Naive Bayes

In [0]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import pickle
```

In [102]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

In [103]:

```
! ls '/content/drive/My Drive/Applied AI/Datasets/New Donors/'
```

```
glove_vectors   Preprocessed_inc_others.csv  train_data.csv
PreProcessed.csv  resources.csv
```

Importing data

In [104]:

```
data = pd.read_csv('/content/drive/My Drive/Applied AI/Datasets/New
Donors/Preprocessed_inc_others.csv')
data.head()
```

Out[104]:

	Unnamed: 0	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved
0	0	ca	mrs	grades_prek_2	53	1
1	1	ut	ms	grades_3_5	4	1
2	2	ca	mrs	grades_prek_2	10	1
3	3	ga	mrs	grades_prek_2	2	1

4	4	wa	mrs	grades_3_5	2	1
	Unnamed: 0	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	project_is_approved

In [105]:

```
data.describe()
```

Out[105]:

	Unnamed: 0	teacher_number_of_previously_posted_projects	project_is_approved	price	quantity	sentiment_sc
count	109248.000000	109248.000000	109248.000000	109248.000000	109248.000000	109248.000
mean	54623.500000	11.153165	0.848583	298.119343	16.965610	0.210
std	31537.325441	27.777154	0.358456	367.498030	26.182942	0.083
min	0.000000	0.000000	0.000000	0.660000	1.000000	-0.189
25%	27311.750000	0.000000	1.000000	104.310000	4.000000	0.154
50%	54623.500000	2.000000	1.000000	206.220000	9.000000	0.208
75%	81935.250000	9.000000	1.000000	379.000000	21.000000	0.264
max	109247.000000	451.000000	1.000000	9999.000000	930.000000	0.663

In [106]:

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

Out[106]:

	Unnamed: 0	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories	cl
0	0	ca	mrs	grades_prek_2	53	math_science	

In [0]:

```
y = y.reshape(-1,1)
```

In [108]:

```
print(X.shape)
print(y.shape)
```

```
(109248, 14)
(109248, 1)
```

Splitting the data

In [0]:

```
from sklearn.model_selection import train_test_split
data_train, data_test, label_train, label_test = train_test_split(X, y, test_size=0.33, stratify=y,
random_state=42)
```

In [110]:

```
print(data_train.shape)
print(data_test.shape)
print(label_train.shape)
print(label_test.shape)
```

```
(73196, 14)
(36052, 14)
(73196, 1)
(36052, 1)
```

In [0]:

```
X_train = data_train
X_test = data_test
y_train = label_train
y_test = label_test
```

In [112]:

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(73196, 14)
(36052, 14)
(73196, 1)
(36052, 1)
```

1. Vectorizing all the features

1.1 School State

In [0]:

```
from sklearn.feature_extraction.text import CountVectorizer
vectorizer_1 = CountVectorizer(list(X_train['school_state'].values), lowercase=False, binary=True)
```

In [0]:

```
X_train_Sstate = vectorizer_1.fit_transform(X_train['school_state'].values)
X_test_Sstate = vectorizer_1.transform(X_test['school_state'].values)
```

In [115]:

```
print(X_train_Sstate.shape)
print(X_test_Sstate.shape)
```

```
(73196, 51)
(36052, 51)
```

1.2 Clean_Categories

In [116]:

```
X_train.head(1)
```

Out[116]:

	Unnamed: 0	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories

	Unnamed: 0	school_state	teacher_prefix	project_grade_category	teacher_number_of_previously_posted_projects	clean_categories
76564	76564	ks	mrs	grades_prek_2	13	literacy_language math_science

In [0]:

```
vectorizer_2 = CountVectorizer(list(X_train['clean_categories'].values),lowercase=False,
binary=True)
```

In [0]:

```
X_train_cat = vectorizer_2.fit_transform(X_train['clean_categories'].values)
X_test_cat = vectorizer_2.transform(X_test['clean_categories'].values)
```

In [119]:

```
print(X_train_cat.shape)
print(X_test_cat.shape)
```

```
(73196, 9)
(36052, 9)
```

1.3 Clean sub_categories

In [0]:

```
vectorizer_3 = CountVectorizer(list(X_train['clean_subcategories'].values), lowercase=False,
binary=True)
```

In [0]:

```
X_train_subcat = vectorizer_3.fit_transform(X_train['clean_subcategories'].values)
X_test_subcat = vectorizer_3.transform(X_test['clean_subcategories'].values)
```

In [122]:

```
print(X_train_subcat.shape)
print(X_test_subcat.shape)
```

```
(73196, 30)
(36052, 30)
```

1.4 Project Grade Category

In [0]:

```
vectorizer_4 = CountVectorizer(list(X_train['project_grade_category'].values), lowercase=False,
binary=True)
```

In [0]:

```
X_train_grade = vectorizer_4.fit_transform(X_train['project_grade_category'].values)
X_test_grade = vectorizer_4.transform(X_test['project_grade_category'].values)
```

In [125]:

```
print(X_train_grade.shape)
print(X_test_grade.shape)
```

```
(73196, 4)
(36052, 4)
```

(36052, 4)

1.5 Teacher Prefix

In [0]:

```
vectorizer_5 = CountVectorizer(list(X_train['teacher_prefix'].values), lowercase=False,
binary=True)
```

In [0]:

```
X_train_prefix = vectorizer_5.fit_transform(X_train['teacher_prefix'].values)
X_test_prefix = vectorizer_5.transform(X_test['teacher_prefix'].values)
```

In [128]:

```
print(X_train_prefix.shape)
print(X_test_prefix.shape)
```

```
(73196, 5)
(36052, 5)
```

1.6 Essay

1.6.1 BOW

In [0]:

```
vectorizer_6 = CountVectorizer(list(X_train['essay'].values), min_df=10)
```

In [0]:

```
#We are considering the words which atleast in atleat 10 documents
X_train_essay_bow = vectorizer_6.fit_transform(X_train['essay'].values)
X_test_essay_bow = vectorizer_6.transform(X_test['essay'].values)
```

In [131]:

```
print(X_train_essay_bow.shape)
print(X_test_essay_bow.shape)
```

```
(73196, 14266)
(36052, 14266)
```

1.6.2 TFIDF

In [0]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_7 = TfidfVectorizer(list(X_train['essay'].values), min_df=10)
```

In [0]:

```
X_train_essay_tfidf = vectorizer_7.fit_transform(X_train['essay'].values)
X_test_essay_tfidf = vectorizer_7.transform(X_test['essay'].values)
```

In [134]:

```
print(X_train_essay_tfidf.shape)
print(X_test_essay_tfidf.shape)
```

```
(73196, 14266)
```

(73196, 14266)
(36052, 14266)

1.7 PROJECT TITLE

1.7.1 BOW

In [0]:

```
vectorizer_8 = CountVectorizer(list(X_train['title'].values), min_df=10)
```

In [0]:

```
X_train_title_bow = vectorizer_8.fit_transform(X_train['title'].values)
X_test_title_bow = vectorizer_8.transform(X_test['title'].values)
```

In [137]:

```
print(X_train_title_bow.shape)
print(X_test_title_bow.shape)
```

```
(73196, 2617)
(36052, 2617)
```

1.7.2 TFIDF

In [0]:

```
vectorizer_9 = TfidfVectorizer(list(X_train['title'].values), min_df=10)
```

In [0]:

```
X_train_title_tfidf = vectorizer_9.fit_transform(X_train['title'].values)
X_test_title_tfidf = vectorizer_9.transform(X_test['title'].values)
```

In [140]:

```
print(X_train_title_tfidf.shape)
print(X_test_title_tfidf.shape)
```

```
(73196, 2617)
(36052, 2617)
```

1.8 Price

Note :

- Since the Naive Bayes will not process with negative values and we can't standardize it

In [0]:

```
X_train_price = X_train['price'].values.reshape(-1,1)
X_test_price = X_test['price'].values.reshape(-1,1)
```

In [142]:

```
print(X_train_price.shape)
print(X_test_price.shape)
```

```
(73196, 1)
(36052, 1)
```

```
(36052, 1)
```

1.9 Previously posted projects

In [0]:

```
X_train_previous = X_train['teacher_number_of_previously_posted_projects'].values.reshape(-1,1)
X_test_previous = X_test['teacher_number_of_previously_posted_projects'].values.reshape(-1,1)
```

In [144]:

```
print(X_train_previous.shape)
print(X_test_previous.shape)
```

```
(73196, 1)
(36052, 1)
```

2. SET-1

2.1 Merge all the features

In [43]:

```
print(X_train_Sstate.shape)
print(X_train_cat.shape)
print(X_train_subcat.shape)
print(X_train_grade.shape)
print(X_train_prefix.shape)
print(X_train_essay_bow.shape)
print(X_train_essay_tfidf.shape)
print(X_train_title_bow.shape)
print(X_train_title_tfidf.shape)
print(X_train_price.shape)
print(X_train_previous.shape)
print('='*50)
print(X_test_Sstate.shape)
print(X_test_cat.shape)
print(X_test_subcat.shape)
print(X_test_grade.shape)
print(X_test_prefix.shape)
print(X_test_essay_bow.shape)
print(X_test_essay_tfidf.shape)
print(X_test_title_bow.shape)
print(X_test_title_tfidf.shape)
print(X_test_price.shape)
print(X_test_previous.shape)
```

```
(73196, 51)
(73196, 9)
(73196, 30)
(73196, 4)
(73196, 5)
(73196, 14266)
(73196, 14266)
(73196, 2617)
(73196, 2617)
(73196, 1)
(73196, 1)
==================================================
(36052, 51)
(36052, 9)
(36052, 30)
(36052, 4)
(36052, 5)
(36052, 14266)
(36052, 14266)
(36052, 2617)
(36052, 2617)
(36052, 1)
```

```
(36052, 1)
```

In [0]:

```
from scipy.sparse import hstack
X_train_1 = hstack((X_train_Sstate, X_train_cat, X_train_subcat, X_train_grade, X_train_prefix, X_t
rain_essay_bow, X_train_title_bow, X_train_previous, X_train_price)).tocsr()
X_test_1 = hstack((X_test_Sstate, X_test_cat, X_test_subcat, X_test_grade, X_test_prefix, X_test_es
say_bow, X_test_title_bow, X_test_previous, X_test_price)).tocsr()
```

In [57]:

```
print(X_train_1.shape)
print(X_test_1.shape)
```

```
(73196, 16984)
(36052, 16984)
```

2.2 Grid Search CV

In [0]:

```
from sklearn.naive_bayes import MultinomialNB
classifier_1 = MultinomialNB()
```

In [0]:

```
from sklearn.model_selection import GridSearchCV
parameters = [
                {
                    'alpha' : [0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
                }
             ]
gridsearch_1 = GridSearchCV(classifier_1, parameters, scoring='roc_auc', cv=10, n_jobs=-1, return_t
rain_score=True)
```

In [60]:

```
print(X_train_1.shape)
print(y_train.shape)
```

```
(73196, 16984)
(73196, 1)
```

In [61]:

```
gridsearch_1 = gridsearch_1.fit(X_train_1, y_train)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/utils/validation.py:760: DataConversionWarning: A c
olumn-vector y was passed when a 1d array was expected. Please change the shape of y to
(n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

In [62]:

```
import warnings
warnings.filterwarnings('ignore')
results = pd.DataFrame.from_dict(gridsearch_1.cv_results_)
results.head()
```

Out[62]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score	split1_test_score	split2_test_
0	0.137218	0.014056	0.014372	0.001395	0.0001	{'alpha': 0.0001}	0.650117	0.655750	0.6

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score	split1_test_score	split2_test_
1	0.123135	0.004729	0.012991	0.000809	0.001	{'alpha': 0.001}	0.650303	0.656382	0.6
2	0.118874	0.005027	0.013573	0.001144	0.01	{'alpha': 0.01}	0.650659	0.656824	0.6
3	0.113551	0.004523	0.012937	0.000709	0.1	{'alpha': 0.1}	0.650738	0.656396	0.6
4	0.111400	0.001876	0.015732	0.003122	1	{'alpha': 1}	0.651656	0.657538	0.6

In [63]:

```
best_alpha_1 = gridsearch_1.best_params_['alpha']
print('best_alhpa:', best_alpha_1)
```

```
best_alhpa: 10
```

Summary:

    - It shows that the best alpha = 10

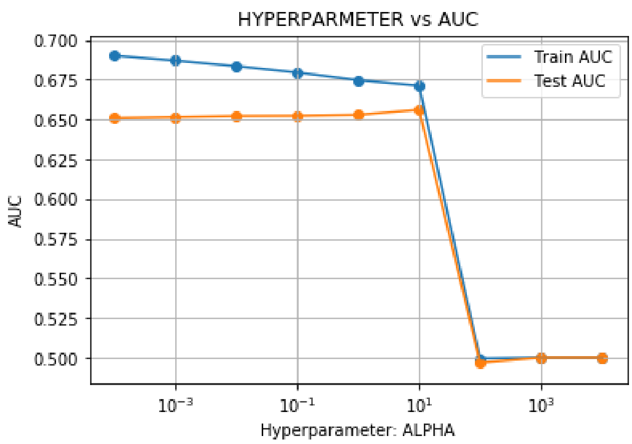
2.3 AUC vs Hyperparameter

In [0]:

```
train_auc_1 = results['mean_train_score']
test_auc_1 = results['mean_test_score']
alpha_1 = results['param_alpha']
```

In [65]:

```
plt.plot(alpha_1, train_auc_1, label='Train AUC')
plt.plot(alpha_1, test_auc_1, label='Test AUC')
plt.scatter(alpha_1, train_auc_1)
plt.scatter(alpha_1, test_auc_1)

plt.title('HYPERPARMETER vs AUC')
plt.xlabel('Hyperparameter: ALPHA')
plt.ylabel('AUC')
plt.xscale('log')
plt.legend()
plt.grid()
plt.show()
```



2.4 Modelling with alpha = 10

In [66]:

```
classifier_withAlpha_1 = MultinomialNB(alpha=10)
```

```
classifier_withAlpha_1.fit(X_train_1, y_train)
```

Out[66]:

```
MultinomialNB(alpha=10, class_prior=None, fit_prior=True)
```

2.5 Cross Validation

In [0]:

```
from sklearn.model_selection import cross_val_score
cv_1 = cross_val_score(estimator=classifier_withAlpha_1, X=X_train_1, y=y_train,cv=10,
scoring='roc_auc')
```

In [165]:

```
best_auc_1 = cv_1.mean()
print('Best AUC:%4f' %best_auc_1)
```

```
Best AUC:0.656119
```

2.6 AUC curve for Train and Test

In [0]:

```
def batch_predict(clf, data):
    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    for i in range(0,tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    if data.shape[0]%1000 != 0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [0]:

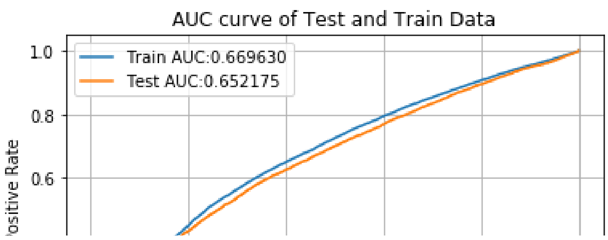
```
y_train_pred_1 = batch_predict(classifier_withAlpha_1, X_train_1)
y_test_pred_1 = batch_predict(classifier_withAlpha_1, X_test_1)
```

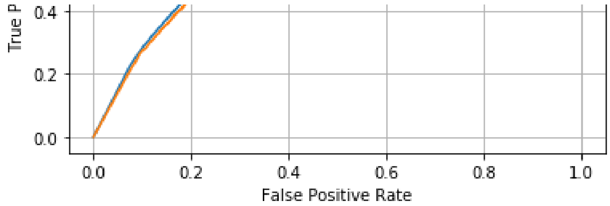
In [0]:

```
from sklearn.metrics import roc_curve, auc
train_fpr_1, train_tpr_1, train_thresh_1 = roc_curve(y_train, y_train_pred_1)
test_fpr_1, test_tpr_1, test_thresh_1 = roc_curve(y_test, y_test_pred_1)
```

In [72]:

```
plt.plot(train_fpr_1, train_tpr_1, label='Train AUC:%4f'%auc(train_fpr_1, train_tpr_1))
plt.plot(test_fpr_1, test_tpr_1, label='Test AUC:%4f'%auc(test_fpr_1, test_tpr_1))

plt.title('AUC curve of Test and Train Data')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.grid()
plt.show()
```





2.7 Confusion Matrix

In [0]:

```
def find_best_threshold(fpr, tpr, threhsold):
    t = threhsold[np.argmax(tpr*(1-fpr))]
    print('The maximum tpr*(1-fpr) is:', max(tpr*(1-fpr)), 'for threshold', (np.round(t,3)))
    return t
```

In [74]:

```
best_t = find_best_threshold(train_fpr_1, train_tpr_1, train_thresh_1)
```

```
The maximum tpr*(1-fpr) is: 0.3970900910561266 for threshold 1.0
```

In [0]:

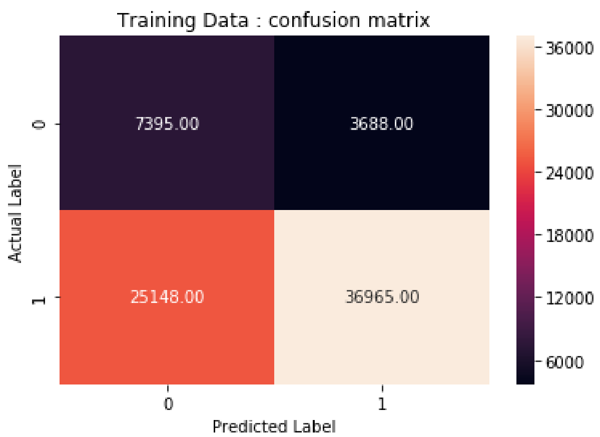
```
def predict_with_threshold(proba, threshold):
    predictions = []
    for i in proba:
        if i>=threshold:
            predictions.append(1)
        else:
            predictions.append(0)

    return predictions
```

In [0]:

```
from sklearn.metrics import confusion_matrix
cm_train_1 = confusion_matrix(y_train, predict_with_threshold(y_train_pred_1, best_t))
cm_test_1 = confusion_matrix(y_test, predict_with_threshold(y_test_pred_1, best_t))
```

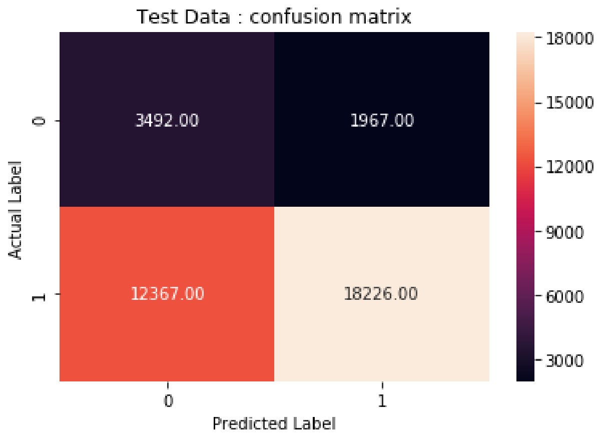
In [77]:

```
sns.heatmap(cm_train_1, annot=True, fmt='.2f')
plt.title('Training Data : confusion matrix')
plt.xlabel('Predicted Label')
plt.ylabel('Actual Label')
plt.show()
```



In [78]:

```
sns.heatmap(cm_test_1, annot=True, fmt='.2f')
plt.title('Test Data : confusion matrix')
plt.xlabel('Predicted Label')
plt.ylabel('Actual Label')
plt.show()
```



2. 8 Top 20 Features

In [0]:

```
#https://datascience.stackexchange.com/questions/65219/find-the-top-n-features-from-feature-set-us
ing-absolute-values-of-feature-log-p/65232#65232
features_list = list( vectorizer_1.get_feature_names() +vectorizer_2.get_feature_names()
+vectorizer_3.get_feature_names() +\
                    vectorizer_4.get_feature_names() +vectorizer_5.get_feature_names()
+vectorizer_6.get_feature_names() +\
                     vectorizer_8.get_feature_names() +
["teacher_number_of_previously_posted_projects"] + ["Price"]
                    )
```

In [80]:

```
len(features_list)
```

Out[80]:

```
16984
```

In [0]:

```
#For postive class , we get the indices
sorted_prob_class1_ind = classifier_withAlpha_1.feature_log_prob_[1,:].argsort()

#For Negative Class
sorted_prob_class0_ind = classifier_withAlpha_1.feature_log_prob_[0, :].argsort()
```

In [82]:

```
sorted_prob_class0_ind[-20:-1]
```

Out[82]:

```
array([10356,  7712,  2656, 14215,  8520, 13980,  7876,  8463, 12863,
        6137, 12901,  7438,  8662,  2494,  8439,  7442, 11231, 16982,
       12371])
```

In [0]:

```
#https://www.geeksforgeeks.org/python-get-last-n-elements-from-given-list/
#since argsort gives it in ascending order we need the last 20 eleemnts
Most_important_word_for_1 = []
Most_important_word_for_0 = []
```

```
Most_important_word_for_0 = []

for i in (sorted_prob_class1_ind[-20:-1]):
    Most_important_word_for_1.append(features_list[i])

for j in (sorted_prob_class0_ind[-20:-1]):
    Most_important_word_for_0.append(features_list[j])
```

In [84]:

```
print('Top 20 words in Postive Class')
print(Most_important_word_for_1)
print('='*50)
print('Top 20 words in Negative class')
print(Most_important_word_for_0)
```

```
Top 20 words in Postive Class
['love', 'use', 'reading', 'work', 'need', 'we', 'nannan', 'many', 'help', 'learn', 'not', 'they',
'the', 'classroom', 'learning', 'my', 'school', 'students',
'teacher_number_of_previously_posted_projects']
==================================================
Top 20 words in Negative class
['reading', 'love', 'come', 'work', 'need', 'we', 'many', 'nannan', 'the', 'help', 'they',
'learn', 'not', 'classroom', 'my', 'learning', 'school',
'teacher_number_of_previously_posted_projects', 'students']
```

3. Set 2

3.1 Merge all the features

In [0]:

```
from scipy.sparse import hstack
X_train_2 = hstack((X_train_Sstate, X_train_cat, X_train_subcat, X_train_grade, X_train_prefix, X_t
rain_essay_tfidf, X_train_title_tfidf, X_train_previous, X_train_price)).tocsr()
X_test_2 = hstack((X_test_Sstate, X_test_cat, X_test_subcat, X_test_grade, X_test_prefix, X_test_es
say_tfidf, X_test_title_tfidf, X_test_previous, X_test_price)).tocsr()
```

In [146]:

```
print(X_train_2.shape)
print(X_test_2.shape)
```

```
(73196, 16984)
(36052, 16984)
```

3.2 Grid Search

In [0]:

```
classifier_2 = MultinomialNB()
```

In [0]:

```
parameters = [
                {
                    'alpha' : [0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
                }
            ]
gridsearch_2 = GridSearchCV(estimator=classifier_2, param_grid=parameters, scoring='roc_auc',
n_jobs=-1, cv=10, return_train_score=True)
```

In [0]:

```
gridsearch_2 = gridsearch_2.fit(X_train_2, y_train)
```

In [91]:

```
results = pd.DataFrame.from_dict(gridsearch_2.cv_results_)
results = results.sort_values(['param_alpha'])
results.head()
```

Out[91]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score	split1_test_score	split2_test_
0	0.122821	0.004070	0.013092	0.000593	0.0001	{'alpha': 0.0001}	0.613831	0.622304	0.6
1	0.127190	0.004452	0.013374	0.000319	0.001	{'alpha': 0.001}	0.614316	0.622410	0.6
2	0.123700	0.005153	0.013480	0.000411	0.01	{'alpha': 0.01}	0.614795	0.622272	0.6
3	0.129093	0.011464	0.013330	0.000664	0.1	{'alpha': 0.1}	0.615036	0.621804	0.6
4	0.122390	0.002708	0.014902	0.002513	1	{'alpha': 1}	0.614182	0.620320	0.6

In [92]:

```
best_alpha_2 = gridsearch_2.best_params_['alpha']
print('best_alhpa:', best_alpha_2)
```

```
best_alhpa: 0.01
```

Summary:

    - It shows that the best alpha could be 0.01

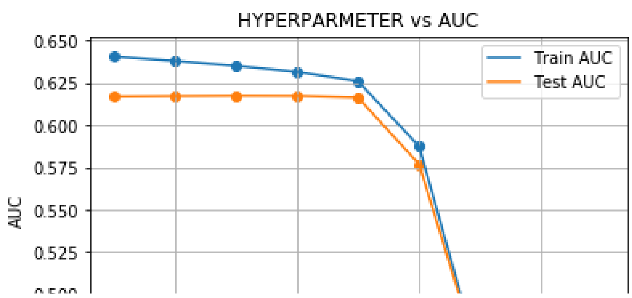
3.3 Plotting AUC vs Hyperparameter

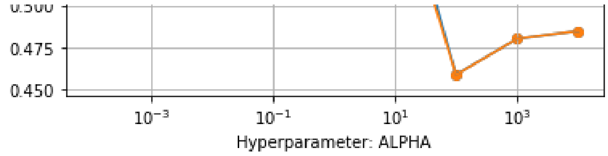
In [0]:

```
train_auc_2 = results['mean_train_score']
test_auc_2 = results['mean_test_score']
alpha_2 = results['param_alpha']
```

In [96]:

```
plt.plot(alpha_2, train_auc_2, label='Train AUC')
plt.plot(alpha_2, test_auc_2, label='Test AUC')
plt.scatter(alpha_2, train_auc_2)
plt.scatter(alpha_2, test_auc_2)

plt.title('HYPERPARMETER vs AUC')
plt.xlabel('Hyperparameter: ALPHA')
plt.ylabel('AUC')
plt.xscale('log')
plt.legend()
plt.grid()
plt.show()
```





Summary

```
- Here also it shows that the AUC is high when alpha = 1
```

3.4 Modelling with Parameters

In [147]:

```python
classifier_withAlpha_2 = MultinomialNB(alpha=best_alpha_2)
classifier_withAlpha_2.fit(X_train_2, y_train)
```

Out[147]:

```
MultinomialNB(alpha=0.01, class_prior=None, fit_prior=True)
```

3.5 Cross Validation

In [0]:

```python
from sklearn.model_selection import cross_val_score
cv_2 = cross_val_score(estimator=classifier_withAlpha_2, X=X_train_2, y=y_train,cv=10,
scoring='roc_auc')
```

In [149]:

```python
best_auc_2 = cv_2.mean()
print('Best AUC:%4f' %best_auc_2)
```

```
Best AUC:0.617436
```

3.6 Plotting ROC curve

In [0]:

```python
def batch_predict(clf, data):
    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    for i in range(0,tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    if data.shape[0]%1000 != 0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred
```

In [0]:

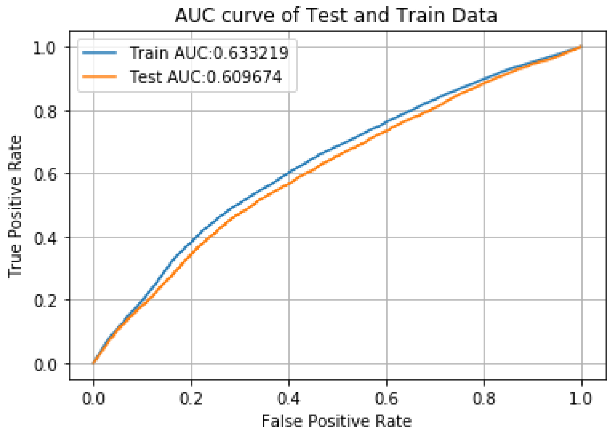
```python
y_train_pred_2 = batch_predict(classifier_withAlpha_2, X_train_2)
y_test_pred_2 = batch_predict(classifier_withAlpha_2, X_test_2)
```

In [0]:

```python
from sklearn.metrics import roc_curve, auc
train_fpr_2, train_tpr_2, train_thresh_2 = roc_curve(y_train, y_train_pred_2)
test_fpr_2, test_tpr_2, test_thresh_2 = roc_curve(y_test, y_test_pred_2)
```

In [153]:

```
plt.plot(train_fpr_2, train_tpr_2, label='Train AUC:%4f'%auc(train_fpr_2, train_tpr_2))
plt.plot(test_fpr_2, test_tpr_2, label='Test AUC:%4f'%auc(test_fpr_2, test_tpr_2))

plt.title('AUC curve of Test and Train Data')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.grid()
plt.show()
```



3.7 Confusion Matrix

In [154]:

```
def find_best_threshold(fpr, tpr, threhsold):
    t = threhsold[np.argmax(tpr*(1-fpr))]
    print('The maximum tpr*(1-fpr) is:', max(tpr*(1-fpr)), 'for threshold', (np.round(t,3)))
    return t


best_t = find_best_threshold(train_fpr_2, train_tpr_2, train_thresh_2)
```

```
The maximum tpr*(1-fpr) is: 0.3606844925473015 for threshold 0.685
```

In [0]:

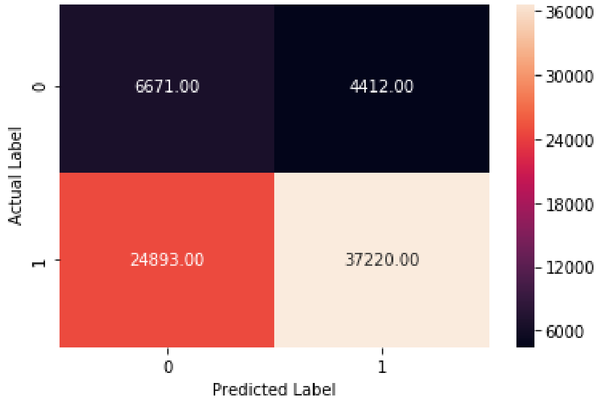
```
def predict_with_threshold(proba, threshold):
    predictions = []
    for i in proba:
        if i>=threshold:
            predictions.append(1)
        else:
            predictions.append(0)

    return predictions
```

In [0]:

```
from sklearn.metrics import confusion_matrix
cm_train_2 = confusion_matrix(y_train, predict_with_threshold(y_train_pred_2, best_t))
cm_test_2 = confusion_matrix(y_test, predict_with_threshold(y_test_pred_2, best_t))
```

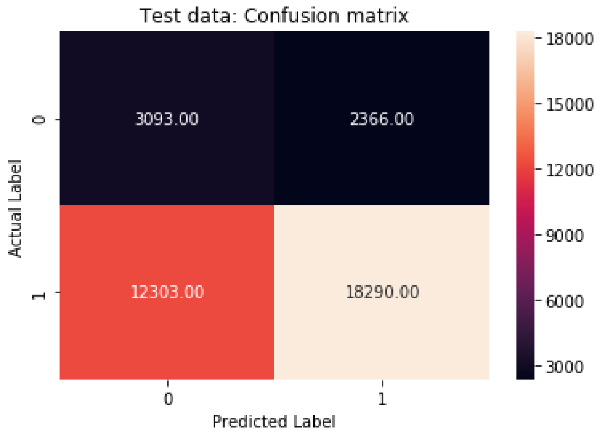
In [157]:

```
sns.heatmap(cm_train_2, annot=True, fmt='.2f')
plt.title('Training Data : confusion matrix')
plt.xlabel('Predicted Label')
plt.ylabel('Actual Label')
plt.show()
```



In [158]:

```
sns.heatmap(cm_test_2, annot=True, fmt='.2f')
plt.title('Test data: Confusion matrix')
plt.xlabel('Predicted Label')
plt.ylabel('Actual Label')
plt.show()
```



3.8 Top 20 Features

In [0]:

```
#https://datascience.stackexchange.com/questions/65219/find-the-top-n-features-from-feature-set-us
ing-absolute-values-of-feature-log-p/65232#65232
features_list = list( vectorizer_1.get_feature_names() +vectorizer_2.get_feature_names()
+vectorizer_3.get_feature_names() +\
                    vectorizer_4.get_feature_names() +vectorizer_5.get_feature_names()
+vectorizer_7.get_feature_names() +\
                     vectorizer_9.get_feature_names() +
["teacher_number_of_previously_posted_projects"] + ["Price"]
                    )
```

In [0]:

```
#For postive class , we get the indices
sorted_prob_class1_ind_2 = classifier_withAlpha_2.feature_log_prob_[1,:].argsort()

#For Negative Class
sorted_prob_class0_ind_2 = classifier_withAlpha_2.feature_log_prob_[0, :].argsort()
```

In [0]:

```
#https://www.geeksforgeeks.org/python-get-last-n-elements-from-given-list/
#since argsort gives it in ascending order we need the last 20 eleemnts
Most_important_word_for_postive_2 = []
Most_important_word_for_neagtive_2 = []

for i in (sorted_prob_class1_ind_2[-20:-1]):
```

```
    Most_important_word_for_postive_2.append(features_list[i])

for j in (sorted_prob_class0_ind_2[-20:-1]):
    Most_important_word_for_neagtive_2.append(features_list[j])
```

In [162]:

```
print('Top 20 words in Postive Class')
print(Most_important_word_for_1)
print('='*50)
print('Top 20 words in Negative class')
print(Most_important_word_for_0)
```

```
Top 20 words in Postive Class
['love', 'use', 'reading', 'work', 'need', 'we', 'nannan', 'many', 'help', 'learn', 'not', 'they',
'the', 'classroom', 'learning', 'my', 'school', 'students',
'teacher_number_of_previously_posted_projects']
==================================================
Top 20 words in Negative class
['reading', 'love', 'come', 'work', 'need', 'we', 'many', 'nannan', 'the', 'help', 'they',
'learn', 'not', 'classroom', 'my', 'learning', 'school',
'teacher_number_of_previously_posted_projects', 'students']
```

Summary

In [168]:

```
#http://zetcode.com/python/prettytable/
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ['Vectorizer', 'Model', 'Alpha', 'AUC']
x.add_row(['BOW', 'Naive Bayes', str(best_alpha_1), str('%4f'%best_auc_1)])
x.add_row(['TFIDF', 'Naive Bayes', str(best_alpha_2), str('%4f'%best_auc_2)])
print(x)
```

```
+------------+-------------+-------+----------+
| Vectorizer |    Model    | Alpha |   AUC    |
+------------+-------------+-------+----------+
|    BOW     | Naive Bayes |   10  | 0.656119 |
|   TFIDF    | Naive Bayes |  0.01 | 0.617436 |
+------------+-------------+-------+----------+
```

Note : That's the end of the code