Bayesian Gaussian Mixture Models / PCA - Unsupervised

```python
In [1]:  from sklearn import datasets
         from sklearn import preprocessing
         from sklearn.model_selection import train_test_split
         from sklearn.decomposition import PCA

         from sklearn import mixture
         from sklearn.mixture import GaussianMixture
         from sklearn.cluster import KMeans
         from sklearn.ensemble import IsolationForest
         from sklearn.neighbors import LocalOutlierFactor
         from sklearn import svm, neighbors
         from sklearn.neighbors import NearestNeighbors

         from sklearn.metrics import classification_report
         from sklearn.metrics import confusion_matrix
         from sklearn.metrics import recall_score
         from sklearn.metrics import roc_auc_score
         from sklearn.model_selection import GridSearchCV
         from sklearn.metrics import make_scorer
         from sklearn.metrics import accuracy_score

         import pandas as pd
         import numpy as np
         import itertools
         import matplotlib.pyplot as plt
         import datetime

         %matplotlib inline
```

```python
In [2]:  train_split = 0.80
         nrows = 250_000
         path = 'c:/users/ugy1/abs/'
         df=pd.read_csv(path+'datasets/processed_abs_loan_'+str(nrows)+'.csv',
                     #usecols=use_list,
                     #sep='\t',
                     #compression=bz2,
                      nrows=nrows,
                      low_memory=False,
                     index_col=0,
                      parse_dates=True
                      )
         df.shape
```

```
Out[2]:  (237024, 58)
```

```python
In [3]:  column_list=df.columns.tolist()
```
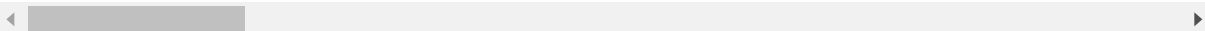
In [4]:
```
df.label.value_counts()
```

Out[4]:
```
0      229634
1        7390
Name: label, dtype: int64
```

In [5]:
```
df.head()
```

Out[5]:

	originalloanamount	originalloanterm	originalinterestratepercentage	graceperiodnumb
0	66711.84	60	3.29	1
1	16258.45	60	0.90	0
2	31930.41	72	2.90	1
3	26065.02	65	0.90	0
4	42091.00	72	3.90	0

5 rows × 58 columns

In [6]:
```
# prepare label for scikit-learn
Y=df.label.values
Y.shape
```

Out[6]: (237024,)

In [7]:
```
# prepare input data for scikit-learn
input=df.values
input.shape
```

Out[7]: (237024, 58)

In [8]:
```
# calculate train/test split

len_train = int(len(input)*train_split)
print(len_train)
```

189619

In [9]:
```
# apply train/test split to labels
y_train = Y[0:len_train]
y_test = Y[len_train:]
x_train = input[0:len_train]
x_test = input[len_train:]
x_train.shape
```

Out[9]: (189619, 58)

In [10]:
```
export_x_test = pd.DataFrame(data=x_test)
```

In [11]:
```
export_x_test.columns=column_list
export_x_test.rename(columns={'label':'True Label'}, inplace=True)
export_x_test.head()
```

Out[11]:

	originalloanamount	originalloanterm	originalinterestratepercentage	graceperiodnumb
0	36863.24	72.0	1.00	1.0
1	23811.32	60.0	1.90	0.0
2	30669.00	48.0	1.00	1.0
3	54083.21	72.0	1.00	0.0
4	31557.75	72.0	3.89	1.0

5 rows × 58 columns
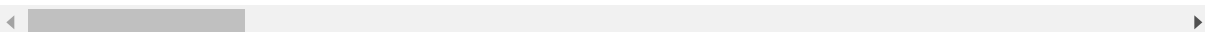
In [12]:
```
#from sklearn.preprocessing import MinMaxScaler
# from sklearn.preprocessing import minmax_scale
# from sklearn.preprocessing import MaxAbsScaler
from sklearn.preprocessing import StandardScaler
# from sklearn.preprocessing import RobustScaler
# from sklearn.preprocessing import Normalizer
# from sklearn.preprocessing import QuantileTransformer
# from sklearn.preprocessing import PowerTransformer
```

In [13]:
```
x_scaler=StandardScaler()
x_train = x_scaler.fit_transform(x_train)
x_test = x_scaler.fit_transform(x_test)
```

In [14]:
```
clf_bgmm = mixture.BayesianGaussianMixture(n_components=2,
                                           covariance_type='full').fit(x_test)
```

In [15]:
```
x_pred=x_test
```

In [16]:
```
prediction_bgmm = clf_bgmm.predict(x_pred)
```

In [17]:
```
np.unique(prediction_bgmm)
```

Out[17]: array([0, 1], dtype=int64)

In [18]:
```
np.bincount(np.array(prediction_bgmm).reshape(1,prediction_bgmm.size)[0])
```
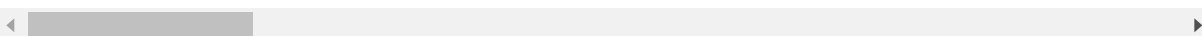
Out[18]: array([26200, 21205], dtype=int64)

In [19]:
```
export_x_test['Predicted Label']=prediction_bgmm
```

In [20]:
```
export_x_test.head()
```

Out[20]:

	originalloanamount	originalloanterm	originalinterestratepercentage	graceperiodnumk
0	36863.24	72.0	1.00	1.0
1	23811.32	60.0	1.90	0.0
2	30669.00	48.0	1.00	1.0
3	54083.21	72.0	1.00	0.0
4	31557.75	72.0	3.89	1.0

5 rows × 59 columns

◄   ►

In [21]:
```
export_x_test.shape
```

Out[21]: (47405, 59)

In [22]:
```
export_x_test.to_csv(path+"prediction/bgmm/predicated_bgmm_abs_loans_"+str(nrows)+".csv", chunksize=10000)
```

In [23]:
```
def plot_confusion_matrix(cm, title, classes=['Non-Current', 'Current'],
                          cmap=plt.cm.Blues, save=False, saveas="MyFigure.png"
):

    # print Confusion matrix with blue gradient colours

    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=90)
    plt.yticks(tick_marks, classes)

    fmt = '.1%'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

    if save:
        plt.savefig(saveas, dpi=100)
```

In [24]:
```python
def plot_gridsearch_cv(results, estimator, x_min, x_max, y_min, y_max,save=F
alse, saveas="MyFigure.png"):

    # print GridSearch cross-validation for parameters

    plt.figure(figsize=(10,8))
    plt.title("GridSearchCV for "+estimator, fontsize=24)

    plt.xlabel(estimator)
    plt.ylabel("Score")
    plt.grid()

    ax = plt.axes()
    ax.set_xlim(x_min, x_max)
    ax.set_ylim(y_min, y_max)

    pad = 0.005
    X_axis = np.array(results["param_"+estimator].data, dtype=float)

    for scorer, color in zip(sorted(scoring), ['b', 'k']):
        for sample, style in (('train', '--'), ('test', '-')):
            sample_score_mean = results['mean_%s_%s' % (sample, scorer)]
            sample_score_std = results['std_%s_%s' % (sample, scorer)]
            ax.fill_between(X_axis, sample_score_mean - sample_score_std,
                        sample_score_mean + sample_score_std,
                        alpha=0.1 if sample == 'test' else 0, color=color)
            ax.plot(X_axis, sample_score_mean, style, color=color,
                alpha=1 if sample == 'test' else 0.7,
                label="%s (%s)" % (scorer, sample))

        best_index = np.nonzero(results['rank_test_%s' % scorer] == 1)[0][0]
        best_score = results['mean_test_%s' % scorer][best_index]

        # Plot a dotted vertical line at the best score for that scorer mark
ed by x
        ax.plot([X_axis[best_index], ] * 2, [0, best_score],
            linestyle='-.', color=color, marker='x', markeredgewidth=3, ms=8
)

        # Annotate the best score for that scorer
        ax.annotate("%0.2f" % best_score,
                (X_axis[best_index], best_score+pad))

    plt.legend(loc="best")
    plt.grid('off')
    plt.tight_layout()
    if save:
        plt.savefig(saveas, dpi=100)

    plt.show()
```

In [25]:
```python
print(classification_report(y_test, prediction_bgmm, target_names=['Non-Current', 'Current']))
print ("AUC: ", "{:.1%}".format(roc_auc_score(y_test, prediction_bgmm)))
cm = confusion_matrix(y_test, prediction_bgmm)
plot_confusion_matrix(cm, title="Bayesian GMM Confusion Matrix",save=True,
                      saveas='prediction/bgmm/cm'+str(' Bayesian GMM Accuracy-')+str(nrows)+'.jpg')
```

	precision	recall	f1-score	support
Non-Current	0.97	0.55	0.71	45938
Current	0.03	0.49	0.06	1467
avg / total	0.94	0.55	0.69	47405

AUC: 52.1%
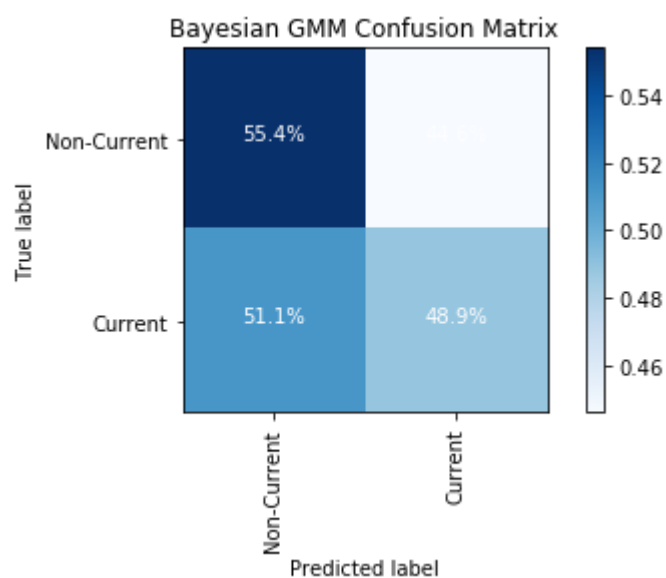
In [26]:
```python
class_names = ['Current', 'Non-Current']

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()


print('ROC_AUC_SCORE ; ', roc_auc_score(y_test, prediction_bgmm))
# Compute confusion matrix
cnf_matrix = confusion_matrix(y_test, prediction_bgmm)
np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names, title= 'Confusion matri
x, BGMM')
plt.savefig('prediction/bgmm/cm'+str(' BGMM-')+str(nrows)+'.jpg')
plt.show()
```
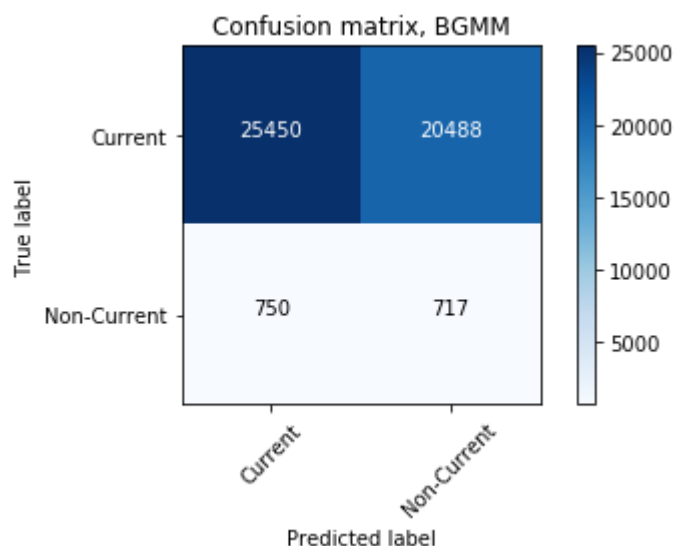
```
ROC_AUC_SCORE ;   0.521380065832
Confusion matrix, without normalization
[[25450 20488]
 [  750   717]]
```

Confusion matrix, BGMM

Applying PCA

In [36]:
```python
pca = PCA(0.99, whiten=True)
pca_x_test = pca.fit_transform(x_test)
pca_x_test.shape
```

Out[36]: (47405, 30)

In [37]:
```python
clf_bgmm = mixture.BayesianGaussianMixture(n_components=2,
                                    covariance_type='full').fit(pca_x_test
)

prediction_bgmm = clf_bgmm.predict(pca_x_test)
```

In [38]:
```python
print(classification_report(y_test, prediction_bgmm, target_names=['Non-Curren
t', 'Current']))
print ("AUC: ", "{:.1%}".format(roc_auc_score(y_test, prediction_bgmm)))
cm = confusion_matrix(y_test, prediction_bgmm)
```

```
              precision    recall  f1-score   support

 Non-Current       1.00      0.63      0.77     45938
     Current       0.08      1.00      0.15      1467

 avg / total       0.97      0.64      0.75     47405

AUC:   81.3%
```

In [ ]: