**078 - Overfitting (DT Part 2)**

**Matthias:** Welcome back. In this lecture we'll talk about overfitting and underfitting, two terms that will in the future be the bane of your existence and haunt your sleepless nights. Well, not exactly, but they are important problems that you're dealing with when working with machine learning tasks. What are they? Overfitting happens when the model matches noise in your training data, underfitting happens when the model cannot match your training data.

[00:00:33] Well, that's perhaps not too informative. Let's look at some concrete examples, and let's first look at two related concepts, that might be a little bit easier to understand at first: bias and variance or the bias variance trade off.

[00:00:50] What are bias errors? Bias errors are errors that come from bad assumptions about your model or about your learning algorithm. These are kind of systematic errors where the structure of your solution does not fit the problem. And this leads to under fitting.

[00:01:10] Variance is the reaction to small fluctuations in the training data. So if either your training algorithm or your model is very sensitive to small fluctuations in the training data, you high have high variance, and this leads to over-fitting.

[00:01:28] Pedro Domingos a well-known professor for machine learning from the United States has expressed it very succinctly: bias, measures the tendency of a system to consistently learn the wrong things. Variance measures its tendency to learn irrelevant details. And that's a very good description of these two concepts.

[00:01:53] Let's look at them in a very simple setting. Let's say we are target shooting, and we want to explore what bias and variance are in this setting.

[00:02:06]In the first picture, you have a case where you're a shooter who has very low bias and very low variance. You want to hit the bull's eye and with every shot, without fail, you hit the bull's eye. You have neither a systematic error nor a random error in your shooting. You are constantly hitting the target you want to hit.

[00:02:31] In the second picture, you have a low bias, but a much higher variance. And as you can see, this means that your impacts are centered around the value that you want to hit. But with each individual shot, you don't hit the bull's eye. You hit either to the left or to the right or too high or too low. But if you look at the average of your shots, they still hit the bullseye. So on average you're doing well, but unfortunately, each individual shot is very far off the target.

[00:03:09] might be another error that you are making, and that is a bias error. So if you have a high bias, but a low variance, then you get the situation shown in the third image. In the third image, you're still very consistently hitting the same spot all the time. Unfortunately, this spot is not in the center of the target where you want to hit, but it's way off in the upper left corner. And that is a case where we have a high bias and a low variance. So you have a systematic error, you are shooting at the wrong point, but you have very little random error, so you're consistently hitting the same spot.

[00:03:51] Of course you can have both systematic errors and random errors, and then you get the situation in the rightmost pictures where we have high bias and high variance.

[00:04:04]Now in this simple scenario, it doesn't look as if there's any trade of between bias and variance. There are just two different kinds of errors that you make, but it's not clear that if you reduce your variance, you will increase your bias or anything like that. But let's look at the case of machine learning and why this might be different.

[00:04:28] We've already seen this picture, and we said that most machine learning problems consist of extracting information from data where that data is noisy. We'll have to talk about what noise actually means and what the reasons for this noise are because different kinds of noise require you to take different measures against it.

[00:04:52] But let's say we have measurements from a sensor and the sensor has some fluctuation in the values that it returns. And so it's purely statistical noise that distorts a signal that would be precise if we could only measure with a better sensor and then we can split our signal into two parts.

[00:05:14]One is the smooth curve that we want to approximate, the other is this pure noise component that we don't want to approximate. Putting them together in a single image, we see that this is the situation that we find ourselves in. We want to approximate this red line, and this is made difficult because our signal is actually jumping all around the place and moving around the true input value.

[00:05:44]How does this situation lead to a trade off between bias and variance? Well, the problem is that we don't really know the shape of the function that we're trying to match, in many cases. And let's visualize this with our well-known example of the fun function that I've randomized with a normal distribution, with a standard deviation two, and then I'm training a linear regression and a decision tree regression on this data, and I'm plotting both of these functions.

[00:06:21]If I do that, we get the following plots. In the leftmost plot, you see the red line is the function we're trying to approximate. The blue line is the noisy measurement that we're getting. So this is a very noisy sensor that's providing data. The orange line is the prediction from our linear regression. You see what happens: our linear regression is not really powerful enough to match the line we're trying to match. It predicts just an upwards trending, average value of the real measurements that we're seeing.

[00:07:01] On the other hand, our decision tree regressor can match the function very exactly. Actually so precisely that in the plot, you cannot see the blue data anymore, all you see is the estimation of the decision tree. If you compare this to the red line that you actually want to match, then this is not a good situation at all.

[00:07:26]In the left image, we have a case of underfitting. Our regression model is just not powerful enough to approximate the function we wanted to approximate. On the right hand side, we have a case of overfitting. Our regression model is so powerful that it can match all the noise and therefore doesn't fit very well to our function that we want to approximate, either.

[00:07:52] Now, if you know exactly what kind of curves you expect, then you can just deal with this problem by choosing an algorithm that only generates curves that are close to this red line, and you can get the best of both worlds, for this single example. But then your algorithm will fail if the curve that you're trying to predict changes and becomes, for example, a piecewise constant function.

[00:08:18] On the other hand, if you're giving your algorithm a lot of flexibility in the function that it wants to approximate, then it also has a lot of flexibility in approximating the noise in your training data. And that's why there is this trade off. How powerful do you make your algorithm? How many curves do you allow it to approximate? If you increase the algorithms power of approximating different functions, you typically also increase its power to match noise that you don't want it to match. And that's why it's a trade off in the machine learning case.

[00:08:59]To elaborate a little bit more and also to elaborate a little bit about what does the noise actually mean: what are possible sources of the noise? Here, I've drawn, again, our curve, our fun curve. And I've also added another plot where I've added a randomized version of the curve and say, just to take a concrete example, you are measuring the acceleration of a race car along a race track. The red line is the actual acceleration that the driver applies. And because your sensor is noisy, you get back this blue line that is jagged and dances around the real value, but that kind of tracks the real value .

[00:09:47] Now, in this case, you have one true target function that you're trying to approximate with your algorithm, and you have noise that distorts this target value. And so the problem is just filtering out the noise, so to say. But this is not always the case. For example, if you have another race car driver and he drives differently, or his car has different performance characteristics, then he might actually apply acceleration according to this yellow curve, and your measurement is this orange jagged line that, again, approximates this yellow curve in some sense. But you see that now you have two different curves that are generated for the same problem by two different race car drivers or two different cars.

[00:10:38]And so if you're developing an application that should predict the acceleration for all kinds of drivers, you have a problem because now you have, both the measurement errors generated by the sensor for each driver, and you have the different curves for the driver which mean that you don't have a single value that you can predict for each of your points along the track. The acceleration is in a range that depends on the kind of car and perhaps also the kind of driver.

[00:11:10]I've plotted here a third, very different acceleration curve. Say, you have an autonomous car that has a very different style of driving than a human driver. So it tries to apply constant deceleration in this leftmost part, and then accelerates again with a constant acceleration in the right most part. Then you would get this kind of curve that is still within the permissible curves, that a solution might take, but it looks quite different from the curves you were seeing previously.

[00:11:41] And so these are two sources of noise or of error in your sample. So the errors might be because of your sensor, which does not return the true value, but has some measurement error. And the error might be because you're kind of conflating different curves and different distributions of value.

[00:12:05] And you don't know which measurement was taken from which driver or which car. So your data is not precise enough to allow you to predict the precise value, but you can only predict a certain range of values.

[00:12:19] And this image, once again, illustrates the bias variance, trade off relatively nicely, I think. If you have a model that can only approximate these smooth curves as they are for the red and the yellow lines, then it will not have a problem with the noise in these curves, because it just has no capability of matching the noise, and so it will place a smooth line through your measurements. On the other hand, it will perform poorly on this green curve that the autonomous car generates. Now, if you extend your model and allow it to match this green line with its jump and it's constant regions, then it may very well happen that it also matches some of the jumps that your smooth curves make and therefore introduces a larger error in your smooth acceleration curves. So that's the trade off: do you allow your algorithm to fit many possible functions, if you are dealing with a regression problem, and thereby increase the risk that it may match noisy data or irrelevant data, or do you restrict your algorithm to a very specific set of functions that it can match, thereby increasing its resistance to noise, but also making it fail when it encounters a function, that's not in its distribution.

[00:14:00] So both over and under fitting are problems and are not good, but in practice, the larger problem is over fitting. So it's generally easy to find a model that will match all your training data. The problem is preventing it from matching the data that you don't want. Now, one of the questions that you may ask is how do you recognize over fitting?

[00:14:26]And it's actually not always clear whether your model is overfitting or not, but there's a general rule of thumb that most people follow: if the performance of the model on your training set is much better than on your validation or test set, then the model is probably overfitting. We can compare the mean absolute error or mean squared error of the model, over its training set to the error that it makes over its test set. And if those two are very different than we expect that the model has over fit and is not really performing very well in generalizing to our test set.

[00:15:11]When you're training neural networks or other models that are trained with stochastic gradient descent, you can typically, during the training process, monitor how well the model is performing on both the training data and on the validation data set that you use to validate your training process.

[00:15:32]Typically you will see a point where the performance on the training data set becomes better, but the performance on the validation set starts to decrease. So actually, the longer you train the worse your model gets on the validation data set. Not on the training set, only on the validation data set.

[00:15:52] And that point where performance on the validation data set starts to get worse, it's typically the point where you stop your training and say, now we have achieved the maximum performance that we can get from the model. It's not a very precise critereon because sometimes models get stuck at a local minimum for some time, and if you continue training, they would increase their performance. But it's a quite good rule of thumb.

[00:16:21] Now, what can we do to reduce overfitting? So it's often clear what we need to do when our model is under fitting. We need to make the model more powerful, or we need to switch to a more powerful model, if it's not possible to tune the hyper parameters of the model we are currently using.

[00:16:40] But the much bigger problem in practice is you have a model and it's overfitting, what can you do?

[00:16:47]One way that's almost always applicable, but that's often very expensive and time consuming is to simply collect more or different training data.

[00:16:58]If the overfitting is due to a lot of noise in your training data, for example, because you have a sensor that measures with a lot of noise, then you can just collect more samples and if you average over enough samples, the noise will cancel out, if it's just random noise and not some systematic error. If you have a systematic error, you can try to get data that eliminates this systematic error, for example, by using other sensors to measure the same value and then trying to reduce the systematic error in your training data by statistical techniques.

[00:17:36] For example, if you have the problem that we have seen in the race car example with different drivers, then just simply collecting more data will not really help you because we have a mix of distributions and there is just no single value that we can predict. So it may be that just getting more of the same data does not help us reduce overfitting.

[00:18:00] In that case, we might. Investigate, whether we can get data that has, for example, more features. So if we could get the information, which driver was driving the car from which we collected the acceleration data, then we could actually fit this much better to a single curve than if we have this broad distribution over many cars.

[00:18:21] Of course that's not always feasible. And it may, in particular, not correspond to the data you can get during the production use of your system. So the feasibility and the economics of collecting more or better training data is something that has to be decided on a case by case basis.

[00:18:41]Another thing we can do is better feature engineering. So for example, if we're training a linear model using stochastic gradient descent, and we have not normalized our data, so that we have values with widely varying averages or standard deviations, we can perform feature engineering steps to make our algorithm fit the data better.

[00:19:06]If we have knowledge about the problem, we can perform some feature engineering steps that actually incorporate this knowledge about the problem in the data, and that lead to much better results than without feature engineering. The problem with feature engineering is, that it's a very labor intensive process typically, and that you have to repeat it, if some of the parameters of your experiment change.

[00:19:34] Now if a model is overfitting, it's because it has, in some sense, too much capacity, it can generate too many functions. And one way of reducing over fitting is therefore decreasing the model capacity. We will see several ways for doing that for decision trees in the upcoming slides.

[00:19:55] Another way to reduce overfitting that's very closely related to decreasing model capacity is regularizing your model. A parametric model, when it overfits, often uses very large values for some of its parameters, because it tries to match extreme movements of the target function that are generated by the random noise.

[00:20:20] So if you prevent the model from using high values for its parameters, you can, in many cases, reduce its tendency to overfit. And one way of doing that is to extend your loss function. In addition to the loss that is due to the prediction error of your model, you can also add a term to the loss function that punishes the model for large values of its internal parameters.

[00:20:44] If you do that, the model will try to fit the values that it's supposed to predict as well as it can, but it will also try to minimize its parameters internally. And so it will perhaps try to fit a smoother function because this small increase in quality that it could get from matching the noise is not worth the increase in parameter values that gets punished by the regularization term in the loss function.

[00:21:17] There are other ways of reducing over fitting that correspond to the data that you feed to the model. For example, you could use cross validation, a way to make more effective use of your training data that we'll probably look at in a later lecture.

[00:21:34] And you could augment the training data, for example, if you feed images to the data, you could generate new images from your existing images, by rotating them slightly or flipping them along one of the axes or maybe changing the color balance slightly. So you can get more training examples for very little effort. And this may increase the robustness of your model to slight variations in the data.

[00:22:04]In particular, if you're using neural networks, there are quite a number of techniques in the design of your networks that you can use. For example, you can use layers like batch normalization, or drop out that prevent the network from over-fitting by modifying its internal structure and we'll probably not get to these layers in this course, because it's just an introduction to neural networks. But when you're dealing with neural networks in earnest, you should probably look up these terms like batch normalization, and drop out and see how they are used in modern neural network architectures.

[00:22:44]Another way to reduce overfitting, especially if you're dealing with models that are trained via stochastic gradient descent is to just stop your training. As we have seen, the point where your performance on the training data continues to improve, but your performance on your validation data starts to decrease, it's often the point where your model starts to overfit and then you can just stop your training process when that happens and hope that you've actually reached the correct point for training your model, the point where you get the best performance out of the data and the model architecture you are using.

[00:23:25]Okay. So we've seen how we can reduce overfitting. Let's look at this in practice.

[00:23:34]I'll show you some methods for reducing over fitting of decision trees. I first reset my random number generator so that we're getting a consistent result. Then I'm generating a decision tree regressor and a set of training and a set of training values and a set of features for training. Then I apply a randomized version of my fun function to the training set, and I also apply a randomized version of the fun function again, to the training set, to generate test values.

[00:24:13]I'm doing this because I want to have the points between the training and the tests that aligned so that it's somehow nicer to view in the graphs. This is of course not something that you should do when creating a data set for a real problem. That you should have different feature values for the training and test sets.

[00:24:35]So finally we fit the regressor to the training set and then we can compute the mean absolute error that this regressor makes over the training set. And if we look at the absolute and squared errors, we see that this regressor has fit the training data perfectly.

[00:24:54]We have this absolute error and mean squared error, which are both zero. And that's of course, a sign that something is probably wrong with your model, if you get a training error of zero. To show you how this contrasts with the error that we get over a test set, I'm computing the mean absolute error for the test set and the mean squared error for the test set and here you see that they are much larger.

[00:25:23] So the test set has 1.5 as absolute error and 4.3 or 4.4 as squared error, and this is a sign that your model has overfit quite severely. The error on the training set is zero. The error on the test set is 1.6 or 4.4, for the absolute and squared error. So there's a huge difference in the error.

[00:25:49] Since we have generated synthetic data, we can look at the best result that we can realistically expect a regressor on this data to have, if it's not over-fitting. And we can obtain that by just using the mean absolute error between the Y train values and the real function without any randomness applied to the X train values.

[00:26:14] So we see how far are our training values away from the original values. And we see that the deviation here are the absolute error is 1.2 and the squared error is 2.2. And so if our regressor is doing a lot better than this, it's probably overfitting. Now of course, in a real scenario, you would not know these values because you do not know the real underlying function that generates the data.

[00:26:45] If you knew this function, you wouldn't perform machine learning in the first place, but just for our estimation here, we can see that anything better than these values is kind of suspicious because if a model has lower error than this, it means that it's closer to the training data than the real data, from which the training data was generated by randomly deviating from the ideal.

[00:27:12] And so this is a relative sure sign that it's, overfitting. Now let's plot our training data and see where this overfitting comes from. Of course, we've already seen these kinds of plots. So here we see that the model matches each point exactly. And if it doesn't seem to match some point this is just an artifact of the plotting method we're using.

[00:27:36]And so here, you see why the performance is so good, because each point is matched exactly. But of course, as soon as you go to the test data, which has the same random distribution, but not the same actual values, and you see that now our function is no longer, really, very well aligned with this data.

[00:27:57] And this is the reason why the error of our regression model was much larger than the error of this line from which the data is actually derived.

[00:28:06] Having seen the problem, let us now come to the solutions.

[00:28:13]Here, I'm evaluating a regressor with the evaluate regressor function that you know from the last lecture. We see that the regressor fits the training data exactly, and therefore strongly overfits, as soon as we have randomness in our data. Now in the evaluate regressor function, we don't really show the errors on the training data. The evaluate regressor function just shows the errors on the test data, not the errors on the training data. It might perhaps be a useful exercise for you to modify it so that it also prints the errors on the training data. And you can then see how the training data has lower error due to over-fitting than the test data, or perhaps doesn't have this lower error because it no longer overfits if we regularize.

[00:29:09]So this is the graph that you've already seen. Let's now take a few actions that can reduce the overfitting. And you know that this evaluate regressor function passes all arguments after the function to the constructor of the regressor.

[00:29:24] So we can use any decision tree regressor arguments in our evaluation. One of the easiest way to make sure that a decision tree doesn't overfit is by limiting the maximum depth. So here we've added a max depth equals two argument, and that means that a decision tree can have no more than four values that it approximates.

[00:29:52] We have the value from zero to slightly below three. Then we have the value from three to six. Then we jump up to this value from 6.4, I believe to seven point something. And then we jump up to the, final value.

[00:30:09] And this is all the approximation that the function can perform because it has just an allowed depth of two. So it has just four discrete values that it can pick to approximate this function. Now, of course, this will reduce overfitting, so you see doesn't track the green dots very closely anymore, but of course it also increases our errors in the cases where we have no randomness or some randomness, because now we can no longer really approximate the function very closely.

[00:30:43]If we increase the depth to say three, then we get a better approximation, but you see that we also start approximating the random errors again. And this is especially visible on the right hand border of the function where the green function, the one that tracks the values with lots of randomness makes this jump upwards to a value of over 10, which is totally not justified and just tracking the noise again.

[00:31:19] But overall it's doing a better job. Though we can already at max depth three, see some areas, for example, slightly below six, where the function seems to track the noise values rather than improving on the function that we wanted to approximate.

[00:31:38]Another way of reducing over fitting is using a different loss function. So in this case, we've restricted again, the depth to two, but use the mean absolute error as criterion instead of the modified mean square error, that is the default.

[00:31:56] You see that in this case it doesn't change a lot and you have to play around with your concrete examples to see whether the change of loss function from square to absolute loss has any significant impact on your problem.

[00:32:13]Another way we could restrict the tree is not by restricting the maximum depth, but by restricting the number of leaf nodes. This leaves the tree some more flexibility to expand into these areas where it thinks that it needs to have additional capacity.

[00:32:31]This works if you have data that is not too noisy. So in this case, it seems to work all right, for some randomness, but if you look at the green line, you see that it doesn't work really well if you have a lot of noise, because the estimateor very strongly approximates some of the noise values. It can't track all of them, but it still tracks some of the outliers that it considers important.

[00:32:57] Now, here we can change the evaluation criteria from the squared error to the absolute error, and if we evaluate this again, we see that it doesn't track some of the values as closely, but it has actually made the performance worse, particularly on the right hand side where it now tracks both this very high and this very low value exactly, which is something that we don't want.

[00:33:23]Limiting the maximum number of leaf nodes on its own is often not a useful way to restrict overfitting, but it can be effective when combined with other methods .

[00:33:34] One of the measures that is commonly applied is limiting the minimum number of samples for which the algorithm can still split a node or the minimum number of samples that a leave node has to have.

[00:33:49] Limiting the minimum number of samples for which the algorithm can split means that if we have less than 16 values in some node, in this case where we have set min sample split to 16, the algorithm is not allowed to further split this node, so it cannot further divide this node and this quite naturally limits the depth to which the algorithm can split the nodes.

[00:34:16] Min sample split does not tell us anything about how it can split nodes with 16 samples or more, and you can see that sometimes it splits such a node in a way that it still tracks some outliers very closely.

[00:34:33] Overall, the graph looks better, especially on the right hand part, but on the left-hand part, we see that min samples split still leads to a graph that has at least a moderate amount of overfitting.

[00:34:47]Related to min samples split is min samples leaf. And the minimum number of samples in a leave node tells us that the algorithm is not allowed to generate leaf nodes with less than eight samples in it. So this is actually a stronger condition than the min samples split 16 that we had, because now we say, well, you can split a node if it has 16 samples, but then you need to distribute the samples evenly in both of the sub nodes.

[00:35:22] And so this is a measure that prevents overfitting quite successfully, but that also sometimes leads to decision trees that actually underfit and don't follow the curve that we want them to follow very precisely.

[00:35:37] So here we can see that this happens a little bit in the case of the green graph, where around the value two, or between two and three, it doesn't really follow the curve that we want it to follow because it's influenced by the high values in our distribution, and it cannot split this down any further.

[00:35:58]So these are some of the parameters that you can tune to make your decision tree fit to your problem.

[00:36:06]You hopefully see that you can achieve a certain trade off between bias and variance for decision trees by choosing these parameters.

[00:36:15] And you can play around with some of the parameter settings on this problems or on other problems that you want to investigate and see how they influence the behavior of the tree. Generally speaking, it's relatively hard to get a single decision tree to fit a curve like this very well, especially if there's a lot of noise involved.

[00:36:39] One possible way to improve this situation is taking many decision trees and training them in such a way that the errors hopefully cancel out. And this is what is called ensembles and more specifically one kind of ensemble is a random forest.

[00:36:56] That's what we'll look at in the next lecture.

[00:36:59] I hope you found this lecture interesting. Thank you for listening and see you soon.