**Intro to Logistic Regression**

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 independent 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 in 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 are 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 plottable. 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, and 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.

**Logistic regression vs Linear regression**

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 sample 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 p1 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, and 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, and 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, and 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.

**Logistic Regression Training**

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 is 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.

**Support Vector Machine**

Imagine that you've obtained a dataset containing characteristics of thousands of human cell samples extracted from patients who were believed to 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.