**Introduction to Classification**

In machine learning classification is a supervised learning approach which can be thought of as a means of categorizing or classifying some unknown items into a discrete set of classes. Classification attempts to learn the relationship between a set of feature variables and a target variable of interest. The target attribute in classification is a categorical variable with discrete values. So, how does classification and classifiers work? Given a set of training data points along with the target labels, classification determines the class label for an unlabeled test case. Let's explain this with an example. A good sample of classification is the loan default prediction. Suppose a bank is concerned about the potential for loans not to be repaid? If previous loan default data can be used to predict which customers are likely to have problems repaying loans, these bad risk customers can either have their loan application declined or offered alternative products. The goal of a loan default predictor is to use existing loan default data which has information about the customers such as age, income, education et cetera, to build a classifier, pass a new customer or potential future default to the model, and then label it, i.e the data points as defaulter or not defaulter. Or for example zero or one. This is how a classifier predicts an unlabeled test case. Please notice that this specific example was about a binary classifier with two values. We can also build classifier models for both binary classification and multi-class classification. For example, imagine that you've collected data about a set of patients, all of whom suffered from the same illness. During their course of treatment, each patient responded to one of three medications. You can use this labeled dataset with a classification algorithm to build a classification model. Then you can use it to find out which drug might be appropriate for a future patient with the same illness. As you can see, it is a sample of multi-class classification. Classification has different business use cases as well. For example, to predict the category to which a customer belongs, for churn detection where we predict whether a customer switches to another provider or brand, or to predict whether or not a customer responds to a particular advertising campaign. Data classification has several applications in a wide variety of industries. Essentially, many problems can be expressed as associations between feature and target variables, especially when labelled data is available. This provides a broad range of applicability for classification. For example, classification can be used for email filtering, speech recognition, handwriting recognition, biometric identification, document classification and much more. Here we have the types of classification algorithms and machine learning. They include decision trees, naive bayes, linear discriminant analysis, k-nearest neighbor, logistic regression, neural networks, and support vector machines. There are many types of classification algorithms. We will only cover a few in this course.

**K-Nearest Neighbours**

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.

**Evaluation Metrics in Classification**

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. let's 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 negatives, true negatives, and false positives. 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 ideal classifiers have progressively smaller values of log loss. So, the classifier with lower log loss has better accuracy.

**Introduction to Decision Trees**

What exactly is a decision tree? How do we use them to help us classify? How can I grow my own decision tree? These may be some of the questions that you have in mind from hearing the term decision tree. Hopefully, you'll soon be able to answer these questions and many more by watching this video. Imagine that you're a medical researcher compiling data for a study. You've already collected data about a set of patients all of whom suffered from the same illness. During their course of treatment, each patient responded to one of two medications. We call them drug A and drug B. Part of your job is to build a model to find out which drug might be appropriate for a future patient with the same illness. The feature sets of this dataset are age, gender, blood pressure, and cholesterol of our group of patients and the target is the drug that each patient responded to. It is a sample of binary classifiers, and you can use the training part of the data set to build a decision tree and then use it to predict the class of an unknown patient. In essence, to come up with a decision on which drug to prescribe to a new patient. Let's see how a decision tree is built for this dataset. Decision trees are built by splitting the training set into distinct nodes, where one node contains all of or most of one category of the data. If we look at the diagram here, we can see that it's a patient's classifier. So as mentioned, we want to prescribe a drug to a new patient, but the decision to choose drug A or B will be influenced by the patient's situation. We start with age, which can be young, middle aged or senior. If the patient is middle aged, then we'll definitely go for drug B. On the other hand, if he has a young or a senior patient, will need more details to help us determine which drug to prescribe. The additional decision variables can be things such as cholesterol levels, gender or blood pressure. For example, if the patient is female, then we will recommend drug A, but if the patient is male, then will go for drug B. As you can see, decision trees are about testing an attribute and branching the cases based on the result of the test. Each internal node corresponds to a test, and each branch corresponds to a result of the test, and each leaf node assigns a patient to a class. Now the question is, how can we build such a decision tree? Here is the way that a decision tree is built. A decision tree can be constructed by considering the attributes one by one. First, choose an attribute from our dataset. Calculate the significance of the attribute in the splitting of the data. In the next video, we will explain how to calculate the significance of an attribute to see if it's an effective attribute or not. Next, split the data based on the value of the best attribute, then go to each branch and repeat it for the rest of the attributes. After building this tree, you can use it to predict the class of unknown cases; or in our case, the proper drug for a new patient based on his or her characteristics.

**Building Decision Trees**

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 training 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 are drug A and other half are 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.