Lecture 14: Clustering

Iain Styles

29 November 2019

Unsupervised Learning

Supervised learning – be that regression of classification – is by far the best developed form of machine learning and has been able to achieve extraordinary results when the combination of sophisticated methods, high-performance computing, and large annotated datasets are brought together. However, an area that is at least equally important (and will perhaps prove to be far more important) is that ow how to extract structure from data where we do not have any *a priori* knowledge of the data (i.e. no labels or annotations), and our task is to try to identify classes in the data without knowing what they are. In the conext of MNIST, can we "discover" that there are ten classes of digit purely from the data?

By far the most common approach to this problem is the task of *clustering*, in which we try to find groups of points that are in some way similar to other members of the same group, but different to members of other groups. In this section, we will examine a few different ways of finding clusters and the assumptions behind them.

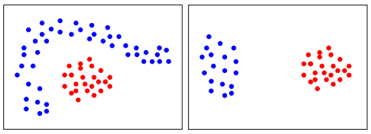

Figure 1: Some simple clusters

There are many hundreds of clustering algorithms, all with different underlying assumptions about the nature of the data. For example, some methods model the data as a probability distribution of a specific form; some are based on computing the local density of points; other are based on constructing a hierarchical representation of the relationships between the points in the dataset. We will consider three approaches:

- A "vector quantisation" approach, in which we try to find a set of prototype vectors, each of which is considered to be typical of a cluster.
- An agglomerative approach, in which points are grouped together hierarchically.
- A mixture modelling approach, in which the data is represented as a mixture of probability density functions.

The k-means Algorithm

Perhaps the simplest and most commonly used clustering technique is the *k*-means algorithm. It is a so-called vector quantisation technique that learns a prototype vector for each class in the data, and then assigns all data points to their nearest prototype. The data space is thus partitioned into cells, one per prototype, in what is known as a *Voronoi Tesselation* of the space. The prototype vectors

are often known as the cluster *centroids*. The partioning of space is somewhat similar to that of LDA, except that there are no labels from which we can learn a distribution: k-means does not explicitly model the distribution of the data.

This technique aims to learn a set of partition N data points into k clusters such that each data point belongs to the cluster that has the nearest centroid. This amounts to a partitioning of the space by lines that bisect those connecting the means (Figure 2). This is somewhat similar to LDA, and leads to a similar partitioning of space, but note that k-means does not explicitly model the distribution of the data. The clusters are defined by the allocation of points to regions.

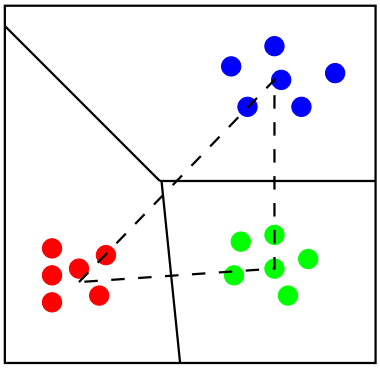

Figure 2: Partitioning of data in the k-means algorithm.

We will consider a simple version of the k-means algorithm known as *Lloyd's algorithm* which can be described as follows:

1. First, assign every data point to a clusters, at random.
2. Compute the centroid (in the data space) of each cluster by computing the average over its assigned members.
3. For each data point, calculate to which of the cluster centroids it is closest and re-assign the point to that cluster.
4. Return to step 2 and repeat until the clusters are stable.

This is sometimes known as naïve k-means because it isn't especially efficient. A more formal description can be found in Algorithm 1. This process is illustrated pictorially in Figure 3. There are a few points to note:

- The initial random seeding of the process means that you should perform multiple repeats in order to ensure stability of convergence.
- k-means is a *parametric* method: the number of clusters to be found has specified. It is is therefore typical to run the algorithm several times with different values of k and to try to determine which value is optimal. There is no established single method for doing this, it depends very much on the nature of the problem and what is already known about it. If k is too large, single clusters will be split; to small and clusters that should be separate will be combined.
- This is a computationally "hard" algorithm: $\mathcal{O}(kN)$. The number of distance calculations increases rapidly with the number of clusters and the number of data points so it can be very costly for large datasets with large numbers of clusters.
- Although we have used Euclidean distance in the examples, k-means can equally be used with other distance metrics.

k-means on MNIST

The application of k-means to the MNIST dataset can be found in the accompanying Jupter Notebook at `https://colab.research.google.com/drive/1t4zdBUUBM_8I0x_ADRTSKpeR6Y2RrE0D`.

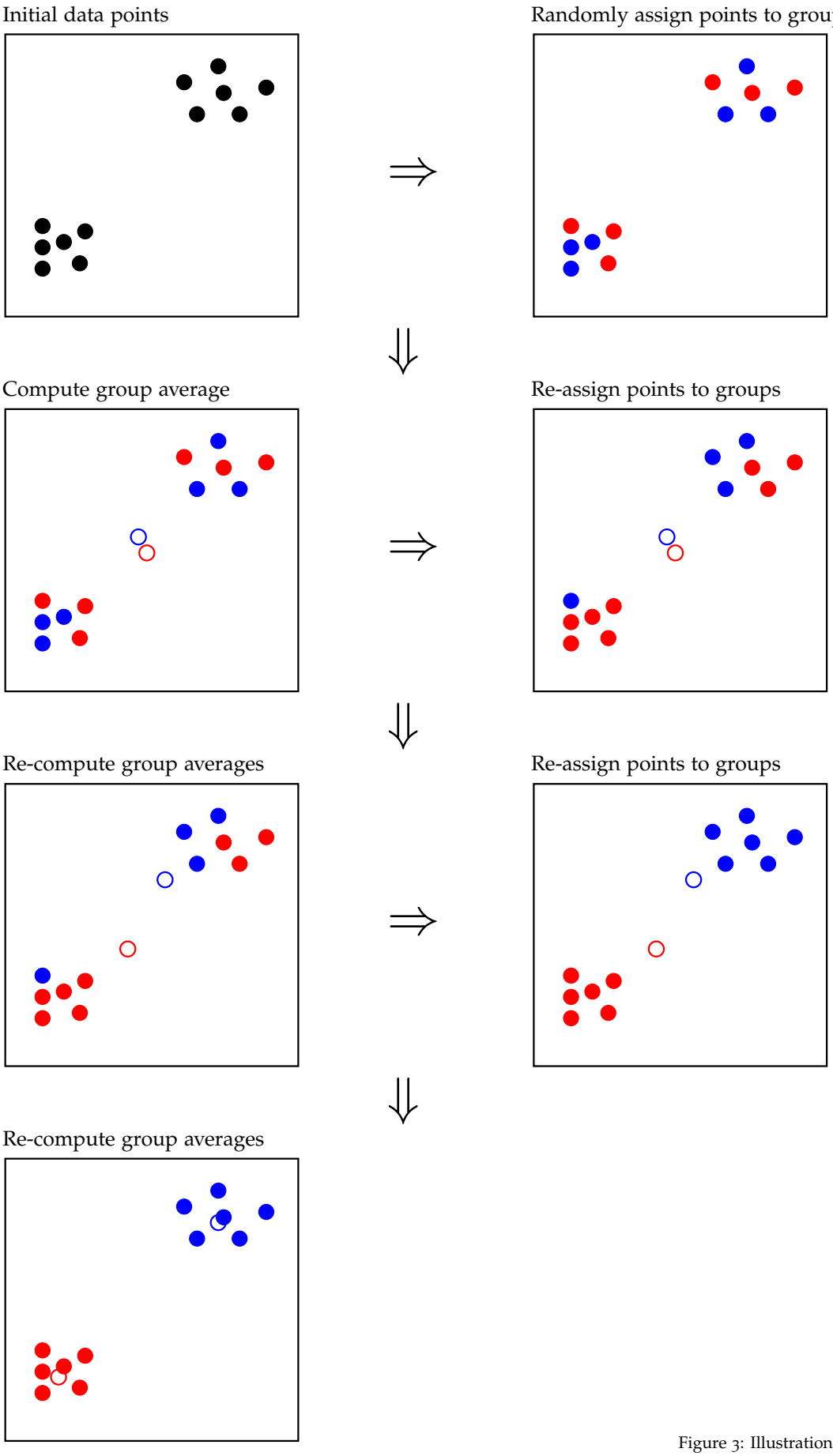

Figure 3: Illustration of the k-means algorithm.

Agglomerative Hierarchical Clustering

The k-means algorithms works by subdividing the whole dataset into "flat" subgroups with no internal structure, and if a different number of clusters is required, the algorithm must be re-run. Whilst this is a very popular approach, it is quite limited in terms of the types of clusters it can find: the strict convex partitioning of space it produces is not appropriate for many types of data. A popular approach that allows some of these problems to be overcome is to build up (agglomerate) clusters in a hierarchical manner, in which we build up a hierarchy of relationships between the data points. At run-time, this is a *non-parametric* approach: all point-point relationships are evaluated, and the resulting hierarchy analysed post-cluster to interpret the data's structure.

In this approach, we start by considering the individual data points. We calculate the separation between every pair of points, and group together the pair that are closest. We then remove that pair of points from the list of points, and replace it with the grouped pair. In the next iteration, all distances are again calculated, but this time we must compute the distance to the pairing computed in the first round. In order to do this, we need a method for calculating the distance between two groups of points. This is known as the *linkage* strategy and there are many approaches to this. Three of the most widely used (illustated in Figure 4) are:

- Single linkage: between the nearest points in the two groups.
- Average linkage: between the centroids of the groups.
- Complete linkage: between the farthest points in the two groups.

Data: Data points $Data[N]$; Number of clusters K
Result: Cluster assignments $ClusterIndex[N]$, cluster centroids $Centroid[N]$

```
% Randomly assign each point to a cluster
for i ← 1 to N do
  | ClusterIndex[i] ← randint(1, K)
end
% Calculate the centroid of each cluster as the mean of
  its members
for i ← 1 to K do
  | Centroid[i] ← mean(Data[Cluster[i]])
end
% Iterate until clusters are stable
Converged ← False;
while Converged == False do
  | % Compute distance from every point to each cluster
  |   mean
  | for i ← 1 to K do
  |   | for j ← 1 to N do
  |   |   | dist[i][j] ← distance(Centroid[i], Data[j])
  |   | end
  | end
  | % Find the nearest mean for each data point
  | for i ← 1 to N do
  |   | for j ← 1 to K do
  |   |   | NewClusterIndex[i] = nearest(Data[i], Centroid[j])
  |   | end
  | end
  | % Generate new clusters
  | for j ← 1 to K do
  |   | NewCentroid[i] ← mean(Data[NewClusterIndex == i])
  | end
  | % Check to see if cluster membership has changed
  | Converged ← True
  | if NewClusterIndex != ClusterIndex then
  |   | Converged ← False
  | end
  | % Replace old clusters with new ones
  | ClusterIndex ← NewClusterIndex Centroid ← NewCentroid
end
```

Algorithm 1: The k-Means Algorithm.

Each of these gives a notably different result. In general, single linkage permits the formation of clusters which are extended in space, because it takes no account of the size of the existing cluster, whilst complete linkage favours more compact clusters. The choice depends on the nature of the dataset.

Once we have selected a linkage strategy, we can proceed to iteratively and hierarchically group points and groups of points together. An example of this is shown in Figure 5 and it is instructive to work through it.

$$
\begin{aligned}
\text{List of points} &\mapsto 0, 1, 2, 3, 4, 5, 6 \\
\text{Group 1 with 2} &\mapsto 0, 3, 4, 5, 6, (1, 2) \\
\text{Group 3 with 4} &\mapsto 0, 5, 6, (1, 2), (3, 4) \\
\text{Group 0 with (1,2)} &\mapsto 5, 6, (3, 4), (0, (1, 2)) \\
\text{Group 5 with 6} &\mapsto (3, 4), (0, (1, 2)), (5, 6) \\
\text{Group (3,4) with (0,(1,2))} &\mapsto (5, 6), ((3, 4), (0, (1, 2))) \\
\text{Group (5,6) with ((3,4),(0,(1,2)))} &\mapsto ((5, 6), ((3, 4), (0, (1, 2))))
\end{aligned}
$$

The most common way to represent this is with the **dendrogram** shown in Figure 5. The height of each "junction" in the dendrogram represents the distance between the points being paired (so $(3, 4)$ join at height 1, for example).

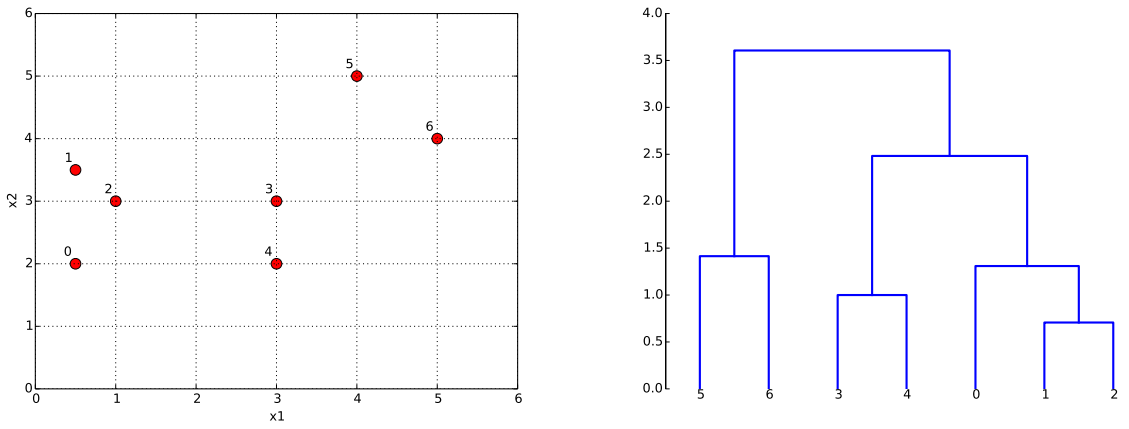

Figure 5: Datapoints for Hierachical Clustering and the resulting dendrogram using euclidean distance with average linkage.

One of the advantages of this method is that it generates all numbers of clusters in one pass. By cutting horizontally across the dendrogram at different heights we can obtain the desired number of clusters just by choosing how many lines we cut. The disadvantage is that it can be very time consuming to compute the dendrogram: if there are a large number of sample points, and pairwise distance calulations have to be done between all of them in the first instance, then this very quickly becomes computationally expensive. Compare this to k-means, where the distance calulation is between the N points and the K cluster centres: $N \times K$ evaluations

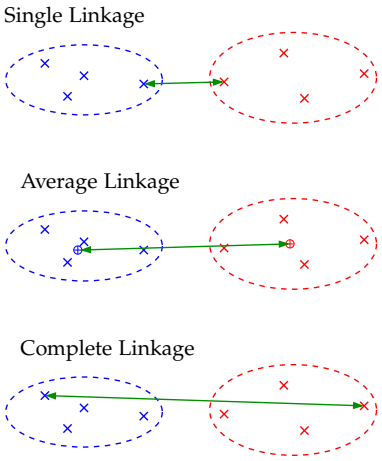

Figure 4: Linkage strategies for clustering of groups.

of the distance function vs the $N(N-1)/2$ calculations needed just to compute the first pairing in hierarchical clustering.

Hierarchical Clustering on MNIST

The application of hierarchical clustering to the MNIST dataset can be found in the accompanying Jupter Notebook at `https://colab.research.google.com/drive/12EDJyTk7XH9_OkEHbjieKTWwOhUNjo-9`.

Gaussian Mixture Models

k-means and agglomerative hierarchical clustering are heuristic approaches to clustering that have at best weak roots in statistics. They are also "hard" clustering techniques in the sense that they give a single cluster assignment to each sample. Perhaps advantageously, they make no strong statistical assumptions about the data, but the lack of an underpinning data model means that their is little to no nuance in their prediction. One way to overcome this is to introduce a statistical prior in a similar way to how it is done in LDA. In *Gaussian Mixture Modelling* (GMM), we make an assumption that the data is drawn from a distribution that can be modelled by a finite number of Gaussians, each with a mean and variance that has to be learned from the data in the absence of labels. With this model, it will be possible to predict which of the mixture components a data point is most likely to belong to.

The starting point for a GMM is its underlying model. We will seek to explain the data by assuming it is drawn from a probability density function of the form

$$p(\mathbf{x}) = \sum_{k=1}^{K} A_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{1}$$

The A_k are the *mixing coefficients* of each Gaussian density *component* $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$. What we aim to do in GMM is to

This distribution is a mixture of Gaussian distributions. In GMM, we have to perform essentially two tasks:

1. Learn the parameters of the GMM which best describe the data.
2. Determine from which component a data point is most likely to have been generated.

That is, we assume that each component is the probability density function for an indepedent data generating process. Our goal is to find which process generated a given data point.

To see how we can do this, let us first make some observations about Equation (1). First, we note that each Gaussian component is, by definition normalised. Since $p(\mathbf{x})$ is also a probability density function, we can show that this means that

$$\sum_{k=1}^{K} A_k = 1 \tag{2}$$

It can also be shown that $0 \leq A_k \leq 1$ by noting that $p(\mathbf{x})$ and $\mathcal{N}(\cdot)$ must always be ≥ 0, and the mixing coefficients therefore can be interpreted as probabilities, since they satisfy the requirements of probability.

We then observe that using the sum and product rules of probability, we can write

$$p(\mathbf{x}) = \sum_{k=1}^{K} p(k)p(\mathbf{x}|k) \tag{3}$$

and the symmetry between this expression and Equation (1) is clear: we can interpret the $A_k = p(k)$ as being the prior probability of choosing a point from component k, and $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = p(\mathbf{x}|k)$ are the class-conditional likelihoods.

We can finally, using Bayes' theorem compute $p(k|\mathbf{x})$, which is know as the *responsibility* of class k for point $\mathbf{x}$, i.e probability that class k "explains" point $\mathbf{x}$:

$$\begin{aligned} r_k(\mathbf{x}) &= p(k|\mathbf{x}) && (4)\\ &= \frac{p(k)p(\mathbf{x}|k)}{\sum_{k'=1}^{K} p(k')p(\mathbf{x}|k')} && (5)\\ &= \frac{A_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{k'=1}^{K} A_{k'} \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_{k'}, \boldsymbol{\Sigma}_{k'})} && (6) \end{aligned}$$

The responsibilities $r_k(\mathbf{x})$ can be viewed as the probability that component k explains $\mathbf{x}$. GMM is therefore a "soft" clustering technique: it predicts the probability with which a data point $\mathbf{x}$ belongs to each of the K classes.

The one remaining question is how we learn the components $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$. This is not a trivial undertaking. One way to do this would be to maxmimise the likelihood as we have done in the past. This is given by the joint distribution over a set of data points

$$p(\mathbf{X}|\mathbf{A}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \prod_{i=1}^{N} \sum_{k=1}^{K} A_k \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k). \tag{7}$$

Even taking the log does not help us to maxmimise this. We have to solve this numerically. This can be done via some non-linear optimisation process, or using the technique of *expectation maxmisation*, a sophisticated technique that is beyond the scope of this course. We will assume that we have access to a technique for maximising the likelihood.

Let us study a toy problem with this method. Let us create an artificial data generating process with three Gaussian components. These are shown in Figure 6, together with their parameters. The reds dots show 100 data points sampled from the distribution in the proportion given by the values of A_k.

We use `sklearn.mixture.GaussianMixture` with expectation maxmisation to learn the parameters of Equation (1). The results are shown in Figure 7.

k	A_k	μ_k	σ_k
1	0.3	-2.0	1.0
2	0.4	1.0	0.7
3	0.3	4.0	0.8

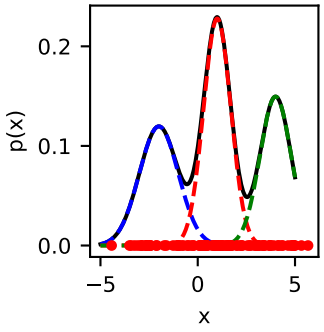

Figure 6: Components of an GMM.

k	A_k	μ_k	σ_k
1	0.26	-2.26	0.95

Cluster assignments can then be made by computing which component has the highest responsibility. We can map the responsibilities out over the range of value of x by evaluating Equation (6) as shown in Figure 8. The convention is to assign point x to the component with the largest responsibility, but we can see from this that more nuanced assignments can be made by noting that the responsibilities are probabilities. There are some clear regions of x, where there is significant uncertainty over the cluster assignment because the responsibilities are not binary. This provides a much more nuanced view than hard clusterings such as that produced by k-means.

A demonstration of this can be found at `https://colab.research.google.com/drive/1Gta0oTu_86UFkAMHqrhpQ7EdKrgmaPXW`.

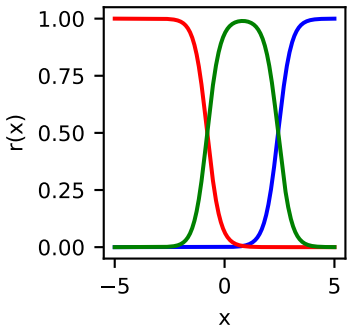

Figure 8: Responsibilities $r_k(x)$of the three classes (1: red, 2: green, 3: blue) in our GMM.