Week 12
Video Transcripts

Video 1 (6:01): Understanding Decision Trees

You're going to look at two machine learning techniques, decision trees and machine learning, and...and k-means clustering. Decision trees are tree structures that contain rules and the idea is to learn the rules that will... will provide you with certain-like you have many many cases in a data set and you want to get a rule that will identify one group of this data correctly, right, so, 1 cluster of the data correctly, that's the idea here. So, let's take a quick example. So, for example, in this-in Wikipedia, there's this picture of a...of a classification tree for 'titanic' survivors, and the idea here is that if you have a person and you have some data for that person, you can predict whether that person survived the 'titanic' or did not. So, let's take a look at this. So, I have an image here that I can upload but the image is on Wikipedia as well. So, what this says is if somebody is a 'male' and the answer is yes, then if the age of that person is greater than nine and a half years old, and the answer is yes, and whether the person had a sibling, oh sorry, if the answer is yes, then the person died.

The odds are that they died. On the other hand, if the age is-the person is 'male'- let me just see if that's correct, actually. So, 'John Brown' is a 'male', so that's a yes. Is his age greater than '9.5'? '30 years old', so that's a yes. If his study have more than two and a half siblings? That's also a yes. Therefore, there's an '89' percent probability that he survived the...the 'titanic', whereas we have 'Jill Jones', she's ''female'', so she's not 'male'. So, she probably survived it anyway, we don't care. 'Hercules Mulligan' is 'male', '2' years old, so he is 'male'. Is age greater than 9.5? No, so the likelihood that he died in the thing with a '17' percent probability.

So, note that '17' percent probability doesn't mean there's an '83' percent chance that 'Mulligan' survived. Okay, it just means that the '17' percent probability that we correctly predicted that he died, you know, all the other possibilities as well. And, the key in a regression tree like this is to look at the number of cases that we have in each of the nodes here. So, we look at this, is sex 'male' and the no side of it for females, then we see that '0.73'- there's a '0.73' percent probability that the person survived the... the...the 'titanic'...whatever, and sinking of the 'titanic'. And, we see that '36' percent of the total cases in our data set are represented by this. So, this is a pretty high confidence. '36' percent is a lot of cases. Whereas if you go down to, you know, travel all the way down to these two died and survived at the bottom of siblings greater than '2.5', the number of cases are just 2 percent, and that's a kind of low number of cases. So, what we are saying is that with a very small number of cases, we're trying to make a guess as to whether they survived or not, and that's not so great. So, typically regression trees what you want to do is, you want to find groups like this, the 'female' survived, that are represented by a large number of cases.

So that you have a little more confidence in it and then that is, you know, probably a better bet. And, the other thing in the decision tree that you want to watch out for is that, though the tree is complete, that means every case can get processed through this, it might actually be better on some branches and

worse on other branches. So, you might want to just take some branches. And the best example of that is, if you have a training strategy, right! In a training strategy, what you're trying to do is, you're trying to predict your- whether you are going to make money or not. So, what you might have is a whole bunch of technical features and fundamental features about a stalk, and you can build out a huge decision tree, what happens if the price learning ratio is greater than something, what happens if the price book is less than something, the ten year- the ten day previous return is greater than something etc. And, you might find that only one set that is one branch of this whole thing actually makes sense and that branch is good enough that you can buy and make money on that. That doesn't mean that, you know, having the entire tree doesn't mean that you can use every result on it. Maybe, you're missing out on many good trades, but what you've done is, with some high confidence, you identified some good trades and so you go with that.

So, the nice thing compared to regression with a decision tree is that you can focus on small parts of your final outcome and ignore the other parts, you know, and see whether they work well or not, you know, I don't care, that's really the...the nice thing about a decision tree. So, that's really going to work here. The two kinds of decision trees, they are classification trees and regression trees. Classification trees are used when there's a feature, value like a rock or a mine, and regression trees when there are continuous variables like the wine quality example we saw last week, And in classification trees, cost is usually measured using the measure of the probability of misclassification or purity, right! So, that is what's called a 'GINI cost function'. And regression trees, where the thing is continuous, that is- so this is like a misclassification, right! So, again, we can go into things like confusion matrices and all that stuff, and with the regression tree, usually you just use the 'mean squared error', because that makes the most amount of sense. And the whole idea of the tree is to somehow find these splitting points, like, you know. Should we split on 'male', 'female' first? Should we split on age first? Should we split on number of siblings first? These are all questions that are asked. If you split on the age, what should we split at? Nine and half, 10, 11, 12, 20, 40, 60, there are many choices.

So, regression tree tries to figure out or...or classification tree tries to- decision tree tries to figure out what the splitting points are, and which particular feature it should split on first.

Video 2 (4:44): Regression Trees_Wines Dataset

So, let's take an example. How do we predict wine quality using a decision tree? So, the first thing you want to do, of course, is import the data. We've already figured out all of this. So, we just get that data from our wine quality database and build our training and testing samples. So, here we've done...done all this before again. So, we will need a 30 percent testing and 70 percent training sample. The only thing we're doing is, we're keeping one data set, x data and y data, which is full, which contains all the data, no training and no testing, and we're going to use this for something called cross validation. So, we'll come to that later, but I'm just creating that here. So, here we have a training and testing that is 70 percent and 30 percent of the data. And, I also have additional two variables that are the entire data set. So, we've got these...these things here, and the next thing you want to do is you want to decide whether you want to pick a classifier or a regressor. So, classifiers, we said are used when you have categorical

variables, like rocks or mines, and regressors are used when you have continuous variables like wine quality rating of '1', from '0' to '10'.
So, if you are doing wine quality over here, we'll use a regressor. So, that's out goal here. So, to do that once again, like we did with regression, you pick a model. So, from 'sklearn.tree', that's where the decision tree models are, we pick the decision tree regressor model and import the tree structure because that's what we're going to draw. And then, we create the model and we get a max depth of three. Depth is how deep you want to go, X greater than '10', the top level. The next level, you know, one branch y greater than '20', another branch, Z greater than '10'. But how...how many levels of this tree do you want to go? So, for simplicity, we're going to use a max depth of three over here, so that we get a tree that we can look at. Otherwise, the tree will become too big to look at. And then, once we've done that we fit the model, the x tree and y tree. And, more details are available here. So, let's do that. And then, since it's...it's a continuous variable, we can actually look at the 'R-squares' and see what they look like. And, they get, you know, fairly low 'R-squares', '0.35' and '0.28'.
But again, note that though the 'R-square' is low, we don't worry too much about it because like I said, we are going to- we, maybe, we want to focus only on identifying good wines, you know. So, let's say our goal is, we don't care about, you know, messing up below '0.7'- for a 7, below a 7.0 classification. But, anything above 7.0, we wanted to call it a good wine, right! So, if we can identify some good wines correctly, then that will be great. So, with 'R-square' is a good way to look at how good a model is. The next thing we can do is actually view the tree. To view the tree, what you want to do is couple of steps and follow this carefully, and then pause the video or, we will go to the next video. But, you want to make sure you do this in the correct order. The first thing you want to do is download and install 'graphviz'. So, if you go to this website over here, it'll take you through the steps of downloading 'graphviz'. You download it and add it to your computer path, and if you're doing it on a Mac, you won't have a problem. You should really do it straightaway. If you're doing it on a Windows, it should work but if it doesn't, then you might need to add this- these steps over here, which is essentially adding the path to 'graphviz' physically into your-manually into your system path, okay!
So, follow the steps and install it but, you know, don't do that until you've tried this thing a little bit, okay! And then, after you've done that, you install 'pydotplus'. Don't install-actually, no, wait a second. So, what you want to do then is you want to make sure that on a Windows machine, make sure that you go to your PC properties, and see that you have this in your path because you want to be able to do this. That you have this in your path because you want to be able to do this before you apply-install 'pydotplus'. If you install 'pydotplus' before 'graphviz' is correctly installed, you're going to get into trouble, right! So, make sure that this is installed correctly, and then you can install 'pydotplus'. And I'm- I think now with the new version, you should be okay with Windows as well. But, just check it on Windows. On a Mac, it should be no problem at all. And on a Mac, you want to install the correct version of 'graphviz', depending on which version operating system you have. If you have anything above mountain lion, then install the version that says mountain lion. Otherwise, install the specific version that's listed in the website that you go to. And, once you've done that, you can install 'pydotplus' in your machine. I've already got it. I'm not going to do that. You just do install 'pydotplus', and you should be good to go. So, what 'pydotplus' is-what 'graphviz' is, the package for showing graphs. And 'pydotplus' is

a...is a... is a-'graphviz' is an independent program. It's not Python. And 'pydotplus' is a Python library that interfaces with the 'graphviz'.

Video 3 (10:22): Regression Trees_Cross Validation

So, what exactly does the decision tree do? Well, decision trees try to minimize entropy. What is entropy? Entropy is the degree of variance inside the data. So, what we're going to do is, we're trying to take our data and split it up into sets where the variance is lower when you combine-when the combined variance that you add up the variances, you know, you don't physically add them up, but you know what I mean. When you take the variance of the two sets and the combined variance is lower than the variance of the combined sets. So, the variance adding up of the variance of the two subsets is lower than the variance of the overall data. You might end up with a higher variance in one of the subsets than you have in the overall data, but the combined variance has to be lower.
So, that's the...the rough goal here. So, when we are doing, so roughly what we are doing is, you think if you...if you take a marble from a box of '100' blue marbles, then you know pretty much that the marble is going to be blue, with a '100' percent probability and then, if we take a marble from a box of '50' blue and '50' red marbles, right, then, the probability it is going to be '50' percent that you get a blue or a red marble. That's the idea here. So, if you take a box of '100' marbles which are blue and red, and we can split them up so they're mix, then your probability when you draw a model-draw a marble probably, that is going to be blue is '50' percent. But, if you can split them up into two groups, where one group has all '50' blue and the other has all '50' red, then the combined probability, you know, if you- because you have these two groups, the probability that you-and you know for certain that one group has all blues and the other has all reds, then the probability of picking a blue or a red marble is one, right! Because, you've split it up and that's what decision tree is trying to do. It's trying to find some rule that splits the marble by color, so, for example, if the blue marbles happen to be bigger than the red marbles, right, let's say the blue marbles are 1 inch radius. That's pretty big for a marble. 1 inch diameter and the red marbles are 'half an inch' diameter. So, you split up marble size and you get two groups that are now all blue or all red, we just split on some other variable.
You split it on a...on a feature of the marble that is not the color. The color is the dependent variable and the feature is the size. And, that's...that's how the decision... decision tree model really works. So, it tries to minimize the combined entropy of...of the model. The regression tree, what it's doing is, it takes- it runs regressions for each of the independent variables on the dependent variable. It picks the variable that has the most explanatory power and splits it at several points. So, it'll take the variable and say "All right, acidity, fixed acidity has the most explanatory power". It splits that on several points and decides which point gives you the best discrimination between the two halves. In the regression tree, that means that it looks at the mean square error in the two halves of each split and then, finally, it picks the...the split point and gives it the lowest mean square error, right! So, in our tree here, what it was doing was, it was looking at the impurity, which is this case was the mean square error since it's a regression tree. And, you're dropping the regression- the mean square error from '0.6509' all the way down to '0.4149' over here or '0.3114' over here, right!

So, this is what's going on over there with this, thank you! And...and...and that's-so it's picking the split point and that works with that stuff there. So, that's how that works. Now, the problem of course with regression trees like, with all models, is that we don't know how well it's going to perform in the outside world, right, when we don't-when we're looking at things that we can't see. So, in our example here, we can scroll up. We are actually using the...the tree we measure it on... on the testing and training samples but in general, we don't know how it's going to work, when we end up with data that's not inside or current data set at all, right! So, that's one possible problem. The second problem is, since we are splitting on entropy, we need a lot of data to actually calculate the true entropy. You can think of there being a true entropy in the world, like we have a '100' marble set but they are, you know, you can think of a billion marble set that is really we are drawing this '100' marble set from, right!
So, there's a true entropy in the billion marble set and we have sample entropy that we are measuring from our marble set. So, we want to know how well the sample entropy reflects the true entropy and the larger sample set we get, the more reasonably we can expect that it will reflect the actual entropy. So, if we get a larger sample set, then presumably, you know, and enough variation captured in that sample set, then presumably that'll work better in the outside-in unseen cases. And, if we don't get a larger sample set, then it won't work as well in unseen cases. That's the idea there. So, if we're working with smaller sample sets, then we can use a technique called cross validation to- even if you're not working with small sample sets, you probably can do that anyway, it's probably helpful anyway, to see how robust our model is. So, the idea here us very straightforward. We have a sample in our training and test combined and we divide it up into a training and a testing. We train on the training and we test it on the testing, okay! And that gives us an estimate of how good our model is.
So, we can train it on the training, test it on the testing, and get a reasonable result. However, if we pick a different training set and a different testing set, and we divide up our original data into different training and testing set, we might get a very dramatically different result. If that happens, then our model is not going to be robust because it's really dependent upon how we split the data into training and testing, and that's not so good. So, we can use a technique called cross validation to see how robust our model is. So, what we do in cross validation is, you take the entire data set and split it up into 'k' smaller sets and then you train the data on the, on 'k' minus '1' of those sets. So, if you have like, '100' data items, you can split it up into ten smaller sets of ten data items each. You do run a training tree on '90', on the first 9, that is '90' of them and test it on the last ten, okay, and then you use the results from your testing to- and store that result. Then, you take another nine which is not the original nine, maybe like 1, 2, 3, 4, 5, 6, 7, and 10, and test it on 9 and then take 1, 2, 3, 4, 5, 6, 7, and then 9 and 10 and test it on the 8 etc., right! So, you-so these are called folds and you run it on different subsets of the data, different subset of the folds and measure the mean square error. The idea being in a regression tree or misclassification in a...in a classification tree. The idea being that if on each time that you run it, you get a means squared that's fairly similar to what you'll be getting with different subsets, then your final results are likely to be more robust because they're not dependent on where you split your training and testing sample. So, that's the idea here. You take the average of all tests as the performance metrics...metrics. So, let's run that on our-it's a very typical way of running, testing your model out.
So, we'll run that on...on our thing here. So, recall from model selection, we pick this thing called, 'cross_val_score'. And, we pick 'KFold'. 'KFold' is going to create the number of folds, the number of

subsets that we are dealing with. So, the first thing we do is, we do a cross validation on 'KFold' and we split it on '5' and we shuffle the data with a random. So, each, when we...when we do the folding, we want to shuffle our data first and then do the picking, rather than just pick randomly, just pick them literally from there. So, I run the cross validation here. Oops, I did not run this first. Run the cross validation there. And now, what I want to do is, I have 10, 5 folds, and I want to do a...a, create many many trees. So, I'm going to create trees that range from, in depth from '1' to '10', right! So, I don't know, for example whether depth three tree or a depth five tree or a depth ten tree is better. I don't know, right! So, what I am going to do is, I want to say, "All right, run them all from '1' to '10'." I run the decision, call the decision tree regressor and then, I fit the data, and making sure that the max depth is controlled over there. And then, I compute the mean of the cross validation score for those trees over there and store that in a table. So, let me see if I run this, I get trees of depth '1' that act. So, for each cross validation score, I am getting that cross validation score and storing them for each depth separately, right! So, depth '1' is negative '0.548', depth '2', '0.512', depth '3', '0.482', 0.482, blah, blah, etc. So, I get these 9 depth values and the idea here is to see whether these depth values or these different values that we're getting from the cross validation tests are reasonably similar.
In this case, we can see they are reasonably similar. If they're reasonably similar, then that means that we are a model that is likely to be more robust. So, what is the tree in this case? In this case, we don't really have a tree because we have many many trees and what we are really doing is, we are not generating a tree. We're generating many trees but we want to get an estimate for the average error of the model, right! That's what we try to do. So, the average error of the model is somewhere in this range here and also get an idea of what depth we should use for our...our tree, right! So, maybe our depth should be somewhere here, point three or four is probably a good depth to use in this stuff here, right! Because the error drops and then starts rising again as the tree gets bigger. And then, once we do that, we can pick a depth and we have a model and now...now we want to actually generate the...the tree itself. So, now we go back and no longer use a training and testing sample, because we've run the thing on each individual folds and we are saying we don't really want to use a training and testing here but we don't want our tree to be subject to our decision on where to-which ones to include in the training, which cases to include in the training and which to include in testing.
So, we run the...the...the data on the entire tree. Now, once we run the data on the entire tree, we are good to go, right! So, that's, that...that becomes our tree there. So, that's where we are in this thing here. That...that becomes our tree and then we use that for our model over here.

Video 4 (5:48): Classification Trees_Rocks and Mines Dataset

Let's look at classification trees. And so, for example, in our wine, sorry, mines and rock example, we want to figure out whether something is a mine or a rock. And, like we said before, if we can figure out the mines very well or the rocks very well, we may be all right. We may not really-we may not want to be accurate on both of them, okay! That's the point there. So, what we're going to do is we-in classification trees, we're looking at the misclassification cost, which is another measure called Gini that does that. So, let's walkthrough this stuff in our mines data set. So, we start by reading our data, of

course, as usual and then, we create our train test sample. I'm not going to do cross validation in this case, even though actually the data is much smaller. You can try that on your own. And, here we're going to pick our- get our decision tree classifier, and we're going to use the decision...decision tree classifier with a criterion of entropy. Entropy in this case is like how varied is our zeros and ones in these models, right! So, we run that. You can use Gini or entropy, either is fine but the both of them focus on misclassification. Entropy uses a slightly more complex algorithm to compute that, to compute the misclassification cost of the-in this thing here.

So, we run the tree, do the fitting, and now we want to run 'pydotplus'. So, we can see our graph, to take a look at it. So, let's go back to finder and look at 'mines' dot 'pdf', and that's correctly done at this time. So, I run this and I show this here. So, this is our tree here. So here, this is a measure of entropy now, right, '0.9848' and our goal is to minimize entropy, that is, to find subsets where the entropy is a lot lower. So, if you look for example, we go here, we get an entropy of zero, right! What does that mean? That means, all our marbles are blue, or all our marbles are red. In this case, all our values are in one set, okay! There are mines I guess in this set. All are...all are ones, right! So, we have '41' samples, out of '145' in this case, which are all mines. That's really good, right! So, if we use this, we know that we correctly- if...if the cost of misclassifying a mine, you know, the precision that this is-a precision is 100 percent here. If something that model says, a mine is actually a mine, then that matters to us. Then, this is actually a very good result for us and it tells us that the-so now reading for '10' should be greater than '0.1579'. For '26', should be greater than '0.8167'. For '8', should be greater than '0.1094'. Then, we get a cell that has, you know, a great result in this thing. Of course, this is on our, I'm assuming, check that, we are on a training sample, right! And, if you look at the other side, we have the impurity in our origin cell is '0.9848', and we notice that here, for example, the purity is '0.9945', right, this one. So, this is, with '25' 0s. This is not a good cell at all but we have another cell here with '11' of these.

So, actually we combine these two, we can get a rule that tells us either we follow this arc that goes from this, to this, to this, to this. And, from this to this, we can get '41', 51, 52 cases but we are fitting a little bit, right! We are trying to, we have to make sure it's not fitted, right, even we do that. And on the rock side, we find that we have, it's not so great. Well, maybe here it is great, actually. We have '26' rocks that identify greatly as rocks with impurity of '0'. So, this cell, if impurity is less than '0.1579' and...and sorry, if the variable '10' is less than '0.1579', and variable '30' is greater than '0.4623', we get a rocks correctly identified. So, we actually have a good-ish model over here, at least to the extent that it fits our data, right! So, that's good. We can go and now set up classification problems. So, we can do our 'confusion matrix' stuff over here, okay! So, we will run that and we can look at the 'confusion matrix' on the training and testing model. And then, to see how well our model has done, we have these trees. We want to check the 'ROC Curve' on the model. So, let's run that, and we look at the 'ROC Curve' over here that we get and notice because we don't have thresholds of '0', you know, where our dependent variable in this case is '0' and '1'. So, threshold doesn't matter here. So, we're getting a curve that looks like this. So, it looks like a bad curve but it's not a bad curve. The reason being that we have either a '0' or a...or a '1'. And [inaudible] in the middle.

So, really we're looking only at this point here. So, this gives us an area of the curve of '0.85' on the in sample 'ROC Curve'. and that's what the setting is, our threshold, because it doesn't...it doesn't mean that much here, right! And, an area of '0.78' on our sample 'ROC Curve'. So, it's really up to us now. We

can decide whether we like this better or our regression model results better, right! That's...that's the idea there. And we can pick between these 2 models and looking at the threshold, the costs, and all that other kind of stuff that we saw before, to decide whether, you know, one or the other methodology was better for us. You could also run a cross validation to see how valid it is, and there are other methods for improving the result on a decision tree. There's a gradient descent model, gradient boosting models that you gradient descent etc. to figure out whether to sort of work on your tree and make it better. We're not going to do all those because of the time but it's a good idea to take a look at that for yourself.

Video 5 (5:23): Clustering_Image Dataset

Clustering algorithms are an example of unsupervised learning. The difference between supervised learning and unsupervised learning is very straightforward. In supervised learning, we give our training cases, and we say that these cases have a bunch of independent variables and a dependent variable. And, we know the value of the dependent variable, so we ask it to, sort of, use the independent variables to find a curve or something that fits to the dependent variable. In unsupervised learning, we do something slightly different. We say, "All right! Here's our space of independent variables. Now, try and see how many features that we have, and now try and see what values of these features will make these cases, make subsets of these cases closer together by some estimate", okay! How...how do they fit closer together? So, we have...we have-you can think of having a vast net or, you know, data points sitting all over the place, and you want to find planes that cut through the space that can group these data points together in a similar sort of way. Okay, that's the idea.

They're called clusters, and they use only value of the feature space. Of course, what we do is, we again work our way through a training and testing sample, and in the training sample we give it the independent variables and it groups them. And then, we want to measure how well that's done by looking whether these groups map on to something in the real world that makes sense to us, right! So, for example, you could take physical characteristics of people and say can we differentiate between these physical characteristics and put them into two groups. When we put them into two groups, maybe we can figure out, men versus women, you know, something like that. And then, when we don't say that this...this case corresponds to a man or this set of features correspond to women, we let the...the algorithm use the features to differentiate between the two groups of people. A popular algorithm for doing clustering is k-means clustering. What it does is, it partitions the data space into clusters and it minimizes the distance between the mean of a cluster and the data points. So, every data point is sitting in n-dimensional space where each dimension is a...a feature. And so, you can measure the distance between 1 data point and another data point, and we want to minimize the...the fine clusters with a mean distance between data points and each cluster is...is minimum, really, right! So, we want to minimize that distance. And, what you need to know in advance is, how many clusters you're going to have in your data set, in...in your domain, right!

So, you want to know how many clusters, like if you're doing men versus women, you have, 2 clusters you know that. You don't want to like say, "Hey! Find me the number of-" You can't say find me the number of clusters, in that case. You need to actually tell the k-means algorithm, how many clusters to

use. So, to set that up before we get moving on this, you want to make sure you have your imports. So, the imports you want to do is, you're going to use a data set that comes with 'sklearn' called a digit recognition data set. So, I'll explain that in a second but let's just make sure we load it in. So, we have NumPy, of course, matplotlib of course, we inline it for any graphs that we draw. We load the digits data set. So, that's the data set we're going to use. And we're going to import from 'sklearn' preprocessing. We're going to import a package called 'scale', which we use to rescale in this case. In our example, we're going to rescale data into a normally distributed mean, 0 standard deviation, 1 data set. So, once we've done that we should be in good shape. So, what we'll do is, we run this to load the actual digits, and run equal 'load_digits' and we can see what we get here. Oops, forgot to do that. And, we get this stuff over here. So, what we're getting is, a optical recognition of handwritten digital-digits data set. And, the goal here is that we have a set of data that is handwritten 1s, 0s, 2s, 3s, 4s, 5s, 6, 7, 8, 9s, and we want to use machine learning to be able to...to...able to recognize, given a new sample. Given a new-somebody else writes a one and gives it to our machine, we want to be able to recognize it as a one. A handwritten recognition problem, right, that's the goal here. So, that's the data we're going to work with and that's the data here, right! So, what it really does is, it takes each digit and it pixilates it. So, you can think of roughly as something like this. You can take a...a digit, put a box around it, and put like, you know, depending on your resolution, put little pixels that point to this stuff. And if, you know, if there's a written part, then the pixel is lit up and, if there's a nonwritten part, it's not lit up. So, you can see which ones are lit up, which ones are not lit up, and that's what this is, sort of, representing over here with these numbers, this data and this thing you can see '1' '6'. So, '0's are you can think of unlit up parts and '1', '6', '10's are the lit-up parts in the-in that particular case. So, each one of these is a case, right, that's a case. Okay, so there are many many '0's and '1's here. So, that's the idea. So, our features are the pixels whether they're lit up or not and the categories we want to do are whether they're '0's, '1's, '2's, etc. And then, the last thing we can do is, we can scale the digits.
So, we will run the scaler and what happens then is that we get our digits from 0s and 1s, instead we have scaled them and normally distributed between mean 0 and standard deviation 1. So, that's our data set.

Video 6 (4:19): K-Means Clustering 1_Image Dateset.

So, let's do some fun stuff, let's render the digits and their associated values. So, we're going to use-write a function here, called 'print_digits', and what 'print_digits' is going to do is it's going to take the...the actual cases and show them on the screen. So, we're going to get them like this, okay. I am going to run that and show it to you here. So, these are the digits we are printing. This is the data that we saw earlier, right! And, we're using that data to actually render the digit. And, we are doing that using matplotlib here. So, again what we do is, we set up a figure and we set up a bunch of subplots and we're going to adjust these so that the-because we have 2 sets of subplots. We have the actual images 0, 1, 2, 3, which are the images we are working with. And, we have the corresponding dependent variable value ‘0’, ‘1’, ‘2’, ‘3’, because we know in our data, we know that this is a ‘0’, this is a ‘1’.

In the data, we know that we just want...want- we're not going to use that for the clustering, but we know that, we're going to use that for testing. So, we have that and we want to make sure that the '0's and '1's are aligned with those things. So, the purpose of the 'subplot_adjust' is to just make sure that they're aligned. If we don't do that, then we're going to get '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ending somewhere here, you know, ending where the four is, so that's not going to work very well. And then, we render-we work through this thing and add each image to the thing. We have the images here, and we create the stuff and add the text of the dependent variable value, the 4, 5, 6, 7, 8, 9 stuff, the actual values. And so, we have our images, we have our target values, and we know that there are 10, we're only going to show 10. We could show more because you know that, our data set is much larger, right! So, this is what we get and we get our image [inaudible]. This is kind of a neat way of showing this stuff. The next thing you want to do is break stuff into a training and testing sample, because that's what we always do.

So, that's again very straightforward. We take a 'test_size' of '25' percent and break our- use the 'train_test_split' function here and break it up into the-take the data and break it up and take the target and break it up into 2 different sets, and we're going to get our thing here. We can take a look at labels. So, the labels are-so the first case is a 5, the second case is a 2, third case is a 0, and that's what our y values are. And, if we want to look at the x values, they will be- 'x_train' is essentially our normalized data, the pixelated stuff normalized. So, we can look at the unique values of 'y_train' just to be on the safe side, and we find there are 10 of them which is right, '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'. And, that gives us our basic structure here. And, the last thing we do, which we do always is always the same thing, is we import the 'cluster' algorithm-the 'cluster' library module and create a 'k-means' clustering algorithm from that. We tell it to initialize it by doing some initialization stuff. So, what it-'k-means' works better when you initialize it, so we initialize it initially, we run this pre-algorithm, you can think of to do the initialization. And, to start with a-but what k-means does is it starts by randomly allocating the..the digits, right!

So, it'll say, "All right, I need 10 categories and that's the number of clusters ‘10’, and it'll randomly assign the data to ten categories." But, if you can, sort of, intelligently start off when you have some, you know, some knowledge about the data, like maybe the means inside the values and all that kind of stuff, then you're better off. So, you can use this 'k-meansplusplus' to actually start the algorithm at a better point, and so that's the 'k-means'. And then, once we do that we fit it. So, fitting is going to be, of course, actually running the algorithm and we've got this stuff running. So, that's our stuff now. So now, we want to actually work with and figure out how well our clustering has worked, and try to understand the-what we can, whether we're getting a good result or not.

Video 7 (4:59): K-Means Clustering 2_Image Dataset

So now, what we've done is we've fitted our model, right! So, we fitted our model. So, our model now has predictions on, not predictions but really it's assigned cluster numbers to groups of-to each individual data point. So, we've told our model to take ten clusters. So, it's assigned ten cluster numbers, 0, 1 to 9, to each individual data point. And, those numbers are in this-clf is our model, in the-in this

attribute called, labels underscore. So, what we're going to do now is we're going to take a look at our training data set, and look at the first 20 cases, and see what cluster numbers were assigned to each case. And, we look at the actual image itself. So, we're going to eyeball. We're going to see what the image is and see what the cluster number is attached to it. And note that the cluster number does not have to correspond to the image. Because-for example, if it takes everyone and assigns it cluster seven, then that's great...great.

Then, we know cluster seven means a '1', right! Because this doesn't know that we're looking at...that we are looking at '1', '2', '3', '4', '0', '1' to '9'. The machine doesn't know that, right! All it knows is that we are looking to differentiate between different data points. So, let's take a look at this. We call our 'print_digits' function, the function we wrote before on 20 of these things. And, we look at this stuff here and try to figure out, what, you know, whether it's doing this accurately or not. So, we look at the first case is a '5' and it puts cluster ‘1’ there. So, let's see if there are any other '5's. Going further down, we see that further down, way down here, there is another '5' and it's put in cluster ‘8’. And then, if you go further down, there's another '5' in cluster ‘8’. So, we have two '8's for a '5' and one '1' for a '5'.

So that's, you know, not great but we're only looking at 20 samples. We look at '2', the second item here, the number, the image is '2' and it's put in cluster ‘3’. So, let's look at if there are any more '2's. Yes, there's another '2' with ‘3’, another '2' with ‘3’. And, we go further on and the last thing, another '2' with ‘3’. Now, that's great. All the 2s that we are seeing in our first 20, it could be messed up later on are all 3s. So, we know that '3' is likely to be cluster ‘2’. Sorry, cluster ‘3’ is likely to be the image '2' and so on and so forth. Right! So, we can look at, say, '0'. '0's is in cluster ‘2’ and we find '0' cluster ‘2’, and well ‘2’ '0's, they're both in cluster ‘2’. So, that's, you know, just eyeballing here. It's not really solid scientific way of looking. We just want to see what...what we're getting out of this. So, that's our basic structure here. So, we say, "Yes, there is some in our first 20. Anyway, there's some discrimination because '2's are in cluster ‘3’, ‘5’s are generally in cluster ‘8’, '3's are-'7's are in cluster ‘6’, looking at this '6's are in cluster, well I can't say much about '6's." Oh! What we can see from this is what we can see from this-so, what we want to do then is we want to use the test sample, now to generate predictions. So, we've got our...our training sample. We've trained our model to differentiate among these things by looking at data points.

Now, we give it the testing sample and it's going to use the feature values, and the trained clustering that we did before to predict which cluster things belong to. So, we do-we run that that and we get a bunch of predictions and, of course, they are just numbers we won't actually see that. So, what we'll do is we'll look at 15 of these values here. We're calling our 'print_cluster' function again. We look at 15 cluster...cluster values for every y prediction for every cluster. So, our...our y's are predicting ‘0’, ‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, ‘9’ as our clusters. So, for each of those clusters, we're going to get the images that are included in that cluster, right! So, if the cluster number is 0, these are the images that are included. We're looking at cluster 0, we see there a bunch of '1's, quite a few, but then there's a '2', '8', '8', '3', '8', '2', '8'. So, that's not so great. And we only have 15 values are printing. If we look at cluster ‘1’, they're all '4's. Now, so that's great. That looks good. Cluster ‘4’ seems to be getting accurately assigned to cluster ‘1’. Cluster ‘2’ is all '0's. Cluster ‘3’ has all '2's.

Possibly, there's some of them that look vaguely like '1's so that's not clear. And there's ‘1’ here that looks like a '3', the second-last one. So, [inaudible] we don't know...we don't know for sure, but pretty

much over the board they're '2's. Cluster ‘4’ looks like they're almost all '6's. Cluster ‘5’ looks like they are mostly ‘5’. There is a- some, looks like a '7', not clear! People don't write very clearly, that's the bottom line. Cluster ‘6’ has mostly '7's. So, there is some sort of relationship here. But, our goal is to find and see whether these relationships are solid or not. So, that's what we want to do. We want to evaluate the model itself. So, let's see if you can do that.

Video 8 (7:21): K-Means Clustering 3_Evaluating the Model

For evaluating the model, we use something called an Adjusted rand index. It's a measure of the similarity between two groups, okay! So, the goal here is to-we have a group of actual test data that is the actual scores that we have, sorry, the actual...the actual numbers that we have, so we know that each case in our data set- we know what the number is, right! We know whether it's a 1 or a 2 or a 3, and we also have the predictions. That's what our model is predicting. And, we want to see what's the similarity between these 2 sets. You know, are they reasonably similar? We're not looking at actual values, but seeing whether the differentiation is similar or not. Okay, and you can look at that thing over there. So, if there's '0.0', then there's no similarity...no similarity at all and that's completely random. Any overlap in that- and ‘1.0’ indicates that the two groups are identical.
So, let's run that and we get '0.567' similarity. That's not so bad. It's a, you know, it's a-we've done a very basic model so it's kind of, it's not great...not great-it's not terrible. But, it's better than random because 0 would be random. So, we are pretty much not the way there [inaudible]. And again, since we have categories over here, we've got cluster numbers and all that kind of stuff, we can again draw a confusion matrix of sorts with this. So, this is going to be a little bit different, the confusion matrix. But, let's take a look at it and then, I'll explain it. So, it's a little bit confusing here. So, what this confusion matrix is doing is, it's- we have a cluster number and an actual prediction, right, so what this does is, each row in this corresponds to a number in the test sample. So, each row is-so, this is row ‘0’ in the test sample, okay! And each column is the cluster assigned to that case by the...by the model. So, what this is saying is that for our actual '0's, none of them were put in cluster ‘0’. None of them were put in cluster ‘1’. So, none of them into cluster ‘0’, None of them into cluster ‘1’, 43 were assigned to cluster ‘2’, and none of them into any other clusters. So, if you're looking at our '0's, we've done a great job of identifying them. If we get a cluster ‘2’, we are reasonably confident that it's a '0' or other, if our model is-let me rephrase that, if our model is seeing a '0', then it's going to put out inside cluster ‘2’.
So, that's a great number. Our testing sample, it's not the training sample. Can it...can it tell us confidently the other way? Well, almost! Because, if we find that we have cluster ‘2’, then the only case in cluster ‘2’ that's not identified as ‘0’ is this ‘1’ here which is '0', '1', '2' '3', ‘4 ‘, '5', '6'. In one case, it took a '6' that was actually a '6' and identified it as a, put it in cluster ‘2’ which is a '0', right! So with '0's, we've done a very good job. You know, so, given a '0', we can with, reasonable confidence and...and cluster number ‘2’, we can with extreme confidence actually, say, "Hey! That's a '0'." But, let's look at something else. Let's look at ‘7’, right! So, that's '0', '1', '2', '3', '4', '5', '6', '7', no, not '7'. Let's look at this ‘1’. So that's-all right, let's look at this ‘1’ here. This '0', '1', '1', '3'...'3'. Let's look at a '3', right! So, the '3'-

if this is a column, the row for '3's, so '3's get assigned to cluster ‘0’ in one case. '1', '2', '3'. Cluster ‘3’ in one case.

And so on and so forth. But cluster ‘8’ in 39 cases. So, in 39 cases, we're getting a 3 in cluster ‘8’. And, in 4 plus 1, 5 plus 1, 6 plus 1, 7 cases, we're getting a '3' in some other cluster. All right, so again that means that we have a precision essentially of 39 plus... 39 divided by 39 plus whatever the denominator is, right! 4, 5, 6, 7. However, if you look at the other way around, and we look at this, at the column, that is column number 8 over here, we find that in column 8, in cluster ‘8’, 39 of them are '3's, but ‘1’ is a '2', 16 are '5's, 11's are 8, and 40 are '9's. So, if given that we get a cluster value of ‘8’ for a figure, can we with confidence, say it's a '3'? Not really, right! Because, we have so many cases, where it could be a...a '9' or it could be a '8'. It could be-this thing [inaudible] in fact there are more ‘9’s than there are '3's in that cluster. So, that's...that's done, a very poor job of identifying the cluster correctly, right! So, this is what you want to do with the confusion matrix. You want to see how well it's actually figured out the clusters themselves. So, you look at this stuff and you can then, you know, decide for each digit, how well it's been doing, maybe try to get more data, or more or a higher pixel rate, or higher resolution, before you actually work and say, "Hey, we get a better [inaudible]."

But, we can see our 56 percent rand score is coming from the fact that we are, you know, doing well on some numbers. Like we are doing well on '0'. We are probably doing well on 0, 1, 2, 3, 4. Yup, we're doing well on 4, because 4 we have 0, 1, 2, 3, 4, right, '50' cases that are in cluster ‘1’, here, '50' cases in cluster ‘1’. And, only 2 plus 1, 3 plus 1, 4 cases that are 5 cases actually that are in other clusters. And even more important, cluster one doesn't contain anything except '4's. So, if you get a cluster ‘1’, you can say 'Hey, that's a 4, right!", even though you may not recognize every 4 correctly. So, that's how you sort of analyze the results of your clustering analysis and looking at this confusion matrix. Finally, what we can do is we can look at the graphical view of the clusters, kind of interesting. And, what we'll do here is, we'll run a principle component. We will take our- so the idea here is that a graph is two-dimensional because it fits on this page. Our data is n-dimensional because we have n features, right, as many features as we have.

So, what we can do is we can reduce our dimensions to 2 using the principle component analysis. So, we import this 'decomposition' from 'sklearn' and run 'pca' on that and say we want only 2 components from our training sample. And, once we've done that we can- let me draw this and show you, we get a really nice picture. It tells us-these are our data points reduced to 2 dimensions, and these are the clusters where they come in here. It will be nice when we look at this if you could see the boundaries were clearer, right! So we can see that some of the boundaries are reasonably clear, right! For example, there are very few on the boundary between the orange and the blue, and the orange and the purple, and the purple and the blue. But, if you look at the blue, the light blue and the green, and the light blue and the brown, and the light blue and the dark blue over there, you know and that area is very, very messy.

So, what you would like to see in a good- the result would be that very few points at the boundaries, you know so that the clusters are reasonably accurate, right! So, there are 10 clusters over here. So, that's the goal in clustering algorithm. A very brief introduction and so it should be up more about it.