3_Train and evaluate a clustering model

October 15, 2021

1 Clustering - Introduction

In contrast to *supervised* machine learning, *unsupervised* learning is used when there is no "ground truth" from which to train and validate label predictions. The most common form of unsupervised learning is *clustering*, which is simllar conceptually to *classification*, except that the the training data does not include known values for the class label to be predicted. Clustering works by separating the training cases based on similarities that can be determined from their feature values. Think of it this way; the numeric features of a given entity can be thought of as vector coordinates that define the entity's position in n-dimensional space. What a clustering model seeks to do is to identify groups, or *clusters*, of entities that are close to one another while being separated from other clusters.

For example, let's take a look at a dataset that contains measurements of different species of wheat seed.

> **Citation**: The seeds dataset used in the this exercise was originally published by the Institute of Agrophysics of the Polish Academy of Sciences in Lublin, and can be downloaded from the UCI dataset repository (Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science).

[ ]:
```
import pandas as pd

data = pd.read_csv('seeds.csv')

# Display a random sample of 10 observations (just the features)
features = data[data.columns[0:6]]
features.sample(10)
```

[ ]:
```
      area  perimeter  compactness  kernel_length  kernel_width  \
145  11.21      13.13       0.8167          5.279         2.687
49   14.86      14.67       0.8676          5.678         3.258
208  11.84      13.21       0.8521          5.175         2.836
139  16.23      15.18       0.8850          5.872         3.472
199  12.76      13.38       0.8964          5.073         3.155
70   17.63      15.98       0.8673          6.191         3.561
102  19.46      16.50       0.8985          6.113         3.892
137  15.57      15.15       0.8527          5.920         3.231
92   18.81      16.29       0.8906          6.272         3.693
```

```
30   13.16      13.82       0.8662          5.454          2.975

     asymmetry_coefficient
145                 6.1690
49                  2.1290
208                 3.5980
139                 3.7690
199                 2.8280
70                  4.0760
102                 4.3080
137                 2.6400
92                  3.2370
30                  0.8551
```

As you can see, the dataset contains six data points (or *features*) for each instance (*observation*) of a seed. So you could interpret these as coordinates that describe each instance's location in six-dimensional space.

Now, of course six-dimensional space is difficult to visualise in a three-dimensional world, or on a two-dimensional plot; so we'll take advantage of a mathematical technique called *Principal Component Analysis* (PCA) to analyze the relationships between the features and summarize each observation as coordinates for two principal components - in other words, we'll translate the six-dimensional feature values into two-dimensional coordinates.

[ ]:
```
features.head()
```

[ ]:
```
    area  perimeter  compactness  kernel_length  kernel_width  \
0  15.26      14.84       0.8710          5.763         3.312
1  14.88      14.57       0.8811          5.554         3.333
2  14.29      14.09       0.9050          5.291         3.337
3  13.84      13.94       0.8955          5.324         3.379
4  16.14      14.99       0.9034          5.658         3.562

   asymmetry_coefficient
0                  2.221
1                  1.018
2                  2.699
3                  2.259
4                  1.355
```

[ ]:
```
data.columns[0:6]
```

[ ]:
```
Index(['area', 'perimeter', 'compactness', 'kernel_length', 'kernel_width',
       'asymmetry_coefficient'],
      dtype='object')
```

[ ]:
```
features[data.columns[0:6]]
```

[ ]:
```
      area  perimeter  compactness  kernel_length  kernel_width  \
0    15.26      14.84       0.8710          5.763         3.312
1    14.88      14.57       0.8811          5.554         3.333
2    14.29      14.09       0.9050          5.291         3.337
3    13.84      13.94       0.8955          5.324         3.379
4    16.14      14.99       0.9034          5.658         3.562
..     …          …           …             …            …
205  12.19      13.20       0.8783          5.137         2.981
206  11.23      12.88       0.8511          5.140         2.795
207  13.20      13.66       0.8883          5.236         3.232
208  11.84      13.21       0.8521          5.175         2.836
209  12.30      13.34       0.8684          5.243         2.974

     asymmetry_coefficient
0                    2.221
1                    1.018
2                    2.699
3                    2.259
4                    1.355
..                     …
205                  3.631
206                  4.325
207                  8.315
208                  3.598
209                  5.637

[210 rows x 6 columns]
```

[ ]:
```
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA

# Normalize the numeric features so they're on the same scale
scaled_features = MinMaxScaler().fit_transform(features[data.columns[0:6]])

# Get two principal components
pca = PCA(n_components=2).fit(scaled_features)
features_2d = pca.transform(scaled_features)
features_2d[0:10]
```

[ ]:
```
array([[ 0.11883593, -0.09382469],
       [ 0.0696878 , -0.31077233],
       [-0.03499184, -0.37044705],
       [-0.06582089, -0.36365235],
       [ 0.32594892, -0.37695797],
       [-0.02455447, -0.31060184],
       [-0.00769646, -0.07594931],
       [-0.05646955, -0.26696284],
```

```
       [ 0.38196305, -0.05149471],
       [ 0.35701044, -0.17697998]])
```

Now that we have the data points translated to two dimensions, we can visualize them in a plot:

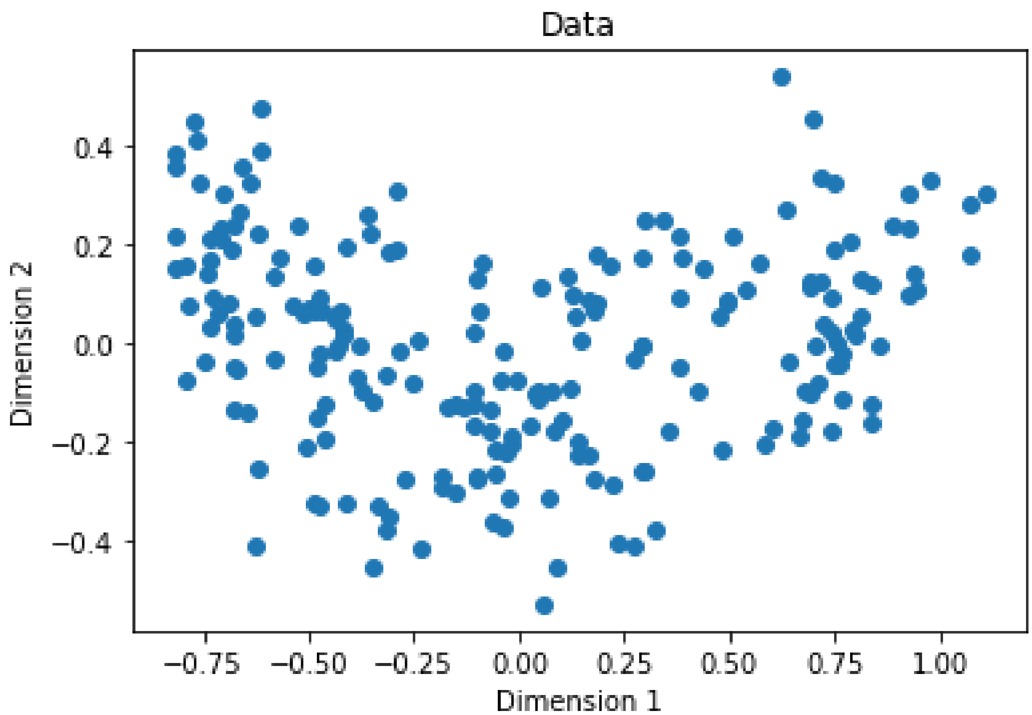
[ ]:
```
features_2d[0:5,0]# the first dimension
```

[ ]:
```
array([ 0.11883593,  0.0696878 , -0.03499184, -0.06582089,  0.32594892])
```

[ ]:
```
import matplotlib.pyplot as plt

%matplotlib inline

plt.scatter(features_2d[:,0],features_2d[:,1])
plt.xlabel('Dimension 1')
plt.ylabel('Dimension 2')
plt.title('Data')
plt.show()
```



Hopefully you can see at least two, arguably three, reasonably distinct groups of data points; but here lies one of the fundamental problems with clustering - without known class labels, how do you know how many clusters to separate your data into?

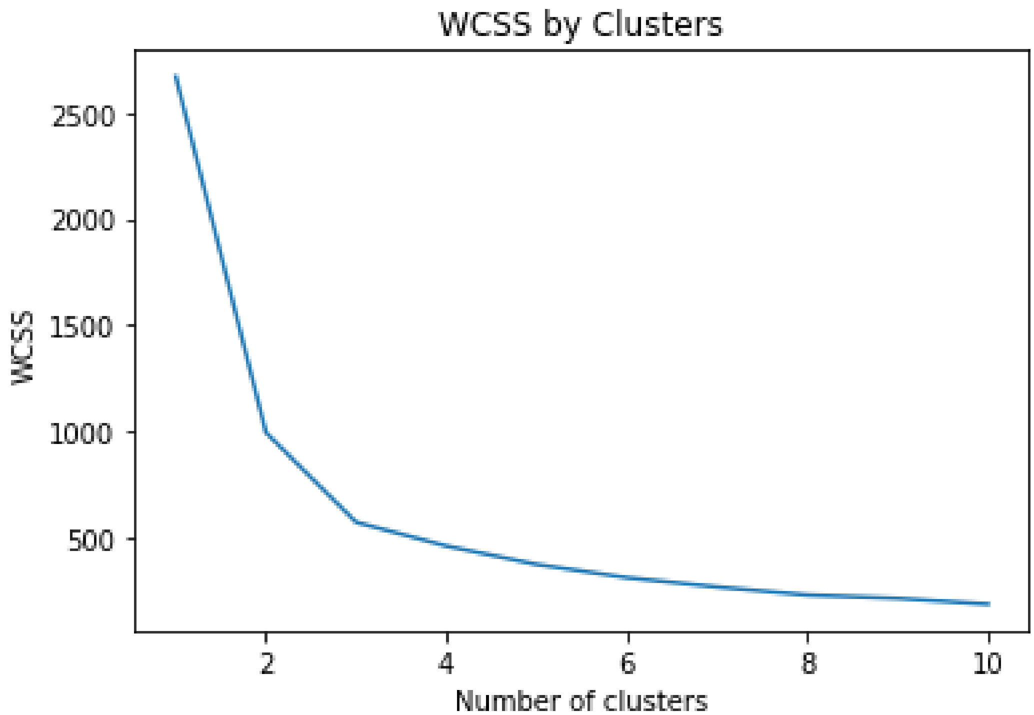
One way we can try to find out is to use a data sample to create a series of clustering models with an incrementing number of clusters, and measure how tightly the data points are grouped within each cluster. A metric often used to measure this tightness is the *within cluster sum of squares* (WCSS), with lower values meaning that the data points are closer. You can then plot the WCSS for each model.

[ ]:
```
# importing the libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
%matplotlib inline

# Create 10 models with 1 to 10 clusters
wcss =[]
for i in range(1,11):
    kmeans = KMeans(n_clusters=i)
    # Fit the data points
    kmeans.fit(features.values)
    # Get the WCSS (inertia) value
    wcss.append(kmeans.inertia_)

#Plot the WCSS values onto a line graph
plt.plot(range(1,11),wcss)
plt.title('WCSS by Clusters')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

```
C:\Users\aduzo\Anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:882:
UserWarning: KMeans is known to have a memory leak on Windows with MKL, when
there are less chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=1.
  f"KMeans is known to have a memory leak on Windows "
```



The plot shows a large reduction in WCSS (so greater *tightness*) as the number of clusters increases from one to two, and a further noticable reduction from two to three clusters. After that, the reduction is less pronounced, resulting in an "elbow" in the chart at around three clusters. This is a good indication that there are two to three reasonably well separated clusters of data points.

1.1 Summary

Here we looked at what clustering means, and how to determine whether clustering might be appropriate for your data. In the next notebook, we will look at two ways of labelling the data automatically.