Practical 5: Clustering (version 1)

Overview

The aim of these lab exercises is to give you some hands-on experience with clustering, with a focus on numeric data. For these exercises, you will work with a small sample data set provided, as well as a couple of the classic data mining data sets included in scikit-learn.

Getting Started

1. On the KEATS page for this module (7CCSMDM1), under **[Week 5]**, find and download the file named **clustering-tutorial-data.csv** containing the small sample data set which we used in the Tutorial and will be used in the first part of this Practical. The file is labelled **Tutorial data**.
2. Note that you will also need the **Tutorial (version 2)** and the **Solution to Tutorial**, found in the same section of the KEATS page.

1 Complete the Tutorial in Python, manually computing K-Means

Start by completing the Tutorial that was given in class on Thursday 15th February. This is posted on the KEATS page, under **[Week 5]**, with the label **Tutorial (version 2)**.

A PDF of my solution written using Excel is also posted, with the label **Solution to Tutorial**.

This shows how to compute **K-Means** clustering, step-by-step, and also shows how to compute the key metrics we have discussed in lecture:

- Within-cluster score
- Between-cluster score
- Overall cluster score
- Calinski-Harabaz score
- Silhouette score

[1.1] For the first part of this Practical, write your solution to the Tutorial using Python. Do not use any of the scikit-learn built-in functions yet! You will do this next—but before you go there, you will find it helpful to your learning if you walk through these steps manually.

[1.2] Compare your answers to those in my PDF solution and make sure your Python code is generating the right answers, including the intermediate calcluations.

2 Now use **scikit-learn** to compute K-Means

Once you have written **K-Means** by hand, you can start using the one provided by **scikit-learn**.

The **sklearn.cluster** package contains the function you want:
`http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html`

Here is how you use it:

```python
# load the clustering package definition
import sklearn.cluster as cluster
#
# K is the number of clusters
# X is the M x N dimensional vector of input data, where:
# M is the number of instances (also called 'samples' or 'points')
# N is the number of attributes (also called 'features' or 'variables')
#
# initialise the clustering function:
km = cluster.KMeans( n_clusters=K )
#
# compute the clusters:
km.fit( X )
```

Once you have fit the model, you will want to look at the outputs:

- km.cluster_centers_ is a $K \times N$ dimensional vector containing the attribute values that define the cluster centres, or *centroids*
- km.labels_ is a K dimensional vector containing the cluster labels, in the range $[0 \ldots (K-1)]$
- km.inertia_ is the within-cluster score (also called 'inertia')

In addition to the **within-cluster score**, you will also want to look at various other metrics (as above). The **sklearn.metrics** package contains a large number of metrics for various data mining activities. You have used some of them already, in previous labs. Here, we will concentrate on the ones defined for **clustering** tasks:
`http://scikit-learn.org/stable/modules/classes.html#clustering-metrics`

Here is how to call the functions that compute the scores we have discussed in class:

```python
# load the metrics package definition
import sklearn.metrics as metrics
#
# compute the silhouette coefficient for each sample (returns a M-dimensional
    vector):
SC_ss = metrics.silhouette_samples( X, km.labels_, metric='euclidean' )
#
# compute the overall silhouette coefficient (returns a scalar):
SC = metrics.silhouette_score( X, km.labels_, metric='euclidean' )
#
# compute the calinski-harabaz score (returns a scalar):
CH = metrics.calinski_harabaz_score( X, km.labels_ )
```

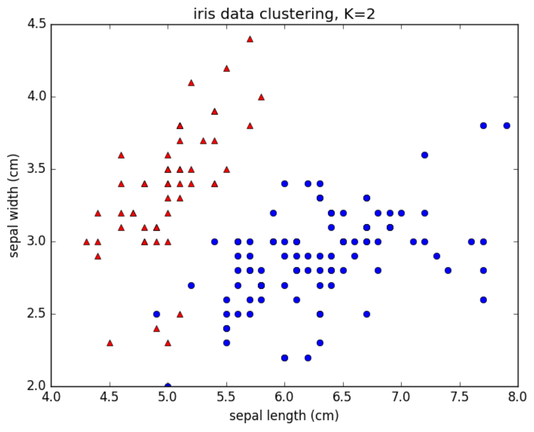
[2.1] Start by using the small sample data set provided (**clustering-tutorial-data.csv**).

[2.2] Then try it using the **iris** data set:

```
# load the datasets package definition
import sklearn.datasets as datasets
#
#-get data from standard package
d = datasets.load_iris()
X = d.data
M = len( d.data ) # number of instances
N = len( d.feature_names ) # number of features
```

[2.3] Plot your results with $K \in \{2, 3, 4, 5\}$ clusters. With the iris data set, there are 4 attributes: *sepal length*, *sepal width*, *petal length* and *petal width*. Select any pair of attributes to plot using a scatter plot, using different colours and/or shapes for the instances from different clusters. Can you visually assess how well your clustering did, as well as looking at the quantitative metrics computed by your code?

My solutions for $K = 2$ and $K = 4$ are shown in Figure 1.



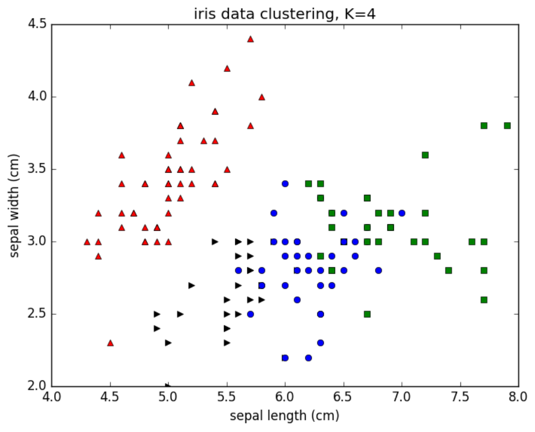


Figure 1: K-Means clustering on the Iris data

3 Now use **scikit-learn** to compute Hierarchical Clustering

As you should remember from the lecture, there are multiple approaches to clustering, and K-Means is only one of them. Another approach is called **Hierarchicial** clustering. Refer back to the lecture slides (Week 5, part 1) if you don't remember what we discussed in class.

The **Agglomerative Hierarchical** clustering package in scikit-learn is here:
`http://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html`

Here is how you use the package:

```
# load the clustering package definition
import sklearn.cluster as cluster
#
# K is the number of clusters
# X is the M x N dimensional vector of input data, where:
# M is the number of instances (also called 'samples' or 'points')
# N is the number of attributes (also called 'features' or 'variables')
#
# initialise the clustering function:
hc = cluster.AgglomerativeClustering( n_clusters=K, linkage="average", affinity="
    euclidean" )
#
# compute the clusters:
hc.fit( X )
```

Once you have fit the model, you will want to look at the outputs, which are similar to the K-Means outputs:

- hc.labels_ is a K dimensional vector containing the cluster labels, in the range $[0 \ldots (K-1)]$
- hc.leaves_ is the number of leaves in the dendrogram which represents the clustering
- hc.children_ is a $(M-1) \times 2$ dimensional vector containing the linkages between samples which indicate how the clusters are formed in the dendrogram

The outputs (aside from the labels) are a bit awkward to understand, but there is a handy function in **scipy** which displays dendrograms:
`https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.cluster.hierarchy.dendrogram.html`

Note that this function was written to work with the **scipy.cluster.hierarchy.linkage** function, described here:
`https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.cluster.hierarchy.linkage.html#scipy.cluster.hierarchy.linkage`

However, you will probably find the **sklearn.cluster.AgglomerativeClustering** package more useful generally, including having access to more metrics. So here is how to estimate the linkage information and produce a dendrogram from the sklearn output:

```
plt.figure()
# initialise data structure for scipy dendrogram printing function
Z = np.empty( [ len( hc.children_ ), 4 ], dtype=float )
# steps (A) and (B) below are thanks to:
# https://github.com/scikit-learn/scikit-learn/blob/70
    cf4a676caa2d2dad2e3f6e4478d64bc\
b0506f7/examples/cluster/plot_hierarchical_clustering_dendrogram.py
# for hints on how to combine the sklearn agglomerative clustering with the
    dendrogra\
m plotting function of scipy.
# (A) compute distances between each pair of children: since we don't
# have this information, we can use a uniform one for plotting
cluster_distances = np.arange( hc.children_.shape[0] )
# (B) compute the number of observations contained in each cluster level
cluster_sizes = np.arange( 2, hc.children_.shape[0]+2 )
for i in range( len( hc.children_ )):
    Z[i][0] = hc.children_[i][0]
    Z[i][1] = hc.children_[i][1]
    Z[i][2] = cluster_distances[i]
    Z[i][3] = cluster_sizes[i]
# plot dendrogram
hierarchy.dendrogram( Z )
plt.show()
```

My results for $K = 3$ are shown in Figure 2.

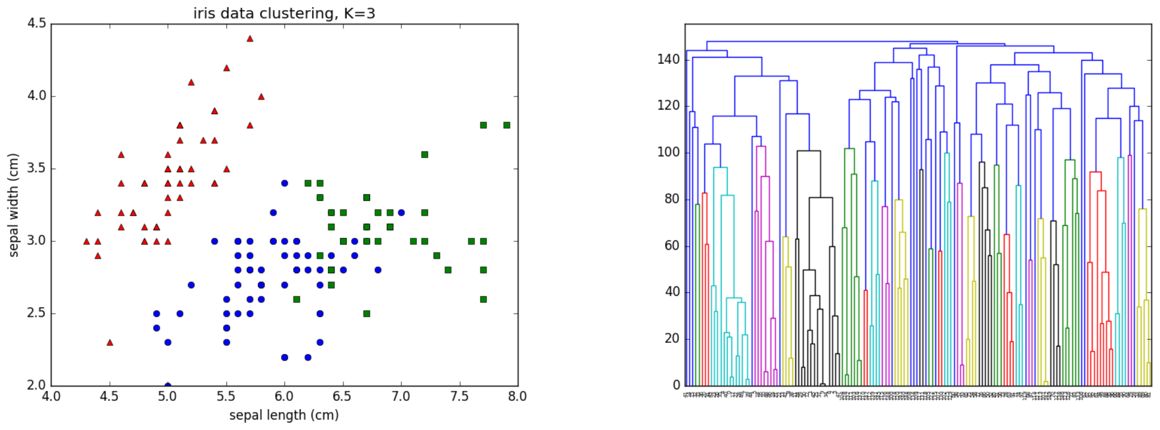


Figure 2: Hierarchical clustering on the Iris data

[3.1] Rewrite your solution to the previous part using Agglomerative clustering instead of K-Means on the Iris data set.

[3.2] How do your scores compare?

4 Now use **scikit-learn** to compute DBSCAN

There is one other clustering approach that you might find helpful: **Density-based** clustering. I have added a few slides to the lecture notes that describes this type of clustering. Please download **Lecture slides, part 1 (version 2)** from the KEATS page for **[Week 5]** and read pages 27–31.

The **sklearn** function is documented here:
`http://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html`

And here is some simple code to show you how to call it:

```python
# load the clustering package definition
import sklearn.cluster as cluster
#
# K is the number of clusters
# X is the M x N dimensional vector of input data, where:
# M is the number of instances (also called 'samples' or 'points')
# N is the number of attributes (also called 'features' or 'variables')
#
# initialise the clustering function:
db = cluster.DBSCAN( eps=0.5, min_samples=1 )
#
# compute the clusters:
db.fit( X )
```

My results for $EPS = 0.5$ and $MIN\_SAMPLES = 1$ are shown in Figure 3. The plot on the left shows the *nearest neighbour* distances for each sample, plotted in sorted order. It can be seen that where the distance $= 0.5$ (on the y-axis), we start to see a sharper incline in the distances. This plot shows the nearest neighbour, hence $MIN\_SAMPLES = 1$. You will likely get better results looking at the k-th nearest neighbour, such as the 5-th or 10-th nearest neighbour.

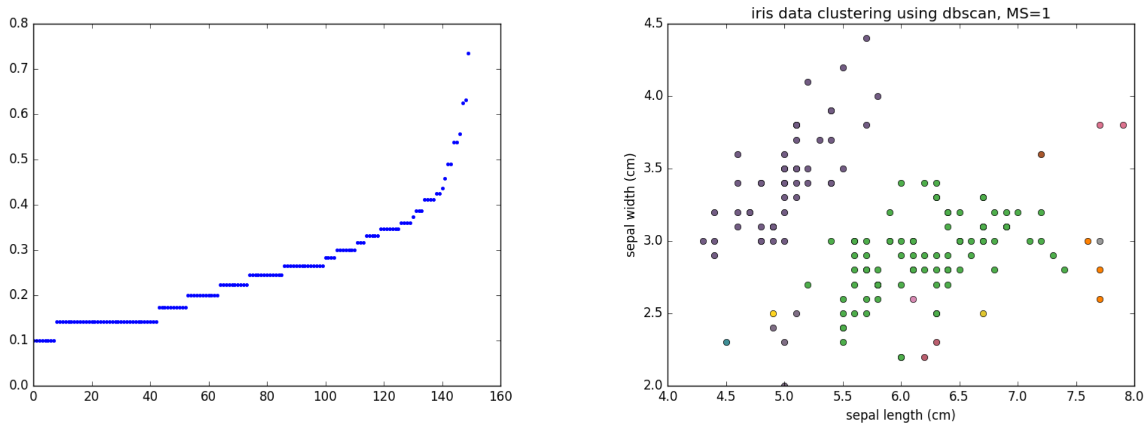


Figure 3: DBSCAN clustering on the Iris data

[4.1] Rewrite your solution to the previous parts using DBSCAN clustering instead of K-Means on the Iris data set.

[4.2] How do your scores compare?