

# CSE 6740 Lecture 17

## How Can I Reduce/Relate the Data Points? (Association and Clustering)

Alexander Gray

agray@cc.gatech.edu

Georgia Institute of Technology

# Today

- ① Associations
- ② Clustering

Central tasks for “data mining”.

# Associations

“What things in the data commonly appear together?”

# Association Rules

Goal: Given discrete features  $X_1, \dots, X_D$ , find frequently-occurring combinations of values.

This is motivated by market basket data where the features correspond to different types of products bought by a customer and their values are specific products, for example  $X_1$  is “toothpaste”, with values “none”, “regular”, or “whitening”, or “sensitive-teeth”. In *market basket analysis* we want to discover that buying whitening toothpaste and hair dye together is common, for example.

# Association Rules

To reduce the computational complexity of this problem, we can consider the problem of finding a subsets  $s_1, \dots, s_D$  of the variable values for each feature such that

$$\mathbb{P} \left[ \bigcap_{d=1}^D (X_d \in s_d) \right] \quad (1)$$

is relatively large.

The intersection of subsets  $\bigcap_{d=1}^D (X_d \in s_d)$  is a conjunctive rule. If a subset  $s_d = S_d$ , where  $S_d$  is the entire range of possible values, we can leave  $X_d$  out of the rule.

# Association Rules

For large datasets, this is often simplified to the case where we only allow a subset  $s_d$  to be  $S_d$  or exactly one of the values  $v_d$  of  $X_d$ .

We can also change to binary variables  $Z_1, \dots, Z_M$  where there is one  $Z$  variable for the indicator function on each subset  $I(X_d \in s_d)$ , having value  $z \in \{0, 1\}$ .

# Association Rules

Then the problem is to find *all* subsets (called *item sets*)  $\zeta$  of these  $1, \dots, M$  variables such that the *support*

$$S(\zeta) \equiv \mathbb{P} \left[ \bigcap_{m \in \zeta} (Z_m = 1) \right] = \mathbb{P} \left[ \prod_{m \in \zeta} (Z_m = 1) \right] \quad (2)$$

is greater than some threshold  $t$ . The support can be estimated using

$$\hat{S}(\zeta) = \hat{\mathbb{P}} \left[ \prod_{m \in \zeta} (Z_m = 1) \right] = \frac{1}{N} \sum_{i=1}^N \prod_{m \in \zeta} z_{im}. \quad (3)$$

# Association Rules

The total number of possible subsets is  $2^M$ , and the list of subsets having this property will in general be huge.

To limit the size of the result, one common approach is to impose that for all  $\xi \subset \zeta$ ,  $S(\xi) \geq S(\zeta)$ .

This leads leads to the following efficient algorithm, called the *Apriori algorithm*: Compute the support for all item sets of size one, then discard those below the threshold. Then compute the support for all itemsets of size two that can be formed from pairs of the surviving single items, and so on. This requires requires only one pass over the data for each item set size.

# Clustering Methods

“Show me the sub-groups in the data.”

Why show sub-groups in the data? Sometimes:

- Computational reasons (e.g. use cluster centers instead of the dataset)
- Statistical reasons (e.g. identify/remove outliers)
- Mainly: Visualization/understanding reasons

# Procedural Methods

When we can speak of a true underlying function (as we do in most density estimation, classification, and regression methods), we can discuss error, error bounds, generalization (minimizing error on future data), what happens to the error as we get more data, etc. In other words we can leverage all the powerful tools of statistics we have discussed.

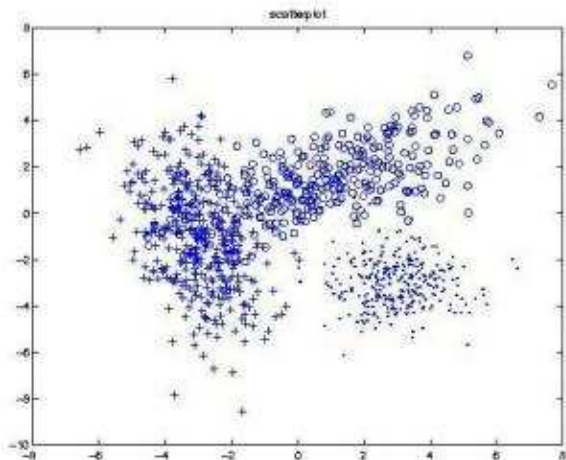
# Procedural Methods

I will call a method which has not been formally related to some function of the underlying density a *procedural* method. This turns out to be common in clustering and density estimation methods.

Though this makes it hard/impossible to say much about these methods analytically, they are nonetheless often still useful in practice.

# Mixture of Gaussians

Treat clustering as a density estimation problem, where each Gaussian is a cluster.



# Mixture of Gaussians

Again:

**Task:** density estimation

**Model class:** set of all possible *mixtures of Gaussians* with  $K$  components

**Loss:** Likelihood

**Optimizer:** EM algorithm

**Generalization mechanism:** Cross-validation

**Evaluation algorithm:**  $N$ -body

# Sum-of-Squares Minimization

Here's a simpler method, which cannot be described as relating to some function of the underlying distribution of the data.

We'll seek a partitioning of the points into  $K$  disjoint subsets  $C_k$  each containing  $N_k$  points, such that the following sum-of-squares objective function is minimized:

$$\sum_{k=1}^K \sum_{i \in C_k} \|x_i - \mu_k\|^2 \quad (4)$$

where  $\mu_k = \frac{1}{N_k} \sum_{i \in C_k} x_i$  is the mean of the points in set  $C_k$ .  
 $C(x_i) = C_k$  will denote that the class of  $x_i$  is  $C_k$ .

# $K$ -means

The  $K$ -means method is as follows:

First initialize the means  $\mu_k$  somehow, for example by choosing  $K$  different points randomly. Then:

- 1 Assign each point according to  $C(x_i) = \arg \min_k \|x_i - \mu_k\|$ .
- 2 Recompute each  $\mu_k$  according to the new assignments.

Stop when no assignments change.

This process can be shown to never increase the sum-of-squares. However, it does not necessarily obtain the global optimum. In practice, this is done, say, 10 times and the result with the lowest sum-of-squares is used.

# $K$ -means

It turns out that  $K$ -means can be interpreted as a “hard” version of the EM algorithm for a mixture of spherical Gaussians. Though close, it is not quite a special case, which would make it parametric. It also cannot be said to be nonparametric (*i.e.* converge to some function of the distribution for a large class of distributions).

Task: clustering

Model class: n/a

Loss: sum-of-squares

Optimizer:  $K$ -means algorithm

Generalization mechanism: visual; cross-validation in principle

Evaluation algorithm:  $N$ -body

# Max-radius Minimization

Here's another clustering objective function, which comes from the CS theory literature. Let's now minimize the maximum radius of clusters

$$\max_k \max_{x_i \in C_k} \|x_i - \mu_k\|. \quad (5)$$

Here each cluster center is always equal to one of the points.

# Gonzalez Algorithm

There turns out to be an effective algorithm for finding good solutions to this, the *Gonzalez algorithm*:

We'll start with no clusters and add cluster centers one at a time:

- 1 Pick an arbitrary point as the center of the first cluster,  $\mu_1$ .
- 2 Find the point  $x_i$  which maximizes  $\min_k \|x_i - \mu_k\|$ , and make it the next cluster center.
- 3 Repeat until we have  $K$  cluster centers.

Each point is assigned the label of its nearest cluster center.

# Gonzalez Algorithm

As with the sum-of-squares objective function, minimizing the max-radius objective function is  $NP$ -hard, since there is an exponential number of possible labelings of the points to consider.

The Gonzalez algorithm can be shown (very simply) to have running time  $O(NK)$  and to compute a partition whose maximum radius is at most twice the optimum. In other words it is a  $2$ -approx algorithm having linear runtime. (With slightly more complication it can be made  $O(N \log K)$ .  $2$  can be shown to be the best approximation factor possible unless  $P = NP$ .)

Even more interestingly, it produces good clusters fairly consistently.

# Gonzalez Algorithm

Note that this method is not generally considered in the context of data analysis. We're including it here because it might as well be.

**Task:** clustering

**Model class:** n/a

**Loss:** max-radius objective function

**Optimizer:** Gonzalez algorithm

**Generalization mechanism:** visual; cross-validation in principle

# Hierarchical Clustering

What about clusters which are not convex blobs?

Here's another idea which can sometimes find arbitrarily-shaped clusters, and relaxes the need to specify the number of clusters  $K$ . It is procedural.

# Hierarchical Clustering

Consider a sequence of partitions of the points. The first is a partition into  $N$  clusters, one for each point. The next partition joins two of the clusters, resulting in  $N - 1$  clusters, and so on, until the  $N^{\text{th}}$  partition, which contains one cluster.

If the sequence has the property that whenever two points are in the same cluster at level  $m$ , they remain together at all subsequent levels, the sequence is called a *hierarchical clustering*.

We are describing the *agglomerative* approach to hierarchical clustering, where we merge clusters in bottom-up fashion. We could also go the other way, the *divisive* approach, which performs top-down splitting.

# Hierarchical Clustering

We merge the “nearest” pair of clusters at each step. Some options for defining the distance between two clusters are:

$$d_{\min}(C_k, C_{k'}) = \min_{x \in C_k, x' \in C_{k'}} \|x - x'\| \quad (6)$$

$$d_{\max}(C_k, C_{k'}) = \max_{x \in C_k, x' \in C_{k'}} \|x - x'\| \quad (7)$$

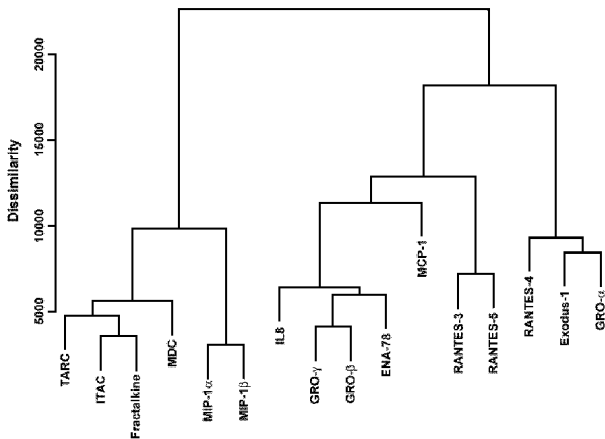
$$d_{\text{avg}}(C_k, C_{k'}) = \frac{1}{N_k N_{k'}} \sum_{x \in C_k} \sum_{x' \in C_{k'}} \|x - x'\| \quad (8)$$

$$d_{\text{cen}}(C_k, C_{k'}) = \|\mu_k - \mu_{k'}\| \quad (9)$$

Using  $d_{\min}$  results in a minimum spanning tree and tends to produce long chains. Using  $d_{\max}$  tends to produce compact clusters. The other two have behavior somewhere between these two extremes.

# Hierarchical Clustering

An advantage of hierarchical clustering for visualization is that you can draw a tree (called a *dendrogram*) representing the clusters and the distances between them.



# Hierarchical Clustering

It is also a deterministic procedure, with no parameters.

**Task:** clustering

**Model class:** n/a

**Loss:** n/a (procedural)

**Optimizer:** greedy bottom-up (or top-down) algorithm

**Generalization mechanism:** visual

**Evaluation algorithm:**  $N$ -body

# Mean-shift

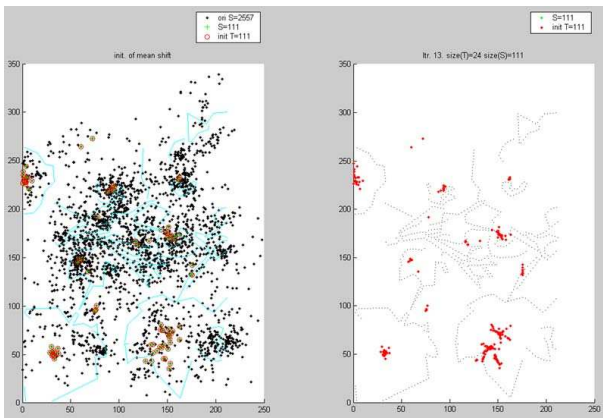
The *mean-shift* algorithm is a way to find local modes (bumps) using nonparametric kernel density estimation.

It can basically find clusters of arbitrary shape.

It is automatic in terms of the number of modes  $K$ , which is found implicitly by the method, and has one parameter  $h$ .

# Mean-shift

The idea is that we move the data points in the direction of the log of the gradient of the density of the data, until they finally converge to each other at the peaks of the bumps.



# Mean-shift

Suppose  $x_i^m$  is the position of the  $i^{\text{th}}$  data point during iteration  $m$  of the procedure. A kernel density estimate  $\hat{f}^m$  is constructed from the points  $\{x_i^m\}$ . We obtain the next round of points according to:

$$x_i^{m+1} = x_i^m + a \nabla \log \hat{f}^m(x_i^m) \quad (10)$$

$$= x_i^m + \frac{a}{\hat{f}^m(x_i^m)} \nabla \hat{f}^m(x_i^m) \quad (11)$$

by simple calculus.

# Mean-shift

$\hat{f}^m(x_i^m)$  is found by kernel density estimation.  $\nabla \hat{f}^m(x_i^m)$  is found by kernel density estimation using the gradient of the original kernel.

It can be shown that the procedure converges (stops), under certain assumptions about how  $h$  and  $a$  are chosen.

# Mean-shift

Why the *log* of the gradient? The denominator  $\hat{f}^m(x_i^m)$  doesn't change the direction of the shift  $x_i^{m+1} - x_i^m$ , and makes the method converge in fairly few iterations.

Points in the low-density tails can move a considerable distance toward regions of high density. It can be shown that points drawn from a spherical Gaussian, if  $a$  is set to the variance, will jump to the centroid in one step.

# Mean-shift

Suppose that we're using the Epanechnikov kernel and that we use the value

$$a = \frac{h^2}{D + 2}. \quad (12)$$

Then it turns out that simple algebra reduces the update rule to the form:

$$x_i^{m+1} = \text{mean position of points } x_j^m \text{ within distance } h \text{ of } x_i^m. \quad (13)$$

This form is called the *mean-shift algorithm*.

# Mean-shift

Though tantalizingly promising, mean-shift has not been shown to converge to some function of the underlying density.

**Task:** clustering

**Model class:** n/a

**Loss:** n/a (procedural)

**Optimizer:** mean-shift algorithm

**Generalization mechanism:** visual

**Evaluation algorithm:**  $N$ -body

# Temporal Clustering

How can we find clusters in a time series?

The *hidden Markov model* (HMM) is effectively a mixture of Gaussians (or discrete distributions) for a time series. As in the mixture of Gaussians case, we will treat clustering as density estimation.

# Hidden Markov Model

This parametric model assumes that there are  $K$  states. Let  $C_t = C(x_t)$  be the state at time  $t$ . When in state  $C_k$  at time  $t$ ,

$$x_t \sim f_k(x) \equiv f(x|C_k) = N(\mu_k, \Sigma_k) \quad (14)$$

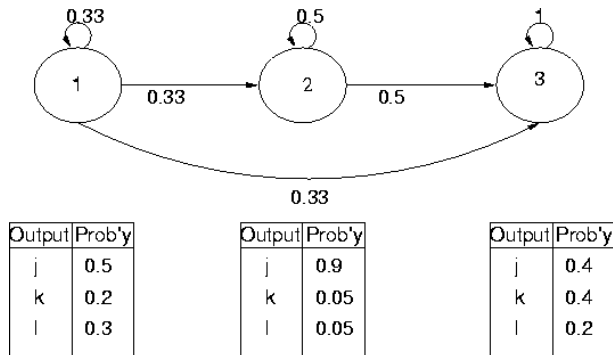
There is a chance  $a_{kk'}$  of jumping from state  $k$  at time  $t$  to  $k'$  at time  $t + 1$  given by

$$\mathbb{P}(C_{t+1} = C_{k'} | C_t = C_k) \quad (15)$$

given by a discrete probability distribution. Since there is a vector of  $K$  probabilities for each state  $k$ , we have a  $K \times K$  transition matrix  $A$ .

There is also a discrete distribution given by  $\pi_k$  for the prior state probabilities.

# Hidden Markov Model



# Hidden Markov Model

Because the next state depends only on the last state, we say this model has the *Markov property*. Because the underlying state labels must be inferred, we say the state is *hidden*.

**Task:** density estimation

**Model class:** set of all possible hidden Markov models

**Loss:** likelihood

**Optimizer:** EM algorithm (called *Baum-Welch* for this model class)

**Generalization mechanism:** visual; cross-validation in principle

**Evaluation algorithm:** belief propagation

# Main Things You Should Know

- What association rule mining is
- What the various clustering methods are
- Procedural vs. non-procedural