Lecture 1

Unsupervised learning algorithm

We now consider unsupervised learning, where we are just given output data, without any inputs. The goal is to discover ?interesting structure? in the data; this is sometimes called knowledge discovery. Unlike supervised learning, we are not told what the desired output is for each input. Instead, we will formalize our task as one of density estimation, that is, we want to build models of the form $p(x_i|\theta)$. There are two differences from the supervised case. First, we have written $p(x_i|\theta)$ instead of $p(y_i|x_i, \theta)$; that is, supervised learning is conditional density estimation, whereas unsupervised learning is unconditional density estimation. Second, x_i is a vector of features, so we need to create multivariate probability models. By contrast, in supervised learning, y_i is usually just a single variable that we are trying to predict.

Unsupervised learning is arguably more typical of human and animal learning. It is also more widely applicable than supervised learning, since it does not require a human expert to manually label the data. Labeled data is not only expensive to acquire6, but it also contains relatively little information, certainly not enough to reliably estimate the parameters of complex models. Below we describe some canonical examples of unsupervised learning.

Discovering clusters

As a canonical example of unsupervised learning, consider the problem of clustering data into groups. For example, Figure 1.8(a) plots some 2d data, representing the height and weight of a group of 210 people. It seems that there might be various clusters, or subgroups, although it is not clear how many. Let K denote the number of clusters. Our first goal is to estimate the distribution over the number of clusters, $p(K|D)$; this tells us if there are subpopulations within the data. For simplicity, we often approximate the distribution $p(K|D)$ by its mode, $K^* = argmax_K p(K|D)$. In the supervised case, we were told that there are two classes (male and female), but in the unsupervised case, we are free to choose as many or few clusters as we like. Picking a model of the ?right? complexity is called model selection, and will be discussed in detail below. Our second goal is to estimate which cluster each point belongs to. Let $z_i \epsilon 1, ..., K$ represent the cluster to which data point i is assigned. (z_i is an example of a hidden or latent variable, since it is never observed in the training set.) We can infer which cluster each data point belongs to by computing $z_i^* = argmax_k p(z_i = k|x_i, D)$ model based clustering, which means we fit a probabilistic model to the data, rather than running some ad hoc algorithm. The advantages of the model-based approach are that one can compare different kinds of models in an objective way (in terms of the likelihood they assign to the data), we can combine them together into larger systems, etc.

Here are some real world applications of clustering

- n astronomy, the autoclass system (Cheeseman et al. 1988) discovered a new type of star, based on clustering astrophysical measurements.

- In e-commerce, it is common to cluster users into groups, based on their purchasing or web-surfing behavior, and then to send customized targeted advertising to each group

- In biology, it is common to cluster flow-cytometry data into groups, to discover different sub-populations of cells

Discovering latent factors

When dealing with high dimensional data, it is often useful to reduce the dimensionality by projecting the data to a lower dimensional subspace which captures the ?essence? of the data. This is called dimensionality reduction. The motivation behind this technique is that although the data may appear high dimensional, there may only be a small number of degrees of variability, corresponding to latent factors. For example, when modeling the appearance of face images, there may only be a few underlying latent factors which describe most of the variability, such as lighting, pose, identity, etc When used as input to other statistical models, such low dimensional representations often result in better predictive accuracy, because they focus on the ?essence? of the object, filtering out inessential features. Also, low dimensional representations are useful for enabling fast nearest neighbor searches and two dimensional projections are very useful for visualizing high dimensional data. The most common approach to dimensionality reduction is called principal components analysis or PCA.

Dimensionality reduction, and PCA in particular, has been applied in many different areas. Some examples include the following

- In biology, it is common to use PCA to interpret gene microarray data, to account for the fact that each measurement is usually the result of many genes which are correlated in their behavior by the fact that they belong to different biological pathways.

- n natural language processing, it is common to use a variant of PCA called latent semantic analysis for document retrieval

- In signal processing (e.g., of acoustic or neural signals), it is common to use ICA (which is a variant of PCA) to separate signals into their different sources

- In computer graphics, it is common to project motion capture data to a low dimensional space, and use it to create animations.

Discovering graph structure

Sometimes we measure a set of correlated variables, and we would like to discover which ones are most correlated with which others. This can be represented by a graph G, in which nodes represent variables, and edges represent direct dependence between variables (we will make this precise in Chapter 10, when we discuss graphical models). We can then learn this graph structure from data, i.e., we compute $\hat{G} = argmaxP(G|D)$.

Matrix completion

Sometimes we have missing data, that is, variables whose values are unknown. For example, we might have conducted a survey, and some people might not have answered certain questions. Or we might have various sensors, some of which fail. The corresponding design matrix will then have ?holes? in it; these missing entries are often represented by NaN, which stands for ?not a number?. The goal of imputation is to infer plausible values for the missing entries. This is sometimes called matrix completion. Below we give some example applications.

Clustering

Briefly, clustering is the collection of procedures used to describe methods for grouping unlabeled data, the X is, into subsets that are believed to reflect the underlying structure of the data generator. The techniques for clustering are many and diverse, partially because clustering cuts across so many domains of application and partially because clustering is a sort of preprocessing often needed before applying any model-based inferential technique.

The three main classes of clustering techniques are hierarchical, partitional, and Bayesian. It is true that Bayesian techniques are often either hierarchical or partitional; however, their character is often so different from what is typically understood by those terms that it is better to put them in a separate class.

The idea of a hierarchical technique is that it generates a nested sequence of clusterings. So, usually one must choose a threshold value indicating how far along in the procedure to go to find the best clustering in the sequence. The nesting can be decreasing or increasing. If it?s decreasing, usually it?s started by treating each data point as a singleton cluster. These procedures are called agglomerative since clusters are merged. If it?s increasing, usually it?s started by treating the whole data set as one big cluster that is decomposed into smaller ones. These procedures are called divisive since clusters are split. It will be seen that a graph-theoretic technique called minimal spanning trees yields a nested sequence of clusters and can be regarded as hierarchical, even though it will be treated here as a graph-theoretic technique.

Hierarchical methods also typically require a dissimilarity, which is a measure of distance on individual data points that lifts to a measure of distance on groups of data points. This does not particularly distinguish hierarchical clustering techniques because partitional clustering techniques often require a distance that functions like a dissimilarity. However, the way a dissimilarity is used is somewhat more consistent and central across hierarchical techniques as a class.

By contrast, nonhierarchical methods such as partitional methods usually require that the number of clusters K and an initial clustering be specified as an input to the procedure, which then tries to improve the initial assignment of data points. The initial clustering takes the place of a dissimilarity and threshold value in hierarchical procedures. It is true that hierarchical methods require an initial clustering; however, it is trivial: n clusters or one cluster. Algorithmically, the difference between hierarchical and partitional methods is that with hierarchical algorithms clusters are found using previously established clusterings, while partitioning methods try to determine all clusters optimally in one shot.

Bayesian clustering techniques are different from the first two classes because they try to generate a posterior distribution over the collection of all partitions of the data, with or without specifying K; the mode of this posterior is the optimal clustering. Bayesian techniques are closer to hierarchical techniques than partitional techniques because often there is an ordering on the partitions. All of these techniques require the specification of a prior and usually come down to some kind of hypothesis test interpretation.

Two problems that can arise in any technique are clumping and dissection. Clumping is the case where a single object fits into two or more clusters that are therefore allowed to overlap. An instance of this is document retrieval: The same word can have two different meanings, so a text cannot be readily fit into exactly one cluster. Overlapping clusters will not be examined here, but awareness of the problem is important. Likewise, the concept of dissection will not be studied here, apart from noting that it is the case where there is a single population that does not contain meaningful clusters but the goal is still to cluster the data for some other purpose. For instance, there may be no meaningful way to find clusters in a homogeneous city, but a post office may still partition the city into administrative regions so letter carriers can be more efficient.

It is important to remember that unsupervised learning is often just one step in a problem. For instance, one could cluster data into groups, do some kind of variable selection within each group, and then use Bayesian methods to generate conditional probabilities for one of the random variables given the others. Note that the last step actually becomes supervised because one of the Xs, or a function of them, has been found to be a good choice for Y.

To fix notation, consider a collection of data x_i, for $i = 1, ..., n$, that can be treated as IID outcomes of a variable $\mathbf{X}$. The goal is to find a class of sets $\mathcal{C} = C_1, ..., C_K$ such that each x_i is in one set C_k and the union of the C_ks is the whole sample $x^n = (x_1, ..., x_n)$. The key interpretative point is that the elements within a C_k are much more similar to each other than to any element from a different C_k . The natural comparison, if possible, is between a known clustering, which is essentially classification data, and whatever clustering a clustering procedure outputs. The difference is an assessment of how little information the algorithm loses relative to perfect information. Even though this is not formalized, it does represent a standard for comparison that could be readily formalized. It is an analog of the concept of goodness of fit.

Lecture 2

k-means clustering

Partitional Clustering

Partitional clustering algorithms produce a partition of examples into a specified number of clusters by either minimizing or maximizing some numerical criterion. A partitional clustering algorithm obtains a single partition of the data instead of a clustering structure, such as the dendrogram produced by hierarchical methods. Partitional methods have advantages in applications involving large data sets for which the construction of a dendrogram is computationally prohibitive. A problem accompanying the use of a partitional algorithm is the choice of the number of desired output clusters. This key design decision will be discussed separately. The number of different partitions of ? examples into g clusters can be calculated by applying the following formula:

$$N(n,g) = \frac{1}{g!}\sum_{m=1}^{g}(-1)^{g-m}(g/m)m^{n}$$

The number of possible partitions grows very fast. For a moderately small clustering problem with ? = 100 examples and g = 5 required clusters, this number is. Thus, even with the fastest computers, combinatorial search of the set of possible labelings for an optimum value of a criterion is clearly computationally prohibitive.

Partitional Criteria

A general approach to partitional clustering involves defining a measure of the adequacy of a partition and seeking a partition of the examples which optimizes that measure. There exist several measures of adequacy (partitioning criteria), based on cluster properties, most notably:

- cluster's compactness: how dissimilar are examples that lie within the same cluster;
- cluster's isolation, or separation: how far is the cluster from other clusters (or from the rest of examples)

Some clustering criteria can be used only when all attributes are continuous, whereas others can use a dissimilarity matrix. We review them separately, after defining an overall clustering criterion.

Having chosen a measure$h(\mathcal{C})$ of the cluster's compactness or isolation, an overall clustering criterion can be defined by a suitable aggregation over all clusters. Let C be the current partition consisting of g clusters: $C = C_1, ..., C_g$. There are several possibilities for defining an aggregation function:

K-means

K-means clustering is the most widely used partitional clustering algorithm. It starts by choosing K representative points as the initial centroids. Each point is then assigned to the closest centroid based on a particular proximity measure chosen. Once the clusters are formed, the centroids for each cluster are updated. The algorithm then iteratively repeats these two steps until the centroids do not change or any other alternative relaxed convergence criterion is met. K-means clustering is a greedy algorithm which is guaranteed to converge to a local minimum but the minimization of its score function is known to be NP-Hard . Typically, the convergence condition is relaxed and a weaker condition may be used. In practice, it follows the rule that the iterative procedure must be continued until 1% of the points change their cluster memberships. A detailed proof of the mathematical convergence of K-means can be found in.

```
1. Select K points as initial centroids.
2. repeat
    (a) Form K clusters by assigning each point to its closest centroid.
    (b) Recompute the centroid of each cluster.
3. until convergence criterion is met.
```

The centroid of a cluster can be thought of as the pure type the cluster represents, whether that object actually exists or is just a mathematical construct. It may correspond to a particular data point in the cluster as in K-medoids or to a point in the convex hull of the cluster, such as the cluster mean. Given an initial clustering, centroid-based methods find the centroids of the clusters, reassign the data points to new clusters defined by proximity to the centroids, and then repeat the procedure. The similarity between two clusters is the similarity between their respective centroids. Thus, the key choices to be made, aside from an initial clustering, are the distance to be used, the centroids to be used, and how many iterations of the procedure to use. Most of these procedures are variants on K-means clustering, which is worth presenting in detail since it readily captures the geometric intuition. Then the use of other measures of location or distances becomes straightforward.

Intuitively, the K-means clustering approach is the following. The analyst picks the number of clusters K and makes initial guesses about the cluster centers. The procedure starts at those K centers, and each center absorbs nearby points, based on distance; often the distances are found using a covariance matrix to define a norm. Then, based on the absorbed cases, new cluster centers, usually the mean, are found. The procedure is then repeated: The new centers are allowed to absorb nearby points based on a norm, new centers are found, and so on.

Usually this is done using the Mahalanobis distance from each point x_i to the current K centers,

$$d\left(\boldsymbol{x}_i, \overline{\boldsymbol{x}}_k\right) = \left[\left(\boldsymbol{x}_i - \overline{\boldsymbol{x}}_k\right)' S^{-1} \left(\boldsymbol{x}_i - \overline{\boldsymbol{x}}_k\right)\right]^{1/2}$$

where $\overline{\boldsymbol{x}}_k$ is the center of the current cluster k and S is the within-cluster covariance matrix defined by $S = (s_{j,\ell})$ for $j, k = 1, ..., p$ as

$$s_{j,\ell} = \frac{1}{n-1} \sum_{i=1}^{n} \left(x_{i,j} - \bar{x}_j\right)\left(x_{i,\ell} - \bar{x}_\ell\right)$$

Smart computer scientists can do K-means clustering (or approximately this) very quickly, sometimes involving the pooled within-cluster covariance matrix $W = (w_{j,\ell})$, where, for $j, \ell = 1, ..., p$,

$$w_{j,\ell} = \frac{1}{K} \sum_{k=1}^{K} \sum_{i=1}^{n_k} d_{i,k} \left(x_{i,j} - \bar{x}_{k,j}\right)\left(x_{i,\ell} - \bar{x}_{k,\ell}\right)$$

in which n_k is the number of points in cluster $k, \overline{x}_{k,\ell}$ is the mean of the ℓth variable in cluster k, and $d_{i,k} = 1$ if observation i is in cluster k and 0 otherwise. In practice, K-means clustering is often the only clustering procedure that is computationally feasible for large p, small n data sets.

The K-means procedure works because the cluster centers change from iteration to iteration, hopefully converging to the correct cluster centers, if K is correct. If the centers move too much, or for too many iterations, the clusters may be unstable. In fact, there is no guarantee that a unique solution for K-means clustering exists: If K is correct and two starting points

fall within the same true cluster, then it can be hard to discover new clusters since the two points will often get closer together and only $K?1$ clusters will be found. This means that it is very important to know K, even though it can be hard to determine. Unfortunately, specifying some values seems to be necessary for all clustering procedures. To use K-means, one can do a univariate search over k:
Try many values of k, and pick the one at the knee of some lack-of-fit curve; e.g., the ratio of the average within-cluster to between-cluster sum of squares.

$$\mathcal{C}_{K-means} = \arg\left(\min_{K} \min_{m_1,\ldots,m_K} \sum_{k=1}^{K} \sum_{i:C(i)=k} \|\boldsymbol{x}_i - m_k\|^2\right)$$

Lecture 3

Hierarchical Clustering

As noted before, hierarchical clustering techniques give a nested sequence of clusterings. So, the process of merging or dividing clusters can be represented as a tree, usually called a dendrogram in the clustering context, where the branching is based on inclusion. The process of generating a dendrogram is usually based on evaluations of distances between points or sets of points at each stage of the clustering procedure. So, hierarchical clustering procedures tend to be rigid in the sense that once a merge or split has been done it cannot be undone. Of course, this is not necessary; in principle one could write algorithms that include a depth-first search to carry out a procedure without locking in earlier choices. Hierarchical clustering procedures are either agglomerative or divisive. Agglomerative clustering means that, starting with all the data points as singleton sets, clusters are merged under a dissimilarity criterion until the entire data set is a trivial cluster.Divisive clustering means the reverse: The starting point takes the entire data set as a cluster and successively splits it until at the end the trivial clustering of n singleton sets results. In both cases, a threshold for how much merging or splitting is to be done must be chosen. There are a great variety of ways to choose the merges and splits and while a split may divide a cluster into more than two subsets and a merge may unite more than two clusters, it is conventional to limit all merges to two input clusters and all splits to two output clusters.

Agglomerative Hierarchical Clustering

At root, hierarchical agglomerative clustering merges sets according to fixed rules. The basic template for hierarchical agglomerative clustering is:

Start with a sample x_i, $i = 1, ..., n$, regarded as n singleton clusters, and a dissimilarity d defined for all pairs of disjoint, nonvoid subsets of the sample. If desired, a fixed number of clusters to form can be chosen. Otherwise, the procedure can be allowed to iterate $n?1$ times.

- At the first step, join the two singletons x_i and x_j that have the minimum dissimilarity $d(x_i, x_j)$ among all possible pairs of data points.
- At the second step, there are n?1 clusters $C_{1,1}, ..., C_{1,n-1}$. Find

  $$(j^*, j'^*) = \arg\min_{j \neq j'} d\left(C_{1,j}, C_{1,j'}\right)$$

  and merge clusters $C_{1,j^*} and C_{1,j'^*}$. Now there are $n?2$ clusters, say $C_{2,1}, ..., C_{2,n?2}$.
- Continue until the n points have been agglomerated into the desired number of clusters or into a single cluster of size n.

Alternatively, choose an error criterion. Find the errors for clusterings with different numbers of clusters. Choose the largest number of clusters for which the drop in error from permitting one more cluster is small.

The two main inputs to this template are the stopping rule and the dissimilarity d, which is taken up in the next subsection. The stopping rule basically is a way to decide where to draw a horizontal line in the dendrogram. Even if the procedure runs n ? 1 iterations, the choice of when to stop merging must be made to specify a particular clustering. The last step in the template is called finding the knee (or elbow). This is an informal way to choose the number of clusters when it is not preassigned. Consider a graph of the number of clusters K versus an error criterion for clustering; the most common error criterion is the ESS in, regarded as a function of K.

In practice, as K increases, the less information there is per extra cluster because there is a finite amount of information in the data. Intuitively, when K is small, there is a high amount of information per cluster because the information in the data is being summarized by a small number of cluster centers. As K increases, however, the information in the data accurately captured by using extra clusters decreases. This means there is a K beyond which permitting one more cluster reduces the error by such a small amount that the extra cluster is not worth including. In these cases, a graph of K versus the error criterion usually has a dent, or a corner, indicating a sudden dropoff of information per extra cluster, indicating a point of diminishing returns. This is called the knee or elbow of the curve and is usually found just by looking at the graph. Even so, there are ambiguous cases, and it can be hard to decide when to stop agglomerating and declare the clustering.

The result of a hierarchical agglomerative cluster analysis is often displayed as a tree and is called a dendrogram; see Fig. 8.2. The lengths of the edges in the binary tree shows the order in which cases or sets were joined together and the magnitude of the distance between them. If three clusters were to be found, the natural place to draw the horizontal line would be just above where $\{x_2\}$ is merged with $\{x_5, x_3\}$, so that the three clusters are x1, x4, and x2,x3,x5.

Choice of d

Note that in the optimization depends on d. In fact, the choice of d determines the set of clusterings from which a final clustering can be chosen. Different choices of d favor different classes of clusterings much like different choices of stop-splitting rules favor qualitatively different classes of trees in recursive partitioning. So, it is important to list the most frequently used dissimilarities along with some of their properties.

Let A and B be two disjoint subsets of x1,...,xn. To use the agglomerative clustering template from the last subsection, it is enough to define d(A ,B) in general for various choices of d. Since the goal is to merge sets of points successively, the d are often called linkages because they control the lengths of line segments joining, or linking, points xi to each other. Four of the most common dissimilarities are:

Nearest-neighbor or single linkage: A metric d on IRp is minimized to define a dissimilarity on sets, also denoted d, given by

$$d(\mathcal{A},\mathcal{B}) = \min_{a\in\mathcal{A},b\in\mathcal{B}} d(a,b)$$

Complete linkage: Here a metric d is maximized to define a dissimilarity on sets given by

$$d(\mathcal{A},\mathcal{B}) = \max_{a\in\mathcal{A},b\in\mathcal{B}} d(a,b)$$

Centroid linkage: The idea is to merge at iteration k +1 those clusters whose centroids at stage k were closest. That is,

$$d(\mathcal{A},\mathcal{B}) = d(\overline{a},\overline{b})$$

where a is a centroid for A and b is a centroid for B

Average linkage: In this case, the metric distances between points in A and B are averaged:

$$d(\mathcal{A},\mathcal{B}) = \frac{1}{\#(\mathcal{N})\#(\mathcal{B})} \sum_{i\in\mathcal{A},j\in\mathcal{B}} d\left(\boldsymbol{x}_i,\boldsymbol{x}_j\right)$$

Single-linkage clustering is essentially a nearest-neighbor criterion: Two points, or clusters, are linked if they are or have each others closest data point; only one link is required to join them. The extension of the single link connection between points to sets ensures there will always be a path with shortest lengths connecting all the points in a cluster. Thus, single-linkage clustering also admits a graph-theoretic interpretation. While this has many good properties, it has some deficiencies: Two clusters of any shape that are well separated apart from a few points on a line between them will be joined if the points between them are close enough to each other. This is called the chaining problem, and it is very serious in many data sets. The tendency toward such barbell-shaped regions can be quite strong because single-linkage clustering tends to form a few big clusters early in the hierarchy. Worse, in the presence of outliers, it can join two sets of points when most of the points are distant. Odd-shaped regions do occur naturally, so it?s unclear in general whether such properties are features or bugs. Indeed, if one anticipates regions that are separated, irregular or not, single-linkage clustering can be a good way to find them.

Complete linkage means that all points in the two clusters are joined by lines of length less than the upper bound on the maximum. This means that all the linkages between points are within the distance at which the cluster was formed. Compared with single linkage, complete linkage tends to form many smaller, tighter clusters. So, completelinkage dendrograms often have more structure to them and can therefore be more informative than single-linkage dendrograms. Also, because of the maximum, complete linkage is sensitive to outliers of the clusters themselves. Arguably, like support vector machines, complete linkage is often driven by a small set of points that may be outliers or inliers and not particularly representative of the data set. This may also lead to non-robustness of the clustering in that small changes in the data can result in big changes to the clustering. Often complete-linkage clustering runs in O(n3) for fixed p, possibly less for sparse similarity matrices. Centroid linkage calculates the distance between means of clusters. This average can be a good trade-off between the extremes of single and complete linkage. The limitation is that the distances at which clusters are formed are averages and do not correspond to any actual links between data points. A downside is that this linkage is not monotonic: If d j is the dissimilarity level for a given iteration j, it may be higher or lower than d j+1; a reversal of the ordering is called an inversion. In contrast, single linkage and complete linkage are monotonic in that the dissimilarity is always increasing. So, the clusters from centroid linkage can be more difficult to interpret. Average linkage is another compromise between single and complete linkage that is more demanding computationally because of the summation. The chaining problem is far less common for this method as compared with single linkage, and the effect of outliers is reduced by the averaging. So, often the result is more, looser clusters than in single linkage but fewer, tighter clusters than in complete linkage. For these reasons, average linkage is fairly popular in applications;

Divisive Hierarchical Clustering

Divisive hierarchical clustering treats the data initially as being one group that gets split successively using a distance measure, in principle until each subset consists of a single element. Clearly, divisive hierarchical clustering is the reverse of agglomerative clustering. In fact, every technique for agglomerative clustering ultimately unites all the data points into one cluster, so doing it backwards gives a technique for divisive clustering. Conversely, every technique for divisive clustering can be done backwards to give an agglomerative technique. However, the different starting points for the two sorts of hierarchical clustering make different techniques seem more natural even though they are mathematically equivalent.

At its root, hierarchical divisive clustering splits sets according to fixed rules. The basic template for hierarchical divisive clustering is:

Start with a sample x_i, $i = 1, ..., n$ regarded as a single cluster of n data points and a dissimilarity d defined for all pairs of points in the sample. Fix a threshold t for deciding whether or not to split a cluster.

- First, determine $d(x_i, x_j)$, the distance between all pairs of data points, and choose the pair with the largest distance dmax between them.
- Compare d_{max} to t. If $d_{max} > t$, then divide the single cluster in two by using the selected pair as the first elements in two new clusters. The remaining n?2 data points are put into one of the two new clusters: x_ℓ is added to the new cluster containing xi if $d(x_i, x_\ell) < d(x_j, x_\ell)$; otherwise x_ℓ is added to the new cluster containing x_i.
- At the second stage, the values of $d(x_i, x_j)$ are found for x_i, x_j within one of the two new clusters to find the pair in the cluster with the largest distance d_{max} between them. If $d_{max} < t$, the division of the cluster stops and the other cluster is considered. Then the procedure repeats on the clusters generated from this iteration.

The procedure can be run up to $n?1$ times, which gives n singleton clusters, in which case a level in the dendrogram must be chosen to specify a clustering, or the procedure can be run until $d_{max} > t$s for all the existing clusters.

Divisive methods are often more computationally demanding than agglomerative methods because decisions must be made about dividing a cluster in two in all possible ways. In fact, the template shows there are two basic problems that must be resolved to implement a divisive procedure. First, a cluster to split must be chosen and then the actual split must be found.

The first problem has three standard answers. (i) Split every cluster until a complete binary dendrogram is obtained. (ii) Always split the largest cluster. (iii) If w is the center of the cluster, choose the cluster with the largest variability around w; i.e., the j with the highest value of $||x_i - w||^2$. The first two are simple: The first ignores whether the clusters are meaningful and lets the practitioner choose where along the dendrogram to stop. Always splitting the largest cluster tends to generate dendrograms where all leaves are about the same size. The third may be the best because it is sensitive to the scatter of points in clusters. It will tend to produce clusterings that are tighter than the others. Other cluster selection rules can be readily formulated; the specific rule chosen tends to act like extra information, favoring one class of clusterings over others

Lecture 4

Association rule mining

Association rules are used for finding potentially interesting and important relations between attributes in (possibly huge) collections of data. Association rules have originally been used in the market basket analysis With their help, merchants can plan arrangement of items on the shelves in their shops, sales strategies, and special discounts. For example, a rule may say that there is a high probability that a customer buying the item A, will also buy the item B. Such a rule suggests that it is beneficial to lower the price of item A while keeping or even raising the price of the item B. Merchants may organize items on shelves so that items A and ? are in close proximity, thus easing the customers' shopping. Alternatively, items A and ? may be placed far from each other so that customers are forced to review several other shelves and possibly buy something else.

There are applications where we are interested in the relations among the different variables that describe the data. The idea is that these relations can reveal some notion of causality through attribute correlation.There are different models focused on discovering these kinds of relationships like bayesian networks for instance. Even clustering methods can be used for finding these relationships. In this chapter we will focus on a more simple model, but that is in essence related to the techniques from these methods.The idea is to lose focus on the examples in the data and to use them only for assess variable relationships. The methods that we are going to describe focus specifically on discrete valued variables, and are usually described for binary data, but can be easily applied to multivalued discrete data using binarization as preprocess.

A binary dataset is a collection of examples where each row is described by a binary attribute. For instance:

	A	B	C	D	...
T1	1	0	0	1	...
T2	0	1	1	1	...
T3	1	0	1	0	...
T4	0	0	1	0	...

The usual meaning for the variable is that an specific event occurs or not for an example. The typical domain does not allow for having many attributes that occur for an example, for instance, individual purchase transactions from a supermarket. This makes easy to represent the data as a sparse matrix that only stores the occurrences of the attributes for each example.

In the market basket analysis a learning example represents a single customer's purchase (contents of his/her market basket). Originally, all attributes were binary, indicating presence (1) or absence (0) of an item. This representation can be generalized so that, instead of indicating the item's presence, the attribute counts its quantity. Since the number of attributes is potentially huge (often more than 105), and purchases rarely exceed few tens of items, data matrices are necessarily very sparse. Learning examples are therefore most often represented as lists of items whose counts are greater than zero. Since the number of all possible association rules is huge - much larger than the number of all possible classification rules - the rule space needs to be sensibly reduced.

Association rule mining can be defined formally as follows: $I = i_1, i_2, ..., i_m$ is a set of literals, or items. For example, goods such as milk, sugar and bread for purchase in a store are items; and $A_i = v$ is an item, where v is a domain value of the attribute A_i, in a relation $R(A_1, ..., A_n)$.

X is an *itemset* if it is a subset of I. For example, a set of items for purchase from a store is an itemset; and a set of $A_i = v$ is an itemset for the relation $R(PID, A_1, A_2, ..., A_n)$, where PID is a key.

$D = t_i, t_{i+1}, ..., t_n$ is a set of transactions, called a *transaction database*, where each transaction t has a *tid* and a t-itemset t = (tid, t-itemset). For example, a customer?s shopping trolley going through a checkout is a transaction; and a tuple $(v_1, ..., v_n)$ of the relation $R(A_1, ..., A_n)$ is a transaction.

A transaction t contains an itemset X iff, for all items, where $i \in X$, i is a t-itemset. For example, a shopping trolley contains all items in X when going through the checkout; and for each $A_i = v_i$ in X, v_i occurs at position i in the tuple $(v_1, ..., v_n)$.

There is a natural lattice structure on the itemsets 2^I, namely the subset/superset structure. Certain nodes in this lattice are natural grouping categories of interest (some with names). For example, items from a particular department such as clothing, hardware, furniture, etc; and, from within say clothing, children?s, women?s and men?s clothing, toddler?s clothing, etc.

An itemset X in a transaction database D has a support, denoted as supp(X). (For descriptive convenience in this book, we sometimes use p(X) to stand for supp(X).) This is the ratio of transactions in D containing X. Or

$$supp(X) = \frac{X(t)}{|D|}$$

where $X(t) = tinD|tcontainsX$.

An itemset X in a transaction database D is called as a large, or frequent, itemset if its support is equal to, or greater than, the threshold minimal support (minsupp) given by users or experts.

The negation of an itemset X is $\neg X$. The support of $\neg X$ is $supp(\neg X) =$

$1 - supp(X)$.

Each association rule has two quality measurements, support and confidence. Usually we are interested only in association rules that cover enough learning examples (say, more than 1% of purchases), and are sufficiently accurate (say, over 80%). Measures that define these two criteria are support and confidence. For a rule

$$A \longrightarrow B$$

where A and ? are conjunctions of (*attribute=value*) conditions, they are defined as follows:

$$support = \frac{n(A \wedge B)}{n} = P(A \wedge B)$$
$$confidence = \frac{n(A \wedge B)}{n(A)} = P(B|A)$$

Here, n is the number of all examples and ? the number of examples that satisfy the condition X. Support is therefore the probability that, for a randomly chosen example, both the antecedent and the consequent conditions are satisfied. Confidence is the actual accuracy of the rule when the antecedent condition is satisfied. Since databases where association rules are generated are typically huge - much too large to fit in the primary (fast) computer memory, the most important part in design of algorithms for mining association rules is to minimize the number of passes through the entire database.

Mining association rules can be broken down into the following two subproblems.

- Generating all itemsets that have support greater than, or equal to, user specified minimum support. That is, generating all frequent itemsets.

- Generating all rules that have minimum confidence in the following simple way: For every frequent itemset X, and any $B \subset X$, let $A = X?B$. If the confidence of a rule $A \longrightarrow B$ is greater than, or equal to, the minimum confidence (or $supp(X)/supp(A) \geq minconf$), then it can be extracted as a valid rule

Lecture 5

Apriori Algorithm

APriori is one of the first efficient algorithms for generating all association rules that satisfy the given conditions on support and confidence. APriori generates association rules in two steps. In the first step attributes are ordered according to some criterion (e.g., alphabetically); this eases checking for set equivalence. The algorithm then generates all i-tuples of conditions with sufficiently high support. These z-tuples (also called frequent itemsets) are subsequently used to generate rules that exceed the given confidence threshold.

Generation of frequent itemsets (i-tuples) starts with 1 -tuples, that is, with a single pass through the entire database where the algorithm searches for all conditions with sufficiently high support. From i-tuples, (i + l)-tuples are generated such that all the subsets of size i of their conditions have sufficiently high support. It is not necessary to pass through the database for this step - all highly supported i-tuples have already been collected and stored in the previous step. Support for the new (i + l)-tuples is checked in another pass through the database. For each i it is necessary to traverse the database only once. Since, by increasing the number of conditions i, support of the itemsets quickly decreases, the set of (i + 1)-tuples quickly becomes empty. This concludes the first step of the APriori algorithm.

In the second step, frequent itemsets (i-tuples) are used to generate all possible association rules that are subsequently checked for their confidence. Typically, there are too many candidate rules for exhaustive checking. For three conditions A, ? and C all possible rules are

$$A \wedge B \longrightarrow C \quad C \longrightarrow A \wedge B$$

$$A \wedge C \longrightarrow B \quad B \longrightarrow A \wedge C$$

$$B \wedge C \longrightarrow A \quad A \longrightarrow B \wedge C$$

Again we have to deal with a combinatorial explosion; this however can be avoided by introducing the following constraint. If rule

$$A \longrightarrow B \wedge C$$

exceeds the threshold values of the minimal support and confidence, the same holds for rules

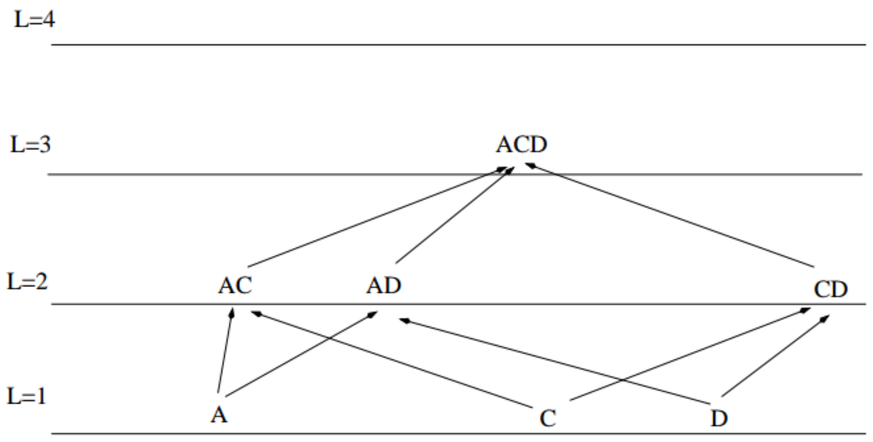
$$A \wedge B \longrightarrow C$$

$$A \wedge B \longrightarrow C$$

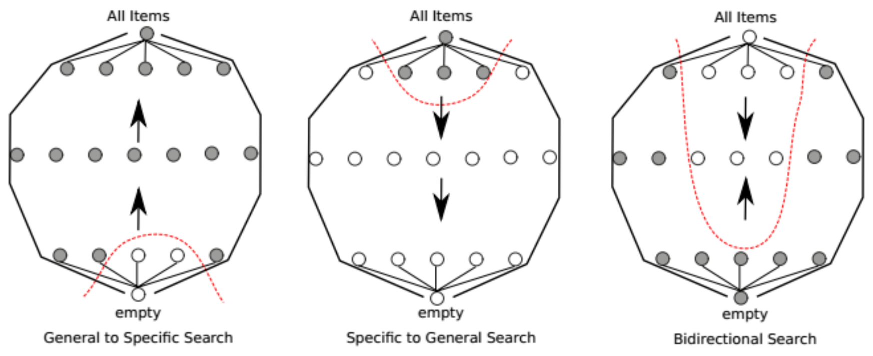
Therefore, once we have the first of the above rules, the others need not be considered at all. This constraint is very useful because it suggests an efficient strategy for generating association rules by incrementally extending their consequent parts. We start with all rules that have only a single conjunction in the consequent part, and keep only those with sufficiently high confidence. Then we use these rules to obtain the rules with the consequent part consisting of two conjunctions by extending the rules in the manner that does not violate the constraint (9.4). Again, only rules with sufficiently high confidence are kept. The process incrementally continues until there are no more sufficiently confident rules available for extension. In practice it often turns out that the number of conjunctions in consequent parts is rather low - usually 2 or 3. This concludes the second part of the APriori algorithm. For rule generation the access to the database is no longer required. For confidence calculation only support of itemsets that constitute the antecedent part is necessary. They, however, were all calculated in the first part of the APriori algorithm and are normally stored in a hash table.

```
Algorithm: Apriori (R,min_sup,min_conf)
C,CCan,CTemp:set of frequent subsets
RAs:Set of association rules, L:integer
L=1
CCan=Frequent_sets_1(R,fr_min)
RAs=$\emptyset$
while $CCan \neq \emptyset$ do
    L=L+1
    CTemp=Candidate_sets(L,R,CCan) = $C(\mathcal{F}_{l+1}(r))$
    C=Frequent_sets(CTemp,min_sup)
    RAs=RAs $\cup$ Confidence_rules(C,min_conf)
    CCan=C
end
```

e will find that some attributes will have a frequency larger than a specific support threshold. Assume that these attributes are {A, C, D}, the next graph shows the space of candidates after prunning candidates that include the non frequent item B.



The patterns of length 2 that are not prunned will be the possible candidates for frequent itemsets of this length. If we assume that all these patterns are frequent we will have that there is only one pattern left as candidate for frequent patterns of length three, so the algorithms stops at this pattern length. Something that has to be noticed is that the specific value used as min sup has effect on the computational cost of the search because it limits what is going to be explored. If it is too high, only a few patterns will appear (we could miss interesting rare occurrences), if it is too low the computational cost will be too high (too many associations will appear). The extreme is when the value is 0, in this case all items are frequent, so all the possible combinations of items will be generated. Unfortunately the threshold value can not be known beforehand. Sometimes, multiple minimum supports could be used, different for different groups of items, to explore the interesting associations. This could reduce the combinatorial explosion due to very frequent items and focus on less frequent but more interesting ones. These thresholds are domain specific and some previous analysis of the data should be performed to guess these values.



Other elements of the problem also have an influence on the computational cost, such as the number of different items, the more items there are the more space is needed to count its support, the size of the database, because the algorithm has to perform multiple passes to count the support, and the average transaction width that affects the maximum length of frequent itemsets. Some specialized algorithms have been developed to deal with these problems in some domains, for example, in bioinformatics, it is usual to have long frequent itemsets. The number of possible frequent itemsets obtained by the apriori algorithm is one of its main problems. This can be improved by targeting subsets of itemsets that have specific properties. There are two kinds of itemsets that are used for reducing the number of candidates:

- *Maximal frequent itemsets*, defined as frequent itemset for which none of its immediate supersets are frequent
- Closed frequent itemsets:, defined as frequent itemset for which none of its immediate supersets have the same support

This is the relationship among these subsets and the whole set of frequent itemsets. Maximal frequent itemsets $\subseteq$ Closed frequent itemsets $\subseteq$ Frequent itemsets The actual reduction on the number of final item sets will depend on the domain and the support threshold, but in practice the number of candidates are sometimes orders of magnitude smaller. There are variations of the way the itemsets are explored by the apriori algorithm that can reduce the computational cost depending on the characteristics of the items. The described algorithm explores the lattice of itemsets from general-to-specific, but the items can be explored from specific-to-general when the maximum itemset size is large or the transactions are dense. In this case an item has to be decomposed in all its subsets if it is not frequent and exploration stops for a candidate when it is maximal frequent. A didirectional search can also be applied using the information from one direction to prune candidates from the other direction. See figure 6.3. Other methods for reducing the computational cost include defining equivalence classes for the items so several items are counted as the same, or using for example prefixes (or suffixes) as equivalence classes when it makes sense in the domain. Also the way the itemsets are explored can be changed from a breadth first search (as apriori has been defined) that can be memory demanding, to a depth first search, that requires

Lecture 6

F-P Growth Algorithm

In Data Mining the task of finding frequent pattern in large databases is very important and has been studied in large scale in the past few years. Unfortunately, this task is computationally expensive, especially when a large number of patterns exist.

The FP-Growth Algorithm, proposed by Han, is an efficient and scalable method for mining the complete set of frequent patterns by pattern fragment growth, using an extended prefix-tree structure for storing compressed and crucial information about frequent patterns named frequent-pattern tree (FP-tree). In his study, Han proved that his method outperforms other popular methods for mining frequent patterns, e.g. the Apriori Algorithm and the TreeProjection. In some later works it was proved that FP-Growth has better performance than other methods, including Eclat and Relim. The popularity and efficiency of FP-Growth Algorithm contributes with many studies that propose variations to improve his performance

The Algorithm

The FP-Growth Algorithm is an alternative way to find frequent itemsets without using candidate generations, thus improving performance. For so much it uses a divide-and-conquer strategy [17]. The core of this method is the usage of a special data structure named frequent-pattern tree (FP-tree), which retains the itemset association information.

In simple words, this algorithm works as follows: first it compresses the input database creating an FP-tree instance to represent frequent items. After this first step it divides the compressed database into a set of conditional databases, each one associated with one frequent pattern. Finally, each such database is mined separately. Using this strategy, the FP-Growth reduces the search costs looking for short patterns recursively and then concatenating them in the long frequent patterns, offering good selectivity.

In large databases, it?s not possible to hold the FP-tree in the main memory. A strategy to cope with this problem is to firstly partition the database into a set of smaller databases (called projected databases), and then construct an FP-tree from each of these smaller databases.

The next subsections describe the FP-tree structure and FP-Growth Algorithm, finally an example is presented to make it easier to understand these concepts.

FP-Tree Structure

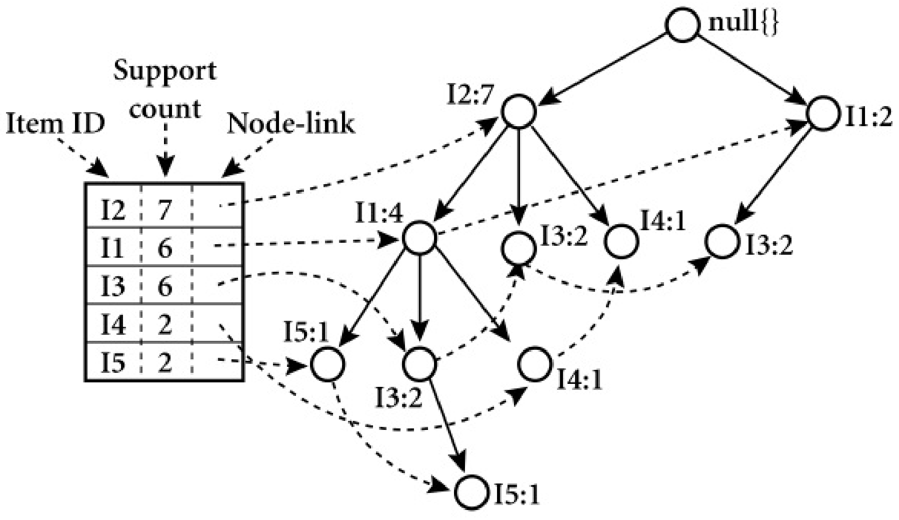
The frequent-pattern tree (FP-tree) is a compact structure that stores quantitative information about frequent patterns in a database.

Han defines the FP-tree as the tree structure io below:

- One root labeled as ?null? with a set of item-prefix subtrees as children, and a frequent-item-header table (presented in the left side of Figure);
- Each node in the item-prefix subtree consists of three fields:
    1. Item-name: registers which item is represented by the node;
    2. Count: the number of transactions represented by the portion of the path reaching the node;
    3. Node-link: links to the next node in the FP-tree carrying the same item-name, or null if there is none.
- Each entry in the frequent-item-header table consists of two fields:

1. Item-name: as the same to the node;
2. Head of node-link: a pointer to the first node in the FP-tree carrying the item-name.

Additionally the frequent-item-header table can have the count support for an item



FP-tree construction

- Input: A transaction database DB and a minimum support threshold ?.
- Output: FP-tree, the frequent-pattern tree of DB.
- Method: The FP-tree is constructed as follows.

  1. Scan the transaction database DB once. Collect F, the set of frequent items, and the support of each frequent item. Sort F in support-descending order as FList, the list of frequent items.

2. Create the root of an FP-tree, T, and label it as ?null?. For each transaction Trans in DB do the following:

   – Select the frequent items in Trans and sort them according to the order of FList. Let the sorted frequent-item list in Trans be [p — P], where p is the first element and P is the remaining list. Call insert tree([p — P], T).

   – The function insert tree([p — P], T) is performed as follows. If T has a child N such that N.item-name = p.item-name, then increment N ?s count by 1; else create a new node N , with its count initialized to 1, its parent link linked to T , and its node-link linked to the nodes with the same item-name via the node-link structure. If P is nonempty, call insert tree(P, N) recursively.

By using this algorithm, the FP-tree is constructed in two scans of the database. The first scan collects and sort the set of frequent items, and the second constructs the FP-Tree.

FP-Growth Algorithm

After constructing the FP-Tree it?s possible to mine it to find the complete set of frequent patterns. To accomplish this job, Han in [1] presents a group of lemmas and properties, and thereafter describes the FP-Growth Algorithm as presented below

- Input: A database DB, represented by FP-tree constructed according to Algorithm 1, and a minimum support threshold ?.

- Output: The complete set of frequent patterns.

- Method: call FP-growth(FP-tree, null).
- Procedure FP-growth(Tree, a) {

```
1. if Tree contains a single prefix path then { // Mining single prefix-path FP-tree
2. let P be the single prefix-path part of Tree;
3. let Q be the multipath part with the top branching node replaced by a null root;
4. for each combination (denoted as ß) of the nodes in the path P do
5. generate pattern $\beta \cup$ a with support = minimum support of nodes in $\beta$;
6. let freq pattern set(P) be the set of patterns so generated;
7. else let Q be Tree;
8. for each item ai in Q do { Mining multipath FP-tree
9. generate pattern $\beta = a_i \cup a$ with $support = a_i.support$;
10. return(freq pattern set(P) $\cup$ freq pattern set(Q) $\cup$ (freq pattern set(P) $\times$ freq pattern set(Q)))
```

Lecture 7

Probabilistic clustering

Clustering objects requires some notion of how similar they are. We have seen how to cluster using distance in feature space, which is a natural way of thinking about similarity. Another way to think about similarity is to ask whether two objects have high probability under the same probability model. This can be a convenient way of looking at things when it is easier to build probability models than it is to measure distances. It turns out to be a natural way of obtaining soft clustering weights (which emerge from the probability model). And it provides a framework for our first encounter with an extremely powerful and general algorithm, which you should see as a very aggressive generalization of k-means.

Probabilistic model-based clustering techniques have been widely used and have shown promising results in many applications, ranging from image segmentation, handwriting recognition, document clustering, topic modeling to information retrieval. Model-based clustering approaches attempt to optimize the fit between the observed data and some mathematical model using a probabilistic approach. Such methods are often based on the assumption that the data are generated by a mixture of underlying probability distributions. In practice, each cluster can be represented mathematically by a parametric probability distribution, such as a Gaussian or a Poisson distribution. Thus, the clustering problem is transformed into a parameter estimation problem since the entire data can be modeled by a mixture of K component distributions.Data points (or objects) that belong most likely to the same distribution can then easily be defined as clusters.

Suppose we have a set of data points $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$ consisting of N

observations of a D-dimensional random variable $\mathbf{x}$. The random variable $\mathbf{x}_n$ is assumed to be distributed according to a mixture of K components. Each component (i.e., cluster) is mathematically represented by a parametric distribution. An individual distribution used to model a specific cluster is often referred to as a component distribution. The entire data set is therefore modeled by a mixture of these distributions. Formally, the mixture distribution, or probability density function, of $\mathbf{x}_n$ can be written as

$$p(\mathbf{x}_n) = \sum_{k=1}^{K} \pi_k p(\mathbf{x}_n|\theta_k)$$

where $\pi_1, ..., \pi_K$ are the *mixing probabilities* (i.e., mixing coefficients or weights), each θ_k is the set of parameters specifying the kth component, and $p(\mathbf{x}_n|\theta_k)$ is the component distribution. In order to be valid probabilities, the mixing probabilities π_k must satisfy

$$0 \leq \pi_k \leq 1(k = 1, ..., K), \texttt{and} \sum_{k=1}^{K} \pi_k = 1$$

An obvious way of generating a random sample $\mathbf{x}_n$ with the mixture model is as follows. Let z_n be a categorical random variable taking on the values $1, ..., K$ with probabilities $p(z_n = k) = \pi_k$ (also denoted as $p(z_{nk} = 1) = \pi_k$). Suppose that the conditional distribution of xn given $z_n = k$ is $p(\mathbf{x}_n|\theta_k)$. Then the marginal distribution of x_n is obtained by summing the joint distribution over all possible values of z_n to given $p(x_n)$ as. In this context, the variable z_n can be thought of as the component (or cluster) label of the random sample x_n. Instead of using a single categorical variable z_n, we introduce a K-dimensional binary random vector z_n to denote the component label for x_n. The K-dimensional random variable zn has a 1-of-K representation, in which one of the element $z_{nk} = (z_n)_k$ equals to 1, and all other elements equal to 0, denoting the component of origin of xn is equal to k. For example, if we have a variable with K = 5 clusters and a particular observation xn of the variable happens to correspond to the cluster where zn4 = 1, then zn will be represented by zn = (0,0,0,1,0)T . Note that the values of znk satisfy $z_{nk} \in \{0,1\}$ and $\sum_{k=1}^{K} z_{nk} = 1$. Because z_n uses a 1-of-K representation, the marginal distribution over z_n is specified in terms of mixing probabilities ?k, such that

$$p(\mathbf{z}_n) = \pi_1^{z_{n1}} \pi_2^{z_{n2}} ... \pi_3^{z_{n3}} = \prod_{k=1}^{K} \pi_k^{z_{nk}}$$

Similarly, the conditional distribution of $\mathbf{x}_n$ given $\mathbf{z}_n$ can be written in the form

$$p(\mathbf{x}_n|\mathbf{z}_n) = \prod_{k=1}^{K} p(\mathbf{x}_n|\theta_k)^{z_{nk}}$$

The joint distribution is given by $p(\mathbf{z}_n)p(\mathbf{x}_n|\mathbf{z}_n)$, and the marginal distribution of $\mathbf{z}_n$ is obtained as

$$p(\mathbf{x}_n) = \sum_{\mathbf{z}_n} p(\mathbf{z}_n)p(\mathbf{x}_n|\mathbf{z}_n) = \sum_{k=1}^{K} \pi_k p(\mathbf{x}_n|\theta)$$

Thus, the above marginal distribution of $\mathbf{x}$ is an equivalent formulation of the mixture model involving an explicit latent variable. The graphical representation of the mixture model is shown in. From the generative process point of view, a given set of data points could have been generated by repeating the following procedure N times, once for each data point $\mathbf{x}_n$.

- Choose a hidden component (i.e., cluster) label $\mathbf{z}_n$ $Mult_K(1, \pi)$. This selects the kth component from which to draw point $\mathbf{x}_n$.
- Sample a data point $\mathbf{x}_n$ from the kth component according to the conditional distribution $p(\mathbf{x}_n|\theta_k)$.

Because we have represented the marginal distribution in the form $p(\mathbf{x}_n) = \sum_{z_n} p(\mathbf{x}_n, \mathbf{z}_n)$, it follows that for every observed data point xn there is a corresponding latent variable $\mathbf{z}_n$. Using Bayes? theorem, we can obtain the conditional probability of $z_{nk} = 1$ given $\mathbf{x}_n$ as

$$p(z_{nk} = 1|\mathbf{x}_n) = \frac{p(z_{nk} = 1)p(\mathbf{x}_n|z_{nk} = 1)}{\sum_{j=1}^{K} p(z_{nj} = 1)p(\mathbf{x}_n|z_{nj} = 1)} = \frac{\pi_k p(\mathbf{x}_n|\theta_k)}{\sum_{j=1}^{K} \pi_j p(\mathbf{x}_n|\theta_j)}$$

where π_k (i.e., $p(z_{nk} = 1)$) is the *prior probability* that data point $\mathbf{x}_n$ was generated from component k, and $p(znk = 1|xn)$ is the *posterior probability* that the observed data point $\mathbf{x}_n$ came from component k. In the following, we shall use $\gamma(z_{nk})$ to denote $p(z_{nk} = 1|\mathbf{x}_n)$, which can also be viewed as the *responsibility* that component k takes for explaining the observation $\mathbf{x}_n$. In this formulation of the mixture model, we need to infer a set of parameters from the observation, including the *mixing probabilities* $\{\pi_k\}$ and the parameters for the *component distributions* $\{\theta_k\}$. The number of components K is considered fixed, but of course in many applications, the value of K is unknown and has to be inferred from the available data. Thus, the overall parameter of the mixture model is $\Theta = \pi_1, ..., \pi_K, \theta_1, ..., \theta_K$. If we assume that the data points are drawn independently from the distribution, then we can write the probability of generating all the data points in the form

$$p(\mathbf{X}|\Theta) = \prod_{n=1}^{N} \sum_{k=1}^{K} \pi_k p(\mathbf{x}_n|\theta_k)$$

or in a logarithm form

$$logp(\mathbf{X}|\Theta) = \sum_{n=1}^{N} log \sum_{k=1}^{K} \pi_k p(\mathbf{x}_n|\theta_k)$$

In statistics, maximum likelihood estimation (MLE) is an important statistical approach for parameter estimation,

$$\Theta_{ML} = \arg\max_{\Theta}\{\log p(\mathbf{X}|\Theta)\}$$

which considers the best estimate as the one that maximizes the probability of generating all the observations. Sometimes we have a priori information $p(\Theta)$ about the parameters, and it can be incorporated into the mixture models. Thus, the maximum a posteriori (MAP) estimation is used instead,

$$\Theta_{MAP} = \arg\max_{\Theta}\{\log p(\mathbf{X}|\Theta) + \log p(\Theta)\}$$

which considers the best estimate as the one that maximizes the posterior probability of Θ given the observed data. MLE and MAP give a unified approach to parameter estimation, which is welldefined in the case of the normal distribution and many other problems. As mentioned previously, each cluster is mathematically represented by a parametric distribution. In principle, the mixtures can be constructed with any types of components, and we could still have a perfectly good mixture model. In practice, a lot of effort is given to parametric mixture models, where all components are from the same parametric family of distributions, but with different parameters. For example, they might all be Gaussians with different means and variances, or all Poisson distributions with different means, or all power laws with different exponents. In the following section, we will introduce the two most common mixtures, mixture of Gaussian (continuous) and mixture of Bernoulli (discrete) distributions.

Lecture 8

Gaussian mixture model.

The most well-known mixture model is the Gaussian mixture model (GMM), where each component is a Gaussian distribution. Recently, GMM has been widely used for clustering in many applications, such as speaker identification and verification, image segmentation, and object tracking. The Gaussian, also known as the normal distribution, is a widely used model for the distribution of continuous variables. In the case of a single variable x, the Gaussian distribution can be written in the form

$$\mathcal{N}\left(x|\mu,\sigma^2\right)=\frac{1}{\left(2\pi\sigma^2\right)^{1/2}}\exp\left\{-\frac{1}{2\sigma^2}(x-\mu)^2\right\}$$

where μ is the mean and σ^2 is the variance. Figure 3.2(a) shows plots of the Gaussian distribution for various values of the parameters. For a D-dimensional vector $\mathbf{x}$, the multivariate Gaussian distribution takes the form

$$\mathcal{N}(\mathbf{x}|\mu,\Sigma)=\frac{1}{(2\pi)^{D/2}}\frac{1}{|\Sigma|^{1/2}}\exp\left\{-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)\right\}$$

where ? is a D-dimensional mean vector, Σ is a $D\times D$ covariance matrix, and denotes the determinant of Σ. Figure shows contours of the Gaussian distribution for various values of the parameters. In the Gaussian mixture model, each component is represented by the parameters of a multivariate Gaussian distribution $p(\mathbf{x}_k|\theta_k)=\mathcal{N}(\mathbf{x}_n|\mu_k,\Sigma_k)$. Based on, the Gaussian mixture distribution can be written as a linear superposition of Gaussians in the form

$$p\left(\mathbf{x}_n|\Theta\right)=p\left(\mathbf{x}_n|\pi,\mu,\Sigma\right)=\sum_{k=1}^{K}\pi_k\mathcal{N}\left(\mathbf{x}_n|\mu_k,\Sigma_k\right)$$

For a given set of observations $\mathbf{X}$, the log-likelihood function is given by

$$l(\Theta) = \log p(\mathbf{X}|\Theta) = \sum_{n=1}^{N} \log p\left(\mathbf{x}_n|\Theta\right) = \sum_{n=1}^{N} \log \sum_{k=1}^{K} \pi_k \mathcal{N}\left(\mathbf{x}_n|\mu_k, \Sigma_k\right)$$

To find maximum likelihood solutions that are valid at local maxima, we compute the derivatives of $logp(\mathbf{X}|\pi, \mu, \Sigma)$ with respect to π_k, π_k, and Σ_k, respectively. The derivative with respect to the mean μ_k is given by

$$\frac{\partial l}{\partial \mu_k} = \sum_{n=1}^{N} \frac{\pi_k \mathcal{N}\left(\mathbf{x}_n|\mu_k, \Sigma_k\right)}{\sum_{j=1}^{K} \pi_j \mathcal{N}\left(\mathbf{x}_n|\mu_j, \Sigma_j\right)} \Sigma_k^{-1}\left(\mathbf{x}_n - \mu_k\right) = \sum_{n=1}^{N} \gamma\left(z_{nk}\right) \Sigma_k^{-1}\left(\mathbf{x}_n - \mu_k\right)$$

where we have used eq and eq for the Gaussian distribution and the responsibilities (i.e., posterior probabilities), respectively. Setting this derivative to zero and multiplying by Σ_k, we obtain

$$\mu_k = \frac{\sum_{n=1}^{N} \gamma\left(z_{nk}\right) \mathbf{x}_n}{\sum_{n=1}^{N} \gamma\left(z_{nk}\right)}$$

and

$$\gamma\left(z_{nk}\right) = \frac{\pi_k \mathcal{N}\left(\mathbf{x}_n|\mu_k, \Sigma_k\right)}{\sum_{j=1}^{K} \pi_j \mathcal{N}\left(\mathbf{x}_n|\mu_j, \Sigma_j\right)}$$

We can interpret that the mean μ_k for the kth Gaussian component is obtained by taking a weighted mean of all the points in the data set, in which the weighting factor corresponds to the posterior probability $\gamma(z_{nk})$ that component k was responsible for generating $\mathbf{x}_n$.

Similarly, we set the derivative of $logp(\mathbf{X}|\pi, \mu, \Sigma)$ with respect to Σ_k to zero, and we obtain

$$\Sigma_k = \frac{\sum_{n=1}^{N} \gamma\left(z_{nk}\right)\left(\mathbf{x}_n - \mu_k\right)\left(\mathbf{x}_n - \mu_k\right)^T}{\sum_{n=1}^{N} \gamma\left(z_{nk}\right)}$$

Similar to eq, each data point is weighted by the conditional probability generated by the corresponding component and with the denominator given by the effective number of points associated with the corresponding component.

The derivative of $logp(\mathbf{X}|\pi, \mu, \Sigma)$ with respect to the mixing probabilities π_k requires a little more work, because the values of π_k are constrained to be positive and sum to one. This constraint can be handled using a Lagrange multiplier and maximizing the following quantity

$$\log p(\mathbf{X}|\pi, \mu, \Sigma) + \lambda \left(\sum_{k=1}^{K} \pi_k - 1 \right)$$

After simplifying and rearranging we obtain

$$\pi_k = \frac{\sum_{n=1}^{N} \gamma \left(z_{nk} \right)}{N}$$

so that the mixing probabilities for the kth component are given by the average responsibility that the component takes for explaining the data points.

It is worth noting that the Equations (3.12), (3.14), and (3.15) are not the closed-form solution for the parameters of the mixture model. The reason is that these equations are intimately coupled with Equation (3.13). More specifically, the responsibilities ?(znk) given by Equation (3.13) depend on all the parameters of the mixture model, while all the results of (3.12), (3.14), and (3.15) rely on ?(znk). Therefore, maximizing the log likelihood function for a Gaussian mixture model turns out to be a very complex problem. An elegant and powerful method for finding maximum likelihood solutions for models with latent variables is called the Expectation-Maximization algorithm or EM algorithm [23].

However, the above equations do suggest an iterative solution for the Gaussian mixture model, which turns out to be an instance of the EM algorithm for the particular case of the Gaussian mixture model. Here we shall give such an iterative algorithm in the context of Gaussian mixture model.

This algorithm is started by initializing with guesses about the parameters, including the means, covariances, and mixing probabilities. Then we alternative between two updating steps, the expectation step and maximization step. In the expectation step, or E-step, we use the current parameters to calculate the responsibilities (i.e., posterior probabilities) according to. In the maximization step, or Mstep, we maximize the log-likelihood with the updated responsibilities, and reestimate the means, covariances, and mixing coefficients using.

As illustrated in Figure, the EM algorithm for a mixture of two Gaussian components is applied to a random generated data set. Plot (a) shows the data points together with the random initialization of the mixture model in which the two Gaussian components are shown. Plot (b) shows he result of the initial E-step, in which each data point is depicted using a proportion of white ink and black ink according to the posterior probability of having been generated by the corresponding component. Plot (c) shows the situation after the first M-step, in which the means and covariances of both components have changed. Plot (d) shows the results after 47 cycles of EM, which is close to convergence.