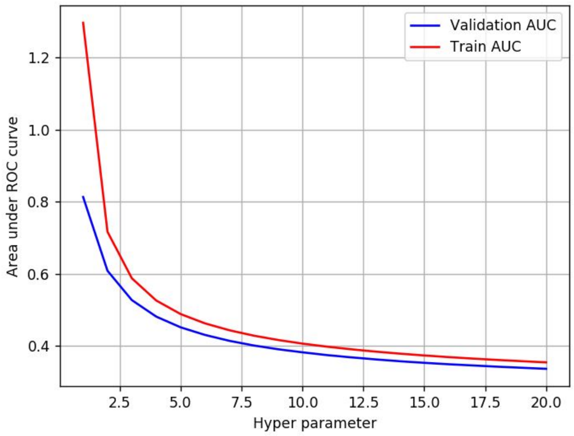
Assignment 6: Apply NB

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)](https://imgur.com/ldZA1zg) and [this (https://ac-classroom-production.s3.amazonaws.com/public/COMMENT/Annotation_2020-05-21_225912_0lyZzN8.jpg)](https://ac-classroom-production.s3.amazonaws.com/public/COMMENT/Annotation_2020-05-21_225912_0lyZzN8.jpg)

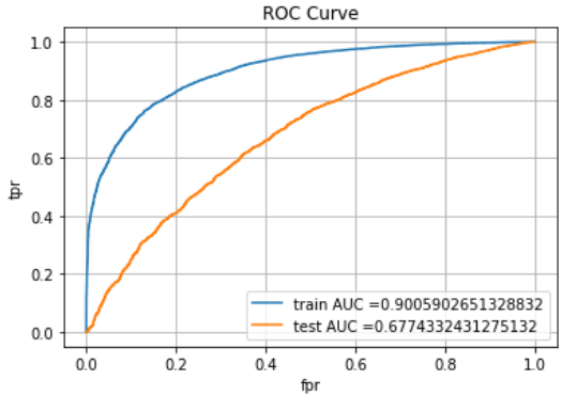
    - Set 1: categorical, numerical features + preprocessed_eassay (BOW)
    - Set 2: categorical, numerical features + preprocessed_eassay (TFIDF)
6. **The hyper paramter tuning(find best alpha:smoothing parameter)**

    - 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)](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/)](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

Naive Bayes

1. Importing libraries

In [1]:

```
import pandas as pd
import numpy as np
from tqdm import tqdm
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.sparse import hstack
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import Normalizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import confusion_matrix
from prettytable import PrettyTable
```

2. Loading Data

In [2]:

```
#taking 80k datapoints
data= pd.read_csv('preprocessed_data.csv', nrows=80000)
data.shape
```

Out[2]:

```
(80000, 9)
```

In [3]:

```
#features
x = data.drop(['project_is_approved'], axis=1)
#class label
y = data['project_is_approved'].values
x.head(1)
```

Out[3]:

	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_p
0	ca	mrs	grades_prek_2	

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

In [4]:

```
# Train Test Stratified Split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.33, strati
fy=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)
```

```
(35912, 8) (35912,)
(17688, 8) (17688,)
(26400, 8) (26400,)
```

4. Make Data Model Ready : encoding features

4.1. encoding numerical features :

4.1.1. price

In [5]:

```
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)
```

```
After vectorizations
(35912, 1) (35912,)
(17688, 1) (17688,)
(26400, 1) (26400,)
```

4.1.2. teacher previously posted projects

In [6]:

```
normalizer = Normalizer()
normalizer.fit(X_train['teacher_number_of_previously_posted_projects'].values.re
shape(-1,1))

X_train_preProject_norm = normalizer.transform(X_train['teacher_number_of_previo
usly_posted_projects'].values.reshape(-1,1))
X_cv_preProject_norm = normalizer.transform(X_cv['teacher_number_of_previously_p
osted_projects'].values.reshape(-1,1))
X_test_preProject_norm = normalizer.transform(X_test['teacher_number_of_previous
ly_posted_projects'].values.reshape(-1,1))

print("After vectorizations")
print(X_train_preProject_norm.shape, y_train.shape)
print(X_cv_preProject_norm.shape, y_cv.shape)
print(X_test_preProject_norm.shape, y_test.shape)
```

```
After vectorizations
(35912, 1) (35912,)
(17688, 1) (17688,)
(26400, 1) (26400,)
```

4.2. encoding categorical features :

4.2.1. school state

In [7]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['school_state'].values)

X_train_state_ohe = vectorizer.transform(X_train['school_state'].values)
X_cv_state_ohe = vectorizer.transform(X_cv['school_state'].values)
X_test_state_ohe = vectorizer.transform(X_test['school_state'].values)
#to get feature_names
school_state_feature = vectorizer.get_feature_names()

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)
```

```
After vectorizations
(35912, 51) (35912,)
(17688, 51) (17688,)
(26400, 51) (26400,)
```

4.2.2 teacher_prefix

In [8]:

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

X_train_teacher_ohe = vectorizer.transform(X_train['teacher_prefix'].values)
X_cv_teacher_ohe = vectorizer.transform(X_cv['teacher_prefix'].values)
X_test_teacher_ohe = vectorizer.transform(X_test['teacher_prefix'].values)
#to get feature_names
teacher_prefix_feature = vectorizer.get_feature_names()

print("After vectorizations")
print(X_train_teacher_ohe.shape, y_train.shape)
print(X_cv_teacher_ohe.shape, y_cv.shape)
print(X_test_teacher_ohe.shape, y_test.shape)
```

```
After vectorizations
(35912, 5) (35912,)
(17688, 5) (17688,)
(26400, 5) (26400,)
```

4.2.3. project_grade_category

In [9]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['project_grade_category'].values)

X_train_grade_ohe = vectorizer.transform(X_train['project_grade_category'].value
s)
X_cv_grade_ohe = vectorizer.transform(X_cv['project_grade_category'].values)
X_test_grade_ohe = vectorizer.transform(X_test['project_grade_category'].values)
#to get feature_names
grade_feature = vectorizer.get_feature_names()

print("After vectorizations")
print(X_train_grade_ohe.shape, y_train.shape)
print(X_cv_grade_ohe.shape, y_cv.shape)
print(X_test_grade_ohe.shape, y_test.shape)
```

```
After vectorizations
(35912, 4) (35912,)
(17688, 4) (17688,)
(26400, 4) (26400,)
```

4.2.4. clean_categories

In [10]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_categories'].values)

X_train_category_ohe = vectorizer.transform(X_train['clean_categories'].values)
X_cv_category_ohe = vectorizer.transform(X_cv['clean_categories'].values)
X_test_category_ohe = vectorizer.transform(X_test['clean_categories'].values)
#to get feature_names
category_feature = vectorizer.get_feature_names()

print("After vectorizations")
print(X_train_category_ohe.shape, y_train.shape)
print(X_cv_category_ohe.shape, y_cv.shape)
print(X_test_category_ohe.shape, y_test.shape)
```

```
After vectorizations
(35912, 9) (35912,)
(17688, 9) (17688,)
(26400, 9) (26400,)
```

4.2.5. clean_subcategories

In [11]:

```
vectorizer = CountVectorizer()
vectorizer.fit(X_train['clean_subcategories'].values)

X_train_subcategory_ohe = vectorizer.transform(X_train['clean_subcategories'].va
lues)
X_cv_subcategory_ohe = vectorizer.transform(X_cv['clean_subcategories'].values)
X_test_subcategory_ohe = vectorizer.transform(X_test['clean_subcategories'].valu
es)
#to get feature_names
subcategory_feature = vectorizer.get_feature_names()

print("After vectorizations")
print(X_train_subcategory_ohe.shape, y_train.shape)
print(X_cv_subcategory_ohe.shape, y_cv.shape)
print(X_test_subcategory_ohe.shape, y_test.shape)
```

```
After vectorizations
(35912, 30) (35912,)
(17688, 30) (17688,)
(26400, 30) (26400,)
```

4.3. encoding text features :

4.3.1 essay (BOW)

In [12]:

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

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)
#to get feature_names
essay_bow_feature = vectorizer.get_feature_names()

print("After count 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 count vectorizations
(35912, 5000) (35912,)
(17688, 5000) (17688,)
(26400, 5000) (26400,)
```

4.3.1 essay (TFIDF)

In [13]:

```
vectorizer = TfidfVectorizer(min_df=10, ngram_range=(1,4), max_features=5000)
vectorizer.fit(X_train["essay"].values)

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)
#to get feature_names
essay_tfidf_feature = vectorizer.get_feature_names()

print("After TFIDF 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)
```

```
After TFIDF Vectorization
(35912, 5000) (35912,)
(17688, 5000) (17688,)
(26400, 5000) (26400,)
```

5. Appling NB on different kind of featurization

5.1. Applying NB on numerical, categorical features + essay(BOW) : Set 1

In [14]:

```
# concatinating all the features
X_tr_1 = hstack((X_train_price_norm, X_train_preProject_norm,
               X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe,
               X_train_category_ohe, X_train_subcategory_ohe, X_train_essay_bow
)).tocsr()

X_cv_1 = hstack((X_cv_price_norm,X_cv_preProject_norm,
               X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe,
               X_cv_category_ohe, X_cv_subcategory_ohe, X_cv_essay_bow)).tocsr()

X_te_1 = hstack((X_test_price_norm,X_test_preProject_norm,
               X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe,
               X_test_category_ohe, X_test_subcategory_ohe, X_test_essay_bow)).t
ocsr()

print("Final Data matrix for SET 1")
print(X_tr_1.shape, y_train.shape)
print(X_cv_1.shape, y_cv.shape)
print(X_te_1.shape, y_test.shape)
```

```
Final Data matrix for SET 1
(35912, 5101) (35912,)
(17688, 5101) (17688,)
(26400, 5101) (26400,)
```

5.1.1. The hyper paramter tuning (finding the best alpha : smoothing parameter)

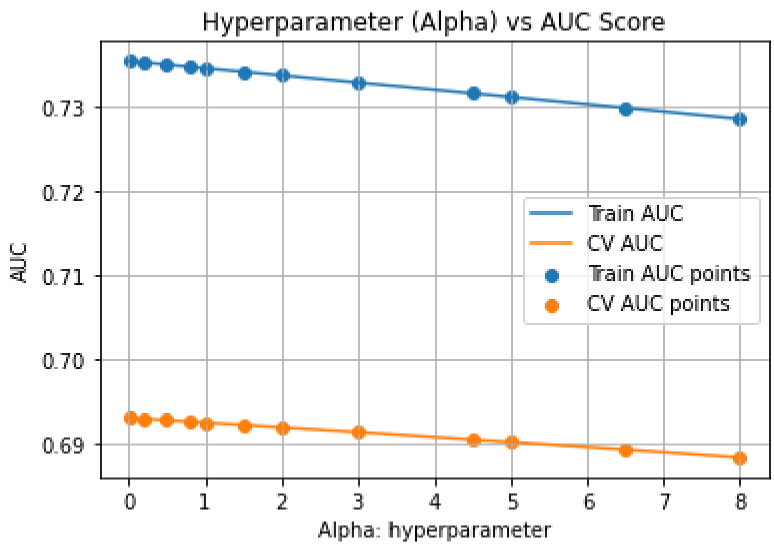
In [15]:

```
def batch_predict(clf, data):
    """function to perform batch predict"""
    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        #predict_proba gives you the probabilities for the target [1] in array f
orm
        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
```

In [16]:

```
train_auc = []
cv_auc = []
alpha = [0.01,0.2,0.5,0.8,1,1.5,2,3,4.5,5,6.5,8]

for i in tqdm(alpha):
    m_nb = MultinomialNB(alpha=i)
    m_nb.fit(X_tr_1, y_train)

    y_train_pred = batch_predict(m_nb, X_tr_1)
    y_cv_pred = batch_predict(m_nb, X_cv_1)

    # roc_auc_score(y_actual, y_prob_score)
    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("Alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyperparameter (Alpha) vs AUC Score")
plt.grid()
plt.show()
```

```
100%|██████████| 12/12 [00:13<00:00,  1.12s/it]
```



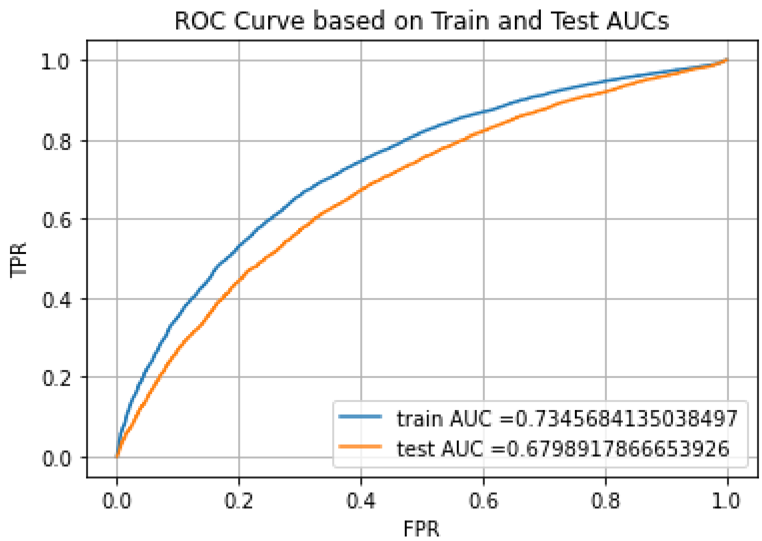
Observation:

From the plot, best alpha found as 1.0 with minum difference between Train and CV AUCs

5.1.2. Finding the AUC on test data and ploting the ROC curve on both train and test

In [17]:

```
best_alpha = 1.0

m_nb = MultinomialNB(alpha = best_alpha)
m_nb.fit(X_tr_1, y_train)

y_train_pred = batch_predict(m_nb, X_tr_1)
y_test_pred = batch_predict(m_nb, X_te_1)

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr
)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))

plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve based on Train and Test AUCs")
plt.grid()
plt.show()
```



5.1.3. Plotting the Confusion Matrix

In [18]:

```
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshold, fpr, tpr):
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    t = threshold[np.argmax(tpr*(1-fpr))]
    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
```

In [19]:

```
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
train_cm = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
test_cm = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
print("Train confusion matrix \n",train_cm)
print("Test confusion matrix \n",test_cm)
```

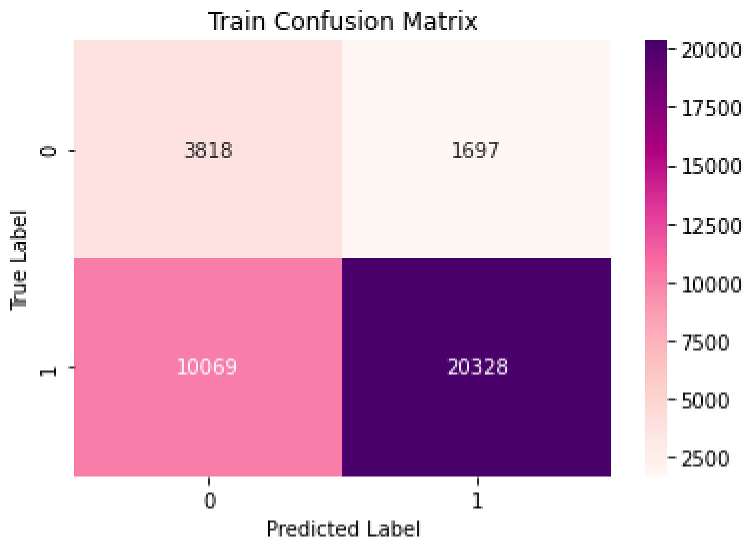
```
The maximum value of tpr*(1-fpr) 0.46297158386729426 for threshold
0.869
Train confusion matrix
 [[ 3818  1697]
 [10069 20328]]
Test confusion matrix
 [[ 2508  1546]
 [ 7761 14585]]
```

In [20]:

```
##https://stackoverflow.com/questions/61748441/how-to-fix-the-values-displayed-i
n-a-confusion-matrix-in-exponential-form-to-nor
sns.heatmap(train_cm, annot=True, fmt="d",cmap="RdPu")
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Train Confusion Matrix')
```

Out[20]:

```
Text(0.5, 1.0, 'Train Confusion Matrix')
```

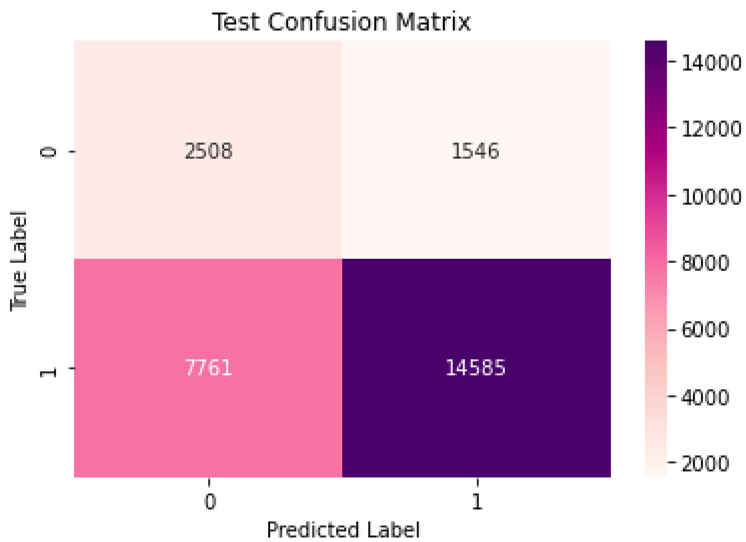


In [21]:

```
sns.heatmap(test_cm, annot=True, fmt="d",cmap="RdPu")
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Test Confusion Matrix')
```

Out[21]:

```
Text(0.5, 1.0, 'Test Confusion Matrix')
```



5.1.4. Code to find the top 20 features from Set 1

In [22]:

```
# list of all the feature names
feature_names_set1 = []

numerical = ['price','teacher_number_of_previously_posted_projects']
categorical= school_state_feature + teacher_prefix_feature + grade_feature + cat
egory_feature +subcategory_feature
text = essay_bow_feature

feature_names_set1.extend(numerical)
feature_names_set1.extend(categorical)
feature_names_set1.extend(text)

# to check len of feature_names equal to dimensions of final stacked matrix
print(len(feature_names_set1))
```

```
5101
```

In [23]:

```
#https://www.semicolonworld.com/question/54572/is-it-possible-to-use-argsort-in-
descending-order
max_ind_pos = m_nb.feature_log_prob_[1].argsort()[::-1][0:20]
max_ind_neg = m_nb.feature_log_prob_[0].argsort()[::-1][0:20]

top_feature_pos = np.take(feature_names_set1,max_ind_pos)
top_feature_neg = np.take(feature_names_set1,max_ind_neg)

print("Top 20 features for positive class: \n\n", top_feature_pos)
print("\n Top 20 features for negative class: \n\n",top_feature_neg)
```

```
Top 20 features for positive class:

 ['students' 'school' 'my' 'learning' 'classroom' 'the' 'not' 'they'
 'my students' 'learn' 'help' 'price' 'many' 'nannan' 'we' 'reading'
 'work' 'need' 'use' 'love']

 Top 20 features for negative class:

 ['students' 'school' 'learning' 'my' 'classroom' 'not' 'learn' 'the
y'
 'help' 'the' 'my students' 'price' 'nannan' 'many' 'we' 'need' 'wor
k'
 'come' 'reading' 'teacher_number_of_previously_posted_projects']
```

5.2. Applying NB on numerical, categorical features + essay (TFIDF) : Set 2

In [24]:

```
X_tr_2 = hstack((X_train_price_norm, X_train_preProject_norm,
               X_train_state_ohe, X_train_teacher_ohe, X_train_grade_ohe,
               X_train_category_ohe, X_train_subcategory_ohe, X_train_essay_tfid
f)).tocsr()

X_cv_2 = hstack((X_cv_price_norm,X_cv_preProject_norm,
               X_cv_state_ohe, X_cv_teacher_ohe, X_cv_grade_ohe,
               X_cv_category_ohe, X_cv_subcategory_ohe, X_cv_essay_tfidf)).tocsr
()

X_te_2 = hstack((X_test_price_norm,X_test_preProject_norm,
               X_test_state_ohe, X_test_teacher_ohe, X_test_grade_ohe,
               X_test_category_ohe, X_test_subcategory_ohe, X_test_essay_tfidf))
.tocsr()

print("Final Data matrix for SET 2")
print(X_tr_2.shape, y_train.shape)
print(X_cv_2.shape, y_cv.shape)
print(X_te_2.shape, y_test.shape)
```

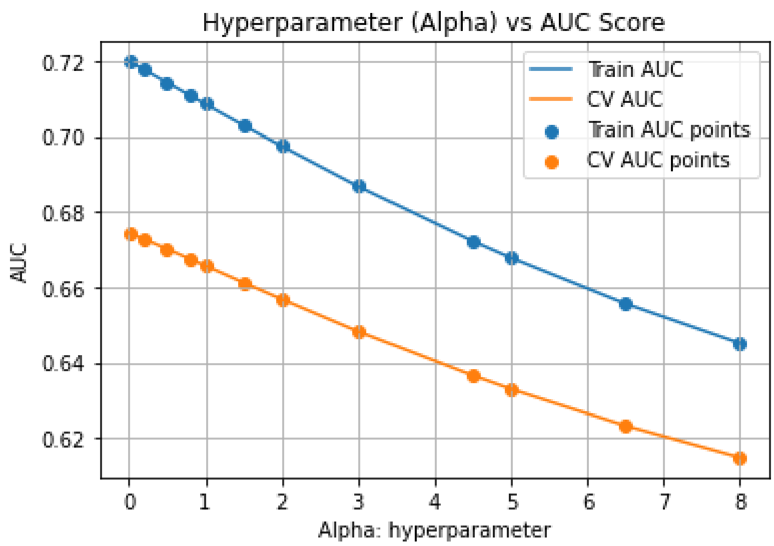
```
Final Data matrix for SET 2
(35912, 5101) (35912,)
(17688, 5101) (17688,)
(26400, 5101) (26400,)
```

5.2.1. The hyper paramter tuning (finding the best alpha : smoothing parameter)

In [25]:

```
train_auc = []
cv_auc = []
alpha = [0.01,0.2,0.5,0.8,1,1.5,2,3,4.5,5,6.5,8]

for i in tqdm(alpha):
    m_nb = MultinomialNB(alpha=i)
    m_nb.fit(X_tr_2, y_train)

    y_train_pred = batch_predict(m_nb, X_tr_2)
    y_cv_pred = batch_predict(m_nb, X_cv_2)

    # roc_auc_score(y_actual, y_prob_score)
    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("Alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("Hyperparameter (Alpha) vs AUC Score")
plt.grid()
plt.show()
```

```
100%|██████████| 12/12 [00:14<00:00,  1.21s/it]
```



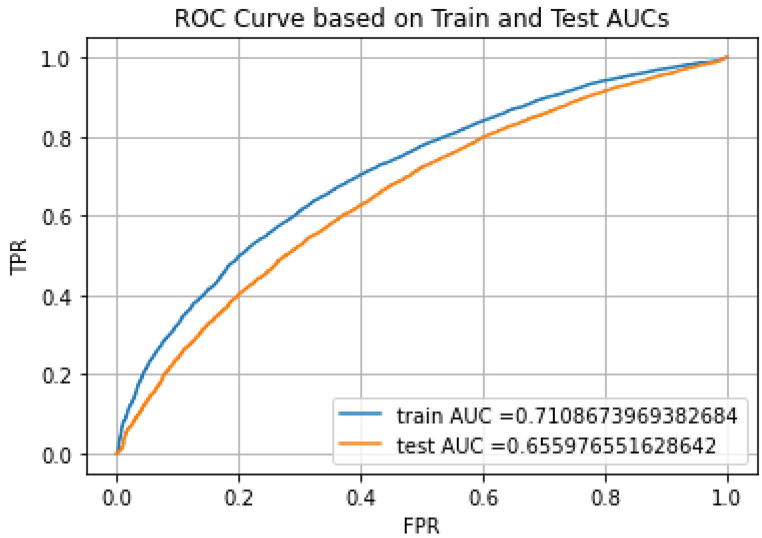
Observation:

From the plot,it looks like 0.8 is best alpha with minum difference between Train and CV AUCs

5.2.2. Finding the AUC on test data and ploting the ROC curve on both train and test

In [26]:

```
best_alpha = 0.8

m_nb = MultinomialNB(alpha = best_alpha)
m_nb.fit(X_tr_2, y_train)

y_train_pred = batch_predict(m_nb, X_tr_2)
y_test_pred = batch_predict(m_nb, X_te_2)

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

plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr
)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))

plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC Curve based on Train and Test AUCs")
plt.grid()
plt.show()
```



5.2.3. Plotting the Confusion Matrix

In [27]:

```
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)
train_cm = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
test_cm = confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t))
print("Train confusion matrix \n",train_cm)
print("Test confusion matrix \n",test_cm)
```

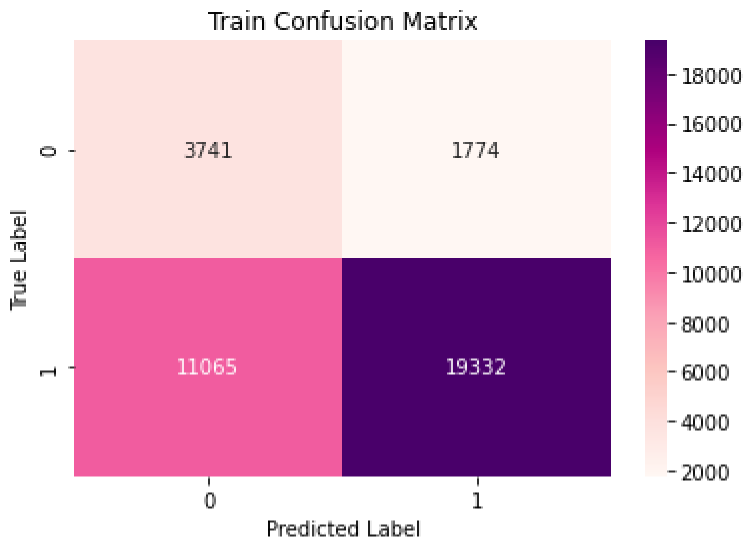
```
The maximum value of tpr*(1-fpr) 0.43140805963608025 for threshold
0.855
Train confusion matrix
 [[ 3741  1774]
 [11065 19332]]
Test confusion matrix
 [[ 2456  1598]
 [ 8487 13859]]
```

In [28]:

```
sns.heatmap(train_cm, annot=True, fmt="d",cmap="RdPu")
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Train Confusion Matrix')
```

Out[28]:

```
Text(0.5, 1.0, 'Train Confusion Matrix')
```

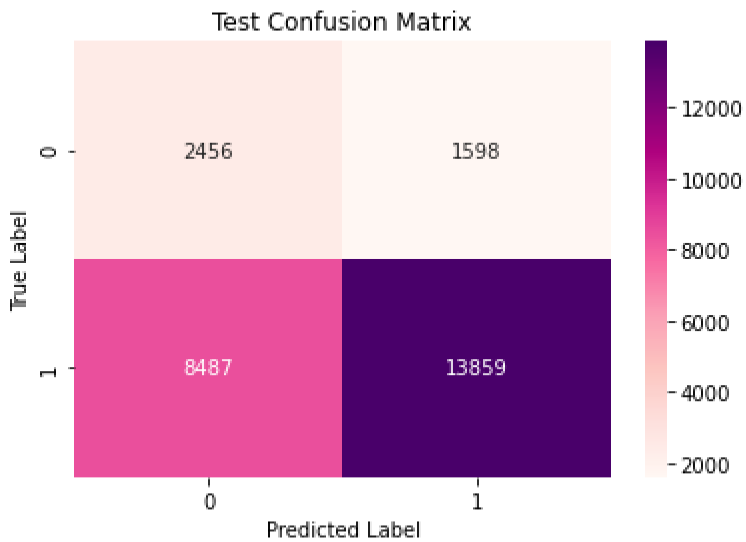


In [29]:

```
sns.heatmap(test_cm, annot=True, fmt="d",cmap="RdPu")
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Test Confusion Matrix')
```

Out[29]:

```
Text(0.5, 1.0, 'Test Confusion Matrix')
```



5.2.4. Code to find the top 20 features from Set 2

In [30]:

```
# list of all the feature names
feature_names_set2 = []

numerical = ['price','teacher_number_of_previously_posted_projects']
categorical= school_state_feature + teacher_prefix_feature + grade_feature + cat
egory_feature +subcategory_feature
text = essay_tfidf_feature

feature_names_set2.extend(numerical)
feature_names_set2.extend(categorical)
feature_names_set2.extend(text)

# to check len of feature_names equal to dimensions of final stacked matrix
print(len(feature_names_set2))
```

```
5101
```

In [31]:

```
#https://www.semicolonworld.com/question/54572/is-it-possible-to-use-argsort-in-descending-order
max_ind_pos = m_nb.feature_log_prob_[1].argsort()[::-1][0:20]
max_ind_neg = m_nb.feature_log_prob_[0].argsort()[::-1][0:20]

top_feature_pos = np.take(feature_names_set2,max_ind_pos)
top_feature_neg = np.take(feature_names_set2,max_ind_neg)

print("Top 20 features for positive class: \n\n", top_feature_pos)
print("\n Top 20 features for negative class: \n\n",top_feature_neg)
```

```
Top 20 features for positive class:

 ['price' 'teacher_number_of_previously_posted_projects' 'mrs'
 'literacy_language' 'grades_prek_2' 'math_science' 'ms' 'grades_3_
5'
 'literacy' 'mathematics' 'literature_writing' 'grades_6_8'
 'health_sports' 'ca' 'students' 'specialneeds' 'specialneeds'
 'health_wellness' 'appliedlearning' 'mr']

 Top 20 features for negative class:

 ['price' 'teacher_number_of_previously_posted_projects' 'mrs'
 'literacy_language' 'grades_prek_2' 'math_science' 'ms' 'grades_3_
5'
 'literacy' 'mathematics' 'literature_writing' 'grades_6_8'
 'health_sports' 'ca' 'specialneeds' 'specialneeds' 'students'
 'appliedlearning' 'appliedsciences' 'grades_9_12']
```

6. Summary

In [33]:

```
# summarizing the results
x = PrettyTable()
x.field_names = ["Vectorizer", "Model", "Hyperparameter", "Train AUC", "Test AU
C"]
x.add_row(["BOW","NB",1,0.73,0.69])
x.add_row(["TFIDF","NB",0.8,0.70,0.66])
print(x)
```

```
+------------+-------+----------------+-----------+----------+
| Vectorizer | Model | Hyperparameter | Train AUC | Test AUC |
+------------+-------+----------------+-----------+----------+
|    BOW     |   NB  |       1        |    0.73   |   0.69   |
|   TFIDF    |   NB  |      0.8       |    0.7    |   0.66   |
+------------+-------+----------------+-----------+----------+
```