K Means

In [18]:
```
from sklearn import datasets
from sklearn import preprocessing
from sklearn.model_selection import train_test_split

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
```

In [19]:
```
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[19]: (237024, 58)

In [20]:
```
column_list=df.columns.tolist()
```

In [21]:
```
df.head()
```

Out[21]:

	originalloanamount	originalloanterm	originalinterestratepercentage	graceperiodnumber	obligorcr
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 [22]:
```
# prepare label for scikit-learn
Y=df.label.values
Y.shape
```

Out[22]: (237024,)

In [23]:
```
# prepare input data for scikit-learn
input=df.values
input.shape
```

Out[23]: (237024, 58)

In [24]:
```
# calculate train/test split

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

```
189619
```

In [25]:
```
# 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[25]: (189619, 58)

In [26]:
```
export_x_test = pd.DataFrame(data=x_test)
```

In [27]:
```
export_x_test.columns=column_list
export_x_test.rename(columns={'label':'True Label'}, inplace=True)
export_x_test.head()
```

Out[27]:

	originalloanamount	originalloanterm	originalinterestratepercentage	graceperiodnumber	obligorcr
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 [28]:
```
#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 [29]:
```
x_scaler=StandardScaler()
x_train = x_scaler.fit_transform(x_train)
x_test = x_scaler.fit_transform(x_test)
```

In [30]:
```
kmeans=KMeans(n_clusters=2, n_init=1, init='random')

clf_kmeans=kmeans.fit(x_train, y_train)
confidence_kmeans=clf_kmeans.score(x_test, y_test)
```

In [31]:
```
x_pred = x_test
```

In [32]:
```
prediction_kmeans = clf_kmeans.predict(x_pred)
```

In [33]:
```
np.unique(prediction_kmeans)
```

Out[33]:
```
array([0, 1])
```

In [34]:
```
export_x_test['Predicted Label']=prediction_kmeans
```

In [35]:
```
export_x_test.head()
```

Out[35]:

	originalloanamount	originalloanterm	originalinterestratepercentage	graceperiodnumber	obligorcr
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 [36]:
```
export_x_test.shape
```

Out[36]: (47405, 59)

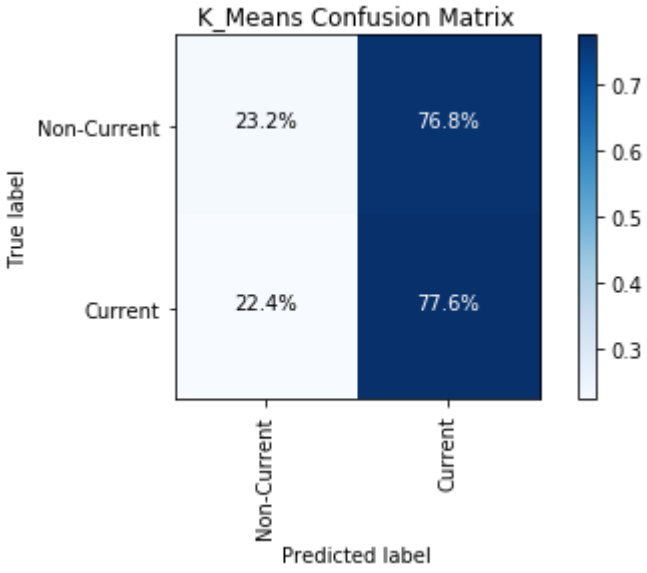
In [37]:
```
export_x_test.to_csv(path+"prediction/Kmeans/predicated_Kmeans_abs_loans_"+str(nr
```

In [38]:
```
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 [39]:
```
def plot_gridsearch_cv(results, estimator, x_min, x_max, y_min, y_max,save=False,

    # 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 marked by
        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()
```

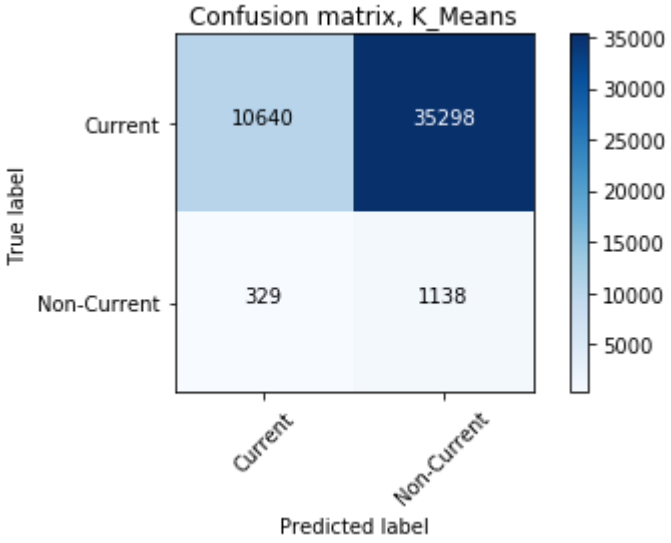
In [40]:
```
print(classification_report(y_test, prediction_kmeans, target_names=['Non-Current
print ("AUC: ", "{:.1%}".format(roc_auc_score(y_test, prediction_kmeans)))
cm = confusion_matrix(y_test, prediction_kmeans)
plot_confusion_matrix(cm, title="K_Means Confusion Matrix",save=True,
                      saveas='prediction/kmeans/cm'+str(' K_Means Accuracy-')+str
```

```
              precision    recall  f1-score   support

Non-Current       0.97      0.23      0.37     45938
    Current       0.03      0.78      0.06      1467

avg / total       0.94      0.25      0.36     47405

AUC:  50.4%
```



In [41]:
```
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_kmeans))
# Compute confusion matrix
cnf_matrix = confusion_matrix(y_test, prediction_kmeans)
np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names, title= 'Confusion matrix,
plt.savefig('prediction/kmeans/cm'+str(' K_Means-')+str(nrows)+'.jpg')
plt.show()
```

```
ROC_AUC_SCORE ;  0.503674657313
Confusion matrix, without normalization
[[10640 35298]
 [  329  1138]]
```


Confusion matrix, K_Means
Current
Non-Current
True label
10640
35298
329
1138
35000
30000
25000
20000
15000
10000
5000
Current
Non-Current
Predicted label

In [ ]: