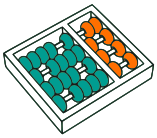

UNICAMP

Clustering [1]

Lesson No. 13

Gustavo de J. Merli - 262948

1 Introduction

Clustering is the process of grouping similar objects together. There are two kinds of inputs we might use. In similarity-based clustering, the input to the algorithm is an N x N dissimilarity matrix or distance matrix D. In feature-based clustering, the input to the algorithm is an N x D feature matrix or design matrix X. Similarity-based clustering has the advantage that it allows for easy inclusion of domain-specific similarity or kernel functions. Feature based clustering has the advantage that it is applicable to "raw", potentially noisy data.

1.1 Measuring (dis)similarity

A dissimilarity matrix D is a matrix where $d_{i,i} = 0$ and $d_{i,j} \geq 0$ is a measure of "distance" between objects i and j. The most common way to define dissimilarity between objects is in terms of the dissimilarity of their attributes:

$$\Delta(x_i, x_{i'}) = \sum_{j=1}^{D} \Delta_j(x_{ij}, x_{i'j}) \tag{1}$$

Some common attribute dissimilarity functions are as follows:

- Squared (Euclidean distance):

$$\Delta_j(x_{ij}, x_{i'j}) = (x_{ij} - x_{i'j})^2 \tag{2}$$

- City block distance:

$$\Delta_j(x_{ij}, x_{i'j}) = |x_{ij} - x_{i'j}| \tag{3}$$

- If x_i is a vector (e.g., a time-series of real-valued data), it is common to use the correlation coefficient. If the data is standardized, then $corr[x_i, x_{i'}] = \sum_j x_{ij} x_{i'j}$, and hence $\sum_j (x_{ij} - x_{i'j})^2 = 2(1 - corr[x_i, x_{i'}])$.
- For ordinal variables, such as low, medium, high, it is standard to encode the values as real-valued numbers, say $1/3, 2/3, 3/3$ if there are 3 possible values. One can then apply any dissimilarity function for quantitative variables, such as squared distance.
- Hamming Distance: For categorical variables, such as red, green, blue, we usually assign a distance of 1 if the features are different, and a distance of 0 otherwise. Summing up over all the categorical features gives

$$\Delta(x_i, x_{i'}) = \sum_{j=1}^{D} \mathbb{I}(x_{ij} \neq x_{i'j}) \tag{4}$$

1.2 Evaluating the output of clustering methods

Intuitively, the goal of clustering is to assign points that are similar to the same cluster, and to ensure that points that are dissimilar are in different clusters. There are several ways of measuring these quantities. However, these internal criteria may be of limited use. An alternative is to rely on some external form of data with which to validate the method. For example, suppose we have labels for each object. (Equivalently, we can have a reference clustering; given a clustering, we can induce a set of labels and vice versa.) Then we can compare the clustering with the labels using various metrics which we describe below. We will use some of these metrics later, when we compare clustering methods.

1.2.1 Purity

Let N_{ij} be the number of objects in cluster i that belong to class j, and let $N_i = \sum_{j=1}^{C} N_{ij}$ be the total number of objects in cluster i. Define $p_{ij} = N_{ij}/N_i$; this is the empirical distribution over class labels for cluster i. We define the purity of a cluster as $pi = max_j p_{ij}$, and the overall purity of a clustering as

$$purity = \sum_i \frac{N_i}{N} p_i \tag{5}$$

1.2.2 Rand index

Let $U = \{u_1, ..., u_R\}$ and $V = \{v_1, ..., v_C\}$ be two different partitions of the N data points, i.e., two different (flat) clusterings. For example, U might be the estimated clustering and V is reference clustering derived from the class labels. Now define a 2 x 2 contingency table, containing the following numbers: TP is the number of pairs that are in the same cluster in both U and V (true positives); TN is the number of pairs that are in the different clusters in both U and V (true negatives); FN is the number of pairs that are in the different clusters in U but the same cluster in V (false negatives); and FP is the number of pairs that are in the same cluster in U but different clusters in V (false positives). A common summary statistic is the Rand index:

$$R = \frac{TP + TN}{TP + FP + FN + TN} \tag{6}$$

This can be interpreted as the fraction of clustering decisions that are correct. Clearly $0 \leq R \leq 1$.

1.2.3 Mutual information

Another way to measure cluster quality is to compute the mutual information between U and V. To do this, let $p_{UV}(i,j) = \frac{|u_i \cap v_j|}{N}$ be the probability that a randomly chosen object belongs to cluster u_i in U and v_j in V. Also, let $p_U(i) = |u_i|/N$ be the be the probability that a randomly chosen object belongs to cluster u_i in U; define $p_V(j) = |v_j|/N$ similarly. Then we have

$$\mathbb{I}(U,V) = \sum_{i=1}^{R} \sum_{j=1}^{C} p_{UV}(i,j) log \frac{p_{UV}(i,j)}{p_U(i) p_v(j)} \tag{7}$$

But this doesn't lie between 0 and 1. To compensate for this, we can use the **normalized mutual information**

$$NMI(U,V) = \frac{\mathbb{I}(U,V)}{(\mathbb{H}(U) + \mathbb{H}(V))/2} \tag{8}$$

2 Dirichlet process mixture models

In many cases, there is no well defined number of clusters. Even in the simple 2d height-weight data, it is not clear if the "correct" value of K should be 2, 3, or 4. It would be much better if we did not have to choose K at all.

In this section, we discuss infinite mixture models, in which we do not impose any a priori bound on K. To do this, we will use a non-parametric prior based on the **Dirichlet process** (DP). This allows the number of clusters to grow as the amount of data increases. It will also prove useful later when we discuss hiearchical clustering.

2.1 From finite to infinite mixture models

Consider a finite mixture model. The usual representation is as follows:

$$p(x_i | z_i = k, \theta) = p(x_i | \theta_k) \tag{9}$$

$$p(z_i = k | \pi) = \pi_k \tag{10}$$

$$p(\pi | \alpha) = Dir(\pi | (\alpha/K) 1_K) \tag{11}$$

The form of $p(\theta_k|\lambda)$ is chosen to be conjugate to $p(x_i|\theta_k)$. We can write $p(x_i|\theta_k)$ as $x_i \sim F(\theta_{z_i})$, where F is the observation distribution. Similarly, we can write $\theta_k \sim H(\lambda)$, where H is the prior.

There is an equivalent representation for this model. Here θ_i is the parameter used to generate observation x_i; these parameters are sampled from distribution G, which has the form

$$G(\theta) = \sum_{k=1}^{K} \pi_k \delta_{\theta_k}(\theta) \tag{12}$$

where $\pi \sim Dir(\frac{\alpha}{K} 1_K)$, and $\theta_k \sim H$. Thus we see that G is a finite mixture of delta functions, centered on the cluster parameters θ_k. The probability that θ_i is equal to θ_k is exactly π_k, the prior probability for that cluster.

If we sample from this model, we will always (with probability one) get exactly K clusters, with data points scattered around the cluster centers. We would like a more flexible model, that can generate a variable number of clusters. The way to do this is to replace the discrete distribution G with a random probability measure. Below we will show that the Dirichlet process, denoted $G \sim DP(\alpha, H)$, is one way to do this.

2.2 The Dirichlet process

A Dirichlet process is a distribution over probability measures $G : \Theta \to \mathbb{R}^+$, where we require $G(\theta) \geq 0$ and $\int_\Theta G(\theta) d\theta = 1$. The DP is defined implicitly by the requirement that $(G(T_1), ..., G(T_K))$ has a joint Dirichlet distribution

$$Dir(\alpha H(T_1), ..., \alpha H(T_k)) \tag{13}$$

for any finite partition $(T_1, ..., T_K)$ of Θ. If this is the case, we write $G \sim DP(\alpha, H)$, where α is called the **concentration parameter** and H is called the **base measure**.

2.3 Applying Dirichlet processes to mixture modeling

The DP is not particularly useful as a model for data directly, since data vectors rarely repeat exactly. However, it is useful as a prior for the parameters of a stochastic data generating mechanism, such as a mixture model. To create such a model, we define $G \sim DP(\alpha, H)$ and we can write the model as follows:

$$\pi \sim GEM(\alpha) \tag{14}$$

$$z_i \sim \pi \tag{15}$$

$$\theta_k \sim H(\lambda) \tag{16}$$

$$x_i \sim F(\theta_{z_i}) \tag{17}$$

2.4 Fitting a DP mixture model

The simplest way to fit a DPMM is to modify the collapsed Gibbs sampler.

$$p(z_i = k|z_{-i}, x, \alpha, \lambda) \propto p(z_i = k|z_{-i}, \alpha) p(x_i|x_{-i}, z_i = k, z_{-i}, \lambda) \tag{18}$$

where z_i is the observation. The first term is given by

$$p(z_i = k|z_{-i}, \alpha) = \frac{N_{k,-i}}{\alpha + N - 1} \tag{19}$$

if k has been seen before and

$$p(z_i = k|z_{-i}, \alpha) = \frac{\alpha}{\alpha + N - 1} \tag{20}$$

otherwise.

The second term is given by

$$p(x_i|x_{-i}, z_i = k^*, z_{-i}, \lambda) = p(x_i|\lambda) = \int p(x_i|\theta) H(\theta|\lambda) d\theta \tag{21}$$

where k^* is a new cluster.

3 Affinity propagation

The idea is that each data point must choose another data point as its exemplar or centroid; some data points will choose themselves as centroids, and this will automatically determine the number of clusters. More precisely, let $c_i \in \{1, ..., N\}$ represent the centroid for datapoint i.

The goal is to maximize the following function

$$S(c) = \sum_{i=1}^{N} s(i, c_i) + \sum_{k=1}^{N} \delta_k(c) \tag{22}$$

The first term measures the similarity of each point to its centroid. The second term is a penalty term that is $-\infty$ if some data point i has chosen k as its exemplar (i.e., $c_i = k$), but k has not chosen itself as an exemplar (i.e., we do not have $c_k = k$).

4 Spectral clustering

An alternative view of clustering is in terms of graph cuts. The idea is we create a weighted undirected graph W from the similarity matrix S, typically by using the nearest neighbors of each point. If we want to find a partition into K clusters, say $A_1, ..., A_K$, one natural criterion is to minimize the **normalized cut**

$$Ncut(A_1, ..., A_K) = \frac{1}{2} \sum_{k=1}^{K} \frac{W(A_k, \bar{A}_k)}{vol(A_k)} \tag{23}$$

where $\bar{A}_k = V \setminus A_k$ is the complement of A_k, $W(A, B) = \sum_{i \in A, j \in B} w_{ij}$, $vol(A) = \sum_{i \in A} d_i$, and $d_i = \sum_{j=1}^{N} w_{ij}$ is the weighted degree of node i.

We can formulate the Ncut problem in terms of searching for real-valued binary vectors $c_i \in (0, 1)^N$, where $c_{ik} = 1$ if point i belongs to cluster k, that minimize the objective. The result turns into an eigenvector problem known as **spectral clustering**. In general, the technique of performing eigenalysis of graphs is called **spectral graph theory**.

4.1 Graph Laplacian

Let W be a symmetric weight matrix for a graph, where $w_{ij} = w_{ji} \geq 0$. Let $D = diag(d_i)$ be a diaogonal matrix containing the weighted degree of each node. We define the graph Laplacian as follows:

$$L = D - W \tag{24}$$

Because each row sums to zero, we have that 1 is an eigenvector with eigenvalue 0. Furthermore, the matrix is symmetric and positive semi-definite.

Theorem 1. *The set of eigenvectors of* $\mathbf{L}$ *with eigenvalue 0 is spanned by the indicator vectors* $\mathbf{1}_{A_1}, ..., \mathbf{1}_{A_K}$*, where* A_k *are the* K *connected components of the graph.*

This suggests the following algorithm. Compute the first K eigenvectors u_k of L. Let $U = [u_1, ..., u_K]$ be an N x K matrix with the eigenvectors in its columns. Let $y_i \in \mathbb{R}^K$ be the i'th row of U. Since these y_i will be piecewise constant, we can apply K-means clustering to them to recover the connected components. Now assign point i to cluster k iff row i of Y was assigned to cluster k.

4.2 Normalized graph Laplacian

In practice, it is important to normalize the graph Laplacian, to account for the fact that some nodes are more highly connected than others. There are two comon ways to do this. One method creates a stochastic matrix where each row sums to one:

$$L_{rw} = D^{-1}L = I - D^{-1}W \tag{25}$$

The eigenspace of 0 is spanned by the indicator vectors 1_{A_k}. This suggests the following algorithm: find the smallest K eigenvectors of L_{rw}, create U, cluster the rows of U using K-means, then infer the partitioning of the original points. (Note that the eigenvectors/ values of L_{rw} are equivalent to the generalized eigenvectors/ values of L, which solve $Lu = \lambda DU$.)

The other method creates a symmetric matrix

$$L_{sym} = D^{-\frac{1}{2}}LD^{-\frac{1}{2}} = I - D^{-\frac{1}{2}}WD^{-\frac{1}{2}} \tag{26}$$

This time the eigenspace of 0 is spanned by $D^{\frac{1}{2}}1_{A_k}$. This suggest the following algorithm: find the smallest K eigenvectors of L_{sym}, create U, normalize each row to unit norm by creating $t_{ij} = u_{ij}/\sqrt{(\sum_k u_{ik}^2)}$, cluster the rows of T using K-means, then infer the partitioning of the original points.

5 Hierarchical clustering

There are two main approaches to hierarchical clustering: bottom-up or **agglomerative clustering**, and top-down or **divisive clustering**. Both methods take as input a dissimilarity matrix between the objects. In the bottom-up approach, the most similar groups are merged at each step. In the top-down approach, groups are split using various different criteria. We give the details below.

5.1 Agglomerative clustering

Agglomerative clustering starts with N groups, each initially containing one object, and then at each step it merges the two most similar groups until there is a single group, containing all the data.

The merging process can be represented by a **binary tree**, called a **dendrogram**. The initial groups (objects) are at the leaves (at the bottom of the figure), and every time two groups are merged, we join them in the tree. The height of the branches represents the dissimilarity between the groups that are being joined. The root of the tree (which is at the top) represents a group containing all the data. If we cut the tree at any given height, we induce a clustering of a given size.

5.1.1 Single link

In **single link clustering**, also called **nearest neighbor clustering**, the distance between two groups G and H is defined as the distance between the two closest members of each group:

$$d_{SL}(G,H) = \min_{i \in G, i' \in H} d_{i,i'} \tag{27}$$

5.1.2 Complete link

In **complete link clustering**, also called **furthest neighbor clustering**, the distance between two groups is defined as the distance between the two most distant pairs:

$$d_{CL}(G,H) = \max_{i \in G, i' \in H} d_{i,i'} \tag{28}$$

5.1.3 Average link

In practice, the preferred method is average link clustering, which measures the average distance between all pairs:

$$d_{avg}(G,H) = \frac{1}{n_G n_H} \sum_{i \in G} \sum_{i' \in H} d_{i,i'} \tag{29}$$

where n_G and n_H are the number of elements in groups G and H.

5.2 Divisive clustering

Divisive clustering starts with all the data in a single cluster, and then recursively divides each cluster into two daughter clusters, in a top-down fashion. Since there are $2^{N-1} - 1$ ways to split a group of N items into 2 groups, it is hard to compute the optimal split, so various heuristics are used. One approach is pick the cluster with the largest diameter, and split it in two using the K-means or K-medoids algorithm with $K = 2$.

Another method is called **dissimilarity analysis**, is as follows. We start with a single cluster containing all the data, $G = \{1, ..., N\}$. We then measure the average dissimilarity of $i \in G$ to all the other $i \in G$:

$$d_i^G = \frac{1}{n_G} \sum_{i' \in G} d_{i,i'} \tag{30}$$

We remove the most dissimilar object and put it in its own cluster H:

$$i^* = \underset{i \in G}{argmax}, G = G \backslash \{i^*\}, H = \{i^*\} \tag{31}$$

The algorithm stops if we pick a point i^* to move that maximizes the average dissimilarity to each $i' \in G$ but minimizes the average dissimilarity to each $i' \in H$:

$$d_i^H = \frac{1}{n_H} \sum_{i' \in H} d_{i,i'}, i^* = \underset{i \in G}{argmax} d_i^G - d_i^H \tag{32}$$

We continue to do this until $d_i^G - d_i^H$ is negative.

5.3 Choosing the number of clusters

It is difficult to choose the "right" number of clusters, since a hierarchical clustering algorithm will always create a hierarchy, even if the data is completely random. We will present a Bayesian approach to hierarchical clustering that nicely solves this problem.

5.4 Bayesian hierarchical clustering

There are several ways to make probabilistic models which produce results similar to hierarchical clustering. Here we present one particular approach called Bayesian hierarchical clustering.

5.4.1 The algorithm

Let $\mathscr{D} = \{x_1, ..., x_N\}$ represent all the data, and let $\mathscr{D}_i$ be the set of datapoints at the leaves of the substree T_i. At each step, we compare two trees T_i and T_j to see if they should be merged into a new tree. Define $\mathscr{D}_{ij}$ as their merged data, and let $M_{ij} = 1$ if they should be merged, and $M_{ij} = 0$ otherwise.

The probability of a merge is given by

$$r_{ij} = \frac{p(\mathscr{D}_{ij}|M_{ij} = 1)p(M_{ij} = 1)}{p(\mathscr{D}_{ij}|T_{ij})} \tag{33}$$

$$p(\mathscr{D}_{ij}|T_{ij}) = p(\mathscr{D}_{ij}|M_{ij} = 1)p(M_{ij} = 1) + p(\mathscr{D}_{ij}|M_{ij} = 0)p(M_{ij} = 0) \tag{34}$$

If $M_{ij} = 1$, the data in $\mathscr{D}_{ij}$ is assumed to come from the same model, and hence

$$p(\mathscr{D}_{ij}|M_{ij} = 1) = \int \left[\prod_{x_n \in \mathscr{D}_{ij}} p(x_n|\theta) \right] p(\theta|\lambda) d\theta \tag{35}$$

If $M_{ij} = 0$, the data in $\mathscr{D}_{ij}$ is assumed to have been generated by each tree independently, so

$$p(\mathscr{D}_{ij}|M_{ij} = 0) = p(\mathscr{D}_i|T_i) p(\mathscr{D}_j|T_j) \tag{36}$$

When finished, we can cut the tree at points where $r_{ij} < 0.5$.

5.4.2 The connection with Dirichlet process mixture models

In this section, one can show that there is a connection between BHC and DPMMs. This will in turn give us an algorithm to compute the prior probabilities $p(M_{ij} = 1)$.

We are now in a position to compute $\pi_k = p(M_k = 1)$, for each node k with children i and j. This is equal to the probability of cluster $\mathscr{D}_k$ coming from the DPMM, relative to all other partitions of $\mathscr{D}_k$ consistent with the current tree. This can be computed as follows: initialize $d_i = \alpha$ and $\pi_i = 1$ for each leaf i; then as we build the tree, for each internal node k, compute $d_k = \alpha\Gamma(n_k) + d_i d_j$, and $\pi_k = \frac{\alpha\Gamma(n_k)}{d_k}$, where i and j are k's left and right children.

6 Clustering datapoints and features

So far, we have been concentrating on clustering datapoints. But each datapoint is often described by multiple features, and we might be interested in clustering them as well. Below we describe some methods for doing this.

6.1 Biclustering

Clustering the rows and columns is known as **biclustering** or **coclustering**.

Here we present a simple probabilistic generative model. The idea is to associate each row and each column with a latent indicator, $r_i \in \{1, ..., K^r\}$, $c_j \in \{1, ..., K^c\}$. We then assume the data are iid across samples and across features within each block:

$$p(x|r, c, \theta) = \prod_i \prod_j p(x_{ij}|r_i, c_j, \theta) = p(x_{ij}|\theta_{r_i, c_j} \tag{37}$$

where $\theta_{a,b}$ are the parameters for row cluster a and column cluster b. Rather than using a finite number of clusters for the rows and columns, we can use a Dirichlet process. We can fit this model using e.g., (collapsed) Gibbs sampling.

6.2 Multi-view clustering

The problem with biclustering is that each object (row) can only belong to one cluster. Intuitively, an object can have multiple roles, and can be assigned to different clusters depending on which subset of features you use.

We now present a model that can capture this phenomenon. This model is called **crosscat** (for cross-categorization), or (non-parametric) **multi-clust**. The idea is that we partition the columns (features) into V groups or **views**, so $c_j \in \{1, ..., V\}$, where $j \in \{1, ..., D\}$ indexes features. We will use a Dirichlet process prior for $p(c)$, which allows V to grow automatically. Then for each partition of the columns (i.e., each view), call it v, we partition the rows, again using a DP. Let $r_{iv} \in \{1, ..., K(v)\}$ be the cluster to which the i'th row belongs in view v. Finally, having partitioned the rows and columns, we generate the data: we assume all the rows and columns within a block are iid. We can define the model more precisely as follows:

$$p(c, r, \mathscr{D}) = p(c) p(r|c) p(\mathscr{D}|r, c) \tag{38}$$

$$p(c) = DP(c|\alpha) \tag{39}$$

$$p(r|c) = \prod_{v=1}^{V(c)} DP(r_v|\beta) \tag{40}$$

$$p(\mathscr{D}|r,c,\theta) = \prod_{v=1}^{V(c)} \prod_{j:c_j=v} \left[\prod_{k=1}^{K(r_v)} \int \prod_{i:r_{iv}=k} p(x_{ij}|\theta_{jk}) p(\theta_{jk}) d\theta_{jk} \right] \tag{41}$$

If the data is binary, and we use a $Beta(\gamma,\gamma)$ prior for θ_{jk}, the likelihood reduces to

$$p(\mathscr{D}|r,c,\gamma) = \prod_{v=1}^{V(c)} \prod_{j:c_j=v} \prod_{k=1}^{K(r_v)} \frac{Beta(n_{j,k,v}+\gamma, \bar{n}_{j,k,v}+\gamma)}{Beta(\gamma,\gamma)} \tag{42}$$

where $n_{j,k,v} = \sum_{i:r_{i,v}=k} \mathbb{I}(x_{ij}=1)$ counts the number of features which are on in the j'th column for view v and for row cluster k. Similarly, $\bar{n}_{j,k,v}$ counts how many features are off. The model is easily extended to other kinds of data, by replacing the beta-Bernoulli with, say, the Gaussian-Gamma-Gaussian model.

References

[1] K. P. Murphy, *Machine Learning : A Probabilistic Perspective*, 1st. Cambridge, Mass. [u.a.]: MIT Press, 2013.