Lecture 7:

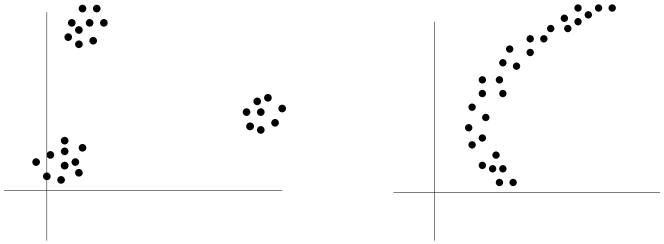
Clustering and Tree Models

Sam Roweis

October 26, 2004

Unsupervised Learning

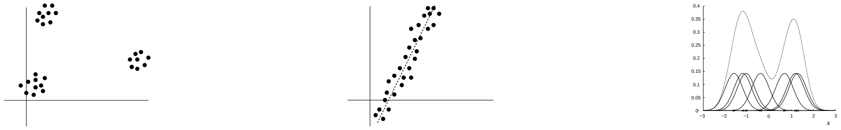
- So far we have only discussed *supervised learning* in which there are both inputs and desired outputs.
  For *regression*, the output(s) were continuous values.
  For *classification*, the output was a discrete (categorical) label.
- Another very important problem in machine learning is *unsupervised learning*, in which there are no outputs, only inputs.



- What should we do here?

Three Unsupervised Models

- The three canonical problems in unsupervised learning are *clustering*, *dimensionality reduction*, and *density modeling*:
  - Clustering: grouping similar training cases together and identifying a "prototype" example to represent each group.
  - Dimensionality reduction: learning to represent each training case using a small number of continuous variables from which the original data can be almost exactly reconstructed.
  - Density modeling: learning a density function from a few samples. This is like quantitative novelty detection: we want to produce a large signal when data similar to training data appears and a small signal when different data appears.

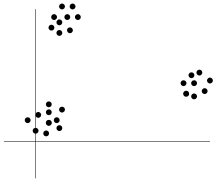


Missing Outputs

- You can think of unsupervised learning as supervised learning in which all the outputs are *missing*:
  - Clustering == classification with missing labels.
  - Dimensionality reduction == regression with missing targets.
- Density estimation is actually very general and encompasses the two problems above and a whole lot more.
- Let's start off by talking about clustering

Clustering

- Clustering: grouping similar training cases together and identifying a "prototype" example to represent each group.



- Several approaches: agglomerative, divisive, fixed number of clusters, hierarchical, ...
- All require a way to measure distance between two data points, e.g.
  Euclidean distance $\|\mathbf{x}-\mathbf{y}\|^2$,
  Mahanalobis distance $(\mathbf{x}-\mathbf{y})^\top\Sigma^{-1}(\mathbf{x}-\mathbf{y})$,
  ...

Cost Function for K-Means

- Q: What cost function is K-means minimizing?
  A: Average squared distance from each datapoint to the nearest cluster centre:

$$E(\{\mu_k\}) = \frac{1}{N}\sum_n \min_k \left[(\mathbf{x}^n-\mu_k)^\top\Sigma^{-1}(\mathbf{x}^n-\mu_k)\right]$$

- The K-means algorithm does coordinate descent in a function $F(\{\mu_k\},\{c_n\})$ which is an upper bound on this error:

$$F(\{\mu_k\},\{c_n\}) = \frac{1}{N}\sum_n \left[(\mathbf{x}^n-\mu_{c_n})^\top\Sigma^{-1}(\mathbf{x}^n-\mu_{c_n})\right]$$

  This upper bound is valid for *any* setting of the c_n.
  After the assignment step for c_n, $F(\mu,c)=E(\mu)$.
  The assignment step lowers this bound as much as possible.

Algorithm: K-means

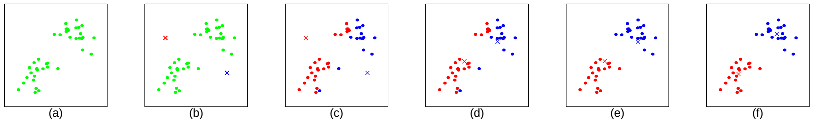
- Select a number of clusters K and covariance Σ.
  Start with initial cluster centres $\mu_1^0,\mu_2^0,\ldots,\mu_K^0$.
- Alternate between two steps.
  Assign each datapoint to the cluster whose centre is closest:

$$c_n^{t+1} = \mathrm{argmin}_k(\mathbf{x}^n-\mu_k^t)^\top\Sigma^{-1}(\mathbf{x}^n-\mu_k^t)$$

  Update cluster centres to the mean of all points assigned to them:

$$\mu_k^{t+1} = \frac{\sum_n[c_k^{t+1}=n]\mathbf{x}^n}{\sum_n[c_k^{t+1}=n]}$$

- When clusters become empty, use a heuristic to reposition their means. Break ties in distance using cluster of smallest size.



Vector Quantization

- K-means clustering is also called *vector quantization* in the engineering and signal processing literature, because the problem is like quantization but for multivariate objects. The cluster centres are called *codebook vectors*.
- More correctly, K-means (or VQ) is an *optimization problem*, and the algorithm above, which is (just) one potential solution to it is called the *Lloyd-Max algorithm*.
- But everybody just calls the algorithm K-means.

Implementation Tricks

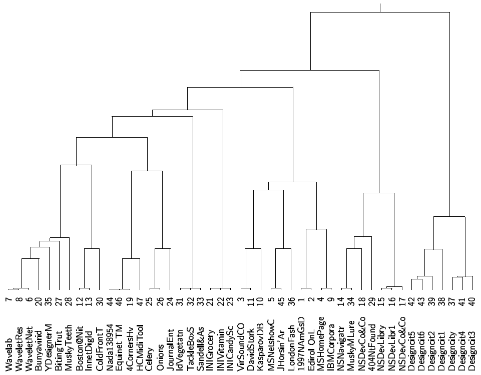
- K-means (and other clustering methods) require tricks to work well.
- Initialization: set μ_k^0 to be K randomly chosen points, or else to the first K points from *furthest-first* clusterting (see later).
- Picking number of clusters: use cross validation on the error function evaluated on a validation set.
- Unused clusters: set to points with biggest errors.
- Ties in distance: add points to smaller clusters first.
- Robust errors: use squared error up to some maximum error then constant error beyond that. (Affects both steps.)
- Local minima: use random restarts, split and merge clusters.

More General Objective Functions

- If we change the distance function, the assignment step is still easy, but updating the cluster centres might be hard.
- We can also restrict the centres. ex: In the *K-mediods* problem, we update the new cluster centres to be one of the points assigned to that cluster, but we have to try every possible point. Expensive!
- Some common distances, their names, and their cost functions:
  K-means (average squared distance)
  K-medians (average distance):
  $E(\{\mu_k\}) = \frac{1}{N}\sum_n \min_k \left[\sqrt{(\mathbf{x}^n - \mu_k)^\top \Sigma^{-1} (\mathbf{x}^n - \mu_k)}\right]$
  K-corners (average abs. error):
  $E(\{\mu_k\}) = \frac{1}{N}\sum_n \min_k \left[\sum_i |x_i^n - \mu_{ki}|\right]$
  K-centres (biggest cluster radius):
  $E(\{\mu_k\}) = \max_n \min_k \left[(\mathbf{x}^n - \mu_k)^\top \Sigma^{-1} (\mathbf{x}^n - \mu_k)\right]$
- Special cases solved, e.g. K-corners: $\mu_{ki}^t = \mathrm{median}_{c_n^t = k}[x_i^n]$

Hierarchical Clustering

- Hierarchical clustering algorithms break the dataset into a series of nested clusters, starting with a single cluster at the top containing all the data and ending with N clusters at the bottom, one for each point. The results can be displayed as a *dendrogram*:



Agglomerative Clustering

- Agglomerative algorithms for hierarchical clustering start with each datapoint in its own cluster and then successively merge similar clusters until a single cluster remains.
- Several methods for merging. Most based on computing cluster distances $d_{cc'}$ from pairwise distances $d_{nn'}$ between all pairs of points and then merging the two clusters with smallest $d_{cc'}$:
  Single linkage: $d_{cc'} = \min_{n \in c, n' \in c'} d_{nn'}$
  Complete linkage: $d_{cc'} = \max_{n \in c, n' \in c'} d_{nn'}$
  Average linkage: $d_{cc'} = \mathrm{mean}_{n \in c, n' \in c'} d_{nn'}$

Divisive Clustering

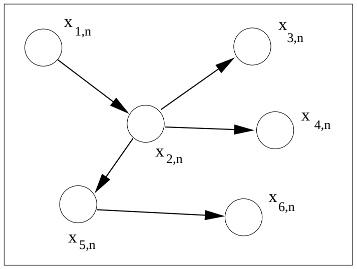
- Divisive algorithms for hierarchical clustering start with all the data in a single cluster and successively split clusters.
- Here's my favourite one: furthest-first traversal.

```
Pick any point, mark it, and set mu(1) equal to it.
for i=2:N
  find the unmarked point furthest from {mu(1)...mu(i-1)}
  [using dist(point,{set})=min(p' in {set}) dist(point,p')]
  mark this point and set mu(i) equal to it
end
```

- For good measure, run K-means until convergence afterward.

Tree Models as Graphs

- If we identify each variable with a node in a graph, we can describe this model by drawing a directed arrow from each node to its children. NB: each node (except root) has exactly one parent but may have more than one child.



- This is a special case of a general way of describing statistical functions using *probabilistic graphical models*.

Tree Models

- A tree model is a unsupervised learning model in which each variable x_i has exactly one other variable as its "parent" x_{π_i}, except the "root" x_{root} which has no parents.
- The probability of a variable taking on a certain value depends only on the value of its parent:

$$p(\mathbf{x}) = p(x_{\text{root}}) \prod_{i \neq \text{root}} p(x_i | x_{\pi_i})$$

- Trees are the next step up from assuming independence. Instead of considering variables in isolation, consider them in pairs.
- WARNING: do not confuse these trees (probability model is a tree) with decision trees (algorithm proceeds in a tree structured fashion). Tree model classifiers assume a tree model of features given the class, but are *not* the same as classification/decision trees!

Maximum Likelihood Node Parameters

- Trees are just a special case of fully observed density models.
- For discrete data x_i with values v_i, each node stores a conditional probability table (CPT) over its values given its parent's value. The ML parameter estimates are just the empirical histograms of each node's values given its parent:

$$p^*(x_i = v_i | x_{\pi_i} = v_j) = \frac{N(x_i = v_i, x_{\pi_i} = v_j)}{\sum_{z_i, z_j} N(x_i = z_i, x_{\pi_i} = z_j)}$$

except for the root which uses marginal counts $N(v_{root})/N$.

- For continuous data, the most common model is a two-dimensional Gaussian at each node, jointly modeling the node and its parent.
- The ML parameters are just to set the mean of $p_i(x_i, x_{\pi_i})$ to be the sample mean of $[x_i; x_{\pi_i}]$ and the covariance matrix to the sample covariance.

Overall Likelihood function

- Overall likelihood is sum of parent-conditional terms, one per node: $\mathbf{V}_i \equiv$ set of joint configurations of x_i and its parent x_{π_i} ($\mathbf{V}_{\text{root}} \equiv$ set of values of root note)

$$\ell(\theta;\mathcal{D}) = \sum_n \log p(\mathbf{x}^n) = \sum_n \left[\log p_r(x_r^n) + \sum_{i\neq r} \log p({x_i}^n|{x_{\pi_i}}^n)\right]$$
$$= \sum_n \sum_i \sum_{\mathbf{y}\in\mathbf{V}_i} [x_i, x_{\pi_i} = \mathbf{y}] \log p_i(x_i|x_{\pi_i})$$
$$= \sum_i \sum_{\mathbf{y}\in\mathbf{V}_i} N_i(\mathbf{y}) \log p_i(\mathbf{y})$$

with $p_i(\mathbf{y}_i) = p(x_i|x_{\pi_i})$ and counts $N_i(\mathbf{y}) = \sum_n [\mathbf{y}_i^n = \mathbf{y}]$.

- Trees are in the exponential family with $\mathbf{y}_i$ as sufficient statistics.

Optimal Tree Structure ≡ MWST

- Let us rewrite the overall likelihood function:

$$\ell(\theta;\mathcal{D}) = \sum_{\mathbf{x}\in\mathbf{V}_{\text{all}}} N(\mathbf{x}) \log p(\mathbf{x})$$
$$= \sum_{\mathbf{x}} N(\mathbf{x}) \left(\log p(x_r) + \sum_{i\neq r} \log p(x_i|x_{\pi_i})\right)$$

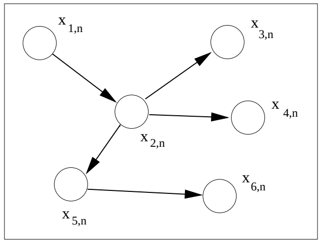
- ML parameters, are equal to the observed frequency counts $q(\cdot)$:

$$\frac{\ell^*}{M} = \sum_{\mathbf{x}\in\mathbf{V}_{\text{all}}} q(\mathbf{x}) \left(\log q(x_r) + \sum_{i\neq r} \log q(x_i|x_{\pi_i})\right)$$
$$= \sum_{\mathbf{x}} q(\mathbf{x}) \left(\log q(x_r) + \sum_{i\neq r} \log \frac{q(x_i, x_{\pi_i})}{q(x_{\pi_i})}\right)$$
$$= \sum_{\mathbf{x}} q(\mathbf{x}) \sum_{i\neq r} \log \frac{q(x_i, x_{\pi_i})}{q(x_i)q(x_{\pi_i})} + \sum_{\mathbf{x}} q(\mathbf{x}) \sum_i \log q(x_i)$$

- NB: second term does not depend on structure.

Structure Learning

- What about the tree structure (links)? How do we know which nodes to make parents of which?



- Bold idea: can we also *learn* the optimal structure? In principle, we could search all combinatorial structures, for each compute the ML parameters, and take the best one.
- But is there a better way? Yes. It turns out that structure learning in tree models can be converted to a good old computer science problem: maximum weight spanning tree.

Edge Weights

- Each term in sum $i \neq r$ corresponds to an edge from i to its parent.

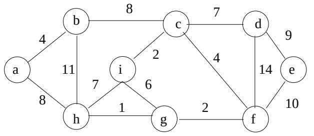
$$\frac{\ell^*}{M} = \sum_{\mathbf{x}} q(\mathbf{x}) \sum_{i\neq r} \log \frac{q(x_i, x_{\pi_i})}{q(x_i)q(x_{\pi_i})} + C$$
$$= \sum_{i\neq r} \sum_{x_i, x_{\pi_i}} q(x_i, x_{\pi_i}) \log \frac{q(x_i, x_{\pi_i})}{q(x_i)q(x_{\pi_i})} + C$$
$$= \sum_{i\neq r} \sum_{y_i} q(y_i) \log \frac{q(y_i)}{q(x_i)q(x_{\pi_i})} + C$$
$$= \sum_{i\neq r} W(i;\pi_i) + C$$

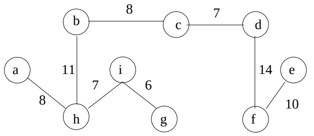
- So the overall likelihood is the sum of weights on edges that we use. We need the maximum weight spanning tree to maximize likelihood.
- The edge weights W are defined by *mutual information*:

$$W(i;j) = \sum_{x_i, x_j} q(x_i, x_j) \log \frac{q(x_i, x_j)}{q(x_i)q(x_j)}$$

Kruskal's algorithm (Greedy Search)

- To find the maximum weight spanning tree A on a graph with nodes U and weighted edges E:
  1. $A \leftarrow \mathrm{empty}$
  2. Sort edges E by nonincreasing weight: $e_1, e_2, \ldots, e_K$.
  3. for $k = 1$ to K $\{A \mathrel{+}= e_k$ unless doing so creates a cycle$\}$





Undirected vs. Directed Trees

- *Any* directed tree consistent with the undirected tree found by the algorithm above will assign the same likelihood to any dataset.
- Amazingly, as far as likelihood goes, the root is arbitrary.
  We can just pick one node and orient the edges away from it.
  Or we can work with undirected models.
- For continuous nodes (e.g. Gaussian), the situation is similar, except that computing the mutual information requires an integral.
- Mutual information is the *Kullback-Leibler* divergence (cross-entropy) between a distribution and the product of its marginals. Measures how far from independent the joint distribution is.

Maximum Likelihood Trees

We can now completely solve the tree learning problem:

1. Compute the marginal counts $q(x_i)$ for each node and pairwise counts $q(x_i, x_j)$ for all pairs of nodes.
2. Set the weights to the mutual informations:
$$W(i;j) = \sum_{x_i, x_j} q(x_i, x_j) \log \frac{q(x_i, x_j)}{q(x_i) q(x_j)}$$
3. Find the maximum weight spanning tree A=MWST(W).
4. Using the undirected tree A chosen by MWST, pick a root arbitrarily and orient the edges away from the root.
   Set the conditional functions to the observed frequencies:
$$p(x_i | x_{\pi_i}) = \frac{q(x_i, x_{\pi_i})}{\sum_{x_i} q(x_i, x_{\pi_i})} = \frac{q(x_i, x_{\pi_i})}{q(x_{\pi_i})}$$