I. Good morning everyone. So you might. Recognise that slide as a slide from last week. Actually, the reason why I'm pulling this up again is I got a couple of questions about PCA, but particularly PCA in combination with regression, which we would then call principal component regression. So I thought I would spend another couple of minutes, just 2 or 3 minutes to explain again the idea behind PCA and PCR to make sure that it's absolutely clear. So I hope that from PCA, what you do remember is the idea of PCA being a dimensionality reduction technique. So the idea behind PCA is that if you have a lot of features, a lot of variables in your data set, and you're not entirely sure how to best include all of them in your model, one way to actually reduce your space would be to use PCA to basically combine multiple of your variables into components, so that instead of having, for example, 50 variables explaining different concepts or features in your data set, you're reducing that to a specific number of components, let's say ten. And then these ten components are combinations of all of your original 50 features. The components are composed in such a way that they should include most of the variance, or most of the information that is in the data set. I think that is relatively self explanatory. The idea is that you want to keep as much information as possible, while still reducing the number of variables necessary to explain that information. The main idea behind that, why it works is that in a lot of data spaces we actually have collinearity. So correlation between the different variables. What that means in terms of information is that they partly explain similar or the same information. So I made this example where you have a questionnaire where different questions in that questionnaire are actually measuring very similar concepts or similar kind of underlying ideas or features. And if that is the case and those questions would be correlated. So the values would actually strongly correlate with each other because they're measuring basically the same thing. And PCA is exploiting this kind of collinearity. And then combining these correlated variables into their features into their components. That also means that the components of PCA are by definition independent from each other. So there is no collinearity between the features or between the components of your PCA result, which is really useful for linear regression. I think that was actually one of the motivations why PCR is kind of a standalone method. So in theory, PCA itself is just a free processing tool. You can use it in combination with any method that you would like. You can just do PCA. Then you have your components and you can use them as variables in any type of model, in your decision tree or in your I don't know, UK and N whatever you're interested in. PCR as a kind of standalone method is a bit. It's a bit confusing why we described that. Yes. If you could close it off. Thank you. When we described that as a standalone method, even though it's basically just a combination of PCA with regression, I think the reason for that is one, one of the main problems in regression is if you have correlated features, correlated variables. So it sometimes happens when you try to run a linear regression in Python, for example. And Python basically gives you an error message and tells you, you know, there's collinearity in your data set cannot do. And then you have to figure out okay, which are the correlate features. And then you can try to remove some of those features etcetera etcetera. Another way of actually achieving that is to use PCA in combination with regression. Because as I mentioned per definition, the components are not correlated with each other. So that would completely remove the problem of collinearity in your regression model. Now PCR as a kind of method goes a step further when you choose the number of components. As I mentioned earlier, you want to choose them in such a way that they include most of the information or variants in your data set. So that's the way you choose how many components you want to have in your data. You want to, for example, capture 70% of the the information in your data set. That's like a rule of thumb you can choose. So that would be kind of your evaluation parameter of your PCA solution. So you would choose as many components as are necessary to achieve this evaluation. Parameter of information captured variance captured in PCR. You know that in regression we can use actually error message error measures as a tool for evaluating our over a linear regression. So PCR basically then uses this evaluation measure of error message a measure from the regression as our evaluation tool for our components. So we choose the number of components which optimises our linear regression. That's why it's basically a kind of combined method. Because instead of doing first PCA and then doing a linear regression, we actually optimising the number of components within our regression model. You've seen that in the computer lab actually where we are using cross-validation to choose the number of components. But we actually cross validating over the linear regression. So we're running multiple linear regressions with different numbers of components. And then we're choosing that number of components which optimises that linear regression. So that's the idea behind PCA and PCR. Are there any questions about that. Or is that a bit more clear than last week? Good. So it takes a bit of thinking, but basically PCA pre-processing, combining features and PCR using those combined features in your linear regression. So the linear regression formula in that case will just be you're predicting y using your features using the components. So each component as a variable. So for example beta zero plus beta one times PCA one plus beta two times PCA two. So these are just your components. In that case you just treat them as variables because the idea is that they are linear combination of your original variables. So you can treat them as your normal variables, with the caveat that you can't really interpret them very easily because they're kind of combination. So difficult to interpret takes away a lot of the interpretability of your linear regression, but removes the problem of collinearity. So quite useful. So that was a bit of a quick look back into. PCA and PCR. Now in today's lecture, we are continuing our I'm not setting edge as default. Are you crazy? Terrible browser. Come on. Okay, here we go. That's the beauty of being a lecturer and not a teacher. I can just tell you my opinion about things. Okay, so it's week nine. You're almost done. The end is near. For better or for worse. The next couple of weeks. What we're doing is basically using each week to discuss one group or one specific method. So this week we will be talking about three best methods decision trees, and kind of methods based on decision trees. Next week we will be talking about support vector machines. And then we will be talking about neural networks in week 11. And then you're done. So you're almost you're almost through. Yes. As I mentioned last week, we talked about K and N. I hope that was additively clear. And we talked about feature selection and dimensionality reduction through PCA and PCR. Now this week what we'll be talking about is decision rule based classifiers. And specifically we will be talking about decision trees, both classification regression trees. We will look into how these trees can then be combined into ensembles instead of being used separately. Which leads us to bagging approaches in general. And we will talk a little bit about boosting trees. And we will talk a little bit about random forests, everyone's favourite methods. If it's not neural networks. So three base methods are really kind of fun to teach because the idea behind them is really intuitive. They basically what they do from a mathematical point of view is they stratify. So they segment your predictor space into different regions. And that sounds a bit odd, but we will see what that actually looks like in a graph shortly. The idea then is that if we want to predict a new point, a new data point, or an unlabelled data point for which we don't have a class label or some kind of output measure yet, we would then use the mean response of that predictor space. So it's similar to the idea of k and n. You will see that repeated multiple multiple times through different measures, different methods. The idea that we are kind of defining a space, a predictor space, or we're defining neighbours and then we are labelling a new point and unlabelled point based on the known behaviour of similar points, close points, points in the same predictive space, anything like that. So there's always this idea of how are we predicting something? We're using previous information, past information that we already have. And then based on that we look for similarity in some sense, and that that's what we use for the prediction. So the idea behind decision trees is specifically that these rules on how we divide our predictor space can be summarised in a tree, hence the name decision tree. And the great thing about decision tree is definitely that it's not just called a tree. It actually looks like a tree. So you can visualise the process that goes behind how the predictions are actually made. Later we will also look into more advanced methods specifically that is combinations of multiple trees. You will see that decision trees are amazing because you can visualise them and they are pretty to look at and they are really straightforward to implement and explain everything, but they are not really good, unfortunately. So single trees are kind of weak methods in many cases, at least many cases if you have a bit more complexity in the data. So if it's a really simple, straightforward model, an idea and data space, a decision tree will do a good enough job. Don't worry about it. But if you have more complexity, you have more features. For example, you have a lot of complicated non-linear relationships. Decision trees on their own don't really do a very good job. So we generally say they are relatively weak predictors compared to other methods. But combining trees is actually really, really powerful. So we will talk about random forests later. And random forests are actually thought to be one of the most powerful predictive tools that you can use. So if you read scientific papers and they do some kind of random forest comparison with a different model, in many cases random forests vastly outperform more complex models, which is a bit surprising because you'll see that the idea behind the Random Forest is rather stupid, but it works. So in machine learning, we don't really care why something works, it works, so we're happy and we'll use it. So let's have a look at an example for what that actually looks like. So when I used this example last year, I had to ask if anyone in the room actually familiar with baseball because I'm not at all. So some of you, probably the ones from the states, know more about baseball than I do. My complete knowledge about baseball from anime. So it's really skewed and not very accurate. So the idea is that this would be our data. We have salaries for people for for baseball players based on their hits and based on the years that they actually professional baseball players. And you can see that salary is coded blue or green. It's low and it's coded yellow or red if it's higher. So with naked eye we can already see we have a bit more red and yellow in this area. And we have a bit more blue and green in this area, but it's not completely consistent. If you actually have a look at the bottom here, you have quite a few blue points down here as well. So just using for example, a linear regression or something like that, which would draw a line through, it would not really capture all of these kind of dynamics in the model. So how would you try to divide that? What do you think? What would be a good a way to dividing that space? I can see hand movements. What do you think? Yeah. So I would probably say like draw a line through here. Does that sound good? And draw a line through here. So that we basically have three spaces. So that would look something like this, for example. And this is something you can achieve with a decision tree which you couldn't achieve with a linear regression for example. So. As I mentioned, decision trees basically use multiple decision rules and systems of decision rules to divide your data space into different regions. In this case, for example, we could say one wall is less than five years, everyone less than five years seems to have a relatively low salary. Apart from these kind of outliers at the bottom, which we'll conveniently ignore in this case because there are just two of them, so they are fully wrongly entered data or really weird players. I don't know how anyone could achieve that, so let's ignore them. Conveniently, let's say less than five years is probably one of the walls, and then maybe have another rule on the hit variable and do something like this, like more than 100 or 110. And you can actually then calculate where exactly these perfect Cut-Off values would lie to divide your space optimally. And you can see the result here on the right. And that is the idea of why we call this a tree. Now, this doesn't look very tree like yet because it's a really simple problem. But if you have more variables and you're actually continuing that, it's kind of growing. It's going from the top to the bottom. So we start by dividing the space on the variable year. So less than 4.5 years for example. And then we're going down. And either we are already predicting a salary if it's less than 4.5 years, or if it's more than 4.5 years. So on the right side then we are dividing that space even further. So you can see it's a really clever kind of idea behind that. The idea that sometimes it's really easy to get a value. So if it's less than 4.5 years, then we're getting all of these values and we're all assigning them the same salary 4.5 but in other spaces it's a bit more complicated. So we want another rule on the right side of our space, which in this case we achieve by saying if they have less than 117.5 hits, whatever that means, then they get a salary assigned six. And if it's more than that, they get the salary assigned 6.74. So this idea behind having these kind of. Stepwise splitting of the data is really the core principle of a decision tree. Then we also have a bit of lingo, a bit of language that we use for decision trees, because of course we do. We typically divide that between internal nodes and terminal nodes. So in this case you have two internal nodes which would be here and here. So there's basically your branches where the inside of the tree. And then you have your terminal nodes which you can also call a leaf if it's a tree. And these would be all of these kind of numbers here. So the very edges of your trees would be called the leaves. And then the number in each leaf is the mean response for the observation that falls there. So basically if you have a specific observation like a specific data point that you have somewhere here, let's take this for example. And you want to make a prediction for it. Then you would look first. Does it have more than 4.5 years. You would say yes. And then you can say does it have more than 100. What about that 117 hits? You would say no. So a prediction for this kind of small point here would be more than 4.5, but less than 117.56 would predict the value six for that specific data point. So each of these data points, you would then basically draw a route through that decision tree and look at which leaf you end up. Some leaves you end up earlier than others. So yes, that's how our decision tree actually segments our space. So you can see that we have three decision spaces. So that's basically three leaves three possible values that each of our data points can find. We have one. We have our two at the bottom here. And we have our three at the top. And these are basically exactly these values here. So this would be our one. Yes. This is our one. And then you have less than 117. That's the bottom here. That would be our two. And then this would be our three. So leaves is always the same number as your decision spaces. Or you predict the spaces. So yes, our regions are therefore called terminal nodes or leaves. And yes, decision trees are typically drawn upside down. So that's the main difference between a decision tree. In the real tree they grow from the ceiling which means the leaves at the bottom of the tree. Theoretically you could draw them the other way round I guess, but it's kind of intuitive to read from the top to the bottom in most languages, so it made sense to also draw out position trees that way. And yes, we always said that the points where the tree actually splits is referred as to an internal node, or kind of, I call it a branch, a branching. So that kind of feels more intuitive for me. But most people will say internal nodes. This is also kind of a nod to graph theory. So if you're familiar with graph based methods and graph theory, then you will be kind of familiar with all this language around nodes and edges and all of that. If the idea is that these regions are distinct and non overlapping. I think that is also a really important distinction to make. So I think we just really, really briefly touched upon that during our clustering discussion. But most methods that we actually discussing are kind of describing distinct groups. So as in there's one or the other label for you. We're not assigning kind of mixed labels to any data points. We're making a decision. There are methods which are more what we call fuzzy. So for example, there are fuzzy clustering algorithms where we don't assign a specific group or label to a point. We actually assign degrees of memberships to multiple groups in decision trees. We don't really do that. I'm sure that someone invented fuzzy decision trees. Someone invented everything. But in this case, let's say all of our groups are distinct non overlapping. So I don't degrease you in one leaf only. Each point belongs to one leaf. And yes, for every observation that falls into that, we make a prediction based on the mean value of the response values based on the training. So that makes sense. We build our tree with our training data, and then we flow in our test data and we look basically where it ends up. And yes, in theory, regions could have any shape. So typically what we do is we kind of follow these rectangular boxes because that's the easiest for simplicity interpretability, especially if you have a two dimensional space. So that's why we draw them. But decision trees can actually look for non-linear relationships as well. So you can kind of look for very flexible spaces in that it doesn't have to be this perfect box. It's mostly a box just for simplicity reasons, because it's the easiest to calculate, the easiest to draw. How do we actually find these boxes. So we are back to our minimisation of the error. So you remember RSS as our residual sum of squares. That's your error message. And you can calculate that for each of the boxes. For here you can actually look for for each value. How much does it actually deviate from the predicted value which is the mean response of the training observations in that space. So we only have one value for each of our boxes and for some of those values, for some of those points, the value will be more off than for others. And I think that's relatively clear. So the salary for each baseball player in that rectangular box is not the same. There will be variation, but the boxes are drawn in such a way that the minimum that that that this deviation is minimised. In other words, the real values and the predicted value for each of the box should be minimal in their deviation and that is just the error, basically. And yes, that's another thing. It's computationally infeasible to consider every possible partition because there's a lot of different partitions, right? So there's a lot of different spots that you can actually draw your line and use your. Yeah, used to divide your space. So what we typically do is something called recursive binary splitting. So we start at the top with our first internal node. And then we think what is the best possible split. At this particular point in time. We ignore the future. We ignore the past. We just think, okay, if this is our space, where should we draw the line to actually. Best minimise our our error for this particular step in time. We call this a greedy approach because we only consider this one step. We don't think about whether it might actually be better to first divide it on a different variable, or add a different kind of value, or anything like that. We just say, okay, what's the first value? We first look at time spent as a baseball player, and then we look at hits made by that particular player. For example, we don't think about the future, we just always think about our current step. Um. That can be kind of a bad thing, honestly, because it can mean that if you don't think about what step would have been better in the future, you actually often end up splitting multiple times on the same variable, for example. So you might split once on a variable very early on, and then in a different branch, split on the same variable again, and you do it again later on in a different branch. So it's kind of ineffective, but it works because it's really quick to do because you don't consider the history and you don't consider the future. Just think best possible decision at this particular point in time. So what does it look like? We have actually a beautiful five step process here where we first select a predictor that's for example time spent. And then the cut point I think we said like 4.7 years or something. So that the splitting of the predictor space leads to the greatest possible reduction in our error. So that is what you just said. We want to split the space in such a way that the points in each of these space, in each of these spaces are most close, or the closest possible to the predicted value in that space, and then we do the same process again. So we think, okay, which variable should we take next. Which values should we take next for that particular variable. And split it further and further stepwise. And we always try to minimise kind of the error between our actual values and our predicted values. And yes, obviously, instead of splitting the entire space, we kind of continue to split our existing spaces. So we first split by age and then we have, for example, our new region. Here. If we first do our split top to bottom, then we think, okay, we have these two spaces here, left and right. How do we best split them? Oh, let's split the right one and we split the right one through the middle here. So we kind of do this recursive step by step splitting off our spaces. And again looking to minimise your RSS. And then you continue that process step by step by step until you reach some kind of stopping criterion, for example. Or for example, you may continue until none of the regions has more than five observations. So there's different ways of deciding when to stop. And that really depends on how far you want to go. A big danger with decision trees is actually that they tend to overfit quite quickly and quite strongly to their data, so decision trees are absolutely prone to overfitting. And you can use the stopping criterion. For example, how much reduction hours do you actually want to achieve with that next split to decide, okay, that reduction is not enough to be worth it or the spaces are getting too small. I want to stop. Yeah. We always said that we predicting for each of the responses of a test observation. Then by using the mean of the training observations in that region. So it's this idea of points in the same region as similar. So we kind of voting all the points of voting together. What should be the value for our test. We just said trees are really prone to overfitting, so there's different ways of avoiding that. We said stopping criterion, and we started checking how many points you have in each leaf. But another way of actually achieving that would be to prune a tree. So yes, we're all using all of this gardening terminology for decision trees. It's very cute. So we said a smaller tree with fewer splits might lead to lower variance and better interpretation at the cost of a little bias. That is, this variance bias trade off thing that we were talking about. So we want to avoid overfitting to training data, even if that means that we are a little less accurate maybe on the training data, but at least we can then accurately predict for our test data in the future. So you can, for example, grow the tree only so long until you reach some kind of decrease which is too small and but yes, this is very true. It is kind of shortsighted because we already said each of these splits is just doing thinking about a split, not about a feature and not about a past. So there might be a really, really good split coming soon, which would be really good for reduction. But because it's two steps ahead, you don't see it, so you stop too early basically in your space in that case. So in that case, what you can do as an alternative is grow a really, really large tree. So go all the way, overfit terribly to your data, and then prune it back. So that also comes back to the idea of graph theory where you think about graph based models. So if you've ever done any graph theory in your undergrad, you might know that, well, you're looking basically for the weakest edge or the weakest link in this case and prune that. So the idea here would be you look at different trees. And you're looking for the smallest possible tree within that which is still doing a good job. So you're basically thinking, okay, this tree is growing all of these directions, but I'm only really interested in this space here. This is kind of overfitting and doing a weird job. So we're only interested in the sub tree of our decision tree and we're pruning off the other edges, which are not so good. And the idea here is that you have a tuning parameter which you can do and use. You would use, you would choose that optimal value of that pruning or tuning parameter using cross-validation. And then you would actually find that optimal sub tree, which is still giving you a really good value of RSS. So a really good value of reduction of error but has a little has a little less complexity and it's a little smaller than that. Yes. Quick summary. I think decision trees are relatively easy to understand. That's why I mentioned I enjoy teaching them because they're not difficult. And for in this case, for example, what we would suggest is you can use recursive binary splitting. Remember this is a stepwise splitting into always two regions only looking at the best possible step. Look, build a really, really large tree on all of your data and all your training data. You stop when each of these leaves has some kind of minimum number of observations. I think earlier we said five depends on how large the data set is really. And then you use some kind of cost complexity pruning where you're looking at a subtree which is still doing a really good job as a function of alpha. Use cross-validation to choose that parameter alpha. ET cetera, et cetera. And you return a subtree which actually does still a good job, but is smaller than the original and therefore less likely to overfit. So pruning trees is relatively simple, as in you basically. You've seen this before. Let's put it that way. You seen a very similar approach before when we were talking about regularisation for regression. So if you think back to your lasso regression, for example, we also had this tuning parameter where we basically having to tune based on the complexity of the model. In a linear regression, the complexity of your model is the number of predictors or the number of variables that you had. So you wanted to reduce how many variables you had in your regression. And here what you basically want to do is the size of the tree. You want to reduce the size of the tree while still optimising how well it kind of fits the data. You want to optimise the error and reduce the tree. So it's basically pruning is very similar to regularisation in regression. So these are very similar concepts to each other. Here we have some beautiful pictures of our baseball example. Continued I don't know why there's a baseball example in there. Honestly, I think I should I should change the context because I have no idea what I'm talking about. Okay, we have all of these variables on the left which I can't interpret, but you will remember that basically part of the three is what we had earlier, right? We had this year split this internal note up here, and we had this internal node down here. But now we've actually built the tree further. So we've done this build a huge tree thing that we were talking about earlier. And you can see that we have more splits than here. I have no idea what RB is. What is. Huh? Like after you. We kind of went there. Ah, so it's running time. I mean, this is a really bad example in that case, isn't it? Because some some players are better at some things than others. Interesting. Okay. So. Yes, that what I wanted to point out. I think this is a good graph, as you can see years again down here. Right. So you can see that the variables are actually used multiple times in the same tree. And that's what makes a decision tree a little bit inefficient sometimes because you're basically you're always only thinking what is the best variable to split on. And it can be the same variable multiple times. And what that tells you. Typically, if a variable is mentioned multiple times that it's a really good predictor variable. So the model really likes it. So that's why it's splitting multiple times on it. And you can also see in our kind of baseball our example and our scatterplot earlier uses a good kind of predictor variable for salary in our plot as well. So that's why it's splitting multiple times on it. Sometimes even on the same value you can see here is less than 3.5. And years less than 3.5. So it's actually doing the same split but on slightly different groups. So that's why it's kind of a bit inefficient maybe, but it does a good enough job in our relatively simple example here on the right hand side, you can also see how the tree grows in size and actually improves in error. So you can see here that the tree size would be the number of internal nodes. And you can see how the tree first reduces the error on all of our different data sets. So training cross validation and test data. And then actually we have a bit of an increase again in our error specifically on our test data and on our cross validation data. So this is overfitting. Visualised. You can see that the error on the training data is still reducing. So we are still improving our fit of the tree to the training data. But actually our error on the test data and on the cross validation data. So what we mean by that is basically remember in cross validation where we kind of splitting our data set and we're training the model on multiple of those sub data sets. And then comparing that that is our cross validation error. So you can see that this is actually increasing. And the test data that is the one that you reporting is also also increasing actually because it's overfitting to training data. So you might say for example this here is actually our sweet spot. So here you actually have the minimal the minimal the minimal error on the test data. You might argue that this is similarly small, but it's kind of really increasing in complexity. And you also have a really high cross validation error. So be careful about this kind of tree here. I think the sweet spot that we're looking for is this three splits which would be years, RBI and hits. And then you're done. So this is kind of a tree that you're looking for small computationally efficient and does a good enough job. So what we've just talked about is regression trees. So we were predicting some kind of continuous number outcome for example salary. But obviously you can also use trees for classification tasks. And they work very similar to that. So in this case what we do is each observation is predicted to belong a specific class, specifically the most commonly occurring class in that training data. In that training data in that specific kind of prediction region. So again, really similar to K. And from its theory you divide the space into your kind of subspaces. You predict the spaces. And then you look for how do all of these data points vote. Which class do they belong to. That's the class that your test data point in that space also should belong to. Because they are similar, they are close to each other. We all voting. So it's basically the same thing. There are some differences between classification and regression trees, obviously, because for example, we can't use RSS, we can't use the error as a measure for the quality of our tree. Because you can't calculate an error with a class label. The class label is not meaningful in that sense. So instead what we look for is a classification error rate. So this is very similar to what we were talking about in logistic regression for example. So basically look for the fraction of the training observations in the region that do not belong to the most common class. So if you have 20 points and they're all voting for example, 15 of those are voting red and then five of them are voting blue. Then the prediction would be red. But you still want to know how many of them actually voted differently. So that was basically your classification error rate. And in practice that does not really work so well for trick growing. So what we actually use is typically the Gini index or the node purity index. So here we are still looking for a very similar concept. So you can still see we said earlier here P hat MK is the proportion of training observations in the mth region from the K class. So we are still looking for that value. So we're still interested in kind of how are each of these points voting in our space. But now we're looking for the total variance across all of the different classes. So we're basically optimising how evenly or how. How consistently are each of these points voting the correct class that they should belong to? So how much kind of purity is in those notes? So we say not purity as in if you have 20 nodes voting and 15 say red five vote blue, then what is the kind of the purity of that vote result? And the idea is that. The it's better that Gini index kind of the more united our points in their voting decisions. Yes, it sounds very much like voting as in political voting. I know it's odd. So now turning to the Gini index would be cross entropy. The only difference is that you basically take the logarithm here. So you can see it's really similar from the calculation. You're still interested in kind of how are each of these points voting per class. Now you take the logarithm of that. You still sum it all up over the whole possible class that you have. And the cross entropy is negative. And to the surprise of no one, if you look at that similar calculations, they are similar numerically. So typically if you calculate Gini index and cross entropy, they will tell you the same thing. So they will both kind of agree on the node purity in that specific node and which one you choose is completely up to you. So there is no major difference between the two. It's a matter of preference and in most cases they agree anyway, so it doesn't really matter. So here we actually have a bit of an example for that, which in this case is not baseball but tourism. So that's a bit closer to what I actually do. So we have five tourists coming to Edinburgh apparently. So we're asking all of them, have they visited a castle? Have they bought a kilt. And what is the number of whisky whisky bottles bought? And what we're trying to predict is whether they are tourist or not. So these are all apparently very touristy behaviours, and we're interested in predicting whether they are tourists or not. So what we now do is basically we would calculate the Gini index or the node purity of our two classes. So in this case for example we have the two classes either tourist or not tourist which is here our last column. And then we can calculate the Gini index for both of these classes. So our Gini index for the class tourist would be three over five. And our Gini index for the class not tourist would be two over five. So we then calculate what we call the Gini index prior to splitting. So at the moment all of our data is split and we are calculating our Gini index as two times our our kind of node purity here for tourist times. No purity for not tourist. And this gives us a Gini index of 0.48 for the whole data set prior to splitting. The reason why we do that is remember what we're trying to do is split in such a way that we improving our Gini index. So we need some kind of baseline value. This is our baseline value for the whole thing. Now we think okay which of our variables should be split on. So what would be our first variable to split into two spaces. And we check first for example the variable. And by boat or not boat. Yes. So that is variable B both are killed or did not buy a kill yet. And what you basically do is look again for all of our tourists for example, which also bought something, all of our non tourists which did not buy something. And then also for example, the differences between that. So he is a tourist which who did not buy a kilt. So that would be for example. Exactly. This is basically creating non purity in your notes because you have two tourists here. So they're three and four but they have different behaviours. So in that case we might say okay maybe that is kind of impacting how well we can split on that variable. So that's why we that's basically what we are calculating with our Gini index. So we have our p t tourist. And we calculated as both as a tourist and not as a tourist times two. And then we do the same thing for our Gini index for non tourists. So we have bought something, are not a tourist and not bought something and not a tourist. And this all gives us a Gini index of 0.266. So we have a reduction in the index from 0.4820.266, which means that by splitting on that, that is basically improvement we can make in node purity in our in our nodes. So you can then compare that to our different variables and that variable which gives you the most improvement. The most reduction in Gini index is the one that you're splitting on at that specific point in time. So remember what we said earlier. This kind of splitting is greedy. So we're only ever checking each of these variables one at a time. And then we kind of ignoring the past ignoring the future. We're just looking for the best Gini index reduction right now. So yes, this is basically how you would do the same thing for a numeric value. So you can actually use RSS in this case because bottles of whisky bought can actually be used. I will briefly skip that because I think you'll look tired. So let's do our ten minute break instead, and then come back and finish our discussion about decision trees. Okay, let's finish up our talking about decision trees. So what we've just spent the last couple of minutes on before the break was to think about how do we choose the best variable to split on. So if you think back to this tree example here, basically what we're trying to find out is which variable should we choose for the actual splitting. And the best variable to do our splitting on is the variable that actually best divides our space. So you'll remember that we had this example at the very beginning where we said, okay, age seems a really good variable to split on because it's kind of defining creating these two groups which behave quite similarly to each other. And this is exactly the same thing which we also tried to do here. The only difference is that how well our model performs is now measured differently. So we're not trying to predict salary. We're predicting just a label. So that's why our calculation is different. But the principle is still the same. So we are still trying to create what we call pure nodes. And when we say pure nodes, what we mean is we basically want all tourists to be in one leaf and all non tourists to be in one leaf. So that would be the perfect split. So if you have this kind of split for example if we said both are killed, let's imagine both are killed as a perfect variable. Then all tourists would buy or killed and no non tourist would buy a killed. So that would make it a perfect splitting variable, because then if you look at whether they both are killed or not, is is a perfect predictor for them being a tourist or not. But that doesn't really happen. So no splitting variable will be perfect. And we therefore calculate this Gini index as a kind of measure of how pure is the other resulting groups. So how well does the split actually work. And that's why we look at these kind of proportions here. So the number of tourists the number of non tourists. And after the split in our kind of two branches of that split. And we do that with the Gini index because we can't really calculate an error for a label. So bought a kilt or not is a binary label. So we can't really calculate an error measure for that. But we could if we have a numeric variable. So you can see the number of bottles of whisky bought for example is a numeric variable. And if you have those then you can actually calculate the error. And in that case it would basically be how well is the bottles of number of bottles of whisky bought a predictor for them being tourists or not. So the idea is for example, tourists buy more bottles of whisky than non tourists. So you look at the number of bottles bought by tourists and the number of bottles bought by non tourists, and you want to kind of maximise the difference between the two. So what does it look like in a numeric example. So you can see here at the top you can see what we have. For example we say split the bottles as kind of the value two. So the algorithm will actually choose the best possible value that we would split. So let's say we choose the value two here. So we're interested in whether someone bought more than two bottles of whisky or not in trying to predict whether they are tourists or not. So if we have the split it's a binary split. So either more than two or not more than two, we can see that we have two resulting regions. You remember that was kind of the splitting of our space into two regions. And we would calculate the medium or the mean value of whisky bottles bought in that group. So in this case, for example, we have space the space number one which is visitors one and two. Let me check. To. Ha ha ha ha ha! Interesting. Ah, here we go. Now. I'm cleverer than before. So we have visitor two and visitor five are buying less or buying less than two bottles of whisky. And visitors are one, three and four are buying more than two bottles of whisky. So that's how we divide them. There we go. There we have group number one. That's visitors two and five. They buy one bottle and two bottles respectively. And we have the second space are two, which are visitors one, three and four. And they are buying four, three and four bottles respectively. So that's how we divide our two spaces. And then we calculate the mean number of bottles bought per that space. And then you basically calculate your error for each of these. So if someone bought one bottle, then how much is the error from the predicted value for them. So here visitor two actually just bought one bottle of whisky. But we would have predicted 1.5 bottles of whisky. So the error is one -1.5 squared. And then we have the next visitor that is visitor number five. They bought two bottles of whisky. We would have predicted 1.5. So the error for that is two -1.5 squared. So all of that is basically our error in predicting whether how many bottles of whisky these different people bought. And before doing any kind of split after splitting, here we go after splitting for larger than two. We kind of do the same for different values of two. So here for example for different values of bottle. Gosh okay. For the value of how many bottles per bottle. So, for example, if we split by bottles larger than three, we would do the same thing. We'd look at how many people bought more than three and less than three. So now only two people actually bought more than that. Three people bought less than that. And we would still calculate our error in predicting how many bottles they actually bought. And we can see the error here is two. The hour earlier was 1.167. So splitting bottles more than two is better because produces a smaller error in our prediction of bottles bought for that group. So the key idea I would like to take away from this is. Tree based methods. Basically, you have regression trees and classification trees and they work very similar as in the divided space they predict by most frequent value or by mean value of prediction in that particular space for your test data. And the main difference between them is one how they find these best splits. Because that depends on whether you have different classes that you're trying to predict, or whether you have a specific value that you're trying to predict. And then they also depend on obviously, how that variable they are splitting on is coded. So we've seen numeric variables where we can use the reduction in error. And we have seen classification or class labels categorical variables where the reduction would use the Gini index where we're looking at the reduction in impurity in the resulting classes. But the basic principle between these two is very similar. So we will have a couple of minutes. Let's do just five maybe today because it's not too much. And I do want to talk about begging next, but do have a quick discussion about decision trees, what you learned about them so far. And then we will discuss them and collect those notes that you make. So do remember to take notes during those discussions as well, because that is obviously not on the slides. And you might find that useful. Good. But. You. Think. I get. This. I got this. Okay. Are we ready to go? Who would like to start today? Yes. It's a very. Intuitive visualised for you. This show is visualiser full of words. No, I invented it. Can be visualised. There we go. So yes, absolutely. So we talked about intuitive methods with k and n and linear regression as well. So I think one of the advantages of decision trees is you can explain them pretty easily to most people at least kind of the basic principle. It's basically just a set of decision rules. So you just have a set of binary decision was either larger not larger than this value or it's either this value, it's not that class. And then you kind of go down this tree so it can be explained relatively easily, which makes it really easy to use in application contexts where you have to explain not only the result, but you can also explain how you've achieved that result. Anyone else? Yes it is. Yes. So it is relatively. Insensitive to outliers. Insensitive? You do know what I mean? In, right? Yeah. Insensitive. Feels like it's about kind of being insensitive on the feelings about that, but no. So what we mean by that? Yes. It's compared to other methods relatively insensitive to outliers. That being said, that does depend a little bit on kind of a decision rules that are being used. So because we have. This kind of greedy approach where you can use the same, the same variable, for example, multiple times, and all of it can lead to being a bit not sensitive to outlier points, but kind of to very dominant variables, if that makes sense. So there's also a type of sensitivity to outliers in a way. So if you have one one variable, one feature which is really kind of dominating the space, it will just be used over and over and over and again. And you're not really learning much about other variables, which can be a bit of a disadvantage in that. But you're right that it's relatively insensitive to kind of point outliers. Yeah. So I added that disadvantage I think can be sensitive to dominant variables. And what I mean by that is, for example, if one variable is kind of really good at explaining a relationship, then it will just be used multiple times, which can be fine. It doesn't really. It's not really a problem. It's still doing a good predictive job, but you kind of dismissing maybe a slightly weaker but still important link, which can which can impact your interpretation of the results, for example. Anyone else. Yes, it can be applied on both qualitative and quantitative, and sometimes we don't have to use dummy variables. It's really kind of flexible regarding data type. So the great thing about it is, for example. No standardisation y no standardisation because each variable is looked at separately, so on its own in each of these splitting steps. So what that means is basically, even if the variables are measured on really different scales, it doesn't really matter, because if you look at them separately at not comparably, then it's not impacting the model. So no need for standardisation, which is quite handy. And yes, can use, as we've seen in classification classes, ordinal data, numeric data for everything in there. It'll it'll do its job. Anything else? Yes. It's short sighted. Yes. It's kind of shortsighted. That's absolutely right. So which is basically what I just said in negative terms. So we only ever. We only ever look at what is good and this specific moment. Split wise, we don't think about anything else. It's pretty short sighted. Yes. It is hard to believe in small variation. Can you? Yeah, that happens a lot. Yeah, it's it's interesting because on the one hand we can we say they are relatively insensitive to outliers ish. On the other hand you can get vastly different trees really quickly. So I think they are kind of. Can have a lot of variation, maybe. And what we mean by that is if you have even if you have the same data, you get basically different ways. Many times if you try the algorithm multiple times, because it just depends on what kind of variables it's looking at in that particular moment, for example. So sometimes if you have, for example, a data set where you're doing cross validation and then you're looking you're building a model on the first part and using different variables in the first split, then on the second cross validation set for example. Then the trees will be completely different because everything is kind of impacted by this first split. So earlier when we had a baseball example, for example, we had our age was a really good predictor variable. If we looked at the whole data set that we had. But if you then, for example, think about, okay, if we do some kind of population just happens to have training data with more data from the right, then suddenly the age wouldn't have been a really good predictor for that. So we wouldn't have done this first split which impacts the whole tree. So the whole tree would look different in that case. Does that mean that it can overfit the model because. Yeah, trees are really prone to overfitting. So the trees that we've looked so far at these kind of single decision trees, one tree as your model, one tree for your decision making, prone to overfitting. Really prone to that. So be careful if you're using a single decision tree. We've talked a lot about kind of ways of overcoming that. So we talked about looking for minimum number of observations in leaves. We've talked about early stopping where we're kind of stopping when there's not enough reduction in Gini impurity or RSS. And we talked about pruning our trees because yes, overfitting. Anything else to add? Would you like to move on? Yes. Yes. I think missing data. Yeah. It depends. It depends on where the missing data is. So if it's kind of in the most important considered variable, then in that case it can kind of struggle a little bit with that. But it's not a fundamental problem as in it's not a problem for the model itself. So it's not kind of impacting the validity or the stability of the model. It might impact the predictive power obviously. So there are other methods like neural networks which really struggle with missing data. So that kind of impacts how much pre-processing you have to do. For example, the neural networks tend to need more pre-processing than decision trees. Decision trees are kind of take your data and you throw it in there and you hope something kind of useful comes out. Okay, let's move on. So so far we've talked about decision trees and we said okay, decision trees are kind of cool. We can like visualise them. They're intuitive. They work reasonably well. But. Actually they are not really good predictors in some cases. So the idea was okay if one tree isn't really clever, what happens if we use more trees? Surely that's the solution and the answer to all of our plays and questions. And turns out, yeah, it's kind of the solution, even if it sounds a bit stupid. So we will talk now about general approaches to combining multiple weak learners to overall improve your predictive power of your model. So we can also call these groups of learners and ensemble a bit like you have for example, musicians coming together and a music ensemble. So therefore we also call this approach ensemble learning. So instead of just using singular models we're using groups of models. And two approaches to that are bagging and random forests. So the idea behind a lot of that actually comes back to what we talked about in lecture four. So in lecture four we were talking about this idea of resampling and bootstrapping and all of that kind of how do we create multiple training data sets to actually kind of train our models in a really robust way. So you remember that bootstrapping was the idea that we create training data sets by sampling repeatedly from our kind of pool of possible data, and we do so with replacement. So you remember when when you learned probably about statistics and probability, you had these kind of balls with kind of coloured balls, and you were taking them out and then recording and putting it back and taking one out again. Recording that is bootstrapping, just like really fast. So you take records out and put them into your training data set and you put them back again, and then you take them out and put the same record back again. So multiple records can be the same in your training data set. It can appear multiple times. That's the big difference to cross-validation. So you were k fold cross-validation. We split our training data. And these are the splits that we keep. And then we train on a part of that. You put it back and you train on a part of that. And you put it back with bootstrapping because you always kind of replace these, these samples. You can train multiple times on the same records. The idea of bootstrapping can be used for model training as well, and if we do that, we would call that bagging. So bagging actually stands for bootstrap aggregation. I always thought it was it meant that we take a back and we throw in multiple models and we mix it up a little bit. That's how I remember it. But no, actually it stands for bootstrap aggregation. So even though we talk about bagging in the context of decision trees, it can be used for different models and different learning processes because it's just an idea, a kind of general approach to how we build models or how we use models. So we call it an ensemble model because it combines multiple weak learners and it can be applied to different models, not just decision trees. So the idea behind bagging is that we want to introduce elements of randomness. So randomness is actually your friend because randomness can improve how reliable your models are performing on your data. So in this case for example, here we say we have multiple observations and independent observations. That's just your data. And you have some kind of variance sigma. So the variance of the mean of those observations can then be calculated obviously as sigma squared over n. So in other words if we average a set of observations we can reduce the variance. The idea behind that is basically if you think about back to principles of data analytics we were looking for, how do we predict the mean of a distribution. For example, if you collect more and more data and you have, for example, a lot of kind of distributions which are overlapping, then at some point you get a pretty good predictor of what the true mean of your population is. If you sample repeatedly and look at the mean of all of these different distributions. So this is basically the idea, but we don't really have these different samples and these multiple different training data sets that we can use to predict for example the mean of distribution in the principal's lecture. So the idea is that we use bootstrapping, which we talked about earlier. And we take repeated samples with replacement from this one training data set we have. And that way we create multiple bootstrapped training data sets. And then we train our method on each of those separately. And we average all of our predictions. And that would be then our overall prediction for in the example I mentioned earlier for the mean. But really what we're doing here is predict a specific value, for example for a decision tree. So the idea is if by building multiple models on multiple subsets of your data, you kind of building a more robust model. And because you're basically reducing the variance in that, and this idea of throwing together multiple models and then averaging their prediction, that's what we call bagging. And that's what we can use for different model approaches. Because you can basically combine multiple learners of any structure. Doesn't have to be a tree. So the prediction at this point could be a numeric prediction value. And but you can also use it for classification trees. So it can be a class label as well. And the class label for this point is then decided by a majority vote. So all of these different methods, all of these different models are basically averaged. And either the predicted the predicted value is averaged or it's the most frequent vote. Yes. So these three ensembles, what you really have to do is you have to decide how many trees do you want to combine, how many bootstrapped samples do you want to use, and then train your trees on and then combine all of your averages. So that is the parameter b that you can see here in the formula. So how many do we want to combine? Simply put, the more you actually use, the more accurate it gets. Even though that's kind of a bit of a levelling off. So at some point you don't really get more improvement. But obviously the longer the computation time, it's the same. The same principle as did you always have more trees, more samples, better accuracy, more computation time. So you have to choose basically the number of trees based on the data complexity, more variation and complex data, more trees, necessary and simpler problems. Less trees necessary. So a really interesting point that I mentioned earlier, which you might have caught, but you might have not. As I said, we combine weak learners. And what I mean by that is the trees themselves. They are not really good. So in this case, for example, we grow them really deep. We don't prune them. They have a lot of overfitted trees, but we have a lot of different overfitted trace. So all of them are kind of trained independently on subsets of the data of kind of bootstrap samples of the data. And that means a single tree terrible, vastly overfits the data. But if you combine all of them and average all of them, they are doing a pretty good job. So this is the idea of bagging basically taking a lot of bad models per definition, but by averaging them you get a pretty good prediction result. So in other words, each individual tree has a high variance and a low bias to overfit. But by combining them, we said we reduce the variance in the model. Yes. So this is relatively straightforward. Obviously the test of a bagged model. And the idea is that these trees are repeatedly fit to bootstrap subsets of our data. So we can actually show that on average, each of these factories uses roughly two thirds of the observations. So there's a mathematical proof in there somewhere that I'm not going to show you because it's very boring, but it is there. So the remaining one third of the observations for each of these trees are basically not used. And you would use that as the so called out of back observations. That's another reason why I thought back as the bag bagging I don't know, it's really silly. And so what does all of that this slide means is for each of these trees there's a different test data set. Basically because each of these trees is using a different part of the data of different subsets of data. And you have to make sure that you actually test and predict out of back out of sample for that particular tree. And then you yield around B over three predictions and we average them. So this is all that this tells you is make sure to always use your test data for measuring your accuracy. And then we average that accuracy over multiple models. Okay, so this idea of if we have a lot of very bad overfitted trees and together they can achieve something great. When I first heard that. When I learned that, and then I read about random forests, there's one thing that came to my mind, and it was this you all watched Lord of the rings, right? The idea that these trees come together, and then they basically make a decision together. And each individual tree, I mean, in a lot of things, they're really clever, but they're not really good, are they? So the hope is that by combining all of these trees in their decision making process, that we can actually achieve a good prediction. And that's the idea of random forests. So if you hear random forest, you kind of can think about the ends in a lot of rings. That's pretty accurate. So the idea behind Random Forests builds on back backtrace. So if you understand backtrace, you understand random forests. The main trick that we do now is that we try to correlate the trace. So we said randomness is good. So we don't want kind of connections and correlations between our models. We want as much randomness in that model as possible because it improves our predictive power. So same thing as before. We still build a number of trees on our bootstrap training sample. But this time, instead of just building these trees the same way. So we kind of give them the data and we tell them, you do your splitting and use Gini or whatever. We now actually tell them, okay, you do your splitting, but you only use the subset of variables that I give you. You can't use all of them, just this random subset of them. And what that does is it forces the trees to kind of change their behaviour. Now suddenly not each of these trees can be built the same because there are variables that they have to use differs. So this actually just added randomness of the predictors that trees can use introduces more randomness to your model, which improves your prediction power in many cases. And yes, the first election of predictors is then taken at each step. So you introduce more and more randomness at each step of the model building. You remember earlier when we said age, it's a really good kind of variable to split on. So probably each of these trees will split on age. But what will happen if we actually tell them, okay, you can't use age as a variable. You have your force to use a different subset of variables. So now suddenly all of these trees are different from each other, and they all tell you something slightly different about your data space. This is the idea behind random forests. So back trees, multiple trees average their performance pretty good. Random forests, multiple random trees with random splits at each point and then average their performance. And this kind of added randomness at each split. And each step that you take leads. That leads to random forests being kind of one of the most powerful predictors and powerful classifiers that you can encounter in many cases. Now, I say that with caution, because that does not mean that random forest is always the best model you can possibly use. It means that in many cases in the literature, when you see comparisons between models in the social science, random forests outperform other methods. And by that I mean they perform like regression and k and all of that. That does not mean that random forest is always the best model. We always we already said that best model depends on your data and your context, what you're trying to achieve. But I always say that if you have very complex data and you're not at least trying a random forest, then you're kind of, I think you're missing out a little bit. So one more thing I wanted to say about my invoice, which I didn't put here, is. Random forests and begging. And all of these approaches have one major disadvantage, and that is interpretability. So we already said we have all of these trees and we're kind of putting them together and they're voting on something. But you lose all the interpretability that you had from your original decision trees. So it's kind of this pretty tree structure. And you could see exactly which each of these points take gone. So you have all of these trees, hundreds, maybe thousands. It's impossible to visualise and it's extremely difficult to interpret. So that's another point where we say when it was a good predictors, but they are not very good at interpretation. So there are ways kind of overcoming that a little bit. You can look at feature importance for example, which is a measure of how important was a variable for for the model in its predictive power. But even that is kind of a weak comparison to having a perfect tree, which tells you exactly what it's doing. So yes, it depends on whether Random Forest is a white model for you. Vastly depends on whether you value interpretability as well. Now let's spend the last five minutes to briefly talk about boosting. And boosting is also a general approach, so just like bagging, it can be applied not just to race but to other methods as well. And you remember that bagging kind of involves multiple trees and then multiple copies of the original training data using bootstrapping, fitting them separately, combining all the trees into a bag, shaking it, looking at the average accuracy. So each of these trees is built on the data separately from each other. We said that can lead to the trees looking very similar and being kind of correlated, which is why development of forests and boosting actually takes that idea into a different direction. So it says, okay, but what if we not build them separately but sequentially? What if we take a decision tree and it does kind of a decent job, but it misses some information in our space. But if we then build another tree on top of that, which kind of covers this missed space where the tree wasn't performing so well, that is the idea behind boosting. So let's let's look at this because this is much more interesting. The intuition behind boosting is really that instead of just fitting one tree or a group of trees, we kind of sequentially, slowly improving our learning step by step. So giving a current model we first fit a residual. We first fit just a decision tree. So just our normal model approach. But then we fit a decision tree to the residuals. So that's the error of your original tree. And then this is added to kind of the fitted function. So the idea is that each of these trees is really tiny. So it's more like like a little part little kind of cluster of leaves and nodes which is added and stuck on to your existing tree whenever there is kind of missed space. So we fitted the residuals because the residuals are our error in prediction. That's what we miss. That's what we kind of not capture in our model. And we kind of fit these two small trees to actually improve exactly where we're missing something. So if I remember back, for example, to our example from the very, very, very beginning, we had this example, we said we can build a tree to kind of divide that into these three spaces. But you can see that we actually miss, for example, these points up here and these points down here. So the idea would be behind boosting that. We say these points are important to us. And there's an error in the model. So we kind of build on mini tree just for kind of subdividing these small spaces even further. The idea behind boosting is really that if we value kind of these small complexities in the data, if you have a lot of complexity that you value and that you want to capture in your model and your model is not capturing yet, you can use this boosting in addition to your existing model to further improve it. For those missed small weird data spaces. Okay. Almost done. Summary. So we've learned decision trees are great. You can interpret and visualise them. And they're really kind of cool because they work on a lot of data. We also learned the kind of may a little bit vague sometimes. So combining trees vastly improves their performance in most cases, not all, most cases. The number of trees considered in the example should be adapted, obviously, to your problem and your data complexity. More complexity, more trees. The disadvantage of that is lack of interpretability. That's the major disadvantage of, for example, wind and forests and basically bagging approaches in general, because instead of being able to interpret our single tree, we have to rely on things like feature importance scores. ET cetera. So in the computer lab we will look at implementation of these decision trees, single decision trees and assemble methods as well. And this will also give us an opportunity to talk about these methods these parameters and test them out. So how many trees should we use. How deep should the trees should be built. ET cetera. ET cetera. So that will all be covered in your computer labs. And next week we will then talk about support vector machines. Okay. Have a nice week and see you tomorrow.