WSABI Summer Research Lab **Summer 2024**

Lecture 16: Decision Trees

Instructor: Ryan Brill *Scribe: Jonathan Pipping*

16.1 Estimating In-Game NFL Win Probabilities

16.1.1 Motivation

We want to estimate the win probability of an NFL game as a function of game state. Why might this be important?

- Pricing (or beating) betting markets
- Valuing players (think WPA)
- Strategic decision-making: making the decision which maximizes win probability (see [BYW])

16.1.2 Existing Work

Early approaches to modeling in-game win probability in the NFL relied on mathematical frameworks like dynamic programming and state-space modeling, with some of the first models appearing about 25 years ago.

Since then, statistical models have become the dominant method. These models ask the question: in "similar" situations across football history, what proportion of the time did the team in possession win the game? With the increased accessibility of play-by-play data, machine learning models can be fit pretty quickly, and so this is the most common approach. We will focus on these models in this lecture.

16.1.3 Proposing a Model

We want to estimate in-game win probability $\text{WP}(x_i)$ as a function of game-state x_i. Each row in our dataset represents a play i, with each play including the following variables:

- yard_line(i): the distance to the goal line at the start of play i
- down(i): the down play i took place on
- distance(i): the distance to the first down marker at the start of play i
- score_diff(i): the score difference between the team on offense and the team on defense
- time(i): the seconds remaining in the game at the start of play i
- team_quality(i): the difference in quality between the offensive and defensive team (we use the point spread)
- receiving_2nd_half_kick(i): is the team on offense receiving the 2nd half kickoff?

- timeouts(i): the number of timeouts remaining for each team at the start of play i

Our outcome variable y_i represents whether the team on offense won (1) or lost (0) the game. Seeing as we have a binary outcome, we propose a **Logistic Regression** model like we learned about in Lecture 4. If win probability is expressed as

$$\text{WP}(x_i) = \mathbb{P}(y_i = 1 | x_i)$$

Then we can model the log-odds of winning through a logistic regression model:

$$\log\left(\frac{\mathbb{P}\left(y_i = 1 \mid x_i\right)}{\mathbb{P}\left(y_i = 0 \mid x_i\right)}\right) = g(x_i)$$

Then if we let

$$g(x_i) = \beta_1 \cdot \text{yard\_line}(i) + \beta_2 \cdot \text{down}(i) + \beta_3 \cdot \text{distance}(i) + \ldots$$

where β_1 is a coefficient vector for a spline function on yard line, β_2 are categorical coefficients for down, β_3 are categorical coefficients for distance, and so on, then we can estimate the win probability as

$$\text{WP}(x_i) = \mathbb{P}\left(y_i = 1 \mid x_i\right) = \frac{1}{1 + e^{-g(x_i)}}$$

What's wrong with this approach?

Logistic regression assumes **additive** relationships between variables, but these game-state variables **interact** with each other in complicated ways and have complex **non-linearities**. For example, score differential and time remaining certainly interact strongly with each other.

We can generally model interactions in additive regression settings by including interaction terms. This is easy with binary variables, but can be extremely difficult with 2 continuous variables with a non-linear relationship. For example, if we wanted to model the interaction between score differential and time remaining, we would need something like:

$$\beta_1 \cdot \text{score\_diff}(i) + \beta_2 \cdot \text{time}(i) + \beta_3 \cdot (\text{score\_diff}\,(i) \cdot \text{time}\,(i))$$

$$\text{or } + \beta_3 \cdot \frac{\text{score\_diff}\,(i)}{\sqrt{1 + \text{time}\,(i)}}$$

but with each of these being a spline function, the number of parameters we'd need for just two variables is significant. Once we start adding even more interacting and non-linear variables, the number of parameters we would need for a logistic regression model becomes prohibitively large.

We must consider a different approach, one that can handle arbitrarily-complex relationships between the variables from the data. This is where **machine learning models** comes in: we will focus on **tree-based models** in this lecture, which are easy to use in `R` and are widely used for NFL win probability modeling.

16.2 Decision Trees*

16.2.1 Class Cricket Example

Consider a class with 20 students, 10 of which play cricket and 10 of which do not. We want to predict whether a student plays cricket based on some things we know about them: their height, their grades, and their class.

*Based on content from Analytics Vidhya on YouTube.

Idea: What if we split our data on an X variable to classify the y variable?

We visualize splitting on each of the three variables in Figure 16.1. Note that a red cross indicates a student who plays cricket, and a green rectangle indicates a student who does not.

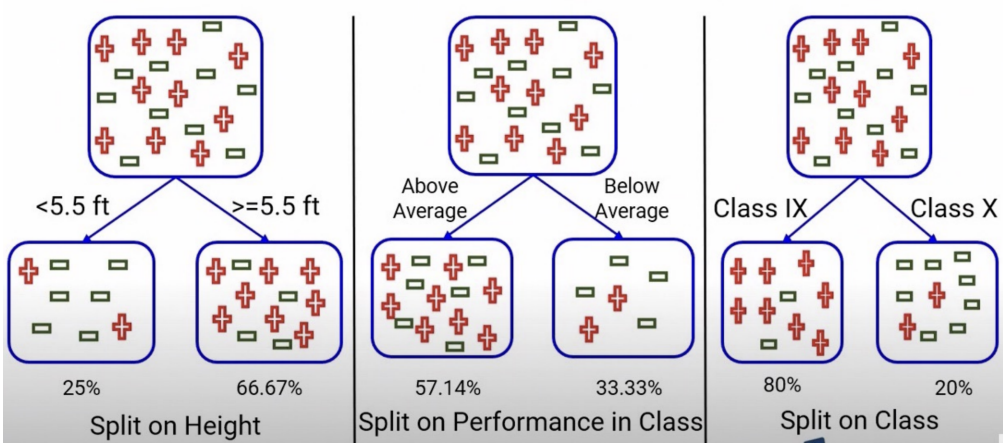


Figure 16.1: Splitting on each of the three variables. A red cross indicates a student who plays cricket, and a green rectangle indicates a student who does not.

Which of these splits is the best? Clearly the rightmost one, as it separates the two classes of students as best as possible. Note that we can have multiple splits in a tree like this, which we visualize in Figure 16.2.

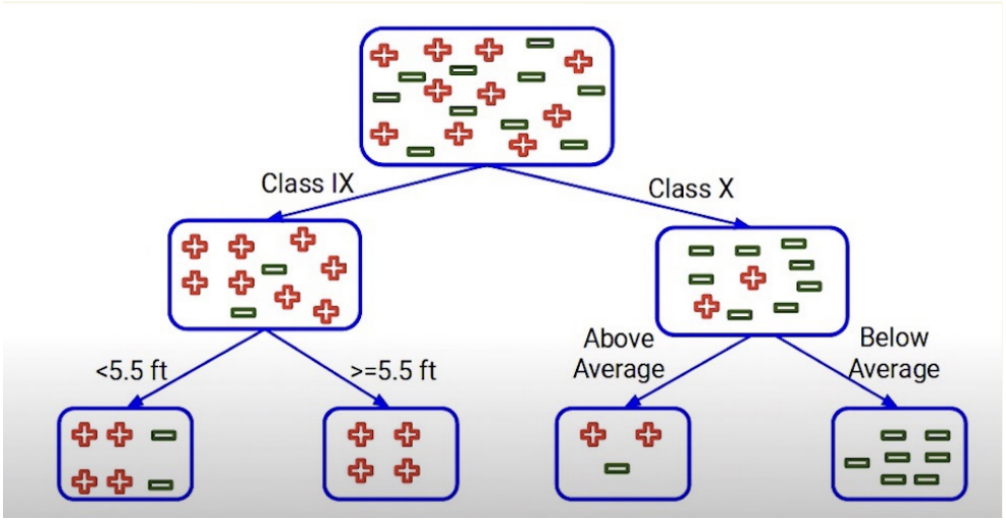


Figure 16.2: A full tree split on each of the three variables. A red cross indicates a student who plays cricket, and a green rectangle indicates a student who does not.

These multiple splits allow variables to interact with each other to classify the y variable better. But how do we fit such a tree from data? How do we select the best split point?

The Answer: Select the split which results in the most homogenous sub-nodes . We'll look into how to quantify this in the next sections.

16.2.2 Gini Impurity

One way to calculate the homogeneity of a sub-node is to calculate the **Gini Impurity**. Within each sub-node, we calculate the Gini Impurity to see how homogeneous the sub-node is.

Definition 16.1 (Gini Impurity of a Sub-Node)**.** *The Gini Impurity of a sub-node i is given by*

$$Gini_i = 1 - \sum_{k=1}^{K} p_k^2$$

where p_k is the proportion of the sub-node that belongs to class k.

Then, we can calculate the **Gini Impurity of a split** by a weighted average of the sub-node Gini Impurities.

Definition 16.2 (Gini Impurity of a Split)**.** *The Gini Impurity of a split I is given by*

$$Gini_I = \sum_{i=1}^{I} \frac{n_i}{N} Gini(i)$$

where n_i is the number of observations in sub-node i and N is the total number of observations.

We calculate the Gini Impurity of a split on Class Performance in Figure 16.3.

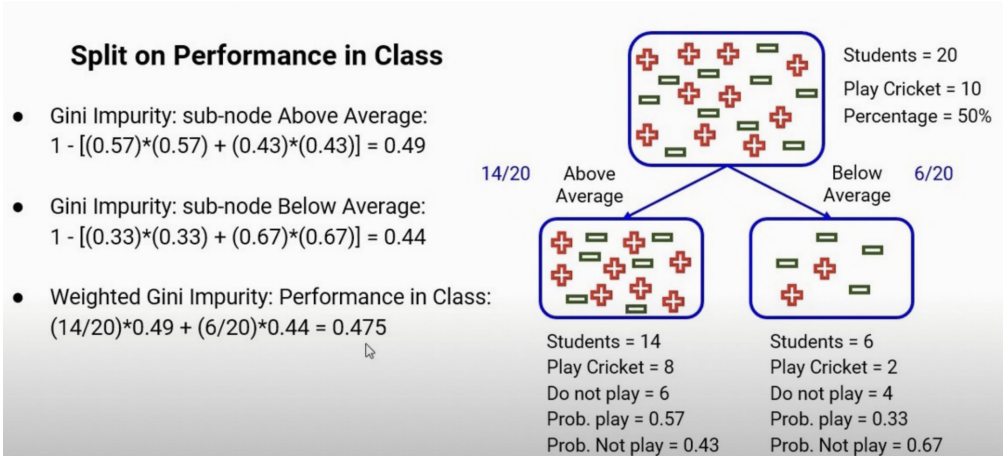


Figure 16.3: Gini Impurity of a split on Class Performance.

How do we use Gini Impurity to find the best split? We calculate the Gini Impurity of a split for each variable and select the variable with the lowest Gini Impurity. Based on Figure 16.4, we see that we should split on Class Performance. We can continue this process recursively to build a full tree.

Split	Weighted Gini Impurity
Performance in Class	0.475
Class	0.32

Figure 16.4: Gini Impurity of a split on each variable. Clearly, we should split on Class first.

16.2.3 Information Gain

Another way to calculate sub-node homogeneity is to calculate the **Information Gain**. Intuitively, we would expect that a split resulting in more homogenous sub-nodes would have a higher Information Gain, as shown in Figure 16.5. We go on to define Information Gain formally below.

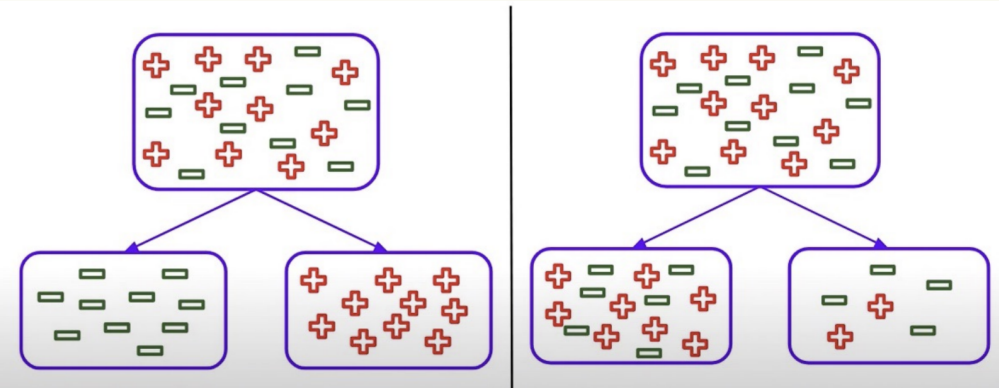


Figure 16.5: Two splits of the same data. The left split has more homogenous sub-nodes, so the information gain is higher.

Definition 16.3 (Information Gain)**.** *The Information Gain of a sub-node i is given by*

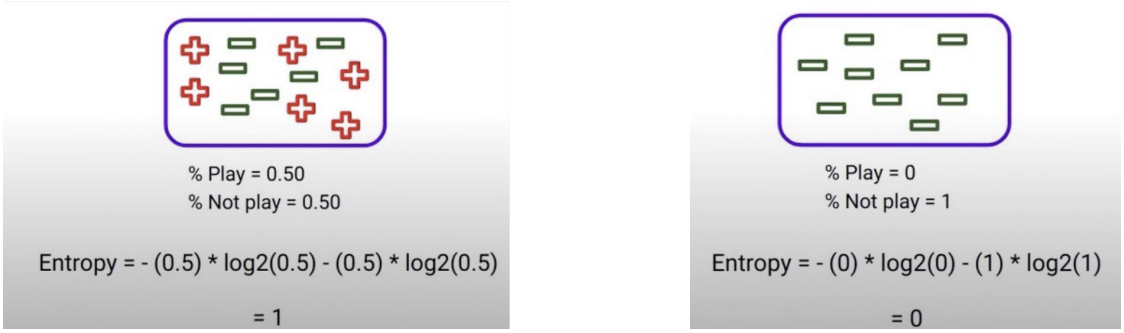
$$IG_i = 1 - Entropy_i$$

where

$$Entropy_i = -\sum_{k=1}^{K} p_k \log_2(p_k)$$

and p_k is the proportion of the sub-node that belongs to class k.

We calculate the entropy of two sub-nodes in Figure 16.6. Note that the maximum entropy case is when the sub-nodes have an equal proportion of each class, and the minimum entropy case is when the sub-node has all of one class.



(a) Minimum entropy case (b) Maximum entropy case

Figure 16.6: Entropy calculation showing minimum and maximum entropy cases.

Similarly to Gini Impurity, the entropy of a split is the weighted average of the entropy of the sub-nodes.

Definition 16.4 (Entropy of a Split)**.** *The entropy of a split I is given by*

$$Entropy_I = \sum_{i=1}^{I} \frac{n_i}{N} Entropy(i)$$

where n_i is the number of observations in sub-node i and N is the total number of observations.

We showcase this in Figure 16.7, then compare the Information Gain of the two splits in Figure 16.8 to determine which split is better. Like with Gini Impurity, we continue this process recursively to build a full tree.

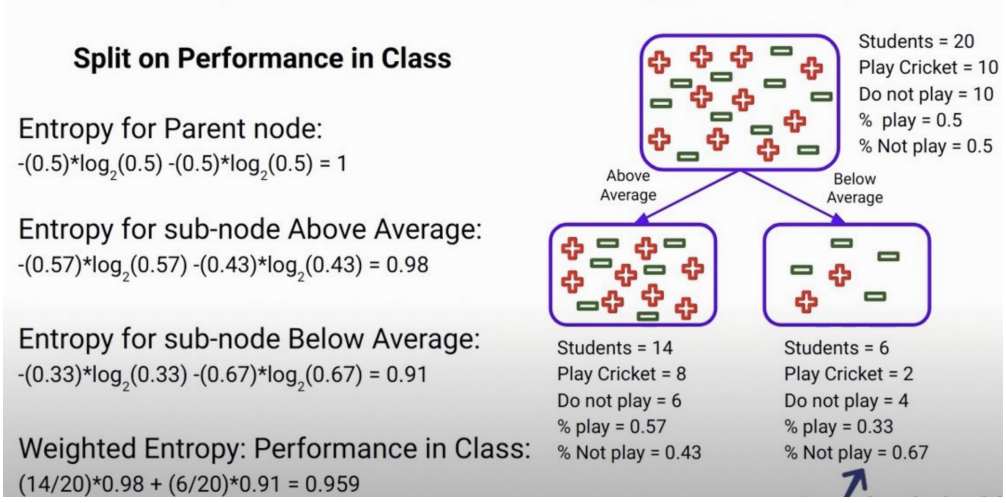


Figure 16.7: Entropy of a split on Class Performance.

Split	Entropy	Information Gain
Performance in Class	0.959	0.041
Class	0.722	0.278

Figure 16.8: Information Gain of a split on each variable. Clearly, we should split on Class first.

16.2.4 Reduction in Variance

If our outcome variable is categorical, Gini Impurity and Information Gain work well. However, if our outcome variable is continuous, we need another way to measure splits: **Reduction in Variance**. We define the variance of a sub-node i as follows:

Definition 16.5 (Variance of a Sub-Node)**.** *The variance of a sub-node i is given by*

$$Var_i = \frac{1}{n_i} \sum_{j=1}^{n_i} (y_j - \bar{y})^2$$

where y_j is the outcome variable for the j^{th} observation in sub-node i, $\bar{y}$ is the mean of the outcome variable in sub-node i, and n_i is the number of observations in sub-node i. We showcase this calculation in Figure 16.9.

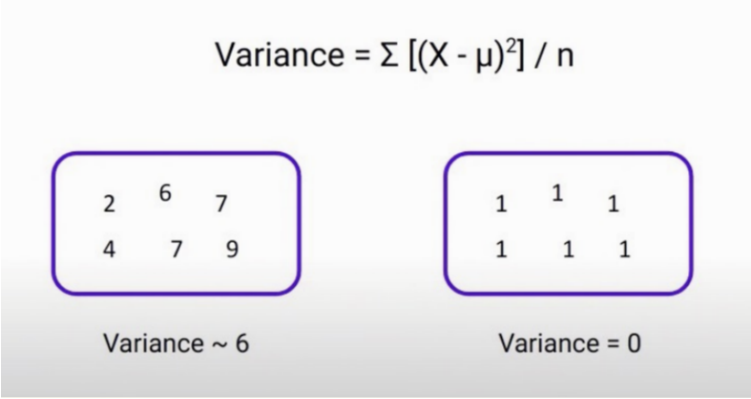


Figure 16.9: Variance calculation for two example sub-nodes.

We proceed to calculate the variance of a split as the weighted average of the variance of the sub-nodes. When fitting a tree, we aim to minimize the variance of these splits.

Definition 16.6 (Variance of a Split)**.** *The variance of a split I is given by*

$$Var_I = \sum_{i=1}^{I} \frac{n_i}{N} Var_i$$

where n_i is the number of observations in sub-node i and N is the total number of observations.

We calculate the variance of a split on Class Performance in Figure 16.10, then compare the variance of two possible splits in Figure 16.11. We see that the split on Class has lower variance, so we should split on Class first, then continue recursively to build a full tree.

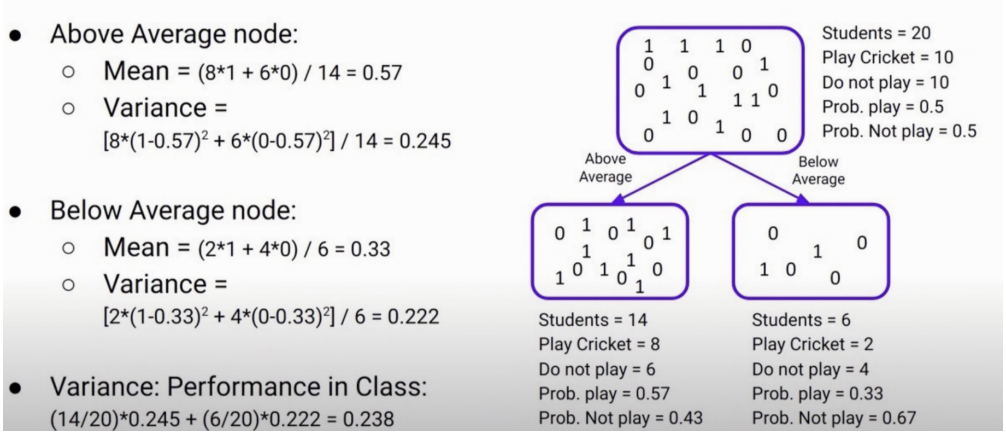


Figure 16.10: Variance of a split on Class Performance.

Split	Variance
Performance in Class	0.238
Class	0.16

Figure 16.11: Variance of a split on each variable. Clearly, we should split on Class first.

16.2.5 Pruning

Now that we know how to select the best split points, we could reasonably continue making splits until all nodes are "pure", meaning that each node contains only one class. However, this would **overfit** heavily, memorizing the noise in training data and performing poorly on new data. To prevent overfitting, we **prune** the tree, meaning adding at least one condition to reduce the tree's complexity:

- **Max Tree Depth**: Prevent the tree from growing too deep.
- **Minimum Samples for Node Split**: Prevent splitting nodes with too few observations.
- **Minimum Samples for Terminal Node**: Prevent terminal nodes with too few observations.
- **Maximum Number of Terminal Nodes**: Prevent the tree from splitting into too many terminal nodes.

These are fairly simple conditions, so fitting a tree is relatively straightforward in `R`. The code to do this is shown below.

```
library(rpart)
# fit tree to training data
tree = rpart(y ~ ., data = training_data,
             # "class" for categorical outcome, "anova" for continuous
             method = "class",
             # if categorical, specify "gini" or "information"
             parms = list(split = "gini"),
             # select pruning conditions
             control = rpart.control(.))
# visualize tree
rpart.plot(tree)
```

16.2.6 Takeaways

- Decision trees are a powerful tool for modeling complex relationships between variables, and they are quite easy to fit! However, they can be quite sensitive to the data and prone to overfitting.
- We can mitigate overfitting by pruning our tree, but it doesn't completely solve the problem. Tomorrow, we will consider **Random Forests**, a more robust alternative to decision trees.

References

[BYW] Brill, R. S., Yurko, R., & Wyner, A. J., *Analytics, Have Some Humility: A Statistical View of Fourth-Down Decision Making*, The American Statistician, 2025.