10 Building and plotting a decision tree

This chapter covers

- Decision-making with and without probabilities
- Maximax and Maximin methods
- Minimax Regret method
- Expected Value method
- Decision trees

Decision-making is a critical process in both personal and professional contexts, where the stakes can be incredibly high—whether it's managing large sums of money, making career-defining choices, or navigating situations where people's lives are at risk. In such scenarios, an analytical and systematic approach to decision-making is not just beneficial but essential. Throughout this chapter, we will explore various decision-making methods, with a particular focus on building and plotting decision trees. Unlike the decision trees used in machine learning (see chapter 6), which are designed to classify data, these trees serve as a graphical representation of the decision-making process, outlining possible choices, chance events, associated probabilities, and potential payoffs. This approach helps to systematically determine the best course of action by applying the Expected Value

method, which uses probabilities to weigh potential outcomes and make informed decisions.

Before diving into the Expected Value method, we will first explore decision-making approaches that do not rely on probabilities. These methods are crucial when probabilities for different outcomes are unknown or cannot be accurately estimated. We will discuss the Maximax, Maximin, and Minimax Regret methods, each suited to different decision-making styles and levels of risk tolerance. The *Maximax method*, favored by risk-takers, selects the option with the highest possible payoff by assuming the best-case scenario will occur. The *Maximin method*, designed for risk-averse decision-makers, chooses the option with the best worst-case scenario, thereby ensuring protection against unfavorable conditions. The *Minimax Regret method* minimizes the worst-case regret by choosing the option where the maximum possible difference between its outcome and the best possible outcome in each scenario is smallest.

When probabilities are known or can be estimated, the Expected Value (EV) method becomes a powerful tool for decision-making. By calculating the weighted average of all possible outcomes, the EV method balances risk and reward, offering a rational approach to choosing among alternatives. Decision trees, as a graphical extension of the EV method, allow for a more comprehensive analysis by visually mapping out decisions, chance events, probabilities, and payoffs. This visual representation simplifies the comparison of different scenarios, making it easier to identify the most advantageous course of action.

Recognizing that good decisions are not necessarily those with the best outcomes but those that are well-informed and systematically analyzed is key to effective decision-making. Even the most carefully considered decisions can lead to unfavorable outcomes due to inherent uncertainties. This principle is reflected in fields like reinforcement learning, where optimal strategies are derived from evaluating expected rewards over time. Therefore, understanding and applying the appropriate decision-making methods is essential in navigating complex and uncertain environments.

This chapter will begin by demonstrating decision-making methods that do not consider probabilities, providing foundational tools for uncertain situations. We will then transition to the EV method, showing how it can be visualized and enhanced using decision trees. By the end of this chapter, you will be equipped with a range of decision-making techniques and the ability to effectively use decision trees to analyze and resolve real-world problems with confidence.

10.1 Decision-making without probabilities

In decision-making, there are scenarios where the probabilities of different outcomes are unknown or cannot be reliably estimated. This introduces a level of uncertainty that requires different approaches to make informed choices. When probabilities are not available, decision-makers rely on various methods to evaluate potential actions and their associated payoffs. These methods, such as the Maximax, Maximin, and Minimax Regret, provide structured ways to assess options based on the best, worst, or regret-minimizing outcomes. In the following sections, we will explore these decision-making

methods with illustrative data, explaining each in detail and demonstrating their application through Python code snippets.

10.1.1 Maximax method

The *Maximax method*, also known as the *optimistic approach*, is one decision-making strategy used in situations where the decision-maker has no control over the states of nature and there are no known or estimated probabilities for these states. This method assumes that the most favorable outcome will occur and focuses on identifying the decision alternative that offers the highest possible payoff. Payoff tables are typically used in this approach (and others), listing the various decision alternatives and the corresponding payoffs under different states of nature. These payoffs can represent profits, costs, or any other relevant measure of value.

For example, consider a pending decision between multiple investment opportunities. The states of nature might include different market conditions—strong or weak—over the next 12 months. The payoffs in the table would show the expected profits under each of these conditions. The Maximax method involves scanning the payoff table to find the maximum payoff for each decision alternative and then selecting the alternative with the highest of these maximum payoffs. This approach appeals to those with a highly optimistic outlook, as it assumes the best possible scenario will materialize.

To effectively apply the Maximax method, or any decision-making strategy, a systematic approach is essential. This process involves several key steps:

1. *List the possible alternatives.* Begin by identifying all possible decision alternatives available. These are the different courses of action that the decision-maker can choose from. Exactly *one* of these alternatives should be selected.
2. *Identify the possible states of nature that can occur.* These are the external conditions or scenarios that can affect the outcomes of the decision but are beyond the control of the decision-maker.
3. *List the payoffs for each combination of decision alternative and state of nature.* Payoffs can be measured in various terms, such as profits, costs, or utility, depending on the context of the decision.
4. *Select a decision-making method that suits the context and goals of the decision, and then decide.* For instance, using the Maximax method involves identifying the highest possible payoff for each alternative and selecting the alternative with the maximum of these values.

By following these steps, you can construct a payoff table, apply a decision-making methodology, and arrive at a well-informed final decision.

LIST THE POSSIBLE ALTERNATIVES

Imagine you have $1 million to invest. Your first step is to identify the investment alternatives under consideration. This involves simply listing the options available to you. They are as follows:

- Invest all $1 million in the stock market.
- Invest all $1 million toward a minority share in a professional soccer club.
- Divide the $1 million into two halves: put $500,000 into the stock market and the other $500,000 toward a small ownership stake in a professional soccer club.
- Deposit all $1 million in a savings account that pays 3.5% annually.

Accordingly, we create a vector called `investment_alternatives` that contains these four options:

```
>>> investment_alternatives = [
>>>     "Stock Market",
>>>     "Professional Soccer Club",
>>>     "Stock Market / Soccer Club (50/50)",
>>>     "Savings Account"
>>> ]
```

In a bit, `investment_alternatives` will be joined with a pair of other vectors to create a data frame that will double as a payoff table.

IDENTIFY THE POSSIBLE STATES OF NATURE

The second step is to determine the potential states of nature that, although beyond your control, can significantly affect the outcomes of your investment alternatives. These states of nature represent various scenarios or conditions under which the investments might perform differently. For instance, market conditions can dramatically influence the performance of any investment strategy within a year's time. Identifying these states of nature is crucial as it allows you to assess the potential risks and returns associated with each investment alternative under different scenarios. By considering these possibilities, you can better understand the range of outcomes and make more informed decisions.

For this example, we will consider two states of nature: a strong economy versus a weak economy. To represent these states, we initialize two vectors accordingly:

```
>>> strong_economy = []
>>> weak_economy = []
```

Next, we will add the payoff amounts to these vectors and combine them with `investment_alternatives` to create a data frame, thereby forming a comprehensive payoff table.

LIST THE PAYOFFS FOR EACH ALTERNATIVE AND STATE OF NATURE COMBINATION

In the third step, we list the payoffs for each combination of investment alternatives and states of nature. This involves determining the potential outcomes or returns for each alternative under both the strong and weak economic scenarios. Payoffs are typically represented as monetary values, such as profits or losses. By assigning specific payoff amounts to each alternative based on the assumed economic conditions, we can construct a detailed payoff table. This table serves as the foundation for applying various decision-making methods and ultimately guides us in selecting the best investment strategy.

With that in mind, let's now add the payoff amounts to the `strong_economy` and `weak_economy` vectors:

```
strong_economy = ([100000, 60000, 80000, 35000])
weak_economy = ([-120000, -20000, -70000, 35000])
```

These are dollar amounts that represent the estimated one-year profit or loss for each investment alternative under different economic conditions. For example, a $1 million investment in the stock market will yield a $100,000 profit in a strong economy, but the same investment will result in a $120,000 loss in a weak economy.

This will become much clearer by creating a payoff table. We can do this by combining the three vectors into a pandas data frame, which will provide a structured and easily readable format for analyzing the estimated one-year profits or losses for each investment alternative under different economic conditions:

```
>>> import pandas as pd
>>> payoff_table = pd.DataFrame({
>>>     'Investment Alternatives': investment_alternatives,
>>>     'Strong Economy': strong_economy,
>>>     'Weak Economy': weak_economy
>>> })
>>> print(payoff_table)
              Investment Alternatives  Strong Economy  Weak Economy
0                        Stock Market          100000       -120000
1            Professional Soccer Club           60000        -20000
2  Stock Market / Soccer Club (50/50)           80000        -70000
3                     Savings Account           35000         35000
```

A *payoff table*, sometimes referred to as a *decision table*, is a structured way to display the possible outcomes of various decision alternatives under different states of nature. Each row represents a different investment alternative, whereas each column represents the potential economic conditions, such as a strong or weak economy. The values within the table indicate the estimated one-year profit or loss associated with each combination of investment alternative and economic state. This table supports various decision-making approaches, including the Maximax method, by providing a clear visual representation of the potential payoffs for each scenario. By analyzing this table, decision-makers can systematically evaluate and compare the outcomes to make more informed and strategic investment decisions.

SELECT A DECISION-MAKING METHOD, AND MAKE A DECISION

The Maximax method involves a straightforward two-step process for decision-making. First, you identify the highest payoff for each investment alternative and store these values in a new column within the payoff table. This step highlights the most optimistic outcomes for each alternative, reflecting the decision-maker's focus on the best possible scenario. Second, you find the highest value in this new column. The investment alternative corresponding to this highest payoff is the one selected under the Maximax method. This approach is suited for decision-makers who prioritize the potential for maximum gains, even if it involves higher risks.

To achieve this programmatically, we begin by creating a deep copy of the `payoff_table` data frame and assigning it to a new data frame called `maximax`. This step ensures that any changes made to `maximax` don't affect the original `payoff_table`:

```
>>> maximax = payoff_table.copy()
```

Then we add a new column called `Maximax` that takes the highest payoff from each row or investment alternative. The `max()` method computes the maximum value across each row within the `Strong Economy` and `Weak Economy` columns and assigns those values to the new column:

```
>>> maximax['Maximax'] = \
>>>    payoff_table[['Strong Economy', 'Weak Economy']].max(axis = 1)
>>> print(maximax)
             Investment Alternatives  Strong Economy  Weak Economy  Maximax
0                       Stock Market          100000       -120000   100000
1            Professional Soccer Club           60000        -20000    60000
2 Stock Market / Soccer Club (50/50)           80000        -70000    80000
3                     Savings Account           35000         35000    35000
```

That completes the first step. The second step involves finding the highest payoff contained in the `Maximax` column. This is done by applying the `max()` method to the `Maximax` column:

```
>>> highest_payoff = maximax['Maximax'].max()
>>> print("Highest Payoff:", highest_payoff)
>>> Highest Payoff: 100000
```

The highest payoff, a $100,000 profit in a strong economy, is associated with the stock market investment alternative. The Maximax method follows a straightforward, optimistic approach, selecting the option with the highest potential gain without considering risk. As a result, it does not account for potential losses—an alternative with a high payoff but also a significant downside may still be chosen under this method. Therefore, according to this method, you should invest all $1 million in the stock market. However, if you were more risk-averse, you might apply a different decision-making approach, such as the Maximin method, which prioritizes minimizing potential losses.

10.1.2 Maximin method

The *Maximin method*, often referred to as the *pessimistic* or *conservative approach*, is a decision-making strategy used in situations where the decision-maker seeks to minimize potential losses. Unlike the Maximax method, which focuses on maximizing potential gains, the Maximin method is about selecting the alternative with the best of the worst-case scenarios. In this approach, the decision-maker identifies the minimum payoff for each investment alternative across all possible states of nature. These minimum payoffs represent the worst possible outcomes for each decision. The decision-maker then chooses the alternative with the highest minimum payoff, ensuring that

even in the most unfavorable conditions, the outcome will be as favorable as possible compared to other alternatives. This method is particularly useful when there is a high level of uncertainty and risk aversion, as it prioritizes minimizing losses over maximizing gains.

To replicate this process programmatically, we will do the following:

1. Create a deep copy of the `payoff_table` data frame, and assign it to a new data frame called `maximin`.
2. Compute the *minimum* values from each row, and store the results in a new column called `Maximin`.
3. Find the *maximum* value in `Maximin` to determine the best investment strategy.

The code is similar to previous snippets:

```
>>> maximin = payoff_table.copy()
>>> maximin['Maximin'] = \
>>>     payoff_table[['Strong Economy',
>>>                   'Weak Economy']].min(axis = 1)
>>> print(maximin)
             Investment Alternatives  Strong Economy  Weak Economy  Maximin
0                       Stock Market          100000       -120000  -120000
1           Professional Soccer Club           60000        -20000   -20000
2 Stock Market / Soccer Club (50/50)           80000        -70000   -70000
3                     Savings Account           35000         35000    35000

>>> highest_payoff = maximin['Maximin'].max()
>>> print("Highest Payoff:", highest_payoff)
Highest Payoff: 35000
```

Creates a deep copy of payoff_table and assigns it to maximin

Creates a new column to store the minimum value from each row or investment alternative

Prints the maximin data frame

Scans the Maximin column in the maximin data frame and extracts the maximum value

Prints the results

If you were to take the Maximin approach, you would choose to invest your $1 million in a savings account that pays 3.5% annually. This method prioritizes the safest, most conservative option by focusing on the minimum possible payoff for each investment alternative. By selecting the option with the highest minimum payoff, the Maximin approach ensures that you will be guaranteed a $35,000 profit after one year, regardless of the state of the economy. This approach emphasizes security and risk aversion, making it an attractive choice for those who prefer to minimize potential losses. However, because it focuses solely on worst-case outcomes, the Maximin method may overlook higher-reward opportunities and is generally not well-suited for optimizing long-term growth.

10.1.3 Minimax Regret method

The *Minimax Regret method* is a decision-making approach that focuses on minimizing the potential regret associated with a decision, rather than maximizing profit or minimizing loss. *Regret*, in this context, is defined as the difference between the payoff of the best possible action and the payoff of the chosen action, given a particular state of nature. The Minimax Regret method involves several steps:

1. *Construct a regret table by calculating the regret values for each decision alternative under each possible state of nature.* We do this by finding the highest payoff for each state of nature and then subtracting the payoff of each alternative from this maximum value. The resulting regret values indicate the potential missed opportunities or losses associated with not choosing the optimal action for each state.
2. *For each decision alternative, identify the maximum regret.* This represents the worst-case scenario for each option.
3. *Select the decision alternative with the smallest maximum regret.* This approach is particularly useful in situations where decision-makers want to avoid the feeling of regret associated with making suboptimal choices, thereby minimizing the potential negative effects of their decisions. The Minimax Regret method provides a balance between risk and reward, helping to mitigate potential losses while still considering the upside potential of different alternatives.

Thus, the Minimax Regret method is inherently more encompassing than either the Maximax or Maximin approach. To implement this in Python, we will do the following:

1. Create a deep copy of `payoff_table`, and assign it to a new data frame called `regret`.
2. Calculate the maximum values for each state of nature.
3. Create a pair of new columns, `Strong` and `Weak`, that each store the derived opportunity loss associated with every investment alternative and state of nature combination.
4. Create a new column, `Maximum`, that stores the maximum regret for each investment alternative.
5. Take the minimum regret from `Maximum`, and choose the investment alternative associated with that value.

Here's the snippet of code:

```
>>> regret = payoff_table.copy()
>>> max_strong = regret['Strong Economy'].max()
>>> max_weak = regret['Weak Economy'].max()
>>> regret['Strong'] = \
>>>     max_strong - regret['Strong Economy']
>>> regret['Weak'] = \
>>>     max_weak - regret['Weak Economy']
>>> print(regret[['Strong', 'Weak']])
   Strong    Weak
0       0  155000
1   40000   55000
2   20000  105000
3   65000       0
```

Creates a deep copy of payoff_table and assigns it to regret

Finds and assigns the maximum value in Strong Economy to the variable max_strong

Finds and assigns the maximum value in Weak Economy to the variable max_weak

Derives an opportunity loss for each row and Strong Economy combination and stores the results in a new column called Strong

Derives an opportunity loss for each row and Weak Economy combination and stores the results in a new column called Weak

Prints the results

```
>>> regret['Maximum'] = \
>>>     regret[['Strong', 'Weak']].max(axis = 1)
>>> print(regret[['Strong', 'Weak', 'Maximum']])
   Strong    Weak  Maximum
0       0  155000   155000
1   40000   55000    55000
2   20000  105000   105000
3   65000       0    65000

>>> min_max_regret = regret['Maximum'].min()
>>> print(min_max_regret)
55000
```

Creates a new column in regret by taking the maximum value from each row across the Strong and Weak columns

Prints the results

Finds the minimum value in the Maximum column from regret and assigns it to min_max_regret

Prints the best minimum opportunity loss, or the alternative with the least regret

The results of the Minimax Regret analysis indicate that your best investment alternative is the one associated with the minimum value in the `Maximum` column. By minimizing the potential regret, the chosen alternative ensures the fewest missed opportunities. In this case, the best investment alternative, according to the Minimax Regret method, is to invest your $1 million in a professional soccer club.

In summary, decision-making methods such as Maximax, Maximin, and Minimax Regret provide different strategies for selecting the best investment alternative without applying probabilities to the various states of nature. The Maximax method focuses on maximizing the potential payoff by choosing the most optimistic scenario, whereas the Maximin method aims to maximize the minimum payoff, ensuring the best outcome in the worst-case scenario. The Minimax Regret method, on the other hand, minimizes potential regret by considering the missed opportunities. Each of these methods offers a unique perspective on decision-making under uncertainty. However, these approaches do not take into account the likelihood of different states of nature occurring. To incorporate probabilities and make more informed decisions, we turn to the EV method, which provides a more comprehensive analysis by weighing the payoffs against their probabilities.

10.1.4 Expected Value method

The EV method is a widely used decision-making technique in uncertain environments where the probabilities of different outcomes are known or can be estimated. It is particularly valuable in fields like finance, economics, and operations research, where decisions often involve various possible scenarios with associated probabilities. The essence of the EV method is to calculate the average outcome by weighting each possible outcome by its probability and summing these values. This approach helps decision-makers choose the option that offers the best expected return or the least expected loss, making it a fundamental tool for rational decision-making under uncertainty.

It differs from other decision-making strategies like Maximax, Maximin, and Minimax Regret in its approach to uncertainty and risk. Unlike these methods, which focus on either the best, worst, or regretful outcome, the EV method calculates the

average outcome by weighting each possible scenario by its probability. This allows for a more balanced and rational decision-making process by considering the entire range of potential outcomes and their likelihoods, rather than focusing on extremes.

The EV method typically culminates in the construction of a decision tree, which is a graphical representation of the decision-making process. A decision tree consists of nodes representing decisions or chance events and branches representing the possible outcomes of these decisions or events. The tree starts with a decision node, followed by chance nodes that branch out into various possible outcomes, each associated with a probability and a payoff. By calculating the expected value at each decision node, decision-makers can systematically evaluate and compare different strategies, ultimately selecting the one with the highest expected value. This structured approach not only simplifies complex decision-making processes but also provides a clear visual aid for understanding and communicating the rationale behind each decision, making it an invaluable tool in various fields requiring strategic planning and risk management.

To illustrate, let's revisit the `payoff_table` data frame and consider the expected value of a $1 million investment in the stock market, assuming the probability of a strong economy is 0.75 and the probability of a weak economy is 0.25. It is important to note that the probabilities across all states of nature must sum to exactly 1.

The expected value is derived from the following formula:

$$\text{EV(stocks)} = \left(P_{\text{strong}} \times \text{Payoff}_{\text{strong}}\right) + \left(P_{\text{weak}} \times \text{Payoff}_{\text{weak}}\right)$$

where P_{strong} and P_{weak} are the probabilities of a strong and weak economy, respectively, and $\text{Payoff}_{\text{strong}}$ and $\text{Payoff}_{\text{weak}}$ are the corresponding payoffs.

Assuming the payoff for investing in the stock market is $100,000 in a strong economy and −$120,000 in a weak economy, we can plug in the values:

$$\text{EV(stocks)} = (0.75 \times 100{,}000) + (0.25 \times -120{,}000)$$

or

$$\text{EV(stocks)} = 75{,}000 + (-30{,}000)$$

or

$$\text{EV(stocks)} = 45{,}000$$

Thus, the expected value of investing in the stock market, given the specified probabilities and payoffs, is $45,000. This positive expected value indicates that, on average, investing in the stock market under these conditions would result in a significant gain.

To achieve the same result programmatically and cascade the same calculation to the remaining investment alternatives, we do the following:

1. Assign probabilities to represent the likelihood of a strong versus weak economy. Because the probabilities across all states of nature (two in this case) must sum to exactly 1, we add a validation step so that if the probabilities do not sum to 1, Python returns an error message.
2. Write an arithmetic operation that calculates the expected value for each investment alternative and stores the results in a new column appended to the `payoff_table` data frame called `EV`.
3. Extract the maximum expected value from the EV column and make an investment decision based on that value.

Here's the snippet of Python code:

```
>>> strong_econ = 0.75
>>> weak_econ = 0.25
>>> if strong_econ + weak_econ != 1:
>>>     raise ValueError('The probabilities ' \
>>>             must sum to 1.')

>>> payoff_table['EV'] = \
>>>   (payoff_table['Strong Economy'] * strong_econ) + \
>>>   (payoff_table['Weak Economy'] * weak_econ)
>>> print(payoff_table)
             Investment Alternatives  Strong Economy  Weak Economy       EV
0                       Stock Market          100000       -120000  45000.0
1           Professional Soccer Club           60000        -20000  40000.0
2 Stock Market / Soccer Club (50/50)           80000        -70000  42500.0
3                    Savings Account           35000         35000  35000.0

>>> max_ev = payoff_table['EV'].max()
>>> print("Maximum Expected Value:", max_ev)
Maximum Expected Value: 45000.0
```

Assigns a 75% probability of a strong economy

Assigns a 25% probability of a weak economy

Throws an error if the probabilities of strong_econ and weak_econ do not sum to 1

Calculates the expected value for each investment alternative and stores the results in a payoff_table column called EV

Prints the revised payoff_table data frame

Extracts the maximum expected value from the EV column

Prints the highest expected value

The results are very close when we examine the expected value across each investment alternative. Even a small adjustment to the `strong_econ` and `weak_econ` estimates could tip the scales from one investment decision to another. Nevertheless, based on the calculated expected values, you should invest your $1 million in the stock market.

The EV method provides a systematic approach to decision-making under uncertainty by calculating the weighted average of possible outcomes based on their probabilities. As demonstrated, the expected values for different investment alternatives help identify the most favorable option: in this case, the stock market, with an expected value of $45,000. This quantitative evaluation aids in making informed choices by considering the likelihood of various scenarios. To further enhance our decision-making process, we can show these (or other) alternatives and their associated probabilities through a decision tree. A decision tree graphically represents decisions, possible events, and their outcomes, allowing for a clearer understanding of the

potential paths and payoffs. Let's proceed to build and plot a decision tree to illustrate this process.

10.2 Decision trees

A decision tree is a graphical representation of the decision-making process, outlining the various paths one can take, the possible outcomes, and the associated probabilities and payoffs. Unlike a payoff table, which presents this information in a tabular format, a decision tree provides a more intuitive and visual approach, making it easier to grasp complex scenarios and their potential effects. It's important to note that these decision trees differ significantly from the decision tree model fit in chapter 6, which was used for data classification. Here, the focus is on mapping out decision processes rather than predicting outcomes based on data.

In a decision tree, there are two main types of nodes: decision nodes and chance nodes. *Decision nodes* signify points where a choice must be made. From these nodes, branches extend to chance nodes, which represent uncertain events with different possible outcomes. Each branch from a *chance node* carries a probability, indicating the likelihood of that particular outcome. The end points of these branches display the *payoffs*, which are the results of the decisions and chance events.

Using the investment alternatives EV solution as an example, a decision tree would start with a decision node representing the choice among the different investment options. From this node, branches would lead to chance nodes representing the economic conditions (strong or weak economy). Each chance node would then branch out to show the payoffs associated with each investment under different economic conditions, weighted by their respective probabilities.

Decision trees can be oriented either horizontally or vertically. When drawn horizontally, they extend from left to right, and when vertical, from top to bottom. However, the interpretation of decision trees works in the opposite direction: horizontally oriented trees are read from right to left, and vertically oriented trees from bottom to top. This reverse interpretation ensures that all probabilities and payoffs are considered cumulatively from the outcomes back to the initial decision.

The visual nature of decision trees makes them superior to payoff tables in many cases, as they allow for a clearer understanding of the sequence of events and the cumulative effect of decisions and chance occurrences. This comprehensive view aids in making more informed and strategically sound decisions.

The first step in constructing a decision tree is to create the schema. A decision tree *schema* is a structured blueprint that outlines the various decision points, chance events, their probabilities, and associated payoffs. It provides a clear framework to follow when building the tree, ensuring that all potential outcomes are accounted for systematically.

Once the schema is established, the tree can be plotted based on this detailed outline. We will demonstrate how to create the schema manually, showing the step-by-step process of defining decision nodes, chance nodes, probabilities, and payoffs. Additionally, we will show how to create the schema programmatically using methods from

Python's `anytree` library, which simplifies the creation and manipulation of tree structures. Finally, we will use Matplotlib to plot the decision tree, providing a visual representation that enhances understanding and aids in the decision-making process.

By following these steps, we can construct a clear and comprehensive decision tree that effectively communicates the decision-making process and helps identify the most favorable outcomes. To better demonstrate this, we will be working with new illustrative data and a new scenario involving a Major League Baseball player in free agency.

10.2.1 Creating the schema

A Major League Baseball player is currently in free agency, a period when he is free to negotiate and sign with any team after his previous contract has ended. He is considering two contract offers: one from the New York Yankees and another from the Los Angeles Dodgers. His primary goal is to sign with the team that will provide the most financial benefit over the lifetime of the contract, which is the same length for both offers.

The Yankees' offer is straightforward, guaranteeing the player $120 million after deducting agent fees, with no performance-based incentives. In contrast, the Dodgers' offer is more complex, potentially paying up to $150 million if specific performance thresholds (such as hits, home runs, etc.) are met. However, if these thresholds are not met, the player could earn as "little" as $90 million. The player understands that his ability to meet these performance targets depends heavily on his health. Regardless of his health, there are probabilities associated with meeting all, some, or none of the performance targets. Given these considerations, the player must carefully evaluate the potential financial outcomes of each contract offer to make the best decision for his career and financial well-being.

Steps and best practices

Creating a decision tree schema is the first step in constructing a decision tree. The schema acts as a blueprint, detailing the various decision points, chance events, probabilities, and payoffs. This structured approach ensures that all possible outcomes and their probabilities are systematically considered. Here are the steps and best practices to create a decision tree schema:

1. *Identify the primary decision points.* These are typically the major choices to be made. In our example, the decision points would be the player's choice between signing with the Yankees or the Dodgers.
2. *Identify and list any relevant variables that affect the decision.* For instance, agent fees need to be deducted from the total pay.
3. *Label each node as either a decision node or a chance node.* Decision nodes are points where a choice must be made. Chance nodes represent uncertain events with different possible outcomes.
4. *Outline the outcomes.* For each decision node, outline the possible outcomes or actions; for each chance node, list the possible events that could occur and their respective probabilities.

5. *Assign a probability to each chance event.* Ensure that the probabilities for all events stemming from a single chance node sum to 1.
6. *Determine and specify the payoffs associated with each possible outcome.* Payoffs are the results of the decisions and chance events.
7. *Structure the schema using indents or tabs to represent different levels of decisions and events.* This hierarchical structure helps show the flow from decision points to chance events and their respective outcomes.

By following these steps and using indents or tabs to structure the information hierarchically, we can create a comprehensive decision tree schema that clearly outlines all possible decisions, chance events, probabilities, and payoffs. This schema will then serve as a guide for constructing and visualizing the actual decision tree.

CREATING THE SCHEMA MANUALLY

The first step in building a decision tree is creating the schema, which serves as a structured blueprint for the decision-making process. This involves defining the decision tree's name, listing relevant variables, and identifying the main decision node. The decision tree's name encapsulates the overall decision context, such as Contract Options for a player considering different team offers. Variables such as agent fees are key factors that influence the decision. The decision node is the starting point of the tree, representing the primary choice to be made, such as whether to sign with the Yankees or the Dodgers. This foundational setup ensures that the decision tree is organized and ready for detailed branching into outcomes and probabilities.

At this stage, we are not using Python to create the schema; instead, we are constructing it manually. This manual process helps in understanding the structure and components of the decision tree before automating it programmatically:

```
name: Contract Options
variables:
    agent fees: 13,000,000
type: decision
Sign with Dodgers:
Sign with Yankees:
    payoff: 120,000,000
```

Assigns the name Contract Options to the decision tree

Declares a variable named agent fees with a value of 13,000,000. Nesting agent fees under variables indicates that agent fees is a specific variable related to the decision tree.

Specifies that the node type is a decision node (and not a chance node)

Defines the first decision branch, indicating one of the possible choices at the decision node

Defines the second decision branch, indicating another possible choice at the decision node

Assigns a payoff value of 120,000,000 associated with one of the two possible decisions

So far, we have created a comprehensive schema for our decision tree. The decision tree is named `Contract Options`. We have defined a variable, `agent fees`, which will be subtracted from the expected value of the decision to sign with the Dodgers. At the root of our decision tree, we have a decision node with two primary choices: `Sign with Dodgers` and `Sign with Yankees`. The Yankees' offer includes a guaranteed payoff of $120,000,000, including the subtraction of agent fees. In contrast, the Dodgers'

contract heavily relies on performance incentives, causing the total contract payout to vary based on different states of nature.

We then take the first step in building out the schema for the `Sign with Dodgers` part of the decision tree by applying the `agent fees` variable and creating a chance node. A chance node represents an uncertain event with various possible outcomes, each associated with a probability. This step sets up the foundation for detailing the probabilities and outcomes related to the player's health and performance incentives.

Additions to the schema are highlighted in **bold**:

```
name: Contract Options
variables:
    agent fees: 13,000,000
type: decision
Sign with Dodgers:
    cost: agent fees
    type: chance
    Stays Healthy:
        p: 0.75
    Does not stay healthy:
        p: 0.25
Sign with Yankees:
    payoff: 120,000,000
```

References the previously defined variable agent fees as a cost

Specifies that the node type is a chance node (not a decision node)

Defines a branch called Stays Healthy

Assigns a probability of 0.75 (or 75%) to Stays Healthy

Defines a branch called Does not stay healthy

Assigns a probability of 0.25 (or 25%) to Does not stay healthy. Note that the probabilities of Stays Healthy and Does not stay healthy sum to 1.

The expected value of the `Sign with Dodgers` decision will be affected by the variable `agent fees` and whether the player can maintain his health over the lifetime of the contract. The schema, therefore, now includes agent fees, which are a fixed cost, and introduces a chance node with probabilities for `Stays Healthy` at 0.75 and `Does not stay healthy` at 0.25. Probabilities from each chance node must sum to exactly 1. These probabilities will play a crucial role in determining the overall expected value of the Dodgers' offer.

We complete the schema by creating additional chance nodes and defining the probabilities and payoffs for each potential outcome. This includes detailing the likelihood and financial results for various performance incentives based on the player's health status.

Once again, changes to the schema are highlighted in **bold**:

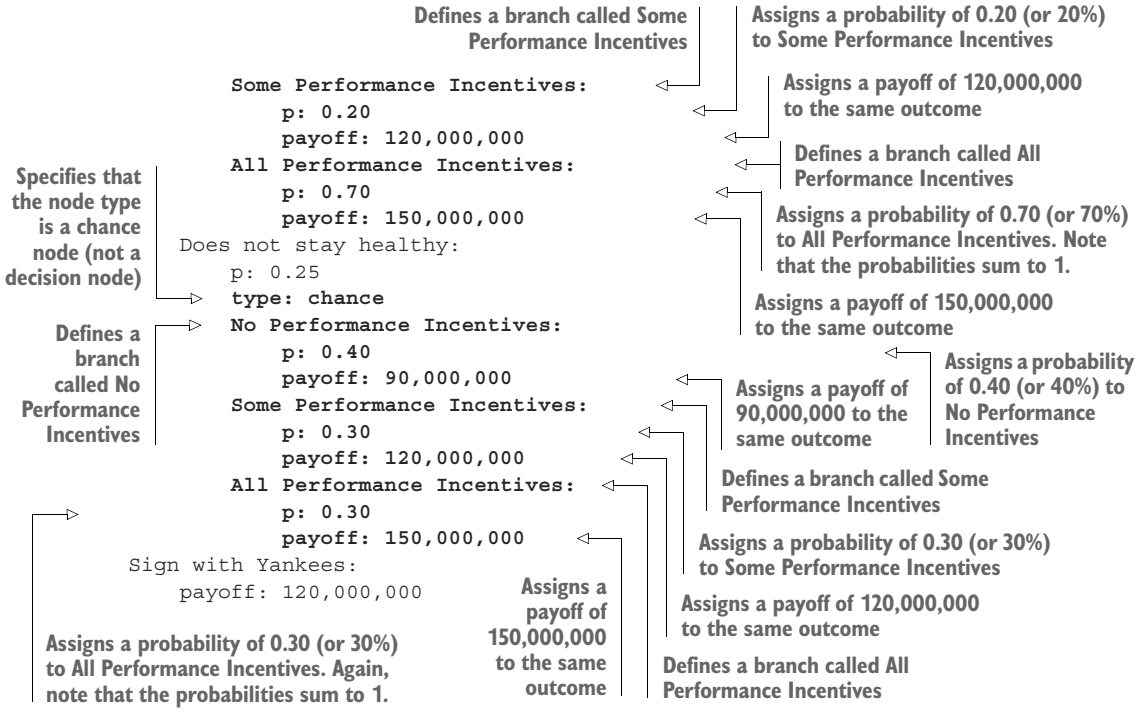
```
name: Contract Options
variables:
    agent fees: 13,000,000
type: decision
Sign with Dodgers:
    cost: agent fees
    type: chance
    Stays Healthy:
        p: 0.75
        type: chance
        No Performance Incentives:
            p: 0.10
            payoff: 90,000,000
```

Specifies that the node type is a chance node (not a decision node)

Defines a branch called No Performance Incentives

Assigns a probability of 0.10 (or 10%) to No Performance Incentives

Assigns a payoff of 90,000,000 to the same outcome



We have finalized the decision tree schema by adding detailed chance nodes and defining the probabilities and payoffs for each potential outcome. For the `Sign with Dodgers` decision, we have introduced a chance node to account for the player's health status. If the player stays healthy (p: 0.75), there are further chance nodes detailing the likelihood of meeting performance incentives: `No Performance Incentives` (p: 0.10) with a payoff of $90,000,000, `Some Performance Incentives` (p: 0.20) with a payoff of $120,000,000, and `All Performance Incentives` (p: 0.70) with a payoff of $150,000,000. These probabilities are illustrative but loosely informed by historical player performance trends and general expectations based on age and durability.

Similarly, if the player does not stay healthy (p: 0.25), there are also chance nodes with probabilities and payoffs: `No Performance Incentives` (p: 0.40) with a payoff of $90,000,000, `Some Performance Incentives` (p: 0.30) with a payoff of $120,000,000, and `All Performance Incentives` (p: 0.30) with a payoff of $150,000,000. The decision to sign with the Yankees offers a straightforward guaranteed payoff of $120,000,000. This comprehensive schema will serve as the blueprint for constructing the decision tree, ensuring that all possible outcomes, their probabilities, and associated payoffs are systematically represented.

CREATING THE SCHEMA PROGRAMMATICALLY

Now that we have detailed the final schema manually, we will transition to demonstrating how to build the same schema using methods from Python's `anytree` library. This programmatic approach will streamline the process and ensure accuracy in representing the decision tree structure. We will use this schema as the source for our

Matplotlib decision tree, illustrating how to show the decision-making process effectively using standard Python libraries and methods.

Even though we are about to create, or re-create, the schema programmatically, the steps remain just the same as before. Thus, we begin by naming the decision tree, defining any variables that will affect the expected value calculations, and establishing the decision node:

```
>>> from anytree import Node
>>> root = Node('Contract Options')
>>> agent_fees = 13000000
>>> dodgers = Node('Sign with Dodgers',
>>>                parent = root)
>>> yankees = Node('Sign with Yankees\n'
>>>                'Expected Payoff: $120,000,000',
>>>                parent = root,
>>>                ´payoff = 120000000)
```

Imports the Node class from the anytree library

Creates the root, or decision, node with the name Contract Options

Assigns the value 13000000 to the variable agent_fees, which represents a fixed cost or fee

Creates a child node labeled Sign with Dodgers under the root node

Creates a child node labeled Sign with Yankees under the root node with an expected value equal to $120,000,000

With just a few lines of Python code, we have established the foundation for our decision tree schema.

Next, we establish a chance node to account for the player's health status, which significantly affects the expected value calculations. The code creates two child nodes under the `Sign with Dodgers` decision node: `Stays Healthy` with a probability of `0.75`, and `Does not stay healthy` with a probability of `0.25`. These chance nodes represent the potential health outcomes and their associated probabilities, providing a detailed view of the decision tree's branching based on the player's health:

```
>>> stays_healthy = Node('Stays Healthy\np=0.75',
>>>                      parent = dodgers)
>>> does_not_stay_healthy = Node('Does not stay healthy\np=0.25',
>>>                              parent = dodgers)
```

Creates the first of two chance nodes under the Sign with Dodgers node

Creates the second of two chance nodes under the Sign with Dodgers node

In this part of the decision tree schema, we introduce chance nodes to account for the player's health status. These nodes, labeled `Stays Healthy` with a `0.75` probability and `Does not stay healthy` with a `0.25` probability, are specified as branches under the `Sign with Dodgers` decision node. This structure models the uncertainties related to the player's health over the contract period.

Even though we're building the schema and not constructing the tree, Python distinguishes between chance nodes and decision nodes through their context and labeling within the schema. Decision nodes, such as `Sign with Dodgers` and `Sign with Yankees`, represent choices or actions. In contrast, chance nodes represent probabilistic events and include probabilities, like `Stays Healthy` with a 75% chance. The hierarchical arrangement and clear labeling of each node within the schema help clarify

their roles and outcomes, ensuring the decision tree accurately reflects the complexities of real-world decision-making scenarios when it's eventually constructed.

The next step in building out the schema involves defining the subtrees under the `Stays Healthy` and `Does not stay healthy` chance nodes. A *subtree* is a smaller section of the decision tree that branches out from a specific node, representing a subset of possible outcomes and decisions:

```
>>> no_incentives = Node('None\np=0.10\npayoff=90M',
>>>                      parent = stays_healthy,
>>>                      payoff = 90000000)
>>> some_incentives = Node('Some\np=0.20\npayoff=120M',
>>>                        parent = stays_healthy,
>>>                        payoff = 120000000)
>>> all_incentives = Node('All\np=0.70\npayoff=150M',
>>>                       parent = stays_healthy,
>>>                       payoff = 150000000)

>>> no_incentives_dodgers = Node('None\np=0.40\npayoff=90M',
>>>                         parent = does_not_stay_healthy,
>>>                         payoff = 90000000)
>>> some_incentives_dodgers = Node('Some\np=0.30\npayoff=120M',
>>>                           parent = does_not_stay_healthy,
>>>                           payoff = 120000000)
>>> all_incentives_dodgers = Node('All\np=0.30\npayoff=150M',
>>>                          parent = does_not_stay_healthy,
>>>                          payoff = 150000000)
>>>
```

Creates the first of three child nodes under the Stays Healthy chance node, with an associated probability of 0.10 and a payoff of $90 million

Creates the second of three child nodes under the Stays Healthy chance node, with an associated probability of 0.20 and a payoff of $120 million

Creates the third of three child nodes under the Stays Healthy chance node, with an associated probability of 0.70 and a payoff of $150 million

Creates the first of three child nodes under the Does not stay healthy chance node, with an associated probability of 0.40 and a payoff of $90 million

Creates the second of three child nodes under the Does not stay healthy chance node, with an associated probability of 0.30 and a payoff of $120 million

Creates the third of three child nodes under the Does not stay healthy chance node, with an associated probability of 0.30 and a payoff of $150 million

The code defines the subtrees for the `Stays Healthy` and `Does not stay healthy` chance nodes, detailing the various performance incentive outcomes and their associated probabilities and payoffs. For both health scenarios, the performance thresholds—`No Performance Incentives`, `Some Performance Incentives`, and `All Performance Incentives`—have consistent payoffs of `90000000`, `120000000`, and `150000000`, respectively. However, the probabilities of these outcomes differ based on the player's health status. If the player stays healthy, there is a higher probability of meeting all performance incentives (70%) compared to when the player does not stay healthy (30%). Conversely, the likelihood of not meeting any performance incentives is higher if the player does not stay healthy (40%) than if he remains healthy (10%). This structure captures the effect of the player's health on the expected value of the Dodgers' contract.

The next step in building out the schema programmatically involves calculating the expected payoffs for the various scenarios and updating the nodes with these values. This process begins by defining variables for the expected payoffs when the

player stays healthy and when he does not. More precisely, the code derives `stays_healthy_payoff` by weighting the payoffs of no, some, and all performance incentives by their respective probabilities. Similarly, `does_not_stay_healthy_payoff` is derived using the probabilities for the same performance thresholds under the scenario where the player does not stay healthy. Finally, these calculated expected payoffs are appended to the corresponding node labels, enhancing the decision tree with quantitative insights that clearly indicate the financial outcomes of each scenario. This comprehensive calculation step ensures that all potential outcomes are accurately represented and quantified in the decision tree:

```
>>> stays_healthy_payoff = (0.10 * 90000000 +
>>>                         0.20 * 120000000 +
>>>                         0.70 * 150000000)
>>> does_not_stay_healthy_payoff = (0.40 * 90000000 +
>>>                                 0.30 * 120000000 +
>>>                                 0.30 * 150000000)

>>> dodgers_expected_payoff = (0.75 * stays_healthy_payoff +
>>>                            0.25 * does_not_stay_healthy_payoff -
>>>                            agent_fees)

>>> dodgers.name += (f'\nExpected Payoff: \
>>>        f"${dodgers_expected_payoff:,.0f}')

>>> stays_healthy.name += (f'\nExpected Payoff: '
>>>        f'${stays_healthy_payoff:,.0f}')
>>> does_not_stay_healthy.name += (f'\nExpected Payoff: '
>>>        f'${does_not_stay_healthy_payoff:,.0f}')
```

Calculates the expected value of the Stays Healthy chance node by plugging probabilities and payoffs into the expected value formula

Calculates the expected value of the Does not stay healthy chance node by again plugging probabilities and payoffs into the expected value formula

Calculates the expected value of the Dodgers' contract by using the expected value formula, combining the weighted payoffs of the chance nodes, and subtracting the agent fees

Appends the expected value of the Dodgers' contract to the label of the dodgers decision node

Appends the expected value to the Stays Healthy chance node

Appends the expected value to the Does not stay healthy chance node

The last snippet of code calculates the expected values for each chance node in the decision tree based on the defined schema. It first determines the expected payoffs for the `Stays Healthy` and `Does not stay healthy` nodes by weighting each potential outcome by its probability. Then, it calculates the overall expected payoff for signing with the Dodgers by combining these values and subtracting the agent fees. The results are, of course, not immediately applied to a visual representation; rather, they are stored in memory, thereby preparing the updated information to be included in the decision tree when it is finally constructed and displayed.

10.2.2 Plotting the tree

The following snippet of Matplotlib code generates a visualization of the decision tree we have designed programmatically. The `plot_tree()` function takes the root node of the tree as an input and generates a horizontal diagram that visually represents the decision-making process.

The `add_edges()` method within `plot_tree()` recursively traverses the tree, positioning each node at specific coordinates to ensure that the nodes are correctly spaced horizontally and vertically. It starts from the root node and moves through each layer of the tree, adjusting the positions of child nodes relative to their parent nodes. This ensures that the tree structure is maintained and visually clear.

After determining the positions of all nodes, the function then uses `plt.text()` to place labels on each node. It distinguishes between different types of nodes by their content, using different background colors for decision nodes and chance nodes. Decision nodes are highlighted in green, and chance nodes are highlighted in orange. This color-coding helps to visually differentiate the various elements of the tree.

Finally, the function sets a title for the plot, removes the axis for a cleaner look, and displays the completed decision tree using `plt.show()`. This visualization step transforms the stored node information into a clear, interpretable diagram that highlights the decision paths and expected outcomes:

```
>>> import matplotlib.pyplot as plt
>>> def plot_tree(root):
>>>     def add_edges(node, pos = {}, x = 0, y = 0,
>>>                   layer = 1, dy = 3.0):
>>>         pos[node] = (x, y)
>>>         neighbors = node.children
>>>         if neighbors:
>>>             next_x = x + 2.
>>>             next_layer = layer + 1
>>>             for i, neighbor in enumerate(neighbors):
>>>                 next_y = y + dy * (i - \
>>>                          (len(neighbors) - 1) / 2)
>>>                 pos = add_edges(neighbor, pos = pos,
>>>                          x = next_x, y = next_y,
>>>                          layer = next_layer,
>>>                          dy = dy / 1.5)
>>>                 plt.plot([x, next_x],
>>>                    [y, next_y], 'k-')
>>>         return pos
>>>     pos = add_edges(root, dy = 12.0)
>>>     for node, (x, y) in pos.items():
>>>         if node.is_leaf:
>>>             plt.text(x, y, node.name,
>>>                      va = 'center')
>>>         else:
>>>             if ('Contract Options' in node.name
>>>                     or 'Yankees' in node.name):
>>>                 plt.text(x, y, node.name,
>>>                          va = 'center',
>>>                          bbox = dict(facecolor = 'green',
>>>                            edgecolor = 'black',
>>>                            boxstyle = \
>>>                              'round, pad = 0.5'))
```

Imports the matplotlib library

Stores the position of the root, or decision, node

Adjusts horizontal spacing to keep adjacent nodes from overlapping

Adjusts vertical spacing for the same reason

Recursively adds child nodes

Creates a visual connection (line) between a parent node and its child nodes

Starts the process of assigning positions to each node in the tree, beginning with the root node

Adds labels or names to the leaf nodes (nodes with no children)

Applies visual formatting to the decision node to distinguish it from chance nodes

```
>>>             else:
>>>                 plt.text(x, y, node.name, va = 'center',
>>>                          bbox = dict(facecolor = 'orange',
>>>                            edgecolor = 'black',
>>>                            boxstyle = \
>>>                              'round, pad = 0.5'))

>>>     plt.title('Decision Tree: Contract Options\n\n')
>>>     plt.axis('off')
>>>     plt.show()

>>> plot_tree(root)
```

Applies visual formatting to the chance nodes to distinguish them from the decision node

Removes axis lines and labels for a cleaner, more focused visualization of the decision tree

Displays the plot

Creates the visual representation of the decision tree as defined by the plot_tree(root) function

Figure 10.1 shows the decision tree visualizing the contract options for a Major League Baseball player considering offers from the New York Yankees and the Los Angeles Dodgers. The tree outlines the possible outcomes and expected payoffs based on performance incentives and the player's health.

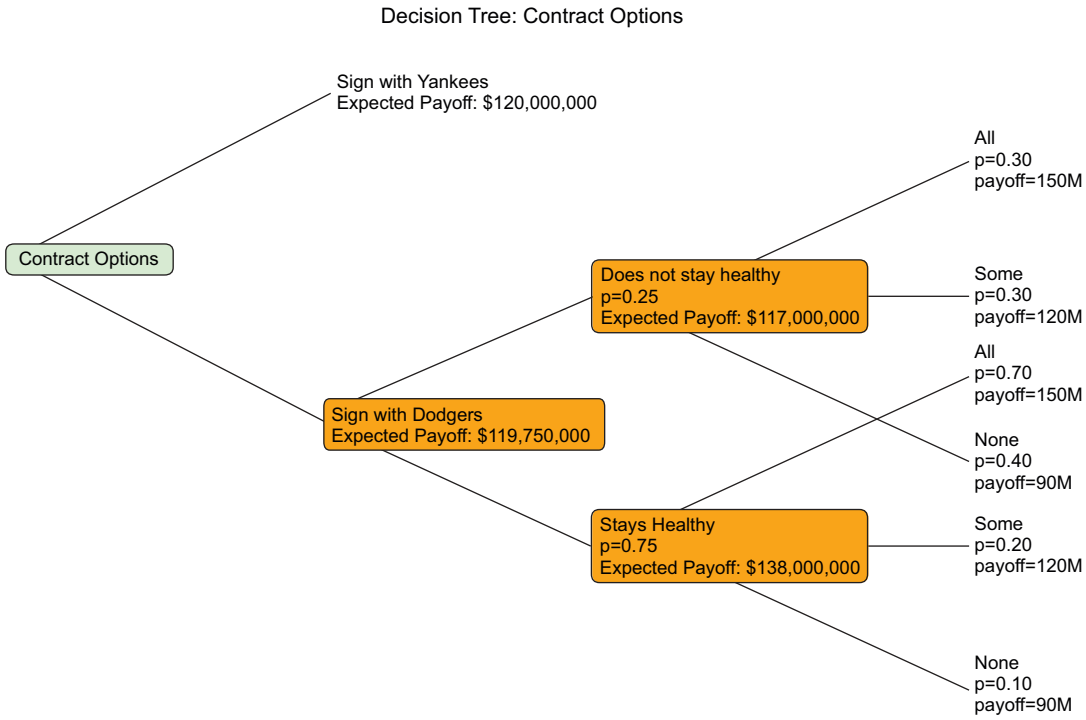


Figure 10.1 A decision tree that illustrates two very different contract offers. The tree outlines the expected payoffs based on different states of nature, allowing the player to accept the contract with the highest expected value.

Some best practices are evident in the diagram, such as using distinct colors to differentiate the decision nodes from chance nodes. Additionally, the use of clear, descriptive labels for each node, including probabilities and expected payoffs, enhances the interpretability of the tree. Ensuring that decision nodes and chance nodes have different shapes or sizes can further improve clarity; for example, chance nodes could be larger or differently shaped than decision nodes.

The tree should be evaluated or interpreted from right to left, despite being constructed from left to right. This reverse interpretation ensures that all probabilities and payoffs are considered cumulatively. Starting from the rightmost nodes, we see the outcomes based on performance incentives for the `Stays Healthy` and `Does not stay healthy` scenarios. Each of these chance nodes has associated probabilities and payoffs, which are combined to determine the expected value for each health scenario.

For instance, the `Stays Healthy` node has an expected payoff calculated from its branches: 10% for $90 million, 20% for $120 million, and 70% for $150 million, resulting in an expected payoff of $138 million. Similarly, the `Does not stay healthy` node has an expected payoff calculated from its branches: 40% for $90 million, 30% for $120 million, and 30% for $150 million, resulting in an expected payoff of $117 million. These expected values are then used to calculate the overall expected payoff for signing with the Dodgers, factoring in the probabilities of staying healthy (75%) and not staying healthy (25%), and subtracting agent fees.

Interpreting the tree, we compare the overall expected payoffs of the decision nodes. The `Sign with Yankees` node has a guaranteed expected payoff of $120 million, whereas the `Sign with Dodgers` node has an overall expected payoff of approximately $119.75 million. Given these calculations, the best decision for the player is to sign with the Yankees, as it offers the highest expected value. This structured approach allows for a clear, data-driven decision-making process, highlighting the most financially beneficial option based on the probabilities and outcomes provided.

In this chapter, we learned how to design and build a decision tree based on the Expected Value method, which combines payoffs and probabilities to make informed decisions. We explored the process of creating a decision tree schema, calculating expected values, and visually representing decision paths and outcomes. Additionally, we discussed other decision-making methods that do not rely on probabilities, such as the Maximax, Maximin, and Minimax Regret methods, highlighting different approaches to evaluating choices and managing uncertainty.

In the next chapter, we will explore mathematical systems that model transitions between different states over time, where each transition depends only on the current state and not on the previous history. These models are useful for predicting future states and understanding dynamic processes in various fields.

Summary

- The Maximax method is an optimistic decision-making approach that focuses on maximizing the maximum possible payoff. Decision-makers using this method choose the option with the highest potential reward, reflecting a high tolerance for risk and a preference for best-case scenarios.

- The Maximin method is a conservative decision-making approach that aims to maximize the minimum payoff. It involves selecting the option with the best worst-case scenario, making it suitable for risk-averse decision-makers who prioritize minimizing potential losses.
- The Minimax Regret method aims to minimize the maximum regret, which is the difference between the actual payoff and the best possible payoff that could have been achieved with perfect foresight. Decision-makers using this method focus on reducing the potential for regret by selecting the option that offers the least unfavorable outcome compared to the best possible alternative.
- The Expected Value method is a decision-making approach that calculates the average outcome by weighting each possible scenario by its probability. Decision-makers use this method to choose the option with the highest expected return or lowest expected loss, providing a balanced and rational evaluation of all potential outcomes and their likelihoods.
- Decision trees are graphical representations of decision-making processes that outline various choices, chance events, probabilities, and payoffs. They provide a clear, visual method for evaluating complex decision scenarios, including sequential decisions, by incorporating probabilities and calculating expected values at each stage.
- Creating the schema for a decision tree involves defining the decision nodes, chance nodes, probabilities, and payoffs. This structured blueprint ensures all possible outcomes are systematically considered, laying the groundwork for constructing the decision tree.
- Constructing the decision tree involves translating the schema into a visual format, typically using nodes and branches to represent decisions and chance events. This step allows for easy interpretation and analysis of the decision-making process, helping to show the potential paths and outcomes clearly.
- Real-world applications of these decision-making methods extend beyond theoretical exercises. Businesses use them to evaluate investment strategies, optimize resource allocation, assess project risks, and make strategic planning decisions in uncertain environments. By applying these frameworks, decision-makers can systematically analyze complex scenarios and improve the quality of their choices.