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: partitional (find a fixed number of clusters), hierarchical (agglomerative, divisive),

- All require a way to measure distance between two data points, e.g. Euclidean distance $\|x - y\|^2$, Mahanalobis distance $(x - y)^\top V^{-1}(x - y)$,

Algorithm: K-means

- Select a number of clusters $K$ (and possibly a covariance $V$).
- 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:
    $$c_{n}^{t+1} = \arg\min_k (x^n - \mu_k^t)^\top V^{-1}(x^n - \mu_k^t)$$
  - Update cluster centres to the mean of all points assigned to them:
    $$\mu_k^{t+1} = \frac{\sum_n [c_{n}^{t+1} = n] V x^n}{\sum_n [c_{n}^{t+1} = n]}$$
- If a cluster becomes empty, use a heuristic to reposition its mean.
  - Break ties in distance using cluster of smallest size.

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:
  $$E(\{\mu_k\}) = \frac{1}{N} \sum_n \min_k (x^n - \mu_k)^\top V^{-1}(x^n - \mu_k)$$

- The K-means algorithm does coordinate descent in a function $F(\{\mu_k\}, \{c_n\})$ which is an upper bound on this error:
  $$F(\{\mu_k\}, \{c_n\}) = \frac{1}{N} \sum_n (x^n - \mu_{c_n})^\top V^{-1}(x^n - \mu_{c_n})$$

  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 with respect to $\{c_n\}$ keeping $\mu$ fixed, the update step minimizes it with respect to $\{\mu_k\}$, keeping $c$ fixed.

Vector and Non-Vector Quantization

- K-means clustering is also called vector quantization in the engineering/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.
- However, sometimes, we want to apply this algorithm to complex data that cannot be expressed easily as a vector (e.g., gene sequence).
- As long as we have a distance measure between data items, we can still perform the assignment step, but for the update step, we cannot “average” so we restrict the centres to lie on one of the original data items (could be expensive to find the best one).
- This is the K-mediods/K-medians problem.
Implementation Tricks

- K-means (and other clustering methods) require tricks to work well.
- Initialization: set $\mu_k^0$ to be $K$ randomly chosen points, or else to the first $K$ points from furthest-first clustering (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.
- Some common distances, their names, and their cost functions:
  - K-means (average squared distance)
    $$E(\{\mu_k\}) = \frac{1}{N} \sum_n \min_k [(x^n - \mu_k)^\top V^{-1} (x^n - \mu_k)]$$
  - K-medians (average distance):
    $$E(\{\mu_k\}) = \frac{1}{N} \sum_n \min_k \left[\sqrt{(x^n - \mu_k)^\top V^{-1} (x^n - \mu_k)}\right]$$
  - K-corners (average abs. error):
    $$E(\{\mu_k\}) = \frac{1}{N} \sum_n \min_k \left[\sum_i |x_{ni} - \mu_{ki}|\right]$$
  - K-centres (biggest cluster radius):
    $$E(\{\mu_k\}) = \max_n \min_k \left[(x^n - \mu_k)^\top V^{-1} (x^n - \mu_k)\right]$$
- Special cases solved, e.g. K-corners: $\mu^t_{ki} = \text{median}_{r_n=k} |x^n_i|$
**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 \( \{\mu(1)\ldots\mu(i-1)\} \)
  - using \( \text{dist(point,\{set\})=\min(p' \in \{set\}) \text{dist}(\text{point},p') } \)
- mark this point and set \( \mu(i) \) equal to it

end

**Tree Models**

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

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

- 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 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. (see CSC412/2506)

**Maximum Likelihood Node Parameters**

- Trees are just a special case of fully observed density models.
- For discrete data \( x_i \) with values \( v_j \), 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:

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

except for the root which uses marginal counts \( N(v_{\text{root}})/N \).
- 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: $V_i \equiv$ set of joint configurations of $x_i$ and its parent $x_{\pi_i}$ ($V_{\text{root}} \equiv$ set of values of root node)

\[
\ell(\theta; D) = \sum_n \log p(x^n) = \sum_n \left[ \log p_t(x^n) + \sum_{i \neq r} \log p(x_i^n|x_{\pi_i}^n) \right]
\]

\[
= \sum_n \sum_i \sum_{y \in V_i} [x_i, x_{\pi_i} = y] \log p_t(x_i|x_{\pi_i})
\]

\[
= \sum_i \sum_{y \in V_i} N_i(y) \log p_t(y)
\]

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

- Trees are in the exponential family with $y_i$ as sufficient statistics.

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.

Optimal Tree Structure $\equiv$ MWST

- Let us rewrite the overall likelihood function:

\[
\ell(\theta; D) = \sum_{x \in V_{\text{all}}} N(x) \log p(x)
\]

\[
= \sum_x N(x) \left( \log p(x) + \sum_{i \neq r} \log p(x_i|x_{\pi_i}) \right)
\]

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

\[
\frac{\ell^*}{M} = \sum_{x \in V_{\text{all}}} q(x) \left( \log q(x) + \sum_{i \neq r} \log \frac{q(x_i|x_{\pi_i})}{q(x_i)} \right)
\]

\[
= \sum_x q(x) \left( \log q(x) + \sum_{i \neq r} \frac{q(x_i|x_{\pi_i})}{q(x_i)} \right) + \sum_x q(x) \sum_i \log q(x_i)
\]

- NB: second term does not depend on structure.

Edge Weights

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

\[
\frac{\ell^*}{M} = \sum_x q(x) \sum_{i \neq r} \log \frac{q(x_i|x_{\pi_i})}{q(x_i)q(x_{\pi_i})} + C
\]

\[
= \sum_{i \neq r} \sum_{x_i, x_{\pi_i}} q(x_i, x_{\pi_i}) \log \frac{q(x_i|x_{\pi_i})}{q(x_i)q(x_{\pi_i})} + C
\]

\[
= \sum_{i \neq r} \sum_{y_i} q(y_i) \log \frac{q(y_i)}{q(x_i)q(x_{\pi_i})} + C
\]

\[
= \sum_{i \neq r} W(i; \pi_i) + C
\]

- 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. $A \leftarrow$ empty
2. Sort edges $E$ by nonincreasing weight: $e_1, e_2, \ldots, e_K$.
3. for $k = 1$ to $K$ \{ $A += e_k$ 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 $A=\text{MWST}(W)$.
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:
   \[ p(x_i | x_{\pi_i}) = \frac{q(x_i, x_{\pi_i})}{\sum_{x_i} q(x_i, x_{\pi_i})} = \frac{q(x_i, x_{\pi_i})}{q(x_{\pi_i})} \]