Machine Learning with Python

Luis Carlos Rojas Torres

September 2021

Chapter 1

Week 1

1.1 What is Machine Learning?

1.1.1 Welcome

Hello and welcome to Machine Learning with Python. In this course you'll learn how Machine Learning is used in many key fields and industries. For example, in the healthcare industry, data scientists use Machine Learning to predict whether a human cell that is believed to be at risk of developing cancer is either benign or malignant. As such, Machine Learning can play a key role in determining a person's health and welfare. You'll also learn about the value of decision trees and how building a good decision tree from historical data helps doctors to prescribe the proper medicine for each of their patients.You'll learn how bankers use Machine Learning to make decisions on whether to approve loan applications; and you will learn how to use Machine Learning to do bank customer segmentation, where it is not usually easy to run for huge volumes of varied data. In this course you'll see how machine learning helps websites such as YouTube, Amazon, or Netflix develop recommendations to their customers about various products or services, such as which movies they might be interested in going to see or which books to buy. There is so much that you can do with Machine Learning. Here you'll learn how to use popular Python libraries to build your model. For example, given an automobile data set, we can use the scikit-learn library to estimate the co2 emission of cars using their engine size or cylinders. We can even predict what the co2 emissions will be for a car that hasn't even been produced yet, and we'll see how the telecommunications industries can predict customer churn.You can run and practice the code of all these samples using the built-in lab environment in this course, you don't have to install anything to your computer or do anything on the cloud. All you have to do is click a button to start the lab environment in your browser. The code for the samples is already written using Python language in Jupyter notebooks and you can run it to see the results or change it to understand the algorithms better. So what will you be able to achieve by taking this course? Well, by putting in just a few hours a week over the next few weeks you'll get new skills to add to your resume such as regression, classification, clustering, scikit-learn, and psy PI. You'll also get new projects that you can add to your portfolio including cancer detection, predicting economic trends, predicting customer churn, recommendation engines, and many more. You'll also get a certificate in Machine Learning to prove your competency and share it anywhere you like online or offline such as LinkedIn profiles and social media, so let's get started.

1.1.2 Introduction to Machine Learning

Hello, and welcome! In this video I will give you a high level introduction to Machine Learning. So let's get started. This is a human cell sample extracted from a patient, and this cell has characteristics. For example, its clump thickness is 6, its uniformity of cell size is 1, its marginal adhesion is 1, and so on. One of the interesting questions we can ask, at this point is: Is this a benign or malignant cell? In contrast with a benign tumor, a malignant tumor is a tumor that may invade its surrounding tissue or spread around the body, and diagnosing it early might be the key to a patient's survival. One could easily presume that only a doctor with years of experience could diagnose that tumor and say if the patient is developing cancer or not. Right? Well, 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 should clean your data, select a proper algorithm for building a prediction model, and train your model to understand patterns of benign or malignant cells within the data. Once the model has been trained by going through data iteratively, it can be used to predict your new or unknown cell with a rather high accuracy. This is machine learning! It is the way that a machine learning model can do a doctor's task or at least help that doctor make the process faster. Now, let me give a formal definition of machine learning. Machine learning is the subfield of computer science that gives ”computers the ability to learn without being explicitly programmed.” Let me explain what I mean when I say “without being explicitly programmed.” Assume that you have a dataset of images of animals such as cats and dogs, and you want to have software or an application that can recognize and differentiate them. The first thing that you have to do here is interpret the images as a set of feature sets. For example, does the image show the animal's eyes? If so, what is their size? Does it have ears? What about a tail? How many legs? Does it have wings? Prior to machine learning, each image would be transformed to a vector of features. Then, traditionally, we had to write down some rules or methods in order to get computers to be intelligent and detect the animals. But, it was a failure. Why? Well, as you can guess, it needed a lot of rules, highly dependent on the current dataset, and not generalized enough to detect out-of-sample cases. This is when machine learning entered the scene. Using machine learning, allows us to build a model that looks at all the feature sets, and their corresponding type of animals, and it learns the pattern of each animal. It is a model built by machine learning algorithms. It detects without explicitly being programmed to do so. In essence, machine learning follows the same process that a 4-year-old child uses to learn, understand, and differentiate animals. So, machine learning algorithms, inspired by the human learning process, iteratively learn from data, and allow computers to find hidden insights. These models help us in a variety of tasks, such as object recognition, summarization, recommendation, and so on. Machine Learning impacts society in a very influential way. Here are some real-life examples. First, how do you think Netflix and Amazon recommend videos, movies, and TV shows to its users? They use Machine Learning to produce suggestions that you might enjoy! This is similar to how your friends might recommend a television show to you, based on their knowledge of the types of shows you like to watch. How do you think banks make a decision when approving a loan application? They use machine learning to predict the probability of default for each applicant, and then approve or refuse the loan application based on that probability. Telecommunication companies use their customers' demographic data to segment them, or predict if they will unsubscribe from their company the next month. There are many other applications of machine learning that we see every day in our daily life, such as chatbots, logging into our phones or even computer games using face recognition. Each of these use different machine learning techniques and algorithms. So, let's quickly examine a few of the more popular techniques. The Regression/Estimation technique is used for predicting a continuous value. For example, predicting things like the price of a house based on its characteristics, or to estimate the Co2 emission from a car's engine. A Classification technique is used for Predicting the class or category of a case, for example, if a cell is benign or malignant, or whether or not a customer will churn. Clustering groups of similar cases, for example, can find similar patients, or can be used for customer segmentation in the banking field. Association technique is used for finding items or events that often co-occur, for example, grocery items that are usually bought together by a particular customer. Anomaly detection is used to discover abnormal and unusual cases, for example, it is used for credit card fraud detection. Sequence mining is used for predicting the next event, for instance, the click-stream in websites. Dimension reduction is used to reduce the size of data. And finally, recommendation systems, this associates people's preferences with others who have similar tastes, and recommends new items to them, such as books or movies. We will cover some of these techniques in the next videos. By this point, I'm quite sure this question has crossed your mind, "What is the difference between these buzzwords that we keep hearing these days, such as Artificial intelligence (or AI), Machine Learning and Deep Learning?" Well, let me explain what is different between them. In brief, AI tries to make computers intelligent in order to mimic the cognitive functions of humans. So, Artificial Intelligence is a general field with a broad scope including: Computer Vision, Language Processing, Creativity, and Summarization. Machine Learning is the branch of AI that covers the statistical part of artificial intelligence. It teaches the computer to solve problems by looking at hundreds or thousands of examples, learning from them, and then using that experience to solve the same problem in new situations. And Deep Learning is a very special field of Machine Learning where computers can actually learn and make intelligent decisions on their own. Deep learning involves a deeper level of automation in comparison with most machine learning algorithms. Now that we've completed the introduction to Machine Learning, subsequent videos will focus on reviewing two main components: First, you'll be learning about the purpose of Machine Learning and where it can be applied in the real world; and Second, you'll get a general overview of Machine Learning topics, such as supervised vs unsupervised learning, model evaluation and various Machine Learning algorithms. So now that you have a sense with what's in store on this journey, let's continue our exploration of Machine Learning! Thanks for watching!

1.1.3 Python for Machine Learning

Hello and welcome. In this video, we'll talk about how to use Python for machine learning. So let's get started. Python is a popular and powerful general purpose programming language that recently emerged as the preferred language among data scientists. You can write your machine-learning algorithms using Python, and it works very well. However, there are a lot of modules and libraries already implemented in Python, that can make your life much easier. We try to introduce the Python packages in this course and use it in the labs to give you better hands-on experience. The first package is NumPy which is a math library to work with N-dimensional arrays in Python. It enables you to do computation efficiently and effectively. It is better than regular Python because of its amazing capabilities. For example, for working with arrays, dictionaries, functions, datatypes and working with images you need to know NumPy. SciPy is a collection of numerical algorithms and domain specific toolboxes, including signal processing, optimization, statistics and much more. SciPy is a good library for scientific and high performance computation. Matplotlib is a very popular plotting package that provides 2D plotting, as well as 3D plotting. Basic knowledge about these three packages which are built on top of Python, is a good asset for data scientists who want to work with real-world problems. If you're not familiar with these packages, I recommend that you take the data analysis with Python course first. This course covers most of the useful topics in these packages. Pandas library is a very high-level Python library that provides high performance easy to use data structures. It has many functions for data importing, manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and timeseries. SciKit Learn is a collection of algorithms and tools for machine learning which is our focus here and which you'll learn to use within this course. As we'll be using SciKit Learn quite a bit in the labs, let me explain more about it and show you why it is so popular among data scientists. SciKit Learn is a free Machine Learning Library for the Python programming language. It has most of the classification, regression and clustering algorithms, and it's designed to work with a Python numerical and scientific libraries: NumPy and SciPy. Also, it includes very good documentation. On top of that, implementing machine learning models with SciKit Learn is really easy with a few lines of Python code. Most of the tasks that need to be done in a machine learning pipeline are implemented already in Scikit Learn including pre-processing of data, feature selection, feature extraction, train test splitting, defining the algorithms, fitting models, tuning parameters, prediction, evaluation, and exporting the model. Let me show you an example of how SciKit Learn looks like when you use this library. You don't have to understand the code for now but just see how easily you can build a model with just a few lines of code. Basically, machine-learning algorithms benefit from standardization of the dataset. If there are some outliers or different scales fields in your dataset, you have to fix them. The pre-processing package of SciKit Learn provides several common utility functions and transformer classes to change raw feature vectors into a suitable form of vector for modeling. You have to split your dataset into train and test sets to train your model and then test the model's accuracy separately. SciKit Learn can split arrays or matrices into random train and test subsets for you in one line of code. Then you can set up your algorithm. For example, you can build a classifier using a support vector classification algorithm. We call our estimator instance CLF and initialize its parameters. Now you can train your model with the train set by passing our training set to the fit method, the CLF model learns to classify unknown cases. Then we can use our test set to run predictions, and the result tells us what the class of each unknown value is. Also, you can use the different metrics to evaluate your model accuracy. For example, using a confusion matrix to show the results. And finally, you save your model. You may find all or some of these machine-learning terms confusing but don't worry, we'll talk about all of these topics in the following videos. The most important point to remember is that the entire process of a machine learning task can be done simply in a few lines of code using SciKit Learn. Please notice that though it is possible, it would not be that easy if you want to do all of this using NumPy or SciPy packages. And of course, it needs much more coding if you use pure Python programming to implement all of these tasks. Thanks for watching.

1.1.4 Supervised vs Unsupervised

Hello, and welcome. In this video we'll introduce supervised algorithms versus unsupervised algorithms. So, let's get started. An easy way to begin grasping the concept of supervised learning is by looking directly at the words that make it up. Supervise, means to observe, and direct the execution of a task, project, or activity. Obviously we aren't going to be supervising a person, instead will be supervising a machine learning model that might be able to produce classification regions like we see here. So, how do we supervise a machine learning model? We do this by teaching the model, that is we load the model with knowledge so that we can have it predict future instances. But this leads to the next question which is, how exactly do we teach a model? We teach the model by training it with some data from a labeled dataset. It's important to note that the data is labeled, and what does a labeled dataset look like? Well, it could look something like this. This example is taken from the cancer dataset. As you can see, we have some historical data for patients, and we already know the class of each row. Let's start by introducing some components of this table. The names up here which are called clump thickness, uniformity of cell size, uniformity of cell shape, marginal adhesion and so on are called attributes. The columns are called features which include the data. If you plot this data, and look at a single data point on a plot, it'll have all of these attributes that would make a row on this chart also referred to as an observation. Looking directly at the value of the data, you can have two kinds. The first is numerical. When dealing with machine learning, the most commonly used data is numeric. The second is categorical, that is its non-numeric because it contains characters rather than numbers. In this case, it's categorical because this dataset is made for classification. There are two types of supervised learning techniques. They are classification, and regression. Classification is the process of predicting a discrete class label, or category. Regression is the process of predicting a continuous value as opposed to predicting a categorical value in classification. Look at this dataset. It is related to CO2 emissions of different cars. It includes; engine size, cylinders, fuel consumption, and CO2 emission of various models of automobiles. Given this dataset, you can use regression to predict the CO2 emission of a new car by using other fields such as engine size, or number of cylinders. Since we know the meaning of supervised learning, what do you think unsupervised learning means? Yes, unsupervised learning is exactly as it sounds. We do not supervise the model, but we let the model work on its own to discover information that may not be visible to the human eye. It means, the unsupervised algorithm trains on the dataset, and draws conclusions on unlabeled data. Generally speaking, unsupervised learning has more difficult algorithms than supervised learning since we know little to no information about the data, or the outcomes that are to be expected. Dimension reduction, density estimation, market basket analysis, and clustering are the most widely used unsupervised machine learning techniques. Dimensionality reduction, and/or feature selection, play a large role in this by reducing redundant features to make the classification easier. Market basket analysis is a modeling technique based upon the theory that if you buy a certain group of items, you're more likely to buy another group of items. Density estimation is a very simple concept that is mostly used to explore the data to find some structure within it. And finally, clustering: Clustering is considered to be one of the most popular unsupervised machine learning techniques used for grouping data points, or objects that are somehow similar. Cluster analysis has many applications in different domains, whether it be a bank's desire to segment his customers based on certain characteristics, or helping an individual to organize in-group his, or her favorite types of music. Generally speaking though, clustering is used mostly for discovering structure, summarization, and anomaly detection. So, to recap, the biggest difference between supervised and unsupervised learning is that supervised learning deals with labeled data while unsupervised learning deals with unlabeled data. In supervised learning, we have machine learning algorithms for classification and regression. In unsupervised learning, we have methods such as clustering. In comparison to supervised learning, unsupervised learning has fewer models and fewer evaluation methods that can be used to ensure that the outcome of the model is accurate. As such, unsupervised learning creates a less controllable environment as the machine is creating outcomes for us. Thanks for watching.

Chapter 2

Week 2

2.1 Linear Regression

2.1.1 Introduction to Regression

Hello and welcome! In this video we'll be giving a brief introduction to regression. So let's get started. Look at this data set. It's related to co2 emissions from different cars. It includes engine size, number of cylinders, fuel consumption, and co2 emission from various automobile models.

The question is: given this data set can we predict the co2 emission of a car using other fields such as engine size or cylinders? Let's assume we have some historical data from different cars and assume that a car such as in row 9 has not been manufactured yet, but we're interested in estimating its approximate co2 emission after production. Is it possible?

We can use regression methods to predict a continuous value such as co2 emission using some other variables. Indeed regression is t**he process of predicting a continuous value**. In regression there are **two types of variables**: a **dependent variable** and **one or more independent variables**.

The dependent variable can be seen as the state, target, or final goal we study and try to predict. And the independent variables, also known as explanatory variables, can be seen as the causes of those states. The independent variables are shown conventionally by X and the dependent variable is notated by Y.

A regression model relates Y or the dependent variable to a function of X i.e. the independent variables.

The key point in the regression is that our dependent value should be continuous and cannot be a discrete value. However, the independent variable, or variables, can be measured on either a categorical or continuous measurement scale.

So, what we want to do here is to use the historical data of some cars using one or more of their features and from that data make a model. We use regression to build such a regression estimation model; then the model is used to predict the expected co2 emission for a new or unknown car.

Basically there are two types of regression models **simple regression** and **multiple regression**. Simple regression is when **one independent variable is used** to estimate a dependent variable. It **can be either linear or non-linear**. For example, predicting co2 emission using the variable of engine size.

Linearity of regression is based on the nature of relationship between independent and dependent variables. When **more than one independent variable is present** the process is called **multiple linear regression**. For example, predicting co2 emission using engine size and the number of cylinders in any given car. Again, depending on the relation between dependent and independent variables it can be either linear or non-linear regression.

Let's examine some sample applications of regression. Essentially we use regression when we want to estimate a continuous value. For instance, one of the applications of regression analysis could be in the area of sales forecasting. You can try to predict a sales person's total yearly sales from independent variables such as age, education, and years of experience. It can also be used in the field of psychology, for example, to determine individual satisfaction, based on demographic and psychological factors. We can use regression analysis to predict the price of a house in an area, based on its size number of bedrooms, and so on. We can even use it to predict employment income for independent variables such as hours of work, education, occupation, sex, age, years of experience, and so on.

Indeed, you can find many examples of the usefulness of regression analysis in these and many other fields or domains, such as finance, healthcare, retail, and more. We have many regression algorithms, each of them has its own importance and a specific condition to which their application is best suited. And while we've covered just a few of them in this course, it gives you enough base knowledge for you to explore different regression techniques.

2.1.2 Simple Linear Regression

In this video, we'll be covering linear regression. You don't need to know any linear algebra to understand topics in linear regression. This high-level introduction will give you enough background information on linear regression to be able to use it effectively on your own problems. So let's get started.

Let's take a look at this data set. It's related to the Co2 emission of different cars. It includes engine size, cylinders, fuel consumption and Co2 emissions for various car models. The question is, given this data set, can we predict the Co2 emission of a car using another field such as engine size?

Quite simply, yes. We **can use linear regression to predict a continuous value such as Co2 emission by using other variables**. Linear regression is the approximation of a linear model used to describe the relationship between two or more variables. In simple linear regression, there are two variables, a dependent variable and an independent variable. The **key point** in the linear regression is that our dependent value **should be continuous and cannot be a discrete value**. However, the independent variables can be measured on either a categorical or continuous measurement scale.

There are two types of linear regression models. They are simple regression and multiple regression. Simple linear regression is when one independent variable is used to estimate a dependent variable. For example, predicting Co2 emission using the engine size variable. When more than one independent variable is present the process is called multiple linear regression, for example, predicting Co2 emission using engine size and cylinders of cars. Our focus in this video is on simple linear regression.

Now let's see how linear regression works. Okay, so let's look at our data set again. To understand linear regression, we can plot our variables here. We show engine size as an independent variable and emission as the target value that we would like to predict. A scatter plot clearly shows the relation between variables where changes in one variable explain or possibly cause changes in the other variable. Also, it indicates that these variables are linearly related. With linear regression **you can fit a line through the data**. For instance, as the engine size increases, so do the emissions. With linear regression you can model the relationship of these variables. A good model can be used to predict what the approximate emission of each car is.

How do we use this line for prediction now?

Let us assume for a moment that the line is a good fit of the data. We can use it to predict the emission of an unknown car. For example, for a sample car with engine size 2.4, you can find the emission is 214.

Now, let's talk about what the fitting line actually is.

We're going to predict the target value y. In our case using the independent variable engine size represented by x_1. The fit line is shown traditionally as a polynomial. In a simple regression problem, a single x, the form of the model would be $\theta_0 + \theta_1 x_1$. In this equation, y hat ($\hat{y}$) is the dependent variable of the predicted value. And x_1 is the independent variable.

Theta 0 (θ_0) and theta 1 (θ_1) are the parameters of the line that we must adjust. Theta 1 is known as the slope or gradient of the fitting line and theta 0 is known as the intercept.

Theta 0 (θ_0) and theta 1 (θ_1)are also called the **coefficients of the linear equation**.

You can interpret this equation as y hat being a function of x1, or y hat being dependent of x1. How would you draw a line through the points? And how do you determine which line fits best?

Linear regression estimates the coefficients of the line. This means we must calculate theta 0 and theta 1 to find the best line to fit the data. This line would best estimate the emission of the unknown data points.

Let's see how we can find this line or, to be more precise, how we can adjust the parameters to make the line the best fit for the data. For a moment, let's assume we've already found the best fit line for our data. Now, let's go through all the points and check how well they align with this line. Best fit here means that if we have, for instance, a car with engine size $x1 = 5.4$ and actual $Co2 = 250$, its Co2 should be predicted very close to the actual value, which is $y = 250$ based on historical data. But if we use the fit line, or better to say using our polynomial with known parameters to predict the Co2 emission, it will return $\hat{y} = 340$. Now if you compare the actual value of the emission of the car with what we've predicted using our model, you will find out that we have a 90 unit error. This means our prediction line is not accurate. This error is also called the **residual error**. So we can say the **error is the distance from the data point to the fitted regression line**.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.1}$$

The mean of all residual errors shows how poorly the line fits with the whole data set. Mathematically it can be shown by the equation **Mean Squared Error**, shown as **MSE**. Our objective is to **find a line where the mean of all these errors is minimized**. In other words, the mean error of the prediction using the fit line should be minimized. Let's reword it more technically. The **objective of linear regression**, is to **minimize this MSE equation and to minimize it, we should find the best parameters theta 0 and theta 1**. Now the question is **how to find theta 0 and theta 1 in such a way that it minimizes this error?**

How can we find such a perfect line? Or said another way, how should we find the best parameters for our line? Should we move the line a lot randomly and calculate the MSE value every time and choose the minimum one? Not really. Actually, we have **two options here**. Option one, we can use a **mathematic approach**, or option two, we can use an **optimization approach**. Let's see how we could easily use a mathematic formula to find the theta 0 and

As mentioned before, theta 0 and theta 1 in the simple linear regression are the coefficients of the fit line. We can use a simple equation to estimate these coefficients. That is, given that it's a simple linear regression with only two parameters, and knowing that theta 0 and theta 1 are the intercept and slope of the line, we can estimate them directly from our data. It requires that we calculate the mean of the independent and dependent or target columns from the data set. Notice that all of the data must be available to traverse and calculate the parameters. It can be shown that the intercept and slope can be calculated using these equations.

We can start off by estimating the value for theta 1. This is how you can find the slope of a line based on the data. X bar is the average value for the engine size in our data set. Please consider that we have nine rows here, rows 0 to 8. First we calculate the average of x1 and of y, then we plug it into the slope equation to find theta 1.

The xi and yi in the equation refer to the fact that we need to repeat these calculations across all values in our data set. And i refers to the ith value of x or y. Applying all values, we find theta 1 equals 39. It is our second parameter. It is used to calculate the first parameter which is the intercept of the line.

Now we can plug theta 1 into the line equation to find theta 0. It is easily calculated hat theta 0 equals 125.74. So these are the two parameters for the line, where theta 0 is also called the bias coefficient, and theta 1 is the coefficient for the Co2 emission column.

As a side note, you really don't need to remember the formula for calculating these parameters, as **most of the libraries used for machine learning in Python**, R and Scala **can easily find these parameters for you**. But it's always good to understand how it works. Now, we can write down the polynomial of the line.

$$\hat{y} = \theta_1 x + \theta_0 = 39x + 125.74 \tag{2.2}$$

So we know how to find the best fit for our data and its equation. Now the question is how can we use it to predict the emission of a new car based on its engine size?

After we found the parameters of the linear equation, making predictions is as simple as solving the equation for a specific set of inputs.

Imagine we are predicting Co2 emission, or y, from engine size, or x for the automobile in record number 9. Our linear regression model representation for this problem would be $\hat{y} = \theta_1 x + \theta_0$. Or if we map it to our data set, it would be $Co2Emission = \theta_0 + \theta_1 EngineSize$.

As we saw, we can find theta 0, theta 1 using the equations that we just talked about. Once found, we can plug in the equation of the linear model. For example, let's use $\theta_0 = 125$ and $\theta_1 = 39$. So we can rewrite the linear model as $Co2Emission = 125 + 39EngineSize$. Now let's plug in the 9th row of our data set and calculate the Co2 emission for a car with an engine size of 2.4. So $Co2Emission = 125 + 39x2.4$. Therefore, we can predict that the Co2Emission for this specific car would be 218.6.

Let's talk a bit about why linear regression is so useful. Quite simply, it is the most basic regression to use and understand. In fact, one reason why linear regression is so useful is that **it's fast**. It also **doesn't require tuning of parameters**. So something like tuning the K parameter and K nearest neighbors, or the learning rate in neural networks isn't something to worry about. Linear regression is also **easy to understand**, and **highly interpretable**.

2.1.3 Model Evaluation in Regression Models

We'll be covering model evaluation. So let's get started. The goal of regression is to build a model to accurately predict an unknown case. To this end, we have to perform regression evaluation after building the model.

We'll introduce and discuss two types of evaluation approaches that can be used to achieve this goal. These approaches are:

- Train and test on the same dataset
- Train/test split.

We'll talk about what each of these are, as well as the pros and cons of using each of these models. Also, we'll introduce some metrics for accuracy of regression models.

Let's look at the first approach. When considering evaluation models, we clearly want to choose the one that will give us the most accurate results. So, the question is, how can we calculate the accuracy of our model? In other words, how much can we trust this model for prediction of an unknown sample using a given dataset and having built a model such as linear regression?

One of the solutions is to select a portion of our dataset for testing. For instance, assume that we have 10 records in our dataset. We use the entire dataset for training, and we build a model using this training set. Now, we select a small portion of the dataset, such as row number six to nine, but without the labels. This set is called a **test set**, which has the labels, but the labels are not used for prediction and is used only as ground truth. The **labels are called actual values of the test set**. Now we pass the feature set of the testing portion to our built model and predict the target values. Finally, we compare the predicted values by our model with the actual values in the test set.

This indicates how accurate our model actually is. There are different metrics to report the accuracy of the model, but most of them work generally based on the similarity of the predicted and actual values.

Let's look at one of the simplest metrics to calculate the accuracy of our regression model.

$$Error = \frac{1}{n}\sum_{j=1}^{n}|y_i - \hat{y}| \tag{2.3}$$

As mentioned, we just compare the actual values y with the predicted values, which is noted as $\hat{y}$ for the testing set. The **error of the model is calculated as the average difference between the predicted and actual values for all the rows**. We can write this error as an equation.

So, the first evaluation approach we just talked about is **the simplest one, train and test on the same dataset**. Essentially, the name of this approach says it all. You train the model on the entire dataset, then you test it using a portion of the same dataset. In a general sense, when you test with a dataset in which you know the target value for each data point, you're able to obtain a percentage of accurate predictions for the model. This evaluation approach would most likely **have a high training accuracy and the low out-of-sample accuracy since the model knows all of the testing data points from the training**. What is **training accuracy** and **out-of-sample accuracy**?

We said that training and testing on the same dataset produces a high training accuracy, but what exactly is training accuracy?

Training accuracy is the **percentage of correct predictions that the model makes when using the test dataset**. However, a high training accuracy isn't necessarily a good thing. For instance, having a high training accuracy **may result in an over-fit the data**. This means that the model is overly trained to the dataset, which **may capture noise and produce a non-generalized model**.

Out-of-sample accuracy is the **percentage of correct predictions that the model makes on data that the model has not been trained on**. Doing a train and test on the same dataset will most likely have low out-of-sample accuracy due to the likelihood of being over-fit. **It's important that our models have high out-of-sample accuracy because the purpose of our model is, of course, to make correct predictions on unknown data**. So, how can we improve out-of-sample accuracy?

One way is to use another evaluation approach called **train/test split**. In this approach, we select a portion of our dataset for training, for example, row zero to five, and the rest is used for testing, for example, row six to nine. The model is built on the training set. Then, the test feature set is passed to the model for prediction. Finally, the predicted values for the test set are compared with the actual values of the testing set. The second evaluation approach is called train/test split. Train/test split involves splitting the dataset into training and testing sets respectively, which are mutually exclusive. After which, you train with the training set and test with the testing set. This **will provide a more accurate evaluation on out-of-sample accuracy** because the testing dataset is not part of the dataset that has been used to train the data. It is **more realistic for real-world problems**. This means that we know the outcome of each data point in the dataset, making it great to test with. Since this data has not been used to train the model, the model has no knowledge of the outcome of these data points. So, in essence, **it's truly out-of-sample testing**.

However, please ensure that you train your model with the testing set afterwards, as you don't want to lose potentially valuable data. The issue with train/test split is that **it's highly dependent on the datasets on which the data was trained and tested**. The variation of this causes train/test split to have a better out-of-sample prediction than training and testing on the same dataset, but it still has some problems due to this dependency.

Another evaluation model, called **K-fold cross-validation**, resolves most of these issues. How do you fix a high variation that results from a dependency? Well, you average it. Let me explain the basic concept of K-fold cross-validation to see how we can solve this problem.

The entire dataset is represented by the points in the image at the top left. If we have K equals four folds, then we split up this dataset as shown here. In the first fold for example, we use the first 25 percent of the dataset for testing and the rest for training. The model is built using the training set and is evaluated using the test set. Then, in the next round or in the second fold, the second 25 percent of the dataset is used for testing and the rest for training the model. Again, the accuracy of the model is calculated. We continue for all folds. Finally, the result of all four evaluations are averaged. That is, the accuracy of each fold is then averaged, keeping in mind that each fold is distinct, where no training data in one fold is used in another.

K-fold cross-validation in its simplest form **performs multiple train/test splits, using the same dataset where each split is different**. Then, **the result is average to produce a more consistent out-of-sample accuracy**. We wanted to show you an evaluation model that addressed some of the issues we've described in the previous approaches.

However, going in-depth with K-fold cross-validation model is out of the scope for this course.

2.1.4 Evaluation Metrics in Regression Models

We'll be covering **accuracy metrics for model evaluation**. So let's get started. **Evaluation metrics are used to explain the performance of a model**.

Let's talk more about the model evaluation metrics that are used for regression. As mentioned, basically, we can compare the actual values and predicted values to calculate the accuracy of our regression model. Evaluation metrics **provide a key role in the development of a model as it provides insight to areas that require improvement**.

We'll be reviewing a number of model evaluation metrics, including **Mean Absolute Error**, **Mean Squared Error**, and **Root Mean Squared Error**, but before we get into defining these, we need to define what an error actually is.

In the context of regression, **the error of the model is the difference between the data points and the trend line generated by the algorithm**. Since there are multiple data points, an error can be determined in multiple ways.

- Mean Absolute Error is the **mean of the absolute value of the errors**. This is the easiest of the metrics to understand, since it's just the average error.

$$MAE = \frac{1}{n}\sum_{j=1}^{n}|y_i - \hat{y}_i| \tag{2.4}$$

- Mean Squared Error is the **mean of the squared error**. It's more popular than Mean Absolute Error because the focus is geared more towards large errors. This is due to the squared term, exponentially increasing larger errors in comparison to smaller ones.

$$MSE = \frac{1}{n}\sum_{j=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.5}$$

- Root Mean Squared Error is **the square root of the mean squared error**. This is one of the most popular of the evaluation metrics because Root Mean Squared Error **is interpretable in the same units as the response vector** or Y units, making it **easy to relate its information**.

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_i - \hat{y}_i)^2} \tag{2.6}$$

- Relative absolute error, also known as residual sum of square, where Y bar ($\bar{y}$) is a mean value of Y, takes the **total absolute error and normalizes it**. By dividing by the total absolute error of the simple predictor.

$$RAE = \frac{\sum_{j=1}^{n}|y_i - \hat{y}_i|}{\sum_{j=1}^{n}|y_i - \bar{y}_i|} \tag{2.7}$$

- Relative squared error is very similar to relative absolute error, but is widely adopted by the data science community as it is used for calculating R-squared.

$$RSE = \frac{\sum_{j=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{j=1}^{n}(y_i - \bar{y}_i)^2} \tag{2.8}$$

$$R^2 = 1 - RSE \tag{2.9}$$

R-squared is not an error per say but is a popular metric for the accuracy of your model. It represents **how close the data values are to the fitted regression line**. **The higher the R-squared, the better the model** fits your data. Each of these metrics can be used for quantifying of your prediction. The choice of metric completely depends on the type of model your data type and domain of knowledge. Unfortunately, further review is out of scope of this course.

2.1.5 Ungraded External Tool

You can find the Jupyter Notebook and its version in pdf before and after was runned in W1 folder.

File: ML0101EN-Reg-Simple-Linear-Regression-Co2-py-v1

2.1.6 Multiple Linear Regression

We'll be covering multiple linear regression. As you know, there are two types of linear regression models, simple regression and multiple regression.

Simple linear regression is when one independent variable is used to estimate a dependent variable. For example, predicting CO_2 emission using the variable of engine size. In reality, there are multiple variables that predict the CO_2 emission.

When multiple independent variables are present, the process is called multiple linear regression. For example, predicting CO_2 emission using engine size and the number of cylinders in the car's engine. Our focus in this video is on multiple linear regression. The good thing is that **multiple linear regression is the extension of the simple linear regression model**.

So, I suggest you go through the simple linear regression video first if you haven't watched it already. Before we dive into a sample dataset and see how multiple linear regression works, I want to tell you what kind of problems it can solve, when we should use it, and specifically, what kind of questions we can answer using it. Basically, **there are two applications for multiple linear regression**.

First, it can be used when we would like **to identify the strength of the effect that the independent variables have on the dependent variable**. For example, does revision time, test anxiety, lecture attendance and gender have any effect on exam performance of students?

Second, it can be used **to predict the impact of changes**, that is, to understand **how the dependent variable changes when we change the independent variables**. For example, if we were reviewing a person's health data, a multiple linear regression can tell you how much that person's blood pressure goes up or down for every unit increase or decrease in a patient's body mass index holding other factors constant.

As is the case with simple linear regression, multiple linear regression is a method of predicting a continuous variable. It uses multiple variables called **independent variables or predictors** that best predict the value of the **target variable** which is also called the **dependent variable**.

In multiple linear regression, the target value Y, **is a linear combination of independent variables** X. For example, you can predict how much CO_2 a car might admit due to independent variables such as the car's engine size, number of cylinders, and fuel consumption.

Multiple linear regression **is very useful because you can examine which variables are significant predictors of the outcome variable**. Also, you can find out **how each feature impacts the outcome variable**. Again, as is the case in simple linear regression, if you manage to build such a regression model, you can use it to predict the emission amount of an unknown case such as record number nine.

Generally, the model is of the form $\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2$ and so on, up to $\theta_n x_n$. Mathematically, we can show it as a **vector form** as well. This means **it can be shown as a dot product of two vectors**; the parameters vector and the feature set vector. Generally, **we can show the equation for a multidimensional space as** theta transpose x ($\bar{y} = \theta^T X$), where theta is an n by one vector of unknown parameters in a multi-dimensional space, and X is the vector of the featured sets, as theta is a vector of coefficients and is supposed to be multiplied by x.

Conventionally, it is shown as **transpose theta**. Theta is also called the **parameters or weight vector of the regression equation**. Both these terms can be used interchangeably, and X is the feature set which represents a car ($X = [1, x_1, x_2, ...]^[T]$). For example, x_1 for engine size or x_2 for cylinders, and so on. **The first element of the feature set would be set to one**, because it turns that theta zero into the intercept or biased parameter when the vector is multiplied by the parameter vector.

Please notice that theta transpose x ($\bar{y} = \theta^T X$) in a one-dimensional space is the equation of a line, it is what we use in simple linear regression. In higher dimensions when we have more than one input or x the line is called a plane or a hyperplane, and this is what we use for multiple linear regression. So, **the whole idea is to find the best fit hyperplane for our data**.

To this end and as is the case in linear regression, we should estimate the values for theta vector that best predict the value of the target field in each row. To achieve this goal, we have to minimize the error of the prediction. Now, the question is, how do we find the optimized parameters?

To find the optimized parameters for our model, we should first understand what the optimized parameters are, then we will find a way to optimize the parameters. In short, optimized parameters are the ones which lead to a model with the fewest errors. Let's assume for a moment that we have already found the parameter vector of our model, **it means we already know the values of theta vector**. Now we can use the model and the feature set of the first row of our dataset to predict the CO_2 emission for the first car, correct? If we plug the feature set values into the model equation, we find $\hat{y}$.

Let's say for example, it returns $\hat{y}_i = 140$ as the predicted value for this specific row, what is the actual value? Y equals 196. How different is the predicted value from the actual value of $y_i = 196$? Well, we can calculate it quite simply as 196 subtract 140, which of course equals $y_i - \hat{y}_i = 196 - 140 = 56$. This is the error of our model only for one row or one car in our case. As is the case in linear regression, we can say the error here is the distance from the data point to the fitted regression model.

The mean of all residual errors shows how bad the model is representing the data set, it is called the mean squared error, or MSE. Mathematically, MSE can be shown by an equation.

$$MSE = \frac{1}{n} \sum_{j=1}^{n} (y_i - \hat{y}_i)^2 \tag{2.10}$$

While this is not the only way to expose the error of a multiple linear regression model, it is one of the most popular ways to do so. The best model for our data set is the one with minimum error for all prediction values. So, **the objective of multiple linear regression is to minimize the MSE equation**.

To minimize it, we should find the best parameters theta, but how? Okay, how do we find the parameter or coefficients for multiple linear regression? There are many ways to estimate the value of these coefficients. However, the most common methods are the **ordinary least squares** and **optimization approach**.

Ordinary least squares tries to estimate the values of the coefficients by minimizing the mean square error. This approach uses the data as a matrix and uses linear algebra operations to estimate the optimal values for the theta. The **problem** with this technique is the **time complexity of calculating matrix operations** as it can take a very long time to finish. **When the number of rows in your data set is less than 10,000**, you can think of this technique as an option. However, for greater values, you should try other faster approaches.

The second option is to use an optimization algorithm to find the best parameters. That is, you can use a process of optimizing the values of the coefficients by iteratively minimizing the error of the model on your training data. For example, you **can use gradient descent** which starts optimization with random values for each coefficient, then calculates the errors and tries to minimize it through y's changing of the coefficients in multiple iterations. Gradient descent is a **proper approach if you have a large data set**. Please understand however, that there are other approaches to estimate the parameters of the multiple linear regression that you can explore on your own.

After you find the best parameters for your model, you can go to the prediction phase. After we found the parameters of the linear equation, making predictions is as simple as solving the equation for a specific set of inputs. Imagine we are predicting CO_2 emission or Y from other variables for the automobile in record number nine. Our linear regression model representation for this problem would be y hat equals theta transpose x. Once we find the parameters, we can plug them into the equation of the linear model. For example, let's use theta zero equals 125, theta one equals 6.2, theta two equals 14, and so on. If we map it to our data set, we can rewrite the linear model as CO_2 emissions equals 125 plus 6.2 multiplied by engine size, plus 14 multiplied by cylinder, and so on. As you can see, multiple linear regression estimates the relative importance of predictors.

$$\hat{y} = 125 + 6.2x_1 + 14x_2 + ... \tag{2.11}$$

$$Co2Em = 125 + 6.2EngSize + 14Cylinders + ... \tag{2.12}$$

For example, it shows cylinder has higher impact on CO_2 emission amounts in comparison with engine size. Now, let's plug in the ninth row of our data set and calculate the CO_2 emission for a car with the engine size of 2.4. So, CO_2 emission equals 125 plus 6.2 times 2.4, plus 14 times four, and so on. We can predict the CO_2 emission for this specific car would be 214.1.

$$Co2Em = 125 + 6.2 * 2.4 + 14 * 4 = 214.1 \tag{2.13}$$

Now, let me address some concerns that you might already be having regarding multiple linear regression. As you saw, you can use multiple independent variables to predict a target value in multiple linear regression. It **sometimes results in a better model compared to using a simple linear regression** which uses only one independent variable to predict the dependent variable. Now the question is **how, many independent variable should we use** for the prediction? Should we use all the fields in our data set? Does adding independent variables to a multiple linear regression model always increase the accuracy of the model? Basically, **adding too many independent variables without any theoretical justification may result in an overfit model**.

An overfit model is a real problem because it is **too complicated for your data set and not general enough to be used for prediction**. So, it is recommended to avoid using many variables for prediction. There are different ways to avoid overfitting a model in regression, however that is outside the scope of this video.

The next question is, **should independent variables be continuous**? Basically, **categorical independent variables can be incorporated into a regression model by converting them into numerical variables**. For example, given a binary variables such as car type, the code dummy zero for manual and one for automatic cars.

As a last point, remember that **multiple linear regression is a specific type of linear regression**. So, there **needs to be a linear relationship between the dependent variable and each of your independent variables**. There are a number of ways to check for linear relationship. For example, you can use scatter plots and then visually checked for linearity. If the relationship displayed in your scatter plot is not linear, then you need to use non-linear regression. This concludes our video. Thanks for watching. (Music)

2.2 Non-Linear Regression

2.2.1 Non-Linear Regression

Non-linear regression basics. These data points correspond to China's gross domestic product or GDP from $1960 - 2014$. The first column is the years and the second is China's corresponding annual gross domestic income in US dollars for that year. This is what the data points look like.

Now, we have a couple of interesting questions:

First, **can GDP be predicted based on time**? Second, can we use a simple linear regression to model it?

Indeed. If the **data shows a curvy trend, then linear regression would not produce very accurate results** when compared to a non-linear regression. Simply because, as the name implies, linear regression presumes that the data is linear.

The scatter plot shows that there seems to be a strong relationship between GDP and time, but the relationship is not linear. As you can see, the growth starts off slowly, then from 2005 onward, the growth is very significant. Finally, it decelerates slightly in the 2010s. It looks like either a logistical or exponential function. So, **it requires a special estimation method of the non-linear regression procedure**.

For example, if we assume that the model for these data points are exponential functions, such as $\hat{y} = \theta_0 + \theta_1 \theta_2^x$, our job is to estimate the parameters of the model, i.e., Thetas, and use the fitted model to predict GDP for unknown or future cases.

In fact, many different regressions exists that can be used to fit whatever the dataset looks like. You can see a quadratic and cubic regression lines here, and it can go on and on to infinite degrees. In essence, we can call all of these **polynomial regression**, where the relationship between the independent variable X and the dependent variable Y is modeled as an Nth degree polynomial in X.

With many types of regression to choose from, there's a good chance that one will fit your dataset well. Remember, it's important to pick a regression that fits the data the best. So, what is polynomial regression? Polynomial regression fits a curve line to your data. A simple example of polynomial with degree three is shown as:

$$\hat{y} = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 \tag{2.14}$$

Where Thetas are parameters to be estimated that makes the model fit perfectly to the underlying data.

Though the relationship between X and Y is non-linear here and polynomial regression can't fit them, a polynomial regression model **can still be expressed as linear regression**. I know it's a bit confusing, but let's look at an example. Given the third degree polynomial equation, by defining $X_1 = X$ and $X_2 = X^2$ and so on, the model is converted to a simple linear regression with new variables as:

$$\hat{y} = \theta_0 + \theta_1 x + \theta_2 x_2 + \theta_3 x_3 \tag{2.15}$$

This model is linear in the parameters to be estimated, right? Therefore, this **polynomial regression is considered to be a special case of traditional multiple linear regression**. So, you can use the same mechanism as linear regression to solve such a problem. Therefore, **polynomial regression models can fit using the model of least squares**.

Least squares is a method for estimating the unknown parameters in a linear regression model by minimizing the sum of the squares of the differences between the observed dependent variable in the given dataset and those predicted by the linear function.

So, what is non-linear regression exactly?

First, non-linear regression is **a method to model a non-linear relationship between the dependent variable and a set of independent variables**.

Second, for a model to be considered non-linear, $\hat{y}$ **must be a non-linear function of the parameters** θ, not necessarily the features X.

$$\begin{gathered} \hat{y} = \theta_0 + \theta_1^2 x \\ \hat{y} = \theta_0 + \theta_1 \theta_2^x \\ \hat{y} = \frac{\theta_0}{1 + \theta_1^{x-\theta_2}} \end{gathered} \tag{2.16}$$

When it comes to non-linear equation, it can be the shape of exponential, logarithmic, and logistic, or many other types. As you can see in all of these equations, the change of $\hat{y}$ depends on changes in the parameters θ, not necessarily on X only. That is, **in non-linear regression, a model is non-linear by parameters**. In contrast to linear regression, **we cannot use the ordinary least squares method to fit the data in non-linear regression**.

In general, estimation of the parameters is not easy. Let me answer two important questions here.

First, how can I know if a problem is linear or non-linear in an easy way? To answer this question, we have to do two things.

- The first is to visually figure out if the relation is linear or non-linear. It's best to plot bivariate plots of output variables with each input variable.
- You can calculate the correlation coefficient between independent and dependent variables, and if, for all variables, it is 0.7 or higher, there is a linear tendency and thus, it's not appropriate to fit a non-linear regression.

The second thing we have to do is to use non-linear regression instead of linear regression when we cannot accurately model the relationship with linear parameters.

The second important question is, how should I model my data if it displays non-linear on a scatter plot? Well, to address this, you have to use either a polynomial regression, use a non-linear regression model, or transform your data, which is not in scope for this course.

2.2.2 Lab: Polynomial Regression

Importing Needed packages

```
import matplotlib.pyplot as plt
import pandas as pd
import pylab as pl
import numpy as np
%matplotlib inline
```

Understading the Data

The file is `FuelConsumption.csv` which contains model-specific fuel consumption ratings and estimated carbon dioxide emissions for new light-duty vehicles for retail sale in Canada.

- MODELYEAR e.g. 2014
- MAKE e.g. Acura
- MODEL e.g. ILX
- VEHICLE CLASS e.g. SUV
- ENGINE SIZE e.g. 4.7
- CYLINDERS e.g 6
- TRANSMISSION e.g. A6
- FUEL CONSUMPTION in CITY(L/100 km) e.g. 9.9
- FUEL CONSUMPTION in HWY (L/100 km) e.g. 8.9
- FUEL CONSUMPTION COMB (L/100 km) e.g. 9.2
- CO2 EMISSIONS (g/km) e.g. 182 –¿ low –¿ 0

Reading the data in

```
df = pd.read_csv("FuelConsumption.csv")

# take a look at the dataset
df.head()
```

Let's select some features that we want to use for regression.

```
cdf = df[['ENGINESIZE','CYLINDERS','FUELCONSUMPTION_COMB','CO2EMISSIONS']]
cdf.head(9)
```

Let's plot Emission values with respect to Engine size:

```
plt.scatter(cdf.ENGINESIZE, cdf.CO2EMISSIONS,  color='blue')
plt.xlabel("Engine size")
plt.ylabel("Emission")
plt.show()
```

Creating train and test dataset

Train/Test Split involves splitting the dataset into training and testing sets respectively, which are mutually exclusive. After which, you train with the training set and test with the testing set.

```
msk = np.random.rand(len(df)) < 0.8
train = cdf[msk]
test = cdf[~msk]
```

Polynomial regression

Sometimes, the trend of data is not really linear, and looks curvy. In this case we can use Polynomial regression methods. In fact, many different regressions exist that can be used to fit whatever the dataset looks like, such as quadratic, cubic, and so on, and it can go on and on to infinite degrees.

In essence, we can call all of these, polynomial regression, where the relationship between the independent variable x and the dependent variable y is modeled as an nth degree polynomial in x. Lets say you want to have a polynomial regression (let's make 2 degree polynomial):

$$\hat{y} = b + \theta_1 x + \theta_2 x^2 \tag{2.17}$$

Now, the question is: how we can fit our data on this equation while we have only x values, such as Engine Size?

Well, we can create a few additional features: 1, x , and x^2 .

PolynomialFeatures() function in Scikit-learn library, drives a new feature sets from the original feature set. That is, a matrix will be generated consisting of all polynomial combinations of the features with degree less than or equal to the specified degree. For example, lets say the original feature set has only one feature, ENGINESIZE. Now, if we select the degree of the polynomial to be 2, then it generates 3 features, degree=0, degree=1 and degree=2:

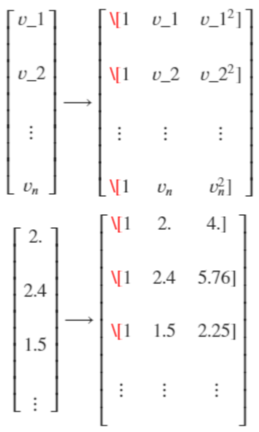
Figure 2.1: Matrix.

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn import linear_model
train_x = np.asanyarray(train[['ENGINESIZE']])
train_y = np.asanyarray(train[['CO2EMISSIONS']])

test_x = np.asanyarray(test[['ENGINESIZE']])
test_y = np.asanyarray(test[['CO2EMISSIONS']])


poly = PolynomialFeatures(degree=2)
train_x_poly = poly.fit_transform(train_x)
train_x_poly
```

`fit_transform` takes our x values, and output a list of our data raised from power of 0 to power of 2 (since we set the degree of our polynomial to 2).

The equation and the sample example is displayed below.

It looks like feature sets for multiple linear regression analysis, right? Yes. It Does. Indeed, Polynomial regression is a special case of linear regression, with the main idea of how do you select your features.

Just consider replacing the x with x_1 , x^2 with x_2 , and so on. Then the degree 2 equation would be turn into:

$$\hat{y} = b + \theta_1 x_1 + \theta_2 x_2 \tag{2.18}$$

Now, we can deal with it as 'linear regression' problem. Therefore, this polynomial regression is considered to be a special case of traditional multiple linear regression. So, you can use the same mechanism as linear regression to solve such a problems.

so we can use LinearRegression() function to solve it:

```
clf = linear_model.LinearRegression()
train_y_ = clf.fit(train_x_poly, train_y)
# The coefficients
print ('Coefficients: ', clf.coef_)
print ('Intercept: ',clf.intercept_)

#Coefficients:  [[ 0.         50.14228586 -1.47509466]]
#Intercept:  [107.48765006]
```

As mentioned before, Coefficient and Intercept , are the parameters of the fit curvy line. Given that it is a typical multiple linear regression, with 3 parameters, and knowing that the parameters are the intercept and coefficients of hyperplane, sklearn has estimated them from our new set of feature sets. Lets plot it:

```
plt.scatter(train.ENGINESIZE, train.CO2EMISSIONS,  color='blue')
XX = np.arange(0.0, 10.0, 0.1)
yy = clf.intercept_[0]+ clf.coef_[0][1]*XX+ clf.coef_[0][2]*np.power(XX, 2)
plt.plot(XX, yy, '-r' )
plt.xlabel("Engine size")
plt.ylabel("Emission")

#Text(0, 0.5, 'Emission')
```

Evaluation

```
from sklearn.metrics import r2_score

test_x_poly = poly.fit_transform(test_x)
test_y_ = clf.predict(test_x_poly)

print("Mean absolute error: %.2f" % np.mean(np.absolute(test_y_ - test_y)))
print("Residual sum of squares (MSE): %.2f" % np.mean((test_y_ - test_y) ** 2))
print("R2-score: %.2f" % r2_score(test_y,test_y_ ) )

#Mean absolute error: 24.74
#Residual sum of squares (MSE): 991.48
#R2-score: 0.77
```

2.2.3 Lab: Non-linear Regression

Chapter 3

Week 3

3.1 K-Nearest Neighbours

3.1.1 Introduction to Classification

We'll give you an introduction to classification. So let's get started.

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 etcetera, 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:

- Decision trees
- Naive bayes
- Linear discriminant analysis
- K-nearest neighbor
- Logistic regression
- Neural networks
- Support vector machines (SVM)

There are many types of classification algorithms. We will only cover a few in this course.

3.1.2 K-Nearest Neighbours

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
- 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.

1. Pick a value for K.
2. Calculate the distance from the new case hold out from each of the cases in the dataset.
3. Search for the K-observations in the training data that are nearest to the measurements of the unknown data point.
4. 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 u**se 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.

3.1.3 Evaluation Metrics in Classification

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:

1. Jaccard index
2. F1-score
3. 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 $\hat{y}$ 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.

$$J(y,\hat{y}) = \frac{|y \cap \hat{y}|}{|y \cup \hat{y}|} = \frac{|y \cap \hat{y}|}{|y| + |\hat{y}| - |y \cap \hat{y}|} \tag{3.1}$$

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.

3.2 Decission Trees

3.2.1 Introduction to Decision Trees

Hello and welcome. In this video, we're going to introduce an examine decision trees. So let's get started. 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 build. 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.

3.2.2 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 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.

3.3 Logistic Regression

3.3.1 Introduction to 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 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.

1. 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.

2. 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.

3. 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.

4. 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.

3.3.2 Logistic Regression vs Linear Regression

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**, $\hat{y}$, 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 $\hat{y}$ 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.

3.3.3 Logistic Regression Training

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 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.

3.4 Support Vector Machine

3.4.1 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. Play video starting at ::13 and follow transcript0:13 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. Play video starting at ::59 and follow transcript0:59 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. Play video starting at :4:37 and follow transcript4:37 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? Play video starting at :4:58 and follow transcript4:58 Basically, SVMs are based on the idea of finding a hyperplane that best divides a data set into two classes as shown here. Play video starting at :5:9 and follow transcript5:09 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. Play video starting at :5:18 and follow transcript5:18 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. Play video starting at :6:12 and follow transcript6:12 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.

Chapter 4

Week 4

Chapter 5

Week 5

Chapter 6

Week 6