Assignment 4 – Solution

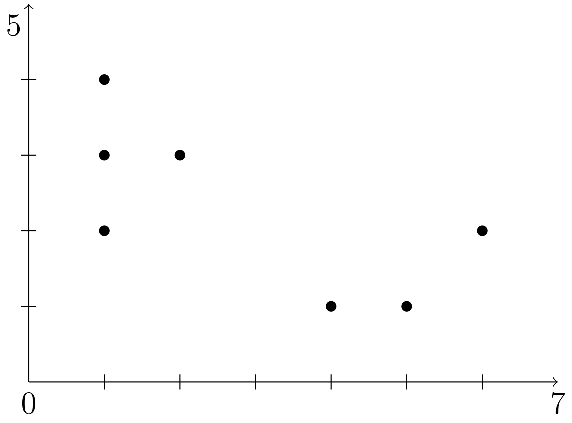
Machine Learning
MSc Business Analytics

Wolfram Wiesemann

1 Individual Assignment

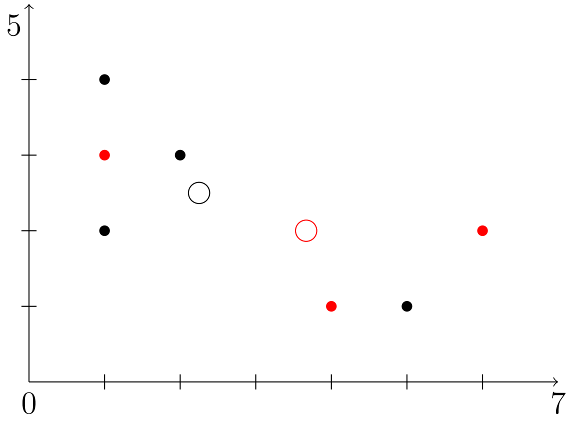
1. *Plot the observations in a two-dimensional graph.*

The graph looks as follows:



2. *Perform K-means clustering with $K = 2$ using the Euclidean norm. Toss a coin 7 times to initialise the algorithm.*

First we assign randomly $C_1 = \{2, 6, 7\}$ (red) and $C_2 = \{1, 3, 4, 5\}$ (black):



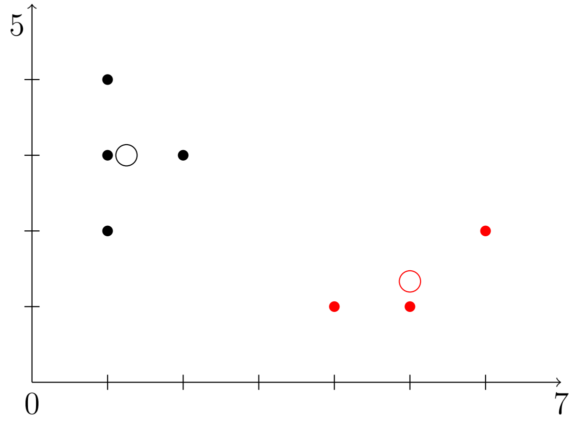
We compute the centroids of the two classes as $\boldsymbol{c}_1 = (\frac{11}{3}, 2)$ and $\boldsymbol{c}_2 = (\frac{9}{4}, \frac{5}{2})$. We now recompute the distances:

Obs. i	x_{i1}	x_{i2}	dist$(\boldsymbol{x}_i, \boldsymbol{c}_1)$	dist$(\boldsymbol{x}_i, \boldsymbol{c}_2)$
1	1	4	2.84	1.95
2	1	3	2.84	1.95
3	1	2	4.01	2.79
4	5	1	1.33	2.79
5	2	3	4.33	3.5
6	6	2	3.07	4.03
7	4	1	2.02	3.05

After reassignment, the new clusters are $C_1 = \{4, 6, 7\}$ and $C_2 = \{1, 2, 3, 5\}$. The new centroids of the two clusters are $\boldsymbol{c}_1 = (5, \frac{4}{3})$ and $\boldsymbol{c}_2 = (\frac{5}{4}, 3)$.

Obs. i	x_{i1}	x_{i2}	dist$(\boldsymbol{x}_i, \boldsymbol{c}_1)$	dist$(\boldsymbol{x}_i, \boldsymbol{c}_2)$
1	1	4	4.01	2.01
2	1	3	4.01	2.01
3	1	2	5.42	2.01
4	5	1	0.67	3.88
5	2	3	5.55	3.09
6	6	2	2.85	4.85
7	4	1	1.66	4.06

The clusters are still $C_1 = \{4, 6, 7\}$ and $C_2 = \{1, 2, 3, 5\}$. The algorithm thus terminates with the following result:



3. *Cluster the data using hierarchical clustering with complete linkage and the Euclidean norm. Draw the resulting dendrogram.*

We calculate the following pairwise distances between the observations:

	$\{1\}$	$\{2\}$	$\{3\}$	$\{4\}$	$\{5\}$	$\{6\}$	$\{7\}$
$\{1\}$	**0**						
$\{2\}$	1	**0**					
$\{3\}$	2	1	**0**				
$\{4\}$	5	4.47	4.12	**0**			
$\{5\}$	1.41	1	1.41	3.6	**0**		
$\{6\}$	5.38	5.09	5	1.41	4.12	**0**	
$\{7\}$	4.24	3.6	3.16	1	2.82	2.23	**0**

(Empty cells can be inferred from symmetry.) We first merge the 'clusters' $\{1\}$ and $\{2\}$:

	$\{1,2\}$	$\{3\}$	$\{4\}$	$\{5\}$	$\{6\}$	$\{7\}$
$\{1,2\}$	**0**					
$\{3\}$	2	**0**				
$\{4\}$	5	4.12	**0**			
$\{5\}$	1.41	1.41	3.6	**0**		
$\{6\}$	5.38	5	1.41	4.12	**0**	
$\{7\}$	4.24	3.16	1	2.82	2.23	**0**

We now merge the 'clusters' $\{4\}$ and $\{7\}$:

	$\{1,2\}$	$\{3\}$	$\{5\}$	$\{6\}$	$\{4,7\}$
$\{1,2\}$	**0**				
$\{3\}$	2	**0**			
$\{5\}$	1.41	1.41	**0**		
$\{6\}$	5.38	5	4.12	**0**	
$\{4,7\}$	5	4.12	3.6	2.23	**0**

We now merge the 'clusters' $\{5\}$ and $\{1,2\}$:

	$\{1,2,5\}$	$\{3\}$	$\{6\}$	$\{4,7\}$
$\{1,2,5\}$	**0**			
$\{3\}$	2	**0**		
$\{6\}$	5.38	5	**0**	
$\{4,7\}$	5	4.12	2.23	**0**

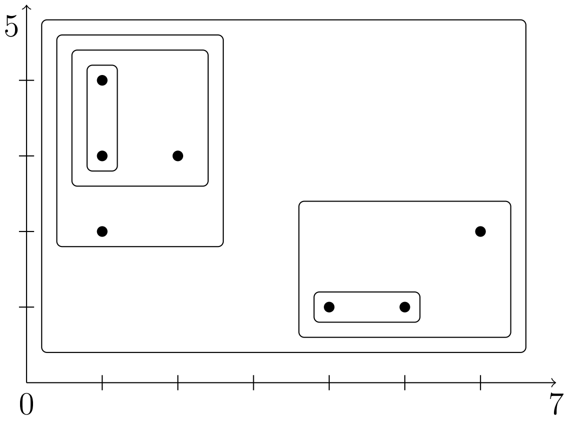
We now merge the 'clusters' $\{3\}$ and $\{1,2,5\}$:

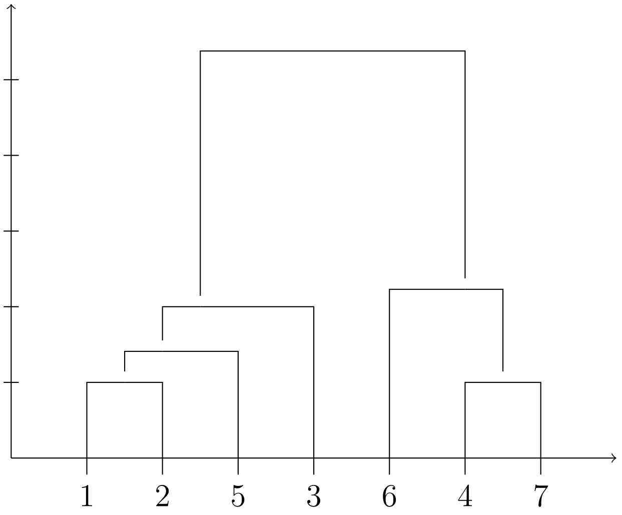
	$\{1,2,3,5\}$	$\{6\}$	$\{4,7\}$
$\{1,2,3,5\}$	**0**		
$\{6\}$	5.38	**0**	
$\{4,7\}$	5	2.23	**0**

We now merge the 'clusters' $\{6\}$ and $\{4,7\}$:

	$\{1,2,3,5\}$	$\{4,6,7\}$
$\{1,2,3,5\}$	**0**	
$\{4,6,7\}$	5.38	**0**

After merging the clusters $\{1, 2, 3, 5\}$ and $\{4, 6, 7\}$, we obtain the following result:





2 Group Assignment

1. *Explore manually the website http://sofifa.com. Under the tab 'All players', press on the Argentinian flag. Notice how the URL of the opened webpage changes to http://sofifa.com/players?na=52. Scrolling down, notice that not all players fit in one page. If you press 'Next', the new URL is http://sofifa.com/players?na=52&offset=100. Can you see the pattern? Next select an individual player and notice how the URL changes. We want to download the numerical attributes available for all 300 Argentinian players.*

By looking at a few pages, we see that `na=x` in the URL refers to the nationality, and that Argentina corresponds to `x=52`. Using `offset=x`, we can retrieve all players with numbers `x`, . . . , `x` + 100.

2. *Explain in detail the code below. In order to better understand the code, you may want to look at the following websites:*
   - *https://www.crummy.com/software/BeautifulSoup/*
   - *http://www.aivosto.com/vbtips/regex.html*
   - *https://docs.python.org/2/library/re.html*

We go through the code step by step:

```
import pandas as pd
from bs4 import BeautifulSoup
import requests
import re
import unicodedata
```

These command import the employed libraries.

```
attributes = ['Crossing', 'Finishing', 'Heading␣Accuracy',
 'Short␣Passing', 'Volleys', 'Dribbling','Curve',
 'Free␣Kick␣Accuracy', 'Long␣Passing', 'Ball␣Control', 'Acceleration',
 'Sprint␣Speed', 'Agility', 'Reactions', 'Balance',
 'Shot␣Power', 'Jumping', 'Stamina', 'Strength',
 'Long␣Shots', 'Aggression', 'Interceptions', 'Positioning',
 'Vision', 'Penalties', 'Composure', 'Marking',
 'Standing␣Tackle', 'Sliding␣Tackle', 'GK␣Diving',
 'GK␣Handling', 'GK␣Kicking', 'GK␣Positioning', 'GK␣Reflexes']
```

These commands define a list that contains all attributes of interest.

```
links = []   # Download data for all 300 Argentinian players
for offset in ['0', '100', '200']:
    page = requests.get ('http://sofifa.com/players?na=52&offset=' + offset)
    soup = BeautifulSoup (page.content, 'html.parser')
    for link in soup.find_all ('a'):
        links.append (link.get ('href'))
links = ['http://sofifa.com' + l for l in links if 'player/' in l]
```

Here we loop over 3 pages as we want to retrieve 300 players, that is, 100 players per page. We use the `requests` package to obtain the HTML pages. The library `BeautifulSoup` is useful for extracting from each page all the player URLs. You can use

```
print soup.prettify()
```

to explore the object. The code above extracts all the URLs found within the `<a>` tags of an HTML page. There are some URLs that do not point to players: We filter them in the last line, where we also make the URLs absolute by adding '`http://sofifa.com`'.

```
# pattern for regular expression
pattern = r"""\s*([\w\s]*)"""   # file starts with empty spaces... players
   name...-other stuff
for attr in attributes:
    pattern += r""".*?(\d*\s*"""+attr+r""")"""  # for each attribute we have
       other stuff...number...attribute...other stuff
pat = re.compile (pattern, re.DOTALL)    # parsing multiline text
```

This code block constructs a regular expression pattern that we need when looping over the players' webpages. We will further comment on this below.

```
rows = []
for j, link in enumerate (links):
    print j, link
    row = [link]
```

We loop over the accumulated links, one for every player. The purpose of the loop is to store the link, the player's name and all the attribute values for each player. We start by storing the link that is already available.

```
    playerpage = requests.get (link)
    playersoup = BeautifulSoup (playerpage.content, 'html.parser')
    text = playersoup.get_text()
    text = unicodedata.normalize ('NFKD', text).encode ('ascii', 'ignore')
```

Here we retrieve the HTML page for the player. The tag information is not very helpful, therefore we convert it to normal text and we are going to parse it using regular expressions. The last line replaces accented Latin characters that appear in some players' names with English letters. Such tedious steps are often necessary when scraping websites, emails and other forms of unstructured data. Getting things to work inevitably require a trial-and-error process. Running the commands from an interpreter for a specific player (link), you can explore the text with

```
print text
```

The output is of the following form:

```
Lionel Messi - FIFA 17 - Feb 14, 2017 - SoFIFA
.......
......
77 Crossing

95 Finishing

71 Heading Accuracy
.....
```

The output is fairly consistent for all players. We note that the text starts with the player's name, followed by a '-'. At some point later in the file we get the pattern of a number followed by the attributes of interest. The pattern we have constructed before the loop captures the required fields:

```
\s*([\w\s]*)
```

This part of the pattern matches the name:

- `\s*` matches one or more spaces (including new lines).
- `([\w\s]*)` captures as much of the text as we can with only letters and spaces. The parentheses are used to capture what is matched inside the parenthesis. Since the name in the text is followed by the character '-', this will do.

Looping over all attributes, we append to the pattern strings such as

```
.*?(\d*\s*Crossing)
```

The `.*?` matches anything in a non-greedy fashion. The '.' stands for any character, the '*' for one or more, and the '?' does the matching in a non-greedy way, that is, it will match as little of the text as possible, as long as what follows matches `\d*\s*Crossing`. For this latter expression, the parenthesis serves to capture what is inside it. Inside it, we match any number of digits followed by any number of spaces and the attribute '`Crossing`'. We loop over all attributes to construct the full pattern.

```
    a = pat.match (text)
    row.append (a.group (1))
    for i in range(2,len(attributes)+2):
        row.append (int (a.group(i).split()[0]))
    rows.append (row)
    print row[1]
```

Here we apply the pattern to the text. The returned object has stored the captured text matched inside the parentheses of our pattern, and we can access it trough the `group` functions. For Messi, ‘`group (1)`’ will have ’Lionel Messi ’, ‘`group (2)`’ will have ’77 Crossing’, ‘`group (3)`’ will have ’95 Finishing’ and so on. We use `split` to keep only the numbers.

```
df = pd.DataFrame (rows, columns = ['link', 'name'] + attributes)
df.to_csv ('ArgentinaPlayers.csv', index = False)
```

Here we store the data in a Pandas `DataFrame` and save it as a CSV file.

3. *How would you change the code to download the first 500 English players instead?*

To download the first 500 English players instead, we need to change the code block

```
for offset in ['0', '100', '200']:
    page = requests.get ('http://sofifa.com/players?na=52&offset=' + offset)
```

to

```
for offset in ['0', '100', '200', '300', '400']:
    page = requests.get ('http://sofifa.com/players?na=14&offset=' + offset)
```

4. *Use the **`sklearn.cluster.KMeans`** Python class to cluster the players into 5 clusters.*

This can be achieved with the following code:

```
from sklearn.cluster import KMeans
import numpy as np
X = np.array (df[attributes])
kmeans = KMeans (n_clusters = 5, random_state = 0)
kmeans.fit (X)
df['label'] = pd.Series (kmeans.labels_,  index = df.index)
```

5. *By inspecting the clusters and looking up individual players online, try to assign meaningful labels to the clusters.*

To print the names of the players in cluster 0, we type

```
df[df.label == 0].name
```

The first few players in cluster 0 are

```
1           Gonzalo Higuain
9              Mauro Icardi
45             Lucas Alario
55              Marco Ruben
59           Nicolas Blandi
61          Dario Benedetto
62              Gustavo Bou
78            Mauro Boselli
95          Franco Di Santo
100        Jonathan Calleri
```

```
116          Silvio Romero
118          Emiliano Sala
124      Maximiliano Lopez
131           German Denis
140       Facundo Ferreyra
143      Sebastian Driussi
153            Julio Furch
157           Lucas Viatri
158         Guido Carrillo
165              Jose Sand
168       Enrique Triverio
171     Juan Ignacio Gomez
172         Leonardo Ulloa
...
```

A web search reveals that most of these players are strikers, so we assign to the cluster 0 the label 'Strikers'. Cluster 1 consists of the players

```
12          Geronimo Rulli
31           Sergio Romero
36        Marcelo Barovero
43           Nahuel Guzman
44         Willy Caballero
53       Sebastian Torrico
54       Agustin Marchesin
64         Mariano Andujar
66           Franco Armani
83           Agustin Orion
105        Mariano Barbosa
108       Fernando Monetti
123             German Lux
126        Albano Bizzarri
149     Juan Pablo Carrizo
164   Cristian Campestrini
175         Guillermo Sara
184       Luciano Pocrnjic
193            Rodrigo Rey
204            Marcos Diaz
215          Javier Garcia
229            Jorge Broun
234           Oscar Ustari
242         Julian Speroni
253           Luis Ardente
291        Nereo Fernandez
```

who (according to a web search) are all goalkeepers. Cluster 2 starts as follows:

```
7              Ever Banega
8        Javier Mascherano
11          Ezequiel Garay
13             Marcos Rojo
14         Mateo Musacchio
16          Pablo Zabaleta
```

```
17              Lucas Biglia
18         Augusto Fernandez
19           Roberto Pereyra
22             Claudio Yacob
27          Cristian Ansaldi
35            Nicolas Pareja
47             Guido Pizarro
48              Marcos Acuna
49                Enzo Perez
57             David Abraham
65               Pablo Perez
67              Lucas Castro
69           Leandro Paredes
75         Esteban Cambiasso
76              Gino Peruzzi
77             Fernando Gago
81         Facundo Roncaglia
82              Emmanuel Mas
84        Matias Kranevitter
87               Oscar Trejo
89         Ramiro Funes Mori
....
```

These players are mostly defensive midfielders and full backs. Cluster 3 starts as follows:

```
6           Nicolas Otamendi
15         Gonzalo Rodriguez
24            Federico Fazio
25            Gustavo Cabral
28        Federico Fernandez
29            Lisandro Lopez
41             Victor Cuesta
46         Martin Demichelis
52       Santiago Gentiletti
63          Nicolas Burdisso
68           German Pezzella
72          Mauro Dos Santos
88           Jonatan Maidana
93          Santiago Vergini
106           Matias Caruzzo
109            Luciano Lollo
111         Martin Mantovani
113    Julio Alberto Barroso
120         Jonathan Schunke
125           Nicolas Spolli
137          Marcos Angeleri
139           Renato Civelli
145         Juan Insaurralde
148        Carlos Izquierdoz
152          Matias Zaldivia
159       Jose Maria Basanta
166         Leandro Desabato
```

```
174            Juan Forlin
177         Fernando Tobio
...
```

These players are almost exclusively central defenders. Finally, cluster 4 starts as follows:

```
0                  Lionel Messi
2                 Sergio Aguero
3                Angel Di Maria
4                  Paulo Dybala
5                Nicolas Gaitan
10               Javier Pastore
20                  Erik Lamela
21             Alejandro Gomez
23                Diego Perotti
26          Fernando Belluschi
30               Manuel Lanzini
32               Luciano Vietto
33                Pablo Batalla
34               Lisandro Lopez
37              Rodrigo Palacio
38                 Angel Correa
39               Ignacio Piatti
40               Eduardo Salvio
42                    Jose Sosa
50                 Diego Valeri
51                 Pablo Piatti
56             Diego Buonanotte
58           Rogelio Funes Mori
60              Pablo De Blasis
70                 Franco Cervi
71                 Mauro Zarate
73               Lautaro Acosta
74             Sebastian Blanco
...
```

These players are mostly playmakers/attacking midfielders, or forwards that like to move out of the box. To summarize, we obtain the following clusters:

Cluster	**Description**
0	Strikers
1	Goalkeepers
2	Defensive midfielders and full backs
3	Central defenders
4	Playmakers/attacking midfielders, outside forwards

It is interesting that KMeans has found side backs to be closer related to defensive midfielders than central defenders. Also, there was a clear distinction between strikers and other types of forwards that were grouped together with attacking midfielders.

6. *For a new and unknown player, the following attributes are available: (...) For each of your 5 clusters from Step 4, compute the cluster centroid. Assign the new player to the*

nearest cluster based on the distance to the cluster centroids, using only the available attributes.

To decide which cluster to assign the new player to, we execute the following code:

```python
centers = kmeans.cluster_centers_
attributesofinterest = ['Crossing', 'Sprint Speed', 'Long Shots', 'Aggression'
    , 'Marking', 'Finishing', 'GK Handling']
indices = [i for i in range (len (attributes)) if attributes[i] in
    attributesofinterest]
centersofinterest = centers[:, indices]
playerattributes = np.array ([45, 40, 35, 45, 60, 40, 15])
for cluster in range (5):
    dis = np.sqrt (((centersofinterest[cluster, :] - playerattributes) ** 2).
        sum())
    print cluster, dis
```

We obtain the following result:

```
0 59.2088045066
1 86.925706223
2 57.3034127353
3 46.3560352058
4 64.7386383339
```

Our best guess is therefore that the player is a central defender.