CS181 Lecture 2 — The Decision Theoretic Framework

Avi Pfeffer

Here is a capsule summary of the decision theoretic framework:

• We use probability to model uncertainty about the domain.

• We use utility to model an agent's objectives.

• The goal is to design a strategy for the agent that maximizes its expected utility.

Decision theory can be viewed as a definition of rationality. The maximum expected utility principle states that a rational agent is one that chooses its strategy so as to maximize its expected utility.

The decision theoretic framework will play two roles for us. The first, and most important, is to provide a precise and concrete formulation of the problem we wish to solve. The second is to provide a guideline and method for designing an agent that solves the problem.

Optional readings for the next four lectures: Russell & Norvig 16, 17, 21; Sutton & Barto "Reinforcement Learning: An Introduction"

1 Problem Formulation

Last time, we defined the goal of AI as building agents "that perform well in complex environments". "Complex" can mean many things, but for this course, the complexity of the environment includes the fact that we have uncertainty about it. There are various sources of uncertainty:

- The agent may not know the current state of the world. There can be a number of reasons for this:
  - Its sensors only give it partial information about the state of the world. For example, an agent using a video camera cannot see through walls.
  - Its sensors may be noisy. For example, if the camera is facing the sun, it may incorrectly perceive objects due to lighting effects.

- The effects of the agent's actions might be unpredictable. For example, if the agent tries to pick up a plant and eat it, it might accidentally crush it instead.
- There are exogenous events, completely outside the agent's control, that it might have to deal with. For example, if it has planned a path from one room to another going through a door, someone might shut the door before it gets through.
- Over and above the agent's uncertainty in a particular situation, we might also have uncertainty about the correct model of the world.

In the decision theoretic framework, all these different kinds of uncertainty will be modeled using probabilities. For the most part we will ignore the last kind of uncertainty, uncertainty over the model itself. Modeling this uncertainty explicitly is a key component of Bayesian learning, which we will touch on later in the course.

Continuing our elaboration of the problem definition, "performs well" means that it gets good utility. Of course, because of the uncertainty, we can't devise a strategy that guarantees that it will get good utility in all circumstances. Instead, we need to try to design the agent so that it gets good utility on average, i.e., in expectation.

Let's make this notion a little more precise. Assume that our agent has some evidence E about the world, derived from its percepts. The agent can take various actions $A_1 ... A_n$. Each action can have various different results. For example, action A_1 has possible results $Result_1(A_1) ... Result_m(A_1)$. In order to frame the problem, we need to assess two things. First, with each result R we need to associate a utility U(R). Second, we need to assess the probability of each possible result for each action, given our evidence. We'll write this as $P_A(R \mid E)$. Now, we can define the expected utility of taking action A_1 as

$$EU(Do(A_1) \mid E) = \sum_{i=1}^{m} P_{A_1}(Result_i(A_1) \mid E) U(Result_i(A_1)) \quad (1)$$

and similarly for the other actions. The maximum expected utility principle, or MEU principle, says that we should choose the action that maximizes this expected utility.

Note that as we have formulated things, this is a one-shot deal. The agent gets to choose one action, so as to maximize one expected utility. In reality, we're interested in designing agents that have repeated interactions with the world. An agent will get some information about the world, take an action, get more information, take another action, and so on. We'll start discussing this type of repeated interaction next time, in the Markov Decision Problem framework. In the meantime, there's plenty to discuss about the single-shot problem.

Let's look at Equation (1) again. We need to distinguish between design time and execution time. (This is similar to the compile time vs run time distinction for compilers.) At execution time, the agent has some evidence E about the world, and needs to choose its action A on the basis of that evidence. At design time, we don't have access to the evidence that the agent will have at execution time. We need to design an agent that will work for any possible evidence. In other words, we need to define a mapping $f : E \rightarrow A$ from the space of possible evidence E to the space of actions A. Ideally, we would like to ensure that for any evidence E, f(E) is the action A such that EU(A | E) is maximal. If we succeed in doing this, the agent will be fully rational. This is very hard to achieve in general. Usually, we will establish full rationality as an

ideal, and try to get as close to it as possible.[1]

2 Examples

Consider the following challenge. We have to design a robot that will go to a strange planet, and try to survive for as long as possible. The robot needs energy in order to survive, and it gets its energy by eating plants. Some of the plants on the planet are nutritious, giving the robot energy, while others are poisonous, resulting in a loss of energy. So the robot needs to navigate around the planet, eating nutritious plants and avoiding poisonous ones.

Let's focus on the subproblem of deciding whether or not to eat a plant. First of all, let's formulate this as a classification problem. The robot's evidence E consists of features of the plant, and the possible actions are the two classifications Nutritious and Poisonous. To be specific, let's say that our robot has a camera, which gives it a 6x6 pixel image of the plant. Each pixel is either on or off. The evidence then consists of the features $X = X_1,...,X_{36}$. Our task is to define a function $f(X) \rightarrow \{\text{Nutritious, Poisonous}\}$. We formulate the utility function by assigning a utility of 1 if the classification is correct, 0 if it is incorrect.

This is the classical formulation of the classification problem in machine learning. We are given a set of features, and a set of possible classifications, and want to learn a function from the features to the classes. There are two kinds of training data one might use:

- Labeled training data: each datum D_i consists of features X_i and the class C_i. This is the supervised learning problem.
- Unlabeled training data: each datum D_i consists of features X_i, but no class label. This is the unsupervised learning problem.

Is this the appropriate formulation for our robot who decides whether or not to eat the plant? If eating a poisonous plant results in a greater loss of energy than is gained by eating a nutritious plant, then the cost of incorrectly classifying a plant as nutritious is greater than the cost of incorrectly classifying it as poisonous. In order to take this point into account, let's use a more refined version of the utility function: 0 if the agent decides not to eat the plant, +10 if the agent eats the plant and it is nutritious (energy gained), -20 if the agent eats the plant and it not nutritious (energy lost).

Sometimes, a more refined utility function is important in a traditional machine learning context. For example, consider the problem of medical diagnosis. Suppose we have a test to determine whether someone has cancer. There are two kinds of errors a test may have:

False positive The test says a person has cancer but in fact they do not.

False negative The test says a person does not have cancer but in fact they do.

There is some cost associated with a false positive, because a person will become needlessly worried, and they may receive needless treatment. However, the cost of a false negative is far

[1] In fact, there is an interesting decision-theoretic meta-problem here. If we have some constraints about the design of the agent, we may not be able to achieve full rationality. We may however have some flexibility in the design of the agent, and be able to sacrifice rationality in some cases in order to achieve greater rationality in others. If we ourselves have a probability distribution over the space E of possible evidence, we can estimate the expected utility of any function $f : E \rightarrow A$ by $EU(f) = \sum_E P(E)EU(Do(f(E)) \mid E)$. Our goal then becomes to choose the f that maximizes this expected utility, subject to the design constraints.

greater, because it may result in a person not being treated in time, and loss of life.

Although this is an important point to consider for real world applications of machine learning, most of the machine learning literature tends to ignore it, and we will too when we discuss the machine learning problem, assuming a simple symmetric utility function.

Let's go back to examining the utility function for our robot. Is the second formulation sufficient? In reality, the agent needs to consider various other factors in determining whether or not to eat a plant that it is not sure about. For example, if the agent is very low on energy and about to die if it doesn't eat, it should be much more willing to eat a suspect plant. Its decision should also depend on the proximity of other plants. It is fairly difficult to take these factors into account in a single-shot utility function.

One more example. Suppose you're building an agent to bid in internet auctions. The possible results of a single auction are that you failed to buy the item, or you bought it for price p. What is an appropriate utility function? It depends on the value v that the user assigns to the item. A natural utility function is 0 if you don't buy the item, $v - p$ if you do.

3 What are Utilities?

We've been blithely throwing around the term "utility" without stopping to think what exactly it means. I assume everyone is comfortable with what "probability" means. Just to be precise, and to make sure we're all on the same page, here's a definition. We assume that there is a space S of possible outcomes, called the sample space. In this class, S will usually be a finite space $\{S_1,...,S_n\}$, though of course continuous sample spaces are also possible. Assuming finite S, a probability distribution over S is a function $P : S \rightarrow [0, 1]$ such that $\sum_{i=1}^{n} P(S_i) = 1$. Two standard interpretations are given for the notion of probability:

Frequentist $P(S_i)$ is the fraction of times outcome S_i will be chosen if the experiment of choosing a sample from S is repeated many times. Under this interpretation $P(S_i)=1$ means that S_i will (almost) always be chosen, and $P(S_i) = 0$ means that S_i will (almost) never be chosen.

Subjectivist $P(S_i)$ is the degree of belief of an agent in outcome S_i. Under this interpretation $P(S_i) = 1$ means the agent is sure that the outcome is S_i, while $P(S_i) = 0$ means the agent is sure it is not S_i.

It's not important for the purposes of this class which interpretation of probability is chosen. Just use whatever you are most comfortable with.

While probability is a well understood notion, and we intuitively understand what probabilities like 1, 0 and 0.637 mean, it is far less clear what utilities mean.

The simplest answer is that utility is a measure of happiness. But what does that mean? What is a happiness of 1, 3.27, or -15?

Can we measure utility by money? This works well for the internet auction example, but is not necessarily appropriate in general. Consider the following game: you have 99% chance of losing $10,000, and 1% chance of winning $1,000,000. Many people would be averse to playing this game, but the expectation of the game is to win $100. This seems to indicate that having $1,000,000 is less than 99 times as good as losing $10,000 is bad, so money is not an ideal measure of utility.

Rather, utility is an encoding of preferences. Given a choice between outcomes A, B and C,

you might rank them as A being better than B which is better than C. But by how much do you prefer A to B and B to C? If you had to choose between two lotteries, one that gets you B for sure, while the other gets you 50% chance of A and 50% chance of C, which would you choose?

The foundation of utility theory is based on the following idea. I can ask you the same type of question about all sorts of different lotteries. If you're rational, your answers to these questions should satisfy certain fundamental properties. For example, the axiom of "substitutability" says that if you are indifferent between two lotteries A and B, then you should be indifferent between two more complex lotteries that are the same except that B replaces A in one part. A full list of axioms can be found in the Russell and Norvig book.

One can then prove that if your preferences satisfy these axioms, then there is some function U that can be assigned to outcomes, such that you prefer one lottery to another if and only if the expectation of U under the first lottery is greater than the expectation under the second. We call this function U your utility function.

An interesting point to note, however, is that the function U encoding your preferences is not unique! In fact, it can be shown that if U encodes your preferences, then so does V = aU +b where a > 0.

The moral of this is that the utility numbers themselves don't mean anything. One can't say, for example, that positive utility means you're happy, and negative utility means you're unhappy, because the utility numbers could just be shifted in the positive or negative direction. All utilities achieve is a way of encoding the relative preferences between different outcomes. Nevertheless, this is a sufficiently important task, and utilities are a sufficiently convenient way of achieving it, that we use utilities ubiquitously in our work.

4 Solving the MEU Problem

As we've discussed, one could almost view "solving the maximum expected utility problem" as a definition of AI. It is important to note, however, that MEU is first and foremost a way for us to describe the problem that an agent needs to solve. The agent itself does not have to explicitly reason about utilities or compute expected utilities in order to solve the problem. Sometimes we

will design an agent that solves an MEU problem without explicitly considering it. Other times, we will have the agent explicitly compute expected utilities of its actions, given the problem formulation.

For example, let us consider again the problem of classifying a plant as Nutritious or Poisonous. Consider the first formulation of the utility: 1 if the classification is correct, 0 otherwise. It is clear that in order to maximize its expected utility in this case, the agent should classify as Nutritious if and only if it thinks Nutritious is more likely than Poisonous. All that is needed is to determine which of the two classes is more likely; the actual probabilities of the two classes are irrelevant. Thus, the agent should classify as Nutritious whether it thinks Nutritious is true with 51% or 99% probability.

Since the actual probabilities are not needed, it seems that perhaps the agent should not have to explicitly compute P(Nutritious) given the plant features. All that is needed is a way of determining, based on the plant features, which of the two classifications is more likely. I.e., all that is needed is a function $f : E \rightarrow \{\text{Nutritious, Poisonous}\}$, such that

$$f(E) = \text{Nutritious} \Leftrightarrow P(\text{Nutritious} \mid E) > P(\text{Poisonous} \mid E).$$

In the classical formulation of the machine learning problem, we try to learn just such a function. As we shall see, there are many approaches to this problem that don't try to estimate the probability.

On the other hand, there are also approaches that try to actually learn the probability P(Nutritious | E) directly. It's clear that if we know this probability, we can make the right classification. This is the approach used by the probabilistic classification methods we shall discuss later in the course.

Now, let's consider the second utility formulation for the "shall I eat" problem: recall that it is 0 if you don't eat, +10 if you eat and it's nutritious, -20 if you eat and it's poisonous. For this formulation, the natural approach is not just to decide which is the most likely class, but to determine P(Nutritious) based on features. Once you have that, you can determine EU(Eat) and EU(¬Eat). E.g, if P(Nutritious)=0.7, we have EU(Eat)=0.7∗10 + 0.3∗(−20) = 1, while EU(¬Eat) = 0, so eating is the correct decision.

One could, however, also treat this formulation as a straightforward classification problem. The features are the same as before. The classes are no longer P and N, but Eat and ¬Eat, where the class is Eat iff Eat is the correct decision. One might get training data for this by watching the actions of another agent.

A third approach, instead of determining P(N), just tries to estimate the expected utility of eating and not eating. Look at Equation (1) defining expected utility again.

$$EU(Do(A_1) \mid E) = \sum_{i=1}^{m} P_{A_1}(Result_i(A_1) \mid E)U(Result_i(A_1)) \quad (2)$$

Within this formula, we can view $Result_i(A_1)$ as a dummy variable. Instead of trying to predict the result, and using that to calculate the expected utility according to the formula, we can just try to predict the value of the entire formula directly. For this approach, we try to learn a function that estimates EU(Do(A) | E). One form of reinforcement learning does precisely this.

Whether or not the agent design explicitly proceeds by solving an MEU problem, the formulation of the utility function affects the agent design. Suppose you're working on a team designing the robot for our challenge problem. Suppose you're particular task is to design the eat or not eat decision maker. I.e., you have to come up with a function that takes the plant features, and maybe other features of the domain, and returns a decision of whether or not to eat. You know that you're working on part of a larger problem. Being a good decision theorist, you decide to formulate your own subproblem as an MEU problem (again this does not mean

you plan to design the agent to explicitly solve the MEU equations). You have a choice as to what formulation to choose; your formulation will affect how hard the problem is to solve, and how well your component will fit into the overall agent. There is an interesting tradeoff here.

5 Prior and Posterior Distributions

Let's focus on the explicit approach of actually trying to solve the MEU equation. Look at Equation (1) again. In order to evaluate the formula, we need to assess all the probabilities of the form $P_{A_j}(Result_i(A_j) \mid E)$, i.e., the probability of each action producing each result,

given the agent's evidence. These probabilities look rather hard and unnatural to estimate. The agent's evidence, which consists of percepts, has no effect on the action results directly. Rather, the percepts provide the agent with information about the state of the world, which does have an effect on the results of actions.

We're going to postulate that there is a set S of possible states of the world. In this course we will generally assume that S is a finite set of states $\{S_1,...,S_N\}$. For example, in the case of our plant eating robot, S is {Nutritious, Poisonous}. The agent does not know what state the world is in. However, even before getting any percepts, it has some probability distribution over the possible states. This probability distribution is called the prior distribution, because it is formed prior to receiving any information about the current state of the world. The prior distribution may be a result of the agent having prior beliefs about the world; for example, our robot might believe a priori that 60% of plants are nutritious, and so assign a prior probability of 0.6 to Nutritious. The prior may also be the result of recent history; for example, if the robot has seen several nutritious plants nearby, it may assign a higher prior to this plant being nutritious.

In addition to the prior distribution, the agent also has percepts that give it information about the actual state of the world. In the case of our robot, the percepts are the actual features of the plant that it is considering whether or not to eat. As we said above, it is hard to estimate the probability of a result R given an action A and the evidence E directly. However, it is quite easy to estimate the effect of an action, if the state S of the world is known. I.e., we can easily assess $P_{A_j}(\text{Result}_i(A_j) \mid S)$. This estimate is part of our model of how the world works. In order to use this to solve Equation (1), we need one more thing: An assessment of the probability of each possible state given the evidence, P(S | E). This probability is called the posterior probability distribution, because it is formed after the agent receives particular evidence E about the state of the world.

Once we have the posterior distribution, we can solve Equation (1) as follows.

$$\begin{aligned} & EU(Do(A_j) \mid E) \\ = & \sum_{i=1}^{m} P_{A_j}(\text{Result}_i(A_j) \mid E) U(\text{Result}_i(A_j)) \\ = & \sum_{k=1}^{N} P_{A_j}(S_k \mid E) \sum_{i=1}^{m} P_{A_j}(\text{Result}_i(A_j) \mid E, S_k) U(\text{Result}_i(A_j)) \\ = & \sum_{k=1}^{N} P(S_k \mid E) \sum_{i=1}^{m} P_{A_j}(\text{Result}_i(A_j) \mid S_k) U(\text{Result}_i(A_j)) \end{aligned} \quad (2)$$

The first line is Equation (1) from before. The second line is standard probability theory ("reasoning by cases"). The third line relies on two assumptions. The first says that the agent's percepts are conditionally independent of the results of actions, given the actual state of the world. In other words, if we know the actual state of the world, then knowing the agent's percepts gives us no additional information about the results of actions. The second assumption says that the agent's beliefs about the state do not depend on the action, i.e., that $P_{A_j}(S_k \mid E)$ is the same for all j.

6 Sensors

In order to solve the MEU problem using Equation (2), we need one more thing: to compute $P(S_k \mid E)$, the posterior probability of each possible world state given the evidence. It turns out that this is hard to estimate directly, because it depends not only on the features but also

on the agent's prior beliefs. For example, given the same plant features, you might think it is poisonous if it is surrounded by other poisonous plants, but nutritious if surrounded by other nutritious plants.

It's much more natural to estimate the opposite probability: P(E|S). The true state of the world determines what you perceive. Just as the effect of the state on actions is part of our model of how the world works, so is the effect of state on percepts. The state does not determine the percepts deterministically but by a probability distribution. For example, a nutritious plant will tend to have certain properties, described by a probability distribution over the features. A poisonous plant will tend to have other properties, described by a different distribution. The distribution P(E|S) is called the sensor model. A sensor is a mechanism for producing percepts about the world. The sensor model is a description, at the level of information, of what the sensor does.

Ok, so now we have the prior distribution P(S) and the sensor model P(E|S). How do we get the posterior distribution P(S|E)? Bayes rule!

Bayes rule says that P(S|E) = P(S)P(E|S)/P(E). In words, the posterior probability of a state S given evidence E is equal to the probability of S (prior) times the probability of E given S (sensor model) divided by the probability of E. An interesting thing to note here is that P(E) does not mention the state S. In fact, P(E) = $\sum_{k=1}^{N} P(S_k)P(E \mid S_k)$. 1/P(E) is called a normalizing factor. We know that for each state S_k, $P(S_k \mid E)$ is proportional to $P(S_k)P(E \mid S_k)$, and that $\sum_{k=1}^{N} P(S_k \mid E) = 1$. So, after computing $P(S_k)P(E \mid S_k)$ for each k, we can just scale the results so that they sum to 1 to get $P(S_k \mid E)$.

Putting everything together, we get the following expression:

$$EU(Do(A_j) \mid E) = \sum_{k=1}^{N} \frac{P(S_k)P(E|S_k)}{P(E)} \sum_{i=1}^{m} P_{A_j}(Result_i(A_j) \mid S_k)U(Result_i(A_j)) \quad (3)$$

A final thing to note about equation (3) is that action A_j maximizes $EU(Do(A_j) \mid E)$ iff it maximizes $P(E)EU(Do(A_j) \mid E)$. So in fact, if our goal is only to solve the MEU problem, and not to obtain a posterior distribution over the states, we don't need to scale by the normalizing factor at all.

7 Types of sensors

There are a wide variety of sensors that are actually used by agents. Here is a small sample:

Camera Perceptual inputs from cameras may be interpreted as single individual images, or a video stream. Also, an agent may have a single camera giving it one visual perspective on the world, or it may use multiple cameras at different locations. The visual image is normally described as a pixel array. It may be a grey-scale image, containing a light intensity (normally between 0 and 255) at every point, or it may be a color image, with RGB intensities. Computer vision is of course an extremely important and interesting kind of sensor. We're not going to cover vision much in this course. We will look at one application, which provides a simple kind of vision problem : optical character recognition.

Touch Many robots come with touch sensors, so that they know when they are in contact with an obstacle or a wall. Robots that manipulate objects need touch sensors to know just

where and what force to apply to an object. Surprisingly, touch is also useful for robot navigation. For example, it is sometimes quite hard to control a robot's motion precisely enough so that it can head for a door and go through it without mishap. A simple solution is for the robot to aim for the wall somewhere near the door, and then slide along the wall until it reaches the door. This sliding kind of motion is called compliant motion.

Sonar Sonar is a common sensor for mobile robots. It works by emitting a sound pulse in a certain direction, and timing how long it takes for the sound to be reflected from an object. A mobile robot will typically be equipped with a circular array of eight sonars. The sonar information will tell it the distance to the nearest object in each of eight directions. Sonar only gives a robot a very rough idea of its surroundings. Furthermore, it is quite a noisy sensor. One problem is that surfaces that are not smooth do not reflect sound back directly to the robot. Nevertheless, sonar is very useful for obstacle avoidance.

Doppler radar We use sensor in a wider sense than just something with which a robot might be equipped. Essentially, it is any mechanism that provides inputs to an AI problem. An example is a Doppler radar, which is similar to sonar in that it emits radio waves and studies how they are reflected. In addition, it measures the change in frequency between the emitted waves and the reflected waves. By the Doppler effect, if a wave is reflected by an object moving away from the radar, its frequency will decrease, while if the object is moving closer the frequency will increase. A Doppler radar is therefore able to detect motion as well as position of objects. Nevertheless, it is still limited: it can detect radial motion, toward and away from the radar, but not lateral motion. An AI problem that uses a Doppler radar is as follows: given a profile of vehicle motions over time, try to determine which vehicle went where, and perhaps what kinds of vehicles they are.

Blood Test Another "sensor" in our wider sense is a blood test. Here, the true state is the actual condition of the patient. The blood test is a noisy sensor providing information about the patient. An AI problem that uses blood test as a sensor is automated medical diagnosis.