Unsupervised learning

Can we find **regularity in data** without the aid of **labels**?

![Graph](image)

Is this **one cluster**? Or **three**? Or some other number?

---

**The K-means algorithm**

The example on the last slide was obtained using the classical **K-means algorithm**.

Given a set \( \{x_i\} \) of \( m \) points, guess that there are \( K \) clusters. Here \( K = 3 \).

Chose at random \( K \) centre points \( c_j \) for the clusters. Then **iterate as follows**:

1. Divide \( \{x_i\} \) into \( K \) clusters, so each point is associated with the closest centre:
   \[
   x_i \in C_j \iff \forall k \ |x_i - c_j| \leq |x_i - c_k|.
   \]
   Call these clusters \( C_1, \ldots, C_K \).

2. Update the cluster centres to be the average of the associated points:
   \[
   c_j = \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i.
   \]
The modern approach is once again probabilistic.

Data from $K$ clusters can be modelled probabilistically as

$$p(x|\theta) = \sum_{k=1}^{K} \pi_k p(x|\mu_k, \Sigma_k)$$

where $\theta = \{\pi, \mu_1, \Sigma_1, \ldots, \mu_K, \Sigma_K\}$ and typically $p(x|\mu, \Sigma) = \mathcal{N}(\mu, \Sigma)$.

This leads to a log-likelihood for $m$ points of

$$\log p(X|\theta) = \log \prod_{i=1}^{m} p(x_i|\theta)$$

where the final step has given us probabilities that are reasonably tractable.

Why is this?

First, if I know which cluster generated $x_i$ then its probability is just that for the corresponding Gaussian

$$p(x_i|z_i, \theta) = \prod_{k=1}^{K} [p(x_i|\mu_k, \Sigma_k)]^{(k)}$$

and similarly

$$p(z_i|\theta) = \prod_{k=1}^{K} [\pi_k]^{(k)}.$$
Clustering as maximum-likelihood

In other words, if you treat the $z_i$ as observed rather than latent $m$

then you can write

$$p(x_i, z_i | \theta) = \prod_{k=1}^{K} [p(x_i | \mu_k, \Sigma_k) \pi_k]^{z(k)_i}.$$  

$$\log p(X, Z | \theta) = \log \prod_{i=1}^{m} \prod_{k=1}^{K} [p(x_i | \mu_k, \Sigma_k) \pi_k]^{z(k)_i}.$$  

Consequently

$$\log p(X, Z | \theta) = \sum_{i=1}^{m} \sum_{k=1}^{K} z^{(k)}_i (\log p(x_i | \mu_k, \Sigma_k) + \log \pi_k).$$  

What have we achieved so far?
1. We want to maximize the log-likelihood $\log p(X | \theta)$ but this is intractable.
2. We introduce some latent variables $Z$.
3. That gives us a tractable log-likelihood $\log p(X, Z | \theta)$.

But how do we link them together?

The EM algorithm

The Expectation Maximization (EM) algorithm provides a general way of maximizing likelihood for problems like this.

Here we apply it to unsupervised learning, but it can also be applied to learning Hidden Markov Models (HMMs) and many other things

Let $q(Z)$ be any distribution on the latent variables. Write

$$\sum_Z q(Z) \log \frac{p(X, Z | \theta)}{q(Z)} = \sum_Z q(Z) \log \frac{p(Z | X, \theta)p(X | \theta)}{q(Z)}$$  

$$= \sum_Z q(Z) \left( \log \frac{p(Z | X, \theta)}{q(Z)} + \log p(X | \theta) \right)$$  

$$= -D_{KL}[q(Z) || p(Z | X, \theta)] + \sum_Z q(Z) \log p(X | \theta)$$  

$$= -D_{KL}[q(Z) || p(Z | X, \theta)] + \log p(X | \theta).$$  

$D_{KL}$ is the Kullback-Leibler (KL) distance.

The Kullback-Leibler (KL) distance

The Kullback-Leibler (KL) distance measures the distance between two probability distributions. For discrete distributions $p$ and $q$ it is

$$D_{KL}[p || q] = \sum_x p(x) \log \frac{p(x)}{q(x)}.$$  

It has the important properties that:
1. It is non-negative $D_{KL}(p || q) \geq 0$.
2. It is 0 precisely when the distributions are equal $D_{KL}[p || q] = 0$ if and only if $p = q$.  

$D_{KL}$ measures the distance between two distributions.
The EM algorithm

If we also define
\[ L[q, \theta] = \sum_Z q(Z) \log p(X, Z|\theta) \]
then we can re-arrange the last expression to get
\[ \log p(X|\theta) = L[q, \theta] + D_{KL}[q||p] \]
and we know that \( D_{KL}[q||p] \geq 0 \) so that gives us an upper bound
\[ L[q, \theta] \leq \log p(X|\theta). \]

The EM algorithm works as follows:
- We iteratively maximize \( L[q, \theta] \).
- We do this by alternately maximizing with respect to \( q \) and \( \theta \) while keeping the other fixed.
- Maximizing with respect to \( q \) is the E step.
- Maximizing with respect to \( \theta \) is the M step.

The EM algorithm for a mixture model summarized:
- We want to find \( \theta \) to maximize \( \log p(X|\theta) \). But that’s not tractable.
- So we introduce an arbitrary distribution \( q \) and obtain a lower bound
  \[ L(q, \theta) \leq \log p(X|\theta). \]
- We maximize the lower bound iteratively in two steps:
  1. \( E \) step: keep \( \theta \) fixed and maximize with respect to \( q \). This always results in \( q(Z) = p(Z|X, \theta_t) \).
  2. \( M \) step: keep \( q \) fixed and maximize with respect to \( \theta \).
- For the mixture model the \( M \) step is
  \[ \theta_{t+1} = \arg \max_{\theta} \sum_{i=1}^m \sum_{k=1}^K \gamma_i^{(k)} \log p(x_i|\mu_k, \Sigma_k) + \log \pi_k. \]
Reinforcement learning and HMMs

*Hidden Markov Models* are appropriate when our agent models the world as follows:

\[
S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow \cdots \rightarrow S_{t-1} \rightarrow S_t \rightarrow \cdots
\]

\[
\text{Pr}(S_0) \rightarrow E_0 \rightarrow S_1 \rightarrow \text{Pr}(S_1|S_0) \rightarrow E_1 \rightarrow S_2 \rightarrow \cdots
\]

**and only wants** to infer information about the *state* of the world on the basis of observing the available *evidence*.

This might be criticised as unnecessarily restricted, although it is very effective for the right kind of problem.

---

Supervised learners learn from *specifically labelled chunks of information*:

This might also be criticised as unnecessarily restricted: there are other ways to learn.

---

**Deterministic Markov Decision Processes**

Formally, we have a set of states

\[
S = \{s_1, s_2, \ldots, s_n\}
\]

and in each state we can perform one of a set of actions

\[
A = \{a_1, a_2, \ldots, a_m\}.
\]

We also have a function

\[
S : S \times A \rightarrow S
\]

such that \(S(s, a)\) is the new state resulting from performing action \(a\) in state \(s\), and a function

\[
R : S \times A \rightarrow \mathbb{R}
\]

such that \(R(s, a)\) is the *reward* obtained by executing action \(a\) in state \(s\).
Deterministic Markov Decision Processes

From the point of view of the agent, there is a matter of considerable importance:

The agent does not have access to the functions $S$ and $R$.

It therefore has to learn a policy, which is a function $p : S \rightarrow A$ such that $p(s)$ provides the action $a$ that should be executed in state $s$.

What might the agent use as its criterion for learning a policy?

Two important issues

Note that in this kind of problem we need to address two particularly relevant issues:

- The temporal credit assignment problem: that is, how do we decide which specific actions are important in obtaining a reward?
- The exploration/exploitation problem. How do we decide between exploiting the knowledge we already have, and exploring the environment in order to possibly obtain new (and more useful) knowledge?

We will see later how to deal with these.

Measuring the quality of a policy

Say we start in a state at time $t$, denoted $s_t$, and we follow a policy $p$. At each future step in time we get a reward. Denote the rewards $r_t, r_{t+1}, \ldots$ and so on.

A common measure of the quality of a policy $p$ is the discounted cumulative reward

$$V^p(s_t) = \sum_{i=0}^{\infty} \epsilon^i r_{t+i} = r_t + \epsilon r_{t+1} + \epsilon^2 r_{t+2} + \cdots$$

where $0 \leq \epsilon \leq 1$ is a constant, which defines a trade-off for how much we value immediate rewards against future rewards.

The intuition for this measure is that, on the whole, we should like our agent to prefer rewards gained quickly.

The optimal policy

Ultimately, our learner’s aim is to learn the optimal policy $p_{\text{opt}}(s) = \arg\max_p V^p(s)$ for some initial state $s$. Define the optimal discounted cumulative reward $V_{\text{opt}}(s) = V^{p_{\text{opt}}}(s)$. How might we go about learning the optimal policy?

The only information we have during learning is the individual rewards obtained from the environment. We could try to learn $V_{\text{opt}}(s)$ directly, so that states can be compared:

Consider $s$ as better than $s'$ if $V_{\text{opt}}(s) > V_{\text{opt}}(s')$.

However we actually want to compare actions, not states. Learning $V_{\text{opt}}(s)$ might help as

$$p_{\text{opt}}(s) = \arg\max_a [R(s, a) + \epsilon V_{\text{opt}}(S(s, a))]$$

but only if we know $S$ and $R$.

As we are interested in the case where these functions are not known, we need something slightly different.
The \( Q \) function

The trick is to define the following function:
\[
Q(s, a) = R(s, a) + \epsilon V_{\text{opt}}(S(s, a)).
\]

This function specifies the discounted cumulative reward obtained if you do action \( a \) in state \( s \) and then follow the optimal policy.

As
\[
p_{\text{opt}}(s) = \arg\max_a Q(s, a)
\]
then provided one can learn \( Q \) it is not necessary to have knowledge of \( S \) and \( R \) to obtain the optimal policy.

\[V_{\text{opt}}(s) = \max_a Q(s, a)\]

and so
\[
Q(s, a) = R(s, a) + \epsilon \max_a Q(S(s, a), a)
\]
which suggests a simple learning algorithm.

Let \( Q' \) be our learner’s estimate of what the exact \( Q \) function is.

That is, in the current scenario \( Q' \) is a table containing the estimated values of \( Q(s, a) \) for all pairs \((s, a)\).

Q-learning

Start with all entries in \( Q' \) set to 0. (In fact random entries will do.)

Repeat the following:

1. Look at the current state \( s \) and choose an action \( a \). (We will see how to do this in a moment.)
2. Do the action \( a \) and obtain some reward \( R(s, a) \).
3. Observe the new state \( S(s, a) \).
4. Perform the update
\[
Q'(s, a) = R(s, a) + \epsilon \max_a Q'(S(s, a), a).
\]

Note that this can be done in episodes. For example, in learning to play games, we can play multiple games, each being a single episode.

The procedure converges under some simple conditions.

Choosing actions to perform

We have not yet answered the question of how to choose actions to perform during learning.

One approach is to choose actions based on our current estimate \( Q' \). For instance, action chosen in current state \( s = \arg\max_a Q'(s, a) \).

However we have already noted the trade-off between exploration and exploitation. It makes more sense to:

- \textit{Explore} during the early stages of training.
- \textit{Exploit} during the later stages of training.

(This also turns out to be sensible to guarantee convergence.)
Choosing actions to perform

One way in which to choose actions that incorporates these requirements is to introduce a constant $\lambda$ and choose actions probabilistically according to

$$\Pr(\text{action } a | \text{state } s) = \frac{\lambda^{Q(s,a)}}{\sum_a \lambda^{Q(s,a)}}.$$  

Note that:
- If $\lambda$ is small this promotes exploration.
- If $\lambda$ is large this promotes exploitation.

We can vary $\lambda$ as training progresses.

There are two further simple ways in which the process can be improved:

1. If training is episodic, we can store the rewards obtained during an episode and update backwards at the end. This allows better updating at the expense of requiring more memory.
2. We can remember information about rewards and occasionally re-use it by re-training.

Q-learning for nondeterministic MDPs

We now have

$$Q(s,a) = \mathbb{E}(R(s,a)) + \epsilon \sum_{\sigma} \Pr(\sigma | s,a) V_{\text{opt}}(\sigma)$$

and the rule for learning becomes

$$Q'_{n+1} = (1 - \theta_{n+1})Q'_n(s,a) + \theta_{n+1} \left[ R(s,a) + \max_{\alpha} Q'_n(S(s,a),\alpha) \right]$$

with

$$\theta_{n+1} = \frac{1}{1 + v_{n+1}(s,a)}$$

where $v_{n+1}(s,a)$ is the number of times the pair $s$ and $a$ has been visited so far.

Nondeterministic MDPs

The Q-learning algorithm generalises easily to a more realistic situation, where the outcomes of actions are probabilistic.

Instead of the functions $S$ and $R$ we have probability distributions

$$\Pr(\text{new state} | \text{current state}, \text{action})$$

and

$$\Pr(\text{reward} | \text{current state}, \text{action}).$$

and we now use $S(s,a)$ and $R(s,a)$ to denote the corresponding random variables.

We now have

$$V^p = \mathbb{E}\left( \sum_{t=0}^{\infty} \epsilon^t r_{t+1} \right)$$

and the best policy $p_{\text{opt}}$ maximises $V^p$.

Alternative representation for the $Q'$ table

But there’s always a catch...

We have to store the table for $Q'$:

- Even for quite straightforward problems it is HUGE!!! - certainly big enough that it can’t be stored.
- A standard approach to this problem is, for example, to represent it as a neural network.
- One way might be to make $s$ and $a$ the inputs to the network and train it to produce $Q'(s,a)$ as its output.

This, of course, introduces its own problems, although it has been used very successfully in practice.