Chapter 7:
Clustering

So far we have only looked at so called *supervised learning* settings. In these scenarios we are given

$$\begin{aligned} &\text{Inputs:} && x_1, ..., x_n \in \mathbb{R}^d \\ &\text{Labels:} && y_1, ..., y_n \\ & && y_i \in \begin{cases} \{-1, +1\} & \text{for binary classification} \\ \mathbb{R} & \text{for regression} \end{cases} \end{aligned}$$

Now we will take a look at what happens if no labels are given, i.e if $y_1, ..., y_n$ are unknown.

This setting is called *unsupervised learning* and we will start by looking at so called *clustering* problems.

7.1 Linear Clustering

Definition: Clustering Clustering is the process of organizing objects into groups -called clusters- whose members are similar in some way.

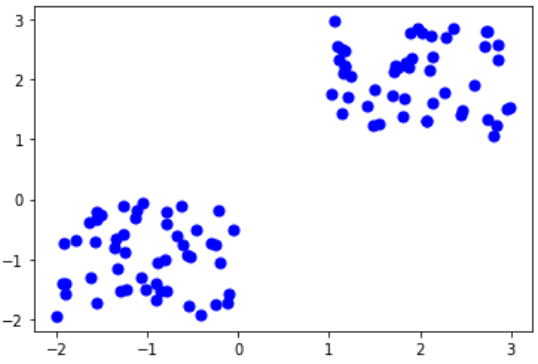

Figure 1: Two sets of points in $\mathbb{R}^2$ that could be clustered.

We are now looking for an algorithm that is capable of correctly clustering our inputs $x_1, ..., x_n$ in a way that allows us to make meaningful interpretations of the resulting clusters.

Example In figure 1 the two axes could represent the weight and tail length of two types of mice (transformed such as to fit neatly into the plot), which we want to distinguish from each other. In that case good algorithm should yield the following clusters

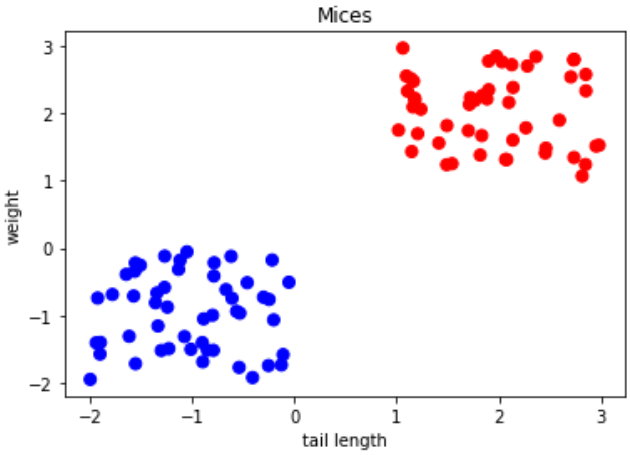

Figure 2: Clustered mices.

In the above scenario the clustering was easy to eyeball but since we don't know what the ground truth labels are we can't be sure that this is the optimal way to cluster our mice. Therefore the results usually need to be inspected manually afterwards.

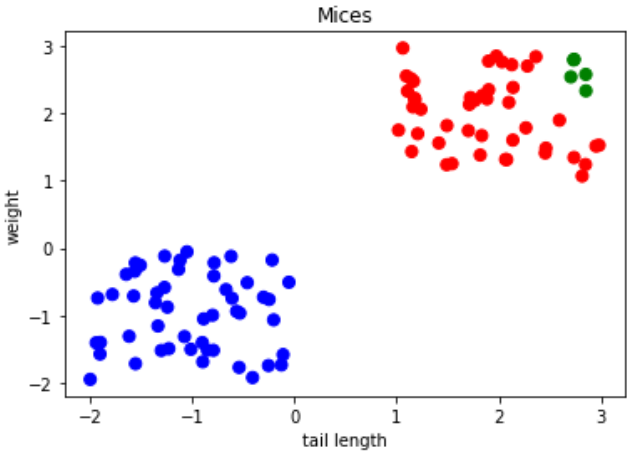

Figure 3: What an even better clustering could look like.

7.1.1 K-means

Now we will look at arguably the most popular clustering algorithm.

Algorithm K-means

Input: Kernel matrix $\mathbf{K}$,$k \in \mathbb{N}$ and $\mathbf{X} = \mathbf{x}_1, ..., \mathbf{x}_n \in \mathbb{R}^d$

1: **function** K-MEANS($\mathbf{K}$,k, $\mathbf{X}$)
2: Initialize cluster centers $\mathbf{c}_1, ..., \mathbf{c}_n$ (e.g., randomly drawn inputs $\mathbf{c}_{i_j} = \mathbf{x}_{i_j}$)
3: **repeat**
4: **for** i = 1 : n **do**
5: Label the input $\mathbf{x}_i$ as belonging to the nearest cluster

$$y_i = \arg\min_{j=1,...,k} \|\mathbf{x}_i - \mathbf{c}_j\|^2$$

6: **end for**
7: **for** j = 1 : k **do**
8: Compute cluster center $\mathbf{c}_j$ as the mean of all inputs of the j-th cluster,

$$\mathbf{c}_j := \text{mean}(\{\mathbf{x}_i : y_i = j\})$$

9: **end for**
10: **until** Clusters don't change between subsequent iterations
11: **return** Cluster centers $\mathbf{c}_1, ..., \mathbf{c}_k$
12: **end function**

The k-means algorithm takes two inputs, k the number of clusters we want to create and a set of inputs $x_1, ..., x_n \in \mathbb{R}$ which will usually be passed in form of a matrix $X \in \mathbb{R}^{n \times d}$ where d is the number of features each input x_i has ($d = 2$ in our mouse example).

We then initialize the cluster centers $\mathbf{c}_1, ..., \mathbf{c}_k$ by randomly picking k of our input data points $\mathbf{x_1}, ...\mathbf{x_n}$. Afterwards we repeat the procedure described in lines 4-9 until our clusters don't change anymore.

On the next page we have applied the algorithm to our mouse example the white crosses represent the centroids. The colored regions indicate to which cluster a new input $\mathbf{x}_{n'}$ would be assigned.

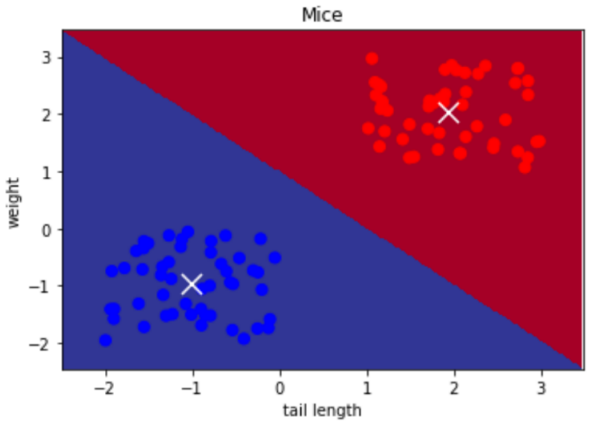

Figure 4: K-means clustering applied to our mouse example

Example: Limitations We will now see that k-means is limited to finding linear boundaries between classes. That means it will partition the feature space into polygons. This partition is also called *Voronoi diagram*. The polygons consist of a centroid and all points of the space that are closer to this particular centroid than to any other.

Theorem 7.1.2 K-means finds piecewise linear cluster boundaries

Proof:
We first look at case $k = 2$. Since we only have two clusters we can partition our inputs $\mathbf{x}_1, ..., \mathbf{x}_n$ by assigning the label $y_i = -1$ to every input that lies in cluster one and $y_i = 1$ to every input that lies in cluster two. Therefore we get the partition

$$A := \{x_i : (x_i, y_i) \in M, y_i = -1\} \quad B := \{x_i : (x_i, y_i) \in M, y_i = +1\}$$

From line 8 of the algorithm we know that the centroids can be computed as

$$\mathbf{c}_{-1} = \frac{1}{|A|} \sum_{x \in A} \mathbf{x}, \quad \mathbf{c}_{+1} = \frac{1}{|B|} \sum_{x \in B} \mathbf{x}$$

Look back at 1.2 Linear Classifier, where we presented the *nearest centroid classifier*. Our formulation here is equivalent to it. What happend is that for $k = 2$ k-means outputs the centroids for a NCC which we can then use to predict new points. This also means that the cluster boundaries have to be linear.

If $k > 2$ we look at pairwise linear boundaries between clusters and use the fact that the intersection of linear boundaries creates polygons which are piecewise linear. □

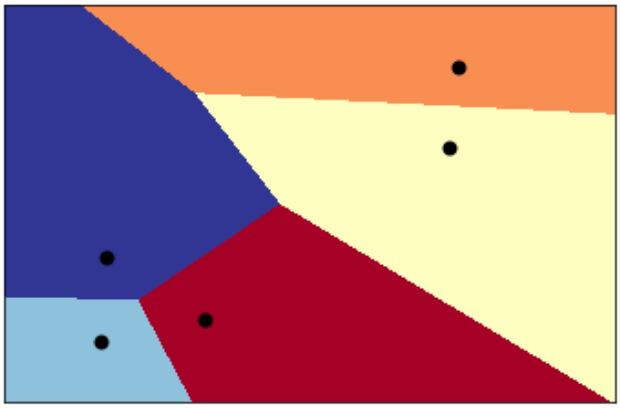

Figure 5: Voronoi Diagram of our mice example for $k = 5$

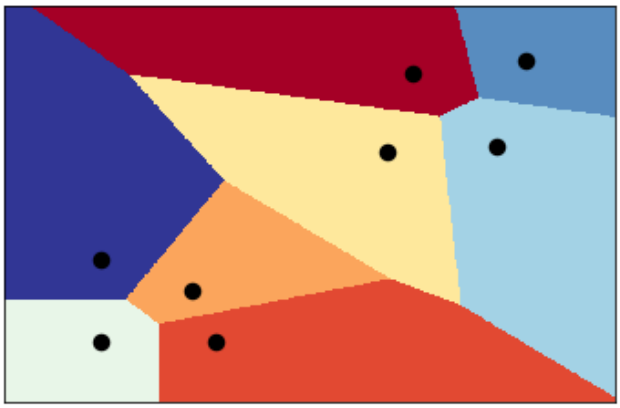

Figure 6: Voronoi Diagram of our mice example for $k = 8$

Next up we will see how to deal with clustering tasks where the optimal decision boundary isn't linear anymore.

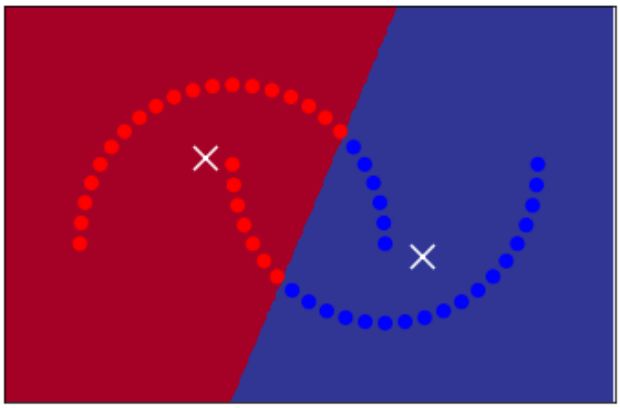

Figure 7: K-means can't cluster this data effectively.

7.2 Non-linear Clustering

7.2.1 Kernel k-means

Just like SVM the k-means clustering algorithm can be kernelized. Therefore we have to change two parts of our algorithm. First of all we will compute the norm $\|\mathbf{x}_i - \mathbf{c}_i\|^2$ line 5 via a kernel function and secondly we won't compute the mean in line 8 explicitly but rather find a proxy for it. We will now see how this works in detail.

Let $\phi : \mathbb{R}^d \to \mathcal{H}$ be a kernel feature map for a kernel $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ and $l_j := \{i \in \{1, \dots, n\} : y_i = j\}$ for all $j = 1, \dots, k$. Then

$$\mathbf{c}_j := \frac{1}{|I_j|} \sum_{i \in I_j} \phi(\mathbf{x}_i)$$

That is, we map all inputs $\mathbf{x}_1, ..., \mathbf{x}_n$ into a higher dimensional space via ϕ where they can be lineraly separated and then apply k-means in that space.

We can then completly express the norm $\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|^2$ in terms of kernel functions.

$$\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|^2 = \underbrace{\|\phi(\mathbf{x}_i)\|^2}_{=k(\mathbf{x}_i,\mathbf{x}_i)} - 2\langle\phi(\mathbf{x}_i), \mathbf{c}_j\rangle + \|\mathbf{c}_j\|^2 \tag{1}$$

where

$$\begin{aligned}
\|\mathbf{c}_j\|^2 &= \langle\frac{1}{|I_j|}\sum_{i\in I_j}\phi(\mathbf{x}_i), \frac{1}{|I_j|}\sum_{i'\in I_j}\phi(\mathbf{x}_i)\rangle \\
&= \frac{1}{|I_j|^2}\langle\sum_{i\in I_j}\phi(\mathbf{x}_i), \sum_{i'\in I_j}\phi(\mathbf{x}_i)\rangle \\
&= \frac{1}{|I_j|^2}\sum_{i,i'\in l_j} k(\mathbf{x}_i, \mathbf{x}_{i'})
\end{aligned}$$

Here we used the kernel property $k(\mathbf{x}, \tilde{\mathbf{x}}) = \langle\phi(\mathbf{x}), \phi(\tilde{\mathbf{x}})\rangle$ and the properties of dot products in general in order to move the sum and $\frac{1}{|I_j|^2}$ out of the dot product. Through an analogous calculation we get

$$\langle\phi(\mathbf{x}_i), \mathbf{c}_j\rangle = \frac{1}{|I_j|}\sum_{i'\in I_j} k(\mathbf{x}_i, \mathbf{x}_{i'})$$

Therefore (1) can be completely expressed in terms of kernel functions.

$$\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|^2 = k(\mathbf{x}_i, \mathbf{x}_i) - \frac{2}{|I_j|}\sum_{i'\in I_j} k(\mathbf{x}_i, \mathbf{x}_{i'}) + \frac{1}{|I_j|^2}\sum_{i,i'\in I_j} k(\mathbf{x}_i, \mathbf{x}_{i'}) \tag{2}$$

Since we usually can't compute $\phi(\cdot)$ explicitly we have to find a different way to assign $\mathbf{c}_j$ in the next step e.g. line 8.

The trick we are applying is to assign points to a cluster C_j directly (notice the capital non bold letter). Since we showed above that we don't need to know the cluster center $\mathbf{c}_j$ we can just skip its calculation in line 8 all together. What count's for the classification isn't a point $\mathbf{c}_j$ but rather the indices in I_j which then determine the value of the R.H.S of equation (2). We can say that I_j is basically a proxy for $\mathbf{c}_j$ which encodes the same information. That means which points should be clustered together. In case of I_j this works by explicitly giving their indices and in case of $\mathbf{c}_j$ by being able to calculate the distance from every input $x_1, ..., x_n$ and then choosing the centroid $\mathbf{c}_j$ which is nearest.

Algorithm Kernel k-means

Input: $k \in \mathbb{N}$ and $\mathbf{X} = \mathbf{x}_1, ..., \mathbf{x}_n \in \mathbb{R}^d$

1: **function** K-MEANS($k, \mathbf{X}$)
2: Randomly assign every $\mathbf{x}_i$ to one of the k clusters C_j, $j = 1, ..., k$

3: **repeat**
4: **for** i = 1 : n **do**
5: Label the input $\mathbf{x}_i$ as belonging to the nearest cluster C_j.

$$y_i = \arg\min_{j=1,...,k} \|\phi(\mathbf{x}_i) - \mathbf{c}_j\|^2$$

Thanks to (2) we don't have to know how $\mathbf{c}_j$ looks.

6: **end for**
7: **for** j = 1 : k **do**
8: Update clusters as

$$C_j = \{x_i : y_i = j\}$$

9: **end for**
10: **until** Clusters don't change between subsequent iterations

11: **return** Final clusters $C_1, ..., C_k$
12: **end function**

As we see the only part that really changes in comparison to k-means is line 8 where we computed each cluster as the set of inputs $\mathbf{x}_i$ whose label y_i is j. With this algorithm we can indeed improve our results drastically in some cases. Take a look at figure 6 again and then at figure 7.

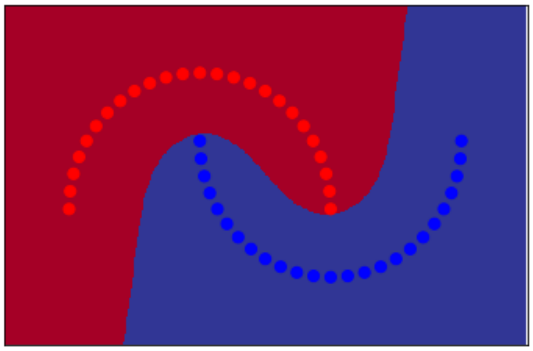

Figure 8: Kernel k-means can improve clustering alot

So far we had to choose the number of clusters k ourselves. In the next part we will deal with an algorithm that somewhat automates this process.

7.3 Hierarchical Clustering

Hierarchial clustering is an approach that generates bigger clusters from smaller clusters, the hierarchy going from small to big. Bigger clusters are formed out of smaller clusters with the biggest possible cluster being the entire set of inputs $\{\mathbf{x}_i, ..., \mathbf{x}_n\}$ and the smallest possible cluster the inputs $\mathbf{x}_i$ themselves.

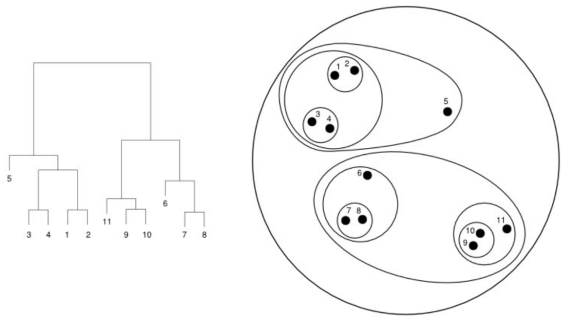

Figure 9: Example of hierarchial clustering of a set

In the figure above we see that the first clusters are assigned as $C_1 = \{1, 2\}, C_2 = \{3, 4\}, C_3 = \{7, 8\}, C_4 = \{9, 10\}, C_5 = \{5\}, C_6 = \{6\}, C_7 = \{11\}$. Notice how this corresponds to the leaves of the tree on the left side of figure 8.

Algorithm Hierarchical clustering

Input: $\mathbf{X} = \mathbf{x}_1, ..., \mathbf{x}_n \in \mathbb{R}^d$

function HierarchialClustering($\mathbf{X}$)
    Assign each input $\mathbf{x}_i$ into a cluster

    repeat
        Link the two clusters with *minimal distance*
    until Only one cluster is left

    return Tree describing cluster hierarchy
end function

The cursively written "*assign*" in line 2 and "*minimal distance*" in line 4 merit further attention. For the assign step we can define our own criteria for how big our initial clusters should be and by which criterion they should be assigned. We might be inclined to not just assign every point to its own cluster, since the whole point of clustering is to associate points with others. A simple approach would be so use k-means for some k to initialize our first clusters.

Example As we can see in figure 8 it would have been possible to put point 5 and 6 into a cluster, however they are rather far appart, so their might have been a limit of how far points can be from each other before becoming ineligible to be clustered together.

In order to calculate the minimal distance in line 4 there are three common options. Let $S_j \subseteq \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ be the set of inputs contained in the jth cluster

$$\begin{aligned}
\text{Simple linkage: } d(i,j) &:= \min_{\mathbf{x}\in S_i, \tilde{\mathbf{x}}\in S_j} \|\mathbf{x}-\tilde{\mathbf{x}}\|^2 \\
\text{Average linkage: } \mathrm{d}(i,j) &:= \operatorname*{mean}_{\mathbf{x}\in S_i, \tilde{\mathbf{x}}\in S_j} \|\mathbf{x}-\tilde{\mathbf{x}}\|^2 \\
\text{Complete linkage: } d(i,j) &:= \max_{\mathbf{x}\in S_i, \tilde{\mathbf{x}}\in S_j} \|\mathbf{x}-\tilde{\mathbf{x}}\|^2
\end{aligned}$$

Notice that all these expressions can be kernelized again. Which linkage we choose depends on our application. If we wanted to prevent "big" distances between points of two clusters we would choose complete linkage for instance.

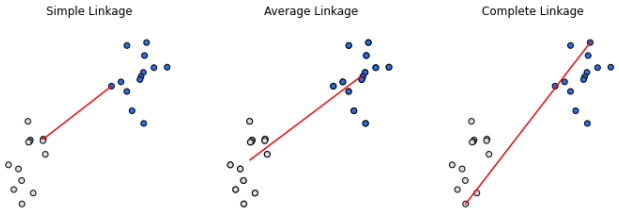

Figure 10: Geometric intuition behind the linkage types