Species+Segmentation+with+Cluster+Analysis+Part+2+-+Exercise

February 11, 2021

1 Species Segmentation with Cluster Analysis

The Iris flower dataset is one of the most popular ones for machine learning. You can read a lot about it online and have probably already heard of it: https://en.wikipedia.org/wiki/Iris_flower_data_set

We didn't want to use it in the lectures, but believe that it would be very interesting for you to try it out (and maybe read about it on your own).

There are 4 features: sepal length, sepal width, petal length, and petal width.

You have already solved the first exercise, so you can start from there (you've done taken advantage of the Elbow Method).

Plot the data with 2, 3 and 5 clusters. What do you think that means?

Finally, import the CSV with the correct answers (iris_with_answers.csv) and check if the clustering worked as expected. Note that this is not how we usually go about clustering problems. If we have the answers prior to that, we would go for classification (e.g. a logistic regression).

1.1 Import the relevant libraries

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
from sklearn.cluster import KMeans
```

1.2 Load the data

Load data from the csv file: 'iris_dataset.csv'.

[3]:
```
# Load the data
data = pd.read_csv('iris_dataset.csv')
# Check the data
data.head()
```

[3]:

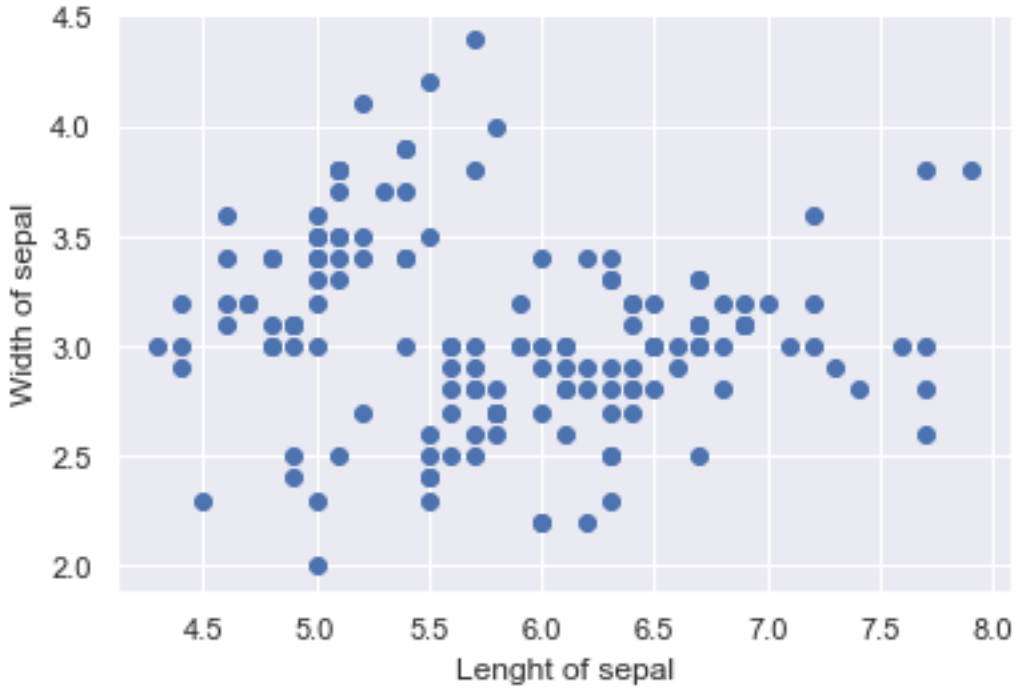
	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

1.3 Plot the data

For this exercise, try to cluster the iris flowers by the shape of their sepal.

Use the 'sepal_length' and 'sepal_width' variables.

[4]:
```
# create a scatter plot based on two corresponding features (sepal_length and␣
 ↪sepal_width; OR petal_length and petal_width)
plt.scatter(data['sepal_length'],data['sepal_width'])
# name your axes
plt.xlabel('Lenght of sepal')
plt.ylabel('Width of sepal')
plt.show()
```



1.4 Clustering (unscaled data)

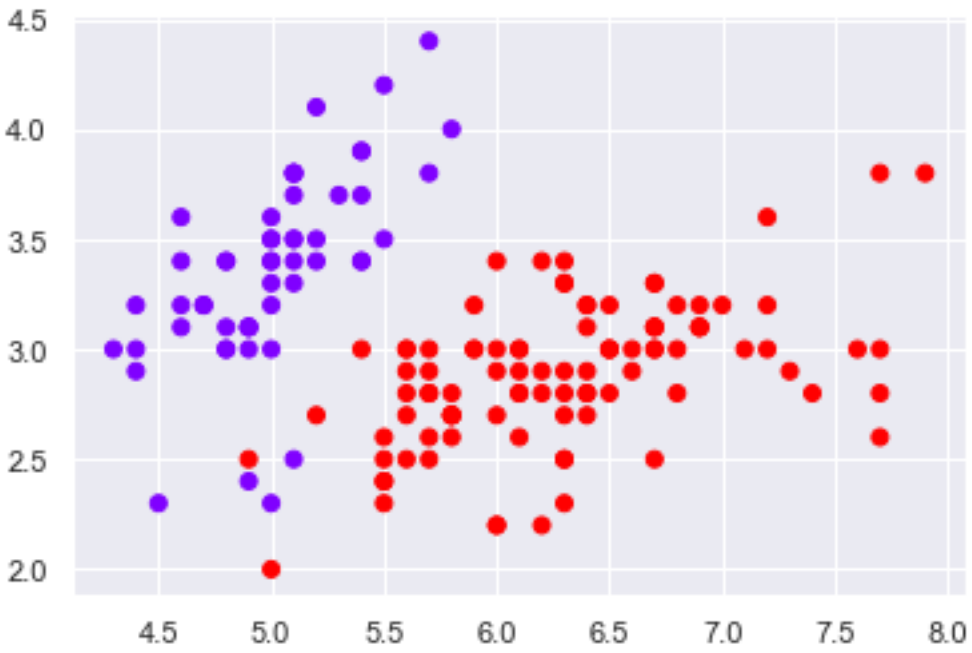
[5]:
```
# create a variable which will contain the data for the clustering
x = data.copy()
# create a k-means object with 2 clusters
kmeans = KMeans(2)
# fit the data
kmeans.fit(x)
```

[5]:
```
KMeans(n_clusters=2)
```

[6]:
```
# create a copy of data, so we can see the clusters next to the original data
clusters = data.copy()
# predict the cluster for each observation
clusters['cluster_pred']=kmeans.fit_predict(x)
```

[7]:
```
# create a scatter plot based on two corresponding features (sepal_length and␣
↪sepal_width; OR petal_length and petal_width)
plt.scatter(clusters['sepal_length'], clusters['sepal_width'], c= clusters␣
↪['cluster_pred'], cmap = 'rainbow')
```

[7]:
```
<matplotlib.collections.PathCollection at 0x153d7b6d5b0>
```



1.5 Standardize the variables

Import and use the scale method from sklearn to standardize the data.

[9]:
```
# import some preprocessing module
from sklearn import preprocessing

# scale the data for better results
x_scaled = preprocessing.scale(data)
```

1.6 Clustering (scaled data)

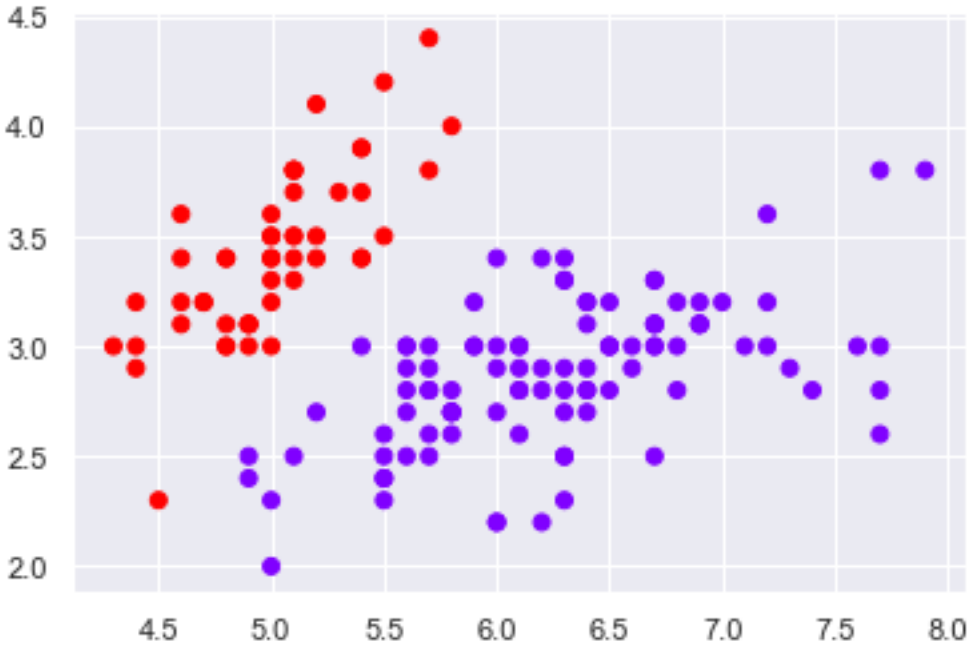
[10]:
```
# create a k-means object with 2 clusters
kmeans_scaled = KMeans(2)
# fit the data
kmeans_scaled.fit(x_scaled)
```

[10]:
```
KMeans(n_clusters=2)
```

[11]:
```
# create a copy of data, so we can see the clusters next to the original data
clusters_scaled = data.copy()
# predict the cluster for each observation
clusters_scaled['cluster_pred']=kmeans_scaled.fit_predict(x_scaled)
```

[12]:
```
# create a scatter plot based on two corresponding features (sepal_length and␣
↪sepal_width; OR petal_length and petal_width)
plt.scatter(clusters_scaled['sepal_length'], clusters_scaled['sepal_width'], c=␣
↪clusters_scaled ['cluster_pred'], cmap = 'rainbow')
```

[12]:
```
<matplotlib.collections.PathCollection at 0x153d7bd2fd0>
```



Looks like the two solutions are identical. That is because the original features have very similar scales to start with!

1.7 Take Advantage of the Elbow Method

1.7.1 WCSS

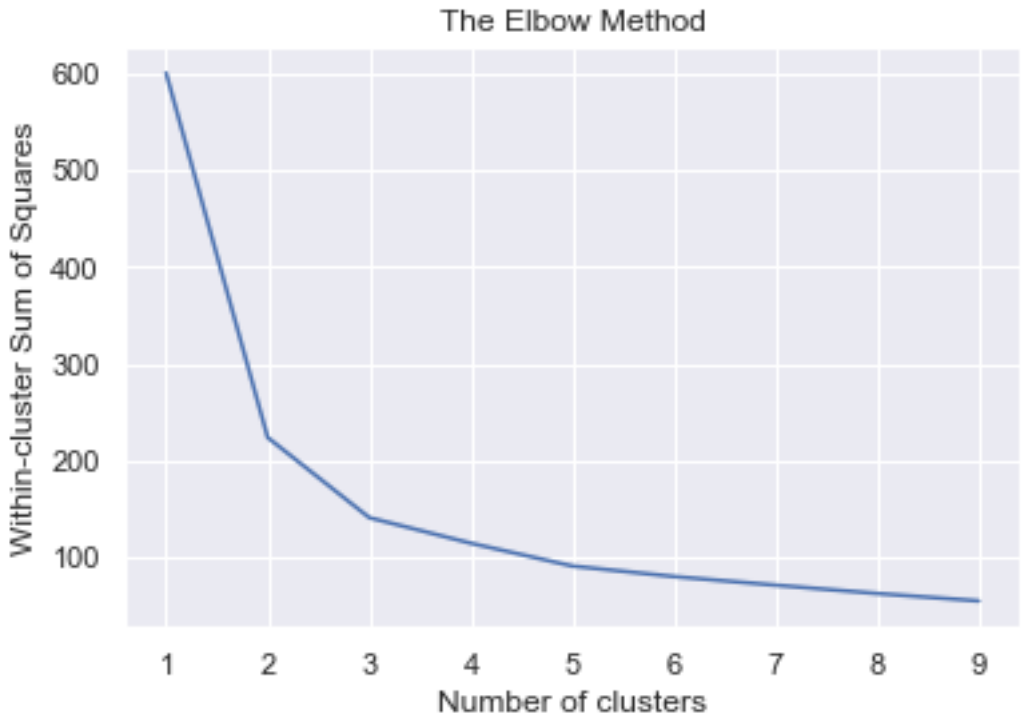
[13]:
```
wcss = []
# 'cl_num' is a that keeps track the highest number of clusters we want to use
 ↪the WCSS method for.
# We have it set at 10 right now, but it is completely arbitrary.
cl_num = 10
for i in range (1,cl_num):
    kmeans= KMeans(i)
    kmeans.fit(x_scaled)
    wcss_iter = kmeans.inertia_
    wcss.append(wcss_iter)
wcss
```

[13]:
```
[600.0000000000003,
 223.73200573676343,
 140.96581663074699,
 114.57790500611009,
 91.0024409856725,
 80.14000417219813,
 71.27443300992768,
 62.66310985043492,
 54.998705093177335]
```

1.7.2 The Elbow Method

[14]:
```
number_clusters = range(1,cl_num)
plt.plot(number_clusters, wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('Within-cluster Sum of Squares')
```

[14]:
```
Text(0, 0.5, 'Within-cluster Sum of Squares')
```



1.8 Understanding the Elbow Curve

Construct and compare the scatter plots to determine which number of clusters is appropriate for further use in our analysis. Based on the Elbow Curve, 2, 3 or 5 seem the most likely.

1.9 2 clusters

Start by separating the standardized data into 2 clusters (you've already done that!)

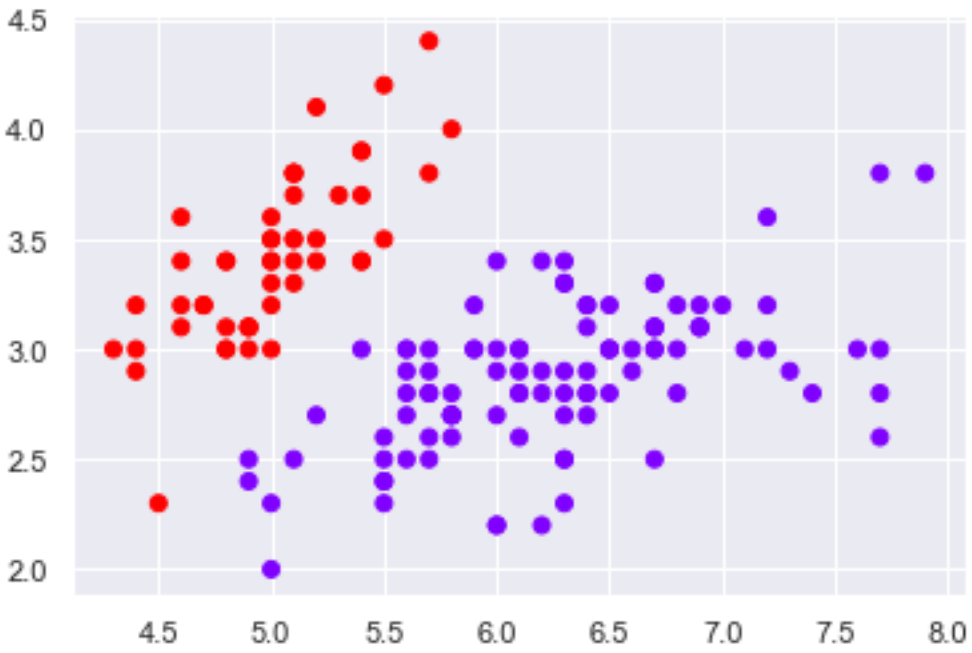
[16]:
```
# create a k-means object with 2 clusters
kmeans_scaled = KMeans(2)
# fit the data
kmeans_scaled.fit(x_scaled)
```

[16]:
```
KMeans(n_clusters=2)
```

Construct a scatter plot of the original data using the standardized clusters

[17]:
```
# create a scatter plot based on two corresponding features (sepal_length and␣
 ↪sepal_width; OR petal_length and petal_width)
plt.scatter(clusters_scaled['sepal_length'], clusters_scaled['sepal_width'], c=␣
 ↪clusters_scaled ['cluster_pred'], cmap = 'rainbow')
```

[17]:
```
<matplotlib.collections.PathCollection at 0x153d9d0f640>
```



1.10 3 clusters

Redo the same for 3 and 5 clusters

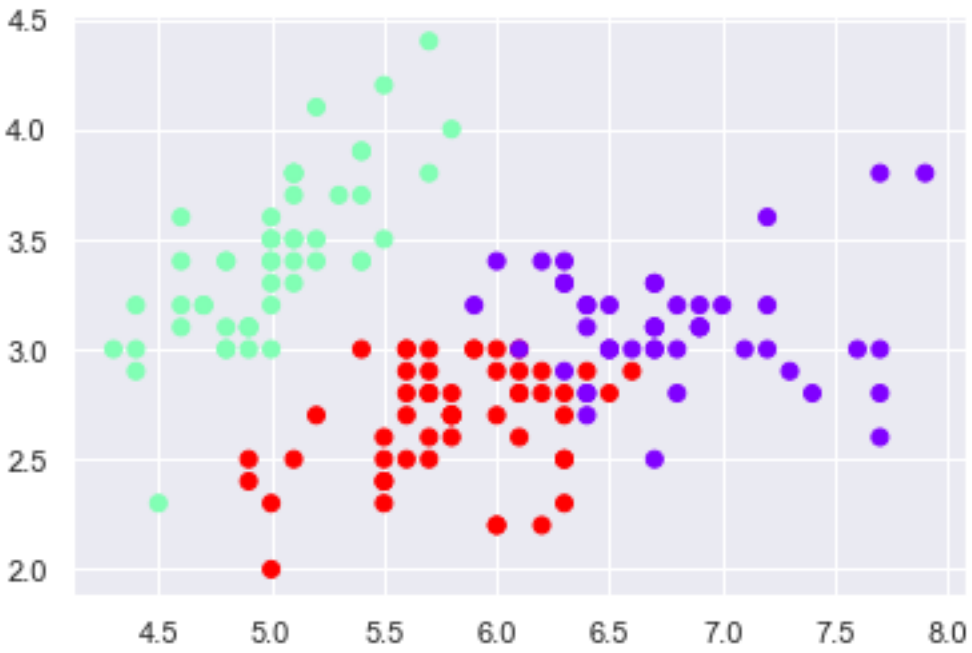
[18]:
```
# create a k-means object with 3 clusters
kmeans_scaled = KMeans(3)
# fit the data
kmeans_scaled.fit(x_scaled)
```

[18]:
```
KMeans(n_clusters=3)
```

[20]:
```
# create a copy of data, so we can see the clusters next to the original data
clusters_scaled = data.copy()
# predict the cluster for each observation
clusters_scaled['cluster_pred']=kmeans_scaled.fit_predict(x_scaled)
```

[21]:
```
# create a scatter plot based on two corresponding features (sepal_length and␣
 ↪sepal_width; OR petal_length and petal_width)
plt.scatter(clusters_scaled['sepal_length'], clusters_scaled['sepal_width'], c=␣
 ↪clusters_scaled ['cluster_pred'], cmap = 'rainbow')
```

[21]:
```
<matplotlib.collections.PathCollection at 0x153d9d25d00>
```



1.11 5 clusters

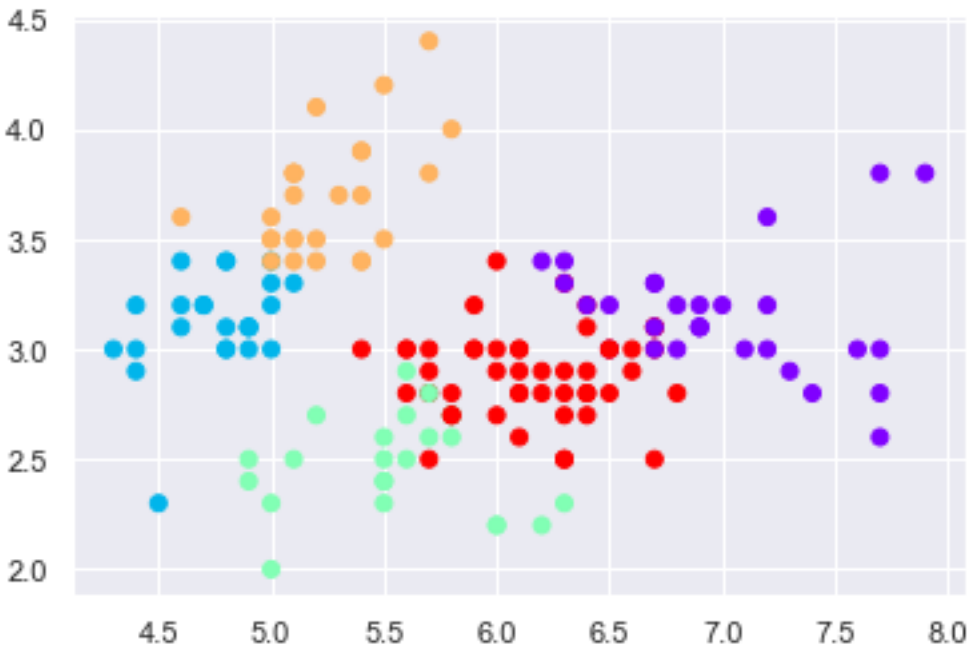
[23]:
```
# create a k-means object with 5 clusters
kmeans_scaled = KMeans(5)
# fit the data
kmeans_scaled.fit(x_scaled)
```

[23]:
```
KMeans(n_clusters=5)
```

[24]:
```
# create a copy of data, so we can see the clusters next to the original data
clusters_scaled = data.copy()
# predict the cluster for each observation
clusters_scaled['cluster_pred']=kmeans_scaled.fit_predict(x_scaled)
```

[25]:
```
# create a scatter plot based on two corresponding features (sepal_length and␣
 ↪sepal_width; OR petal_length and petal_width)
plt.scatter(clusters_scaled['sepal_length'], clusters_scaled['sepal_width'], c=␣
 ↪clusters_scaled ['cluster_pred'], cmap = 'rainbow')
```

[25]:
```
<matplotlib.collections.PathCollection at 0x153d9e091c0>
```



1.12 Compare your solutions to the original iris dataset

The original (full) iris data is located in iris_with_answers.csv. Load the csv, plot the data and compare it with your solution.

Obviously there are only 3 types, because that's the original (truthful) iris dataset.

The 2-cluster solution seemed good, but in real life the iris dataset has 3 SPECIES (a 3-cluster solution). Therefore, clustering cannot be trusted at all times. Sometimes it seems like x clusters are a good solution, but in real life, there are more (or less).

[40]:
```
data_answer = pd.read_csv('iris_with_answers.csv')
data_answer.head()
```

[40]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

[41]:
```
data_answer['species'].unique()
```

[41]:
```
array(['setosa', 'versicolor', 'virginica'], dtype=object)
```

[42]:
```
data_answer['species'] = data_answer['species'].replace({"setosa":1,␣
↪"versicolor":2, "virginica":3})
```

```
data_answer.head()
```

[42]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	1
1	4.9	3.0	1.4	0.2	1
2	4.7	3.2	1.3	0.2	1
3	4.6	3.1	1.5	0.2	1
4	5.0	3.6	1.4	0.2	1

[44]:
```
plt.scatter(data_answer['sepal_length'], data_answer['sepal_width'],␣
 ↪c=data_answer['species'], cmap = 'rainbow')
plt.show
```

[44]: <function matplotlib.pyplot.show(close=None, block=None)>