Unsupervised Learning

1 Introduction

The term "unsupervised learning" or "learning without a teacher" is generically associated with the idea of using a collection of observation $\mathbf{X}_1, \ldots, \mathbf{X}_n$ sampled from a distribution $p(\mathbf{X})$ to describe properties of $p(\mathbf{X})$. This definition is extremely generic, and could describe, for example, any procedure of descriptive statistics.

In practice, the term almost universally refers to:

clustering: procedures that identify "groups" in the data.

mixture decomposition: a venerable area of statistics devoted to identifying the parametric densities of individual populations composing a "superpopulation".

principal component analysis: a method strictly related to SVD and the K-L transform, which seeks to find uncorrelated "features" obtained as linear combinations of the original features. Related fields are Factor Analysis and Analysis of Variance (ANOVA).

association rule discovery: a data mining technique that finds collections of attributes (better, attribute values) that frequently appear together in observations.

multidimensional scaling: the process of identifying a Euclidean space of small dimensions, and a possibly nonlinear mapping from the original space to the new space, in a way that the distance between pairs of training points in the original space are almost equal to the distances between their projections.

Other procedures are grouped under the name "unsupervised learning", because of the generic connotation of the term.

Lesson: *the term unsupervised learning by itself is relatively meaningless, and needs to be appropriately qualified.*

In our lectures, we will explore two related topics: clustering and mixture decomposition.

2 Clustering

2.1 A cynical introduction

Just like "unsupervised learning", "clustering" is a poorly defined term. In the literature the following definitions are common:

- The process of finding groups in data.
- The process of dividing the data into homogeneous groups.
- The process of dividing the data into groups, where points within each group are close (or similar) to each other.
- The process of dividing the data into groups, where points within each group are close (or similar) to each other, and where points of different groups are far (or dissimilar) from each other.
- The process of dividing the feature space into regions with relatively high density of points, separated by regions with relatively low density of points.

These definitions are NOT equivalent. For example, "finding" groups in the data set is not the same as partitioning the dataset into homogeneous groups. As another example, consider that the last definition could describe a 2-cluster case that does not satisfy the previous definition: cluster 1 is a cylindrical spiral which is "wrapped around" cluster 2, which is cylindrical, elongated, and aligned with the axis of the spiral.

Also, the definitions are generic: they do not specify what the terms "group" is, what "homogeneous", "close", "far", "relatively high density", "relatively low density", etc., actually mean. This is both a curse and a blessing. It is a curse because in general we have no idea of what a clustering method does until we see its formal, algorithmic specification. It is a blessing, because by changing the actual meaning of these terms, one can come up with an endless variety of clustering algorithms. Researchers have long benefited from this blessing: the variety of existing clustering algorithms is incredible, and newer ones are continuously published and patented. And, since the new algorithms solve new problems, they perform better (on those problems) than the older methods!

These remarks might seem a bit cynical. However, there is an underlying reality with which we have to deal: if we are looking for a "good" clustering algorithm in the literature, we are almost sure to find hundreds of different papers, most of which will be completely irrelevant for the problem we wish to solve.

Finally, a word of caution. We humans are extremely good at recognizing spatial patterns. We can quickly interpret 2- and 3-dimensional data plots, and pick out clusters. The problem is that we project our "intuition" onto higher-dimensional spaces. One of the aspects of the *curse of dimensionality* is the fact that the geometry of high-dimensional Euclidean spaces does not conform to our intuition developed in 2 or 3 dimensions. This can be a cause for frustration. Clustering algorithms in high dimensions might fail to identify clusters that we "see" when

we visualize the data by projecting it onto 2 or 3-dimensional spaces, where we can plot it using Matlab. Remember the homework problem where we discussed how the projections on 1 dimensions of a singular distribution can have densities? This is an indication that our intuition can be completely off-track.

2.2 A taxonomy of clustering methods

The best we can do for this class is to provide you with the tools to quickly analyze clustering algorithms.

The taxonomy described here has been derived from [5], with additions from [4, 3, 2], and with some changes. We can recognize a variety of "orthogonal[1]" characteristics that help us describing clustering algorithms.

Hierarchical vs. Partitional Methods. Hierarchical clustering algorithms induce on the data a clustering structure parameterized by a similarity parameter. Once the learning phase ends, the user can then obtain immediately different data clusterings by specifying different values of the similarity index. Partitional methods essentially produce one partition of the data into clusters[2]

Agglomerative vs. Divisive Methods. Agglomerative methods start by assigning each sample to its own cluster, and proceed by merging clusters. Divisive methods start by assigning all the samples to a unique claster, and proceed by splitting clusters.

Monothetic vs. Polythetic Methods. Monothetic methods learn clusters using one feature at a time. Polythetic methods use collections of features (e.g., by combining them linearly). The vast majority of clustering methods are polythetic, although people have devised (mostly unsuccessful) monothetic methods.

Hard vs. Fuzzy. In hard clustering each sample belong to one and only one cluster (once a strategy of how to assign samples that lie on cluster boundaries have been decided.) In fuzzy clustering, samples have different degrees of membership to different clusters. In general, membership functions are normalized so that the sum of the degrees of membership of a point to clusters sum to 1. For example, say that we cluster skies into sunny and cloudy. A sky that is 30% covered by clouds could be characterized as a 70% sunny - 30% cloudy sky.n

Using Deterministic vs. Probabilistic *Clusters.* If clusters are deterministic, a point either belongs to a cluster or does not belong to it. If clusters are probabilistic, a point belongs to a certain cluster with a certain probability. An example of probabilistic clusters are the components of a Gaussian mixture. Since each component is Gaussian, it has infinite size, and overlaps all the other components of the mixture. Hence, we cannot be

[1] By orthogonal we mean characteristics describing aspects that can be varied separately.

[2] Although certain methods, like tree-structured vector quantizer produce a hierarchical structure and yet are usually classified as partitional method. This division, which is commonly adopted in the literature, is either arbitrary, or relies on bad choices of the terms.

sure to which cluster a point belongs, and we characterize this uncertainty in terms of probability[3]. It is important to note the differences between probabilistic clusters and fuzzy clusters: a point always belongs to one and only one probabilistic cluster; however, we do not know to which. In fuzzy clusters, a point can belong to multiple clusters, and we know which ones they are! Also, probabilities and fuzzy membership functions are manipulated differently: probabilities rely on sums and products (the probability of the union of disjoint events is the product of the probabilities, the probability of the intersection of independent events is the product of the probabilities, etc. etc.), while usually the operation applied to fuzzy membership functions are the maximum and the minimum.

Using Deterministic vs. Stochastic *Algorithms.* If all the steps in the clustering algorithm are deterministic, the method is deterministic. Some methods use randomized steps, and fall in the other category.

Incremental vs. Non-Incremental. Some methods need all the sample points from the very beginning. Other can be initialized with fewer samples, and incrementally refined. The latter methods are sometimes better suited for large datasets.

3 Hierarchical Methods

As mentioned above, the term "hierarchical methods" is a bit of a misnomer. In practice, it is used to denote a very specific class of methods that operate as follows:

- assign each sample to a separate cluster;
- group the pair of clusters for which a similarity criterion is smallest;
- assign to the merge the value of the similarity between the merged clusters;
- iterate the previous 2 steps until all the data is grouped in a single cluster.

Note that these methods are also *agglomerative* in nature.

The clustering process can be represented using a tree called a **dendrogram** (greek term that means drawing of a tree, by the way.) A dendrogram differs from a regular tree in that the direction from the root to the leaves is interpreted as a similarity axis. Merges are then drawn at the coordinate corresponding to the attached similarity value.

An advantage of the dendrogram is that one can "cut" the tree at a desired similarity value, and obtain a tree with inner nodes corresponding to similarity values smaller than the specified one. Hence one can construct a variety of trees by "decapitating" the full dendrogram.

A wide variety of hierarchical methods can be constructed by varying the cluster similarity index. The following are the most common such indexes:

[3]Of course, even in this case, each sample will belong to one cluster only. However multiple identical samples can come from different clusters.

Nearest-neighbor or Single-Link. The (dis)similarity between two clusters A and B is the minimum of the pairwise distances between elements of A and of B. This method can produce long, stringy clusters, and can easily create "bridges" among clusters, that humans would not select.

Furthest-neighbor or Complete-Link. The (dis)similarity between two clusters A and B is the maximum of the pairwise distances between elements of A and of B. This method tend to produce tight, "compact" clusters - it will not recognize very elongated clusters until late in the clumping process.

Group Average. The (dis)similarity between two clusters A and B is the average of all the pairwise distances between the elements of A and of B. This is somewhat in the middle between the single-link and the complete-link methods.

Centroid, or mean. The (dis)similarity between two clusters A and B is the distance between the centroids (arithmetic means) of the two clusters.

Median. The (dis)similarity between two clusters A and B is the distance between the medians (the median is the point whose jth coordinate is the median of the jth coordinates of the points in the cluster. I am honestly not sure of how to easily characterize the difference between the median and the mean methods.

Ward's method The (dis)similarity between two clusters A and B is the increase of the dispersion of the points around the centroids of their cluster due to the cluster merging. The dispersion is measured as the sum of the squared distances between the points and the centroid. Before the merge, one measures the dispersion of the points of A around the centroid of A, and does the same for B. After the merge, one computes the centroid of the merged cluster, and sums the squared distances of all the points of A and B from the new centroid. Note that this same measure is used in a divisive approach: the tree-structured vector quantizer. Like the complete-link approach, this method will not favor very elongated clusters, but its aversion to them is somewhat more moderate.

Write your own and publish a paper. Provided that nobody else has done it before...

If the data conforms to our intuition of clusters (namely, we have nice, tight-and-roundish data clumps that are well separated from each other) all the above methods produce similar results. In fact, all the methods we will describe here produce similar results.

Duda, Hart, and Stork mention that from the dendogram one could determine whether groups are "natural": if splits in the dendrogram are spread across all values of the similarity scale, then there is no "natural" number of clusters. The way to read this statement is the following: once we select a metric (more on this later), and a cluster-distance index, we have implicitly defined what a "cluster" is. This definition is indeed rather precise, but potentially difficult to explain in words. More specifically, even in 2 dimensions it would be hard to tell, by visually inspecting a data set, how the algorithm we selected will cluster it. Hence, the fact that the dendrogram suggests that there is a natural number of clusters for a specific problem need not match the judgement of an expert user that analyzes the data....

A similar, but more precise statement is contained in Hastie, et al., (p. 475). Here the authors mention a somewhat objective way of judging how well the hierarchical clustering algorithm represents the data. This is how it works. Let N be the number of samples. Once we choose a metric, we have $N(N-1)/2$ pairwise distances. For each pair of points (i, j), we can also define the *cophenetic dissimilarity*[4]: look at the dendrogram, and find the node $n_{i,j}$ where the two points are merged in the same cluster. Then the cophenetic distance between points i and j is the value of the similarity index of the node $n_{i,j}$. Now we can create a 2-dimensional scatterplot with $N(N-1)/2$ points $p_{i,j}$, each corresponding to an unordered pair. The coordinates of $p_{i,j}$ are the original dissimilarity between points i and j, and their cophenetic dissimilarity. From the set of points $\{p_{i,j}\}$ we can compute the correlation coefficient between the two coordinates, known as the *cophenetic correlation coefficient*. This can be used to judge how well the cluster hierarchy captures the original distances between points.

3.1 Final Remarks

One can construct hierarchical divisive algorithms. These algorithms will start by grouping all the data in a single cluster, and dividing it until each distinct data point belongs to a cluster. As with agglomerative hierarchical algorithms, these produce a hierarchical clustering structure that can be represented using a dendrogram. Methods of this class are not very common.

The main benefits of hierarchical methods is their ability to construct clusters having arbitrary shapes: they are not limited to discovering "roundish and tight" clumps of data.

Their main downside is their high computational cost, which makes them unsuitable for very large data sets.

---

4 Partitional Algorithms

Partitional clustering algorithms divide the data into a set of clusters. Unlike hierarchical methods, they do not yield a hierarchical clustering structure. Almost universally, their operation consists of minimizing an objective function. Almost universally, minimizing the objective function is a hard problem, in fact NP-completeness has been proved for many clustering problems. Hence, partitional algorithms often work using iterative algorithms, that are only guaranteed to converge to a local minimum.

4.1 k-Means, and related algorithms

The k-Means algorithm is commonly associated with the minimization of a squared error criterion. More specifically, the squared error of a data set $\mathcal{D}$ of N points with respect to a clustering

[4]Don't you wish you had taken a course in Ancient Greek? Then you too would be able to come up with such impressive-sounding words!

$\mathcal{C}$ of the space into K clusters $C_1, \ldots, C_K$, each of which is associated with a representative sample $\mathbf{c}_k$, is

$$e^2(\mathcal{D}, calC) = \sum_{k=1}^{K} \sum_{i=1}^{N} 1_{\{\mathbf{x}_i \in C_k\}} \|\mathbf{x}_i - \mathbf{c}_k\|^2,$$

where $\|\mathbf{x}_i - \mathbf{c}_k\|^2$ is the squared Euclidean distance between the ith data point ($\mathbf{x}_i$) and the representative sample $\mathbf{c}_k$ of the kth cluster, and $1_{\{\mathbf{x}_i \in C_k\}}$ equals to 1 if $\mathbf{x}_i$ belongs to cluster C_k, and to zero otherwise (here we assume that a sample belongs to one and only one cluster).

The seminal paper by Linde, Buzo, and Gray [7], cite two optimality criteria that must be satisfied by a clustering algorithm of this class in order to yield a solution that is a local minimum of the objective function:

Criterion nr 1 The representative sample $\mathbf{c}$ of a cluster $\mathcal{C}$ must be the point that minimizes

$$\sum_{\mathbf{x} \in \mathcal{C}} \|\mathbf{x} - \mathbf{c}\|^2$$

(or, more generally, minimizing the objective function over the samples of the cluster).

Criterion nr 2 A point $\mathbf{x}$ must be assigned to the cluster $\mathcal{C}^*$ whose centroid $\mathbf{c}^*(\mathbf{x})$ is the closest to $\mathbf{x}$

$$\mathbf{c}^*(\mathbf{x}) = \arg\min_k \|\mathbf{x} - \mathbf{c}_k\|^2$$

(or, more generally, minimizing the objective function for the sample $\mathbf{x}$).

The representative samples of a cluster are often called "centroids" (you might recall that we used the same term when describing the assigment of labels to nodes in a classification tree.)

The $k-$Means algorithm performs a heuristic search while satisfying both requirements. The algorithm works as follows

1. Perform an initial selection of centroids.
2. Iterate on samples: for each sample
   Find the closest centroid.
   Assign the sample to the corresponding cluster.
   Recompute the centroid of that cluster.
3. Repeat step 2 until a convergence criterion is met.

Typical convergence criteria include terminating after an iteration where the cluster memberships of the samples is unchanged, terminating after an iteration where the centroids are left unchanged, terminating after an iteration where the value of the objective function is unchanged, or terminating after a maximum number of iterations is reached. When the data set is large convergence can be somewhat slow, and the above terminating criteria are substituted by threshold criteria (e.g., terminating after an iteration where the objective function decreases by less than ϵ, etc.).

A method that closely resembles the $k-$Means algorithm (to the point that it is sometimes called $k-$Means) is the Vector Quantization (VQ) algorithm of Linde, Buzo, and Gray (also known as LBG, or as the Modified Lloyd Algorithm) [7]:

1. Perform an initial selection of centroids.

2. Iterate on samples: for each sample
   Find the closest centroid.
   Assign the sample to the corresponding cluster.

3. After iterating on all samples, recompute the centroids using the new assignment of points to clusters.

4. Repeat step 2 until a convergence criterion is met.

You might note a parallel similarity here: the $k-$Means and the LBG have the same relation as the original Perceptron training algorithm and the batched Perceptron training algorithm. As in that case, it is difficult to say whether the original $k-$Means performs better than the LBG algorithm. LBG has a distinct computational advantage: the computation of distances between points and centroids can be done more efficiently, because neither points nor centoids change during the computation.

Unwittingly, we have just gained a profound insight into the nature of these algorithms: $k-$Means is really a vector quantizer, that is, an algorithm that performs lossy compression of the data with respect to a squared Euclidean distance distortion. It conforms to the third definition of clustering reported at the beginning of this manuscript. Nowhere, in its definition, we can see embedded a notion of "cluster" as a relatively dense group of data separated from other relatively dense groups of data by relatively empty regions.

4.1.1 Variations of the k-Means algorithm

Here we describe some extensions to the $k-$Means (and LBG) algorithms.

The first attempts to modify the algorithm to adapt it more to the fourth definition of clustering, by permitting splitting and merging of clusters. At the end of each iteration, we compute the within-cluster "scatter" (e.g., the mean squared distance of the points to their centroids), and measures of between-cluster "scatters" (e.g., the mean squared distance between the points of a cluster and the centroid of another cluster). Clusters having within-cluster scatter "close" to the average between-cluster scatter are declared too spread and split. Pairs of clusters where the within-cluster scatter is "close" to the corresponding between-cluster scatter (namely, where the points of one cluster are not too far from centroid of the other) are merged. Another clustering algorithm that relies on splitting and merging is the ISODATA method, proposed in 1965.

Another class of methods attempt to improve over $k-$Means by "appropriately" selecting starting values of the centroids. In so doing, one hopes to ensure faster convergence to a better, hopefully optimal, solution. For example, one could train a tree-structured vector quantizer (Section 4.2) and use the produced centroids as seeds for the $k-$Means algorithm; the hope here is that the seed points are better-adapted to the distribution of the data than randomly selected ones. Alternatively, one could identify a *piercing set* of data (a set of points that are at a minimum prespecified distance). The hope is to cover well the feature space, namely, to avoid having large regions of the space that are not covered well by the initial tessellation.

In practice, it is debatable whether these approaches actually work as desired. If we observe what happens during the first few iteration of the original k-Means (or LBG), with randomly selected initial centroids, we see that the centroid positions varies wildly, and that the objective function decreases quickly. The final centroids are usually very different from the initial ones. After the first few iterations, the method finally identifies a local minimum, and from this point on we observe an orderly but slower improvement of the objective function while the algorithm converges. The same behavior is also usually observed for the other methods described above: the final centroids often are completely different from the seeds. It is conceivable, however, that an appropriate selection of centroids could prevent convergence to particularly bad local minima that would result from extremely rare, extremely unfortunate random selections of centroids.

A third class of methods is used when the number of clusters is very high. Here, finding the nearest neighbor of a point can be computationally expensive, to the point that it is cheaper to maintain a multidimensional index (see [1], available through the home page) of the centroid positions, use it for the nearest neighbor search, and update is when the centroids are moved around. To minimize the updates to the index, this approach should be used with methods that batch the centroid updates.

We finally mention the possibility of using different classes of objective functions. For example, minimizing the average Euclidean distance (rather than its square) yields the $k-$Median or $k-$Medoid approach.

4.2 Tree-Structured Vector Quantizers

A main downside of the methods described in the previous section is their computational cost: albeit less demanding than agglomerative methods, LGB and $k-$Means require substantial resources when applied to large data sets. Tree-Structured Vector Quantizers (TSVQ)s are methods that trade off the optimality defined by the two criterions of Section 4.1, for speed and efficiency during both training and labeling new samples. Note tha labeling samples uses the same algorithm as the tree-based classifier.

The basic idea is similar to that of tree-based classifiers:

- Put all the data into a cluster;
- Divide the cluster into two parts using the "best" splitting hyperplane;
- Iterate the previous step on a selected cluster until the desired number of clusters is reached.

The "best" splitting hyperplane of course depends on the criterion. For Squared Error, the second step corresponds to dividing optimally the space into two clusters; them the splitting hyperplane is the one perpendicular to the line joining the centroids of the clusters, and equidistant from them. Simpler versions of TSVQ rely on monothetic splits. They are substantially faster to train and especially during classification, however they generally produce significantly worse partitions of the data.

TSVQs also differ on how they select the next cluster to be split. Some methods produce a balanced tree: when possible, nodes are split until a predefined leaf depth is reached. Other methods are greedy: at each step the split the node that yields the best improvement of the objective function.

It is worth noting that TSVQs can be considered a divisive hierarchical method.

4.3 Mixture Decomposition

In the literature, clusters are sometimes called "classes". All too often, and quite erroneously, clusters are indeed treated as semantic classes (albeit having unknown labels) or as "subclasses" of a population. This is in general a completely unjustified interpretation, which should be avoided at all costs. Nowhere, in the theory of clustering that we have discussed so far there is anything that could support considering clusters as subpopulations.

However, clustering has borrowed from statistics an approach, called mixture decomposition, which relies exactly on the assumption that the population we observe is composed of an unknown number of subpopulations, the distribution of each of which is known up to a finite number of parameters.

Consider this example, loosely derived from Duda, Hart, and Stork. We are fishing for salmon, and we measure the weight and length of each fish. Our catch will contain 2, 3, and 4 year old fish, that hatched in different streams. Since pacific salmons always spawn in the stream where they hatched, different rivers have different strains with different growth rates. The weights and lengths of fish of the same age from the same river have a distribution which is possibly well approximated using a bivariate Normal density. Therefore, the overall population of fish we are dealing with can be described by a density which is a linear combination of bivariate Normal densities, namely, by a *mixture* density.

A mixture density is said to be *identifiable* if its components and the corresponding weights (mixing coefficients) can be retrieved from the mixture itself. Some technical conditions can be used to tell if a class of mixtures is identifiable. If a mixture is identifiable, then we might be able to estimate the number of components, the parameters of the components and the mixing coefficients from a sample drawn from the mixture, just like we are able to estimate the parameters of an individual distribution.

A problem arises almost immediately: the methods that we used to estimate the parameters of individual distributions often fall short in the case of a mixture, either because of computational constraints or because of intrinsic problems. In the case of mixture of Gaussians, the maximum likelihood approach that worked so well for the Gaussian density does not work anymore here. Any elementary test in statistics contains a section that explains that the likelihood of a Gaussian density is unbounded. For example, consider two one-dimensional Gaussians. Fix the first (zero mean, unit variance), let the second be arbitrary, and fix the mixing coefficients to .5 and .5. Observe some data. Pick an observation, call it X, and center the second Gaussian at X. For every T, one can choose a small enough value of the variance of the 2nd Gaussian that ensures the likelihood is larger than T (write it down on a piece of paper, it is very easy to show).

Thankfully, not all is lost. In fact, under some smoothness conditions there is a local maximum

of the likelihood that is close to the actual value of the parameters, and that converges to the actual value of the parameters at the desired rate!

Unfortunately, finding maxima of the likelihood of a mixture is no simple matter. The algorithm most commonly used for this purpose is the Expectation-Maximization (E-M) algorithm.

We describe this iterative algorithm in the context of a mixture of Gaussians. A mixture of k Gaussians has a density of the form

$$f(\mathbf{x}, \theta) = \sum_{k=1}^{K} \pi_k \frac{1}{\sqrt{2\pi \det \Sigma_k}} \exp\left\{-(\mathbf{x} - \mu_k)' \Sigma^{-1} (\mathbf{x} - \mu_k)/2\right\},$$

where the parameter vector θ is $[\pi_1, \ldots \pi_K, \mu_1, \ldots, \mu_K, \Sigma_1, \ldots, \Sigma_K]$, π_k is the mixing coefficient[5] of the kth component, μ_k is its mean vector and Σ_k is its covariance matrix.

For each sample $\mathbf{x}_i$ we define a vector $\mathbf{p}^{(i)} = [p_1^{(i)}, \ldots, p_K^{(i)}]$, containing the probabilities that sample $\mathbf{x}_i$ was generated according to the 1st, 2nd,..., Kth component of the mixture. The algorithm starts with an initial guess for the vector θ, $\hat{\theta}_0$; then it iterates over the following two steps, where the subscript j denotes the index of the iteration:

E-Step Using the guess $\hat{\theta}_{j-1}$, compute for each sample the value of the vector $\mathbf{p}^{(i)}$, $\hat{\mathbf{p}}_j^{(i)}$.

M-Step Using the collection $\{\hat{\mathbf{p}}_j^{(i)}\}$ find value of the parameter maximizing the likelihood, $\hat{\theta}_j$.

The algorithm terminates when the change $\hat{\theta}_j$ after an iteration falls below a prespecified threshold.

The algorithm works because given $\hat{\theta}$ it is easy to calculate the collection of vectors $\{\hat{\mathbf{p}}^{(i)}\}$, and, conversely, given the collection of vectors $\{\hat{\mathbf{p}}^{(i)}\}$ it is easy to find the value of $\hat{\theta}$.

More specifically, for the Gaussian mixture case, the E-step uses the parameter estimates obtained from the *previous* iteration and computes

$$\hat{p}_\ell^{(i)} = \frac{\hat{\pi}_\ell \left(\sqrt{2\pi \det \Sigma_\ell}\right)^{-1} \exp\left\{-(\mathbf{x}_i - \hat{\mu}_\ell)' \hat{\Sigma}^{-1} (\mathbf{x}_i - \hat{\mu}_\ell)/2\right\}}{\sum_{k=1}^{K} \hat{\pi}_k \left(\sqrt{2\pi \det \hat{\Sigma}_k}\right)^{-1} \exp\left\{-(\mathbf{x}_i - \hat{\mu}_k)' \hat{\Sigma}^{-1} (\mathbf{x}_i - \hat{\mu}_k)/2\right\}},$$

where the superscript $'$ denotes transpose, while the M-step uses the collection of posteriors $\{\hat{\mathbf{p}}\}$ obtained during the *current* iteration to estimate the parameters as follows:

$$\hat{\pi}_\ell = \frac{\sum_{i=1}^{N} p_\ell^{(i)}}{\sum_{k=1}^{K} \sum_{i=1}^{N} p_k^{(i)}},$$

$$\hat{\mu}_\ell = \frac{\sum_{i=1}^{N} p_\ell^{(i)} \mathbf{x}_i}{\sum_{i=1}^{N} p_\ell^{(i)}},$$

[5]The collection of mixing coefficients is a probability mass function: $\sum_{k=1}^{K} \pi_k = 1$.

and $\hat{\sigma}_\ell$ is computed from $\hat{\mu}_\ell$ as

$$\hat{\sigma}_\ell = \frac{\sum_{i=1}^{N} p_\ell^{(i)} (\mathbf{x}_i - \hat{\mu}_\ell)(\mathbf{x}_i - \hat{\mu}_\ell)'}{\sum_{i=1}^{N} p_\ell^{(i)}}.$$

In the above equations, the index of the current iteration has been purposely omitted, to simplify the notation.

An interesting question is how to initialize the algorithm. This is customarily done using a regular clustering algorithm, such as the $k-$Means, to produce an initial partition of the data, from which mixing coefficients, centroids an initial estimates of the variance are obtained.

Mixture decomposition via the E-M algorithm is a computationally expensive affair, and can take a long time to converge especially in high dimensions. In the case of Gaussian Mixtures, the algorithm has also a tendency to fragility: recall that the likelihood is unbounded. If one of the estimates of the covariance matrices yields a near-singular covariance matrix, the E-M algorithm finds itself trapped in a place where the likelihood diverges. A typical solution to the problem is to force the algorithm to produce covariance matrices where the smallest eigenvalue is not larger than a predefined small constant.

5 How many clusters?

We have explored several classes of clustering algorithms. We have not, nevertheless, addressed the question of "how many clusters" there are in the data. This is potentially an ill-posed question: there might not be "a number of clusters". Some methods, most noticeably the agglomerative hierarchical methods, produce a data structure (the dendrogram) that in some occasions can be used to determine the number of clusters in the dataset. Recently, attention has been devoted to the use of certain quantities, such as the fractal dimension of the data, to determine whether there are "natural" scales at which to look for clusters in the data. A good place to start investigating this topic is Professor Christos Faloutsos' home page at http://www-2.cs.cmu.edu/ christos/.

6 Combining classification and unsupervised learning methods

The simplest idea for combining classification and unsupervised learning methods consists of partitioning the feature space using just the feature vectors and labeling each partition using the labels. For example, one could partition the data using a kdb-tree, and label each leaf of the tree using the labels of the data.

There are, nevertheless, algorithms that combine properties of clustering and of classification algorithms. Here we discuss the Learning Vector Quantization methods proposed by Kohonen[6]. These methods are sometimes described as being special cases of Neural Networks. I find this approach overly complex, except for people that speak Neural Networks. A much simpler interpretation is the following: a Learning Vector Quantizer is just a vector quantizer where

centroids have labels, and are iteratively selected to best represent the corresponding classes. This is done through iterative algorithms that "push" the centroids towards regions containing numerous samples of the associated class, and away from regions that contain numerous samples of other classes. Their description is a slight modification of the one found in [6].

Here we describe the three variants, two of which are shipped with the matlab package used for the project: LVQ1 and LVQ3.

6.1 LVQ1

LVQ1 is the simplest of the two versions. It starts by assigning to each class a number of representative samples (centroids) proportional to the number of samples of the class, and by selecting initial values for these centroids. Initial values could be obtained, for example, by applying regular VQ algorithms to the samples of individual classes.

Call μ_i the ith centroid, and let $c = \arg\min_i \|\mathbf{x} - \mu_i\|$ be the index of the closest representative sample to $\mathbf{x}$. Each iterations of LVQ1 consists of the following steps, executed for each training vector:

- $\mu_c(t+1) = \mu_c(t) + \alpha(t)[\mathbf{x}(t) - \mu_c(t)]$ if $\mathbf{x}$ and μ_c belong to the same class,
- $\mu_c(t+1) = \mu_c(t) - \alpha(t)[\mathbf{x}(t) - \mu_c(t)]$ if $\mathbf{x}$ and μ_c belong different classes,
- $\mu_i(t+1) = \mu_i(t)$ for all $i \neq c$.

The learning rate $\alpha(t)$ is either a constant or a value that decreases with time. The terminating conditions for the iterations are similar to those of regular vector quantizers.

6.2 LVQ2.1

This algorithm is similar to LVQ1, but attempts to accelerate convergence by selectively updating 2 centroids at a time. More specifically, consider a labeled point $(\mathbf{x}, y)$.
Let $\mu_y = \arg \min_{i|label(\mu_i)=y} \|\mathbf{x} - \mu_i\|$ be the closest centroid to $\mathbf{x}$ belonging to the same class.
Let $\mu_{\overline{y}} = \arg \min_{i|label(\mu_i)\neq y} \|\mathbf{x} - \mu_i\|$ be the closest centroid to $\mathbf{x}$ belonging to a different class.
LVQ2.1 concentrates the attention to points $\mathbf{x}$ that are near the midplane between μ_y and $\mu_{\overline{y}}$.
In particular, $\mathbf{x}$ is taken into account if

$$\min\left\{\frac{\|\mathbf{x}-\mu_y\|}{\|\mathbf{x}-\mu_{\overline{y}}\|}, \frac{\|\mathbf{x}-\mu_{\overline{y}}\|}{\|\mathbf{x}-\mu_y\|}\right\} > \frac{1-w}{1+w},$$

where the parameter w is called *window width*.

If $\mathbf{x}$ falls into the window, then

$$\begin{aligned}\mu_y(t+1) &= \mu_y(t) + \alpha(t)[\mathbf{x}(t) - \mu_y(t)] \\ \mu_{\overline{y}}(t+1) &= \mu_{\overline{y}}(t) + \alpha(t)[\mathbf{x}(t) - \mu_{\overline{y}}(t)]\end{aligned}$$

6.3 LVQ3

LVQ2.1 had a problem: it looked only at points that are near the midplanes between centroids of different classes, and that belong to one of these classes.

LVQ3 goes one step further, and keeps updating the centroids so that they better describe the data distribution.

Let μ_1 and μ_2 be the closest and second closest centroids to $\mathbf{x}$.

Case 1 Either μ_1 or μ_2 belong to the same class as $\mathbf{x}$ and $\mathbf{x}$ falls in the window.
If μ_1 belong to the same class as $\mathbf{x}$,

$$\begin{aligned}\mu_1(t+1) &= \mu_1(t) + \alpha(t)[\mathbf{x}(t) - \mu_1(t)] \\ \mu_2(t+1) &= \mu_2(t) - \alpha(t)[\mathbf{x}(t) - \mu_2(t)]\end{aligned}$$

otherwise

$$\begin{aligned}\mu_1(t+1) &= \mu_1(t) - \alpha(t)[\mathbf{x}(t) - \mu_1(t)] \\ \mu_2(t+1) &= \mu_2(t) + \alpha(t)[\mathbf{x}(t) - \mu_2(t)]\end{aligned}$$

Case 2 Both μ_1 and μ_2 belong to the same class as $\mathbf{x}$. Then

$$\begin{aligned}\mu_1(t+1) &= \mu_1(t) + \epsilon\alpha(t)[\mathbf{x}(t) - \mu_1(t)] \\ \mu_2(t+1) &= \mu_2(t) + \epsilon\alpha(t)[\mathbf{x}(t) - \mu_2(t)],\end{aligned}$$

where ϵ is a small constant (Kohonen suggest .1 to .5).

Default Nothing happens

6.4 Comparison

LVQ1 and LVQ3 are more "robust" than LVQ2, which is probably why they are included with the matlab package used for the class. Kohonen suggests to use LVQ2 only for few iterations and with small learning rate (it could be used, for example, to tune the results of the other two versions of the algorithm).

Kohonen also suggests how to compute an optimal adaptive learning rate for LVQ1. The corresponding material can be found in the paper, which is available on-line.

References

[1] V. Castelli. Multidimensional indexing structures for content-based retrieval. Research Report RC 22208, IBM, 02/13/2001.

[2] R.O. Duda, P.E. Hart, and D.G. Sytork. *Pattern Classification.* John Wiley & Sons, second edition, 2001.

[3] T. Hastie, Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics, 2000.

[4] Quin He. A review of clustering algorithms as applied to IR. Technical Report UIUCLIS–1999/6+IRG, University of Illinois at Urbana-Champaign, 1999.

[5] A.K. Jain, M.N. Murthy, and P.J. Flynn. Data clustering, a review. *ACM Computing Surveys*, 31(3):265–323, September 1999.

[6] T. Kohonen, J. Hynninen, J. Kangas, J. Laaksonen, and K. Torkkola. LVQ_PAK: The learning vector quantization program package. Report A 30, Helsinki University of Technology, 1996.

[7] Y. Linde, A. Buzo, and R.M. Gray. An algorithm for vector quantizer design. *IEEE Trans. Communications*, COM-28(1):84–95, January 1980.