4 An unsupervised intermezzo: K-means clustering

In this lecture we almost exclusively treat *supervised* learning problems. Such problems are characterized by the fact that each training sample comes with a correct target value (for instance a class label or a regression value). The task is then to learn from the training data an algorithm that should approximate the known target values on the training data, and generalize well to examples not seen during training.

In *unsupervised* learning, no correct target values are supplied – only the input patterns $\mathbf{x}_i$. The learning task is here *to discover structure* in the data.

For instance, the $\mathbf{x}_i$ might be vectors of customer data of some mail order warehouse. The company running this business is probably interested in grouping their customers into groups that have similar customer profiles, in order to facilitate group-specific advertisment campaigns. More generally, discovering *clusters* within an a priori unstructured sample set is often a first step in exploratory data analysis in the natural and social sciences. The guiding idea is to find K clusters such that the training samples $\mathbf{x}$, $\mathbf{x}'$ that are assigned to the same cluster G should lie closely together, that is, $\|\mathbf{x} - \mathbf{x}'\|$ should be small. Conversely, if $\mathbf{x}$, $\mathbf{x}'$ are assigned to two different clusters G, G', then $\|\mathbf{x} - \mathbf{x}'\|$ should be large. This is easy if samples $\mathbf{x}$ are just numerical feature vectors. When samples come in a heterogeneous data format – for instance, $\mathbf{x}$ is a description of a customer involving numerical data *and* class data *and* symbolic descriptions – then finding a distance measure $\|\cdot\|$ in the first place is a challenging task. Solving this task often requires some ingenuity, and the distance measure found may not satisfy all the mathematical requisites of a metric. In such cases, there are many specific clustering techniques that work with different kinds of pseudo-distances.

There are many other unsupervised data-structuring tasks besides simple clustering of real-valued sample vectors $\mathbf{x}$:

- In a time series that is generated by alternating, different generators, one might want to discover the number of such generators and when they start and end generating a time series (unsupervised time series segmentation). A nice example for this is the discovery of different sleep phases in EEGs of human subjects. A common approach is to train a set of K signal predictor devices (e.g., neural networks) in mutual competition. In such *mixture of experts training*, at each time step only one of the K predictors is allowed to adjust its parameters – namely the one that could best predict the next signal value. Starting from K randomly initialized predictors, this setup leads to a competitive differentiation of predictors, each of which learns to specialize on the prediction of a particular generating mode in the time series.
- In an auditory signal that is an additive mixture of generators, one might wish to single out the generating signals. Surprisingly enough, this is possible if the original generators are statistically independent. Check Paris Smaragdis' FAQ page on *blind signal separation* and *independent component analysis* at http://web.media.mit.edu/~paris/ica.html for compelling audio demo tracks where a raw signal that is an overlay of several speakers or muscial instruments is

separated into almost crystal-clear audiotracks of the individual speakers or instruments.
- In symbolic machine learning and data mining, there are numerous unsupervised learning techniques that aim at distilling concise symbolic descriptions (e.g., context-free grammars or automata) from (huge) symbolic datasets.
- Another field that is, technically speaking, a case of unsupervised learning is *data compression.* Here the task is to detect regularities (= redundancies) in a large dataset that can be used to rewrite the data in a condensed form.

All in all, the field of unsupervised machine learning is as large, as important and as fascinating as the field of supervised learning. It is sad that a semester is short and enforces concentration on only one branch of ML. All we will do is to describe a particularly simple technique for clustering, called *K*-means clustering. Without some such unsupervised technique, one cannot really use RBF networks – the determination of well-placed and well-shaped input filters ϕ_i is a typical unsupervised learning task.

We can be brief, because *K*-means clustering is almost self-explanatory. Given: a training data set $(\mathbf{x}^i)_{i=1,...,N} = ((x^i_1, ..., x^i_n)^\mathsf{T})_{i=1,...,N}$ of real-valued feature vectors, and a number *K* of clusters that one maximally wishes to obtain. The algorithm goes like this:

Initialization: randomly assign the training samples to *K* sets S_j ($j = 1, ..., K$).

Repeat: For each set S_j, compute the mean $\mu_j = \frac{1}{|S_j|}\sum_{\mathbf{x}\in S_k}\mathbf{x}$. Create new sets S'_j by putting each sample x into the set S'_j where $\|\mu_j - \mathbf{x}\|$ is minimal. Dismiss empty S'_j and reduce *K* to *K′* by subtractring the number of dismissed empty sets (this happens rarely). Put $S_j = S'_j$ (for the nonempty sets) and $K = K'$.

Termination criterium: Stop when in one iteration the sets remain unchanged.

It can be shown that at each iteration, the error quantity

$$J = \sum_{j=1}^{K}\sum_{\mathbf{x}\in S_k}\|\mathbf{x} - \mu_j\|^2 \tag{4.1}$$

will not increase. The algorithm typically converges quickly and works well in practice. It finds a local minimum or saddle point of *J*. The final clusters S_j may depend on the random initialization. The clusters are bounded by straight-line boundaries; each cluster forms a *Voronoi cell*. Thus, *K*-means cannot find clusters defined by nonlinear boundaries. Figure 4.1 shows an example of a clustering run using *K*-means.

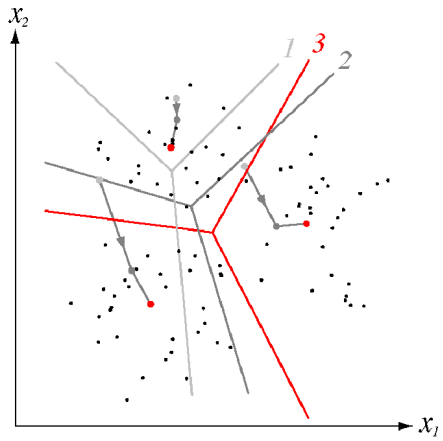

Figure 4.1: Running K-means with $K = 3$ on two-dimensional samples $\mathbf{x}$. Dots mark cluster means $\boldsymbol{\mu}_j$, lines mark cluster boundaries. The algorithm terminates after three iterations. (Picture taken from chapter 10 of the Duda/Hart/Stork book).

This is how K-means clustering may be used in conjunction with an RBF network:

Problem statement: given d-dimensional training data $\mathbf{x}_i$ ($i = 1,..., N$, may be preprocessed), find a "good" number M of "good" RBF filters filters ϕ_j ($j = 1,..., L$) to train a RBF network $y_k(\mathbf{x}) = y_k(\mathbf{x}) = \sum_{j=1}^{L} w_{kj}\phi_j(\mathbf{x}) + w_{k0}$ for n classes $k = 1, ..., n$.

Idea: Cluster the $\mathbf{x}_i$ by K-means clustering, create one filter per cluster, use Gaussian RBFs with covariance matrix Σ determined by cluster properties to represent data of each cluster.

Algorithm:

1) Use intuition (or trial and error with cross-validation...) to fix a desired target clustering number K.
2) Run K-means clustering on training data $\mathbf{x}_i$, which partitions the training samples into K' (maybe $K' < K$, but most likely $K' = K$) sets $D_1, ..., D_{K'}$ with means $\boldsymbol{\mu}_1, ..., \boldsymbol{\mu}_{K'}$.
3) For cluster j ($j = 1, ..., K'$) let $\mathbf{x'}_1^j,...,\mathbf{x'}_{N_j}^j$ be the normalized (i.e., cluster mean subtracted) data points in D_j, and let $\mathbf{X}^{\prime j}$ be the matrix that contains $\mathbf{x'}_1^j,...,\mathbf{x'}_{N_j}^j$ as columns. Compute the covariance matrix $\Sigma_j = \mathbf{X}^{\prime j}\,(\mathbf{X}^{\prime j})^{\mathsf{T}}$. Put

$$\phi_j(\mathbf{x}) = \exp\left(-\frac{1}{2}(\mathbf{x}-\mu_j)^{\mathsf{T}}\Sigma_j^{-1}(\mathbf{x}-\mu_j)\right).$$

Notes:

The RBF filters filters ϕ_j created in this way should accomodate to the shape and orientation of their cluster due to Σ_j. Some experimentation with Σ_j might however further improve the overall classificator performace: one might for instance try to

flatten out the filters ϕ_j for clusters with only few members, by using $\kappa_j \Sigma_j$ instead of Σ_j, where $\kappa_j > 1$.

Strictly speaking, we don't have a *radial* basis function network any more, because the filters ϕ_j are not radially symmetric. However, one still speaks of RBF networks.

The RBF filters are derived from pdf's of multivariate Gaussians. However, they are not strictly pdfs because our ϕ_j don't integrate to unity. For the purposes of using the ϕ_j as input filters in a RBF network, that does not matter (the weights of the network will be adjusted automatically to make up for different scalings of the filter functions).