#h1 Chapter 16 - Decision Trees

#h2 The Agony of Choice

#h3 16.1 – Introduction

#pg A decision tree is a classification algorithm where a hierarchical sequence of decisions will determine the predicted output of the algorithm. Each decision operates on a single input variable, comparing the value of that input variable to the pretrained split value. This comparison will direct the decision tree algorithm into one of 2 (or more) branches, where a different decision rule is applied, and so on. Any input will traverse the tree along some path to a terminating node associated with an outcome value or label. The decision rules are learned (estimated) during training. This branching into different decision paths from a common starting point rightfully earns the algorithm the name decision tree, as the paths resemble the branch (or roots) or a tree. For example, in a decision tree predicting the risk of a specific cancer with a hereditary component, the first decision rule can be whether the patient has a family history of the cancer. If so, we traverse down to one of the two children of the current node. If not, we traverse the other until we exhaust all the rules and end up with a decision or prediction. In decision trees, we refer to these rules as splits. Building the tree thus means determining the optimal splits of our data and in what order.

#im ../assets/figures/016/016-01.png 50 192 Figure 16.1 - Choose your own data science adventure decision tree.

#h3 16.2 - Pros and cons of Decision Trees

#pg We traditionally use decision trees as machine learning algorithms best applied to categorical data. This categorical-only assumption is somewhat of a misperception, as the metrics used by the algorithms are equally applicable to both categorical and numerical inputs. However, decision trees are primarily used as classifiers. Getting a decision tree to do regression is possible in principle, but numerical methods offer more efficient solutions. The strength of decision trees is threefold. First, decision models are quite intuitive and transparent. We see how the algorithm makes the decisions and in what order. Reminiscent of early knowledge-based systems with their cascade of [if…then] rules, the decision tree gives us an explanatory path from input to decision. Second, unlike some of the numeral classifiers we have already discussed (like logistic regression), the decision tree has an implicit ranking of the regressors in terms of importance. We first consider important variables for making the correct decision, while noisy, non-predictive regressors end up further down the tree if they make it at all (see the section on pruning). Third, the entire algorithm is usually completely non-parametric. In principle, there is no need to set learning rate, batch size, regularization terms, or other parameters that require tuning to find the optimal result (however, see the section on pruning). However, decision trees can be computationally expensive if there are many regressors. Although the number of data points considered by subsequent nodes decreases at each split, each node still needs to consider all possible splits over all the regressors, even those that have already been used. Second, the cascading effect of splitting the data makes the algorithm quite sensitive to noise. Decisions near the tree's root, even when based on noisy data, determine the entire structure of the resulting tree. Therefore, a small amount of noise can dramatically change the order in which we apply the decision rules, and the algorithm cannot quite recover from these early erroneous decisions. Luckily, there are ways to mitigate and take advantage of this phenomenon, using ensemble methods we shall encounter at the end of this chapter.

#im ../assets/equations/016/png/016-01.png 50 32 Equation 16.1 - Gini Impurity Index. Suppose a sample representing some split or predicted outcome only contains examples of a single class. In that case, the impurity will always be 0 because in all cases, p(i) is 0 (for classes that are not in the data) or p(i) = 1, the probability of the only class in the data. The theoretical maximum Gini impurity is derived below.

#h3 16.3 – Gini Impurity Index - Theory

#pg So, given our current data, how do we decide on the optimal split? But perhaps first, what do we mean by optimal split? An optimal split occurs when we split the data by thresholding on some input variable x such that most if not all the data belonging to a specific outcome of y end up on one side of the split. In contrast, the other possible outcome of y ends up on the other side. To quantify this, we will turn to the Gini index of impurity, as detailed in Equation 16.1. This index measures the homogeneity of a data set in terms of some outcome. It has maximum purity if a dataset contains only data points belonging to a single class. On the other hand, if all classes are equally represented in the data, it has the highest amount of impurity. You should read Eq. 16.1 as follows: the Gini impurity of a set of points is the sum of the probability of each class multiplied by its complement (1-p) across all classes. The probability of each class is simply the fraction of points assigned to that class label. Some examples will demonstrate more clearly what the impurity measures are capturing.

#h3 16.4 – Gini Impurity Index - Example

#pg Below are seven scenarios. First, we only see one class (red) in our 16 data points. Below, we gradually increase the number of examples for the other class (blue) until they are completely balanced. Notice how the Gini impurity increase from 0 to 0.5. After that, shifting further to more blue than red examples decreases the Gini impurity to 0. For a binary classification problem, the theoretical upper bound of the impurity is 0.5. Should we have three different classes, it will be 0.666. For m different classes, we find that the theoretical limit can easily be derived from the fact that it is highest if all classes m are equally represented in y. Equal represented means equal probability, and equal probability across m classes is 1/m since m∙1/m = 1. Plugging this fraction into equation 16.1 yields Eq. 16.2, which further simplifies to Eq. 16.3.

#im ../assets/figures/016/016-02.png 50 512 Figure 16.2 - The Gini impurity index captures the heterogeneity of the sample. The impurity is small for a dataset that mostly contains examples of one class. When all classes have a similar probability of occurring in a dataset, the impurity is high. Decision trees try to find the best input variable that splits this parent dataset into two smaller child datasets, which, when averaged, have an average impurity lower than the parent.

#h3 16.4 – Gini Gain - Party Game Example

#pg So, what does the Gini impurity have to do with building a decision tree? Imagine I am in a room with 100 strangers and asked to separate these individuals into two groups by age based on a single observable characteristic. All the people I label as 50 or older move to the right side of the room. All the people younger than 50 should make their way to the left side of the room. Without using any of their characteristics that I can observe, my guess on whether they are 50 years or older should be chance.

#im ../assets/equations/016/png/016-02.png 50 32

#im ../assets/equations/016/png/016-03.png 50 32 Equations 16.2-3 - The theoretical upper limit to the Gini impurity is a function of the number of distinct classes.

#br

#pg If somebody gives you nothing more than a name of a person, you would be hard-pressed to predict their age accurately (of course, names do change in popularity over time, so you wouldn't be entirely in the dark, but for the sake of this example, ignore that). I know that roughly 50% of the individuals in the room are younger than 50, and the rest are 50 or older. It would be high if I were to compute the Gini impurity index on this. All in all, not a situation we or our machine learning algorithm want to be in. Now, I can use that single observable feature to move people either to the left or right side. I pick whether their hair has a decent amount of gray (let us not split hairs over what counts as a decent amount). When I use that to move people around, I end up with roughly 50 people on one side and roughly 50 people on the other side of the room. But did I sort them correctly based on their age? A quick survey amongst the participants reveals that we have done well. Only four members were incorrectly predicted to be older than 50, and another four we incorrectly predicted to be younger than 50. That is an accuracy of 92%, which is not bad at all. We can also compute the Gini index again, now for both groups created by the split based on hair color. This reveals that both groups are highly homogenous, as they contain almost exclusively one class (in both groups, 46 of the 50 individuals are of the same class). And a highly homogenous sampling yields a low Gini impurity (or high Gini purity). Relative to the situation before the split when we had a high Gini impurity, after it, we now have two groups with low Gini impurity. We refer to this reduction as Gini Gain, which we optimize for when trying to find the best split possible within a variable and across variables. We used a categorical variable to split the data in this specific example. But we can use a continuous variable as well. In that case, we will look for the optimal value to split on, for instance, if the number of gray hairs make up at least 40% of a person's hair.

#h3 16.5 – Gini Gain - Synthetic Cancer Risk Data

#pg We can put these ideas into practice quite easily. I've created two small synthetic data sets on cancer risk that you can play with in the demo notebooks, which we will use here to work through. Specifically, both data sets have four regressors: age, gender, a person's favorite color, and the presence of the BRAC1/2 genes, associated with an outcome of whether that individual develops breast cancer at some point. The data set is an exaggeration to further build up an intuition for Gini impurity. Again, the data is also purely fictional, so please don't read into too much.

#im ../assets/figures/016/016-03.png 50 384 Figure 16.3 - Computing the best split in our cancer data set for the variable 'sex assigned at birth'.

#h3 16.6 – Gini Gain - Finding the best split within a variable

#pg With these three regressors, we need to find the best possible split across all inputs and across all input variables. Which input variables yield the highest Gini Gain when split on its optimal point? This process starts with computing the optimal split for each variable in turn. We should start there. Our input data can be both categorical and numerical. How many ways can we split a variable? As many ways as it has unique values. Thus, for a numerical value, the number of splits can be as high as the number of data points - 1. For a categorical variable, it can be as high as the number of classes - 1 (equally dividing any sequence of n will produce n-1 splits). An important and useful step is to sort the data along the variable of interest. In this way, the splits are self-evident. For numerical values, at each split, the values on one side are smaller, and on the other side higher of the split value (which is the average of the value in our data just to the left and just to the right). For categorical variables, it will show where the class labels change. If we sort on 'sex assigned at birth', for example, depending on how we sort, the first n samples in our data will all be of label 'Female', followed by the remainder of the samples all labeled 'Male' (in the case our data only has two values for 'sex assigned at birth').

#im ../assets/figures/016/016-04.png 50 384 Figure 16.4 - Computing the best split in our cancer data set for the variable 'gene found during screening'.

#br

#pg It is easy to find and utilize these transitions and consider them as splits. There is no point in splitting a categorical variable where the resulting two halves are not homogenous on that input variable. Such a decision is arbitrary and cannot be made again for new data. This limits the number of possible splits for categorical variables quite a bit. Before splitting the data, we need a baseline reading for the current Gini Impurity, which is easily calculated from our current sample using Equation 16.1. Note that this current sample can be the absolute starting point, which is the entire dataset, referred to as the root. Or we can be somewhere further down the decision tree branches. In this case, we refer to the current point in the tree as the parent node, with the paths created by splitting the data at the parent node leading to two new child nodes. For each possible split, we compute two new Gini impurities, one for each of the two halves of the data generated by a specific split. These three values together will be used to compute the Gini Gain as the difference between the Gini Impurity at the parent (the data before it is split) and the weighted average of the Gini Impurity at both children. It is a weighted average because it considers the number of examples in each of the two children. A very small number of examples in a split tends to inflate the purity. In extreme cases, a sample only contains a single data point, which must have an impurity of 0. Clearly, that is a bit artificial. The weighing of the Gini impurity based on the number of examples shifts the average Gini impurity towards the sample that can more reliably estimate the impurity, which means it has more examples. The animations in Figures 16.3-5 visualize what the demo notebook computes for each of the four input variables. It also shows that the best split is found for the gender variable and has an information gain.

#im ../assets/figures/016/016-05.gif 50 384 Figure 16.5 - Computing the best split in our cancer data set for the variable' favorite color'.

#h3 16.7 - Entropy and Information Theory

#pg An alternative to the Gini Gain is the Information Gain. Whereas the Gini Gain is based on Gini impurity, Information Gain is based on the measure of entropy. Since it is such an important metric, not just for decision trees but throughout the field of data science, statistics, and machine learning (we will see entropy again in our discussion on deep belief neural networks), it is worthwhile discussing it in more detail. Entropy has its origin in physics, where it describes the amount disorder or randomness in a system. The 2nd law of thermodynamics states that an isolated system, left unperturbed by outside forces and energy, will always arrive at a state of equilibrium and greater disorder. Entropy is the measure of the disorder of a system. Entropy also describes how much energy is available to do work. The more disordered a system, the higher the entropy, and the less of a system's energy is available to do work. An ice cube of each dropped in a pot of boiling water initially has available energy, transferring the heat from the water into the ice cube and melting it. After a while, the ice cube is completely melted, the system is in equilibrium, and the water temperature doesn't change. Physical systems always move in the direction of disorder (higher entropy), and the process is irreversible, linking entropy and the second law of thermodynamics to the arrow of time.

#im ../assets/figures/016/016-06.gif 50 384 Figure 16.6 - Computing the best split in our cancer data set for the variable 'age'.

#br

#pg Later, entropy became the cornerstone of information theory, the scientific study of the quantification, storage, and communication of digital information. Here, we used entropy to estimate the amount of uncertainty involved in the value of a random variable or the outcome of a random process. It has many applications across disparate fields like cryptocurrency, compression, digital data transmission, linguistics, etc. A popular description of entropy is the amount of surprise associated with some observation. If you are surprised about the observation, you will have gained some information about the system producing the observation. If you are not, you have learned nothing new, and we gain no information between the time before the observation and the time after. A more scientific word for surprise is probability. If something has a high probability of happening, we are not surprised when we observe, and vice versa. This framework of quantifying the amount of information (regardless of what it represents, like words, images, or stock market prices) is the work of Claude Shannon. He not only quantified information but also quantified with a measure something our digital age is built on – bits. To compute entropy, we first compute the probability of observing each state our system can be in and multiply that probability with the log of the same probability. We do this for every state and add those numbers together. Multiplying the result with -1 and we have our measure of entropy, expressed in bits.

#im ../assets/equations/016/png/016-04.png 50 32 Equation 16.4 - Information theoretic entropy is computed as the product of the probability of observing each possible state of the system times the logarithm of that same value, summer across all states, multiplied with -1 to make the measure increase with increased levels of information.

#br

#pg To understand this equation, let's first look at a single bit of information. A bit of information represents knowing whether some variable took one of its two possible states. An entropy of 0 bits for a two-state variable means we will never be surprised, as the variable will always be in one of its two states upon observation. In contrast, 1 bit of information for the same variable means the maximum amount of surprise. The two states are perfectly balanced in their probability of being observed. So, imagine tossing a perfectly unbiased coin. Half the time, it comes up heads, half the time, it will be tails. In other words, we have no way of predicting the outcome. And thus, every toss is as surprising as the last one. Using equation 16.4, we can compute the information in bits as follows:

#im ../assets/equations/016/png/016-05.png 50 32 Equation 16.5 - Amount of information (in bits) gained from flipping a completely unbiased coin. Each outcome is just as likely as any other (pheads = ptails), which equates to the maximum amount of surprise possible.

#br

#pg At the other extreme, the coin always comes up heads. This edge case is not computable, as it calls for the logarithm of zero, which is not defined. However, if we were to compute it for a coin that comes up heads 99% of the time, we can see that the entropy nears 0 and can be said to be theoretically zero for the case in which the coin comes up heads 100% of the time, see Equation 16.6.

#im ../assets/equations/016/png/016-06.png 50 32 Equation 16.6 - Amount of information (in bits) gained from flipping an almost completely biased coin. One outcome now is almost a given, which equates to the minimum amount of surprise possible.

#br

#pg But between those two extremes, we can still express the information gained in bits. For example, imagine a coin biased slightly to heads with a probability of 0.7. The amount of information is now:

#im ../assets/equations/016/png/016-07.png 50 32 Equation 16.7 - Amount of information (in bits) gained from flipping a biased coin. With unequal outcome probabilities, we just don't gain as much new information s with an unbiased coin.

#br

#pg We have gained some information because there is still some unpredictability. Still, since it is more often heads., we are less surprised about the outcome in general and have gained less information than the completely unbiased coin. As you can see, entropy doesn't have to be an integer number. It can be expressed as a real number, even though bits are 0 or 1. When variables can be in more than one state, the possible amount of information they can carry grows beyond 1. Another way of thinking about bits of information is how many I need to represent every possible state my variable can be in. For an unbiased coin, we need two states (one for heads, one for tails). For a completely unbiased coin I need no bits at all since there is only one possible state. For a slightly biased coin, the entropy is between 0 and 1, meaning I still need one bit since there are still two possible states. An interesting example is the entropy of our alphabet used for writing in English. Focusing only on the 26 letters, and forgetting about special characters, capitalization, and punctuation, what is the minimum number of bits we need to represent? One bit allows us to represent two possible states. Add another bit, and we have four possible states. To represent all 26 letters, we need n bits where 2n >= 26. In order words, we need to find n number of bits that can take on 26 or more possible states. That would be n=5, allowing us to represent 32 states. At n=4 it is 16, which is not enough. Even though we can't have fractional bits, we can use our formula for entropy to compute the theoretical number of bits needed in terms of information. If all letters were equally likely, the probability of observing any one of them is 1/26. Therefore, the entropy is:

#im ../assets/equations/016/png/016-08.png 50 32 Equations 16.8 - Amount of information (in bits) gained from observing the next letter in a string if that string was completely random. In languages using a Latin alphabet, like English, not all letters are equally probable. In addition, the sequence of letters is not entirely random either. Some letters in English are typically followed by certain other letters, and some letters will never be found adjacent to each other.

#br

#pg That number fits well with our observation on how many bits we needed to represent all 26 letters: more than 4 bits, less than 5. However, we know the letter aren't all equally likely. Observing an E is far more likely than observing a Q or K in English. Overall, our surprise in observing the most common letters is low, which we know should be associated with lower entropy. The table below has those letter frequencies and plugging them into equation 16.4 reveals that the actual entropy of the alphabet in English is about 4.2, less than if we had assumed the letters were all equally likely. However, it is still >4, so we will need 5 bits to store all 26 letters unless we start grouping them into pairs. Mutual information is a key extension of the amount of information in a variable. Mutual information is a measure of the dependence between two variables. More specifically, it quantifies the "amount of information" about one random variable by observing the other random variable. In other words, how much of my surprise of observing A is reduced by having already observed B? This is a useful metric in machine learning. If you can show the mutual dependence between variables, we have reason to believe we can build an algorithm that allows us to predict A from B, or B from A. That last statement is important. It is a mutual dependence, and we can't make any statement about whether A is causing B, B is causing A, or something is causing both A and B to happen. Remember, correlation (and mutual dependence) does not mean causation.

#ts 50

#th

#tc Letter

#tc Probability

#tl

#tr

#tc E

#tc 0.111607

#tl

#tr

#tc A

#tc 0.084966

#tr

#tl

#tc R

#tc 0.075809

#tr

#tc I

#tc 0.075448

#tl

#tr

#tc O

#tc 0.071635

#tl

#tr

#tc T

#tc 0.069509

#tl

#tr

#tc N

#tc 0.066544

#tl

#tr

#tc S

#tc 0.057351

#tl

#tr

#tc L

#tc 0.054893

#tl

#tr

#tc C

#tc 0.045388

#tl

#tr

#tc U

#tc 0.036308

#tl

#tr

#tc D

#tc 0.033844

#tl

#tr

#tc P

#tc 0.031671

#tl

#tr

#tc M

#tc 0.030129

#tl

#tr

#tc H

#tc 0.030034

#tl

#tr

#tc G

#tc 0.024705

#tl

#tr

#tc B

#tc 0.020720

#tl

#tr

#tc F

#tc 0.018121

#tl

#tr

#tc Y

#tc 0.017779

#tl

#tr

#tc W

#tc 0.012899

#tl

#tr

#tc K

#tc 0.011016

#tl

#tr

#tc V

#tc 0.010074

#tl

#tr

#tc X

#tc 0.002902

#tl

#tr

#tc Z

#tc 0.002722

#tl

#tr

#tc J

#tc 0.001965

#tl

#tr

#tc Q

#tc 0.001962

#tr

#te

#br

#ca Table 16.1 - Frequency of the letters in the English alphabet. Since not all letters appear in English with the same probability, the entropy of letters in an English text is lower.

#h3 16.8 - A deeper relationship between information and physical reality

#pg You might feel like a stretch to use entropy in two seemingly different settings, like physics and information theory. However, both cases describe the same property of a system: its disorder. There is still order and structure when the ice cube is dropped into a pot of boiling water. Depending on where you look, you might see molecules bouncing off each other frantically, whereas, in other places, they barely move. Like the famous quote: 'life is a box of chocolates, you never know what ya gonna get', a random sampling of some of the water molecules in the pot at this point has some probability of being a molecule in the boiling water or the ice cube. Eventually, the ice cube has completely melted by heat transferring from the boiling water, and the water temperature is at rest with that heat dissipated. No matter where we look in the pot, we'll see molecules bouncing around in the same way, there will be no surprises, and the entropy will be at its highest. In physics, the notion of entropy seems to be at the heart of the reality of our universe and why it is the way it is (or better: appears the way it appears to us), from its relationship with the 2nd law of thermodynamics and how it explains the arrow of time. Whereas traditionally, the directions of space are thought of as symmetric, time is not. For some reason, the now appears constrained by the before, not the after. Imagine a teacup smashed against the wall. There are an infinite number of ways that can play out. But in the vast majority of cases, it will involve the teacup shattering into thousands of pieces. Yet nothing prevents the atoms from ending up in the exact same spot, with the teacup unharmed as if nothing happened. Since there are so many more ways to be disordered, the outcome of any event will likely be a more disordered one. And disorder begets more disorder, and we find ourselves on a trajectory with a fixed direction: the direction of time. The surprising theoretical link between the notion of 'information' and the physical law underlying our universe has led some physicists to speculate whether information itself might be the fundamental building block of reality. It could be information that gives rise to things like time, space, mass, energy, object, galaxies, solar systems, planets, humans, brains, ideas, etc. This idea, sometimes referred to as 'It from Bit', is controversial, as you can imagine. But physics often shows that just because an idea is controversial doesn't mean it is wrong. Far from it.

#h3 16.9 - Information Gain in decision trees

#pg The way we use entropy to find the optimal split of our data is highly like how we use the Gini impurity to compute Gini gain. First, we compute the entropy of the data at the parent node. Then, after sorting the data, we compute the entropy for the data on either side of each possible split. A weighted average, based on the number of data points in each split, is computed and compared to the entropy of the data at the parent node, with their difference defining information gain. Note that we are aiming to lower the entropy. High entropy levels mean that we have roughly an equal number of data points for each class in each split. We aim to find a split such that both children are more homogenous than their parents, preferrable with all data points belonging to one class ending up in the left child node and all the data points belonging to the other class in the right node. The results obtained using Gini Gain (using the Gini impurity index) and Information Gain (using entropy) are quite similar. Although when examined closely, the entropy measures provide splits of higher quality (and therefore accuracy), the Gini impurity index is easier and faster to compute, without any risk of underflow errors, when taking the logarithm yields values smaller than machine precision.

#h3 16.10 - Building the decision tree

#pg Building a decision tree is typically implemented as a recursive algorithm regardless of your metric of choice. We continue to split a dataset up until there is no more gain, or we simply run out of data points for a node to split up. The easiest way to implement a recursive decision tree algorithm is to use a stack, a data structure designed to keep track of which nodes can still be split.

We initialize our tree by creating a root node, which is given the entire training dataset, a matrix X with n data points, and m regressors. We place this root node on our stack. We then continue to process the data until the stack is empty by taking a node of the stack and the training data that was parsed leading up to it. Deeper down the decision tree, we will have progressively fewer data to train on since early branches have divided the data into one or more child nodes. With this data in hand, we then iterate over each regressor, and within each regressor, we iterate over all possible splits. Having done so, we find the regressor x split combination which yielded the highest gain across all splits and regressors. We add this to the node as its decision rule and split data accordingly. We create two child nodes, each receiving one of the two new datasets generated by the split and put them on a stack. On each subsequent iteration, if the stack contains nodes, a node is taken off the stack, along with its data, and the process of finding a split that yields a gain is repeated, generating more nodes. If the data can no longer be split, or any possible split on the data does not yield a higher gain, we stop adding nodes on this branch and backtrack by picking nodes of the stack that had been created earlier while growing the tree. We can also set a stopping criterium by not allowing the tree to grow beyond a certain depth. Once we reach such a depth, no more nodes are created underneath it.

#h3 16.11 - Pruning

#pg Pruning refers to the bottom-up removal of nodes based on some criterium. Bottom-up refers to the fact that we begin at the node terminals, nodes with no children. We can use several pruning algorithms, and you will find three examples below. However, we feel that a deep dive into the pros and cons of pruning algorithms is beyond the scope of this course, so we have kept the discussion brief. We recommend the reader explores some of the reference materials we list at the end of this chapter.

#h3 16.12 - Pruning by information gain

#pg We can use information gain to prune our decision tree in two ways. The first approach is not adding child nodes if the gain isn't sufficiently large. Alternatively, we add all possible child nodes and prune subtrees with the lowest information gain after constructing the tree until we reach the desired number of leaves.

#h3 16.13 - Reduced Error Pruning (REP)

#pg REP belongs to the post-Pruning category of algorithms. In REP, we perform pruning using a validation set. If pruning a particular node results in a tree that performs no worse (based on some threshold) on the validation set than the original tree, pruning this node is justified.

#h3 16.14 - Cost-complexity pruning

#pg We apply cost-complexity pruning after we constructed the entire tree. In cost-complexity pruning, we calculate a so-called Tree Score based on the Residual Sum of Squares (RSS) for the subtree, and a Tree Complexity Penalty, a function of the number of leaves in the subtree. The Tree Complexity Penalty compensates for the difference in the number of leaves. We calculate the Tree Score for all subtrees in the decision tree and then pick the subtree with the lowest Tree Score. However, we can observe from the equation that the value of alpha determines the choice of the subtree. We find the value of alpha using cross-validation. We repeat the above process for different values of alpha, which gives us a sequence of trees. The value of alpha that, on average, yields the lowest Tree Score is the final value of alpha. Finally, our pruned decision tree will correspond to that alpha.

#h3 16.15 - Combining decision trees - Random Forests

#pg Random forests are an interesting extension of decision trees. Instead of building a decision tree, we build many of them, each from a random subset of the data (something we refer to as bagging), so rather than having a single tree predict an outcome, we have many, referred to as a random forest. Decision trees vote by committee, using a majority vote to determine the final predicted outcome. Interestingly, this often yields more accurate results. Using the 'wisdom of the crowds', the decision trees protect each other from making a prediction error. Decision trees can be pretty sensitive to slight variations in input, as a small outlier early on in building the tree has a ripple effect on the nodes below it, aggravating the error further. Thus, by creating many such trees, each using a random subset of the data, we mitigate this problem and instead assume that other trees will not have fallen victim to the small outlier of one tree.

#h3 16.16 – Conclusions

#pg The appeal of decision trees is that they are incredibly transparent. We can follow the entire decision-making process from the root node to the final predicted outcome, interpreting each step almost as a self-explaining narrative. "If the patient is older than 45 (first decision) and the patient is female (second decision), and there is a family history of breast cancer (third decision), then there is an increased risk of the patient developing breast cancer herself (prediction)". This is a far cry from the classification and regression algorithms discussed previously, where numerical weights dictate the mapping from inputs to predicted output. But comparing weights directly across input variables is usually impossible, given that the input variables represent data expressed in potentially very different units.

#h3 16.17 - Notebooks

#pg The notebooks demo how to compute the Gini Coefficient and Information Gain in isolation. In addition, I included two notebooks that demo the entire process of constructing the tree, one using a naïve method and the other using the more efficient and preferred recursive approach.

#h3 16.18 - References

#bs

#be

#bp Ash, R. B. (2012). Information Theory. United States: Dover Publications.

#bp Ananthaswamy, A. (2019). Through Two Doors at Once: The Elegant Experiment That Captures the Enigma of our Quantum Reality. United States: Penguin Publishing Group.

#bp Ferris, T. (2009). The Mind's Sky: Human Intelligence in a Cosmic Context. United States: Random House Publishing Group.

#bp Maimon, O. Z., Rokach, L. (2014). Data Mining with Decision Trees: Theory and Applications (2nd Edition). Singapore: World Scientific Publishing Company.

#bp Murphy, K. P. (2012). Machine Learning: A Probabilistic Perspective. United Kingdom: MIT Press.

#be