Alpha value

[26]:
```
train_aucs_tfidf = []
cv_aucs_tfidf = []
log_alphas_tfidf = []
alphas = [0.00001,0.0005, 0.0001,0.005,0.001,0.05,0.01,0.1,0.
↪5,1,5,10,50,100,500,1000]

for alpha in tqdm(alphas):
    nb_tfidf = MultinomialNB(alpha = alpha,class_prior=[0.5,0.5])
    nb_tfidf.fit(X_tr_tfidf, y_train)
    y_train_pred_tfidf = batch_predict(nb_tfidf, X_tr_tfidf)
    y_cv_pred_tfidf = batch_predict(nb_tfidf, X_cr_tfidf)

    train_aucs_tfidf.append(roc_auc_score(y_train,y_train_pred_tfidf))
    cv_aucs_tfidf.append(roc_auc_score(y_cv, y_cv_pred_tfidf))

for a in tqdm(alphas):
    b = math.log(a)
    log_alphas_tfidf.append(b)
```

```
100%|          | 16/16 [00:03<00:00,  4.53it/s]
100%|          | 16/16 [00:00<00:00, 167772.16it/s]
```

1.15.2 Plotting the log of Alphas vs AUC Scores with that Alpha

[27]:
```
plt.figure(figsize=(8,4))
plt.plot(log_alphas_tfidf, train_aucs_tfidf, label='Train AUC')
plt.plot(log_alphas_tfidf, cv_aucs_tfidf, label='CV AUC')

plt.scatter(log_alphas_tfidf, train_aucs_tfidf, label='Train AUC points')
plt.scatter(log_alphas_tfidf, cv_aucs_tfidf, label='CV AUC points')

plt.legend()
plt.xlabel("Log Alpha: Hyperparameter")
plt.ylabel("AUC")
plt.title("Alpha: Hyperparameter v/s AUC")
plt.grid()
plt.show()
```



1.15.3 The curves fall steeply and join after Alpha = 1 (Seventh dot from the right = 50)

1.15.4 Using GridSearchCV to get the best parameter possible

[28]:
```
nb_tfidf = MultinomialNB(class_prior=[0.5,0.5])

parameters = {'alpha':[0.00001,0.0005, 0.0001,0.005,0.001,0.05,0.01,0.1,0.
↪5,1,5,10,50,100,500,1000]}

clf_tfidf = GridSearchCV(nb_tfidf, parameters, cv= 10,␣
↪scoring='roc_auc',return_train_score=True,verbose=2)

clf_tfidf.fit(X_tr_tfidf, y_train)

train_auc_tfidf = clf_tfidf.cv_results_['mean_train_score']
train_auc_std_tfidf = clf_tfidf.cv_results_['std_train_score']
cv_auc_tfidf = clf_tfidf.cv_results_['mean_test_score']
cv_auc_std_tfidf = clf_tfidf.cv_results_['std_test_score']
```

```
Fitting 10 folds for each of 16 candidates, totalling 160 fits
[CV] alpha=1e-05 …
[CV] … alpha=1e-05, total=   0.1s
[CV] alpha=1e-05 …
[CV] … alpha=1e-05, total=   0.0s
[CV] alpha=1e-05 …

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:    0.0s
[CV] … alpha=1e-05, total=   0.0s
[CV] alpha=1e-05 …
[CV] … alpha=1e-05, total=   0.0s
[CV] alpha=1e-05 …
[CV] … alpha=1e-05, total=   0.0s
[CV] alpha=1e-05 …
[CV] … alpha=1e-05, total=   0.0s
[CV] alpha=1e-05 …
[CV] … alpha=1e-05, total=   0.0s
[CV] alpha=1e-05 …
[CV] … alpha=1e-05, total=   0.0s
[CV] alpha=1e-05 …
[CV] … alpha=1e-05, total=   0.0s
[CV] alpha=1e-05 …
[CV] … alpha=1e-05, total=   0.0s
[CV] alpha=0.0005 …
[CV] … alpha=0.0005, total=   0.0s
[CV] alpha=0.0005 …
[CV] … alpha=0.0005, total=   0.0s
[CV] alpha=0.0005 …
[CV] … alpha=0.0005, total=   0.0s
[CV] alpha=0.0005 …
[CV] … alpha=0.0005, total=   0.0s
[CV] alpha=0.0005 …
[CV] … alpha=0.0005, total=   0.0s
[CV] alpha=0.0005 …
[CV] … alpha=0.0005, total=   0.0s
[CV] alpha=0.0005 …
[CV] … alpha=0.0005, total=   0.0s
[CV] alpha=0.0005 …
[CV] … alpha=0.0005, total=   0.0s
[CV] alpha=0.0005 …
[CV] … alpha=0.0005, total=   0.0s
[CV] alpha=0.0005 …
[CV] … alpha=0.0005, total=   0.0s
[CV] alpha=0.0001 …
[CV] … alpha=0.0001, total=   0.0s
[CV] alpha=0.0001 …
[CV] … alpha=0.0001, total=   0.0s
[CV] alpha=0.0001 …
[CV] … alpha=0.0001, total=   0.0s
[CV] alpha=0.0001 …
[CV] … alpha=0.0001, total=   0.0s
[CV] alpha=0.0001 …
[CV] … alpha=0.0001, total=   0.0s
[CV] alpha=0.0001 …
```

```
[CV] … alpha=0.0001, total=   0.0s
[CV] alpha=0.0001 …
[CV] … alpha=0.0001, total=   0.0s
[CV] alpha=0.0001 …
[CV] … alpha=0.0001, total=   0.0s
[CV] alpha=0.0001 …
[CV] … alpha=0.0001, total=   0.0s
[CV] alpha=0.0001 …
[CV] … alpha=0.0001, total=   0.0s
[CV] alpha=0.005 …
[CV] … alpha=0.005, total=   0.0s
[CV] alpha=0.005 …
[CV] … alpha=0.005, total=   0.0s
[CV] alpha=0.005 …
[CV] … alpha=0.005, total=   0.0s
[CV] alpha=0.005 …
[CV] … alpha=0.005, total=   0.0s
[CV] alpha=0.005 …
[CV] … alpha=0.005, total=   0.0s
[CV] alpha=0.005 …
[CV] … alpha=0.005, total=   0.0s
[CV] alpha=0.005 …
[CV] … alpha=0.005, total=   0.0s
[CV] alpha=0.005 …
[CV] … alpha=0.005, total=   0.0s
[CV] alpha=0.005 …
[CV] … alpha=0.005, total=   0.0s
[CV] alpha=0.005 …
[CV] … alpha=0.005, total=   0.0s
[CV] alpha=0.001 …
[CV] … alpha=0.001, total=   0.0s
[CV] alpha=0.001 …
[CV] … alpha=0.001, total=   0.0s
[CV] alpha=0.001 …
[CV] … alpha=0.001, total=   0.0s
[CV] alpha=0.001 …
[CV] … alpha=0.001, total=   0.0s
[CV] alpha=0.001 …
[CV] … alpha=0.001, total=   0.0s
[CV] alpha=0.001 …
[CV] … alpha=0.001, total=   0.0s
[CV] alpha=0.001 …
[CV] … alpha=0.001, total=   0.0s
[CV] alpha=0.001 …
[CV] … alpha=0.001, total=   0.0s
[CV] alpha=0.001 …
[CV] … alpha=0.001, total=   0.0s
[CV] alpha=0.001 …
```

```
[CV] … alpha=0.001, total=   0.0s
[CV] alpha=0.05 …
[CV] … alpha=0.05, total=   0.0s
[CV] alpha=0.05 …
[CV] … alpha=0.05, total=   0.0s
[CV] alpha=0.05 …
[CV] … alpha=0.05, total=   0.0s
[CV] alpha=0.05 …
[CV] … alpha=0.05, total=   0.0s
[CV] alpha=0.05 …
[CV] … alpha=0.05, total=   0.0s
[CV] alpha=0.05 …
[CV] … alpha=0.05, total=   0.0s
[CV] alpha=0.05 …
[CV] … alpha=0.05, total=   0.0s
[CV] alpha=0.05 …
[CV] … alpha=0.05, total=   0.0s
[CV] alpha=0.05 …
[CV] … alpha=0.05, total=   0.0s
[CV] alpha=0.05 …
[CV] … alpha=0.05, total=   0.0s
[CV] alpha=0.01 …
[CV] … alpha=0.01, total=   0.0s
[CV] alpha=0.01 …
[CV] … alpha=0.01, total=   0.0s
[CV] alpha=0.01 …
[CV] … alpha=0.01, total=   0.0s
[CV] alpha=0.01 …
[CV] … alpha=0.01, total=   0.0s
[CV] alpha=0.01 …
[CV] … alpha=0.01, total=   0.0s
[CV] alpha=0.01 …
[CV] … alpha=0.01, total=   0.0s
[CV] alpha=0.01 …
[CV] … alpha=0.01, total=   0.0s
[CV] alpha=0.01 …
[CV] … alpha=0.01, total=   0.0s
[CV] alpha=0.01 …
[CV] … alpha=0.01, total=   0.0s
[CV] alpha=0.01 …
[CV] … alpha=0.01, total=   0.1s
[CV] alpha=0.1 …
[CV] … alpha=0.1, total=   0.0s
[CV] alpha=0.1 …
[CV] … alpha=0.1, total=   0.0s
[CV] alpha=0.1 …
[CV] … alpha=0.1, total=   0.0s
[CV] alpha=0.1 …
```

```
[CV] … alpha=0.1, total=   0.0s
[CV] alpha=0.1 …
[CV] … alpha=0.1, total=   0.0s
[CV] alpha=0.1 …
[CV] … alpha=0.1, total=   0.0s
[CV] alpha=0.1 …
[CV] … alpha=0.1, total=   0.0s
[CV] alpha=0.1 …
[CV] … alpha=0.1, total=   0.0s
[CV] alpha=0.1 …
[CV] … alpha=0.1, total=   0.0s
[CV] alpha=0.1 …
[CV] … alpha=0.1, total=   0.0s
[CV] alpha=0.5 …
[CV] … alpha=0.5, total=   0.0s
[CV] alpha=0.5 …
[CV] … alpha=0.5, total=   0.0s
[CV] alpha=0.5 …
[CV] … alpha=0.5, total=   0.0s
[CV] alpha=0.5 …
[CV] … alpha=0.5, total=   0.0s
[CV] alpha=0.5 …
[CV] … alpha=0.5, total=   0.0s
[CV] alpha=0.5 …
[CV] … alpha=0.5, total=   0.0s
[CV] alpha=0.5 …
[CV] … alpha=0.5, total=   0.0s
[CV] alpha=0.5 …
[CV] … alpha=0.5, total=   0.0s
[CV] alpha=0.5 …
[CV] … alpha=0.5, total=   0.0s
[CV] alpha=0.5 …
[CV] … alpha=0.5, total=   0.0s
[CV] alpha=1 …
[CV] … alpha=1, total=   0.0s
[CV] alpha=1 …
[CV] … alpha=1, total=   0.0s
[CV] alpha=1 …
[CV] … alpha=1, total=   0.0s
[CV] alpha=1 …
[CV] … alpha=1, total=   0.0s
[CV] alpha=1 …
[CV] … alpha=1, total=   0.0s
[CV] alpha=1 …
[CV] … alpha=1, total=   0.0s
[CV] alpha=1 …
[CV] … alpha=1, total=   0.0s
[CV] alpha=1 …
```

```
[CV] … alpha=1, total=   0.0s
[CV] alpha=1 …
[CV] … alpha=1, total=   0.0s
[CV] alpha=1 …
[CV] … alpha=1, total=   0.0s
[CV] alpha=5 …
[CV] … alpha=5, total=   0.0s
[CV] alpha=5 …
[CV] … alpha=5, total=   0.0s
[CV] alpha=5 …
[CV] … alpha=5, total=   0.0s
[CV] alpha=5 …
[CV] … alpha=5, total=   0.0s
[CV] alpha=5 …
[CV] … alpha=5, total=   0.0s
[CV] alpha=5 …
[CV] … alpha=5, total=   0.0s
[CV] alpha=5 …
[CV] … alpha=5, total=   0.0s
[CV] alpha=5 …
[CV] … alpha=5, total=   0.0s
[CV] alpha=5 …
[CV] … alpha=5, total=   0.0s
[CV] alpha=5 …
[CV] … alpha=5, total=   0.0s
[CV] alpha=10 …
[CV] … alpha=10, total=   0.0s
[CV] alpha=10 …
[CV] … alpha=10, total=   0.0s
[CV] alpha=10 …
[CV] … alpha=10, total=   0.0s
[CV] alpha=10 …
[CV] … alpha=10, total=   0.0s
[CV] alpha=10 …
[CV] … alpha=10, total=   0.0s
[CV] alpha=10 …
[CV] … alpha=10, total=   0.0s
[CV] alpha=10 …
[CV] … alpha=10, total=   0.0s
[CV] alpha=10 …
[CV] … alpha=10, total=   0.0s
[CV] alpha=10 …
[CV] … alpha=10, total=   0.0s
[CV] alpha=10 …
[CV] … alpha=10, total=   0.0s
[CV] alpha=50 …
[CV] … alpha=50, total=   0.0s
[CV] alpha=50 …
```

```
[CV] … alpha=50, total=   0.0s
[CV] alpha=50 …
[CV] … alpha=50, total=   0.0s
[CV] alpha=50 …
[CV] … alpha=50, total=   0.0s
[CV] alpha=50 …
[CV] … alpha=50, total=   0.0s
[CV] alpha=50 …
[CV] … alpha=50, total=   0.0s
[CV] alpha=50 …
[CV] … alpha=50, total=   0.0s
[CV] alpha=50 …
[CV] … alpha=50, total=   0.0s
[CV] alpha=50 …
[CV] … alpha=50, total=   0.0s
[CV] alpha=50 …
[CV] … alpha=50, total=   0.0s
[CV] alpha=100 …
[CV] … alpha=100, total=   0.0s
[CV] alpha=100 …
[CV] … alpha=100, total=   0.1s
[CV] alpha=100 …
[CV] … alpha=100, total=   0.0s
[CV] alpha=100 …
[CV] … alpha=100, total=   0.0s
[CV] alpha=100 …
[CV] … alpha=100, total=   0.0s
[CV] alpha=100 …
[CV] … alpha=100, total=   0.0s
[CV] alpha=100 …
[CV] … alpha=100, total=   0.0s
[CV] alpha=100 …
[CV] … alpha=100, total=   0.0s
[CV] alpha=100 …
[CV] … alpha=100, total=   0.0s
[CV] alpha=100 …
[CV] … alpha=100, total=   0.0s
[CV] alpha=500 …
[CV] … alpha=500, total=   0.0s
[CV] alpha=500 …
[CV] … alpha=500, total=   0.0s
[CV] alpha=500 …
[CV] … alpha=500, total=   0.0s
[CV] alpha=500 …
[CV] … alpha=500, total=   0.0s
[CV] alpha=500 …
[CV] … alpha=500, total=   0.0s
[CV] alpha=500 …
```

```
[CV] … alpha=500, total=   0.0s
[CV] alpha=500 …
[CV] … alpha=500, total=   0.0s
[CV] alpha=500 …
[CV] … alpha=500, total=   0.0s
[CV] alpha=500 …
[CV] … alpha=500, total=   0.0s
[CV] alpha=500 …
[CV] … alpha=500, total=   0.0s
[CV] alpha=1000 …
[CV] … alpha=1000, total=   0.0s
[CV] alpha=1000 …
[CV] … alpha=1000, total=   0.0s
[CV] alpha=1000 …
[CV] … alpha=1000, total=   0.0s
[CV] alpha=1000 …
[CV] … alpha=1000, total=   0.0s
[CV] alpha=1000 …
[CV] … alpha=1000, total=   0.0s
[CV] alpha=1000 …
[CV] … alpha=1000, total=   0.0s
[CV] alpha=1000 …
[CV] … alpha=1000, total=   0.0s
[CV] alpha=1000 …
[CV] … alpha=1000, total=   0.0s
[CV] alpha=1000 …
[CV] … alpha=1000, total=   0.0s
[CV] alpha=1000 …
[CV] … alpha=1000, total=   0.0s

[Parallel(n_jobs=1)]: Done 160 out of 160 | elapsed:   13.1s finished
```

[29]:
```
clf_tfidf.best_params_
```

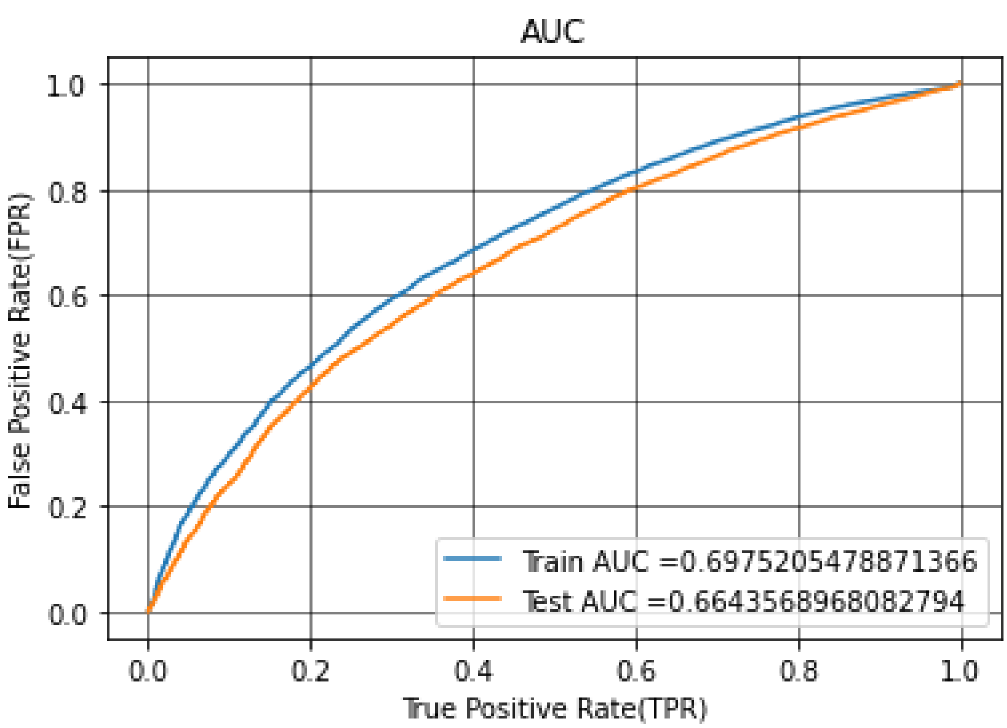
[29]:
```
{'alpha': 1e-05}
```

1.15.5 GridSearchCV gives the best parameter as 0.00001

1.15.6 Plotting the AUC Graph with the obtained best hyperparameter

[30]:
```
nb_tfidf = MultinomialNB(alpha = 0.00001,class_prior=[0.5,0.5])

nb_tfidf.fit(X_tr_tfidf, y_train)
# roc_auc_score(y_true, y_score) the 2nd parameter should be probability␣
↪estimates of the positive class
# not the predicted outputs

y_train_pred_tfidf = batch_predict(nb_bow, X_tr_tfidf)
```

```
y_test_pred_tfidf = batch_predict(nb_bow, X_te_tfidf)

train_fpr_tfidf, train_tpr_tfidf, tr_thresholds_tfidf = roc_curve(y_train,␣
 ↪y_train_pred_tfidf)
test_fpr_tfidf, test_tpr_tfidf, te_thresholds_tfidf = roc_curve(y_test,␣
 ↪y_test_pred_tfidf)

plt.plot(train_fpr_tfidf, train_tpr_tfidf, label="Train AUC␣
 ↪="+str(auc(train_fpr_tfidf, train_tpr_tfidf)))
plt.plot(test_fpr_tfidf, test_tpr_tfidf, label="Test AUC␣
 ↪="+str(auc(test_fpr_tfidf, test_tpr_tfidf)))
plt.legend()
plt.xlabel("True Positive Rate(TPR)")
plt.ylabel("False Positive Rate(FPR)")
plt.title("AUC")
plt.grid(color='black', linestyle='-', linewidth=0.5)
plt.show()
```



1.15.7 Calculating the Confusion Matrix on Train Data

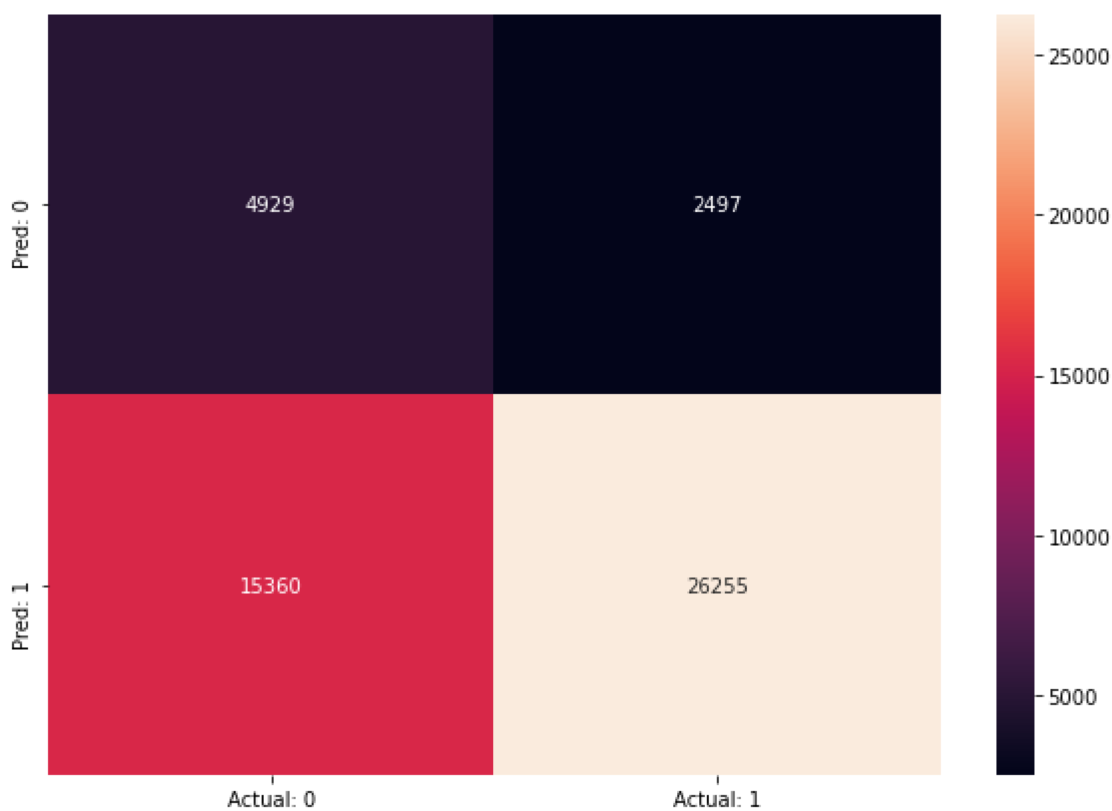
[31]:
```
best_threshold = find_best_threshold(tr_thresholds_tfidf, train_fpr_tfidf,␣
↪train_tpr_tfidf)

conf_mat_train_tfidf = confusion_matrix(y_train,␣
↪predict_with_best_t(y_train_pred_tfidf, best_threshold))
print("Train confusion matrix")
print(conf_mat_train_tfidf)
```

```
the maximum value of tpr*(1-fpr) 0.4187607769642976 for threshold 0.44
Train confusion matrix
[[ 4929  2497]
 [15360 26255]]
```

1.16 Plotting the Confusion Matrix

[55]:
```
df_cm = pd.DataFrame(conf_mat_train_tfidf, index = ["Pred: 0", "Pred: 1"],␣
↪columns = ["Actual: 0", "Actual: 1"])
plt.figure(figsize = (10,7))
sns.heatmap(df_cm, annot=True, fmt='g')
```

[55]: <AxesSubplot:>



True Positives : 26255

True Negatives : 4929

False Positives : 15360

False Negatives : 2497

1.16.1 Calculating the Confusion Matrix on Test Data

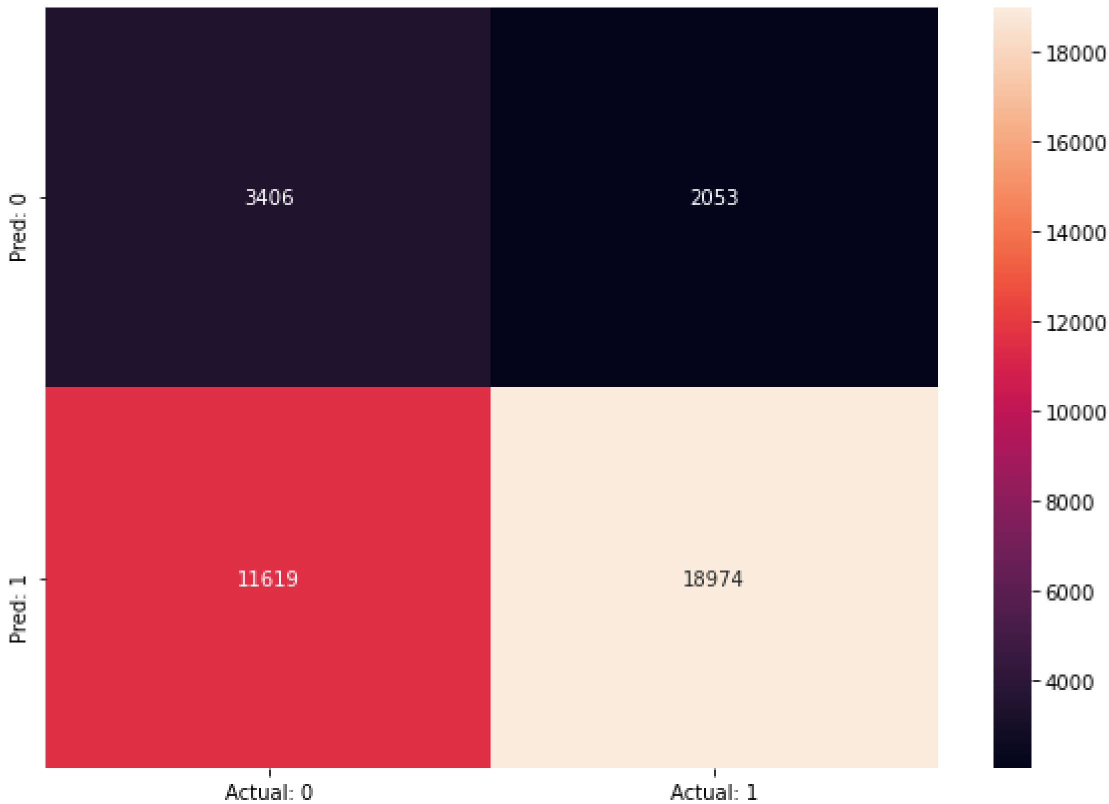
[33]:
```
conf_mat_test_tfidf = confusion_matrix(y_test,␣
 ↪predict_with_best_t(y_test_pred_tfidf, best_threshold))
print("Test confusion matrix")
print(conf_mat_test_tfidf)
```

```
Test confusion matrix
[[ 3406  2053]
 [11619 18974]]
```

1.17 Plotting the Confusion Matrix

[57]:
```
df_cm = pd.DataFrame(conf_mat_test_tfidf, index = ["Pred: 0", "Pred: 1"],␣
 ↪columns = ["Actual: 0", "Actual: 1"])
plt.figure(figsize = (10,7))
sns.heatmap(df_cm, annot=True, fmt='g')
```

[57]: <AxesSubplot:>



True Positives : 18974

True Negatives : 3406

False Positives : 11619

False Negatives : 2053

3. Summary

as mentioned in the step 5 of instructions

1.18 Getting the top positive and negative features

[35]:
```
bow_features = []

for feature in vectorizer_state.get_feature_names():
    bow_features.append(feature)
for feature in vectorizer_grade.get_feature_names():
    bow_features.append(feature)
for feature in vectorizer_teacher_prefix.get_feature_names():
    bow_features.append(feature)
for feature in vectorizer_cat.get_feature_names():
    bow_features.append(feature)
```

```python
for feature in vectorizer_subcat.get_feature_names():
    bow_features.append(feature)

bow_features.append("price")
bow_features.append("prev_proj")

for feature in vectorizer_bow.get_feature_names():
    bow_features.append(feature)
```

[36]:
```python
len(bow_features)
```

[36]: 5101

[37]:
```python
pos_class_prob_sorted = nb_bow.feature_log_prob_[1, :].argsort()[::-1]
for i in pos_class_prob_sorted[:30]:
    print(bow_features[i])
```

```
students
school
my
learning
classroom
the
they
not
my students
learn
help
price
many
nannan
we
need
reading
work
use
prev_proj
love
able
day
come
class
would
technology
our
also
books
```

[38]:
```
neg_class_prob_sorted_tfidf = nb_bow.feature_log_prob_[0, :].argsort()[::-1]
for i in neg_class_prob_sorted_tfidf[:30]:
    print(bow_features[i])
```

```
students
school
learning
my
classroom
not
learn
help
they
my students
the
price
nannan
many
we
need
work
come
prev_proj
love
able
materials
reading
use
skills
day
class
our
want
year
```

[39]:
```
tfidf_features = []

for feature in vectorizer_state.get_feature_names():
    tfidf_features.append(feature)
for feature in vectorizer_grade.get_feature_names():
    tfidf_features.append(feature)
for feature in vectorizer_teacher_prefix.get_feature_names():
    tfidf_features.append(feature)
for feature in vectorizer_cat.get_feature_names():
    tfidf_features.append(feature)
for feature in vectorizer_subcat.get_feature_names():
    tfidf_features.append(feature)
```

```
tfidf_features.append("price")
tfidf_features.append("prev_proj")

for feature in vectorizer_tfidf.get_feature_names():
    tfidf_features.append(feature)
```

[40]:
```
pos_class_prob_sorted = nb_tfidf.feature_log_prob_[0, :].argsort()[::-1]
for i in pos_class_prob_sorted[:30]:
    print(tfidf_features[i])
```

```
price
prev_proj
mrs
literacy_language
grades_prek_2
math_science
ms
grades_3_5
mathematics
literacy
literature_writing
grades_6_8
specialneeds
specialneeds
health_sports
ca
appliedlearning
students
appliedsciences
grades_9_12
mr
music_arts
health_wellness
tx
fl
visualarts
environmentalscience
ny
history_civics
earlydevelopment
```

[41]:
```
pos_class_prob_sorted = nb_tfidf.feature_log_prob_[1, :].argsort()[::-1]
for i in pos_class_prob_sorted[:30]:
    print(tfidf_features[i])
```

```
price
prev_proj
mrs
```

```
literacy_language
grades_prek_2
math_science
ms
grades_3_5
literacy
mathematics
literature_writing
grades_6_8
ca
health_sports
students
specialneeds
specialneeds
appliedlearning
grades_9_12
appliedsciences
mr
health_wellness
music_arts
ny
tx
fl
history_civics
visualarts
environmentalscience
nc
```

1.19 Printing the Summary Table

[49]:

```
x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Alpha", " Test AUC"]

x.add_row(["BOW", "Naive Bayes", 0.0001, 0.69])
x.add_row(["TFIDF", "Naive Bayes", 1e-05, 0.66])

print(x)
```

```
+------------+-------------+--------+-----------+
| Vectorizer |    Model    | Alpha  |  Test AUC |
+------------+-------------+--------+-----------+
|    BOW     | Naive Bayes | 0.0001 |    0.69   |
|   TFIDF    | Naive Bayes | 1e-05  |    0.66   |
+------------+-------------+--------+-----------+
```

1.20 Conclusions

1.21 1. There are similar words in top positive and negative features of both models

1.22 2. Bag of Words gives a better Test AUC Score thatn TFIDF

2 References Used:

1. http://zetcode.com/python/prettytable/
2. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html

[ ]:

```

```