1 Reinforcement Learning

For an excellent and more thorough treatment of reinforcement learning, I recommend Reinforcement Learning: An Introduction by Richard Sutton and Andrew Barto. The most recent draft is available for free at http://incompleteideas.net/book/bookdraft2017nov5.pdf. In their book, they describe reinforcement learning as follows:

Reinforcement learning is learning what to do—how to map situations to actions—so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. These two characteristics—trial-and-error search and delayed reward—are the two most important distinguishing features of reinforcement learning.

Reinforcement learning is a fascinating field, where agents learn policies (that is, rules regarding which action to take given the current state) based on interacting with the environment and receiving feedback or “rewards”. Therefore, RL problems are often described by the tuple \( \{S, A, R\} \), which correspond to state, action, and reward. The idea is that the agent is in a current state \( S \) and can take any number of actions. For each action, the agent will transition to a new state \( S' \) and receive a reward \( R' \), which will both depend on which action the agent takes. Often, the reward will be zero for a very long time before a non-zero reward is achieved.

There are many forms of reinforcement learning. Some are based on statistical models. These often involve updating the models using new observations and then sampling subsequent actions based on their estimated probability of being optimal. Two of the most common approaches of this variety are Thompson Sampling and Upper Confidence Bound (UCB).

In this class, we focused on examples of Markov Decision Processes (MDPs). MDPs assume the Markov property holds. The Markov Property says that if I know the agent’s current state, then learning about the agent’s previous actions and states gives me no relevant information. In other words, I can ignore the agent’s history, since everything important from the past is captured in the agent’s present state and our current estimates of the environment. This is not always a reasonable assumption, but it is reasonable in the gridworld example that is commonly used to introduce people to MDPs.

Below is an example of gridworld. In gridworld, we assume that the agent is free to move between all the white boxes and will receive a reward of zero whenever he/she is in a white box. The agent cannot leave the grid or enter the black box. Any effort to move out of the grid or into the black box will result in the agent staying put. If the agent makes it to one of the boxes with +20 or -20, then the agent will receive the corresponding reward and will remain put forever (but will only collect the reward once – not each period). Often, the agent discounts future rewards using a discount factor \( \gamma \), and sometimes the agent’s intended actions are perturbed by noise so that they only go in the intended direction say 80% of the time.

Suppose we want to figure out what the agent’s optimal policy would be, that is, for each white box, which action should the agent take? One way to figure this out is value iteration. Value iteration calculates the value of each action for each cell and then selects the action with the highest value. It repeats this process, updating its estimates of the values of the actions as it gets more information. Please see the corresponding R code for an implementation of this.

Value iteration is straightforward when you have information about your environment, such as the transition probabilities and rewards. There is an extension called Q-learning that can be used
when you don’t have information about your environment, or at least don’t have full information
about your environment, but you are still operating in a finite MDP. In Q-learning, the agent
follows a policy with some randomness to explore the space and updates its beliefs about optimal
actions based on realized rewards.

Figure 1: Gridworld Example

2 K-Means Clustering

K-means clustering is a clustering method that finds K clusters of “similar” observations. It works as follows:

1. Randomly initialize K centroids
2. Calculate the distance to each centroid for each point
3. Assign points to the centroid closest to them
4. Update the centroids by taking the average of all points assigned to that cluster
5. Repeat steps 2-4 until convergence

K-Means can be sensitive to the random initialization, so in practice, the algorithm is run several
times, each with different random initializations. For each run, the total within-cluster sum of
squares is calculated, and whichever run had the smallest within-cluster sum of squares is selected.

Clustering methods identify “similar” observations, but some major challenges are explaining
why observations are similar and figuring out what number of clusters K to use. To explain
clusters, it can help to look at variable means by cluster to see if patterns jump out. It can also
be helpful to run PCA and plot the clusters.

3 Hierarchical Clustering

In this course, we have learned about hierarchical agglomerative clustering, which we are calling
hierarchical clustering. There is another kind of hierarchical clustering called hierarchical divisive
clustering, which is a top-down approach (rather than the bottom-up approach of hierarchical
agglomerative clustering).

In hierarchical clustering, each observation starts in its own cluster. Clusters are then iteratively
merged until eventually all observations belong to the same cluster. An appealing feature of
hierarchical clustering is that you can specify the number of clusters K at the end, which makes
it easy to try a number of different K’s at essentially no extra cost.

There are two important metrics you need to specify when running hierarchical clustering.
The first is the distance measure to use, such as Euclidean distance. Hierarchical clustering can
easily accommodate different distance measures and so can be great for binary data, such as sparse
vectors representing which movies each customer watched. In that example application, something
like Jaccard or cosine distance is likely to be more appropriate when trying to identify users with
similar preferences. The other metric you need to specify is the linkage criterion. One option is
complete-linkage, which looks at the farthest points in each cluster to determine which clusters to
merge. Single-linkage looks at the closest points and so can capture more flexible patterns when
they exist, but may risk putting nearly all observations in the same cluster when the patterns are
more ordinary. Average-linkage considers the average distance between all pairs of points in each
cluster. Below is an example showing how single-linkage and average-linkage performed on two
data sets.

![Figure 2: On the left, the clusters were formed using single linkage. On the right, the clusters were
formed using average linkage.](image)

To understand hierarchical clusters, you can do the same visualizations as for K-Means. How-
ever, you can also examine the resulting dendrogram, which is a tree structure that shows how
the clusters were formed. In a typical dendrogram, the observations are listed along the bottom
(e.g. where the x-axis normally is) and a tree structure shows the height (distance) at which each
cluster merges. When the merge height is large, the clusters are dissimilar, when it’s small, the
clusters are very similar.

![Cluster Dendrogram](image)

Figure 3: Dendrogram Example