

# Ward Unstructured Hierarchical Clustering

Ward Clustering is a hierarchical clustering. Unstructured clustering is performed using connectivity. Clustering is restricted to the k-Nearest Neighbors graph: it's a hierarchical clustering with structure prior.

```
In [1]: from sklearn import datasets
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

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)
```

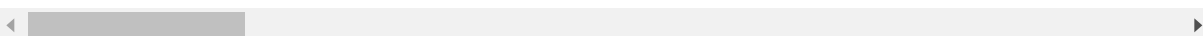
```
In [3]: column_list=df.columns.tolist()
```

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

Out[4]:

	<b>originalloanamount</b>	<b>originalloanterm</b>	<b>originalinterestratepercentage</b>	<b>graceperiodnuml</b>
<b>0</b>	66711.84	60	3.29	1
<b>1</b>	16258.45	60	0.90	0
<b>2</b>	31930.41	72	2.90	1
<b>3</b>	26065.02	65	0.90	0
<b>4</b>	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)

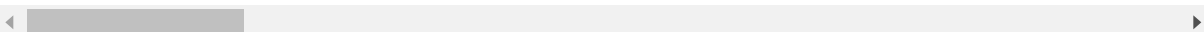
In [9]: `export_x_test = pd.DataFrame(data=x_test)`

```
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	graceperiodnuml
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]: from sklearn.neighbors import kneighbors_graph
connectivity = kneighbors_graph(x_test, n_neighbors=10, include_self=False)

ward = AgglomerativeClustering(n_clusters=2, connectivity=connectivity,
                               linkage='ward').fit(x_test)
```

C:\Progra~1\Anaconda3\_4\lib\site-packages\sklearn\cluster\hierarchical.py:193: UserWarning: the number of connected components of the connectivity matrix is 2 > 1. Completing it to avoid stopping the tree early.  
affinity='euclidean')

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

```
In [15]: prediction_ward = ward.labels_
```

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

Out[16]: array([0, 1], dtype=int64)

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

```
Out[17]: array([46194, 1211], dtype=int64)
```

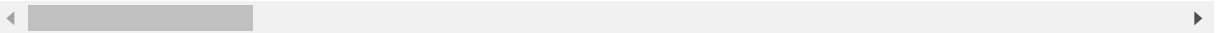
```
In [18]: export_x_test['Predicted Label']=prediction_ward
```

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

```
Out[19]:
```

	originalloanamount	originalloanterm	originalinterestratepercentage	graceperiodnuml
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/ward_u/predicated_ward_u_abs_loans_"+str(nrows)+".csv", chunksize=10000)
```

```
In [22]: 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 [23]: def plot_gridsearch_cv(results, estimator, x_min, x_max, y_min, y_max, save=False, 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 marked 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)

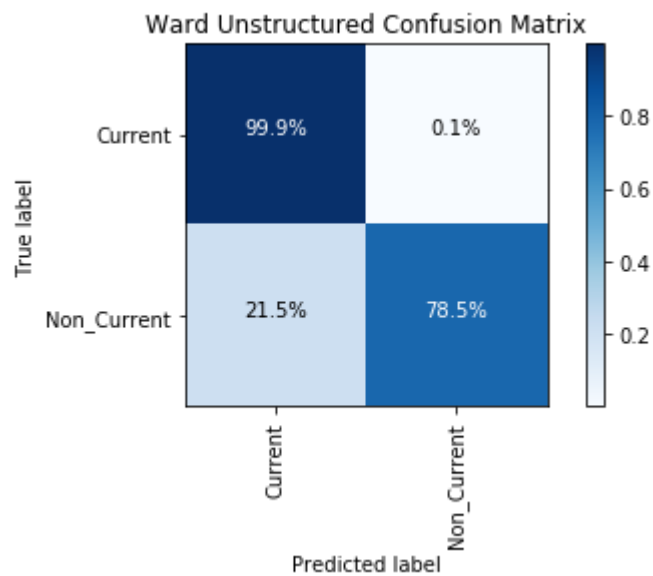
    plt.show()

```

```
In [24]: print(classification_report(y_test, prediction_ward, target_names=['Current',
'Non-Current']))
print ("AUC: ", "{:.1%}".format(roc_auc_score(y_test, prediction_ward)))
cm = confusion_matrix(y_test, prediction_ward)
plot_confusion_matrix(cm, title="Ward Unstructured Confusion Matrix",save=True
,
                        saveas='prediction/ward_u/cm'+str(' Ward Unstructured Ac
curacy-')+str(nrows)+' .jpg')
```

	precision	recall	f1-score	support
Current	0.99	1.00	1.00	45938
Non-Current	0.95	0.78	0.86	1467
avg / total	0.99	0.99	0.99	47405

AUC: 89.2%



```

In [25]: 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_ward))
# Compute confusion matrix
cnf_matrix = confusion_matrix(y_test, prediction_ward)
np.set_printoptions(precision=2)

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

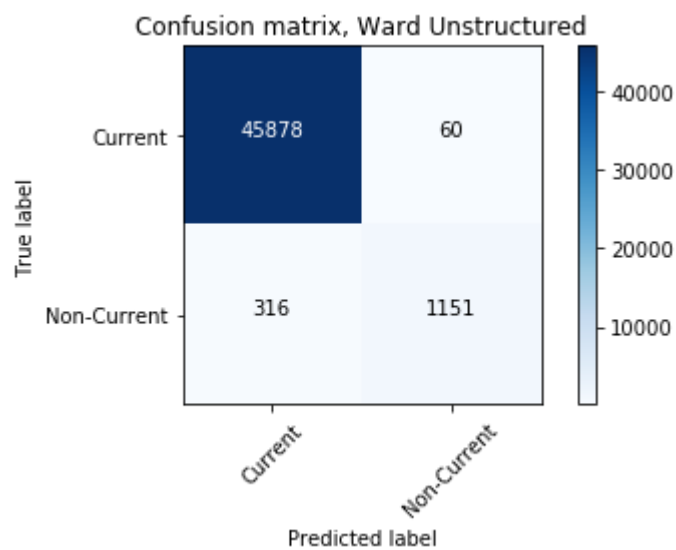
```



ROC\_AUC\_SCORE ; 0.891644151064

Confusion matrix, without normalization

```
[[45878  60]  
 [ 316 1151]]
```



In [ ]: