Project Course 4_Pokemon_Final

June 14, 2021

1 Pokemon type Clustering

1.1 1. Main objectives

The main objective of this work is to clusterise the different types of Pokemons in order to finally observe if one of these kinds present better features than the others. We already have some Pokemon classification as we will see in the data description, but we will drop these corresponding features and try to infer it without knowing.

Data has been obtained from https://www.kaggle.com/abcsds/pokemon .

1.2 2. Description of the data

The data is imported and the first rows are also shown.

[1]:
```python
import pandas as pd, seaborn as sns, matplotlib.pyplot as plt, numpy as np

filepath = 'Pokemon.csv'

data = pd.read_csv(filepath, sep = ',')

data.head()
```

[1]:
```
   number                 name  type1   type2  total  hp  attack  defense  \
0       1            Bulbasaur  Grass  Poison    318  45      49       49
1       2              Ivysaur  Grass  Poison    405  60      62       63
2       3             Venusaur  Grass  Poison    525  80      82       83
3       3        Mega Venusaur  Grass  Poison    625  80     100      123
4       3  Gigantamax Venusaur  Grass  Poison    525  80      82       83

   sp_attack  sp_defense  speed  generation  legendary
0         65          65     45           1      False
1         80          80     60           1      False
2        100         100     80           1      False
3        122         120     80           1      False
4        100         100     80           1      False
```

Each column corresponds to the following:

Number: The ID for each pokemon

Name: The name of each pokemon

Type 1: Each pokemon has a type, this determines weakness/resistance to attacks

Type 2: Some pokemon are dual type and have 2

Total: Sum of all stats that come after this, a general guide to how strong a pokemon is

HP: Hit points, or health, defines how much damage a pokemon can withstand before fainting

Attack: The base modifier for normal attacks (eg. Scratch, Punch)

Defense: The base damage resistance against normal attacks

SP Atk: Special attack, the base modifier for special attacks (e.g. fire blast, bubble beam)

SP Def: Special defense, the base damage resistance against special attacks

Speed: Determines which pokemon attacks first each round

Generation: The generation of games where the pokemon was first introduced

Legendary: Some pokemon are much rarer than others, and are dubbed "legendary"

We have 13 features and 1072 rows. Le'ts also take some extra information. For the *type2* feature he have a large number of null values.

[2]:
```
data.shape
```

[2]:
```
(1072, 13)
```

[3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1072 entries, 0 to 1071
Data columns (total 13 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   number      1072 non-null   int64
 1   name        1072 non-null   object
 2   type1       1072 non-null   object
 3   type2       574 non-null    object
 4   total       1072 non-null   int64
 5   hp          1072 non-null   int64
 6   attack      1072 non-null   int64
 7   defense     1072 non-null   int64
 8   sp_attack   1072 non-null   int64
 9   sp_defense  1072 non-null   int64
 10  speed       1072 non-null   int64
 11  generation  1072 non-null   int64
 12  legendary   1072 non-null   bool
```

```
dtypes: bool(1), int64(9), object(3)
memory usage: 101.7+ KB
```

1.3 3. Feature engineering

In order to do clustering, we will remove the **type1**, **type2**, **name**, **numberID**, **legendary** and **generation**. Except *name* and *numberID*, they are categorical data, and we are not allowed to work with them. We will only work with continuous data.

[7]:
```
data = data.drop(['name', 'number', 'type1', 'type2', 'legendary',␣
↪'generation'], axis = 1)
```

[8]:
```
data
```

[8]:

	total	hp	attack	defense	sp_attack	sp_defense	speed
0	318	45	49	49	65	65	45
1	405	60	62	63	80	80	60
2	525	80	82	83	100	100	80
3	625	80	100	123	122	120	80
4	525	80	82	83	100	100	80
…	…	…	…	…	…	…	…
1067	580	100	145	130	65	110	30
1068	580	100	65	60	145	80	130
1069	500	100	80	80	80	80	80
1070	680	100	165	150	85	130	50
1071	680	100	85	80	165	100	150

```
[1072 rows x 7 columns]
```

Let's first take a look to the correlations:

[9]:
```
corr_mat = data.corr()

for x in range(len(data.columns)):
    corr_mat.iloc[x,x] = 0

corr_mat.abs().idxmax()
```

[9]:
```
total          attack
hp              total
attack          total
defense         total
sp_attack       total
sp_defense      total
speed           total
dtype: object
```

We observe that the general indicator of how strong a pokemon is (*total*) is highly correlated with the power *attack*.

Now, the **skew of each feature is evaluated**, and we will correct it by applying the logarithm for those who have skew > 0.75.

[10]:
```
skew_columns = data.skew().sort_values(ascending = False)

skew_columns = skew_columns.loc[skew_columns > 0.75]
skew_columns
```

[10]:
```
hp            1.760494
defense       1.143146
sp_defense    0.926515
dtype: float64
```

[11]:
```
for col in skew_columns.index:
    data[col] = np.log1p(data[col])
```

Now all the features are reescaled using the **MinMaxScaler**.

[12]:
```
from sklearn.preprocessing import MinMaxScaler

float_cols = [x for x in data.columns]

mm = MinMaxScaler()

data[float_cols] = mm.fit_transform(data[float_cols])
```

Once we have done with the sclaning, we can check the correlation with a pairplot. We will also differentiate by the **legendary** feature, to see if there is a kind of clustering. The **type1** and **type2** features have also been checked before and it is observed a random distribution in the correlation plot.

[13]:
```
sns.set_context('notebook')

data['legendary'] = df['legendary']
sns.pairplot(data[float_cols + ['legendary']], hue = 'legendary')
```

[13]: <seaborn.axisgrid.PairGrid at 0x7f8b192f16d0>

A kind of clustering is observed for the **legendary** feature.

[14]:
```python
data = data.drop('legendary', axis = 1)
```

[15]:
```python
data
```

[15]:

	total	hp	attack	defense	sp_attack	sp_defense	speed
0	0.150526	0.646223	0.237838	0.567873	0.298913	0.461574	0.205128
1	0.242105	0.704391	0.308108	0.633990	0.380435	0.544121	0.282051
2	0.368421	0.762836	0.416216	0.706822	0.489130	0.633068	0.384615
3	0.473684	0.762836	0.513514	0.811133	0.608696	0.705892	0.384615
4	0.368421	0.762836	0.416216	0.706822	0.489130	0.633068	0.384615
…	…	…	…	…	…	…	…
1067	0.426316	0.808316	0.756757	0.825841	0.298913	0.671122	0.128205
1068	0.426316	0.808316	0.324324	0.621131	0.733696	0.544121	0.641026

```
1069  0.342105  0.808316  0.405405  0.697082   0.380435    0.544121  0.384615
1070  0.531579  0.808316  0.864865  0.863895   0.407609    0.737898  0.230769
1071  0.531579  0.808316  0.432432  0.697082   0.842391    0.633068  0.743590

[1072 rows x 7 columns]
```

1.4 4. Three variations of unsupervised model (Clustering)

In this section we apply 3 different models of clustering. After that we will evaluate which of them performs better.

1.4.1 4.1 K-means

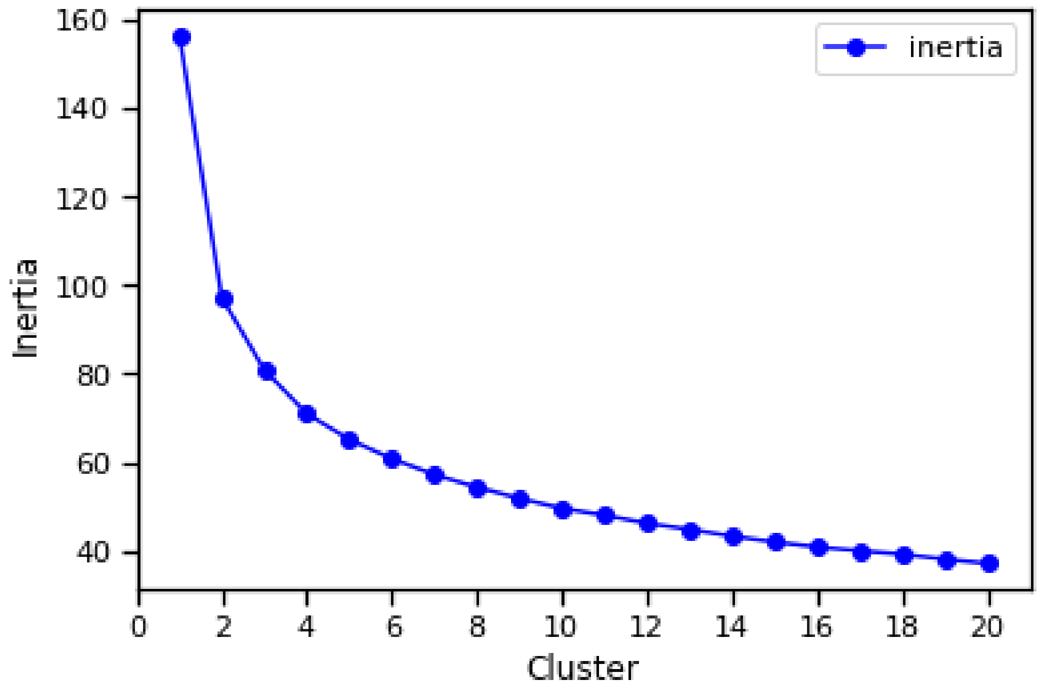
The **K-means** model is applyed for thefferent number of clusters. For each one of them the inertia is computed, and at this point the elbow rule is used in order to take the best number of **k**.

[16]:
```
from sklearn.cluster import KMeans
km_list = list()

for clust in range(1,21):
    km = KMeans(n_clusters=clust, random_state= 42)
    km = km.fit(data)
    km_list.append(pd.Series({'clusters': clust, 'inertia':km.inertia_, 'model':
 ↪ km}))
```

[17]:
```
plot_data = pd.concat(km_list, axis = 1).T
plot_data = plot_data[['clusters', 'inertia']].set_index('clusters')
plot_data.head(4)
```

[17]:
```
          inertia
clusters
1         156.274
2         97.0283
3         80.7911
4         70.9049
```

[18]:
```
ax = plot_data.plot(marker = 'o', color = 'blue')
ax.set_xticks(range(0,21,2))
ax.set_xlim(0,21)
ax.set(xlabel='Cluster', ylabel='Inertia');
```

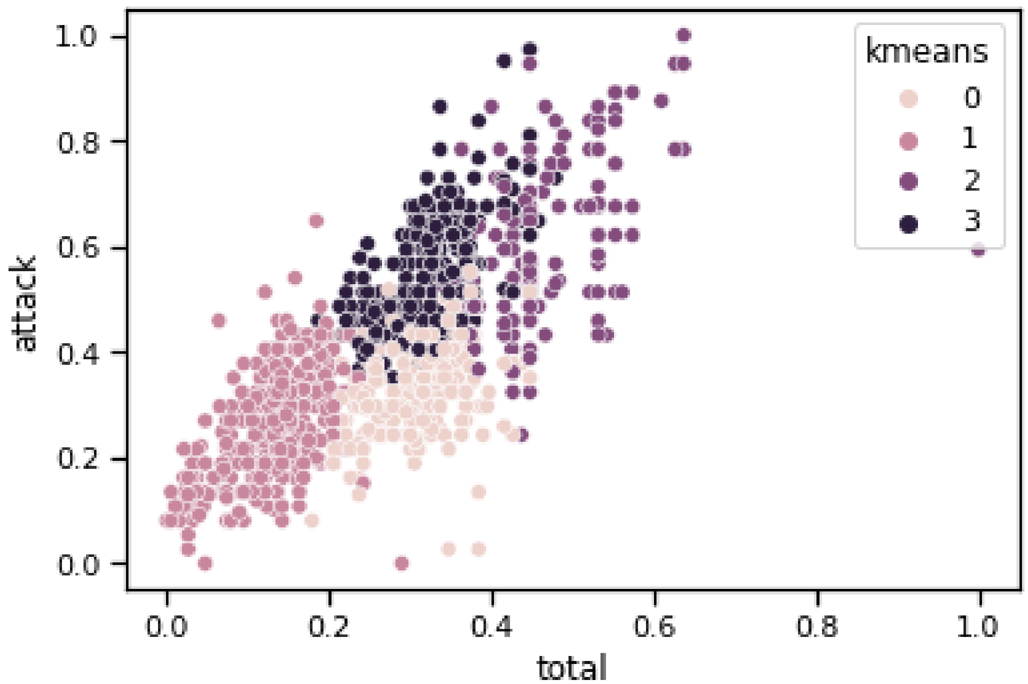


We will take $\mathbf{k = 4}$.

[19]:
```
km = KMeans(n_clusters=4, random_state= 42)
km = km.fit(data[float_cols])
data['kmeans'] = km.predict(data[float_cols])
```

[20]:
```
sns.scatterplot(data=data, x="total", y="attack", hue="kmeans")
```

[20]: <AxesSubplot:xlabel='total', ylabel='attack'>



The algorithm differentiate 4 considerably good clusters. Visually we might only have taken 2 clusters, but we really do not know how many clusters there are in this data.

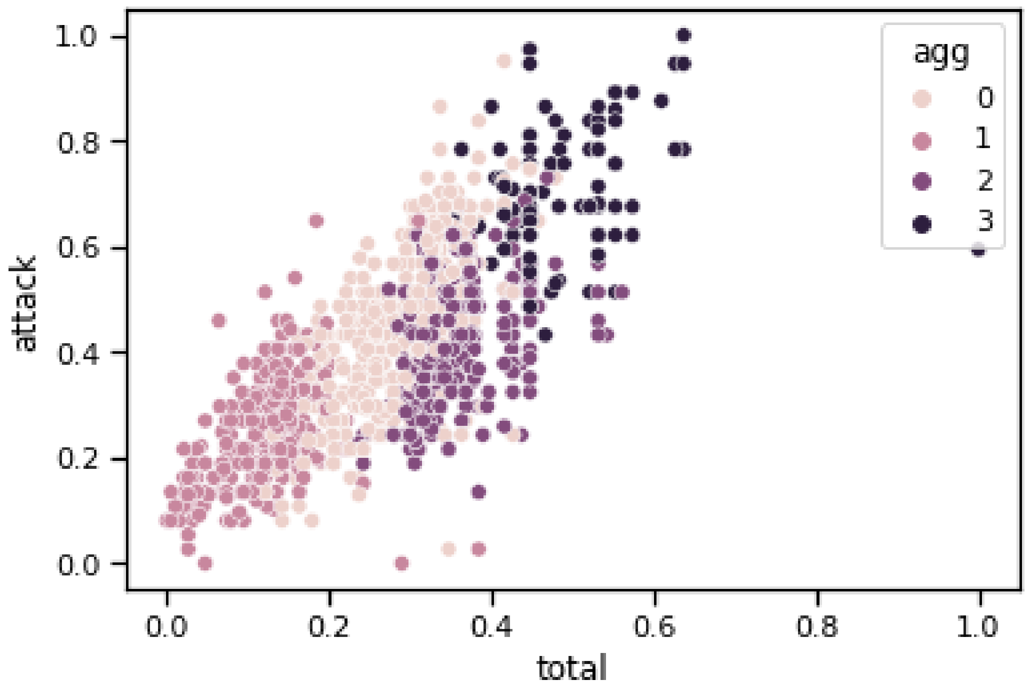
1.4.2 4.2 Agglomerative Clustering

Now the Agglomerative Clustering is applyied for the same number of clusters **k = 4**.

[21]:
```
from sklearn.cluster import AgglomerativeClustering

agg = AgglomerativeClustering(n_clusters=4, linkage='ward',␣
↪compute_full_tree=True)
agg = agg.fit(data[float_cols])

data['agg'] = agg.fit_predict(data[float_cols])
```

[22]:
```
sns.scatterplot(data=data, x="total", y="attack", hue="agg")
```

[22]: <AxesSubplot:xlabel='total', ylabel='attack'>



The cluster differentiation in not that clear.

1.4.3 4.3 DBSCAN

DBSCAN is applyied for **k=4**

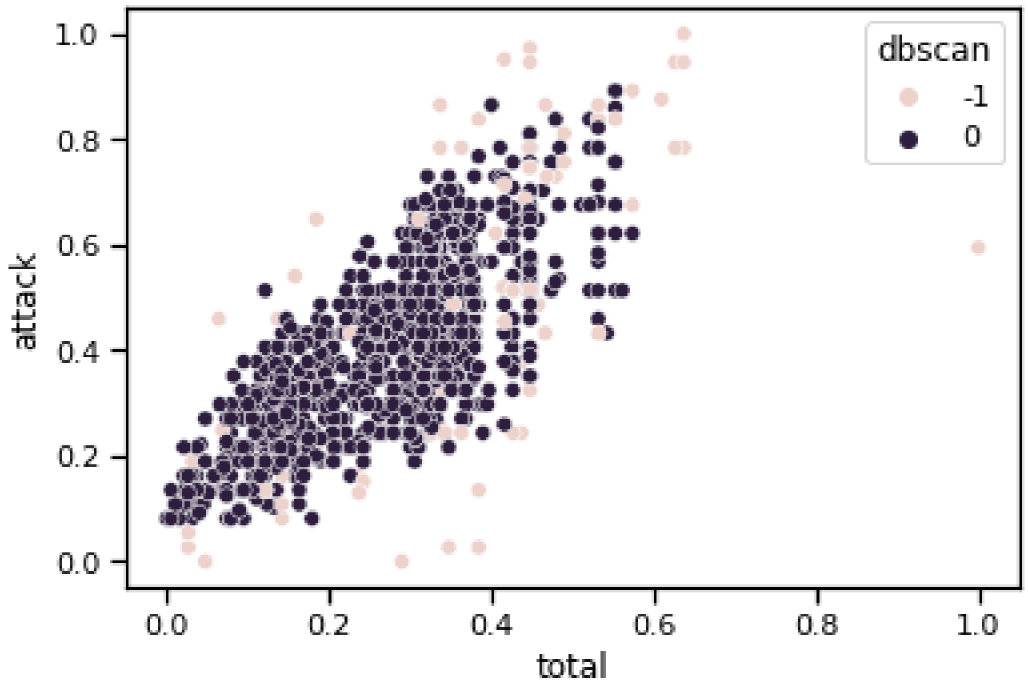
[23]:
```
from sklearn.cluster import DBSCAN
from sklearn import metrics
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler

db = DBSCAN(eps=0.2, min_samples=15).fit(data[float_cols])
clustering_labels = db.fit_predict(data[float_cols])

data['dbscan'] = clustering_labels
```

[24]:
```
data['dbscan'].value_counts()
```

[24]:
```
 0    986
-1     86
Name: dbscan, dtype: int64
```

[25]:
```
sns.scatterplot(data=data, x="total", y="attack", hue="dbscan")
```

[25]: <AxesSubplot:xlabel='total', ylabel='attack'>



Choosing the parameters is really complicated. Nonetheless, visually we could see that at least we have 2 clusters, but DBSCAN is not capable of determining them well, this is because **we have different cluster densities**.

1.5 5. Which model is recommended?

In this case I would recommend **k-means**. It is capable of differentiationg clusters quite well, and with different cluster densities.

The DBSCAN is complicated to apply since density of points is not uniform, and choosing parameters is not trivial either. Very quickly we obtain a large number of clusters, or directly simply one just like I have shown in the last figure.

1.6 6. Future suggestions

In the future may be different distances for the Agglomerative Clustering method could be checked, and also a grid cross validation for parameters of DSCAN as well. Nonethess, in this example not very good results would have been obtained.

[ ]:

```

```