Module 9 Transcripts

Video 1: Introduction to module nine [01:35]

Welcome back to module nine on Decision trees, part one. Decision trees are an extremely important machine learning method, and there is a lot to say about them. So, they deserve a special treatment that spreads over two modules.
In module nine, we are going to discover individual decision trees, whereas module 10 is going to be devoted to combinations of multiple trees to tree ensembles. Similar to the previous module on *k*-Nearest neighbors, we will start with a practical example to support our intuition. Afterwards, we will learn how to construct a decision tree. In particular we're going to discuss how we can create the branches of a tree based on categorical as well as numerical features and how to choose the best branch using the information gain based on either the Entropy or the Gini index. Equally important, we'll also discuss how not to over grow a tree so as to avoid overfitting. We will discuss pre-pruning, where we stopped growing a tree when the improvement becomes too small, as well as post-pruning, where we cut down a tree once it has been grown.

Video 2: Divide and conquer [09:37]

This session will be all about decision trees. Decision trees actually used very broadly also in areas outside machine learning because they are very natural for humans to interpret. Think about this example. Imagine you get a job offer after graduating and you have to decide whether you want to accept that job or not. You could come up with a simple set of rules. For example, you could say, is the salary at least 40 thousand pounds? If it is not the case, you directly declined the offer. If that is the case, you may go for the second most important criterion, which is the commuting time. It was a commute less than one hour. If that's not the case, you decline the offer. If that is the case, you come to your third priority. That is, does the job of a free coffee. If that is the case, you accept the offer, otherwise you decline the offer and wait for better offers, possibly with some free coffee. Now, we want to use these trees not to make decisions, but to classify new data points. Before we do so, let's fix some terminology.

In contrast to real life, you will see that our trees grow from top to bottom, rather than from bottom to top. Accordingly, we call the topmost node, the root node. Moreover, all blue nodes on this slide. So basically, all inner nodes are called decision nodes. All other nodes, which are the terminal nodes, are called leaf nodes. Let's now see how we can use decision trees for classification problems. And as always, let's start with an example.

Imagine you work for a Hollywood studio. And you want to decide for pile of movie

proposals whether they are worth pursuing or not. You don't have the time to read through all these proposals. So, what do you want to do is you want to predict the movie success for each of these proposals based on two simple properties. Namely the estimated budget, which could be low or high, as well as the number of A-list celebrities, which again could be low or high. Those are the two input variables. In our case our categorical and the outcome. Your prediction is also categorical in this case, and it has three possible categories. Namely, it could be a critical success, which are the points that are red here. It could be a mainstream hit, which are the green points, or it could be a box office bust, so it could fail at the box office, in which case it is going to be a purple point here. The question now is, how can we classify these points that you see in a simple way using a tree technique? Well, let's start with the beginning.

Let's consider two possible splits of the data here. We could either split our data points based on the number of A-list celebrities. We could say. Let's look at all the movies that have few A-list celebrities on one side and all the movies that have many A-list celebrities on the other side. If we split the data according to these into these two classes, then we will see that a class with a low number of A-list celebrities will contain one critical success, one mainstream hit, and 10 box office bust. The other class of movies which are the ones with a high number of A-list celebrities, we'll have nine critical successes, nine mainstream hits, and no box office bust.

But this is not the only way to separate the data. We could also separate the data based on the estimated budget. If we do so, we will see that all the movies with all the proposals with a low budget will end up with nine critical successes. Five-box office bust and 0 mainstream hits. Whereas the ones with a high estimated budget end up with one critical success. Five-box office bust and 10 mainstream hit. So which split should we choose? Well, if we compare the two splits, we will see that at distinguishing between the number of A-list celebrities leads to higher purities into two classes, because we end up with one class that almost only contains box office bust. And we end up with another class that contains mainstream hit and critical successes in equal quantities. But 10 is better than the AMA split which led to mixed bags in both cases.

So, what we're doing is we're combining two fundamental principles of classification and regression trees. Here, we combine the divide and conquer strategy with a greedy approach. The divide and conquer strategy says, when you have a tricky problem, such as classifying all of these movies, split up the problem into smaller problems that you know, you can handle with. So rather than trying to build the entire tree at once, we start with the first split in the root node, that is divide and conquer. The second principle that we're using here is the greedy principle. The greedy principle tells us, make decisions now without thinking too much about the future. And we have applied this principle here by just looking at the purities of the two figures that we have created, we have not worried about future splits that we

need to do. The combination of divide and conquer and greedy is going to be what defines classification trees and regression trees.

Let's take this one step further. We have now made the split based on divide and conquer and greedy principle that we say, well, for all the movies, we first classify them in low or high number of A-list celebrities. We can now add our unexplained. We could for example, split the ones, the movies with a high number of A-list celebrities into those with a low and a high budget. If we do so, we end up with the following subclass purities. The movies with a high number of A-list, celebrities and a low budget are mostly critical successes. Actually, all of them are critical successes. The ones with a high number of A-list celebrities, as well as a high budget, are mostly mainstream hit nine out of 10 or mainstream, hence there's only one critical success and no box office bust. Alternatively, we can split the movies that have a low number of A-list celebrities into those with a low budget and a high budget. If we do so, then we end up with the movies with a low number of A-list celebrities and a low number, low budget, which are mostly box office bust5 out of 6 or box-office bust. There is one critical success. On the other hand, the movies with a low number of A-list, celebrities and a high budget are mostly box office bust, again, five out of six. But there is now one mainstream hit rather than a critical success. So again, applying the greedy principle here, we need to decide whether we split the movies with a high number of A-list celebrities depending on their budget or the movies with a low number of A-list celebrities depending on your budget.

If we compare these two splits, we see that marginally split amongst the high-level A-list celebrity movies is better because it creates pure end nodes. If we stop the process here, we end up with a tree that is shown here. This tree first categorises the movies based on the number of A-list celebrities. If the number of A-list celebrities is low, when we look at the number of movies from our database that we, that we have in that part of the search space. And we see that 10 out of 12 movies or Box Office bust. So, we take a majority vote in that leaf node, we will say any movie with a low number of A-list celebrities is a box office bust. If the number of A-list celebrities is high, then we make a second choice based on the estimated budget. That was the second split that we introduced. If the estimated budget is low, then we see those eight out of eight movies that fall into that part of the search space where critical successes. So, taking a majority vote, the leaf node under this part of the tree will classify any sample as a critical success. And finally, the remaining load will be classified as a mainstream head because nine out of ten movies where mainstream hit. So, the majority vote here says, we should classify any movie falling into that part of the tree as mainstream head.?

Video 3: How to split trees [04:11]

Let us first discuss how we can create a split based on a particular input variable. That depends, actually, whether that input variable is a categorical input variable or a numeric input variable. Let's start with categorical input variables. In this case, there are three popular choices that are implemented in practice. The first choice is that we create a child node for every possible category of that input variable. So, for example, if we have an input variable that is income level and it's a categorical input variable with three categories, namely low, medium, and high, we would create three different child nodes representing each of these three categories. A different possibility is to introduce only two child nodes. Namely, we look at a particular category, such as the category low, and we could, we construct two child nodes, one for low and one for any other category than low. Finally, instead of just considering a single category, we could create binary splits for any possible subset of categories. So, for example, we could again split on the income level. But now we look at the subset medium or high, and we create one child node for any sample whose income level is either medium or high and one child node for any other sample, which in this case is simply these are all the samples that have an income level of low.

Now, as I mentioned before, all three of these splitting possibilities for categorical predictors are actually being used in practice. The reason for that is because they all come with different advantages and disadvantages. We will see later on that the first splitting possibility which creates a child node for every different category can lead to a biased selection of splits. In particular, the algorithm may prefer splits that create many different child nodes because those splits seem to be very advantageous under the greedy rule. The second splitting possibility into binary, the binary splits that distinguish whether the samples are of a particular category, or any other category don't have this issue, but they typically lead to very deep trees because we make very small decisions in every level of the tree. The third choice, finally, alleviates some of the disadvantages of the second choice but it has its own problems. Namely, it requires us to evaluate a large number of possible splits. Consider, for example, an input variable that has 10 possible categories. In that case, we would have to consider every possible split into a non-empty subset that is not the entire set of categories, which would be 2^n-2. In this case, 2^10-2.

Let's now look at splitting possibilities for numerical input variables. In that case, one typically considers binary splits only. And the splitting possibilities that are considered are all the midpoints between consecutive values of this, numeric input variable that can be found in the training data. So, if you see the points here on this graph, then you can think of these points being ordered from left to right, and we consider the splitting possibilities 1, 2, 3, 4, 5 and 6. Always splitting between all the samples whose input variable value is less than or equal to the midpoint versus those that are greater than the midpoint.

Video 4: Measuring purity with the entropy criterion [08:14]

Now that we know how we can split nodes of the classification or regression tree, we should investigate um, which splits we should actually conduct, in particular, which node of the tree should be split upon and upon which of the explaining variables should we split. In order to uh, solve this dilemma, we will use the purity criteria. In particular, what we want to do is we want to conduct those splits that maximally increase the purity of the tree. We will see that the increase in purity is going to be measured by something that is known as information gain. But the information gain will depend on a measure of purity, and the first measure of purity that we're going to consider is the so called entropy. The entropy is actually an expression from information theory, and it measures the capacity of a communication channel. Let's have a look at this. The entropy is defined as the minimum expected number of bits, so 01 values that are required to encode one realisation of a random variable 'x'. And this is the formula for the entropy of a random variable 'x'. It is the sum of all possible realisations of 'x', the negative of the probability of the realisation times the two logarithm of that probability.

Let's have a look at a simple example. Consider a random variable 'x', that takes on the two values, zero and one, both with probability one half. In that case, the entropy would be calculated as minus one half. This is the probability of the realisation zero times the two logarithm of one half minus the same expression again, now for the realisation one. It turns out that this is just minus the two logarithm of one half. And if you remember, this is the same as the two logarithm of two. Because the, we can take out this division and this is nothing else than one. So, in other words, this formula shows us that if we have a random variable that produces zero or one with equal probability, then we need on average one bit to encode one realisation of that random variable. This is not very surprising.

Now let's have a look at another extreme case. Imagine our random variable would never take the realisation zero, rather it would always produce the realisation one. In this case, we would end up with minus zero times the two logarithms of zero, which is to find a zero minus one times the two logarithm of one, which is also zero because the two logarithm of one is zero. So, in other words, our entropy formula would tell us that we don't need any, we don't need any bits to start realisations of that random variable, which also makes sense because we know what the random variable is going to produce, it is always going to produce a one.

Now let's have a look at a slightly more sophisticated example. Imagine a random variable that produces four possible values, namely it produces 'A' with a probability of 0.7, it produces 'B' with probability, let's say 0.26, it produces 'C' with probability of 0.02, and it produces 'D' also with a probability of 0.02. So, in other words, the different realisations of the random variable here are highly unbalanced. Now, Now, what we could do is we could encode this random variable with two bits, namely,

the bit combination 00 could be used for the Character A, 01 could be used for the Character B, 10 could be correspond to Character C and 11 could correspond to Character D. The entropy formula tells us however, that we need less than two bits to describe realisations of this random variable on average. In fact, if we look at the formula for the entropy, we would obtain something like the following. We would have this expression here which will value is roughly to 1.09. So, in other words, for this random variable, we do not need two bits to encode the realisations of this random variable on an average, as this encoding here would suggest. Rather, we only need on average 1.09 bits. How could that be possible?

Well, one encoding that improves upon our two-bit scheme would be the following one. Whenever we observe an 'A', we encode that with a single bit zero. If we observed a 'B', we encode that with two bits, namely one followed by zero, a 'C' we could encode with 110 and a 'D' we could encode with 111. If you think about this for a moment, you will realise that this encoding is unique in the sense that, whenever we get a sequence of zeros and ones, we can exactly decode the message. Now, let's look at the expected length in terms of bits to encode one single character according to this encoding scheme. Well, it would be 0.7, which is the probability of the random variable being A times one bit plus 0.26, which is the probability of producing a B times two bit, plus 0.02 times free bits for the C and the probability of 2% times free for the encoding of D. And this is about 1.34. So, in other words, we have found an encoding here, which is much better than our two-bit encoding that we have seen before. it only requires 1.3 bits.

What Shannon says in his entropy information criterion is that the optimal encoding would, on average, use 1.09 bits. This is what we have calculated here. Now, what does this entropy mean in our context? It turns out, and I hope that this example illustrates that to a certain degree, entropy is a measure for an order for disorder. The higher the entropy, the more bits we need to, uh, to represent the realisation of the random variable meaning the more uncertain the outcome of the random variable is. In other words, in a tree, entropy is our enemy because high entropy associated with a particular node of the tree implies that the data samples that we have in that note are very heterogeneous.

Video 5: Selecting splits via the information gain [02:41]

Let's see how the formula for the entropy would work in the context of a tree. In the context of a tree, we use our entropy formula just as we have introduced it before. However, now the probabilities no longer correspond to probabilities of certain realizations of a random variable. Rather, they correspond to the fraction of the training data that belongs to a particular category. So, in other words, it could be that in a node of the tree, we have 50 percent of the samples associated with that node being of category one, 25 per cent being of category two, and 25 per cent being of category three, in which case the entropy formula would say we should take minus

0.5 times the 2 logarithm of 0.5 minus 0.25 times the 2 logarithm of 0.25 and again minus 0.25 times the 2 logarithm of 0.25 for category C. Now, we have a way to measure the disorder of a node in the tree. What we want to do is we want to identify splits that maximally decrease this disorder. And this is going to be measured by a criterion that is known as information gain. To calculate information gain for a particular split, we do the following. We first calculate the entropy of the node that we are about to split, and then we calculate the entropies of the children nodes that would emerge as a result of this split. We then subtract from the entropy of the node to be split, the weighted combination, the weighted linear combination of the entropies of the child nodes. The weights here are the proportions of the training data in this node and that we are about to split that fall into each of the children nodes. So, in other words, if 50 percent of the training data from this node n that we are about split go into child node 1, w1 will be 0.5.

Video 6: Measuring purity with the Gini index criterion [05:59]

It turns out that the entropy criterion is not the only way that is available to us to measure the purity of a node. In fact, we can alternatively use the so-called Gini Index. The Gini index, formally, is defined as follows. It is the probability that a randomly chosen training record within a node of the tree is mislabelled if it is labelled randomly according to the proportions of the training data in that node. And this is the formula to calculate the Gini index within a node where again P_i is the fraction of the training data in that node that is of Category C_i. In order to understand that criterion better, let's have a look at a simple example. Let's imagine that we have a node N in the tree where, let's say, 50 per cent of the data in that node is of category 1, 30 per cent of the training data associated with that node is of category 2 and the remaining 20 per cent of category 3. Now the Gini index says that we are interested in the probability of randomly choosing a training record. And the probability of this randomly chosen training record being mislabelled if we label it randomly according to the fractions of training data in that node. Let's try to make sense out of this. What is the probability that we randomly select one training sample of node N and that training sample is of category 1? Probability of randomly selecting a training sample of category 1. This is, of course, 50 per cent, 0.5, because 0% of the samples in that node are of category 1. And likewise, the same for category 2 and 3, the probabilities are 0.3 and 0.2.

Now, what is the probability of mislabelling a training sample, a random training sample, given that this training sample is of category 1? The answer is, we are supposed to label this training sample according to the fractions present in the node. So, in other words, we will randomly label this training sample with probability 0.5 of category 1, which in this case would be the correct labelling. We would randomly, with probability 0.3, categorise it as belonging to category 2, and with probability 0.2, we would categorise it as belonging to category 3. So, number one is the probability of mislabelling this training sample assuming that it's from category

1 is 0.5. We can do exactly the same with category 2, selecting a random training sample. The probability of randomly selecting a training sample that is of category 2 is 0.3. So, randomly selecting a sample of category 2 is 0.3. And the probability of mislabelling this sample given that it is of, sorry, of category 2, this probability is now 70 per cent. Because with probability 50 per cent, we label it as category 1. Only with probability 30 per cent, we label it as category 2 and with probability 20 per cent, we label it as belonging to category 3. So, it's going to be 0.7. And likewise, the probabilities for category 3 will be 0.2 and 0.8. And we see that 0.5*0.5 + 0.3*0.7 + 0.2*0.8. This formula here is exactly the formula for the Gini index.

So, the interpretation of the Gini index is if we were to randomly pick one training sample from that node, we subsequently randomly label that training sample according to the fractions of categories being present in that node. And the probability of this overall process leading to a mislabeling of the training sample, this is the Gini index, and it is another measure for disorder. In fact, it turns out that the entropy and the Gini Index behave very similarly. The higher the entropy, the less orderly a node is. The higher the Gini index, the less orderly a node is as well. There is one important difference, however. The entropy can be any non-negative number because it represents a number of bits that is required to represent a realisation of a random variable. Whereas the Gini index is a probability, and as such, it will always range between zero and one.

Video 7: Why not use the misclassification rate? [03:23]

It's time to discuss an apparent mismatch in the tree construction process, that is often pushed under the rock in the literature. Remember that we construct classification trees by maximising the information gain, which is either based on the entropy or the Gini index. But we often evaluate the performance of the classification tree by looking at the error rate, which is a different criteria. So, a natural question that arises is, why don't we just grow trees by looking at the error rate directly from the beginning? Let's explore why this is the case with a simple example.

Consider the following tree displayed here, in this tree we have a binary output variable, a tick indicates yes, and a cross indicates no. And our tree is built based on 30 samples, to simplify the exposition in that tree I do not show which features I split on they are irrelevant for the purposes of our discussion. We see, when we look at the tree that eventually we obtain a perfect tree that correctly classifies all the data. But would we actually obtain this tree if we were using either the information gained using entropy, or if we want to minimise the error rate. Let's investigate that. If we optimise the information gain using the entropy, then the entropy of the root node would be 1.113, whereas the entropy of the children would be one for the left child and 0.811 for the right child. This gives the first split and information gain of 0.239, that is the split would indeed be taken because the

information gain is positive. The other splits clearly improve the information gained as well, so without even calculating them, I can assure you they would be taken as well. So, what about the error rate? The root node misclassifies 10 samples, if we were to take a majority vote. The two children of the root node, however, would also together jointly misclassified 10 samples if we were to take majority votes in each one of them. In other words, the first split would not improve the error rate and hence would never be taken.

The bottom line of this story is that the error rate can be too crude as a measure of progress in the tree construction process. It may tell us to stop the construction process early, even if additional gains could be achieved further down the tree. The entropy on the other hand is a more sensitive criterion that can detect even minor immediate progress that may pay off later on in the tree construction process, and this is why we normally prefer the entropy and the information gain when we construct trees.

Video 8: Decision tree pre-pruning [02:11]

We now know how to grow a classification tree. In principle, we can do so until either there are no more features left to split on or until all of the training data has been correctly classified, in which case, there is no point introducing any further splits. It turns out that this strategy does not normally work well in practice. In fact, you might already guess that this is a manifestation of the variance problem. If we try to correctly classify all of our training data or we try to use all of our features to classify data, then we are likely to overfit to the training data, meaning we end up with a decision tree that might work very well on the training data, but it will not generalise well to new data. You remember the higher order polynomial functions that we have discussed to fit to a point cloud? This is exactly the same problem. Now, one strategy that is commonly used in practice is called decision tree pre-pruning. It's a very simple strategy which looks at the information gains that we obtain by choosing the best split at every iteration of the tree generation algorithm. And once that information gain becomes too small, we stop, because introducing any further splits seem to not be useful in terms of the bias-variance trade-off. It turns out, however, that the decision tree pre-pruning comes at a certain cost and that cost is the likelihood that we may prematurely terminate the construction of the tree. This is a manifestation of the greedy principle which hurts us here. Let's look at this with a simple example.

Video 9: Decision tree post-pruning [02:52]

As we have seen in the previous example, decision tree pre-pruning needs to be exercised with caution. It's a myopic strategy, and due to the greedy principle, we may uh, miss uh, good splits that come later down the tree. Because of this, many

algorithms use what is known as decision tree post-pruning, particularly popular ah algorithm, because it's very efficient for decision tree post-pruning is a so-called cost complexity pruning. I don't want to go through all the glorious details of this algorithm, but I want to present the main principle here.

What we do is the following: We split our data that we have into a training dataset and a validation dataset. This can by the way, also be done using cross validation. Now we grow a complete tree using the training data, so we don't use any pre-pruning on that tree. That tree is likely to be too large. It is likely to be over fitted. Now, what we do is the following, we consider a weighting parameter alpha that weights two competing objectives. The first one is the impurity of the tree that we obtain on the training data, which we want to be as small as possible. The second one is the size of the tree, measured in the number of leaf nodes, which we also want to be as small as possible. So, what we do now is for every possible value of alpha, we find a sub tree of the completely grown tree that we have just determined in step two, that minimises this weighted objective function. If alpha is very small, we end up with the entire tree - T Zero. If alpha is very large, we end up with a very small tree, probably just a root node, because in that case, the penalty for the size of the tree by far outweighs the penalty for the impurity. We find this for every value of alpha, we find a different sub tree and then using the validation data set or cross validation, whichever you use, we take the best tree corresponding to the best value of alpha in terms of performance on the validation data set. Now the key to the success of cost complexity pruning and that's the part that I don't want to discuss here, is that all these sub trees for the different values of alpha can actually be found very efficiently.

Video 10: Regression trees [02:47]

So far, we have only discussed classification trees where the outcome variable is categorical. Let's now discuss briefly regression trees where the outcome is numeric.

It turns out that everything we have learned so far can be used almost unmodified, to work with regression trees as well. There only two things that need to change. First of all, how do we predict the outcome? It's no longer categorical outcome, but a numeric outcome. And secondly, how do we choose the best split? How do we define the information gain of a split? Now the first question is simply solved by replacing the majority vote in the leaf notes with the average outcome. So, we take all the samples that are part of a particular leaf note. We take all the outcome variables, outcome variable values of these samples, and we take the average that is going to be our prediction. In terms of the information gain, we will change the purity criteria, we no longer use the Entropy or the Gini Index rather, we use performance indicator for a numerical outcome variables such as the mean squared error.

It is interesting to compare regression trees with linear regression, which is another popular approach, of course, for numeric prediction problems for regression problems. You see on the left that a linear regression leads to a linear function of the input variables. This is not the case for regression trees. Regression trees are piecewise constant where each piece is a rectangle that's due to the access parallel nature of the splits that we introduce in the regression trees. So, in our terminology of bias variance trade-off, we see that the two methods introduce a different form of bias. Linear regression trees, linear regression assumes that the function that the outcome variable is a linear function of the input variables, whereas regression trees assume that the outcome is a piecewise constant rectangular function of the input variables. In practice, typically neither of the two is going to be true. We are going to introduce a real bias. Nevertheless, for many problems, one assumption is more realistic than the other, in which case one method will work better than the other.

Video 11: Case study: churn detection and prevention [09:05]

Let's talk about Churn Detection and Prevention. First of all, let's discuss what is churn. So, churn originally meant something was moved around, jostled, or shaken, and it was used to describe the process of making butter by shaking or sloshing milk. Or what happens to the water in a lake when a boat goes by and leaves a wake, so it churns the water. More recently, it's entered the business lexicon to refer to when a long-standing customer stops buying a service or, less commonly, a product. So, the customers should be long-standing, and they should stop patronizing it for it to be churn. So, going to get your hair cut once and then not going back to the salon probably wouldn't be considered churn. Some people might consider that to be churn as well, but mainly we're talking about sort of your long-standing, your subscription customers who stopped subscribing to your service.

So, how do you calculate churn? Typically, what you do is you divide the number of customers that you've lost during the time period. For example, in one month, how many customers did we lose, and we divide that by the total number of customers that you had at the beginning of that time period. So, for example, you could have 5,000 customers at the beginning of the month, and you lost 500 of them during the month, so you had a 10 per cent churn. Now you may have added other customers as well, but we're really concerned about the ones that you lost here. So, you don't strictly need to use the number of customers. That's one model of churn. Another one could be the value of business lost because some of your customers might be actually better customers than others. So, the number of customers assumes everybody contributes equally to your company. But if you think about the business value, you may actually want to consider, hey, we're losing big customers here and the small ones, the cheaper ones are staying on, and that's not good. So, there's a lot of different ways of

calculating churn, but many of them centre on customers lost or the business value of customers lost.

Now, let's think of some examples of churn. Probably the best case is mobile phone subscriptions. Mobile phone subscriptions typically have around a 30 per cent churn per year. They get a two-year contract, and then they go month-to-month on their cell phone subscriptions, probably something that all of us have experienced. If at any point during that contract period, even within the first two years or after the contract's over and they're month-to-month, they switch to a different carrier, then we would consider that to be churn. Credit cards also have a high churn rate. Typically, at the end of each year, you have to re-up; you have to pay again the fee for the credit card to continue for another year, and about 20 per cent of customers don't actually continue each year. There's businesses and software as a service subscription. So, in other words, you are a business to business company, and you are selling software as a service that has a very low churn rate, usually around 6 per cent. Retail banks have a high churn rate, typically around 20-25 per cent per year, and you think about it in the past, actually, a little harbinger of bad things to come. If you think about daily newspaper subscriptions back in 2003, the churn rate at that point in time was around 60 per cent. So, they were losing 60 per cent of their customers subscribers per year for daily newspaper deliveries.

So, why do companies want to minimise churn? First thing, it's very expensive. Customer acquisition costs can be quite high in some sectors. And when there's a lot of churn, you end up having to divert resources to get new customers. And Forrester Research has done some work on this and found that it costs roughly five times more to acquire a new customer than to retain an old one. And the money that's diverted to customer acquisition essentially cuts into your profits. So, in addition to that loyal customers, or returning customers often spend more, and even a small reduction in churn, can lead to a large increase in profits. So, the best estimates would be that a 5 per cent reduction in churn can lead to 25 - 95 per cent increase in profits. Overall, in the economy, companies lose about $1.6 trillion on churn every single year, and most companies have no strategy to deal with churn. So, that's why it's important.

Now, typically the way companies that do have a strategy to deal with it. What they do is they survey the people who leave. And this has probably happened to you. You decided to cancel your subscription, and they send you a little note saying, ‘Oh, we're sorry to see you go. Could you please tell us why you're leaving? Did you think it was too expensive? Did you think it was too inconvenient?’ Blah, blah, blah, blah. And you're supposed to fill in this survey. Now that's fine. That's actually better than nothing because there you're trying to understand what's leading people to leave. But by then, it's already too late. So, what if you could predict churn before it even happened. So, rather than trying to interview customers after they leave, they may not even answer your message, or they may not tell you the true reason they're leaving. It would be a

lot better if you could stop them from leaving in the first place, so that might happen if you were to contact them before they're about to cancel and offer them some incentive.

For example, a few months before my cellphone subscription expired, I received a text message from the company saying, ‘Hello, I have a much better plan for you. I have better coverage, better data, better roaming, and it's actually cheaper than what I'm charging you right now. And if you were to commit to two years of this deal, I can give you this price.’ And I actually took that. I took that deal at the time.

So, how do decision trees fit into this? There are many ways to predict churn, and they have varying degrees of complexity. However, the more complex the algorithm that you have, the less transparent it is and the more difficult it is to explain who gets targeted for incentives. As it turns out, decision trees are very simple. They're very easy to explain, and they're usually quite accurate, which makes them ideal for analysing churn. And that's why many companies use simple decision trees to do these kinds of targeted incentives.

So, in this particular case, we would have a very simple model where the first question is, ‘Are you under a long-term contract or not?’ Most people who are under a long-term contract do not switch carriers. In other words, if you've got a two-year contract, you're not going to leave it in the middle. But once the contract is over, then you're much more likely to switch. If you're month-to-month, or you're approaching it, you may want to know the second question. So, these are kind of a series of yes or no questions. "Are you in a long-term contract; yes, or no? Yes, okay; unlikely to switch. No, all right. No, I'm not in a long-term contract, but do I have any other bundled services such as home Internet? Because, as we know, people who have bundled home Internet with their mobile phone subscriptions are much less likely to switch. So, that's the next question you would ask. And then for a very simple model; this is a three-question model. You might want to know how long have you actually been month-to-month, for example? Has it been over a year? They'll say greater than 12 months or less than 12 months. I'm dichotomising that. Basically zero, less than 12 months, one, 12 months or longer. But you could do that empirically by trying to see where the breakpoint is and try and find a good place to test it. As it turns out, people who have been month-to-month for over a year, who are not on a contract and who have no bundled Internet, are highly likely to switch carriers. So, it makes sense to offer these customers some kind of incentive to prevent them from switching. Now, a decision tree could also be more complex, and this is a three question one, and many companies have 10 questions or something like that. But the basic idea here is that you have a simple model by answering a series of yes or no questions, you can actually figure out who is more likely to quit, and then you can target just those customers for an incentive for staying.

Video 12: Summarising module nine [01:32]

This brings us to the end of Module nine: The first part of Decision trees. In this module, we have seen how trees can be grown efficiently by combining two different principles. We used the divide and conquer principle, that splits a complex task of growing an entire tree into more manageable task of branching individual notes one at a time, and we used the greedy principle to choose splits based on their immediate information gain and never revisit a decision once it has been taken. We also learned how we can control the bias-variance tradeoff via tree pruning. We use pre-pruning while we grow the tree, and post-pruning after we have grown the tree. Pre-pruning is in some sense more efficient, but it is also short-sighted. It can miss beneficial splits further down the tree. In practice therefore, both types of tree pruning are used together. In the next module, we will move from the construction of a single tree to the construction of tree ensembles.