ml-04-ex1-KMeans-elbow_nocode

November 3, 2020

Claudio Sartori

Elaboration from the example given in Sebastian Raschka, 2015

https://github.com/rasbt/python-machine-learning-book

1 Machine Learning - Lab

1.1 Working with Unlabeled Data – Clustering Analysis

1.1.1 Find the best number of clusters with k_means

1.1.2 Overview

- Grouping objects by similarity using k-means
  - Using the elbow method to find the optimal number of clusters
  - Quantifying the quality of clustering via silhouette plots

[1]:
```
from IPython.display import Image
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, silhouette_samples

%matplotlib inline

rnd_state = 42 # This variable will be used in all the procedure calls allowing␣
↪a random_state parameter
               # in this way the running can be perfectly reproduced
               # just change this value for a different experiment
```

2 Grouping objects by similarity using k-means

In this example we will use an *artificial* data set

1. load the data file from `'ex1_4dim_data.csv'`
2. check the shape and plot the content

3. observe the plot and decide which are the most interesting columns, to use in the plots of the clusters

- make a 2d plot of the two most promising columns

4. Use the elbow method to find the optimal number of clusters: test `KMeans` with varying number of clusters, from 2 to 10, fitting the data and computing the inertia and the silhouette score

5. Choose the optimal number of clusters looking at the plots, then cluster the data, plot the clusters and plot the scores of the individual samples
6. For comparison, repeat 5 with two clusters

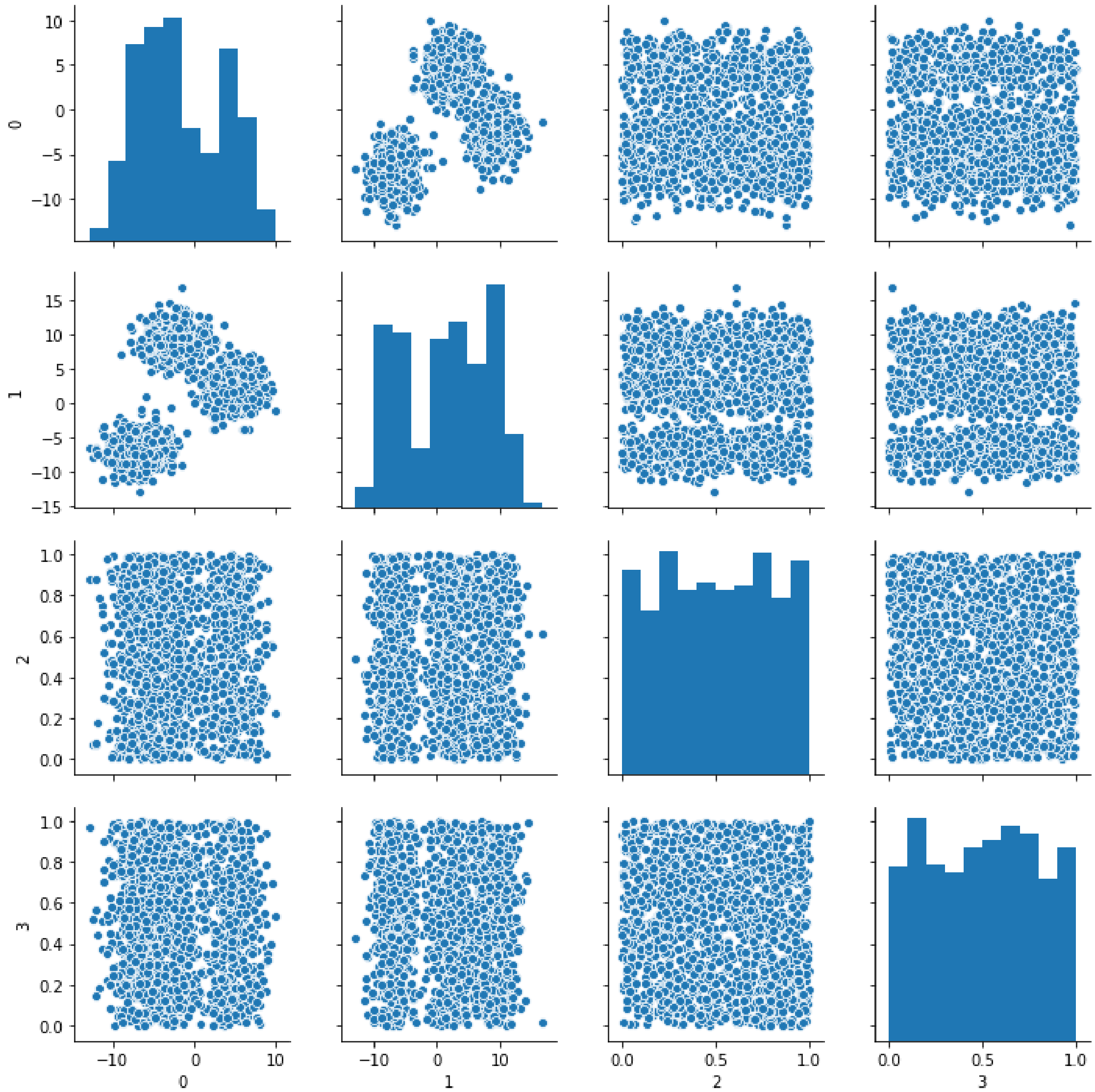
[2]:
```
data_file = 'ex1_4dim_data.csv'
delimiter = ','
# to fill
```

[3]:
```
# to fill
```

[3]:
```
(1500, 4)
```

[4]:
```
# to fill
```

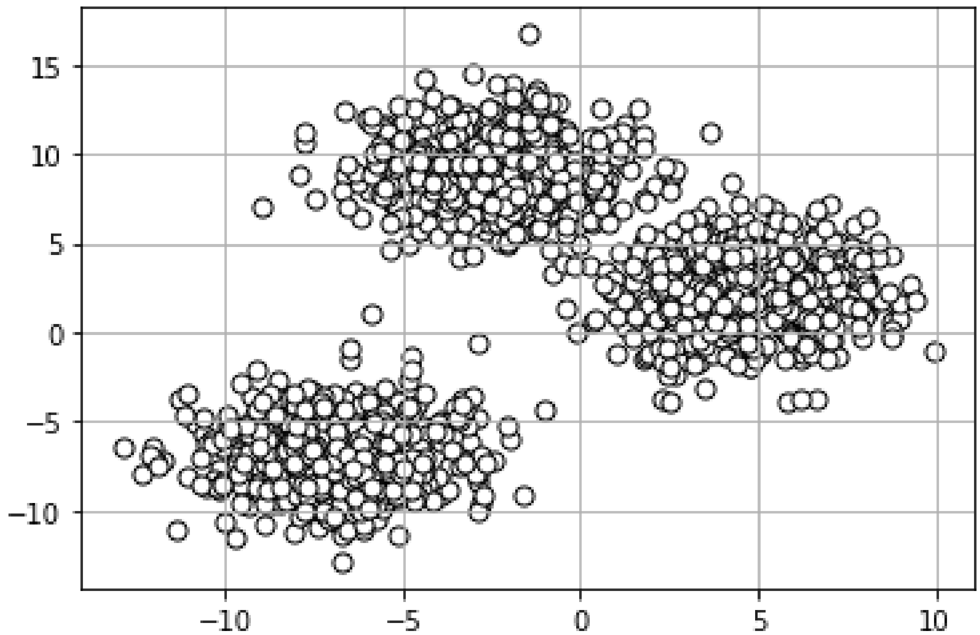
[4]:
```
<seaborn.axisgrid.PairGrid at 0x1a179b6810>
```



2.0.1 3. Observe the pairplots

In this simple example you can easily see that the two most interesting columns are `0` and `1`.

[5]:
```
# to fill
```



[6]:
```
from plot_clusters import plot_clusters
```

[7]:
```
help(plot_clusters)
```

```
Help on function plot_clusters in module plot_clusters:

plot_clusters(X, y, dim, points, labels_prefix='cluster',
points_name='centroids', colors=<matplotlib.colors.ListedColormap object at
0x10e9b3dd0>, points_color=(0.09019607843137255, 0.7450980392156863,
0.8117647058823529, 1.0))
    Plot a two dimensional projection of an array of labelled points
    X:      array with at least two columns
    y:      vector of labels, length as number of rows in X
    dim:    the two columns to project, inside range of X columns, e.g. (0,1)
    points: additional points to plot as 'stars'
    labels_prefix: prefix to the labels for the legend ['cluster']
    points_name:   legend name for the additional points ['centroids']
    colors: a color map
    points_color: the color for the points
```

2.1 Using the elbow method to find the optimal number of clusters

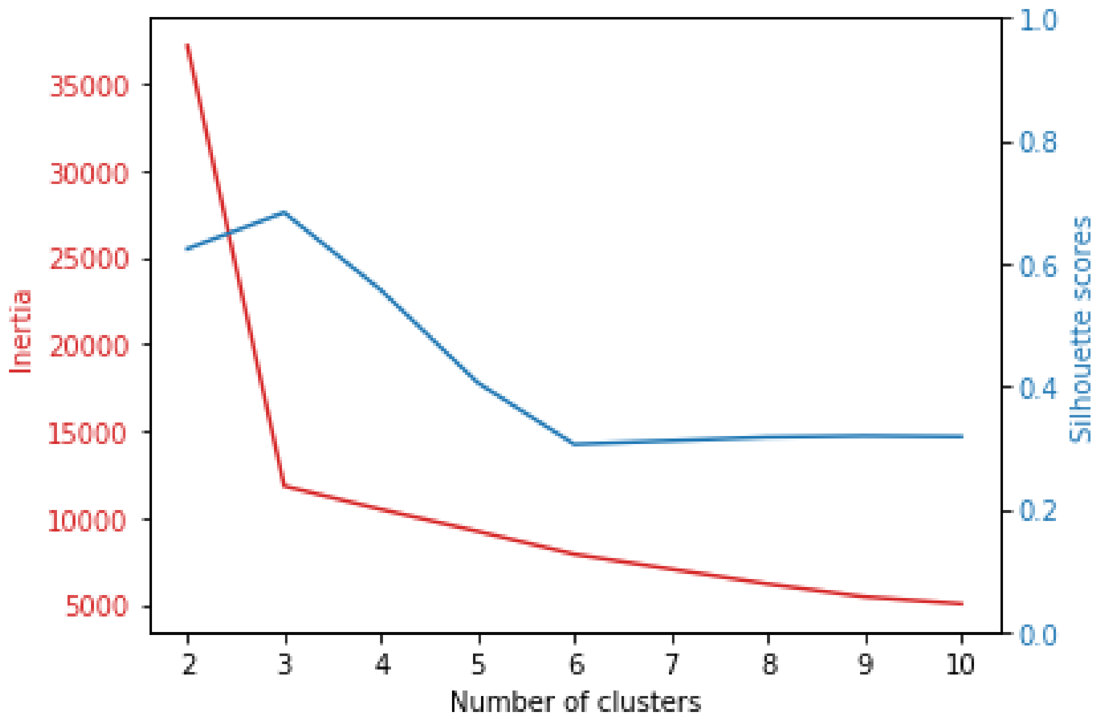
We will try **k_means** with a number of clusters varying from 2 to 10

- prepare two emptys lists for inertia and silhouette scores
- For each value of the number of clusters:

- initialize an estimator for KMeans and fit_predict
- we will store the distortion (from the fitted model) in the variable distortions
- using the function silhouette_score from sklearn.metrics with arguments the data and the fitted labels, we will fill the variable silhouette_scores

Then we will plot the two lists in the y axis, with the range of k in the x axis. The plot with two different scales in the y axis can be done according to the example shown in the notebook two_scales.ipynb.

[10]:
```
# to fill
```



2.1.1 5. Cluster with the optimal number

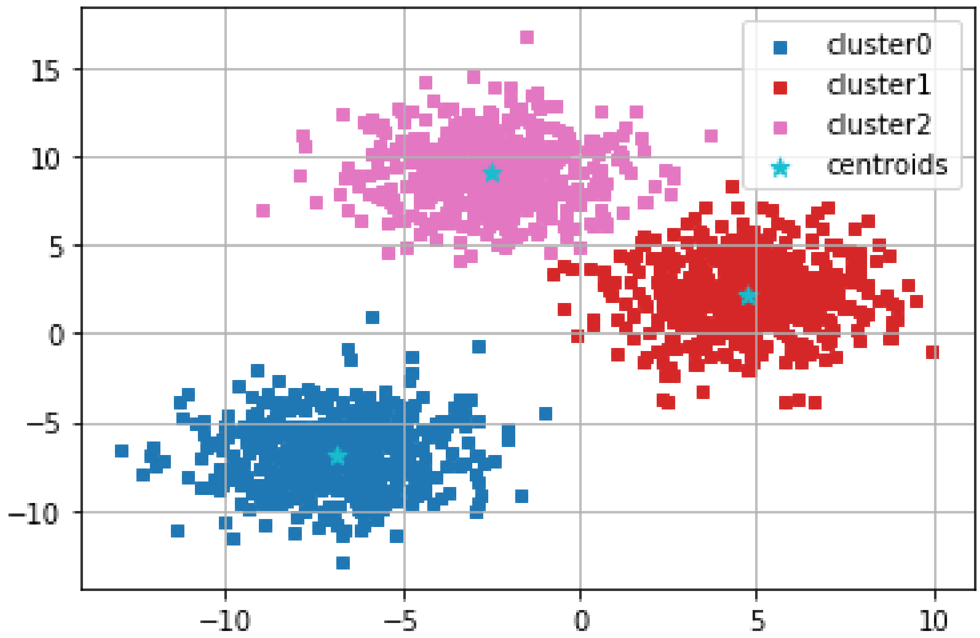
Choose the best value of n_clusters using the elbow method and cluster the data, then print the centroids.

[13]:
```
# to fill
```

[13]:
```
array([[-6.89370123, -6.83658926,  0.52620605,  0.52371904],
       [ 4.75108211,  2.11850327,  0.4917521 ,  0.49502881],
       [-2.50474216,  9.09132188,  0.49394552,  0.48136246]])
```

Hint: for plot_clusters to work convert pandas to numpy.

[14]:
```
plot_clusters(<# to fill>)
```



[15]:
```
# to fill
```

```
Distortion: 11831.85
```

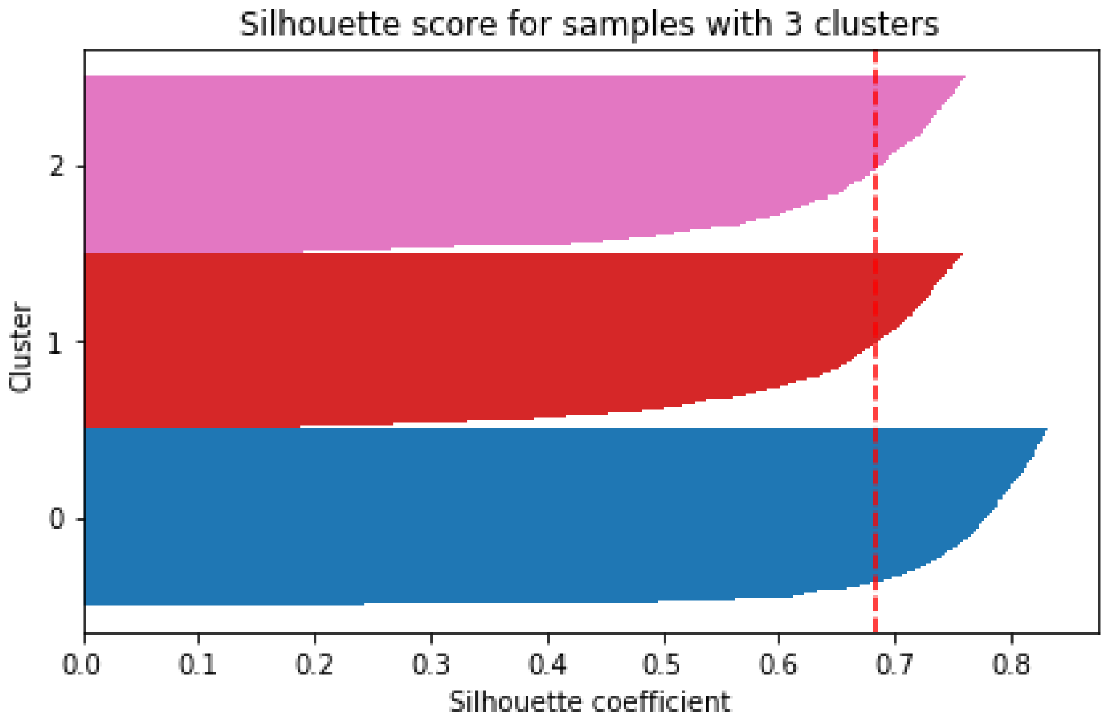
2.2 Quantifying the quality of clustering via silhouette plots

[16]:
```
from plot_silhouette import plot_silhouette
```

[17]:
```
help(plot_silhouette)
```

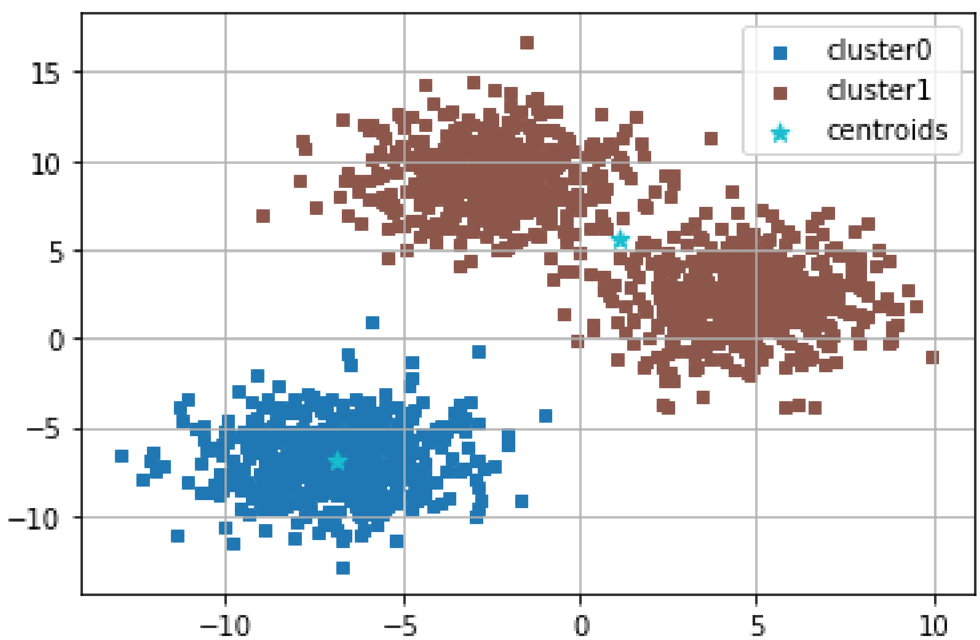
```
Help on function plot_silhouette in module plot_silhouette:

plot_silhouette(silhouette_vals, y, colors=<matplotlib.colors.ListedColormap
object at 0x10e9b3dd0>)
    Plotting silhouette scores for the individual samples of a labelled data
set.
    The scores will be grouped according to labels and sorted in descending
order.
    The bars are proportional to the score and the color is determined by the
label.

    silhouette_vals: the silhouette values of the samples
    y:               the labels of the samples
```

[18]:
```
# to fill
```



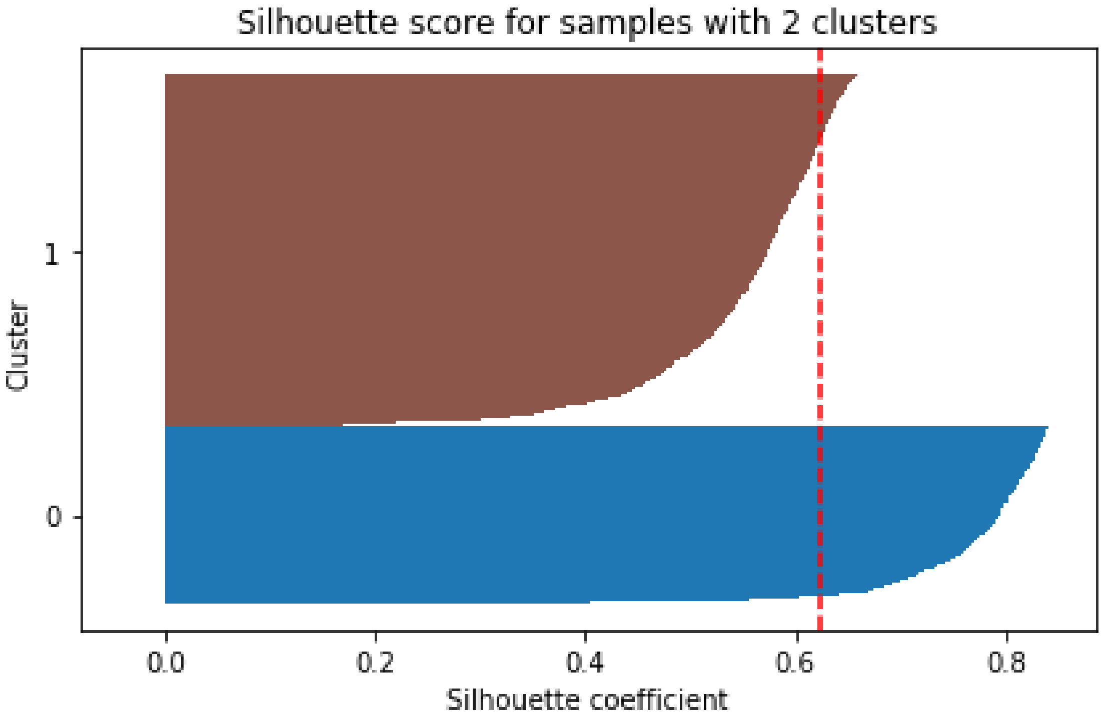
2.2.1 6. Comparison to "bad" clustering:

[20]:
```
# to fill
```



[21]:

```
# to fill
```



Silhouette score for samples with 2 clusters
Cluster
1
0
0.0
0.2
0.4
0.6
0.8
Silhouette coefficient

[ ]:

```
```