K-Nearest Neighbors

Hello and welcome. In this video, we'll be covering the K-Nearest Neighbors algorithm. So, let's get started. Imagine that a telecommunications provider has segmented his customer base by service usage patterns, categorizing the customers into four groups. If demographic data can be used to predict group membership the, company can customize offers for individual perspective customers. This is a classification problem. That is; given the dataset with predefined labels, we need to build a model to be used to predict the class of a new or unknown case. The example focuses on using demographic data, such as; region, age, and marital status to predict usage patterns. The target field called custcat has four possible values that correspond to the four customer groups as follows; Basic Service, E Service, Plus Service, and Total Service. Our objective is to build a classifier. For example, using the row zero to seven to predict the class of row eight, we will use a specific type of classification called K-Nearest Neighbor. Just for sake of demonstration, let's use only two fields as predictors specifically, age and income, and then, plot the customers based on their group membership. Now, let's say that we have a new customer. For example, record number eight, with a known age and income. How can we find the class of this customer? Can we find one of the closest cases and assign the same class label to our new customer? Can we also say that the class of our new customer is most probably group four i.e Total Service, because it's nearest neighbor is also of class four? Yes, we can. In fact, it is the first nearest neighbor. Now, the question is, to what extent can we trust our judgment which is based on the first nearest neighbor? It might be a poor judgment especially, if the first nearest neighbor is a very specific case or an outlier, correct? Now, let's look at our scatter plot again. Rather than choose the first nearest neighbor, what if we chose the five nearest neighbors and did a majority vote among them to define the class of our new customer? In this case, we'd see that three out of five nearest neighbors tell us to go for class three, which is Plus Service. Doesn't this make more sense? Yes. In fact, it does. In this case, the value of K in the K-Nearest Neighbors algorithm is five. This example highlights the intuition behind the K-Nearest Neighbors algorithm. Now, let's define the K Nearest Neighbors. The K-Nearest Neighbors algorithm is a classification algorithm that takes a bunch of labeled points and uses them to learn how to label other points. This algorithm classifies cases based on their similarity to other cases. In K-Nearest Neighbors, data points that are near each other are said to be neighbors. K-Nearest Neighbors is based on this paradigm. Similar cases with the same class labels are near each other. Thus, the distance between two cases is a measure of their dissimilarity. There are different ways to calculate the similarity or conversely, the distance or dissimilarity of two data points. For example, this can be done using Euclidean distance. Now, let's see how the K-Nearest Neighbors algorithm actually works. In a classification problem, the K-Nearest Neighbors algorithm works as follows. One, pick a value for K. Two, calculate the distance from the new case hold out from each of the cases in the dataset. Three, search for the K-observations in the training data that are nearest to the measurements of the unknown data point. And four, predict the response of the unknown data point using the most popular response value from the K-Nearest Neighbors. There are two parts in this algorithm that might be a bit confusing. First, how to select the correct K and second, how to compute the similarity between cases, for example, among customers. Let's first start with the second concern. That is; how can we calculate the similarity between two data points? Assume that we have two customers, customer one and customer two, and for a moment, assume that these two customers have only one feature, H. We can easily use a specific type of Minkowski distance to calculate the distance of these two customers, it is indeed the Euclidean distance. Distance of X\_1 from X\_2 is root of 34 minus 30 to power of two, which is four. What about if we have more than one feature? For example, age and income. If we have income and age for each customer, we can still use the same formula but this time, we're using it in a two dimensional space. We can also use the same distance matrix for multidimensional vectors. Of course, we have to normalize our feature set to get the accurate dissimilarity measure. There are other dissimilarity measures as well that can be used for this purpose but as mentioned, it is highly dependent on datatype and also the domain that classification is done for it. As mentioned, K and K-Nearest Neighbors is the number of nearest neighbors to examine. It is supposed to be specified by the user. So, how do we choose the right K? Assume that we want to find the class of the customer noted as question mark on the chart. What happens if we choose a very low value of K? Let's say, K equals one. The first nearest point would be blue, which is class one. This would be a bad prediction, since more of the points around it are magenta or class four. In fact, since its nearest neighbor is blue we can say that we capture the noise in the data or we chose one of the points that was an anomaly in the data. A low value of K causes a highly complex model as well, which might result in overfitting of the model. It means the prediction process is not generalized enough to be used for out-of-sample cases. Out-of-sample data is data that is outside of the data set used to train the model. In other words, it cannot be trusted to be used for prediction of unknown samples. It's important to remember that overfitting is bad, as we want a general model that works for any data, not just the data used for training. Now, on the opposite side of the spectrum, if we choose a very high value of K such as K equals 20, then the model becomes overly generalized. So, how can we find the best value for K? The general solution is to reserve a part of your data for testing the accuracy of the model. Once you've done so, choose K equals one and then use the training part for modeling and calculate the accuracy of prediction using all samples in your test set. Repeat this process increasing the K and see which K is best for your model. For example, in our case, K equals four will give us the best accuracy. Nearest neighbors analysis can also be used to compute values for a continuous target. In this situation, the average or median target value of the nearest neighbors is used to obtain the predicted value for the new case. For example, assume that you are predicting the price of a home based on its feature set, such as number of rooms, square footage, the year it was built, and so on. You can easily find the three nearest neighbor houses of course not only based on distance but also based on all the attributes and then predict the price of the house as the medium of neighbors. This concludes this video. Thanks for watching

**Evaluation metrics for classifiers**

Hello, and welcome! In this video, we’ll be covering evaluation metrics for classifiers. So let’s get started. Evaluation metrics explain the performance of a model. Let’s talk more about the model evaluation metrics that are used for classification. Imagine that we have an historical dataset which shows the customer churn for a telecommunication company. We have trained the model, and now we want to calculate its accuracy using the test set. We pass the test set to our model, and we find the predicted labels. Now the question is, “How accurate is this model?” Basically, we compare the actual values in the test set with the values predicted by the model, to calculate the accuracy of the model. Evaluation metrics provide a key role in the development of a model, as they provide insight to areas that might require improvement. There are different model evaluation metrics but we just talk about three of them here, specifically: Jaccard index, F1-score, and Log Loss. Let’s first look at one of the simplest accuracy measurements, the Jaccard index -- also known as the Jaccard similarity coefficient. Let’s say y shows the true labels of the churn dataset. And y ̂ shows the predicted values by our classifier. Then we can define Jaccard as the size of the intersection divided by the size of the union of two label sets. For example, for a test set of size 10, with 8 correct predictions, or 8 intersections, the accuracy by the Jaccard index would be 0.66. If the entire set of predicted labels for a sample strictly matches with the true set of labels, then the subset accuracy is 1.0; otherwise it is 0.0. Another way of looking at accuracy of classifiers is to look at a confusion matrix. For example, let’s assume that our test set has only 40 rows. This matrix shows the corrected and wrong predictions, in comparison with the actual labels. Each confusion matrix row shows the Actual/True labels in the test set, and the columns show the predicted labels by classifier. Look at the first row. The first row is for customers whose actual churn value in the test set is 1. As you can calculate, out of 40 customers, the churn value of 15 of them is 1. And out of these 15, the classifier correctly predicted 6 of them as 1, and 9 of them as 0. This means that for 6 customers, the actual churn value was 1, in the test set, and the classifier also correctly predicted those as 1. However, while the actual label of 9 customers was 1, the classifier predicted those as 0, which is not very good. We can consider this as an error of the model for the first row. What about the customers with a churn value 0? Let’s look at the second row. It looks like there were 25 customers whose churn value was 0. The classifier correctly predicted 24 of them as 0, and one of them wrongly predicted as 1. So, it has done a good job in predicting the customers with a churn value of 0. A good thing about the confusion matrix is that it shows the model’s ability to correctly predict or separate the classes. In the specific case of a binary classifier, such as this example, we can interpret these numbers as the count of true positives, false positives, true negatives, and false negatives. Based on the count of each section, we can calculate the precision and recall of each label. Precision is a measure of the accuracy, provided that a class label has been predicted. It is defined by: precision = True Positive / (True Positive + False Positive). And Recall is the true positive rate. It is defined as: Recall = True Positive / (True Positive + False Negative). So, we can calculate the precision and recall of each class. Now we’re in the position to calculate the F1 scores for each label, based on the precision and recall of that label. The F1 score is the harmonic average of the precision and recall, where an F1 score reaches its best value at 1 (which represents perfect precision and recall) and its worst at 0. It is a good way to show that a classifier has a good value for both recall and precision. It is defined using the F1-score equation. For example, the F1-score for class 0 (i.e. churn=0), is 0.83, and the F1-score for class 1 (i.e. churn=1), is 0.55. And finally, we can tell the average accuracy for this classifier is the average of the F1-score for both labels, which is 0.72 in our case. Please notice that both Jaccard and F1-score can be used for multi-class classifiers as well, which is out of scope for this course. Now let's look at another accuracy metric for classifiers. Sometimes, the output of a classifier is the probability of a class label, instead of the label. For example, in logistic regression, the output can be the probability of customer churn, i.e., yes (or equals to 1). This probability is a value between 0 and 1. Logarithmic loss (also known as Log loss) measures the performance of a classifier where the predicted output is a probability value between 0 and 1. So, for example, predicting a probability of 0.13 when the actual label is 1, would be bad and would result in a high log loss. We can calculate the log loss for each row using the log loss equation, which measures how far each prediction is, from the actual label. Then, we calculate the average log loss across all rows of the test set. It is obvious that more ideal classifiers have progressively smaller values of log loss. So, the classifier with lower log loss has better accuracy. Thanks for watching!

**building decision trees**

Hello and welcome. In this video, we'll be covering the process of building decision trees. So, let's get started. Consider the drug data set again. The question is, how do we build a decision tree based on that data set? Decision trees are built using recursive partitioning to classify the data. Let's say we have 14 patients in our data set, the algorithm chooses the most predictive feature to split the data on. What is important in making a decision tree, is to determine which attribute is the best or more predictive to split data based on the feature. Let's say we pick cholesterol as the first attribute to split data, it will split our data into two branches. As you can see, if the patient has high cholesterol we cannot say with high confidence that drug B might be suitable for him. Also, if the patient's cholesterol is normal, we still don't have sufficient evidence or information to determine if either drug A or drug B is in fact suitable. It is a sample of bad attributes selection for splitting data. So, let's try another attribute. Again, we have our 14 cases, this time we picked the sex attribute of patients. It will split our data into two branches, male and female. As you can see, if the patient is female, we can say drug B might be suitable for her with high certainty. But if the patient is male, we don't have sufficient evidence or information to determine if drug A or drug B is suitable. However, it is still a better choice in comparison with the cholesterol attribute because the result in the nodes are more pure. It means nodes that are either mostly drug A or drug B. So, we can say the sex attribute is more significant than cholesterol, or in other words it's more predictive than the other attributes. Indeed, predictiveness is based on decrease in impurity of nodes. We're looking for the best feature to decrease the impurity of patients in the leaves, after splitting them up based on that feature. So, the sex feature is a good candidate in the following case because it almost found the pure patients. Let's go one step further. For the male patient branch, we again test other attributes to split the sub-tree. We test cholesterol again here, as you can see it results in even more pure leaves. So we can easily make a decision here. For example, if a patient is male and his cholesterol is high, we can certainly prescribe drug A, but if it is normal, we can prescribe drug B with high confidence. As you might notice, the choice of attribute to split data is very important and it is all about purity of the leaves after the split. A node in the tree is considered pure if in 100 percent of the cases, the nodes fall into a specific category of the target field. In fact, the method uses recursive partitioning to split the trading records into segments by minimizing the impurity at each step. Impurity of nodes is calculated by entropy of data in the node. So, what is entropy? Entropy is the amount of information disorder or the amount of randomness in the data. The entropy in the node depends on how much random data is in that node and is calculated for each node. In decision trees, we're looking for trees that have the smallest entropy in their nodes. The entropy is used to calculate the homogeneity of the samples in that node. If the samples are completely homogeneous, the entropy is zero and if the samples are equally divided it has an entropy of one. This means if all the data in a node are either drug A or drug B, then the entropy is zero, but if half of the data or drug A and other half or B then the entropy is one. You can easily calculate the entropy of a node using the frequency table of the attribute through the entropy formula where P is for the proportion or ratio of a category, such as drug A or B. Please remember though that you don't have to calculate these as it's easily calculated by the libraries or packages that you use. As an example, let's calculate the entropy of the data set before splitting it. We have nine occurrences of drug B and five of drug A. You can embed these numbers into the entropy formula to calculate the impurity of the target attribute before splitting it. In this case, it is 0.94. So, what is entropy after splitting? Now, we can test different attributes to find the one with the most predictiveness, which results in two more pure branches. Let's first select the cholesterol of the patient and see how the data gets split based on its values. For example, when it is normal we have six for drug B, and two for drug A. We can calculate the entropy of this node based on the distribution of drug A and B which is 0.8 in this case. But, when cholesterol is high, the data is split into three for drug B and three for drug A. Calculating it's entropy, we can see it would be 1.0. We should go through all the attributes and calculate the entropy after the split and then choose the best attribute. Okay. Let's try another field. Let's choose the sex attribute for the next check. As you can see, when we use the sex attribute to split the data, when its value is female, we have three patients that responded to drug B and four patients that responded to drug A. The entropy for this node is 0.98 which is not very promising. However, on the other side of the branch, when the value of the sex attribute is male, the result is more pure with sex for drug B and only one for drug A. The entropy for this group is 0.59. Now, the question is between the cholesterol and sex attributes which one is a better choice? Which one is better at the first attribute to divide the data-set into two branches? Or in other words, which attribute results in more pure nodes for our drugs? Or in which tree do we have less entropy after splitting rather than before splitting? The sex attribute with entropy of 0.98 and 0.59 or the cholesterol attribute with entropy of 0.81 and 1.0 in it's branches. The answer is the tree with the higher information gain after splitting. So, what is information gain? Information gain is the information that can increase the level of certainty after splitting. It is the entropy of a tree before the split minus the weighted entropy after the split by an attribute. We can think of information gain and entropy as opposites. As entropy or the amount of randomness decreases, the information gain or amount of certainty increases and vice versa. So, constructing a decision tree is all about finding attributes that return the highest information gain. Let's see how information gain is calculated for the sex attribute. As mentioned, the information gained is the entropy of the tree before the split minus the weighted entropy after the split. The entropy of the tree before the split is 0.94, the portion of female patients is seven out of 14 and its entropy is 0.985. Also, the portion of men is seven out of 14 and the entropy of the male node is 0.592. The result of a square bracket here is the weighted entropy after the split. So, the information gain of the tree if we use the sex attribute to split the data set is 0.151. As you could see, we will consider the entropy over the distribution of samples falling under each leaf node and we'll take a weighted average of that entropy weighted by the proportion of samples falling under that leave. We can calculate the information gain of the tree if we use cholesterol as well. It is 0.48. Now, the question is, which attribute is more suitable? Well, as mentioned, the tree with the higher information gained after splitting, this means the sex attribute. So, we select the sex attribute as the first splitter. Now, what is the next attribute after branching by the sex attribute? Well, as you can guess, we should repeat the process for each branch and test each of the other attributes to continue to reach the most pure leaves. This is the way you build a decision tree. Thanks for watching.

**Logistic Regression**

Hello and welcome. In this video, we'll learn a machine learning method called Logistic Regression which is used for classification. In examining this method, we'll specifically answer these three questions. What is logistic regression? What kind of problems can be solved by logistic regression? In which situations do we use logistic regression? So let's get started. Logistic regression is a statistical and machine learning technique for classifying records of a dataset based on the values of the input fields. Let's say we have a telecommunication dataset that we'd like to analyze in order to understand which customers might leave us next month. This is historical customer data where each row represents one customer. Imagine that you're an analyst at this company and you have to find out who is leaving and why? You'll use the dataset to build a model based on historical records and use it to predict the future churn within the customer group. The dataset includes information about services that each customer has signed up for, customer account information, demographic information about customers like gender and age range and also customers who've left the company within the last month. The column is called churn. We can use logistic regression to build a model for predicting customer churn using the given features. In logistic regression, we use one or more independent variables such as tenure, age and income to predict an outcome, such as churn, which we call the dependent variable representing whether or not customers will stop using the service. Logistic regression is analogous to linear regression, but tries to predict a categorical or discrete target field instead of a numeric one. In linear regression, we might try to predict a continuous value of variables such as the price of a house, blood pressure of a patient, or fuel consumption of a car. But in logistic regression, we predict a variable which is binary such as yes/no, true/false, successful or not successful, pregnant/not pregnant, and so on, all of which can be coded as zero or one. In logistic regression dependent variables should be continuous. If categorical, they should be dummy or indicator coded. This means we have to transform them to some continuous value. Please note that logistic regression can be used for both binary classification and multi-class classification. But for simplicity in this video, we'll focus on binary classification. Let's examine some applications of logistic regression before we explain how they work. As mentioned, logistic regression is a type of classification algorithm, so it can be used in different situations. For example, to predict the probability of a person having a heart attack within a specified time period, based on our knowledge of the person's age, sex and body mass index. Or to predict the chance of mortality and an injured patient or to predict whether a patient has a given disease such as diabetes based on observed characteristics of that patient such as weight, height, blood pressure and results of various blood tests and so on. In a marketing context, we can use it to predict the likelihood of a customer purchasing a product or halting a subscription as we've done in our churn example. We can also use logistic regression to predict the probability of failure of a given process, system or product. We can even use it to predict the likelihood of a homeowner defaulting on a mortgage. These are all good examples of problems that can be solved using logistic regression. Notice that in all these examples not only do we predict the class of each case, we also measure the probability of a case belonging to a specific class. There are different machine algorithms which can classify or estimate a variable. The question is, when should we use logistic regression? Here are four situations in which logistic regression is a good candidate. First, when the target field in your data is categorical or specifically is binary. Such as zero/one, yes/no, churn or no churn, positive/negative and so on. Second, you need the probability of your prediction. For example, if you want to know what the probability is of a customer buying a product. Logistic regression returns a probability score between zero and one for a given sample of data. In fact, logistic regression predicts the probability of that sample and we map the cases to a discrete class based on that probability. Third, if your data is linearly separable. The decision boundary of logistic regression is a line or a plane or a hyper plane. A classifier will classify all the points on one side of the decision boundary as belonging to one class, and all those on the other side as belonging to the other class. For example, if we have just two features and they're not applying any polynomial processing we can obtain an inequality like Theta zero plus Theta 1x1 plus theta 2x2 is greater than zero, which is a half-plane easily plotable. Please note that in using logistic regression, we can also achieve a complex decision boundary using polynomial processing as well, which is out of scope here you'll get more insight from decision boundaries when you understand how logistic regression works. Fourth, you need to understand the impact of a feature. You can select the best features based on the statistical significance of the logistic regression model coefficients or parameters. That is, after finding the optimum parameters, a feature X with the weight Theta one close to zero has a smaller effect on the prediction than features with large absolute values of Theta one. Indeed, it allows us to understand the impact an independent variable has on the dependent variable while controlling other independent variables. Let's look at our dataset again. We defined the independent variables as X and dependent variable as Y. Notice, that for the sake of simplicity we can code the target or dependent values to zero or one. The goal of logistic regression is to build a model to predict the class of each sample which in this case is a customer, as well as the probability of each sample belonging to a class. Given that, let's start to formalize the problem. X is our dataset in the space of real numbers of m by n. That is, of m dimensions or features and n records. Y is the class that we want to predict, which can be either zero or one. Ideally, a logistic regression model so-called Y hat can predict that the class of the customer is one, given its features X. It can also be shown quite easily that the probability of a customer being in class zero can be calculated as one minus the probability that the class of the customer is one. Thanks for watching this video.

**the difference between linear regression and logistic regression**

Hello and welcome. In this video, we will learn the difference between linear regression and logistic regression. We go over linear regression and see why it cannot be used properly for some binary classification problems. We also look at the sigmoid function, which is the main part of logistic regression. Let's start. Let's look at the telecommunication dataset again. The goal of logistic regression is to build a model to predict the class of each customer and also the probability of each sample belonging to a class. Ideally, we want to build a model, y hat, that can estimate that the class of a customer is one given its feature is x. I want to emphasize that y is the label's vector, also called actual values, that we would like to predict and y hat is the vector of the predicted values by our model. Mapping the class labels to integer numbers, can we use linear regression to solve this problem? First, let's recall how linear regression works to better understand logistic regression. Forget about the churn prediction for a minute and assume our goal is to predict the income of customers in the dataset. This means that instead of predicting churn, which is a categorical value, let's predict income, which is a continuous value. So, how can we do this? Let's select an independent variable such as customer age and predict the dependent variable such as income. Of course, we can have more features but for the sake of simplicity, let's just take one feature here. We can plot it and show age as an independent variable and income as the target value we would like to predict. With linear regression, you can fit a line or polynomial through the data. We can find this line through training our model or calculating it mathematically based on the sample sets. We'll say, this is a straight line through the samples set. This line has an equation shown as a plus bx1. Now, use this line to predict the continuous value, y. That is, use this line to predict the income of an unknown customer based on his or her age, and it is done. What if we want to predict churn? Can we use the same technique to predict a categorical field such as churn? Okay, let's see. Say, we're given data on customer churn and our goal this time is to predict the churn of customers based on their age. We have a feature, age, denoted as x1, and a categorical feature, churn, with two classes, churn is yes and churn is no. As mentioned, we can map yes and no to integer values zero and one. How can we model it now? Well, graphically, we could represent our data with a scatterplot, but this time, we have only two values for the y-axis. In this plot, class zero is denoted in red and class one is denoted in blue. Our goal here is to make a model based on existing data to predict if a new customer is red or blue. Let's do the same technique that we used for linear regression here to see if we can solve the problem for a categorical attribute such as churn. With linear regression, you again can fit a polynomial through the data, which is shown traditionally as a plus bx. This polynomial can also be shown traditionally as Theta0 plus Theta1 x1. This line has two parameters which are shown with vector Theta where the values of the vector are Theta0 and Theta1. We can also show the equation of this line formally as Theta transpose x. Generally, we can show the equation for a multidimensional space as Theta transpose x, where Theta is the parameters of the line in two-dimensional space or parameters of a plane in three-dimensional space, and so on. As Theta is a vector of parameters and is supposed to be multiplied by x, it is shown conventionally as transpose Theta. Theta is also called the weights factor or confidences of the equation, with both these terms used interchangeably and X is the feature set which represents a customer. Anyway, given a dataset, all the feature sets x Theta parameters can be calculated through an optimization algorithm or mathematically, which results in the equation of the fitting line. For example, the parameters of this line are minus one and 0.1 and the equation for the line is minus one plus 0.1 x1. Now, we can use this regression line to predict the churn of a new customer. For example, for our customer or, let's say, a data point with x value of age equals 13, we can plug the value into the line formula and the y value is calculated and returns a number. For instance, for p\_1 point, we have Theta transpose x equals minus 1 plus 0.1 times x1, equals minus 1 plus 0.1 times 13, equals 0.3. We can show it on our graph. Now, we can define a threshold here, for example, at 0.5 to define the class. So, we write a rule here for our model, y hat, which allows us to separate class zero from class one. If the value of Theta transpose x is less than 0.5, then the class is zero. Otherwise, if the value of Theta transpose x is more than 0.5, then the class is one. Because our customers y value is less than the threshold, we can say it belongs to class zero based on our model. But there is one problem here. What is the probability that this customer belongs to class zero? As you can see, it's not the best model to solve this problem. Also, there are some other issues which verify that linear regression is not the proper method for classification problems. So, as mentioned, if we use the regression line to calculate the class of a point, it always returns a number such as three or negative two, and so on. Then, we should use a threshold, for example, 0.5, to assign that point to either class of zero or one. This threshold works as a step function that outputs zero or one regardless of how big or small, positive or negative the input is. So, using the threshold, we can find the class of a record. Notice that in the step function, no matter how big the value is, as long as it's greater than 0.5, it simply equals one and vice versa. Regardless of how small the value y is, the output would be zero if it is less than 0.5. In other words, there is no difference between a customer who has a value of one or 1,000. The outcome would be one. Instead of having this step function, wouldn't it be nice if we had a smoother line, one that would project these values between zero and one? Indeed, the existing method does not really give us the probability of a customer belonging to a class, which is very desirable. We need a method that can give us the probability of falling in the class as well. So, what is the scientific solution here? Well, if instead of using Theta transpose x, we use a specific function called sigmoid, then sigmoid of Theta transpose x gives us the probability of a point belonging to a class instead of the value of y directly. I'll explain this sigmoid function in a second, but for now, please except that it will do the trick. Instead of calculating the value of Theta transpose x directly, it returns the probability that a Theta transpose x is very big or very small. It always returns a value between 0 and 1, depending on how large the Theta transpose x actually is. Now, our model is sigmoid of Theta transpose x, which represents the probability that the output is 1 given x. Now, the question is, what is the sigmoid function? Let me explain in detail what sigmoid really is. The sigmoid function, also called the logistic function, resembles the step function and is used by the following expression in the logistic regression. The sigmoid function looks a bit complicated at first, but don't worry about remembering this equation, it'll make sense to you after working with it. Notice that in the sigmoid equation, when Theta transpose x gets very big, the e power minus Theta transpose x in the denominator of the fraction becomes almost 0 and the value of the sigmoid function gets closer to 1. If Theta transpose x is very small, the sigmoid function gets closer to 0. Depicting on the in sigmoid plot, when Theta transpose x gets bigger, the value of the sigmoid function gets closer to 1. Also, if the Theta transpose x is very small, the sigmoid function gets closer to 0. So, the sigmoid functions output is always between 0 and 1, which makes it proper to interpret the results as probabilities. It is obvious that when the outcome of the sigmoid function gets closer to 1, the probability of y equals 1 given x goes up. In contrast, when the sigmoid value is closer to 0, the probability of y equals 1 given x is very small. So what is the output of our model when we use the sigmoid function? In logistic regression, we model the probability that an input, x, belongs to the default class y equals 1, and we can write this formally as probability of y equals 1 given x. We can also write probability of y belongs to class 0 given x is 1 minus probability of y equals 1 given x. For example, the probability of a customer staying with the company can be shown as probability of churn equals 1 given a customer's income and age, which can be, for instance, 0.8. The probability of churn is 0 for the same customer given a customer's income and age can be calculated as 1 minus 0.8 equals 0.2. So, now our job is to train the model to set its parameter values in such a way that our model is a good estimate of probability of y equals 1 given x. In fact, this is what a good classifier model built by logistic regression is supposed to do for us. Also, it should be a good estimate of probability of y belongs to class 0 given x that can be shown as 1 minus sigmoid of Theta transpose x. Now, the question is, how can we achieve this? We can find Theta through the training process. So, let's see what the training process is. Step one, initialize Theta vector with random values as with most machine learning algorithms. For example, minus 1 or 2. Step two, calculate the model output, which is sigmoid of Theta transpose x. For example, customer in your training set. X and Theta transpose x is the feature vector values. For example, the age and income of the customer, for instance, 2 and 5, and Theta is the confidence or weight that you've set in the previous step. The output of this equation is the prediction value, in other words, the probability that the customer belongs to class 1. Step three, compare the output of our model, y hat, which could be a value of, let's say, 0.7, with the actual label of the customer, which is for example, 1, for churn. Then, record the difference as our model's error for this customer, which would be 1 minus 0.7, which of course, equals 0.3. This is the error for only one customer out of all the customers in the training set. Step four, calculate the error for all customers as we did in the previous steps and add up these errors. The total error is the cost of your model and is calculated by the models cost function. The cost function, by the way, basically represents how to calculate the error of the model which is the difference between the actual and the models predicted values. So, the cost shows how poorly the model is estimating the customers labels. Therefore, the lower the cost, the better the model is at estimating the customers labels correctly. So, what we want to do is to try to minimize this cost. Step five, but because the initial values for Theta were chosen randomly, it's very likely that the cost function is very high, so we change the Theta in such a way to hopefully reduce the total cost. Step six, after changing the values of Theta, we go back to step two, then we start another iteration and calculate the cost of the model again. We keep doing those steps over and over, changing the values of Theta each time until the cost is low enough. So, this brings up two questions. First, how can we change the values of Theta so that the cost is reduced across iterations? Second, when should we stop the iterations? There are different ways to change the values of Theta, but one of the most popular ways is gradient descent. Also, there are various ways to stop iterations, but essentially you stop training by calculating the accuracy of your model and stop it when it's satisfactory. Thanks for watching this video.

**training a logistic regression model**

Hello and welcome. In this video, we will learn more about training a logistic regression model. Also, we will be discussing how to change the parameters of the model to better estimate the outcome. Finally, we talk about the cost function and gradient descent in logistic regression as a way to optimize the model. So, let's start. The main objective of training and logistic regression is to change the parameters of the model, so as to be the best estimation of the labels of the samples in the dataset. For example, the customer churn. How do we do that? In brief, first we have to look at the cost function, and see what the relation is between the cost function and the parameters theta. So, we should formulate the cost function, then using the derivative of the cost function we can find how to change the parameters to reduce the cost or rather the error. Let's dive into it to see how it works. But before I explain it I should highlight for you that it needs some basic mathematical background to understand it. However, you shouldn't worry about it as most data science languages like Python, R and Scala have some packages or libraries that calculate these parameters for you. So, let's take a look at it. Let's first find the cost function equation for a sample case. To do this we can use one of the customers in the churn problem. There's normally a general equation for calculating the cost. The cost function is the difference between the actual values of y and our model output y hat. This is a general rule for most cost functions in machine learning. We can show this as the cost of our model comparing it with actual labels, which is the difference between the predicted value of our model and actual value of the target field, where the predicted value of our model is sigmoid of theta transpose x. Usually the square of this equation is used because of the possibility of the negative result and for the sake of simplicity, half of this value is considered as the cost function through the derivative process. Now, we can write the cost function for all the samples in our training set. For example, for all customers we can write it as the average sum of the cost functions of all cases. It is also called the mean squared error and as it is a function of a parameter vector theta it is shown as J of theta. Okay good, we have the cost function. Now, how do we find or set the best weights or parameters that minimize this cost function? The answer is, we should calculate the minimum point of this cost function and it will show us the best parameters for our model. Although we can find the minimum point of a function using the derivative of a function, there's not an easy way to find the global minimum point for such an equation. Given this complexity, describing how to reach the global minimum for this equation is outside the scope of this video. So, what is the solution? Well we should find another cost function instead, one which has the same behavior but it's easier to find its minimum point. Let's plot the desirable cost function for our model. Recall that our model is y hat. Our actual value is y which equals zero or one, and our model tries to estimate it as we want to find a simple cost function for our model. For a moment assume that our desired value for y is one. This means our model is best if it estimates y equals one. In this case, we need a cost function that returns zero if the outcome of our model is one, which is the same as the actual label. And the cost should keep increasing as the outcome of our model gets farther from one. And cost should be very large if the outcome of our model is close to zero. We can see that the minus log function provides such a cost function for us. It means if the actual value is one and the model also predicts one, the minus log function returns zero cost. But if the prediction is smaller than one, the minus log function returns a larger cost value. So, we can use the minus log function for calculating the cost of our logistic regression model. So, if you recall, we previously noted that in general it is difficult to calculate the derivative of the cost function. Well, we can now change it with the minus log of our model. We can easily prove that in the case that desirable y is one, the cost can be calculated as minus log y hat, and in the case that desirable y is zero the cost can be calculated as minus log one minus y hat. Now, we can plug it into our total cost function and rewrite it as this function. So, this is the logistic regression cost function. As you can see for yourself it penalizes situations in which the class is zero and the model output is one, and vice versa. Remember however that y hat does not return a class as output but it's a value of zero or one which should be assumed as a probability. Now, we can easily use this function to find the parameters of our model in such a way as to minimize the cost. Okay, let's recap what we have done. Our objective was to find a model that best estimates the actual labels. Finding the best model means finding the best parameters theta for that model. So, the first question was, how do we find the best parameters for our model. Well, by finding and minimizing the cost function of our model. In other words, to minimize the J of theta we just defined. The next question is, how do we minimize the cost function? The answer is, using an optimization approach. There are different optimization approaches, but we use one of the most famous and effective approaches here, gradient descent. The next question is, what is gradient descent? Generally, gradient descent is an iterative approach to finding the minimum of a function. Specifically in our case gradient descent is a technique to use the derivative of a cost function to change the parameter values to minimize the cost or error. Let's see how it works. The main objective of gradient descent is to change the parameter values so as to minimize the cost. How can gradient descent do that? Think of the parameters or weights in our model to be in a two-dimensional space. For example, theta one, theta two for two feature sets, age and income. Recall the cost function, J, that we discussed in the previous slides. We need to minimize the cost function J which is a function of variables theta one and theta two. So, let's add a dimension for the observed cost, or error, J function. Let's assume that if we plot the cost function based on all possible values of theta one, theta two, we can see something like this. It represents the error value for different values of parameters, that is error which is a function of the parameters. This is called your error curve or error bowl of your cost function. Recall that we want to use this error bowl to find the best parameter values that result in minimizing the cost value. Now, the question is, which point is the best point for your cost function. Yes, you should try to minimize your position on the error curve. So, what should you do? You have to find the minimum value of the cost by changing the parameters. But which way? Will you add some value to your weights or deduct some value? And how much would that value be? You can select random parameter values that locate a point on the bowl. You can think of our starting point being the yellow point. You change the parameters by delta theta one and delta theta two, and take one step on the surface. Let's assume we go down one step in the bowl. As long as we are going downwards we can go one more step. The steeper the slope the further we can step, and we can keep taking steps. As we approach the lowest point the slope diminishes, so we can take smaller steps until we reach a flat surface. This is the minimum point of our curve and the optimum theta one, theta two. What are these steps really? I mean in which direction should we take these steps to make sure we descend, and how big should the steps be? To find the direction and size of these steps, in other words to find how to update the parameters, you should calculate the gradient of the cost function at that point. The gradient is the slope of the surface at every point and the direction of the gradient is the direction of the greatest uphill. Now, the question is, how do we calculate the gradient of a cost function at a point? If you select a random point on this surface, for example the yellow point, and take the partial derivative of J of theta with respect to each parameter at that point, it gives you the slope of the move for each parameter at that point. Now, if we move in the opposite direction of that slope, it guarantees that we go down in the error curve. For example, if we calculate the derivative of J with respect to theta one, we find out that it is a positive number. This indicates that function is increasing as theta one increases. So, to decrease J we should move in the opposite direction.This means to move in the direction of the negative derivative for theta one, i.e. slope. We have to calculate it for other parameters as well at each step. The gradient value also indicates how big of a step to take. If the slope is large we should take a large step because we are far from the minimum. If the slope is small we should take a smaller step. Gradient descent takes increasingly smaller steps towards the minimum with each iteration. The partial derivative of the cost function J is calculated using this expression. If you want to know how the derivative of the J function is calculated, you need to know the derivative concept which is beyond our scope here. But to be honest you don't really need to remember all the details about it as you can easily use this equation to calculate the gradients. So, in a nutshell this equation returns the slope of that point and we should update the parameter in the opposite direction of the slope. A vector of all these slopes is the gradient vector and we can use this vector to change or update all the parameters. We take the previous values of the parameters and subtract the error derivative. This results in the new parameters for theta that we know will decrease the cost. Also we multiply the gradient value by a constant value mu, which is called the learning rate. Learning rate, gives us additional control on how fast we move on the surface. In sum, we can simply say, gradient descent is like taking steps in the current direction of the slope, and the learning rate is like the length of the step you take. So, these would be our new parameters. Notice that it's an iterative operation and in each iteration we update the parameters and minimize the cost until the algorithm converge is on an acceptable minimum. Okay, let's recap what we have done to this point by going through the training algorithm again, step-by-step. Step one, we initialize the parameters with random values. Step two, we feed the cost function with the training set and calculate the cost. We expect a high error rate as the parameters are set randomly. Step three, we calculate the gradient of the cost function keeping in mind that we have to use a partial derivative. So, to calculate the gradient vector we need all the training data to feed the equation for each parameter. Of course, this is an expensive part of the algorithm, but there are some solutions for this. Step four, we update the weights with new parameter values. Step five, here we go back to step two and feed the cost function again, which has new parameters. As was explained earlier, we expect less error as we are going down the error surface. We continue this loop until we reach a short value of cost or some limited number of iterations. Step six, the parameter should be roughly found after some iterations. This means the model is ready and we can use it to predict the probability of a customer staying or leaving. Thanks for watching this video.

**Support Vector Machine**

Hello and welcome. In this video, we will learn a machine learning method called, Support Vector Machine, or SVM, which is used for classification. So let's get started. Imagine that you've obtained a dataset containing characteristics of thousands of human cell samples extracted from patients who were believedto be at risk of developing cancer. Analysis of the original data showed that many of the characteristics differed significantly between benign and malignant samples. You can use the values of these cell characteristics in samples from other patients, to give an early indication of whether a new sample might be benign or malignant. You can use Support Vector Machine, or SVM, as a classifier to train your model to understand patterns within the data that might show, benign or malignant cells. Once the model has been trained, it can be used to predict your new or unknown cell with rather high accuracy. Now, let me give you a formal definition of SVM. A Support Vector Machine is a supervised algorithm that can classify cases by finding a separator. SVM works by first mapping data to a high dimensional feature space so that data points can be categorized, even when the data are not otherwise linearly separable. Then, a separator is estimated for the data. The data should be transformed in such a way that a separator could be drawn as a hyperplane. For example, consider the following figure, which shows the distribution of a small set of cells only based on their unit size and clump thickness. As you can see, the data points fall into two different categories. It represents a linearly non separable data set. The two categories can be separated with a curve but not a line. That is, it represents a linearly non separable data set, which is the case for most real world data sets. We can transfer this data to a higher-dimensional space, for example, mapping it to a three-dimensional space. After the transformation, the boundary between the two categories can be defined by a hyperplane. As we are now in three-dimensional space, the separator is shown as a plane. This plane can be used to classify new or unknown cases. Therefore, the SVM algorithm outputs an optimal hyperplane that categorizes new examples. Now, there are two challenging questions to consider. First, how do we transfer data in such a way that a separator could be drawn as a hyperplane? And two, how can we find the best or optimized hyperplane separator after transformation? Let's first look at transforming data to see how it works. For the sake of simplicity, imagine that our dataset is one-dimensional data. This means we have only one feature x. As you can see, it is not linearly separable. So what can we do here? Well, we can transfer it into a two-dimensional space. For example, you can increase the dimension of data by mapping x into a new space using a function with outputs x and x squared. Now the data is linearly separable, right? Notice that as we are in a two-dimensional space, the hyperplane is a line dividing a plane into two parts where each class lays on either side. Now we can use this line to classify new cases. Basically, mapping data into a higher-dimensional space is called, kernelling. The mathematical function used for the transformation is known as the kernel function, and can be of different types, such as linear, polynomial, Radial Basis Function, or RBF, and sigmoid. Each of these functions has its own characteristics, its pros and cons, and its equation. But the good news is that you don't need to know them as most of them are already implemented in libraries of data science programming languages. Also, as there's no easy way of knowing which function performs best with any given dataset, we usually choose different functions in turn and compare the results. Now we get to another question. Specifically, how do we find the right or optimized separator after transformation? Basically, SVMs are based on the idea of finding a hyperplane that best divides a data set into two classes as shown here. As we're in a two-dimensional space, you can think of the hyperplane as a line that linearly separates the blue points from the red points. One reasonable choice as the best hyperplane is the one that represents the largest separation or margin between the two classes. So the goal is to choose a hyperplane with as big a margin as possible. Examples closest to the hyperplane are support vectors. It is intuitive that only support vectors matter for achieving our goal. And thus, other trending examples can be ignored. We tried to find the hyperplane in such a way that it has the maximum distance to support vectors. Please note that the hyperplane and boundary decision lines, have their own equations. So finding the optimized hyperplane can be formalized using an equation which involves quite a bit more math, so I'm not going to go through it here in detail. That said, the hyperplane is learned from training data using an optimization procedure that maximizes the margin. And like many other problems, this optimization problem can also be solved by gradient descent, which is out of scope of this video. Therefore, the output of the algorithm is the values w and b for the line. You can make classifications using this estimated line. It is enough to plug in input values into the line equation. Then, you can calculate whether an unknown point is above or below the line. If the equation returns a value greater than 0, then the point belongs to the first class which is above the line, and vice-versa. The two main advantages of support vector machines are that they're accurate in high-dimensional spaces. And they use a subset of training points in the decision function called, support vectors, so it's also memory efficient. The disadvantages of

Support Vector Machines include the fact that the algorithm is prone for over-fitting if the number of features is much greater than the number of samples. Also, SVMs do not directly

provide probability estimates, which are desirable in most classification problems. And finally, SVMs are not very efficient computationally if your dataset is very big, such as when you have more than 1,000 rows. And now our final question is, in which situation should I use SVM? Well, SVM is good for image analysis tasks, such as image classification and hand written digit recognition. Also, SVM is very effective in text mining tasks, particularly due to its effectiveness in dealing with high-dimensional data. For example, it is used for detecting spam, text category assignment and sentiment analysis. Another application of SVM is in gene expression data classification, again, because of its power in high-dimensional data classification. SVM can also be used for other types of machine learning problems, such as regression, outlier detection and clustering. I'll leave it to you to explore more about these particular problems. This concludes this video, thanks for watching.

**Intro to clustering**

Hello and welcome. In this video we'll give you a high level introduction to clustering, its applications, and different types of clustering algorithms. Let's get started! Imagine that you have a customer dataset and you need to apply customer segmentation on this historical data. Customer segmentation is the practice of partitioning a customer base into groups of individuals that have similar characteristics. It is a significant strategy, as it allows the business to target specific groups of customers, so as to more effectively allocate marketing resources. For example, one group might contain customers who are high profit and low risk. That is; more likely to purchase products or subscribe for a service. Knowing this information allows a business to devote more time and attention to retaining these customers. Another group might include customers from nonprofit organizations and so on. A general segmentation process is not usually feasible for large volumes of very data, therefore you need an analytical approach to deriving segments and groups from large datasets. Customers can be grouped based on several factors, including; age, gender, interests, spending habits and so on. The important requirement is to use the available data to understand and identify how customers are similar to each other. Let's learn how to divide a set of customers into categories, based on characteristics they share. One of the most adopted approaches that can be used for customer segmentation is clustering. Clustering can group data only unsupervised, based on the similarity of customers to each other. It will partition your customers into mutually exclusive groups. For example, into three clusters. The customers in each cluster are similar to each other demographically. Now we can create a profile for each group, considering the common characteristics of each cluster. For example, the first group made up of affluent and middle aged customers. The second is made up of young, educated and middle income customers, and the third group includes young and low income customers. Finally, we can assign each individual in our dataset to one of these groups or segments of customers. Now imagine that you cross join this segmented dataset with the dataset of the product or services that customers purchase from your company. This information would really help to understand and predict the differences and individual customers preferences and their buying behaviors across various products. Indeed, having this information would allow your company to develop highly personalized experiences for each segment. Customer segmentation is one of the popular usages of clustering. Cluster analysis also has many other applications in different domains. So let's first define clustering and then we'll look at other applications. Clustering means finding clusters in a dataset, unsupervised. So what is a cluster? A cluster is a group of data points or objects in a dataset that are similar to other objects in the group, and dissimilar to datapoints in other clusters. Now the question is," What is different between clustering and classification?" Let's look at our customer dataset again. Classification algorithms predict categorical classed labels. This means assigning instances to predefined classes such as defaulted or not defaulted. For example, if an analyst wants to analyze customer data in order to know which customers might default on their payments, she uses a labeled dataset as training data and uses classification approaches such as a decision tree, Support Vector Machines or SVM, or logistic regression, to predict the default value for a new or unknown customer. Generally speaking, classification is a supervised learning where each training data instance belongs to a particular class. In clustering however, the data is unlabeled and the process is unsupervised. For example, we can use a clustering algorithm such as k-means to group similar customers as mentioned, and assign them to a cluster, based on whether they share similar attributes, such as; age, education, and so on. While I'll be giving you some examples in different industries, I'd like you to think about more samples of clustering. In the retail industry, clustering is used to find associations among customers based on their demographic characteristics and use that information to identify buying patterns of various customer groups. Also, it can be used in recommendation systems to find a group of similar items or similar users and use it for collaborative filtering, to recommend things like books or movies to customers. In banking, analysts find clusters of normal transactions to find the patterns of fraudulent credit card usage. Also they use clustering to identify clusters of customers. For instance, to find loyal customers versus churned customers. In the insurance industry, clustering is used for fraud detection in claims analysis, or to evaluate the insurance risk of certain customers based on their segments. In publication media, clustering is used to auto categorize news based on his content or to tag news, then cluster it so as to recommend similar news articles to readers. In medicine, it can be used to characterize patient behavior, based on their similar characteristics. So as to identify successful medical therapies for different illnesses or in biology, clustering is used to group genes with similar expression patterns or to cluster genetic markers to identify family ties. If you look around you can find many other applications of clustering, but generally clustering can be used for one of the following purposes; exploratory data analysis, summary generation or reducing the scale, outlier detection- especially to be used for fraud detection or noise removal, finding duplicates and datasets or as a pre-processing step for either prediction, other data mining tasks or as part of a complex system. Let's briefly look at different clustering algorithms and their characteristics. Partitioned-based clustering is a group of clustering algorithms that produces fear like clusters, such as; K-Means, K-Medians or Fuzzy c-Means. These algorithms are relatively efficient and are used for medium and large sized databases. Hierarchical clustering algorithms produce trees of clusters, such as agglomerative and divisive algorithms. This group of algorithms are very intuitive and are generally good for use with small size datasets. Density-based clustering algorithms produce arbitrary shaped clusters. They are especially good when dealing with spatial clusters or when there is noise in your data set. For example, the DB scan algorithm. This concludes our video. Thanks for watching!

**K-Means Clustering**

Hello and welcome. In this video, we'll be covering K-Means Clustering. So let's get started. Imagine that you have a customer dataset and you need to apply customer segmentation on this historical data. Customer segmentation is the practice of partitioning a customer base into groups of individuals that have similar characteristics. One of the algorithms that can be used for customer segmentation is K-Means clustering. K-Means can group data only unsupervised based on the similarity of customers to each other. Let's define this technique more formally. There are various types of clustering algorithms such as partitioning, hierarchical or density-based clustering. K-Means is a type of partitioning clustering, that is, it divides the data into K non-overlapping subsets or clusters without any cluster internal structure or labels. This means, it's an unsupervised algorithm. Objects within a cluster are very similar, and objects across different clusters are very different or dissimilar. As you can see, for using K-Means we have to find similar samples. For example, similar customers. Now, we face a couple of key questions. First, how can we find the similarity of samples in clustering? Then how do we measure how similar two customers are with regard to their demographics? Though the objective of K-Means is to form clusters in such a way that similar samples go into a cluster, and dissimilar samples fall into different clusters, it can be shown that instead of a similarity metric, we can use dissimilarity metrics. In other words, conventionally the distance of samples from each other is used to shape the clusters. So we can say K-Means tries to minimize the intra-cluster distances and maximize the inter-cluster distances. Now, the question is, how can we calculate the dissimilarity or distance of two cases such as two customers? Assume that we have two customers, we will call them Customer one and two. Let's also assume that we have only one feature for each of these two customers and that feature is age. We can easily use a specific type of Minkowski distance to calculate the distance of these two customers. Indeed, it is the Euclidean distance. Distance of x1 from x2 is root of 34 minus 30\_2 which is four. What about if we have more than one feature, for example age and income. For example, if we have income and age for each customer, we can still use the same formula but this time in a two-dimensional space. Also, we can use the same distance matrix for multidimensional vectors. Of course, we have to normalize our feature set to get the accurate dissimilarity measure. There are other dissimilarity measures as well that can be used for this purpose, but it is highly dependent on datatype and also the domain that clustering is done for it. For example you may use Euclidean distance, Cosine similarity, Average distance, and so on. Indeed, the similarity measure highly controls how the clusters are formed, so it is recommended to understand the domain knowledge of your dataset and datatype of features and then choose the meaningful distance measurement. Now, let's see how K-Means clustering works. For the sake of simplicity, let's assume that our dataset has only two features; the age and income of customers. This means, it's a two-dimensional space. We can show the distribution of customers using a scatter plot: the Y-axis indicates age and the X-axis shows income of customers. We try to cluster the customer dataset into distinct groups or clusters based on these two dimensions. In the first step, we should determine the number of clusters. The key concept of the K-Means algorithm is that it randomly picks a center point for each cluster. It means we must initialize K which represents number of clusters. Essentially, determining the number of clusters in a dataset or K is a hard problem in K-Means, that we will discuss later. For now, let's put K equals three here for our sample dataset. It is like we have three representative points for our clusters. These three data points are called centroids of clusters and should be of same feature size of our customer feature set. There are two approaches to choose these centroids. One, we can randomly choose three observations out of the dataset and use these observations as the initial means. Or two, we can create three random points as centroids of the clusters which is our choice that is shown in the plot with red color. After the initialization step which was defining the centroid of each cluster, we have to assign each customer to the closest center. For this purpose, we have to calculate the distance of each data point or in our case each customer from the centroid points. As mentioned before, depending on the nature of the data and the purpose for which clustering is being used different measures of distance may be used to place items into clusters. Therefore, you will form a matrix where each row represents the distance of a customer from each centroid. It is called the Distance Matrix. The main objective of K-Means clustering is to minimize the distance of data points from the centroid of this cluster and maximize the distance from other cluster centroids. So in this step, we have to find the closest centroid to each data point. We can use the distance matrix to find the nearest centroid to datapoints. Finding the closest centroids for each data point, we assign each data point to that cluster. In other words, all the customers will fall to a cluster based on their distance from centroids. We can easily say that it does not result in good clusters because the centroids were chosen randomly from the first. Indeed, the model would have a high error. Here, error is the total distance of each point from its centroid. It can be shown as within-cluster sum of squares error. Intuitively, we try to reduce this error. It means we should shape clusters in such a way that the total distance of all members of a cluster from its centroid be minimized. Now, the question is, how can we turn it into better clusters with less error? Okay, we move centroids. In the next step, each cluster center will be updated to be the mean for datapoints in its cluster. Indeed, each centroid moves according to their cluster members. In other words the centroid of each of the three clusters becomes the new mean. For example, if point A co-ordination is 7.4 and 3.6, and B point features are 7.8 and 3.8, the new centroid of this cluster with two points would be the average of them. Which is 7.6 and 3.7. Now, we have new centroids. As you can guess, once again we will have to calculate the distance of all points from the new centroids. The points are reclustered and the centroids move again. This continues until the centroids no longer move. Please note that whenever a centroid moves, each points distance to the centroid needs to be measured again. Yes, K-Means is an iterative algorithm and we have to repeat steps two to four until the algorithm converges. In each iteration, it will move the centroids, calculate the distances from new centroids and assign data points to the nearest centroid. It results in the clusters with minimum error or the most dense clusters. However, as it is a heuristic algorithm, there is no guarantee that it will converge to the global optimum and the result may depend on the initial clusters. It means, this algorithm is guaranteed to converge to a result but the result may be a local optimum i.e. not necessarily the best possible outcome. To solve this problem, it is common to run the whole process multiple times with different starting conditions. This means with randomized starting centroids, it may give a better outcome. As the algorithm is usually very fast, it wouldn't be any problem to run it multiple times. Thanks for watching this video.

Hello and welcome. In this video, we'll look at k-Means accuracy and characteristics. Let's get started. Let's define the algorithm more concretely, before we talk about its accuracy. A k-Means algorithm works by randomly placing k centroids, one for each cluster. The farther apart the clusters are placed, the better. The next step is to calculate the distance of each data point or object from the centroids. Euclidean distance is used to measure the distance from the object to the centroid. Please note, however, that you can also use different types of distance measurements, not just Euclidean distance. Euclidean distance is used because it's the most popular. Then, assign each data point or object to its closest centroid creating a group. Next, once each data point has been classified to a group, recalculate the position of the k centroids. The new centroid position is determined by the mean of all points in the group. Finally, this continues until the centroids no longer move. Now, the questions is, how can we evaluate the goodness of the clusters formed by k-Means? In other words, how do we calculate the accuracy of k-Means clustering? One way is to compare the clusters with the ground truth, if it's available. However, because k-Means is an unsupervised algorithm we usually don't have ground truth in real world problems to be used. But there is still a way to say how bad each cluster is, based on the objective of the k-Means. This value is the average distance between data points within a cluster. Also, average of the distances of data points from their cluster centroids can be used as a metric of error for the clustering algorithm. Essentially, determining the number of clusters in a data set, or k as in the k-Means algorithm, is a frequent problem in data clustering. The correct choice of K is often ambiguous because it's very dependent on the shape and scale of the distribution of points in a dataset. There are some approaches to address this problem, but one of the techniques that is commonly used is to run the clustering across the different values of K and looking at a metric of accuracy for clustering. This metric can be mean, distance between data points and their cluster's centroid. Which indicate how dense our clusters are or, to what extent we minimize the error of clustering. Then, looking at the change of this metric, we can find the best value for K. But the problem is that with increasing the number of clusters, the distance of centroids to data points will always reduce. This means increasing K will always decrease the error. So, the value of the metric as a function of K is plotted and the elbow point is determined where the rate of decrease sharply shifts. It is the right K for clustering. This method is called the elbow method. So let's recap k-Means clustering. k-Means is a partition-based clustering which is A, relatively efficient on medium and large sized data sets. B, produces sphere-like clusters because the clusters are shaped around the centroids. And C, its drawback is that we should pre-specify the number of clusters, and this is not an easy task. Thanks for watching.

**Hierarchical Clustering**

Hello and welcome. In this video, we'll be covering Hierarchical Clustering. So, let's get started. Let's look at this chart. An international team of scientists led by UCLA biologists used this dendrogram to report genetic data from more than 900 dogs from 85 breeds, and more than 200 wild gray wolves worldwide, including populations from North America, Europe, the Middle East and East Asia. They used molecular genetic techniques to analyze more than 48,000 genetic markers. This diagram shows hierarchical clustering of these animals based on the similarity in their genetic data. Hierarchical clustering algorithms build a hierarchy of clusters where each node is a cluster consisting of the clusters of its daughter nodes. Strategies for hierarchical clustering generally fall into two types, divisive and agglomerative. Divisive is top down, so you start with all observations in a large cluster and break it down into smaller pieces Think about divisive as dividing the cluster. Agglomerative is the opposite of divisive. So it is bottom up, where each observation starts in its own cluster and pairs of clusters are merged together as they move up the hierarchy. Agglomeration means to amass or collect things, which is exactly what this does with the cluster. The agglomerative approach is more popular among data scientists and so it is the main subject of this video. Let's look at a sample of agglomerative clustering. This method builds the hierarchy from the individual elements by progressively merging clusters. In our example, let's say we want to cluster six cities in Canada based on their distances from one another. They are: Toronto, Ottawa, Vancouver, Montreal, Winnipeg and Edmonton. We construct a distance matrix at this stage, where the numbers in the row i column j is the distance between the i and j cities. In fact, this table shows the distances between each pair of cities. The algorithm is started by assigning each city to its own cluster. So if we have six cities, we have six clusters each containing just one city. Let's note each city by showing the first two characters of its name. The first step is to determine which cities, let's call them clusters from now on, to merge into a cluster. Usually, we want to take the two closest clusters according to the chosen distance. Looking at the distance matrix, Montreal and Ottawa are the closest clusters so we make a cluster out of them. Please notice that we just use a simple one-dimensional distance feature here, but our object can be multidimensional and distance measurement can either be Euclidean, Pearson, average distance or many others depending on data type and domain knowledge. Anyhow, we have to merge these two closest cities in the distance matrix as well. So, rows and columns are merged as the cluster is constructed. As you can see in the distance matrix, rows and columns related to Montreal and Ottawa cities are merged as the cluster is constructed. Then, the distances from all cities to this new merged cluster get updated. But how? For example, how do we calculate the distance from Winnipeg to the Ottawa/Montreal cluster? Well, there are different approaches but let's assume for example, we just select the distance from the center of the Ottawa/Montreal cluster to Winnipeg. Updating the distance matrix, we now have one less cluster. Next, we look for the closest clusters once again. In this case, Ottawa, Montreal and Toronto are the closest ones which creates another cluster. In the next step, the closest distances between the Vancouver cluster and the Edmonton cluster. Forming a new cluster, the data in the matrix table gets updated. Essentially, the rows and columns are merged as the clusters are merged and the distance updated. This is a common way to implement this type of clustering and has the benefit of caching distances between clusters. In the same way, agglomerative algorithm proceeds by merging clusters, and we repeat it until all clusters are merged and the tree becomes completed. It means, until all cities are clustered into a single cluster of size six. Hierarchical clustering is typically visualized as a dendrogram as shown on this slide. Each merge is represented by a horizontal line. The y-coordinate of the horizontal line is the similarity of the two clusters that were merged where cities are viewed as singleton clusters. By moving up from the bottom layer to the top node, a dendrogram allows us to reconstruct the history of merges that resulted in the depicted clustering. Essentially, hierarchical clustering does not require a prespecified number of clusters. However, in some applications, we want a partition of disjoint clusters just as in flat clustering. In those cases, the hierarchy needs to be cut at some point. For example here, cutting in a specific level of similarity, we create three clusters of similar cities. This concludes this video. Thanks for watching.

**more details about hierarchical clustering**

Hello and welcome. In this video, we'll be covering more details about hierarchical clustering. Let's get started. Let's look at agglomerative algorithm for hierarchical clustering. Remember that agglomerative clustering is a bottom up approach. Let's say our data set has n data points. First, we want to create n clusters, one for each data point. Then, each point is assigned as a cluster. Next, we want to compute the distance proximity matrix which will be an n by n table. After that, we want to iteratively run the following steps until the specified cluster number is reached, or until there is only one cluster left. First, merge the two nearest clusters. Distances are computed already in the proximity matrix. Second, update the proximity matrix with the new values. We stop after we've reached the specified number of clusters, or there is only one cluster remaining with the result stored in a dendogram. So in the proximity matrix, we have to measure the distances between clusters and also merge the clusters that are nearest. So, the key operation is the computation of the proximity between the clusters with one point and also clusters with multiple data points. At this point, there are a number of key questions that need to be answered. For instance, how do we measure the distances between these clusters and how do we define the nearest among clusters? We also can ask, which points do we use? First, let's see how to calculate the distance between two clusters with one point each. Let's assume that we have a data set of patients and we want to cluster them using hierarchy clustering. So our data points are patients with a featured set of three dimensions. For example, age, body mass index, or BMI and blood pressure. We can use different distance measurements to calculate the proximity matrix. For instance, Euclidean distance. So, if we have a data set of n patience, we can billed an n by n dissimilarity distance matrix. It will give us the distance of clusters with one data point. However, as mentioned, we merge clusters in agglomerative clustering. Now the question is, how can we calculate the distance between clusters when there are multiple patients in each cluster? We can use different criteria to find the closest clusters and merge them. In general, it completely depends on the data type, dimensionality of data and most importantly, the domain knowledge of the data set. In fact, different approaches to defining the distance between clusters distinguish the different algorithms. As you might imagine, there are multiple ways we can do this. The first one is called single linkage clustering. Single linkage is defined as the shortest distance between two points in each cluster, such as point a and b. Next up is complete linkage clustering. This time we are finding the longest distance between the points in each cluster, such as the distance between point a and b. The third type of linkage is average linkage clustering or the mean distance. This means we're looking at the average distance of each point from one cluster to every point in another cluster. The final linkage type to be reviewed is centroid linkage clustering. Centroid is the average of the feature sets of points in a cluster. This linkage takes into account the centroid of each cluster when determining the minimum distance. There are three main advantages to using hierarchical clustering. First, we do not need to specify the number of clusters required for the algorithm. Second, hierarchical clustering is easy to implement. And third, the dendrogram produced is very useful in understanding the data. There are some disadvantages as well. First, the algorithm can never undo any previous steps. So for example, the algorithm clusters two points and later on, we see that the connection was not a good one. The program can not undo that step. Second, the time complexity for the clustering can result in very long computation times in comparison with efficient algorithms such as K-means. Finally, if we have a large data set, it can become difficult to determine the correct number of clusters by the dendrogram. Now, lets compare hierarchical clustering with K-means. K-means is more efficient for large data sets. In contrast to K-means, hierarchical clustering does not require the number of cluster to be specified. Hierarchical clustering gives more than one partitioning depending on the resolution or as K-means gives only one partitioning of the data. Hierarchical clustering always generates the same clusters, in contrast with K-means, that returns different clusters each time it is run, due to random initialization of centroids. Thanks for watching.

**DB scan**

Hello and welcome. In this video, we'll be covering DB scan. A density-based clustering algorithm which is appropriate to use when examining spatial data. So let's get started. Most of the traditional clustering techniques such as K-Means, hierarchical, and Fuzzy clustering can be used to group data in an unsupervised way. However, when applied to tasks with arbitrary shaped clusters or clusters within clusters, traditional techniques might not be able to achieve good results that is, elements in the same cluster might not share enough similarity or the performance may be poor. Additionally, while partitioning based algorithms such as K-Means may be easy to understand and implement in practice, the algorithm has no notion of outliers that is, all points are assigned to a cluster even if they do not belong in any. In the domain of anomaly detection, this causes problems as anomalous points will be assigned to the same cluster as normal data points. The anomalous points pull the cluster centroid towards them making it harder to classify them as anomalous points. In contrast, density-based clustering locates regions of high density that are separated from one another by regions of low density. Density in this context is defined as the number of points within a specified radius. A specific and very popular type of density-based clustering is DBSCAN. DBSCAN is particularly effective for tasks like class identification on a spatial context. The wonderful attributes of the DBSCAN algorithm is that it can find out any arbitrary shaped cluster without getting effected by noise. For example, this map shows the location of weather stations in Canada. DBSCAN can be used here to find the group of stations which show the same weather condition. As you can see, it not only finds different arbitrary shaped clusters it can find the denser part of data-centered samples by ignoring less dense areas or noises. Now, let's look at this clustering algorithm to see how it works. DBSCAN stands for Density-Based Spatial Clustering of Applications with Noise. This technique is one of the most common clustering algorithms which works based on density of object. DBSCAN works on the idea that if a particular point belongs to a cluster it should be near to lots of other points in that cluster. It works based on two parameters; radius and minimum points. R determines a specified radius that if it includes enough points within it, we call it a dense area. M determines the minimum number of data points we want in a neighborhood to define a cluster. Let's define radius as two units. For the sake of simplicity, assume it has radius of two centimeters around a point of interest. Also, let's set the minimum point or M to be six points including the point of interest. To see how DBSCAN works, we have to determine the type of points. Each point in our dataset can be either a core, border, or outlier point. Don't worry, I'll explain what these points are in a moment. But the whole idea behind the DBSCAN algorithm is to visit each point and find its type first, then we group points as clusters based on their types. Let's pick a point randomly. First, we check to see whether it's a core data point. So, what is a core point? A data point is a core point if within our neighborhood of the point there are at least M points. For example, as there are six points in the two centimeter neighbor of the red point, we mark this point as a core point. Okay, what happens if it's not a core point? Let's look at another point. Is this point a core point? No. As you can see, there are only five points in this neighborhood including the yellow point, so what kind of point is this one? In fact, it is a border point. What is a border point? A data point is a border point if A; its neighbourhood contains less than M data points or B; it is reachable from some core point. Here, reachability means it is within our distance from a core point. It means that even though the yellow point is within the two centimeter neighborhood of the red point, it is not by itself a core point because it does not have at least six points in its neighborhood. We continue with the next point. As you can see, it is also a core point and all points around it which are not core points are border points. Next core point and next core point. Let's pick this point. You can see it is not a core point nor is it a border point. So, we'd label it as an outlier. What is an outlier? An outlier is a point that is not a core point and also is not close enough to be reachable from a core point. We continue and visit all the points in the dataset and label them as either core, border, or outlier. The next step is to connect core points that are neighbors and put them in the same cluster. So, a cluster is formed as at least one core point plus all reachable core points plus all their borders. It's simply shapes all the clusters and finds outliers as well. Let's review this one more time to see why DBSCAN is cool. DBSCAN can find arbitrarily shaped clusters. It can even find a cluster completely surrounded by a different cluster. DBSCAN has a notion of noise and is robust to outliers. On top of that, DBSCAN makes it very practical for use in many real-world problems because it does not require one to specify the number of clusters such as K in K-means. This concludes this video. Thanks for watching.

**recommendation systems**

Hello, and welcome! In this video, we’ll be going through a quick introduction to recommendation systems. So, let’s get started. Even though peoples’ tastes may vary, they generally follow patterns. By that, I mean that there are similarities in the things that people tend to like … or another way to look at it, is that people tend to like things in the same category or things that share the same characteristics. For example, if you’ve recently purchased a book on Machine Learning in Python and you’ve enjoyed reading it, it’s very likely that you’ll also enjoy reading a book on Data Visualization. People also tend to have similar tastes to those of the people they’re close to in their lives. Recommender systems try to capture these patterns and similar behaviors, to help predict what else you might like. Recommender systems have many applications that I’m sure you’re already familiar with. Indeed, Recommender systems are usually at play on many websites. For example, suggesting books on Amazon and movies on Netflix. In fact, everything on Netflix’s website is driven by customer selection. If a certain movie gets viewed frequently enough, Netflix’s recommender system ensures that that movie gets an increasing number of recommendations. Another example can be found in a daily-use mobile app, where a recommender engine is used to recommend anything from where to eat, or, what job to apply to. On social media, sites like Facebook or LinkedIn, regularly recommend friendships. Recommender systems are even used to personalize your experience on the web. For example, when you go to a news platform website, a recommender system will make note of the types of stories that you clicked on and make recommendations on which types of stories you might be interested in reading, in future. There are many of these types of examples and they are growing in number every day. So, let’s take a closer look at the main benefits of using a recommendation system. One of the main advantages of using recommendation systems is that users get a broader exposure to many different products they might be interested in. This exposure encourages users towards continual usage or purchase of their product. Not only does this provide a better experience for the user but it benefits the service provider, as well, with increased potential revenue and better security for its customers. There are generally 2 main types of recommendation systems: Content-based and collaborative filtering. The main difference between each, can be summed up by the type of statement that a consumer might make. For instance, the main paradigm of a Content-based recommendation system is driven by the statement: “Show me more of the same of what I've liked before." Content-based systems try to figure out what a user's favorite aspects of an item are, and then make recommendations on items that share those aspects. Collaborative filtering is based on a user saying, “Tell me what's popular among my neighbors because I might like it too.” Collaborative filtering techniques find similar groups of users, and provide recommendations based on similar tastes within that group. In short, it assumes that a user might be interested in what similar users are interested in. Also, there are Hybrid recommender systems, which combine various mechanisms. In terms of implementing recommender systems, there are 2 types: Memory-based and Model-based. In memory-based approaches, we use the entire user-item dataset to generate a recommendation system. It uses statistical techniques to approximate users or items. Examples of these techniques include: Pearson Correlation, Cosine Similarity and Euclidean Distance, among others. In model-based approaches, a model of users is developed in an attempt to learn their preferences. Models can be created using Machine Learning techniques like regression, clustering, classification, and so on. This is the end of our video. Thanks for watching!

**Content-Based Recommender Systems**

Hello, and welcome. In this video, we'll be covering Content-Based Recommender Systems. So let's get started. A Content-based recommendation system tries to recommend items to users based on their profile. The user's profile revolves around that user's preferences and tastes. It is shaped based on user ratings, including the number of times that user has clicked on different items or perhaps even liked those items. The recommendation process is based on the similarity between those items. Similarity or closeness of items is measured based on the similarity in the content of those items. When we say content, we're talking about things like the items category, tag, genre, and so on. For example, if we have four movies, and if the user likes or rates the first two items, and if Item 3 is similar to Item 1 in terms of their genre, the engine will also recommend Item 3 to the user. In essence, this is what content-based recommender system engines do. Now, let's dive into a content-based recommender system to see how it works. Let's assume we have a data set of only six movies. This data set shows movies that our user has watched and also the genre of each of the movies. For example, Batman versus Superman is in the Adventure, Super Hero genre and Guardians of the Galaxy is in the Comedy, Adventure, Super Hero and Science-fiction genres. Let's say the user has watched and rated three movies so far and she has given a rating of two out of 10 to the first movie, 10 out of 10 to the second movie and eight out of 10 to the third. The task of the recommender engine is to recommend one of the three candidate movies to this user, or in other, words we want to predict what the user's possible rating would be of the three candidate movies if she were to watch them. To achieve this, we have to build the user profile. First, we create a vector to show the user's ratings for the movies that she's already watched. We call it Input User Ratings. Then, we encode the movies through the one-hot encoding approach. Genre of movies are used here as a feature set. We use the first three movies to make this matrix, which represents the movie feature set matrix. If we multiply these two matrices we can get the weighted feature set for the movies. Let's take a look at the result. This matrix is also called the Weighted Genre matrix and represents the interests of the user for each genre based on the movies that she's watched. Now, given the Weighted Genre Matrix, we can shape the profile of our active user. Essentially, we can aggregate the weighted genres and then normalize them to find the user profile. It clearly indicates that she likes superhero movies more than other genres. We use this profile to figure out what movie is proper to recommend to this user. Recall that we also had three candidate movies for recommendation that haven't been watched by the user, we encode these movies as well. Now we're in the position where we have to figure out which of them is most suited to be recommended to the user. To do this, we simply multiply the User Profile matrix by the candidate Movie Matrix, which results in the Weighted Movies Matrix. It shows the weight of each genre with respect to the User Profile. Now, if we aggregate these weighted ratings, we get the active user's possible interest level in these three movies. In essence, it's our recommendation lists, which we can sort to rank the movies and recommend them to the user. For example, we can say that the Hitchhiker's Guide to the Galaxy has the highest score in our list and it's proper to recommend to the user. Now, you can come back and fill the predicted ratings for the user. So, to recap what we've discussed so far, the recommendation in a content-based system is based on user's taste and the content or feature set items. Such a model is very efficient. However, in some cases, it doesn't work. For example, assume that we have a movie in the drama genre, which the user has never watch. So, this genre would not be in her profile. Therefore, shall only get recommendations related to genres that are already in her profile and the recommender engine may never recommend any movie within other genres. This problem can be solved by other types of recommender systems such as collaborative filtering. Thanks for watching.

**collaborative filtering**

Hello, and welcome. In this video, we'll be covering a recommender system technique called collaborative filtering. So let's get started. Collaborative filtering is based on the fact that relationships exist between products and people's interests. Many recommendation systems use collaborative filtering to find these relationships and to give an accurate recommendation of a product that the user might like or be interested in. Collaborative filtering has basically two approaches; user-based, and item-based. User-based collaborative filtering is based on the user similarity or neighborhood. Item-based collaborative filtering is based on similarity among items. Let's first look at the intuition behind the user-based approach. In user-based collaborative filtering, we have an active user for whom the recommendation is aimed. The collaborative filtering engine first looks for users who are similar. That is users who share the active users rating patterns. Collaborative filtering basis this similarity on things like history, preference, and choices that users make when buying, watching, or enjoying something. For example, movies that similar users have rated highly. Then it uses the ratings from these similar users to predict the possible ratings by the active user for a movie that she had not previously watched. For instance, if two users are similar or are neighbors in terms of their interested movies, we can recommend a movie to the active user that her neighbor has already seen. Now, let's dive into the algorithm to see how all of this works. Assume that we have a simple user item matrix, which shows the ratings of four users for five different movies. Let's also assume that our active user has watched and rated three out of these five movies. Let's find out which of the two movies that our active user hasn't watched should be recommended to her. The first step is to discover how similar the active user is to the other users. How do we do this? Well, this can be done through several different statistical and vectorial techniques such as distance or similarity measurements including Euclidean Distance, Pearson Correlation, Cosine Similarity, and so on. To calculate the level of similarity between two users, we use the three movies that both the users have rated in the past. Regardless of what we use for similarity measurement, let's say for example, the similarity could be 0.7, 0.9, and 0.4 between the active user and other users. These numbers represent similarity weights or proximity of the active user to other users in the dataset. The next step is to create a weighted rating matrix. We just calculated the similarity of users to our active user in the previous slide. Now, we can use it to calculate the possible opinion of the active user about our two target movies. This is achieved by multiplying the similarity weights to the user ratings. It results in a weighted ratings matrix, which represents the user's neighbors opinion about are two candidate movies for recommendation. In fact, it incorporates the behavior of other users and gives more weight to the ratings of those users who are more similar to the active user. Now, we can generate the recommendation matrix by aggregating all of the weighted rates. However, as three users rated the first potential movie and two users rated the second movie, we have to normalize the weighted rating values. We do this by dividing it by the sum of the similarity index for users. The result is the potential rating that our active user will give to these movies based on her similarity to other users. It is obvious that we can use it to rank the movies for providing recommendation to our active user. Now, let's examine what's different between user-based and item-based collaborative filtering. In the user-based approach, the recommendation is based on users of the same neighborhood with whom he or she shares common preferences. For example, as User 1 and User 3 both liked Item 3 and Item 4, we consider them as similar or neighbor users, and recommend Item 1 which is positively rated by User 1 to User 3. In the item-based approach, similar items build neighborhoods on the behavior of users. Please note however, that it is not based on their contents. For example, Item 1 and Item 3 are considered neighbors as they were positively rated by both User 1 and User 2. So, Item 1 can be recommended to User 3 as he has already shown interest in Item 3. Therefore, the recommendations here are based on the items in the neighborhood that a user might prefer. Collaborative filtering is a very effective recommendation system. However, there are some challenges with it as well. One of them is data sparsity. Data sparsity happens when you have a large data set of users who generally rate only a limited number of items. As mentioned, collaborative based recommenders can only predict scoring of an item if there are other users who have rated it. Due to sparsity, we might not have enough ratings in the user item dataset which makes it impossible to provide proper recommendations. Another issue to keep in mind is something called cold start. Cold start refers to the difficulty the recommendation system has when there is a new user, and as such a profile doesn't exist for them yet. Cold start can also happen when we have a new item which has not received a rating. Scalability can become an issue as well. As the number of users or items increases and the amount of data expands, collaborative filtering algorithms will begin to suffer drops in performance, simply due to growth and the similarity computation. There are some solutions for each of these challenges such as using hybrid based recommender systems, but they are out of scope of this course. Thanks for watching.