Example_of_Lab_exam

January 3, 2021

Julian Sherollari©

1 Machine Learing Exam

Example of Lab exam - Find the best number of clusters with k_means Prof. Claudio

1.0.1 Tasks

Find the clusters in the included dataset. The solution must be produced as a Python Notebook. The notebook must include appropriate comments and must produce:

1. The boxplots of the attributes and a comment on remarkable situations, if any (2pt)
2. A pairplot of the data (see Seaborn pairplot) and a comment on remarkable situations, if any (2pt)
3. A clustering schema using a method of your choice exploring a range of parameter values (5pt)
4. The plot of the global inertia (SSD) and silhouette index for the parameter values you examine (4pt)
5. The optimal parameters of your choice (4pt)
6. A pairplot of the data using as hue the cluster assignment with the optimal parameter (3pt)
7. A plot of the silhouette index for the data points, grouped according to the clusters (4pt)
8. A sorted list of the discovered clusters for decreasing sizes (7pt)

[18]:
```
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
```

[19]:
```
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
```

[20]:
```
data_file = 'lab_exercise.csv'
delimiter = ','
X = np.loadtxt(data_file, delimiter = delimiter)
```

```
---------------------------------------------------------------------------
OSError                                   Traceback (most recent call last)
<ipython-input-20-2273bcf5d769> in <module>
      1 data_file = 'lab_exercise.csv'
      2 delimiter = ','
----> 3 X = np.loadtxt(data_file, delimiter = delimiter)

/opt/anaconda3/lib/python3.8/site-packages/numpy/lib/npyio.py in loadtxt(fname,
 ↪dtype, comments, delimiter, converters, skiprows, usecols, unpack, ndmin,␣
 ↪encoding, max_rows)
    959             fname = os_fspath(fname)
    960         if _is_string_like(fname):
--> 961             fh = np.lib._datasource.open(fname, 'rt', encoding=encoding
    962             fencoding = getattr(fh, 'encoding', 'latin1')
    963             fh = iter(fh)

/opt/anaconda3/lib/python3.8/site-packages/numpy/lib/_datasource.py in␣
 ↪open(path, mode, destpath, encoding, newline)
    193
    194     ds = DataSource(destpath)
--> 195     return ds.open(path, mode, encoding=encoding, newline=newline)
    196
    197

/opt/anaconda3/lib/python3.8/site-packages/numpy/lib/_datasource.py in␣
 ↪open(self, path, mode, encoding, newline)
    533                                     encoding=encoding, newline=newlin )
    534         else:
--> 535             raise IOError("%s not found." % path)
    536
    537

OSError: lab_exercise.csv not found.
```

[21]:
```
X.shape
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-21-bc5f1a0adac8> in <module>
----> 1 X.shape

NameError: name 'X' is not defined
```

In order to exploit the Pandas DataFrame features we generate df from X

df = pd.DataFrame(X) df.boxplot()

1.1 1

Columns 0 and 3 have a range much smaller than 1 and 2. The distributions onf 0 and 3 seem to be equal. Poddibly, a min-max rescaling could point out some additional insight. Let's look at the pairplots and consider if it is worth to do this transformation.

[ ]:
```
sns.pairplot(df)
```

1.2 2

The pairplots show that the two most interesting columns are 1 and 2, their pairplot shows evident clusters. The pairplots of 0 and 3 show that those columns are uniformly distributed and do not show any pattern.

[16]:
```
int_cols = [1,2] # Interesting columns
```

1.3 3

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.

[ ]:
```
k_range = range(2,11)
```

[ ]:
```
distortions = []
silhouette_scores = []
for i in k_range:
    km = KMeans(n_clusters=i,
                init='k-means++',
                n_init=10,
                max_iter=300,
                random_state=rnd_state)
    y_km = km.fit_predict(X)
    distortions.append(km.inertia_)
    silhouette_scores.append(silhouette_score(X,y_km))
```

1.4 4

[ ]:
```
fig, ax1 = plt.subplots()

color = 'tab:red'
ax1.set_xlabel('Number of clusters')
ax1.set_ylabel('Inertia', color=color)
ax1.plot(k_range, distortions, color=color)
ax1.tick_params(axis='y', labelcolor=color)

ax2 = ax1.twinx() # instantiate a second axes that shares the same x-axis

color = 'tab:blue'
ax2.set_ylabel('Silhouette scores', color=color) # we already handled the␣
 ↪x-label with ax1
ax2.plot(k_range, silhouette_scores, color=color)
ax2.tick_params(axis='y', labelcolor=color)
ax2.set_ylim(0,1) # the axis for silhouette is [0,1]

fig.tight_layout() # otherwise the right y-label is slightly clipped
plt.show()
```

1.5 5

[ ]:
```
good_k = np.argmax(silhouette_scores) + k_range.start
print("The value of K providing the maximum silhouette index is {}".
 ↪format(good_k))
```

The value of K providing the maximum silhouette index is 4

1.6 6

[ ]:
```
km = KMeans(n_clusters=good_k,
            init='k-means++',
            n_init=10,
            max_iter=300,
            tol=1e-04,
            random_state=rnd_state)
y_km = km.fit_predict(X)
```

In order to show the ”hue” with the Seaborn pairplot it is necessary to add the cluster column and then specify as vars the data columns you want to plot.

[ ]:
```
df['cluster'] = y_km
```

[ ]:
```
sns.pairplot(df[int_cols + ['cluster']], vars = df[int_cols], hue = 'cluster')
```

1.7 7

The silhouette scores for the individual samples are computed with the function silhouette_samples The function plot_silhouette produces a 'horizontal bar-plot', with one bar for each sample, where the length of the bar is proportional to the silhouette score of the sample. The bars are grouped for cluster and sorted for decreasing length. A vertical line represents the silhouette score, i.e. the average on all the samples

[ ]:
```
# from plot_silhouette import plot_silhouette
from plot_silhouette2 import plot_silhouette
```

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

Help on function plot_silhouette in module plot_silhouette2

[ ]:
```
cluster_labels = np.unique(y_km)
n_clusters = cluster_labels.shape[0] # it is the number of rows
# Compute the Silhouette Coefficient for each sample, with the euclidean metric
silhouette_score_samples = silhouette_samples(X, y_km, metric='euclidean')
plt.title('Silhouette score for samples with {} clusters'.format(good_k))
plot_silhouette(silhouette_score_samples, y_km)
```

1.8 8

groupby().size() returns a dataframe, resetting the index we can name the newly created column with the counts

[ ]:
```
counts = pd.DataFrame(y_km).groupby(0).size().reset_index(name='Count')
print(counts)
```

Then we rename the first column as Cluster, sort by descending values of the counts and print the dataframe without showing the index

[17]:
```
counts = counts.rename(columns = {0: 'Cluster'}).sort_values(by = 'Count',␣
↪ascending = False)
print(counts.to_string(index = False))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-17-b4aa70f59bfa> in <module>
----> 1 counts = counts.rename(columns = {0: 'Cluster'}).sort_values(by =␣
↪'Count', ascending = False)
      2 print(counts.to_string(index = False))

NameError: name 'counts' is not defined
```

[ ]:
```

```