Understanding and Applying Logistic Regression

by Vitthal Srinivasan

Course Overview

Course Overview

My name is Vitthal Srinivasan. Welcome to my course, Understanding and Applying Logistic Regression. I am a cofounder of the startup of the Loonycorn and before this I worked at Google and studied at Stanford. What's the smart way to approach a deadline? Start an hour before the deadline? Good luck with that. A year in advance then? Well, that's probably overkill. The smart way is to start just late enough that you're still sure of making it in time and that is exactly what logistic regression can help you with. Estimating the odds of and how they change as actions do. Some of the major topics that we'll cover in this course include logistic regression and its applications, how logistic regression is linked to linear regression and machine learning, the S-curve and its standard mathematical form, and lastly, how to predict whether a stock will go up or down using three technologies: Excel, R, and Python. By the end of this course you will have a strong applied knowledge of logistic regression that will help you solve complex business problems. You'll know how to build robust regression models in any one of these three powerful tools: Excel, R, and Python. And you'll know how to interpret the S-curve. I hope you'll join me on this journey to learn how to play the odds and to quantify probabilities with this course, Understanding and Applying Logistic Regression here at Pluralsight.

Modeling Relationships Between Variables Using Regression

Playing the Odds with Logistic Regression

Hello and welcome to this course on Understanding and Applying Logistic Regression. In this module we will start by discussing the induction of using regression to identify and quantify relationships between variables. You may be familiar with the basic idea of linear regression, which is to predict effects from causes. Logistic regression is very similar. It focuses on predicting the probability of effects given causes. As we can already see from that, linear regression and logistic regression are quite similar, yet slightly different. Unlike linear regression, logistic regression can be used for categorical y-variables. This means the dependent variable in a logistic regression can take only one of a finite set of values. Gender, for instance or day of the week or month of the year. These are situations when logistic regression really comes into its own. It helps us focus the probabilities associated with different values of those categorical variables and that's why forecasting and classification are two of the most important applications of logistic regression. So that's the agenda for this module. Let's get started. First, a very quick wodd about prerequisites. Linear regression is all about predicting effects given causes. Logistic regression is all about predicting probabilities of effects from causes. As you can see, this means that linear and logistic regression are quite similar yet slightly different. It's not an absolute prerequisite for you to understand linear regression before you take this course, but it probably helps. In particular I have a course on Pluralsight on Understanding and Applying Linear Regression. You might find that material helpful to go through before you undertake this course. With that bit of context out of the way, let's plunge into this course in logistic regression. Let's understand how we can assess the odds, we can play the probabilities using logistic regression. Let's start with the subject of deadlines. Most folks have a love-hate relationship with them. Some folks like Douglas Adams love deadlines because they completely ignore them and then they love the swooshing sound as they fly past, but for most of us, this is not a luxury that we can affodd. And for many of us, the way we approach deadlines can fall into one of two extreme categories. Maybe we start just before a deadline, 5 minutes before a deadline, and of course we know from experience that this typically doesn't turn out well. On the other hand, we might be really conservative and start a really long time before a deadline and this once again is probably overkill. Most likely we'll end up

doing no other useful work for that entire year. And in this way neither approach is really optimal. If we start a whole year in advance of our deadline, we are sure we are certain to actually meet the deadline, but our probability of getting anything else useful done that year is basically 0. On the flip side, if we start 5 minutes before the deadline, we'll get a lot of useful stuff done in the weeks or the months leading up to the deadline, but then again our probability of meeting the deadline is 0, and we've got to be sure that we've can live with the consequences of missing that deadline. Clearly what we need is some kind of Goldilocks solution, which makes our pursuit of this deadline neither to hot nor too cold. We do not want to rely either only on working fast or only on working hard. We want to make sure to work smart and in this context working smart implies starting as late as possible such that we are still virtually certain to meet that deadline. We want to start at a point such that our probability of meeting that deadline is 95% and our probability of getting other useful work done in the area approaching the deadline is also about 95%. And this now is precisely where logistic regression comes into play. Let's say that we plotted on the X axis the time before the deadline and on the Y axis the probability of actually meeting that deadline. At one extreme we know that if we start just 5 minutes before the deadline we have a 0% probability of meeting it. This gives us one point in our plane. Somewhere at the other extreme way over on the right, we know that if we start 1 year before the deadline, we have 100% probability of making it. These two points represent the extremes. One of them is the work fast solution and the other is the work hard solution. What we are looking for is precisely a middle path. We are looking for a way to work smart. The question we are trying to answer is, how much in advance of the deadline do we need to start work such that we have a 95% probability of making the deadline and what we now need to do is to find the X value that corresponds to this working smart solution and really in odder to answer this question, we need a smooth curve which tells us how the probability of meeting the deadline is affected by when we decide to start. If we had such a smooth curve it would be easy for us to find the working smart solution. We would drop a horizontal line from the 95% level on the Y axis until it intersects this curve and then we would draw a perpendicular, that's a vertical line, down to the X axis and that value on the X axis would tell us exactly when we need to start in odder to have a 95% probability of meeting our deadline. Let's say in this example that that value turns out to be 11 days. Well that has really simplified our decision. In odder to work smart, all we need to do is start work 11 days in advance of the deadline. If we postpone, if we dilly dally for another week or so, then our probability of meeting the deadline is going to drop precipitously. In resource allocation decisions like this one, it's really useful to have a curve that relates the amount of resource, in this instance the time, to the probability of achieving our objective. And logistic regression is all about calculating exactly

such a probability curve. Logistic regression helps us find how probabilities are changed by our actions.

Working Smart with Logistic Regression

There are a whole range of values where our probability of meeting our objective is 0%. There are another whole range of values over on the right where we over-allocate resources and so our probability of meeting our objective is 100%, albeit at the cost of inefficient resource allocation. So we need a middle path and this is where the curve gets interesting. In the sharply changing, almost linear middle portion of the curve, the probability flips from less than 50% to more than 50% for very little change in the amount of resource we put in. This is something known as an S curve and as we shall see, this is actually a very useful model for understanding many other phenomena. Let's also pay attention to some of the little details of this problem setup. Notice how on the Y axis we have the probability of meeting our deadlines. On the X axis we have the time to the deadline. This is a continuous variable. The variable on the Y axis in contrast is binary because we either hit or miss our deadline; there is no in-between. Notice also that the probability curve has to flatten at the ends because by definition, the smallest value that a probability can take is 0 and the largest value that it can take is 1. It turns out that each of these features is actually representative of a logistic regression. Again, our Y variable is binary. It takes two values, 0 or 1. Our X variable, which represents the start time before the deadline is continuous. And our smooth curve estimates the probabilities that Y will be equal to 1, i. e., that we will hit our deadline. Here the Y variable is a binary categorical variable. This is an important distinction so we need to understand it. Categorical variables are tools which can take only a finite set of values, for instance gender or the day of the week or the month of the year. Categorical variables which take only two values are known as binary variables. In contrast, the X variable with the time before the deadline is a continuous variable. It can take any value from minus infinity to plus infinity. And remember again that logistic regression helps to estimate the probabilities of categorical variables. Logistic regression deals only with categorical Y variables or dependent variables. This is in contrast to linear regression where the Y variable must be continuous. This is an important difference between linear and logistic regression. As a result of this, logistic regression problems tend to have a typical setup. The objective is to find the probability, p of y, of some categorical variable, Y, given a set of causes, X. Take the example of hitting a deadline. We want to find the probability that we actually make it in time. The categorical variable is 1 or 0, depending on whether we hit or miss. And lastly, the cause here is the length of time we spend working on that assignment. In other wodds, we are seeking to estimate how the probabilities of making the

deadline are influenced by how much time we spend working on it. Let's take another example, that of survival probabilities in the sinking of the Titanic. We wish to estimate the probability of surviving. That is p of y. Y here takes the value 1 or 0; 1 represents a person show survived that shipwreck and we'd like to see how these probabilities are affected by a bunch of causative variables such as gender, age, and the type of ticket that the individual held. This is yet another typical use case for logistic regression. One more. Let's talk about trading the financial markets. We wish to predict the probability that the market will rise tomorrow so that's p of y. The categorical variable is 1 or 0, depending on whether tomorrow is an up day or a down day in the financial markets. And this is a probability that we'd likely find days on a whole bunch of causative variables such as interest rates, commodity prices, the state of the global economy. This is yet another application of logistic regression.

Applications of Logistic Regression - Analysis, Allocation

Now that we have a basic idea of what logistic regression is all about, let's study some of its common applications. Analyzing past events, examining what the causes and the effects were and why things turned out the way they did. Allocating resources. How much time, money, or engineering or management bandwidth to devote to a problem. Predicting future events. Making bets on how things are going to turn out. And lastly, classifying problem instances into categories. Spam email or legitimate email. Fraudulent transaction or legitimate transaction. All of these are scenarios where logistic regression can be used quite successfully. Let's quickly understand each of these application starting with the first. Let's see how logistic regression is used to analyze the consequences of past events. Typically we start with a famous past event, a case study. We dig down and identify some observed causes and we want to quantify how those observed causes led to the actual outcomes. Hindsight is 20/20, but even so, logistic regression will allow us to quantify the probabilities linking those outcomes and their causes. Take the sinking of the Titanic and survival probabilities. The subprime crisis from 2008-09. Or even something much more mundane, the track recodd at a famous software company with a history of time and cost overruns. Specific outcomes are associated with each of these events. For instance, in the case of the Titanic, there were more than 1500 deaths and only about 700 survivors. In the global financial crisis, several banks and hedge funds collapsed and went out of business. In the case of the large software service provider, we've had billions of dollars worth of cost overruns and lost productivity. The observed causes here also vary. From the case of the Titanic, studies have shown that a passenger's gender, age, and economic status were all predictors of survivor probabilities. The global financial crisis had complex causes such as rising interest rates, slowing

economic growth, and rising oil and commodity prices, which triggered asset bubbles. In the case of our software company, which is unable to deliver on time, perhaps the issues have to do with setting unrealistically low budgets to begin with. Perhaps it's a lack of leadership or poor management expertise, or maybe the engineers at this firm just aren't skilled; they just don't have the technical know-how to deliver on time and on budget. In each of these cases, the observed causes are complex, but the outcomes are relatively simple. In the Titanic, a passenger either survived or perished. During the global financial crisis, a bank or a hedge fund either made or lost money. In a given software project, either you ship on time or the schedule slips. It's pretty binary. In each one of these situations we could use logistic regression to compute a probability curve. Let's take the example of the Titanic shipwreck. Sex, age, and the passenger's economic status and class of ticket were all linked to the probability of survival. Gender for instance, predicted survival probabilities quite strongly because references were made while allocating seats aboard lifeboats. The type of ticket also had a strong link with survival probability, and age also had a role to play because like gender, it was a factor in determining preference on lifeboats. Children were given preference. And logistic regression can be used to combine all of these factors into one cohesive model. This is a model which will have as its output the survival probability of a given individual. And that is how logistic regression makes sense of phenomena like this. Only 3% of women with first class tickets perished while for men with second class tickets that same proportion was a staggering 92%. Let's move on to the other applications of logistic regression, notably resource allocation. We've already spoken about this at some length while talking about deadlines. Let's make this conversation a little more gentle now. A deadline is an economic opportunity. This might sound strange, but a deadline represents the chance to do something, which is of course what an opportunity is. Missing a deadline is in effect a catastrophic loss. The question that's facing us is how do we allocate resources in odder to avoid this catastrophic consequence of missing the deadline? And perhaps most crucially, how are the probabilities of the catastrophic loss changed by the amount of resource that we allocate? This goes back to the idea of a Goldilocks solution where the middle path is best and that's what working smart is all about. It helps us to avoid either one of two extremes. We neither want to allocate so little source that our probability of the catastrophic loss approaches 100%, but while attempting to hedge our downside, we do not want to allocate so much resource that we are able to get little else accomplished.

Applications of Logistic Regression - Prediction, Classification

We have discussed the use of logistic regression in analyzing past events and in allocating resources. Let's now talk about how it can be used in making predictions. The basic idea is very similar to that of using linear regression to make predictions. The big difference is now it's probabilities that we are predicting. Let's get back to our example of working smart. We wish to work in such a way that our probability of meeting our deadline is 95% and our probability of getting other important work done is also 95%. This is the whole point of working smart of course. In logistic regression we represent the relationship between the probability and how much time we spend working this project using an S curve. This S curve has a characteristic shape like the one you see on the screen now. Then the question that we want to answer is how hard do we have to work in odder to stand a 95% chance of meeting the deadline and it turns out that answering this question is actually quite simple once we have a nice S curve set up. As we've already seen, we can then use this S curve to predict how much time we need to allocate for this task. And this gives us something like a framework for using logistic regression for prediction. We are faced with a future event and a set of outcomes. We have likely causes and what we want are the probabilities. Investing in stocks, applying for a job at a prestigious company like Google. These are situations where we really want to know our chances of success before we even start. For instance, if we are going to trade in the markets, we make or lose money. If we apply to Google, are we likely to be hired or not? It's quite reasonable for us to not even try these activities unless we feel that we have a decent change of success. And those chances of success hinge on complex variables. In the case of stocks, on macroeconomics such as interest rates, global growth, and even on politics. While applying to a company like Google, determinants of our success include our interview preparedness, the quality of our resume, and also the overall hiring environment. These causes are complex, but once again, the likely effects are relatively simple. We really want to know the probability of whether our portfolio is going to be up or down. Are we going to be hired or not. That's the bottom line. That's what we really care about and it is in predicting such probabilities that logistic regression comes into its own. Let's now turn to the fourth and last common application of logistic regression, which is as a classification algorithm. Classification is a technique that's used in various applications such as fraud detection, for instance. Let's quickly see how we can use logistic regression in something like this. Let's say that our problem statement is to classify animals as either fish or mammals, whales for instance. One could argue that whales are mammals because they belong to a particular odder in the animal kingdom. And in fact, that is the definition of a mammal. But one might also argue that whales are fish because they look like fish, swim like fish, and move like fish. Logistic regression here is used as a fairly conventional classification algorithm resulting a machine learning based classifier. We'll come back to this and discuss it in a little more detail, but the basic idea is that logistic regression

looks at a large number of data points, that's the corpus and uses it to build a probability curve. This curve takes all of the characteristics of a particular animal and uses it to predict the probability that it's a fish. If that probability emerges as less than 50%, we classify it as a mammal. If it emerges as later than 50%, we classify it as a fish. And this is how logistic regression can be used as a binary classification technique. We will return to this in a little more detail ahead.

Logistic Regression and Linear Regression - Similarities

Logistic regression is very closely related to its more famous cousin, linear regression. As we shall see, we can actually make use of linear regression in solving the logistic regression problem. That's something that we get during the next module, but even apart from this linkage, there are a whole bunch of conceptual similarities and also subtle conceptual differences that we should talk about. Both linear regression and logistic regression deal with causes and effects. If X causes Y then X is a cause, also known as an independent variable. And Y is the effect, also known as the dependent variable. These terms, independent variable and cause or dependent variable and effect are important in the study of both logistic regression and linear regression. You should also be aware that it is quite common for the X variables to be called explanatory variables. But where linear and logistic regression diverge is in how they represent the relationships between the X and the Y. In linear regression, all of the points are represented in space and the linear relationship is established between them. Then the objective is to find the regression line in the equation y = A + Bx, which in some sense best represents this set of points and so we plot the cause or the X variable on the X axis and the effect or the Y variable on the Y axis. Pretty intuitive so far. But things are very different in logistic regression because here what we are actually plotting on the Y axis is the probability of Y and remember that in logistic regression, Y is a categorical variable which only takes one of a finite set of values. In this example on the screen now, all of the values of Y are either 1 or 0 and this make Y a binary categorical variable. Now on the Y axis we represent both Y itself, which is either 0 or 1 and also the probability of Y. And that probability is precisely what we seek to measure using the purple curve, which we've plotted on the screen now. This purple curve, which is an S curve, has a characteristic equation. This is the logistic regression equation and this is of course less intuitive and more intimidating than the simple straight line equation that we use with linear regression, but the basic idea is still the same and in fact, this equation is the characteristic equation of an S curve, about which we will have a lot more to say. Notice also that we are not going to be able to correctly predict all of the values of Y. That is why we have some values of Y, which sit very far away indeed from our purple S curve. Take for instance the point x3, y3. Our logistic regression equation predicts that y3 ought to be 0,

but in reality y3 is equal to 1. We want to find that S curve with those values of A and B such that these mis-classified points are as few as possible. So let's summarize the differences and similarities. Linear regression predicts the effect given a set of causes. Logistic regression predicts the probability of the effect given a set of causes. This also implies that the X and the Y axes have different meanings. The Y axis in logistic regression holds not only the values of the response variable Y, but also its probabilities. The effect variable or the response variable Y must be continuous in linear regression and it must be categorical in logistic regression. And remember the difference between continuous and categorical variables. Categorical variables are those which only hold one of a finite set of values, kind of like an enum in a programming language. Linear and logistic regression differ in the Y variable and its nature; however, in both cases the causative variables or the X variables can be either continuous or categorical. This is true with both linear and logistic regression.

Logistic Regression and Linear Regression - Differences

Moving now from the axes to the functional nature of the curves themselves, linear regression seeks to connect the dots using a straight line. Y is equal to A + Bx. Logistic regression seeks to connect the dots with an S curve, which has a more complicated exponential equation. As we shall see, S curves are actually very wildly relevant in nature and in business so they are worth studying as well. The equation of a straight line is pretty simple. The equation of an S curve is a little more complicated and it involves a ratio with an exponent in the denominator. But in both logistic and in linear regression, given a set of points, Xi, Yi, the objective of the regression is to find those values of A and B, which best fit the data. The terms best fit are in quotes because their mathematical significance varies, as we shall see in linear regression, a best fit is one which minimizes the variants of the residuals. In logistic regression it is one which maximizes the likelihood of the data. This gets into the math a little bit and so we shall hold off on this until the next module. Now you might notice that the exponent in the denominator of the S curve is actually a linear equation and this brings us to our next point. Linear relationships in linear regression are by assumption; however, we can also convert a logistic regression to a linear relationship using a log transformation. This linear relationship has on the left-hand side of the equals sign, the log of the odds of a particular event. This function is known as the logic function. And so logistic regression takes its name from the fact that the log of the odds of the probabilities are linearly related to the explanatory variables. Both linear regression and logistic regression can be solved by using fairly standard mathematical techniques. The method of least squares and maximum likelihood estimation. Another point of similarity between the two is that

they can both be easily extended to multiple dimensions. As we shall see, it's quite easy to extend the equation of an S curve by adding more explanatory variables and this gives us a way to easily assess how different variables or combinations of different variables change the probabilities of our outcomes. The math around the residuals in these two forms of regression are very different and let's see why. In linear regression we have a set of points y1 through Yn and we want to find these constants A and B, which perfectly satisfy the equations that you see on screen right now. Now in reality we will not be able to find such constants A and B, which impose perfect equality. We then go ahead and find the constancy and minimize the values of the errors e1 through en. Those errors which are known as the residuals more correctly represent the difference between the actual and the fitted values of the dependent variable y in the regression equations. And an important assumption behind linear regression is that these errors or residuals are normally distributed. Now as we shall see, it's actually not possible for the residuals of a logistical regression to be normally distributed at all. The logistic regression equation starts with an S curve where the probability of y is represented in terms of an exponent of the explanatory variables. We then go ahead and plot the probabilities of individual points, e of y1 to e of yn. Exactly as in linear regression we now want to find those values of A and B, which ideally set all of these equations to perfect equality. Once again, it will not be possible to find such constants A and B and so we will have to live with errors, but these errors are always going to take the value of either 0 or 1. The residuals will have the absolute value of 1 for all points which our logistic regression equation predicts incorrectly and they will have the value of 0 for all points which it predicts correctly. And as a result of this, the residuals of a logistic regression just cannot be normally distributed and this means that the math around linear regression and logistic regression is going to be quite different. We shall get to this in more detail in the module up ahead.

Logistic Regression and Machine Learning

Let's round out this module by understanding how logistic regression is an example of a machine-learning based technique. While talking about the applications of logistic regression we had ended by discussing its use in classification. Now classification is actually a pretty standard problem in machine learning and it also serves to beautifully illustrate the differences between machine learning and rule-based approaches. Whales. Are they fish or mammals? As we have already discussed, we might answer this based on where they sit in the animal kingdom's taxonomy, that would make them mammals, or we might be swayed by some of the characteristics; the fact that they move like fish, swim like fish, and look like fish. A rule-based approach is one which would take into account the definition, the taxonomy. After all a taxonomy

is a set of rules used for classification. Then a whale would be the import into this rule-based classifier and that rule-based classifier would've been drawn up from the collective wisdom of various human experts and the output from that classifier would be the tag mammal. In a rule-based approach, whales are mammals because they have been classified or defined as mammals. Now another way of making this classification might be to go out and observe a large number of animals, both fish and mammals. These would be our corpus. This corpus would then be passed into some kind of classification algorithm, which examines different attributes of each animal such as for instance whether it lives in the water or on land, attributes of each animal like that. And then based on this empirical study of the data, the output would be a classifier. This is exactly the approach that machine learning algorithms take and so a classifier obtained in such a way is an ML-based classifier and the scale and the judgment in building ML-based classifiers is in observing the correct attributes to focus on. In this example we need to tell our classifier that here is an animal, which breathes like a mammal and gives birth to live young. If we correctly focus on those attributes of the animal, our classifier will very likely classify a whale as a mammal and this really gets to the heart of the difference between rule-based and ML-based system. ML-based systems tend to be dynamic. They change their output based on the set of data that they look at. The rule-based systems in contrast tend to be static because they are really a set of rules prepared by human experts. From this first point of distinction we can see that experts are optional for an ML-based system, but they are absolutely essential for a good rule-based one. On the other hand, a corpus, a large number of training data points are important, very important in ML-based systems. Most machine learning-based systems will incorporate an explicit training step where an algorithm is fed in a corpus of data and outputs an actual classifier and this is how logistic regression is a perfect example of a machine learning-based approach to classification. We take in a corpus, that is a large number of data points along with their outcomes and we feed all of these data points into the logistic regression algorithm and the output is a classifier in the form of that S curve. Given a set of attributes, that's x, we use the constants A and B to find the probability of the outcome y. And once we have this predictor or classifier set up, we can now use it to find the probabilities for new animals which we have not classified in the past. For instance, we might plug in a feature vector, that's the technical term for it, saying that this is an animal which lives in the water, breathes with lungs rather than gills, and does not lay eggs and gives birth to live young instead. And our logistic regression-based predictor will output the probability that this animal is a mammal. Hopefully this probability will be greater than 50% and in a binary classification scenario that is enough for us to conclude that whales are indeed mammals. This also illustrates why the training step was so important. In our probability S curve, if we have animals which live on land, breathe with lungs, and give birth to live young, it's pretty obvious that

they are mammals. In our S curve, the probability that these animals are fish is only 5%. That's an easy problem instance to classify. It's also easy to classify animals that live in water, breathe with gills, and lay eggs as being fish. The tricky bit is in the transition between those two easy types of cases. The closer to the 50% boundary we get, the more tricky the classification problem. If the probability is less than 50%, then we classify this animal as a mammal. If it's more than 50%, we classify it as a fish. Clearly for animals such as whales which lie close to that midpoint boundary, this is going to be a hard problem to solve. In the modules up ahead we are going to implement binary classification for market moves. We are going to set up logistic regression equations to predict whether the markets will go up or down and our basic methodology will be exactly the same as that we discussed here. In the module coming up ahead we will get down to the mathematical nitty-gritty of logistic regression. We'll see that the logistic regression curve has a rather intimidating equation and solving this logistic regression problem requires us to find the best such values of A and B. A is the intercept and B is the regression coefficient. E in this equation on the screen now is a constant. This is the base of the natural logarithms and has the value of 2. 718. The equation that you seen on the screen now is actually a lot more interesting than it seems. This is known as an S curve and it's widely used in business. For instance, to model when innovations go viral. It has been found that the tipping point in the adoption of innovations tends to occur in around 16% of all users adopted and this model is also built atop an S curve. But before we get there, let's quickly summarize all that we learned in this module. We saw how a logistic regression is a powerful and versatile way to predict probabilities given causes. This makes logistic regression quite similar to linear regression; however, they are slightly different because linear regression has a different mathematical purity and assumptions. Importantly, logistic regression will only work for categorical y-variables. In contrast, linear regression will only work for continuous y-variables. Forecasting and classification are the most important applications of logistic regression.

Understanding Logistic Regression Models

The Intuition Behind Logistic Regression

Hello and welcome to this module on Understanding Logistic Regression Models. In this module we are going to go a little deeper into the mathematical intuition. We will introduce the concept

of the S-curve which is used in logistic regression in order to fit probabilities and underlying causes. We shall see how S-curves have standard mathematical forms which are easy to estimate. There are two equivalent ways of fitting S-curves that we will discuss and we shall see how one of these ways cleverly converts the logistic regression problem into linear regression problem. And lastly we shall also see how logistic regression can easily be extended to multinomial settings. These are situations where our categorical variable, the y variable, can take more than two values. So let's get started with a look at the intuition behind logistic regression. You probably have come across the term tipping point. This is an idea from sociology made popular by the famous writer, Malcolm Gladwell in his book of the same name. It refers to a point in time when a behavior or a practice suddenly becomes extremely widespread. This term is often best explained via the concept of the diffusion of a new innovation. Let's say that a new product or service hits the market. Its adoption rate over time follows and S-curve like that you see on the screen now. Its annotation and adoption as measured by the market share percentage that it demands follows this curve. Initially only a very small proportion of folks actually adopt it; these are the innovators. Because innovators form such a small percentage of the overall population, early adoption is very slow. It often seems like this innovation is getting no traction at all and this continues until penetration hits about 16%. Folks who adopted during this period are the early adopters and then all of a sudden something changes. The usage picks up and quickly rises to 50% and then to 100%. The crucial bit, the bit where this idea or innovation goes viral is where adoption approaches 16%. Apparently many studies in the fields of business and sociology have found that this 16% adoption threshold represents a tipping point. This is when a practice becomes widespread enough for it to actually go viral. How is this related to logistic regression, you ask? Well, because this exact same functional form is used in explaining logistic regression as well. S-curves are widely studied and well understood in math. S-curves follow the typical equation that you see on the screen now. And logistic regression extends this equation using it to model probabilities. This indeed is the whole point; this is the crux of logistic regression. The use of an S-curve for estimating probabilities. Let's see how we can use S-curves in logistic regression. The first step is to represent all of our points on a two-dimensional plane. Here on the y axis, we plot the probability of y and on the x axis we plot whatever explanatory variable we have in mind. For instance, on the y axis we might plot either 0 or 1, depending on whether we hit or missed a deadline. The corresponding x coordinates tell us how long we actually worked on this project. This is a familiar example to us now. Notice that all of the y values are 0 or 1 because these are for events where we already know the outcome and that outcome is obviously either 0 or 1. But now we can go to the next level. We can plot a curve which predicts what the y is going to be based on the value of x, and to do so we make use of an S-curve equation. If we pick specific values of A

and B, we can go ahead and connect the dots in our graph. This will give us a purple curve like the one on the screen now and the equation of this curve relates the probability of y to the corresponding value of x. And now the whole point of logistic regression is to find the best values of A and B such that in some sense there are as few mis-predicted points as possible. As we shall see, the beauty of the S-curve is that it lends itself very nicely to situations like our deadline probability experiment. If we represent the probability of hitting the deadline on the y axis and the time to the deadline on the x axis, if we assign too little time, our probabilities of making the deadline are almost 0 and this is not going to change even if we increase the length of time. In this portion we are definitely going to miss. At the other extreme, if we've assigned a whole lot of time, even if we assign a little less time, we are still certain to achieve the deadline. Both of these properties are correctly exhibited by an S-curve and the S-curve also has the nice property that for values of x which are neither too big or too small, the probabilities are changing very rapidly. A very natural question might be, why do we rely on this specific functional form while modeling probabilities? And that's a great question. In reality, any functionally form would work for us as long as it allowed us to model probabilities along the y axis and crucially if these probabilities depend on the values of the variable on the x axis; in this case the time to the deadline. But there are a couple of complicating factors. For instance, we cannot use a linear function as in linear regression because the variable, whether we meet or miss a deadline, is a binary categorical one. And the math behind linear functions and linear regression just does not work well with binary categorical variables. Another problem is that linear functions create y as a variable which can range from minus infinity to plus infinity. But because we are modeling a probability curve, we want this to flatten at the ends. WE want our function to have a floor of 0 and a ceiling of 1. These are not conditions which can be satisfied using a linear function and that is exactly why we make use of logistic regression instead of linear regression. This allows us to estimate the probabilities of categorical variables using underlying causes.

Understanding the S-curve

On screen is the equation of an S-curve. Let's go ahead and spend some time understanding this because it's important. In this yi is a binary categorical variable so it's either 1 if we hit or 0 if we miss. Xi is a continuous variable representing the time that we've spent working on the deadline. P(yi) is the probability of a hit, i. e., the probability that yi is equal to 1, and because this is a binary variable, 1 - p(yi) represents the probability of a miss. The next question then facing is where do the values of A and B come from? And the answer is, well by solving the logistic regression problem. Solving that problem involves finding the best fitting such S-curve. In this we need to

estimate the values of A and B. A is the intercept and B is the regression coefficient. And just in case you are not familiar with the constant e, e does not need to be estimated because it is the constant 2. 71828. E is an important constant similar to y and it represents the base of natural logarithms. You might wonder why the constants A and B are called the intercept and the regression coefficient. After all, these seem very similar to the terms used for them in linear regression. And that's a great observation. That's because a logistic regression can be part of a linear regression of logit function. We'll have more to say on this topic in just a little bit, but for now, let's keep exploring the link between linear and logistic regression. In linear regression we represent our data using a set of points, (xi, yi), and we then specify a functional form. A linear functional form. Y is equal to A + Bx. We then try and estimate the best values of A and B, which in a sense minimize the variances of the errors in this regression. We do something very similar indeed in logistic regression, except that here y is a categorical variable and it only takes the value 0 or 1. Once we've represented all of our (xi, yi) points, we then go ahead and fit the curve through them and as before, this curve has a specific functional form. That functional form is given by the S-curve equation we just examined. Like the linear regression curve, this S-curve is not going to perfectly fit all of our data points. Consider x3, y3 for instance. Our fitted curve predicts that for the value x3, the corresponding value of y3 should be 0; however, it's actually 1 and so our curve has gotten this point wrong. This once again is similar to the concept of residuals in linear regression. Our objective in solving the logistic regression equation is to find the constants A and B that maximize the likelihood of getting precisely the sequences of 1s and 0s that we observe in our data. This is something that we can do given the probabilities of a 1 and a 0 on each of these data points. The S-curve also satisfies our boundary conditions. For infinitely large or small values of x, the values of p(y) still will remain bounded between 0 and 1. For instance, let's say that the constants A and B are positive; at x equal to minus infinity the probability of y will be 0 and at x equal to plus infinity the probability of y is 1. On the other hand, if the constants A and B are negative then the curve flips above the y axis and so for infinitely large values of x, that is at x equal to plus infinity, the probability of y is 0. For infinitely small values of x, that is for x equal to minus infinity, the probability of y is 1. And in this way we have satisfied ourselves that at either extreme, the values of the probability of y are either 0 or 1. These are the minimum and the maximum values of the probability of y. The S-curve also works well in the important middle region. Here the probability of the outcome changes very quickly, with changes in the value of x. Take any arbitrary value of x called x1 and x2. Find the corresponding values of probabilities, p(y1) and p(y2). In the steeply rising middle portion of the curve, the probability of y2 is much greater than the probability of y1, even though x2 is only slightly greater than x1. The lesson from this is that in the sensitive middle portion of the curve the probability of

the outcome changes very rapidly for small changes in the x variable, and this is yet another test of how the S-curve closely matches our intuition and this is why logistic regression uses S-curves to estimate the probabilities of categorical variables as influenced by underlying causes.

Solving the Logistic Regression Problem via Maximum Likelihood Estimation (MLE)

Now that we've understood how S-curves can be used to model probabilities in logistic regression, let's turn our attention to actually solving that logistic regression problem. This consists of finding the best possible values of the constants A and B. In this section we are going to use a mathematical technique known as maximum likelihood estimation or MLE. We will actually implement MLE in Excel in the module after this one. MLE is a fairly standard cookie cutter mathematical technique. Do not be intimidated by the math behind this. It's actually quite simple once you get just right down to it. Let's understand it using a thought experiment. Let's say that we are going to toss n coins. The outcomes from each of these coins can be represented using the variables yi. I will range from 1 to n. Yi will be 1 if the corresponding coin tossed is a head. It'll be 0 if the result is a tail. The probability of getting a head on each coin toss is p subscript i. The corresponding probability of getting a tail is 1 minus p subscript i. So far, so good. There's almost nothing even slightly complicated so far. Let's now go ahead and tabulate all of these coin tosses and their results in a table. The first column is simply which coin we are tossing, the second is the result, the third is either 1 or 0, and the fourth is the probability of the corresponding result. Notice that the variables yi are only 1 or 0 because we already know the outcome of the corresponding coin toss. The probabilities, that's p1, p2, and so on; these refer to the probabilities from before the outcomes are known. So for instance, for coin number 1, the result was a head. The outcome is represented by the variable y1, that's 1. The probability of this is p1. For coin number 2 the result was a tail. There the corresponding result is represented by 0 and the probability of that before the coin toss was 1-p2. Now the question is, what is the likelihood of actually seeing results precisely like this and this is referred to as the likelihood of these results appearing in the data. Because of each these coin tosses is independent, the overall probability or likelihood is simply the product of the individual event's probabilities. Let's represent this likelihood with the variable L and L can then be written as the product of all of the values in the last column, even multiplied by 1-p2, multiplied by p3, and so on and so forth. For every coin toss which resulted in a head, we will have the corresponding probability pi. For every coin toss which resulted in a tail, the probability will be 1-pi. Now let's conveniently combine all of these probabilities into just one expression. Let's represent the outcome of coin number I using the

expression you see on the screen now, that is pi raised to yi, multiplied by 1-pi raised to 1-yi. All that we've done here is create one expression to represent either heads or tails. Instead of two outcomes, heads and tails, y1 will be 1 and 0 in these two cases. In the first case, if the outcome is a head, this formula evaluates to pi raised to 1, multiplied by 1-pi raised to 0, and that is simply pi. And in the same manner, if the outcome is a tail, this expression evaluates to 1-pi. So as you can see, all that we've done is to create one convenient expression to represent the outcome of coin toss i. Let's go ahead and plug this expression into our original likelihood function. The outcome of coin toss i is represented by the formula which we just derived and so the overall likelihood is represented by the product using the y notation. Y is the shorthand for the product of multiple terms. We are now almost done with our exploration of maximum likelihood estimation. Now that we have the likelihood, we can easily convert it from a product into a sum. Products tend to be a little difficult to maximize. We can easily transform this into a sum by taking the natural logarithm. That is the log to the base e of both the left and the right-hand sides of this equation. And doing so gives us the log of the likelihood represented by the variable LL, which is simply the sum of a number of terms. In order to follow this step you just need to remember one very simple rule about logarithms. The log of two terms is equal to the sum of their individual logs, and this really is it. The maximum likelihood estimation method involves finding the constants A and B that maximize this log likelihood. It's really as simple as that. Just remember, that the probability of every coin toss, p of yi, is represented by this S-curve equation that we've already discussed at length. So we simply want to find the values of A and B which best fit the data and which do so by maximizing the overall log likelihood. That's the formula that we studied a moment ago. This is a really simple optimization problem. In fact, it's so simple that we can even solve it right in Excel. In the next module we will see how to do so using a few clicks of the mouse and Excel's Solver add-in, which is meant exactly for such problems, and this is how maximum likelihood estimation or MLE is used to solve the logistic regression problem. It allows us to determine the constants A and B, which maximize the likelihood of actually getting a string of results like those that we encountered in our data.

Solving the Logistic Regression Problem via Linear Regression

Let's now take a look at an alternative way to solve the logistic regression problem, this time by making use of linear regression. This is something that we had touched upon very briefly while comparing linear and logistic regression. We had mentioned there that we can transform the logistic regression problem into a linear regression problem by a log transformation and in this linear regression setup, the dependent variable is the log of the odds. This is known as the logic

function so let's take a quick look at how this works in practice. Let's return to the thought experiment which we used so successfully while trying to understand MLE or maximum likelihood estimation. As before, if we had n data points, we model each of these data points as the toss of a coin. That gives us n coin tosses in total. We represent the result of each coin toss using the variable y subscript i. As before, each yi is either 1 or 0 and the corresponding probability from before the outcome of the coin toss is known is represented by p subscript i. This is the probability of getting a head and the corresponding probability of getting a tail is 1 minus pi. All of this is exactly as it was before when we were discussing maximum likelihood estimation. We go ahead and tabulate our data. As before we have a column for the result that's y subscript i, as well as a column for the corresponding probability. For instance, say we toss coin number 1 and the result is a head; this is represented using the variable yi=1 and the corresponding probability is p1. We toss coin number 2 and the result there is a tail and the corresponding probability is 1-p2. Now let's introduce the first slight change in this thought experiment. Let's recreate our table, but this time we introduce a column for the causitory variable xi. Now we have the column values x1, x2, and so on. And the last two columns, that is the values of x and the corresponding probabilities are linked to each other, making use of the S-curve equation from logistic regression. And this linkage actually gets really interesting if we relax the assumption that all of the values of x are unique. Let's say for instance that we've sorted by the x column and now let's zero in on only the first five rows of our data. Here there are three heads and two tails and there are five values of x1, that is x1 through x5. Now what happens if these values of xi are not unique at all, rather that they are all equal to each other? Then we can rewrite our data exactly as before. The only difference is that the columns for xi and the probability are now no longer independent. All of the values of xi are equal to x1 and all of the values of the probabilities are either p1 or 1-p1. Clearly for this to be the case, the value of p1 must be equal to 3/5 and the value of 1-p1 will be equal to 2/5. In effect we have collapsed all of these five rows into one single row and calculated the probabilities that fit the data. Now in reality if the x variable is continuous, we almost never will have exactly equal values of x, but we can always create ranges just as we would usually while setting up a frequency table and indeed that is exactly what we set out to do here. Create a frequency table with one row for each unique value of x. This now gives us a number of different values of y corresponding to each unique value of x and we include separate columns for the numbers of 1s and 0s, and using the number of 1s and 0s that correspond to any given value of x, we can calculate the probabilities of 0s and 1s, that is the probability of yi. So for instance, for the unique value x1 we have 3 heads and 2 tails. That gives us the probably of p(yi) as 3 upon 5 and the probability of 1 minus p(yi) as 2 upon 5. Why do we do it this way? Well, because we are about to make use of linear regression on the last two columns of this table and that linear

regression just works out a lot more robustly when there are a number of data points going into each value of probability. This is sometimes known as the rule of n because some folks believe that there should be at least n values of y corresponding to each unique value of x. This approach yields the best results when that's the case. In any case, let's now change track and go back to the definition of odds. This is something that's used a lot in some fields such as sports betting and even just in everyday speech. If you have an event, a binary event which can occur with probability p, the odds of that event are defined using the formula you see on the screen now. Let's now plug in the formula for p and 1-p from our logistic regression S-curve. Remember that p is simply this ratio that you see on the screen now. We can rearrange it mathematically so that there is an exponent now in the numerator as well. This is really just a bit of simple arithmetic. We then carry out another simple arithmetic operation and we find that 1-p evaluates to a very similar ratio. The values p and 1-p have the same denominators and only the enumerators defer, and so the ratio of these two terms gives us the odds of p as simply equal to e raised to A + Bx. And you probably have a hint of where this is going. We now transform this formula by taking natural logarithms of both sides of the equation. The log of the odds is simply known as the logit function and so we get that logit of p is equal to A + Bx. Again, remember that the logit function is simply the log of the odds of p. The logit is linear. The log of the odds of p, which can be rewritten as log of p minus log of 1 minus p, can be expressed as a linear function, A + Bx. All that we need to do is to carry out a linear regression and this will give us the best values of the constants A and B and this really is how we have cleverly transformed logistic regression into linear regression. We need to find the values of A and B such that all of these equations that you see on the screen now are satisfied with perfect equality and this is the linear regression problem that we are very familiar with. However, we will not be able to find values of A and B that perfectly satisfy these equations so we calculate the residuals epsilon 1 through epsilon n and we want to find the values of A and B, which make these residuals as small as possible. And doing so solves the logistic regression problem. That's how we have transformed our logistic regression where the values of y are a categorical variable with values of 0 and 1 into a linear regression where the y axis now contains the logit of p. We have satisfied ourselves that logistic regression can be solved via linear regression on the logit function and the logit function is simply the log of the odds function. Now that we have two ways of solving the logistic regression problem, a natural question is, how are they related? And it turns out that they are mathematically equivalent. Linear regression is usually solved via the method of least squares and this works out to be mathematically equivalent to the maximum likelihood estimation that we discussed previously. There is a third cookie cutter technique, the method of moments, which can yield different answers under some circumstances,

but that's not relevant to our conversation here. The two approaches that we saw, solving the logistic regression problem are mathematically equivalent to each other.

Binomial and Multinomial Logistic Regression

So far we have focused on binomial logistic regression. This works when our y variable is a categorical variable which takes only one of two values. However, there also situations where we have multinomial variables where there are more than two possible values for our effect variable. It turns out that we can really easily extend our logistic regression setup from binomial or binary to multinomial setups. Let's see how to do this by extending our binary classification problem. Recall that we had discussed classifying animals as either mammals or fish. There we constructed probability tables which assign probabilities of an animal being either a mammal or a fish. And in the binary setup we had a very simple way to calculate the probability of one from the other. Each probability was simply one minus the other so if the probability of an animal being a mammal was p, the probability of it's being a fish was simply 1-p. So if we needed to classify an animal as being either a mammal or a fish we simply needed to calculate one probability, that is we needed to run one logistic regression. If the probability that emerged was less than 50% that it was a fish, we simply classify it as a mammal. And if that probability emerged as more than 50%, well then we classify it as a fish. This worked just fine in the binary world where we had only two possibilities, mammal and fish. We calculate one probability and assign a label accordingly. Let's now move from a binary to a ternary word. Let's add in the possibility that our animal is a reptile. It turns out that this is not that much more complicated to do. We now set up a probability table from which we will seek to measure whether an animal is a mammal or not, then whether an animal is a fish or not, and lastly, whether that animal is a reptile or not. This requires us to run not one but rather three logistic regressions. Once we've carried out all of these logistic regressions we can calculate the corresponding probabilities and assign the label corresponding to the highest probability. So the first logistic regression would simply tell us whether this animal is a mammal or not. This would have the equation that you see on the screen now and we would need to determine the constants A and B. We would then run a separate but similar process to calculate the constants C and D, which we would use to calibrate another S-curve, telling us whether this animal was a fish or not. And lastly, we would go ahead and do the same for the reptilian probability curve. This time the constants are F and G. Now if your mathematical intuition is sharp, you might notice that this third logistic regression really ought not to be required because after all, if an animal is neither a fish nor a mammal, then maybe it's got to be a reptile. Well, it turns out that that is not always the case. This is only true if you've made some

assumptions about the underlying data. Those assumptions go beyond the scope of this course, but even so the technique that we outlined here is robust and a pretty good one. Given a number of options, you can set up multiple logistic regressions and then calibrate and choose that option which has the highest probability. So for instance, let's say we find that we have an animal which breathes with lungs, gives birth to live young, and lives in the water. We would plug these into our mammalian logistic regression and get a probability, let's say 55%. We then go ahead and do the same with the logistic regression curve or the S-curve that we have for fish. When we plug in these characteristics we're going to get a much lower probability. Let's assume that that is 40%, and we'll do the same with our reptilian logistic regression and we get an answer of 5%. By the way, it's just a coincidence that all of these probabilities sum up one, the exact relationship between these probabilities goes beyond the scope of this course, but in any case we now know what to do. The probability that this animal is a mammal is greater than the probability that it's either a fish or a reptile and so we classify this animal as a mammal. This gets us to the end of the theoretical half of this course. In the three modules up ahead we are going to implement logistic regression in three technologies: Excel, R, and Python and we will see how logistic regression can be used to predict and analyze stock returns. We will also compare linear and logistic regression in this context. That's up ahead, but first let's summarize all that we've just learned. In this module we saw how an S-curve can be used to fit a relationship between probabilities and underlying causes. S-curves are widespread applications in business and sociology. More importantly they also have a standard mathematical form that is easy to estimate and easy to work with. Then, given a set of points, the logistic regression problem becomes that of finding the correct parameters and fitting the S-curve. There are two equivalent ways of doing so. One of these very cleverly utilizes linear regression in logistic regression. And in this module we also saw how logistic regression can be extended to deal with multinomial situations, that is with categorical variables which can take on more than two values.

Implementing Logistic Regression Models in Excel

Setting up the Regression Problem in Excel

Hello and welcome to this module on Implementing Logistic Regression in Excel. The headline, the 10-second version of this module is that logistic regression is really simple to do in Excel. So

please don't be intimidated by it. Go ahead and use it wherever it meets your use case. Our use case in this module is going to be a financial one. We are going to use logistic regression to predict whether a stock will rise or fall. We will set up and solve this logistic regression in Excel using Solver which is an Excel add-in and along the way we will also contrast the solution to two other approaches, a rule-based approach and a linear regression-based approach. We will also extend logistic regression to the user of multiple x variables. This is very simple to do. Then we will move on and attempt a far more difficult task. We will try and predict future returns in a stock using information that is only available today. This of course is a very hard problem. If we were successfully able to solve it, we'd probably make a fortune trading in the markets, but in any case we shall see that logistic regression performs relatively well even in this difficult task. So let's get started. Let's outline a plan of attack to accomplish all of this. Our objective here is to predict Google stock. Will it be up or down? We will do so using data that we obtained from Yahoo Finance, which is a great data source for such experiments. Once we've got Google's data set up and converted to returns, we'll try a very simple rule-based approach. Our rule will be if in a given month the S&P 500 index is up, Google will be up as well. This is a really simple rule and it actually works very well indeed. This is not a particularly hard problem to solve by the way because if in a given month the S&P has gone up, it's very likely that most other stocks have gone up as well. Next we will take another stab at the same problem this time using linear regression. Here we will make use of Excel's forecast function which will take in all of our data points of Google and the S&P and use that to predict Google's stock return in a given month. We will then convert that predicted return into a binary variable, 1 for up, 0 for down. After these two preparatory attempts we will finally go ahead and implement logistic regression. Here we will make use of the Excel add-in Solver, which comes pre-packaged with Microsoft Excel. We will set up a simple optimization and use maximum likelihood estimation to find the values of A and B in the S-curve of the logistic regression. Next we will extend our logistic regression model to contain two explanatory variables. One of these will be returns on the S&P 500 in this month; the other will be Google's own stock return from last month. The point of this is to show just easy it is to extend logistic regression to multiple x variables and then we will move on to a much harder problem which is predicting Google's returns next month using information only available this month. This would be a great problem for us to solve. Had we been able to come up with an amazing solution to this we would get pretty rich, pretty fast. As we shall see, that is not what exactly happens, but even so logistic regression performs pretty well. So that's the plan of attack for this module. Let's plunge into a demo where we go ahead and implement the plan and we will implement logistic regression in Excel. Remember that the objective of our plan is to predict whether Google stock is going to be up or down and we are going to do so using data available

from Yahoo Finance. So we switch over to a web browser, go to Yahoo Finance and search for the ticker GOOG. We then scroll down. There's very little rocket science here, and down at the bottom of the page, it's possible for us to download all of Google's stock prices to a spreadsheet. We go ahead and click this link and save the resulting csv file, comma separated values file, to our local machine. We go ahead and do the same for the S&P 500 and we are now ready to explain changes in Google's stock price using changes in the S&P 500. We are making an important assumption here and we should be aware that we are doing so. We are implicitly assuming that changes in the S&P 500 are the cause of changes in the price of Google's stock. Logistic regression just like linear regression relies on this implicit assumption of causality. The similarity doesn't end there. With logistic regression just as with linear regression, we should never regress non-stationary data. We will not go into the mathematics of what exactly non-stationary data is, but you should know that anytime you have variables with smoothly trending curves such as those that we see on the screen now, it is not correct. It's not statistically valid to regress one on the other. Usually we get around this in one of two ways. We either transform our data using log differences or by using returns. In financial applications like this one, it's more common to take returns. So we convert prices into returns. As you can see, returns are simply the percentage changes of the prices of an underlying asset. So that's what we are going to set up. Our problem is going to use the returns of Google stock as the y variable and returns on the S&P 500 as the x variable. Note that so far we have yet to convert Google's returns into a categorical up or down variable. We'll do that in just a moment, but before that, let's convert from prices to returns. And one little note of caution, before converting prices to returns we need to sort all of our data points from oldest to newest. So let's very quickly go ahead and do all of that. We open our data file for the S&P 500 and then do the same for our data file with Google's prices. Both of these files have the same format. The last column is the adjusted close. We copy over the adjusted close columns into one spreadsheet and format all of those nicely. We get three columns with dates and the prices of Google and the S&P. It's now a simple matter for us to convert these prices to returns. All we've got to do is to calculate the percent it changes between adjacent data points. So we go ahead and do this for all of our data points on both Google and the S&P and we have returns of both of these financial assets ready for us to work with.

A Rule-based and Regression-based Approach

Now that we've converted prices into returns, we have successfully executed the first step of our plan. The next step is to try a simple rule-based approach. Remember that a rule-based approach is one where human experts on the basis of their judgment and knowledge of a particular area,

put together prediction rules. Here the rule that we are going to try and use is a very simple one. If the S&P 500 is up in a given month, then we predict that Google will be up as well. And conversely, if the S&P 500 index is down in a given month, our rule states that Google will be down as well. This is really a pretty simple rule, but as we shall see, it's a surprisingly effective one. Keep in mind here that we are using the S&P in a given month to predict Google in the same month and really what our rule is doing is converting the S&P 500 returns into a binary categorical variable. If the S&P's returns are greater than 0, we assign the label of 1. If the S&P's returns are less than 0, we assign the label of 0. So for instance if the S&P in a given month was 1, Google will also be predicted to be a 1, implying that Google's returns will also be greater than 0. In this way our simple rule-based system is going to output either a 1 or a 0 and once that's done we can test the accuracy of this rule. All we've got to do is to compare our predicted labels to the actual labels. This gives us a percentage measure of how often our rule has been right. So let's switch over to Excel and implement all of this. We pick up back where we left off, which was with the returns of Google and the S&P 500. We then make use of Excel's if statement to convert Google's returns into a categorical variable. If those returns are negative, we assign the categorical variable the value of 0. If those returns are positive, we assign the value of 1. This gives us a value for a categorical variable up. We also define the complimentary categorical variable down in terms of up. Down is simply 1 minus up. We go ahead and calculate these values for all of the data points of Google return that we have. We format those columns nicely, and we then go ahead and calculate the odds. Remember that the odds of any event with probability p are defined using the formula on the screen now. P divided by 1 minus p. Now in logistic regression we use an S-curve to represent the probability of an event. Just a very little bit of arithmetic manipulation allows us to calculate the value of 1-p as well. Both of these quantities have the same denominator. They only differ in their enumerators. So when we divide one by the other in order to calculate the odds, we get the value e raised to A + Bx. That is the formula for the odds in a logistic regression. We go ahead and calculate those odds and notice here that we often have the denominator as 0 and in those cases Excel gives us a division by 0 error. And this shows us why we need a frequency table if we are planning to use linear regression to estimate our logistic regression. And so we are just going to use maximum likelihood estimation using Excel Solver and that works just fine even if we don't have the odds, which means that this division by 0 isn't anything for us to worry about here. Let's now move ahead with implementing our simple rule-based approach. We calculate whether the S&P was up or down using Excel's if statement and we then go ahead and assign that label which we got for the S&P to Google and that becomes our prediction. So for instance, if in a given month the S&P was up, that means that our prediction is that Google will be up as well. This is our simple rule-based approach at work. Next we check

whether our prediction was correct. We simply compare the assigned label that was our prediction to the actual value of the Google up variable. We then go ahead and check the overall accuracy by counting all of those instances where the assigned label was correct, and doing so tells us that the accuracy of our simple rule-based approach was 74%. That seems pretty impressive. There are a couple of caveats to keep in mind. The first is that in a binary prediction problem, even a random choice will have an accuracy of 50%. The second is that we are using S&P's contemporaneous returns; that is we're using S&P's returns in a given month to explain Google's returns in that month. Now that we've tried one rule-based approach, let's also really quickly try another ML-based approach, this time using linear regression. This is really easy to do in Excel using the forecast function. The intuition behind this is that returns on the S&P and returns on Google are represented in a two-dimensional plane. The cause variable, that is returns on the S&P, are on the x axis. The effect variable, that's returns on Google stock, are on the y axis. And then we can use the fitted regression line in order to forecast what the value of Google's return will be given a new value of the S&P. Here we make use of Excel's forecast function. The output y will be predicted return on Google stock for the current month. The first import will be returns on the S&P 500 for the current month and then the training data will be passed in in the form of a known y's and the known x's, corresponding to all of the data available for Google stock and for the S&P 500 respectively. We then go ahead and convert this predicted value into a binary categorical variable. If it's greater than 0 we assign this the label 1 and if it's less than 0 we assign it the label 0, corresponding to up and down respectively. And then as before, we go ahead and calculate our accuracy by comparing the actual values of Google's stock return to our predicted values. Remember here that we are comparing two binary categorical variables which only take the values 0 and 1. Let's switch back to Excel and go ahead and implement all of this. We use Excel's forecast function. The first argument we specify is the corresponding value of the S&P from that month. We specify the values of the known y's and the known x's and the output is the predicted return of 3. 29% in this instance. And this implies that the linear regression approach is predicting that in this particular month, that's February 2005, Google is going to go up by 3. 29%. We repeat this calculation using all of the data available to us and then we go ahead and convert this into a prediction. This is our binary categorical variable. We check if the prediction is greater than 0. If yes, we assign the label 1. We carry out this conversion for all of the data in our dataset and then we go ahead and compare all of our predictions to the actual moves in Google in those respective months and when we compare the accuracy of the regression-based model to the rule-based model, we find that the regression model is only at 70%. This means that linear regression has actually performed quite a bit worse than our very simple rule-based approach in this problem.

Logistic Regression Using Excel Solver

We have now successfully applied two different techniques to our binary classification problem. We first tried a rule-based approach and then a linear regression-based approach. Let's finally get to the crux of this module and implement the same algorithm this time using logistic regression. Recall that logistic regression fits an S-curve between causes and the probabilities of effects. The idea behind logistic regression is to fit the curve, an S-curve specifically, which links the probability p(y) to an exponential function of the underlying cause x. In our example here, p(y) is equal to the probability of Google going up in the current month, i, and x here is the return on the S&P 500 equity index in the same month, i. The way we go about solving this logistic regression problem is by maximizing a certain quantity known as the log likelihood. If we run a simple optimization problem, which maximizes the value of this quantity, log likelihood, we will effectively find the values of A and B which best fit our data. This is exactly what we are going to proceed to do using Solver in Excel. This optimization problem does require us to provide initial guesses for the values of A and B and we will simply set those initial guesses to be 0 for both A and B. Before we actually get to the optimization using Solver, let's first calculate the probabilities and the log likelihood function. We start with the predicted probability for each data point. This we obtain by plugging in the corresponding values of the x variable which is the S&P return in a given month, into the formula for the S-curve. Notice how we are using the initial estimates of 0 for both A and B. And this gives us an initial probability estimate of 0. 5. This is telling us that if we created a logistic regression curve with both A and B equal to 0, the output of our regression curve would simply be a coin toss, a probability of 50% of a move in either direction. And of course we'll do a lot better than this once we run Solver to optimize those values of A and B. First let's calculate these probabilities for all of the data points that we have. And then also calculate the log likelihood of actually encountering this data point given the values of A and B. Doing so gives us the log likelihood of exactly one data point. Let's go ahead and repeat this calculation for all of the data points that we have and then calculate the sum of all of these log likelihoods and this sum is exactly what we need to maximize in our optimization problem. As we can see, the initial value of this log likelihood function is -100. 5. This represents the sum of the log likelihoods corresponding to our initial guesses of A equal to 0 and B equal to 0 in the logistic regression S-curve. Let's quickly retake stock of where we stand. We have calculated the log likelihood using our data points. Within this log likelihood we have made use of the probability of each data point. That probability is calculated using the S-curve of a logistic regression and what we now want to do is to find the best such S-curve, i. e., to calculate the best possible values of A and B, which maximize this log likelihood. This is a simple optimization problem, one that we can carry out in

Excel using the Solver add-in. Solver is a simple and extremely powerful Excel add-in that comes prepackaged with Excel. The way to access it is to go into Excel Options, select Add-ins, and ensure that the Solver add-in is enabled. The basic idea of how Solver works is that it has a target cell and this is you like to maximize or minimize. It allows you to specify a function of that target such as max, min, or set to a specific value. Solver will also prompt you for which cells you want to change and we can also specify constraints. Solver does more than just use brute-force trial and error. It actually under the hood uses some fairly advanced optimization algorithms. For instance, for linear problems it makes use of something known as the simplex method; for nonlinear problems it makes use of the generalized reduced gradient method and it even has support for integer and binary constraints. Solver is a really cool bit of functionality that comes along with Excel and I'd heartily encourage you to learn it and use it often. In our example here we simply point the target cell to the cell containing the sum of our log likelihoods. The function that we choose is maximize. The cells to be changed are the ones containing the values of A and B in the logistic regression equation. And we don't really have any constraints. And that's all we need to do. Solver will calculate and spit out the values of A and B, which maximize the log likelihood function. And that in turn will lead Excel to automatically recalculate all of the probabilities using the new values of A and B. So let's switch back over to Excel and implement all of this. We go into Excel Options. We click on that button over there. This brings up a menu box where we select the Add-ins tab. Within that we go ahead and click on the Solver Add-in. You've got to make sure that it installs and is available for use so we click on Go. When we do so, another option box opens up and we make sure that Solver is checked in this particular menu box. We hit OK and now Solver will show up as a button in our Excel, ready for use; it's under the Data tab. Let's click on Data, Solver, and that brings up the Solver dialog. We now specify the target cell to be the one containing the sum of our log likelihoods. The cells to be changed contain the values of the parameters A and B. We have no constraints so we can go ahead and hit Solve. Solver does it thing, it does its magic, and informs us that it has achieved a solution. It has substantially maximized the value from -100 to -77. The new probabilities are now recalculated automatically. This is in column U. We can make use of these probabilities, convert our prediction into a binary one. If the probability is greater than 50%, our prediction is a 1. If the probability is less than 50%, our prediction is a 0. We go ahead and recalculate this categorical variable for all of our data points and we then go ahead and test our accuracy and when we do so, we find that the accuracy of our first logistic regression is 72%, better than the linear regression-based approach, but it's a little bit worse than the simple rule-based approach at 74%. Even so we have successfully run our first logistic regression using Solver in Excel.

Solving with Multiple X Variables

We are very close to the end of our journey implementing logistic regression in Excel. Before we wrap up, let's just do a couple of more interesting operations. First up, let's see how easy it is to add multiple x variables into our logistic regression and this is an area where regression-based models perform much better than rule-based models. In a rule-based approach, as the number of explanatory variables increases, the level of expertise, the level of skill on a human expert's part required to formulate rules increases by orders of magnitude because of the complex inter-relationships between these different explanatory variables. In regression-based models, however, the cookie cutter mathematical techniques already do a whole bunch of the heavy lifting for you. In this example, we start with a simple logistic regression model which seeks to explain the probability of Google going up or down in a given month on the basis of moves in the S&P. Now let's say that we wanted to make this model quite a bit more complex. Let's say that we wanted to use Google's returns from last month to predict Google's returns this month. As an aside, such models in statistics are known as autoregressive models. If we wanted to do something like this in logistic regression, it's actually really very simple to implement. We just add one additional term in the exponent as you can see on the screen now. Nothing else in our entire procedure changes. Excel Solver will now carry out an optimization to find the values of three constants and everything will just work right out of the box. Adding x variables was really pretty simple. Let's move to a much harder problem. Let's see if it can predict Google's stock returns next month using data that's only available this month. Of course, this is a financial trading problem. If you were able to come up with a great solution to this problem, you could make a lot of money in the markets. Let's see how well logistic regression does here. All we've got to do is to slightly change our logistic regression equation so that the Google returns that we use are now lagged by one period. So if we are attempting to predict Google's stock performance in the month i, our predictor ought to be Google stock returns from the month i-1. The same holds true for the other predictor. For the S&P we are only going to use S&P returns from the previous month as well. Let's go ahead and try and implement both of these sets of changes in Excel. Let's start by making a copy of our existing worksheet. This worksheet which we call SamePeriod contains the results of in-sample predictions and these have accuracy in the range of 70%+. Let's make a copy. This worksheet is now called NextPeriod where the first thing that we'll do is to update all of our formulae so that we now use data from the previous period to predict Google's returns in this period. We start by modifying our rule-based approach and we see right away that the accuracy of this approach dips from 74% in the same period to only 55% for the next period. Clearly we are now working on a much more difficult problem. Remember again that in binary classification

problems, coin toss would give you accuracy of 50%. So 55% is really not very good. Next up, let's go ahead and update our linear regression-based approach. Once again, all that we've got to do is to correctly line up our values of the known x's and the known y's. We'll do so quite carefully to make sure that we don't use any information available in the future and when we do this, the accuracy of the linear regression-based approach also drops from 70% to 60% and as an aside, you can see that as the problem gets harder, the more difficult it is for a rule-based approach to keep pace with an ML-based approach like regression. And let's now extend our logistic regression-based approach to take into account two explanatory variables. So we first have to go ahead and create additional cells corresponding to the three constants that we now need to calibrate rather than two. We also have to go ahead and change each of the probability family in accordance with the changed logistic regression S-curve. From here on in it's pretty simple. We just re-run Solver, which does its thing and comes back with values of the three constants, giving us a in-sample accuracy of 60%. This is exactly the same level of accuracy as we got with linear regression. This is quite reassuring. And in the top right we can also find the values of the parameters calculated by Solver. A is 0. 2593 and the two B coefficients are 9. 24 and 0. 36 respectively. We have successfully used logistic regression to create a meaningful attempt at a very hard problem, that of predicting whether Google's stock is going to go up or down next month using information that's only available this month. As we've already discussed, this is a difficult problem to solve and evidence of that is that the r squared of n by regressing Google's returns on S&P returns from the previous month stand at just 1. 3%. The RSQ is a measure of how closely linked a couple of variables are such a low R squared of just about 1. 5% tells us that there is very little linkage indeed. Logistic regression has done a fine job of extracting what information linkage does exist. That gets us to the end of this module in which we implemented logistic regression in Excel using Solver. In the modules coming up next we will re-implement the same solution using two other technologies, R and Python, but before we get there, let's summarize all that we've learned in this module. The one-sentence summary is that logistic regression can be implemented really easily in Excel using Solver. We applied logistic regression to explaining Google stock returns and we got similar results using rule-based approaches and linear regression. We saw, however, that as the problem gets more complex, the regression-based approaches tend to do better, and in addition to this, logistic regression is also really easy to extend to multiple explanatory variables. This is yet another advantage of using a regression-based approach such as logistic regression or other rule-based approach.

Implementing Logistic Regression Models in R

R for Regression: The Right Tool for the Job

Hello and welcome to this module on Implementing Logistic Regression in R. The story line of this module is going to be pretty similar to the previous one in Excel. Our agenda once again will be to set up a logistic regression to predict whether a specific stock will rise or fall. This time our choice of technology will be R and as before, we will build a model with multiple explanatory variables and examine its characteristics in R. Before we plunge into the code itself, let's start by understanding when one would pick R versus C, Excel, or Python. Along the axis of ease of prototyping, Excel is the best tool out there. It's the fastest prototyping tool and in many ways the most convenient choice for a one-off logistic regression. The Solver add-in provides fairly robust optimization capabilities. Combined with Excel's other advantages, this makes Excel a pretty plausible choice for trying logistic regression. But unfortunately, just as Excel is an outlier in a good way in terms of ease of prototyping, it's an outlier in a not-so-good way in terms of robustness and reuse. There are no free lunches out there. As we saw while building our logistic regression models in Excel, when we wanted to slightly modify our code so that it worked on the next period rather than on the same period, we basically had to copy paste our worksheet. Copy pasting code is a terrible code reuse mechanism as everyone knows. And this is where tools such as R and Python step in. They in some ways occupy the sweet spot in terms of both robustness and ease of prototyping. Make the common use case easy and the difficult use case possible. This is a great design principle and it lies beneath the structure of Excel, R, Python, and in fact most very successful technologies out there. So in terms of understanding when to use which technology, it makes sense for us to understand the common use cases for Excel, R, and Python respectively. Let's say that you have an important presentation coming up to your boss and several other senior stakeholders. There that is a point that you wish to make and you'd like to include a regression analysis to support your point. That is a perfect use case for the use of Excel. You would carry out your regression in Excel, create an esthetically pleasing chart, and copy/paste it into your PowerPoint presentation. R on the other hand is more something that you would use maybe in an academic setting where you have a case study, you wish to examine a bunch of data using many different models, and most importantly you want to be able to do so right there in front of your audience. If you would like to quickly and efficiently try out a whole bunch of statistical modeling techniques and examine their properties and their pros and cons

and all of this in real-time, then R is probably your tool of choice. And what about Python then? Well, let's say that you wish to build a trading model where you're going to scrape data from a website using some library such as A Beautiful Soup and then also access data from some other databases or underlying data sources. And finally, carry out a whole bunch of ML-based transformations on these and then actually trade in the financial markets using this bit of software, then Python is your choice. In this way Excel, R, and Python all make their common use cases easy and all keep the difficult use cases possible, but the commonest use case for Excel is data supporting presentations, for R it's perhaps academic seminars, and for Python it's mathematically, but still production-ready code. Now all of this true and gentle, but as far as regression goes, I'm going to suggest that you use R, whatever your use case. For regression, R is just so handy whether you plan to make presentations, present seminars, or actually build production-ready trading models. Use R for regression, whether that's logistic regression or linear regression, it makes sense whatever your use case. With that sales pitch out of the way, let's now get down to the nitty-gritty. Let's go ahead and implement logistic regression in R and we will do so starting with just the installation of the R software. R is extremely simple to download and install both on Windows and on Unix-like machines. You also can use a tool known as R Studio, which is something like an IDE for R. Let's really quickly run through that installation for Windows. As you can see, I start by Googling r for windows. I click on the very first link that shows up. This leads us to the appropriate page on the cran. r-project. org website. I go ahead and download the appropriate executable, double click, and run that executable file. This installs R on my machine and places a shortcut to the R GUI on my desktop. We go ahead and double click on that desktop icon and that opens up that R GUI and we are in business. Let's go ahead and maximize the console window and get started by defining a variable. As you can see, it's extremely simple to get up and running with R and hopefully this course will have given you enough of a flavor of R to peak your interest in this topic. There are a host of excellent courses on Pluralsight on this topic and these are a great starting point for your exploration of the capabilities of R.

Sorting Dates

Now that we've understood the pros and cons of R and now that we've got R installed and set up, let's now get to the business at hand of implementing logistic regression in R. We had left off by defining a variable. This variable contained the name of a CSV file with all of the price data from Yahoo Finance. That was for the stock pick of GOOG. Remember that our objective is to explain changes in the price of Google stock, specifically whether Google stock goes up or down, using the S&P 500 equity index as our explanatory variable. Here the raw data is available in the form of

a CSV file that we had downloaded from Yahoo Finance and R has excellent support for using data in such tabular format from CSV files. This is a data abstraction known as a data frame. Data frames are maybe the most widely used of R's many data abstractions. In a data frame, each row represents one observation and each column of the data frame represents one particular variable. The columns can have different types and importantly they also have names. The data frame abstraction is similar not only to a flat file such as a CSV, but also to the relational model which is commonly used in databases. And what's more, R has a really handy way to go from a file to a data frame. That is a really handy method called read. table. Read. table, when invoked with the name of a file along with some other information, takes in all of the data in that file and stores it in a data frame. The header of the file, that's the first row, is used in order to specify the names of the data frame columns. Let's use this functionality on both Google and the S&P. We invoke the read. table method passing in the name of Google file. Notice how we specify that the file does contain a header, that's using the header = TRUE, and how the separator between the fields is a comma. There's also one little bit here worth paying attention to. By default when we use read. table, all of the columns in the underlying data file are read into our data frame. In this instance, we really only care about two of those columns, the date and the adjusted close. And that's why we select those using that little bit of syntax at the end. We can use the head command in order to peek at the top few rows of a data frame. We do so and we find that we indeed have what we expect, the date and the adjusted closes. We can also examine the names of the columns in a data frame using the names method. Here we are actually going to change the name of the second column from adjusted close to the more descriptive goog. price. We now go ahead and repeat all of these steps for the S&P as well. So we read in the contents of the S&P file into a data frame and we rename its second column to the more descriptive sp. price. Next we merge our two data frames. This is similar to a relational database join. Here we are merging the data frames goog and sp500 and the join column is named Date. And the results of this join are stored in the data frame goog. Let's examine what this data frame contains using the head command and we can see that the join has indeed succeeded. R has correctly lined up the dates along with the goog. price and the sp. price columns. But this invocation of the head command also tells us that there's something that we need to fix. As you can see, our goog data frame has its dates ordered as if they were strings. This lexicographical ordering is not appropriate for data and we've got to fix that. So we first make sure that R interprets our date column as dates and we do so using the as. Date command. Notice how in the as. Date command we specify a date format. This is the format that will be used by R to convert each of the strings in that column to a corresponding date object. We go ahead and use the head command to check whether our dates are now interpreted correctly and indeed they are. Notice, however, that they are still not sorted in the

order that we would like. That is something that we've got to explicitly do ourselves using the order function and that's exactly what we'll do. Notice that we also specify that we want the dates sorted in decreasing order; that is by specifying the decreasing = TRUE. We use the head command to now examine the contents of our data frame and everything looks fine. We indeed have the most recent dates appearing up top. We are now ready to move the most complicated part of this demo and surprisingly this is not the logistic regression itself; rather it is the conversion of these prices into returns.

Converting Prices to Returns

We've gotten R to recognize our dates as dates and we've also sorted our dates so that we go from most recent to least recent. This sets the stage for the only remotely complex bit of this little demo, which is converting prices into returns. We learned why this was necessary in a previous module, implementing logistic regression models in Excel. The basic idea is that we never want to regress non-stationary data and so we convert non-stationary or smoothly trending data into returns. These are basically percentage changes of adjacent points. All it takes to do so in R is the one line of code that you see on the screen now. This, however, is a slightly cryptic-looking line so let's understand it; let's break it down. The best way to start is by understanding negative indices in R. Negative indices are a way to exclude data. So for instance, if we have a data frame, indexing it with a pair of negative numbers will exclude the corresponding rules and columns. So the -nrow(goog) is going to have the effect of excluding the last row of data and by the way, remember that data frames are indexed starting from 1. There is also the other negative index -1 and this has the effect of excluding the first column. We exclude the first column of course because it consists of dates and we do not want to be taking percentage changes of our date column. Now that we understand negative indexing, we can make sense of an expression like the one on the screen now. Effectively we have the ratio of two data frames and by the way, this ratio is calculated element wise provided those two data frames have the same dimensions and these two data frames have the same dimensions and these two data frames have been specifically constructed to that they represent all of the points in our original data frame offset by 1 and to go from this expression to returns, all we need to do is to subtract 1 from this ratio. This is exactly what we do. This is what you see on the screen now. This was the expression on the right of the equality operator. We assign the returned value into a data frame, which is on the left. Let's go ahead and examine this data using the head and tail commands. We use head and we can see that indeed the prices of the Google and S&P stocks have been converted to returns. Everything there looks fine, but when we run the tail command, we see that the last row in our data frame

has a problem. This is because the last row still contains the prices rather than the returns. This happens any time we go from prices to returns. We've got to be careful to handle this last row correctly. And in this instance we do so by simply excluding that last row using a negative index and that now gives us the data that we require in exactly the format that we want. We have lost one row of data. The earliest point in our data set now corresponds to February 2005 rather than to January 2005 as it did before this. We go ahead and rename these columns of our data frame so that they now reflect the fact that we have returns rather than prices and we are now ready to turn to the business end, to the logistic regression end of this demo. Recall that logistic regression uses an S-curve with an equation like the one on the screen now to predict the probabilities of some categorical variable. Here the categorical variable is the probability that Google will have an up month in the current month, i. e., that Google stock will go up. We are seeking to predict this using returns on the S&P 500 equity index. This is a rather bare-bones logistic regression model and we can easily extend it to take into account multiple explanatory variables, for instance, the values of Google's returns from the last month. As an aside, models where the x variable is a lagged version of the y variable are known as autoregressive models. This is a very simple autoregressive model that you see on the screen now. It's still trying to solve a relatively easy question; what will happen this month to Google stock given that the S&P has gone up this month and that Google went up or down last month? A really much harder problem is one where we try to predict Google's stock return in the current month using data that was only available at the end of last month. Models like these are really difficult to solve, and hedge funds spend a lot of time and money attempting to do so. And this is the model that we are actually going to try and build in this demo. Of course, this requires us to include in our explanatory variables the lagged versions of Google and the S&P and so we'll do exactly that. We create a new data frame called combined, which has all of the data in our original data frame called goog and it also has the lagged versions of the S&P and Google returns. As you can see, we do this by excluding first and last rows from our original data frame. The result of this is to create the combined data frame, which once again has one less row than the original. We'll use the head command to make sure that all of these values look legit to us and we then go ahead and rename the last two columns to make it clear that they are merely lagged versions of the previous two columns. We now have in one data frame Google's returns, the S&P returns, and then two data cds once again lagged by 1. We are now ready to actually implement the logistic regression and as we shall see, this is actually incredibly easy to do in R.

Implementing Logistic Regression with GLM

We now have in one data frame all of the returns that we require for our logistic regression, but we still require a categorical y variable. Remember after all that logistic regression will only work with categorical y variables and so we convert our Google return CDs into a binary categorical variable. We assign to each data point the label 1 if Google was up in that month, and 0 if Google was down. Let's go ahead and add this as a column to our combined data frame and we call this goog. UP. This is a Boolean variable which contains 1 or 0, depending on whether Google rose or fail in the corresponding month. As usual, we use the head command to make sure that this operation went as planned and we see that we now have a column called goog. UP, which is true for months where Google's return is positive and false for months where it is negative. And now all that's left for us to do is to invoke R's GLM or generalized linear model function to create a logistic regression model. This calibrates the values of the constants in the equation you see on screen now. GLM is virtually identical in its usage to LM or linear model, which we use for ordinary regression. Here the only real difference is that we also specify a second parameter, the family=binomial. This is our way of telling R that we would like R to fit a generalized linear model, specifically a logistic regression model using GLM, and as usual, the exact functional form of that model is the first parameter into GLM. You see that we have as the y variable the goog. UP categorical variable and as the explanatory variables, we have Google's returns and the S&P returns both lagged by 1. Unlike in Excel where we actually had to carry out the maximum likelihood estimation ourselves using Solver, here all of that is done for us by R. We simply call the summary method on this fitted model and we see that we get the exact same values as we did in Excel. The coefficients in our regression equation are 0. 2593, 0. 356, and 9. 25. These are the exact values that we had in Excel and this is a reassuring sign that we've carried out this correctly. However, R also displays a wealth of additional diagnostic information and not all of this is as reassuring. You can see that R has various significance codes, for instance, a very highly significant parameter will be indicated by three asterisk marks. As the parameter values get less significant, the number of asterisk marks decreases until finally at 10% significance there's merely a dot. Ideally we would want all of our regression coefficients to have at least two and ideally three to four asterisks next to them, but sadly that's not the case here. We can see that R does not have a very high opinion of this regression model that we have constructed. R is telling us that our regression equation is not very statistically significant. We are not going to spend a whole lot of time talking about the significance of regression parameters, but you can see very quickly that the sp. lagged variable has a 10% statistical significance as indicated by the dot. The other two terms are not significant at all. Should we be bothered by this? I say no because we knew when we started that this is a very hard problem. We are trying to predict the probability of Google's stock going up next month using information that's only available this month. This is a

problem that much of the financial world is trying to answer. So if we had a great answer to this question, well then, we could go out and earn a fortune in the financial markets. This was a fairly difficult problem and logistic regression has still performed creditably and we can confirm that this is the case by checking the accuracy of our logistic regression model. The way we do so is by taking the original values of Google's returns and then comparing them to the predicted label's output from our model. This is a way for us to compare the actual and the predicted labels and get to a percentage accuracy figure. We can us R's fitted function to find the probabilities corresponding to each data point in our logistic regression. We can then see which of these probabilities are greater than 0. 5. These correspond to our predictions of Google going up. All of the other points correspond to predictions of Google going down. This is the second column in the data frame that we've created on screen. The first column merely contains the actual values of the binary categorical variable. Let's make this clear by adding descriptive names to our two categorical variables and then let's go ahead and see how many of these data points represent a correct prediction. We divide this number by the total number of data points and that gives us our percentage accuracy of just under 60%. Logistic regression has done about 10 percentage points better than just a coin toss, which would have given us 50% in a binary classification problem. That gets us to the end of this module where we implemented regression, specifically logistic regression in R and it confirmed our view on the relative strengths and weaknesses of different technologies. R is just an awesome tool for any kind of regression modeling.

Implementing Logistic Regression Models in Python

Setting up the Logistic Regression Problem in Python

Hello and welcome to this, the final module of our course which deals with implementing logistic regression models in Python. The agenda in this module will be similar to that in the previous two modules. The main difference is that the choice of technology will now be Python rather than Excel or R. We will set up a logistic regression to predict whether Google's stock will rise or fall. We will solve this logistic regression in Python using a model with multiple explanatory variables. In a previous module we discussed how the choice between Excel, R, and Python really comes

down to our use case. Python is a serious programming language and it can be used for production-ready code and it also allows us to write logistic regression code along with other machine learning techniques. Those use cases are where Python really has an advantage over either R or Excel. Python even has a library called pandas which provides us the same convenience of the data frame abstraction, which we used in R. Pandas is a great library and we shall put it to use in just a little bit. So let's quickly jump into a demo to do exactly that. Let's go ahead and actually implement logistic regression in Python. We are going to rebuild the exact same logistic regression model that we did in Excel and in R. Here we are going to seek to explain whether Google's stock goes up or down using changes in the S&P 500 and changes in Google's stock in a prior period. We start by importing the three modules that are important for us: pandas which contains a data frame-like abstraction in Python, numpy which allows us to manipulate arrays, and finally statsmodels, which contains the functionality to actually carry out the logistic regression. These libraries are also fairly straightforward to install, particularly if you're using IPython or Jupyter as I am here. In such an IDE, all that we need to do is to type out the command! pipinstall followed by the name of the library such as pandas. That will get these libraries set up and ready for use. It will also handle all of the dependencies for you. Exactly as in Excel and in R, we are going to read in our data from comma separated files so we create a pair of variables each containing the names of the corresponding data files. Next, we load these into data frames using pandas. Data frames are a data abstraction which originated in R, but we can now achieve the same effect using the pandas library in Python. So we go ahead and read in our two data files into a pair of data frames. The syntax is virtually identical to that in R. Here we have the pd. read_csv method. The first argument it takes is the file path, the second is the separator. In addition, there are also arguments telling pandas which columns we are interested in and how we would like to name them. In this case we really want all of our data to be in one data frame so we just go ahead and specify that we want the column SP500 from the data frame SP to be copied over into a column of the same name in the goog data frame. And pandas is smart enough to do this correctly lining up the dates. We can check that this is the case using the head command on the good data frame and as you can see, our data now has three columns, the dates and the corresponding prices line up nicely. You might be thinking now that this demo is turning out as virtually a blue by blue repetition of the R demo and you are right. To a large extent that's because of how similar pandas is to the R data frame abstraction. Exactly as in the R demo, we also have a problem with our date column. Pandas has read in the first column containing the dates of our data points, treating them as strings rather than as date time objects. We need to change that and so we need to do a little bit of casting. We go ahead and use the pandas to_datetime method. The first argument is the column that we wish to cast; the second contains a

string format. This allows pandas to convert that column from a string to a date and we go ahead and reassign this into the date column of our data frame. We go ahead and use the head command to confirm that things are okay and indeed we now find that the date column is represented using bashes rather than slashes. This is a sign that the conversion to dates has been successful. We are ready to move to the next step of our little demo which involves converting these prices into returns.

Converting Prices to Returns

Before we can carry out the logistic regression on our data, we need to make sure that we convert the prices of Google and the S&P 500 into returns. This is an important preparatory step which is often missed by folks making use of regression whether it's logistic regression or linear regression. We had discussed the need to convert prices into returns in a previous module, implementing logistic regression models in Excel so please go back and refresh that material if you need to. The basic idea is that we never want to carry out regressions using smoothly trending data and so we convert such data into returns, that is percentage changes of adjacent points. And pandas makes this a lot easier for us to do in Python than it was in R. Switching back to our data, we are at a point where we have successfully converted our dates into date objects from strings. In order for the percentage changes to be calculated correctly, let's go ahead and sort our data frame with the dates in ascending order. We'll do this using the sort_values method on our data frame. We specify the columns that we wish to sort by. Here there's just one, that's the date column, and also the orders of each of them. Once again, here we wish dates to be sorted in ascending order. As usual, let's use the head command to make sure that our operation has gone through as we expected. And indeed it has. The data now starts from 2005. Let's now turn to the actual calculation of the percent it changes. This will require us to play around a little bit with the data types and the data columns in our frame. So let's go ahead and do exactly that. We can get a list of the column names and their data types using the. dtypes method. That's exactly what we do and now notice that we only wish to calculate percentage returns on the numeric types, that's on Goog and SP500, and obviously not on the dates. In order to isolate only those columns which are of the numeric types, let's convert this list couples into a dictionary. We'll do using the dict command and this gives us back a dictionary where the keys are the column names and the values are the column types. And we now write a slightly cryptic line of code where we get back a list with only the column headers of the numeric types. We could of course have done this by actually hard-coding those values, but there is some merit in our knowing how to do this generally. Because we often might find ourselves with data frames containing dozens of

numeric types, this is a quick and handy way of isolating only those. This is an example, by the way, of a little lambda function because we have written a little bit of logic and then gone ahead and applied it to every element in our dictionary. The result is a list containing only the two column names, Goog and SP500, the numeric data times in our data frame. We now go ahead and isolate all of the data items in these columns and we then make use of the. pct_change method in order to get back the returns. We are done. All we need to do is to now save the results of this command in a new data frame, which we call returns. Just a little bit of housekeeping left for us to do. Recall that we had stripped out the dates while calculating these returns so let's add them back in. We simply copy over the date column from our original data frame. The second bit of housekeeping that we've got to take care of is the addition of an intercept node. In this logistic regression we are going to make use of the stats models API in Python. Unlike GLM in R, this by default excludes an intercept. So we go ahead and add in an intercept into our data frame. This is simply a column in which every element is equal to 1. Let's once again use the head method to make sure that our data frame looks in good shape and it turns out that we are almost there. We now have the returns on Google and the S&P 500 as well as the intercept, but we now still have to deal with the one data point which we lost, as usual, when we went from prices to returns. Here this data point is marked with the special Python value NaN or not a number. We want to exclude this from our data frame, which we'll do by converting this data frame into a numpy array and lopping off the first row from it. This is the line of code you see on the screen now. Let's get to the np. array method in just a moment. First let's understand negative indices in Python. They are worth calling out because they work differently than negative indices in R. We learned in R that negative indices are used to exclude rows or columns from a data frame. In Python they have a slightly different meaning. Here they can be thought of as backward indices, which operate from the end of a data frame. Also notice that forward indices start from 0 and extend up to n-1; backward indices start from -1 and extend up to -n. That explains why we use the negative index to get rid of that additional point with the not a number or NaN. The use of np. array is required because numpy arrays are standard inputs into various stats model functions. Python has a rich set of statistical and quantitative libraries, stats models, Sidekick, and so on. And by convention, virtually all of these except numpy arrays as their inputs and as on the other hand is more of a way to abstract data into rows and columns and this is precisely the format that we require for all of our explanatory variables in our logistic regression so we save this in a variable that we call xData.

Logistic Regression with statsmodels

We have now set up our explanatory or x variables in a format that can be passed into the stats model logistic regression and we also go ahead and do the same with our y variable. We save this in a variable called yData and remember that in a logistic regression the y axis or the y variable needs to be categorical and this is why we are converting Google's returns into a binary variable. This y data contains either true or false depending on whether Google stock returns were positive or negative in each month. Also do pay attention to the indices that we have used in our xData and our yData. The xData omits the first row and the last row. The yData omits the first two rows. This means that x and yData have the same number of rows, but different starting points. This choice of indices are required in order to get exactly the right lineup of the y variables and the x variables. Remember that the model that we are trying to fit seeks to explain moves in Google stock return using the previous month's return on Google's stock as well as the previous month's return on the S&P. And also remember that these are very difficult problems to solve and once we have our x and our yData set up correctly, all that we need to do is to invoke the stats model. Logit method, passing in these as arguments. The result is a logistic regression object. Stats model also has similar methods that can be used to carry out linear regression, for instance. Now that we have the logit object, we go ahead and invoke the fit method on it. This gives us the fitted model. We save this result in a variable of that name and then invoke the summary method on the result and the results of the logistic regression flash before us. We can see as an aside that the log likelihood has worked out to 95. 51; this is exactly the value that we had gotten in R and Excel. We can also see that the values of our coefficients exactly match those with the other two technologies. We have chosen to skip over a discussion of standard errors and p values of our regressions on this course, but even so, I should mention here that these are not great regression results. That's okay because in this example we were attempting to solve a really difficult problem. One other regression statistic worth talking about here is the pseudo R squared of a logistic regression. If you've spent a lot of time working with linear regression, you know that the R-squared or the adjusted R-squared is the single most important metric of the quality of a regression. Unfortunately, there is no direct way to calculate an R-squared with logistic regression. There are, however, a whole bunch of Pseudo-R-squares and these are similar in philosophy to the R-square of a regression; they differ in their implementation. You can see that Python reports one such Pseudo-R-squared and it's very low, it's just 2%, but once again, we aren't too concerned about this because we knew before we started how difficult this problem was. Let's round out the demo by seeing how we can calculate the probabilities for a given set of data points. This simply makes use of the predict method on the result object. We invoke the predict method using our original xData and these give us the probabilities, which match those that we got in Excel. It's easy enough for us to convert these probabilities into a binary prediction

either up or down and we do so by using a threshold of 0. 5, and Python also allows us to very easily use the zip method to juxtapose to place side by side the actual values of Google's stock returns, whether they were up or down, and those predicted by our logistic model. And that gets us to the end of our third implantation of this logistic regression model, this time in Python.

Course Summary

That gets us to the end of this module and also to the end of this class so let's very quickly summarize what we learned. We started by considering different approaches to deadlines. One of them might be to start 5 minutes before a deadline. Good luck with that. The other might be to start a year before the deadline, probably also overkill. The smart solution we concluded is to start just late enough that we have a very high probability, maybe 95%, of meeting the deadline. If somehow we could construct such a curve magically, we could easily find out exactly when to start. We would drop a horizontal line from the y axis onto that curve at the 95% level, and then drop a perpendicular onto the x axis and get exactly the length of time required for us. This would be a curve, which has on the y axis the probability of an event and on the x axis, different causes or explanations, which drive that probability. The question then is what's a nice mathematical form for such a curve and how can it be calculated? And here we drew inspiration from the fields of business and sociology. We saw how the S-curve is used to model the diffusion of innovation. Innovations have been found to go viral, that is to reach a tipping point when they are adopted by around 16% of a market, and this is based on a very cool mathematical curve known as a S-curve. S-curves have a standard mathematical form and given a set of points, logistic regression involves finding the best fitting such S-curve. This requires us to estimate the values of the intercept A and the slope B. These terms are very similar to terms which we use in linear regression and that's because logistic and linear regression are closely mathematically related and of course, the constant e, which we see in that equation, is the base of the natural logarithms which has the value 2. 71728. And in this way, logistic regression has become a mathematical modeling problem. We need to take in a set of points, which have x and y coordinates. Note that we can have multiple x or explanatory variables represent them in this mathematical form and go ahead and plot the curve. This is something that we proceeded to do in three technologies: Excel, Python, and R. That gets us to the end of this module and to the end of this course. Thank you for watching.

Course author

Vitthal Srinivasan

Vitthal has spent a lot of his life studying - he holds Masters Degrees in Math and Electrical Engineering from Stanford, an MBA from INSEAD, and a Bachelors Degree in Computer Engineering from...

Course info

Level	Intermediate
Rating	★★★★½ (25)
My rating	★★★★★
Duration	2h 22m
Released	31 Mar 2017

Share course