Module 8 Transcripts

Video 1: Introduction to module eight [02:25]

Hello and welcome back to module 8, the first module in phase two of our certificate program. In phase two, we will finally be able to reap the benefits of our formal understanding of the machine learning principles that we have gained in phase one of the program, and we will start to investigate different machine learning methods.

This week in module 8, we will explore *k*-Nearest Neighbor methods. Remember from our earlier discussions that every machine learning method, no matter whether it's a parametric or a non-parametric one has an underlying bias. That is an implicit assumption about the shape of the function that maps the inputs to the outputs and that we want to learn.

Recall also from module five of our program that such a bias is absolutely needed. Otherwise, learning is not feasible at all. So, what's the bias of *k*-Nearest Neighbor methods? It is the assumption that similar inputs lead to similar outputs. To that end, we need to understand how we measure the similarity of data samples. We'll do so in three steps: we will first discuss how we can convert categorical predictors to numerical ones, we then discuss how we can normalise numerical features so that they live on the same scale, and finally we will discuss how we can measure distances between samples that contains numerical predictors only.

Along the way we will also have the opportunity to learn some general machine learning skills. We will revisit the bias-variance trade-off. We will have our first encounter of parameter tuning; the k in k-Nearest Neighbor is actually a parameter that we need to tune, and we will use the training set validation set approach in this module to find the optimal value of k. And finally, we will also encounter the curse of dimensionality that plagues distance-based machine learning methods.

Video 2: Motivation [04:29]

Imagine you visit a dark dining restaurant. A restaurant where the food is being served in complete darkness. Directly the food itself by weight as either that applied or that use specific optical devices that help them to see in the dark. The idea is by eating the food in the dark, you get much more of an appreciation of the taste of the food. Now you mentioned, for example, in a dark dining restaurant, you are being served a tomato. How would you recognize what type of food you're actually eating?

Let's look at these problems through the lenses of a machine learner. Imagine you have some training dataset which, which relates to your previous food experiences. For example, you might have eaten an apple before, you have eaten bacon before bananas, carrots, and so on. And for each of these samples from the past, you have the value of the output variable, which is the type of foods such as fruit, protein, or vegetable. And you have the values of variable, various input variables, which in this case here, sweetness and crunching has both measured on a scale of one to ten. So how would you approach this problem then? Well, you could create a scatter plot on all your previous food experiences in a 2D graph where you have the level of sweetness on the x-axis and the level of crunching on the y-axis. You would then probably observe that your previous food experiences naturally cluster into similar types of food. So for example, vegetables tend to be rather crunchy, but not that sweet. Fruits, on the other hand, are characterized by distinct sweetness. They can come in crunchy forms such as apples or a non crunchy forms such as bananas, but they haven't come that are typically sweet. Proteins, on the other hand, such as bacon, fish, cheese, and so on, tend to be less on the crunchy side. And again, similar to vegetables tend to be not overly sweet. So now let's compare our experience of eating a tomato in terms of sweetness and crunching has to our previous food experiences. In particular, we may want to place the tomato in this 2D sweetness crunch in a graph. And then look at the previous food experiences that are very similar to that experience of eating a tomato, both in terms of crunches and in terms of sweetness.

For example, if we would look at the closest foods expands that we had in the past, then this may be eating an orange. Orange is a fruit. So as a result, you could conclude from that, that perhaps whatever you're eating a tomato, but you don't know Nell is a fruit as well. Could be a bit more conservative and say, well, I don't just want to rely on the closest food expands, but I want to rely on the closest to past food experiences. In that case, eating a tomato could come closest to eating oranges and eating grapes. Both oranges and grapes are fruits. So again, your conclusion might be whatever you're eating right now must be a fruit. We can be even more conservative and compare our experience of eating a tomato to the three closest past food experiences. In which case, we would compare eating a tomato to eating oranges, grapes, as well as eating nuts. Remember, oranges and grapes are both fruits, but not our proteins. So, what we have to do here is to take a majority vote. Two out of three closest pass food experiences are fruits. One out of those three proteins. In which case we decide that whatever we are eating must be a fruit. Let's try to see how we can formalize that idea.

Video 3: Distance functions [03:22]

The fundamental aspect of k-Nearest Neighbour Algorithms is to define what we mean by the term near. So, we need to define the concept of proximity. For this, we use distance functions. Imagine you want to measure the distance between a new observation, which is given by the input variables X_1 to X_p, to previous observations that we have seen in the past, which are given by X_i1 to X_ip, where *i* ranges, let's say, from 1 to *n*. How can we do this? A standard way of measuring distances is by using a q-norm distance, where *q* can be any natural number. Particularly, prominent choices of *q* are q=1 which relates to the Manhattan or taxicab uh, distance; q = 2, which relates to the Euclidean or bird's eye distance; and finally, q going to infinity, which relates to the maximum distance. You see the formulas here, and I will ask you to do some exercises using these formulas in a moment.

But let's look at a simple example to see how these different distance concepts relate to another. Consider, uh, the following map of Manhattan, and let's assume you want to go from point A to point B. If you measure that distance by the Euclidean or bird's eye distance, then what you're doing is you're drawing a straight line from A to B and you measure the length of that line. This is the path that a bird might follow if the bird wants to fly from A to B. We, unfortunately, don't have that luxury, when we want to go from point A to point B because there are going to be many skyscrapers along our way. So, what we have to do is we have to use the Manhattan distance whenever we're travelling in Manhattan, which is, for example, given by the, uh, by the pink line here. The pink line travels from A to B in right angles because we need to walk along the pavements which are following the different blocks of Manhattan. Any such, uh, any such right-angled line connecting A to B is going to have the same Manhattan distance. The third and final distance that we consider here is the maximum or infinity norm distance, which is given by the red lines. The maximum distance takes the larger of the two distances, which is the horizontal distance that you have to cover by going from A to B, which is given by the dotted line here, as well as the solid red line which is the vertical distance of travelling from A to B. You might see that in this example, the larger the value of q, the smaller the distances become. This is actually, uh, not, this is actually not a random incident. It happens systematically for whichever data set you use.

Video 4: Scaling [04:30]

We need to bear in mind that whenever we use distance functions in machine learning, we should be aware of the issue of scaling. In the example that we have shown on the Web, you might have seen that our nearest neighbour classifier would indeed be heavily affected if we introduced another feature called Spiciness.

If we look at the chilli grade uh that we have shown on the Web and which you see here as well. Spiciness ranges from 0-15 million, depending on the type of chilli that you're eating, if you were eating chilli. Now, what would happen to our nearest neighbour classifier in that case? We would basically completely ignore in our distance calculations the sweetness and crunchiness. We would only measure the spiciness because the umm scaling is a very different one. Instead of going from 1-

10, the spiciness ranges from 0-15 million, which is going to be heavily impacting the distance calculations.

Now, in order to avoid this undesirable artifact, we need to scale some of the fields. Imagine, for example, you want to scale the Jaft input variable. You have two popular ways of achieving that. You can either resort to a min-max normalisation, or you can apply Z-score normalisation. Let's look at these two types of normalisations in turn. In the min-max normalisation, we first calculate the minimum value across all our training samples of the Jaft input variable. You see this as the minimum term here on the slide. We then calculate the maximum value for the Jaft field for the Jaft input variable across all our training samples. That's the maximum term that you see here.

Now, for all of our training samples, we now replace the Jaft input variable value with the formula that we see on top of the slide. In particular, we take the previous input value, we subtract from it the minimum value across all the training samples and we divide that difference by the difference of the maximum and the minimum occurrence of that input variable. This makes sure that subsequently all of our training samples are going to be scaled between zero and one. The second approach to scale an input variable is a so called Z-score normalisation. What we do here is we first, If we want to scale the Jaft feature again, we first calculate the mean value of the Jaft feature as well as the standard deviation of the Jaft feature across all the training samples. you receive the formulas for this on the slide. Those are the formulas for Mu J and Sigma J. Now, you go through all the training samples, and you replace the Jaft input variable value with the difference of its previous value and the mean value Mu J, divided by the standard deviation Sigma J.

Now, which of the two types of normalisations should we use in practise? It turns out that the min-max normalisation typically works well, if the predictor in question, the input variable in question, is roughly uniformly distributed. A Z-score normalisation, in turn, works better if we have the presence of outliers. So, if we have a few values that lie far apart from the rest. You can actually think of Z-score transformation as treating the data as normally distributed and turning a normal distribution with mean Mu J and standard deviation Sigma J into a standard normal distribution. Please make sure that whenever you apply a transformation or scaling to your training data, then you have to apply exactly the same scaling subsequently also to any new data point that you want to classify.

Video 5: Binary and categorical predictors [01:49]

Since nearest neighbour methods strongly rely on the concept of proximity, and the concept of distance functions to measure proximity, we need to modify binary and categorical input variables. We need to convert them to numbers so that we can use our distance formulas from before. This is relatively straightforward for binary predictors, such as yes, no or male, female predictors. We simply convert them into zero, for let's say no, and one for yes. This is for a categorical predictor with M possible values. You will introduce M minus one or M different binary predictors, which you can subsequently transform to 0-1 values. This is done in statistics for specific probabilistic reasons that you have probably discussed earlier. However, here we are not bound to these restrictions. We are also free to replace a categorical predictor with a numeric predictor that has different values for different possible categories. For example, a high, medium, low predictor could be transformed into a number that is three for high, two for medium and one for cold.

Video 6: How to choose *k* [02:44]

If we consider the data set of red and blue points here, it is a data set with one binary outcome variable, namely red or blue, as well as two numeric input variables. Those are the positions on the x and y-axis. Then we see that for small values of k, such as the one nearest neighbour shown on the left here. The graph, the boundary between what is predicted to be a blue sample and what is predicted to be a red sample, becomes very wiggly. On the other hand, if we choose a large value of k such as k = 99, the boundary between what is predicted to be the red region and what is predicted to be the blue region becomes much smoother, is almost a line.

Now, this is not, um, happening by accident. This is actually a systematic effect, and it relates to the bias-variance trade-off that we have discussed earlier. In fact, if we look at the k-nearest neighbour algorithm with a small value of k, then we have a small bias. We are becoming very flexible as a model, but we tend to have a large variance. If we take any slightly different data set, the decision boundary again is very wiggly, and it might look very different to the one that we saw on the previous slide.

Now, exactly the opposite is happening when the number k of nearest neighbours that we consider in our algorithm is large. In that case, we have a small variance. We get a smooth decision boundary. And that decision boundary tends to be relatively stable, even if some of the samples change but the bias is large as well. We, might miss the true model because the true model might simply have more of the curvature in the decision boundary than what we can, uhh, than what we can show when we choose k to be large. So, the only optimal value of k tends to lie somewhere in the middle. It trades-off the bias and the variance. How can we find this optimal value of k? There are some rules of thumb, such as use the square root of the number of samples. But I would be cautious with such rules of thumb. Rather, I would use the training set-validation set approach, or the cross-validation approach that we have used before. Let the data speak for itself rather than using any rules of thumb.

Video 7: Training and validation sets [02:48]

Our *k*-nearest neighbour example gives us a good opportunity to repeat what we have discussed earlier in terms of training set, validation set and test set. Imagine, for example, we want to choose the optimal value of *k* for the *k*-nearest neighbour simply by looking at the performance of our classifier on the training data. It turns out that, in this case, very likely the best choice of *k* is going to be *k* equals 1, because the sample that is closest to any of the training samples that we have is going to be precisely the training sample that we're looking at and, of course, we would then correctly classify each and every sample. But, in that case, all we are doing is we are reproducing the knowledge that we have already. We are not going to be able to properly generalise from our training sample experience. So, in order to choose a proper value of *k*, we need to split up our data into a training data set and a validation data set. For example, we could use 80 per cent of our data as training data and 20 per cent of our data as validation data. We could also use 50, 50 or 2/3rds, 1/3rd. This is really up to the application at hand.

Now, if we do so, we would want to choose different values of *k*, train each *k*-nearest neighbour method on the training data and then evaluate the misclassification rate or any other performance indicator that you might wish to use instead, on the validation data set. We then choose the best model, the best value of *k*, based on the performance of the validation set. Once we have chosen the best model, the best value of *k*, can we expect to see the performance that this model had on the validation set on new data? It turns out that this is not the case. There is a bias in the validation data set that we have introduced by using that validation data set to choose the best value for *k*. If we are interested in the true out-of-sample performance of the best model that we chose using the validation set, we need to keep some data aside as a test set and, subsequently, measure the performance of the best choice of *k* on this test dataset.

Video 8: The curse of dimensionality [07:05]

In the following, I will try to explain the curse of dimensionality from two different but related viewpoints. In the first viewpoint, I will try to convince you that in high dimensions, the closest samples are actually far away from each other, even if the samples are uniformly distributed. So, even if we apply a nice scaling to the samples. The second complementary view is that in high dimensions, data clusters at the corners of the space. Again, even if the samples are uniformly distributed. So, let's try to understand that in a bit more detail.

Let's take a look at the first view, where I claimed that in high dimensions the closest samples are far away from each other. Let's look at a small numerical experiment. Here are the assumptions that I'm going to make. Let's assume we have 5,000 training samples with p numerical features, where p could be 10, 20, 30, and all of those features are scaled, so that they live in the interval 0-1. In fact, our training data will be uniformly distributed on this hyperrectangle, on this hypercube, the 0, 1 hypercube in p dimensions. So, in some sense, this is the case of a perfectly scaled training dataset. We will measure the distances by the maximum norm. This is just to make the exposition simple. And the question is going to be, if you have any particular point in those 5,000 training samples, how close are the five nearest neighbours to that point? All right, let's try to answer this question.

In order to answer that question, we need to find the smallest hypercube that covers 1,000th of the volume of the 0, 1 hypercube in our p. Why 1/1,000? Because we are interested in the five nearest neighbours, we have 5,000 samples, so five out of the 5,000 is exactly 1/1,000. All right. How do we find the smallest hypercube that covers this volume? Well, the 0, 1 hypercube has a volume of one, and a hypercube with a side length of C, where C is any particular number, has a volume of C to the power p. That's just the definition of the volume. So, what we want to find, is we want to find the C, such that C to the power of P, the volume is equal to 1/1,000. Because 1/1,000 is exactly 1/1,000 of the volume of the 0, 1 hypercube. And it turns out, the formula, you can see it here on the screen, is that C = (1/1,000) ^ 1/p. That's how I look at how this function looks like as a function of the dimension p. We see that here, in particular, we see that in three dimensions, we only need a hypercube of 10% of the side length of the overall cube. So, in other words, to get the five closest samples of any given point, we need to go, we have a hypercube that has a side length of 10% of the overall space, where the data loosen in three dimensions. In 100 dimensions, on the other hand, this hypercube needs to have a side length of 93% of the overall data. So, in other words, in order to find the five closest samples in 100 dimensions, we need to have, or we need to look at pretty much 93% of the space in each dimension. So, these samples can be very far away as far as the input variable values are concerned.

Here's another take on this. In R1, in one dimensional space, to fill 50% of the 0, 1 interval, well, we just need to go between to 0.5. In R2, to fill 50% of the 0, 1 rectangle, square, we already need side lengths of 71%. In R3, to fill 50% of the volume of the three-dimensional cube, we already need side lengths of 80% of the overall 0, 1 cube. So, that was the first view on the curse of dimensionality. The second view is that in high dimensions, data clusters at the corners of the space. Let's try to make sense out of this statement. To this end, let me remind you that the volume of an n-dimensional ball of radius R is given by this formula here, where gamma is Euler's gamma function. You don't need to remember the details of that formula; we will use it in a moment to see what is happening if n grows large. So, let's now determine how much of this volume is concentrated at an Epsilon slice at the surface of this n-dimensional ball, where Epsilon is a small number. To this end, we can first of all calculate the volume of the entire ball, which is given here. And we can also calculate the volume of the interior of that ball, where we just take an Epsilon slice away around the surface. And if we take the difference of those two volumes and observe what happens to the distance when n gets large, we see that the distance is just converging to the volume of the ball itself.

So, in other words, what we're trying to say here is that n grow as n grows large. Most of the volume of this multidimensional ball is going to be concentrated just at a shell. So, in other words, the weight of an n-dimensional orange, if you wish, is going to be all in the skin, not in the middle. And this is where our standard intuition of how data is distributed in a space breaks down. We think of data, uniformly distributed data in R2 or R3, everything looks nice, but once you go to R50 or R60, things look very different from what our intuition tells us.

Video 9: *k*-nearest neighbours for regression [01:14]

So far, we have discussed the *k*-nearest neighbour method for classification problems where the outcome variable is either going to be binary or a categorical variable. It turns out that we can use exactly the same algorithm with only very minor changes for regression problems as well. The key change is, first of all, we replace the majority vote by the average of the *k*-nearest neighbours that we have identified. And, secondly, we need to choose our best value of *k* in a model selection phase in a different way. Rather than using performance measures for classification problems, such as the misclassification rate, the sensitivity or specificity, we now use performance measures for regression problems, such as the mean squared error and the other indicators that we have seen before.

Video 10: Case study: analysing music [06:34]

I 'd like to talk about analysing music. Have you ever wondered who wrote a certain song that you heard on the radio, or maybe a streaming service, and then you use some other service to figure out what song it was? They're actually our apps out there that let you whistle or sing a tune into your phone, and they turn around and tell you which exact piece of music you just whistled. Getting to that level of complexity, which is guessing musical excerpt from amongst all the music in the whole world, is quite difficult. But we can start to break down the process of analysing music by looking at a few simple examples.

For example, between the Beatles and the Byrds, the Byrds is a band that was known to have a similar sound to the Beatles and actually had quite famous hits back in the day. For example, which band played the following, I can't stay, and I'm probably feel a lot better when you're go... Since they're only two bands to choose from in this example, we would say it's a classification problem, or a categorisation problem when you have only two categories. And data scientists have studied this issue by looking at the similarity of the unknown song to the known songs by the two different bands. So, how can song be similar? How can one song be similar to another? Well, pause here, push the pause button and ask yourself, what criteria would you apply to trying to figure out which band performed a certain piece of music? Here are some ideas. You might look to hear the accent of the singer, the voice timber of the singer, the kinds of guitars that are used, whether there was a tambourine in there, the lyrics of the song, and so forth to try and see, oh yeah, I think that's that's probably a Beatles song. Some companies have taken a very scientific approach to this, including Spotify, the music streaming company, and they've developed some very interesting additional complementary services.

So, they've created a database of characteristics which they're calling "Audio features" of songs, and they published an API so that developers can grab information about songs, and they can use those characteristics do their own analyses. There are many, many different features that they give. For example, the acousticness, the danceability, the energy of the song, the liveliness of the song, the loudness of the song, the tempo, the time signature and other features that you could then turn around and analyse. So, you could actually use this approach to analyse, to classify a song from a band. What you would do, is you take each song from the two bands according to those features, and you try and look at them. And then, when you're confronted with an unknown song, you could try and match that unknown song along all 11 of those dimensions to try and find the closest, say, the closest five songs from amongst all the known songs. And then, if the unknown song clusters well with one or the other band, for example, it clusters better with the Birds than it does with the Beatles, you might have higher confidence that, in fact, that song was from the Birds. And you can see how this approach might be useful in all situations, not just music band classification such as analysing or classifying different purchasing patterns from customers deciding whether to offer a coupon to a customer or not or trying to decide which customer segment a buyer belongs to.

Now, speaking of the Beatles, moving to another level of difficulty so that first one was "Relatively simple" trying to decide between two bands. You probably know that John Lennon and Paul McCartney are one of the most famous song writing duos in history. They agreed early on that they were going to have Lennon and McCartney as author attribution regardless of who did what for each song, so they may have worked on things together, they may have done things separately, but they always had Lennon and McCartney as the attribution for that particular song.

Now, in many cases, you might know that the person who sang the song who sang Lead Vocals was often the person who wrote the song in the Beatles. However, there's not always 100 per cent agreement as to which of those two wrote the song. So, it's a controversy so called "Controversy" over the song called In My Life. In most cases, everyone agrees who wrote the song. But in the case of In My Life, the memories are a bit hazy. They each had a different recollection about who wrote the song. So, you can see this problem is actually somewhat analogous to the band classification problem we just discussed. But teasing apart the main author might require even more work that includes language, chord progressions and so forth. So, the solution here this mathematicians Mark Glickman, Jason Brown and Ryan Song apparently worked on this problem for 10 years, applying different techniques, including nearest neighbours and Bayesian classification. So, they concentrated on the patterns in the chords and the notes, so either by themselves or in small chunks. So, in other words, what notes did the person typically write in a song or what two notes in a row did the person typically write, three notes in a row, three chords in a row, et cetera. And they determined that in spite of the fact that Paul claimed that he wrote that song and John actually said that Paul partially wrote that song. It's most likely that John Lennon actually wrote In My Life.

So, even though this is a fun exercise and a fun example, you can also see that has many business applications, such as tagging emails as spam or trying to analyse sentiments of text such as being positive or negative, for example, how customers react to a new product by tweeting about it or writing reviews on Amazon, you could do an automatic analysis of these things and try and get the sentiment as positive or negative, applying very, very similar techniques.

Video 11: Summarising module eight [01:22]

This brings us to the end of module eight on k-nearest neighbour methods. We saw that the fundamental assumption of k-nearest neighbour methods is that similar inputs lead to similar outputs. We have also discussed how to measure similarity through distances, which involves the conversion of categorical to numerical features if they are present in the data, as well as normalisation to bring features to the same scale. Along the way, we have also revisited some generic machine learning principles. We have revisited bias-variants tradeoff that guides the choice of k. We have used the training set, validation set approach to choose the best k based on data. And we saw the curse of dimensionality in action, and how it affects any distance-based machine learning method. In the next two modules, we will cover the topic of decision trees. I look forward to seeing you then.