Spectral clustering_unsupervised

Typical Spectral clustering is done three steps:

1. Create a similarity graph between objects to cluster.
2. Compute the first k eigenvectors of its Laplacian matrix to define a feature vector for each object.
3. Run k-means on these features to separate objects into k classes.

In practice Spectral Clustering is very useful when the structure of the individual clusters is highly non-convex or more generally when a measure of the center and spread of the cluster is not a suitable description of the complete cluster. For instance when clusters are nested circles on the 2D plan.

If affinity is the adjacency matrix of a graph, this method can be used to find normalized graph cuts.

When calling fit, an affinity matrix is constructed using either kernel function such the Gaussian (aka RBF) kernel of the euclidean distanced d(X, X):

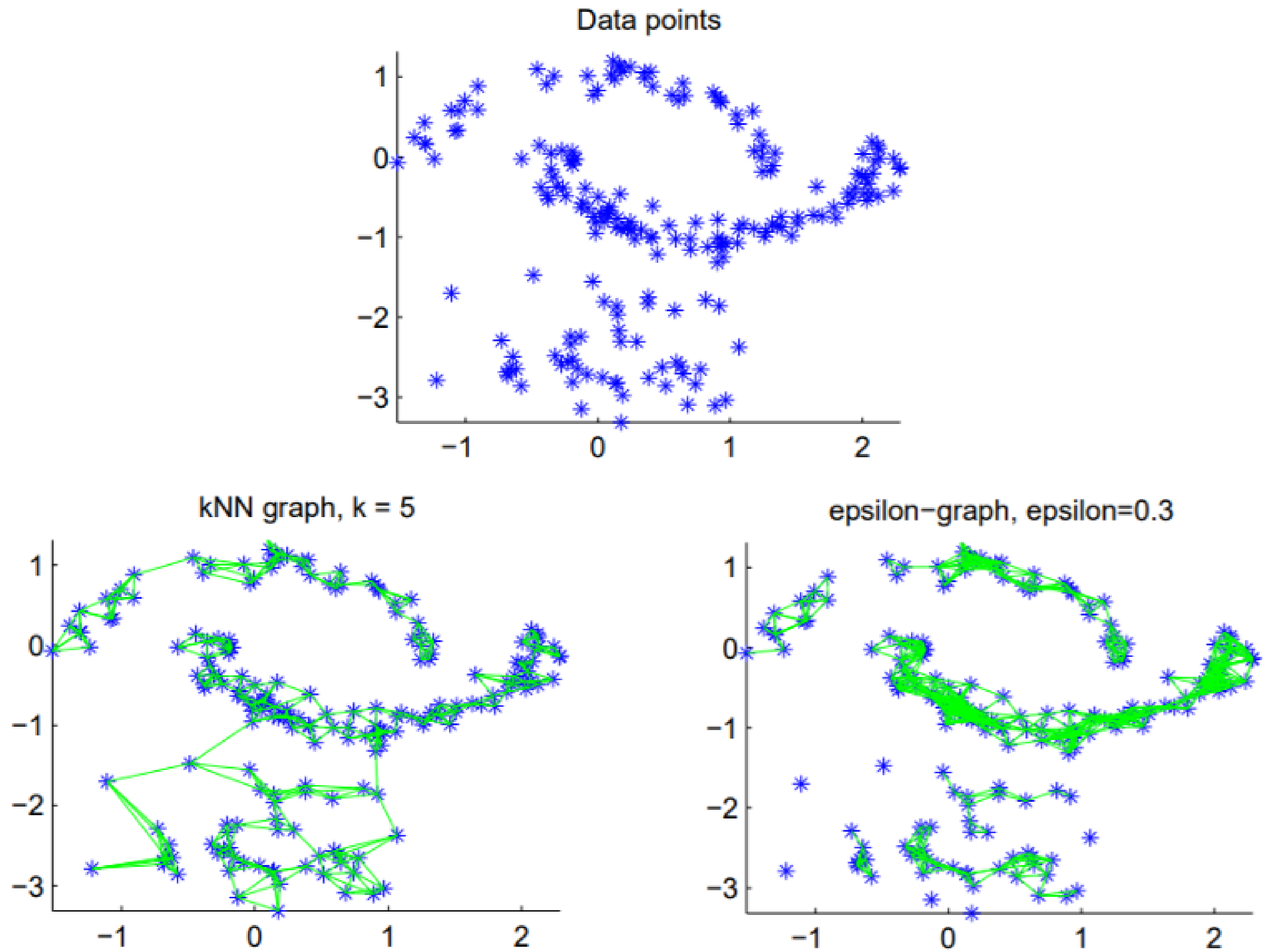
np.exp(-gamma * d(X,X) ** 2) or a k-nearest neighbors connectivity matrix.

Alternatively, using precomputed, a user-provided affinity matrix can be used.

Step 1

There are different ways to construct a graph representing the relationships between data points:

ε-neighborhood graph: Each vertex is connected to vertices falling inside a ball of radius ε where ε is a real value that has to be tuned in order to catch the local structure of data. k-nearest neighbor graph: Each vertex is connected to its k-nearest neighbors where k is an integer number which controls the local relationships of data.

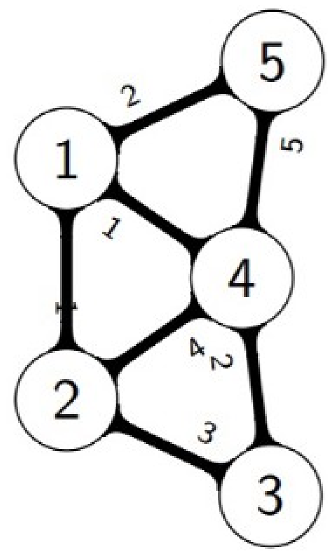


Step 2

Now that we have our graph, we need to form its associated Laplacian matrix.

$\mathbf{A}$	adjacency matrix
$\mathbf{W}$	weight matrix
$\mathbf{D}$	(diagonal) degree matrix
$\mathbf{L} = \mathbf{D} - \mathbf{W}$	graph **Laplacian** matrix

$$\mathbf{L} = \begin{pmatrix} 4 & -1 & 0 & -1 & -2 \\ -1 & 8 & -3 & -4 & 0 \\ 0 & -3 & 5 & -2 & 0 \\ -1 & -4 & -2 & 12 & -5 \\ -2 & 0 & 0 & -5 & 7 \end{pmatrix}$$



Step 3

Run K-Means

K is choosen by projecting the points into a non-linear embedding and analyzing the eigenvalues of the Laplacian matrix one can deduce the number of clusters present in the data. When the similarity graph is not fully connected, the multiplicity of the eigenvalue λ = 0 gives us an estimation of k.

In [1]:
```
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn import cluster
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.cluster import AgglomerativeClustering
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
from sklearn import metrics

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

%matplotlib inline
```

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)

In [3]:
```
column_list=df.columns.tolist()
```

In [4]:
```
df.head()
```

Out[4]:

	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 [5]:
```
# prepare label for scikit-learn
Y=df.label.values
Y.shape
```

Out[5]: (237024,)

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

Out[6]: (237024, 58)

In [7]:
```
# calculate train/test split

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

```
189619
```

In [8]:
```
# 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[8]: (189619, 58)

In [9]:
```
export_x_test = pd.DataFrame(data=x_test)
```

In [10]:
```
export_x_test.columns=column_list
export_x_test.rename(columns={'label':'True Label'}, inplace=True)
export_x_test.head()
```

Out[10]:

	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 [11]:
```
#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 [12]:
```
x_scaler=StandardScaler()
x_train = x_scaler.fit_transform(x_train)
x_test = x_scaler.fit_transform(x_test)
```

In [13]:
```
n_clusters=2
eigen_solver=['None', 'arpack', 'lobpcg', 'amg']

affinity = ['rbf', 'sigmoid', 'polynomial', 'poly', 'linear', 'cosine', 'neare
st_neighbors']
n_neighbors = 10
assign_labels = ['kmeans', 'discretize']
n_jobs=-1

# spectral = cluster.SpectralClustering(
#         n_clusters=n_clusters, eigen_solver=eigen_solver[3], random_state=5
4, n_init=10,
#     n_neighbors=n_neighbors, eigen_tol=0.0, n_jobs=n_jobs, assign_labels=ass
ign_labels[0],
#         affinity=affinity[1]).fit(x_test)

spectral = cluster.SpectralClustering(
        n_clusters=n_clusters, affinity= 'nearest_neighbors', random_state=54)
.fit(x_test)
```

```
C:\Progra~1\Anaconda3_4\lib\site-packages\sklearn\manifold\spectral_embedding
_.py:234: UserWarning: Graph is not fully connected, spectral embedding may n
ot work as expected.
  warnings.warn("Graph is not fully connected, spectral embedding"
```

In [14]:
```
#x_pred = x_test
```

In [15]:
```
prediction_spectral = spectral.labels_
```

In [16]:
```
np.unique(prediction_spectral)
```

Out[16]:
```
array([0, 1])
```

In [17]:
```
np.bincount(np.array(prediction_spectral).reshape(1,prediction_spectral.size)[
0])
```

Out[17]:
```
array([47367,    38], dtype=int64)
```

In [18]:
```
export_x_test['Predicted Label']=prediction_spectral
```

In [19]:
```
export_x_test.head()
```

Out[19]:

	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 × 59 columns

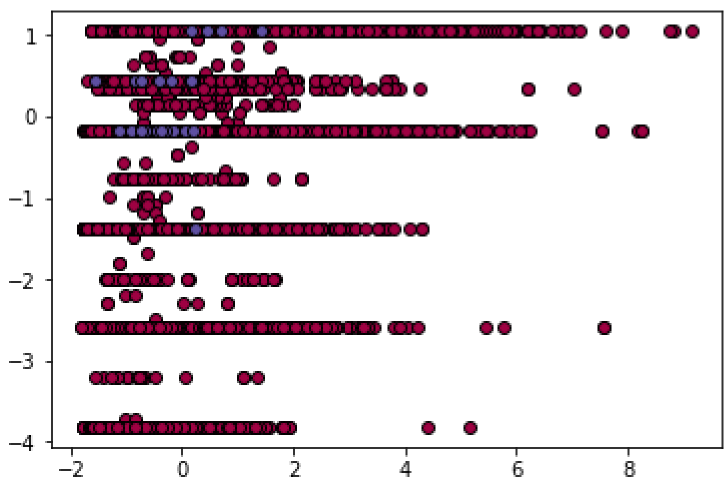
In [20]:
```
export_x_test.shape
```

Out[20]: (47405, 59)

In [21]:
```
export_x_test.to_csv(path+"prediction/spectral/predicated_spectral_abs_loans_"
+str(nrows)+".csv", chunksize=10000)
```

In [22]:
```
# print('Estimated number of clusters: %d' % n_clusters)
# print("Homogeneity: %0.3f" % metrics.homogeneity_score(y_test, prediction_sp
ectral))
# print("Completeness: %0.3f" % metrics.completeness_score(y_test, prediction_
spectral))
# print("V-measure: %0.3f" % metrics.v_measure_score(y_test, prediction_spectr
al))
# print("Adjusted Rand Index: %0.3f"
#       % metrics.adjusted_rand_score(y_test, prediction_spectral))
# print("Adjusted Mutual Information: %0.3f"
#       % metrics.adjusted_mutual_info_score(y_test, prediction_spectral))
# print("Silhouette Coefficient: %0.3f"
#       % metrics.silhouette_score(x_test, prediction_spectral))
```

In [23]:
```
labels= spectral.labels_
core_samples_mask = np.zeros_like(spectral.labels_, dtype=bool)
#core_samples_mask[birch.core_sample_indices_] = False
# Black removed and is used for noise instead.
unique_labels = set(labels)
colors = [plt.cm.Spectral(each)
          for each in np.linspace(0, 1, len(unique_labels))]
for k, col in zip(unique_labels, colors):
    if k == -1:
        # Black used for noise.
        col = [0, 0, 0, 1]

    class_member_mask = (labels == k)

    xy = x_test[class_member_mask & core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=14)

    xy = x_test[class_member_mask & ~core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=6)

#plt.title('Estimated number of clusters: %d' % n_clusters_)
plt.show()
```

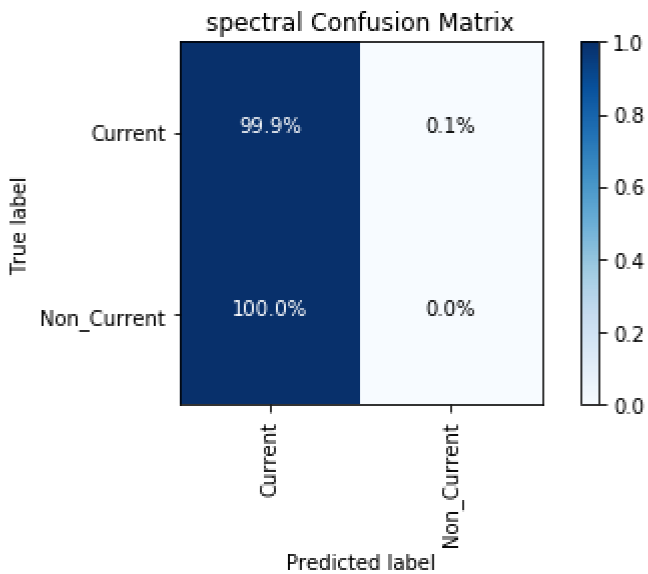


In [24]:
```
def plot_confusion_matrix(cm, title, classes=['Current', 'Non_Current'],
                          cmap=plt.cm.Blues, save=False, saveas="MyFigure.png"
):

    # 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 [25]:
```
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()
```

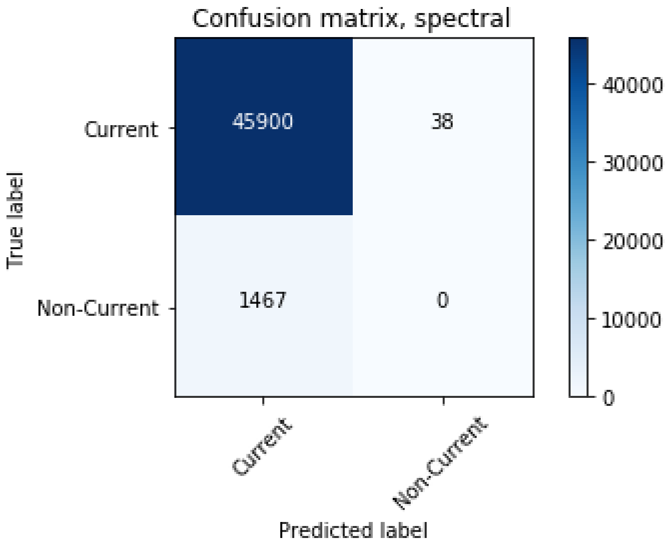
In [26]:
```
print(classification_report(y_test, prediction_spectral, target_names=['Curren
t', 'Non_Current']))
print ("AUC: ", "{:.1%}".format(roc_auc_score(y_test, prediction_spectral)))
cm = confusion_matrix(y_test, prediction_spectral)
plot_confusion_matrix(cm, title="spectral Confusion Matrix",save=True,
                      saveas='prediction/spectral/cm'+str(' spectral Accuracy-
')+str(nrows)+'.jpg')
```

```
             precision    recall  f1-score   support

    Current       0.97      1.00      0.98     45938
Non_Current       0.00      0.00      0.00      1467

avg / total       0.94      0.97      0.95     47405

AUC:  50.0%
```



In [27]:
```
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_spectral))
# Compute confusion matrix
cnf_matrix = confusion_matrix(y_test, prediction_spectral)
np.set_printoptions(precision=2)

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

```
ROC_AUC_SCORE ;  0.49958639906
Confusion matrix, without normalization
[[45900    38]
 [ 1467     0]]
```



In [ ]: