Module 13: Logistic Regression

Video Transcripts

Video 1: Motivation

In the last module, you learned how to classify data using the k-nearest neighbors approach. KNN is very intuitive and it often does the job very well. However, it has one significant downside, which is that to classify a new item, you must find the nearest neighbors from amongst all of the training data. This can be quite expensive—both in terms of computation and memory. This is very different from what we did with linear regression where, to decide the value of a new point, all we needed were the coefficients of the linear model, not the entire dataset.

In this module, we will start to learn about classification models that involve only a small number of coefficients, like linear regression. The simplest of these models—a direct analog to linear regression, but for classification—is called logistic regression. As we will see, naming it logistic regression—as opposed to logistic classification—makes sense since the output of the model will be not the discrete-valued class to which the input belongs, but a continuous-valued probability that it belongs to this or that class.

To demonstrate the technique, we will use the Iris dataset. This dataset consists of measurements of the lengths and widths of petals and sepals of 150 iris flowers. You can see here the data in a pandas DataFrame. There are five columns: The sepal length, sepal width, petal length, petal width. And finally, the species of iris as an integer: 0, 1, or 2. Here, a 0 stands for the species setosa, a 1 is versicolor, and a 2 is a virginica type iris. In all of these, the sepal—which is the outer protective part of the flower—is typically wider and longer than the interior petals, which are short and narrow. Our goal will be to build a model that can classify iris flowers into setosa, versicolor, and virginica—given the size of their sepals and petals.

Let's begin with a simpler problem. Let's see if we can distinguish between virginica and versicolor by their petal lengths alone. In this plot, we see that virginica petals are typically larger than versicolor. And so, we might imagine placing the threshold about here at 4.9 centimeters and declaring that all flowers with petals larger than that number are classified as virginica and all smaller are versicolor. Sounds good. Now we need a methodology for computing this threshold.

Let's try using what we know, which is linear regression. That is, let's find a straight line that minimizes the sum of the squares of the vertical distances from the line to the data. Then, wherever that line crosses the midpoint between versicolor and virginica, that's where we will place our threshold. And here is the result. The linear regression line defines a threshold of 4.9 centimeters. This worked out well, but what if we now add an outlier flower to the dataset? If we repeat the computation, we get a new threshold, which is not too far from 4.9. But notice also how much the regression line was affected: From the green line without the outlier to the red line with the outlier. In other words, the linear regression model is sensitive to outliers.

Logistic regression avoids the problem of sensitivity that we just witnessed with linear regression. Indeed, in this case, the sensitivity of the decision threshold is one million times smaller with logistic regression than it was with linear regression. But it also has another significant advantage, which is that it produces class membership probabilities—instead of bare decision thresholds. We will see how these probabilities will be necessary to build classification models for more than two categories of objects.

Video 2: One Feature, Two Classes

Imagine now that you had to guess whether an unseen flower was a versicolor or virginica based only on its petal length. A reasonable way to go about this would be to go back to the dataset and find all flowers with similar petal lengths and then decide by majority vote. Here's an illustration of such a strategy. First, we grid the petal length space into some number of bins. Then, we count the number of each type of flower in each bin. The horizontal pink lines here represent the probability that a flower, taken from that bin, is virginica. And it is computed as follows.

The probability that the outcome Y equals virginica, given that we are sampling from bin i, equals the number of virginica flowers in bin i divided by the total number of flowers in bin i. The probability that the flower is versicolor is, of course, just one minus this quantity. As we move from left to right, from smaller to larger petal lengths, we will expect the probability of virginica to rise from zero to one—just as we expect the probability of versicolor to decrease from one to zero. The threshold petal length separating virginica and versicolor should then be placed at the bin corresponding to equal parts of the two flower types. That is, where the probability equals 0.5.

We are now going to formalize the example. Instead of the species of flower, we will refer to the output class 'Y'. We will use class names 0 and 1 instead of versicolor and virginica. And instead of petal length, we will consider some arbitrary feature called 'X'. To derive logistic regression, we will begin by assuming that the marginal distribution of X—remember X is akin to the petal length—for each of the classes zero and one are Gaussian.

In formal notation, this means that the probability of X, given Y equals 0, is normally distributed with mean mu_0 and variance v_0 squared. And the probability of X, given Y equals 1, is normally distributed with mean mu_1 and variance v_1 squared. Furthermore, we will assume that the two variances are the same. V_0 squared equals v_1 squared. And we will denote them both by v^2. Let's now do the same thing we did before. Let's look at a narrow bin and find the probability that the items within that bin are of Type 1. This probability will turn out to be very important. So, we're going to give it its own symbol, sigma. This might be a little confusing because we had previously used sigma for the standard deviation. But the reason will soon become clear.

Alright. Just as before, this probability is the ratio of the number of items of class one divided by the total number of items. This equals the ratio of the blue shaded area divided by the sum of blue and orange. If we now divide the numerator and denominator by the blue area, we see that sigma will depend only on the ratio of the two areas, blue divided by orange. So, let's focus on that term. In fact, this ratio is an important quantity with its own name, the odds ratio.

Next, we use the assumption that the marginal distributions are Gaussian. We plug the formula for the Gaussian distribution into the numerator and denominator. And keep in mind that they have different means, mu_0 and mu_1. Because the variances were assumed to be the same, they cancel out. And we're left with a single exponential function of the difference between two squares, (x – mu_1)^2 and (x – mu_0)^2. After a few more lines of algebra, we arrive at our conclusion that the odds ratio is equal to an exponential of a linear function of X. Well, that's a mouthful. So, let's have a closer look.

The odds ratio equals an exponential: e^–z, where z is a linear function of X with coefficients beta_0 and beta_1. Furthermore, we can compute these betas from the parameters of the marginal distributions, mu_0 and mu_1, and the common variance, v^2. OK. We can now use this expression for the odds ratio to complete our formula for sigma. And so we get that sigma of x equals 1 over 1 + e^–z, where z equals beta_0 + beta_1 times x. This function is known as the sigmoid function because of its similarity to the letter S. It is also known as the logistic function.

Once we have found the particular logistic function that best fits our data, we can then use it to compute the threshold that separates the two classes. Let's call this threshold x-bar. As we said before, we choose the threshold such that the probabilities of each class are the same and equal to 0.5. So, sigma of x-bar must equal 0.5. Plugging this into the formula, we get that z-bar must equal 0, since e^0 equals 1. And so, beta_0 plus beta_1 x-bar must equal 0. And therefore the threshold, x-bar, equals negative beta_0 over beta_1. Next, we will address how to obtain the betas.

We already derive some formulas, but only by making some pretty broad assumptions. We assume not only that the marginals were Gaussian, but also that we know their mean and variance, and that their variance are equal. We don't know these quantities, but we can approximate them with the sample means and variances. Let's call this the approximate solution. In the real world, this approximate solution will take us only so far. We will soon be extending this method to cases where no closed-form solution exists and we will need a more powerful tool for computing the betas. For this, we will adopt the same approach as we have seen before with other models, such as linear regression. That is, we will formulate logistic regression as an optimization problem. Let's do that.

The cost function that we will use to choose the parameters beta_0 and beta_1 is the likelihood function. Here's how it is defined. The likelihood of a set of parameters, in this case beta_0 and beta_1, is the probability that the dataset that we have observed could have been generated by the model implied by those parameters. Let's look at an example. Here we have a dataset with measurements of two classes: Green and red. What is the likelihood of this Candidate logistic function?

Well, this function implies the green and red marginal distributions that are well to the right of the data that we have. It's not that it's impossible that this dataset could have arisen from the blue model, it is just very unlikely. So, this logistic function has very low likelihood. This pink model, on the other hand, has a higher likelihood. And our objective will be to find the model with the highest likelihood, given the data. To quantify the likelihood of a candidate beta_0 and beta_1, we take the product of the probabilities of the data samples according to those parameters. In this example, the probability of the red point of class 1 is sigma, and of the green point of class 0 is 1 minus sigma. So, we take the product of 1 minus sigma(x_i) for all green points and multiply that by the product of sigma(x_i) for all red points. If we assign a class label of 0 to green and 1 to red, then we can express this as a single product of 1 minus sigma(x_i) to the 1 minus y_i times sigma(x_i) to the y_i. Here, N in this formula is the number of samples. Take a moment with this formula to make sure that you understand why it is true.

The next step is to take the logarithm of both sides, this has the beneficial effect of converting the product into a sum and it has no effect on the result. Then, using the properties of logarithms and with a couple of lines of algebra, we get to the final objective function for our optimization. That is, the sum over the data of 1 minus y_i times log of 1 minus sigma plus y_i times log of sigma. This formula is known to information theorists as the cross entropy, or rather the negative of the cross entropy. So, we will adopt that name. Instead of maximizing the likelihood, for classification problems, we will minimize the cross entropy. And here is the complete optimization problem for logistic regression. It reads: find beta_0 and beta_1 that minimize the cross entropy, defined as the negative of this formula involving sigma, where sigma is defined as the logistic function.

So, to summarize. In this video, we have derived a formula for the sigmoid or logistic function, which will serve as a basis for simple classification problems. We have posed the problem of selecting the parameters of the model as an optimization problem in which we minimize the cross entropy. That's it for logistic regression. In the next couple of videos, we will learn about extensions of this technique, to problems with a) More than one input and, b) More than two classes. See you then.

Video 3: One Feature, Two Classes in Code

In this video, we will demonstrate how to run logistic regression with scikit-learn. We're going to do it with the Iris dataset, which we can load from the datasets module of scikit-learn. And that's what we do here. This data is a dictionary that contains the input data. And we are going to cast that into a DataFrame. The target information is included in data.target. And we're going to add that under the species column into our DataFrame for ease of reference.

So, this is what it looks like. We have four columns that have the sepal length, the sepal width, the petal length, and the petal width. Now, these columns have long names, and so we'd rather rename them to something a little bit shorter. I'm going to call the sepal length sl, the sepal width sw, the petal length pl, and the petal width pw. So, after I rename that, this is what the DataFrame looks like. You can see that the species, whether a setosa, versicolor, or virginica, is encoded as a 0, 1, or 2. Setosa is 0, versicolor is 1, and virginica is 2.

Alright. So, now we can do logistic regression and we'll start with the simplest case. That is: two classes, one input. The two classes are going to be versicolor and virginica. So, let's select from our DataFrame those rows that correspond to virginica and versicolor, and save them to a new DataFrame that we'll call iris_2species. Now from there, we can select the column that is petal length, and that will be our input data. And the output data, or the target, will be the species column. To run logistic regression in scikit-learn, we use the LogisticRegression class, which is part of a linear model module. And what we have to do, is to instantiate one of these class and run on it the fit method, passing in the input data and the target. And this will produce an object, which we'll call lr, that contains the coefficients, or the result of the logistic regression. And this is what it looks like.

So, the intercept attribute of lr contains beta0. And then the coefficients, coef_, is going to contain the coefficients that correspond to each input. So, in this case there's only one input. We get beta1. And we can apply, in this case, the formula that we derived for the threshold. And that's going to be negative beta0 over beta1. So, let's see what we got. Alright. The threshold is 4.87, meaning that flowers with a petal length of greater than 4.87 are going to be considered virginicas and less than that are going to be considered versicolors.

Now, in the lecture, we also derived some formulas for approximating this threshold. And let's try out those formulas here. So, we take from our input, let's just take the rows corresponding to versicolor and save those in X1, and the virginicas, we will save them in X2. Then we can take the means of those and their variances, and because we had assumed that the variance of the two would be the same—and clearly this will probably not be the same—let's take the mean of the two variances and use those in the formula. So, our approximate beta0 was computed with this formula. Our approximate beta1 was this. And then we can use the same formula to compute the approximate threshold. What do we get?

OK, so the threshold is 4.906. That's the approximate threshold, which we can compare it to 4.87. What does this look like in a plot? Alright, so here we have plotted the versicolor flowers at a level of zero. So, they're down here. The virginicas are at a level of one. So, they're up here. And then we've evaluated this sigmoid function, while using the betas from scikit-learn and from our approximation. And this is what we get. So, the green is, let's call it the true logistic regression. And the pink is the approximate and they're pretty similar. However, these formulas won't carry us any further. They are only good for the simplest case.

Video 4: Many Features

In the previous video, we introduced the logistic regression model for classification problems. Let's recall that our original motivation for doing this was to classify iris flowers into three categories, according to the length and width of their sepals and petals. So far, however, we have only considered classification into two categories according to a single measurement. In this video, we will take the next step and learn how to incorporate all four measurements into the model.

We'll begin with two features, x_1 and x_2. The dataset is a collection of N samples, and each sample has values x_1^i and x_2^i and a categorical value y^i. We can plot the data in the x_1, x_2 plane and color each sample according to its category, y^i. Now, consider a new, uncategorized sample, x, shown here as a green dot. How shall we decide whether to classify x as orange or blue? We can do the same thing as last time. That is: We can draw a small box around the new sample, count the number of blue and orange dots inside the box, and then decide by majority vote.

To formalize this strategy, let's again assume that the marginal distributions of the orange and blue classes are Gaussian, with equal covariance matrices, capital sigma. And furthermore, that sigma is diagonal. The same formula for the log odds from the one-dimensional case still applies here in the two-dimensional case, except that now the terms are vector valued. The notation for the elements of x, mu_1, mu_2, and capital sigma are shown here. After some tedious algebra, we arrive at this complicated formula for the log odds. But the details are not the point here. The point is that the form of the log odds is identical to what we found in the single-input case.

The log odds are again a linear combination of the inputs with an intercept term beta_0 and coefficients beta_1 and beta_2 for the two inputs. The fact that we made some very unrealistic assumptions to get to this formula is not very important because we will not use this formula going forward. Again, we will rely on the machinery of optimization to find the parameters. But this result shows us a path forward to add even more input features, all we need to do is add more linear terms to the log odds. And so, we arrive at the general statement for the two-class logistic regression with M inputs. Keep in mind that the random variable X is now multi-dimensional and lowercase x is a vector-valued sample. The result is that the probability that a sample x is of class Y equals 1, equals the logistic function sigma of x, where z is a linear combination of the M features with coefficients beta_0 through beta_M. And the betas are found just as before by minimizing the cross entropy.

Next, we're going to address a common problem with logistic regression. We will illustrate this in the single-feature case, but it shows up with multiple features as well. Imagine you have a dataset where there is no overlap between the two classes. That is, there exists a threshold that exactly splits the data with perfect accuracy. What will the optimization problem do in this case? Well, let's consider two candidate solutions – sigma_A in pink and sigma_B in light blue color.

To evaluate the likelihood of each of these, we need to take the product of the probabilities of each data point, which are the heights of the vertical blue and pink lines in the figure. And so we see that sigma_B must have a higher likelihood than sigma_A, because in every case the vertical lines are longer. But we could then take one step further to a sigma_C that is even steeper than sigma_B, and then to a sigma_D, and so forth ad infinitum. In this case, there is no solution to the optimization problem. With each iteration of the optimizer, the coefficients—beta_0 and beta_1—will continue to grow without bound and the solution will converge toward an infinitely steep step function, like the one indicated here as sigma infinity.

This bad situation was caused by the fact that the data is cleanly separable. And it is unfortunate because separable data should be the easiest case to solve and it shouldn't cause the algorithm to explode. To avoid this problem, we must add a regularization term to the cost function to penalize large coefficient values. You've already learned about regularization. So, I won't go into it deeply here. I will just mention that we can use either L1 regularization or L2 regularization to fix this problem. Both of these amount to appending a term to the cross entropy cost function. In the L1 or lasso case, it is a sum of the absolute values of the coefficients. And in the L2 or ridge regression case, it is the sum of the squares of the coefficients. The effect in both cases is to limit the size of the coefficients.

One last note, the regularization parameter in scikit-learn is called c—not lambda. And it corresponds to the inverse of lambda. So, to increase the strength of the regularization, you actually need to decrease c. In the demo video, I will demonstrate how to do this and how to use L1 regularization for feature selection.

Video 5: Many Features in Code

And now let's go on to the next level, which is two classes and two inputs. So, instead of only using one of the columns from two species, let's use both the petal length and the petal width. And again, our target is going to be the species. But aside from that, the code doesn't change very much. Again, we call LogisticRegression, we construct it, call the fit method, and pass in the data. And this is going to give us intercepts and coefficients. And now, there are two coefficients corresponding to the two inputs.

Alright, so now let's plot. And our plots are now going to be two-dimensional because we have two inputs. So, the data now lives not on a line as before, but in the petal length, petal width plane. This function here, plot_region, created this gray and green region corresponding to the virginica and versicolor flowers. So, the model has determined that anything that is to the right of this decision boundary, should be considered a virginica flower. Whereas, if it's to the left, we'll consider it a versicolor flower. And this, of course, is not a perfect model. But logistic regression could not possibly have classified this perfectly, because a line, there is no line that can separate these two datasets cleanly. So, this is as good as logistic regression could do.

Video 6: Multiclass

In the last video, we developed the machinery for building logistic regression models that can use any number of features to distinguish between two classes of items. However, most of the interesting problems in the world involve more than two types of things. For example, we can think about recognizing handwritten digits, or predicting customer satisfaction, or credit scoring, or many others in medicine, engineering, business, et cetera. In this video, we will complete the picture of logistic regression by extending our techniques to problems involving more than two classes.

We will cover three methods for adapting binary classification models to multiclass problems. The first two, one-vs-rest and one-vs-one, are generic and can be used with other classification models aside from logistic regression. One-vs-rest classification requires that the underlying binary classification model must return a continuous probability for the prediction, not simply a binary class assignment. One-vs-one classification does not have this requirement. It can be used to extend any binary classifier into multiclass problems. The last approach, multinomial regression, is a generalization of logistic regression to the multiclass case.

To illustrate, we will use a two-dimensional problem with four classes. We will use a capital letter K for the number of classes. So, in this case, K equals four. The one-vs-rest, or OVR, strategy uses a binary classifier to build K separate models, each one pitting one class against all of the others. The first of these classifiers will be trained to distinguish class one from all other classes. Here we show the decision boundary for that model.

But what we will actually use from the model is the continuous-valued probability of class 1 as a function of x. If the binary classifier is logistic regression, then this will be a logistic function, which we denote as sigma_1(x). The one-vs-rest model consists of the K separate binary classifiers. To predict the class of a new data point—shown here in green—we simply evaluate each of the binary classifiers and select the one that produces the largest value. That is, we select the class that the point x is most likely to belong to according to the K binary classifiers.

Here we see the class regions that result from the one-vs-rest approach. Note that the boundaries are linear despite the fact that they were obtained by taking the maximum over sigmoid functions. The one-vs-rest extension of logistic regression, therefore, is a linear classifier. The OVR approach made use of the probability estimate generated by the binary classifier. However, not all binary classifiers produce a probability estimate. The perceptron, for example, generates a decision boundary, but no probability estimate. For this type of classifier, we can use the one-vs-one approach.

In one-vs-one, we run the binary classifier on every pair of classes. So, if we have three classes we run one versus two, one versus three, two versus three. And the total number of executions of the binary classifier is therefore K times K minus 1 over 2. For 4 classes, we must construct 4 times 3 over 2 is equal to 6 binary models. For 5 classes, we must build 5 times 4 over 2 is equal to 10 models, et cetera. We will focus, for now, on the simpler case of three classes. The decision boundaries for the three binary classifiers are shown.

To classify a new data point, we evaluate all three binary classifiers and decide by majority vote. In this case, the green sample is regarded as class 3 by two of the models and as class 1 by the other. Hence, it is predicted to be of class 3. These are the class regions created with one-vs-one. The boundaries of these regions are boundaries of the binary classifiers. Thus, if we…we can conclude that if the binary classifiers are linear, then the one-vs-one extension will also be linear. Note, though, that there is a problem with this picture. The region in the middle is not shaded. This is because the voting in this region resulted in a three-way tie amongst all of the classes. So, the data cannot be classified. This is a drawback of the one-vs-one approach.

Here are a few pros and cons of these two strategies. One-vs-rest has the advantage over one-vs-one, that the number of models that need to be trained grows linearly with the number of classes—as opposed to quadratically with one-vs-one. However, the size of the training problems is smaller with one-vs-one by a factor of K minus one. Furthermore, each of the model training runs with one-vs-one involves data from a single class at a time. And so, the data is likely to be more balanced between positive and negative samples than with one-vs-rest. As we saw, one-vs-rest is only possible if the binary classifier returns a probability, whereas one-vs-one can always be used. And finally, one-vs-one has the drawback that it may not be able to classify some data points for which there is a non-unique winner in the voting process, whereas one-vs-rest does not have this problem.

Next, we will talk about multinomial regression, which is our most general form of logistic regression. Multinomial regression is similar to one-vs-rest in that we will calculate a continuous probability for each class. And it is similar to one-vs-one, in that each of the models will be trained by comparing one class to just one other. The derivation begins by specifying a reference class to which all other classes will be compared. We will use class K as the reference. However, the result will be the same with any other selection.

Next, we built K minus 1 logistic regression models by comparing each of classes one through K minus 1 to the reference class K. Let's begin with the model, 1 versus K. The train coefficients beta_1,0 through beta_1,M, combined linearly with the features to produce the log odds, which is the logarithm of the probability of class 1 over the probability of class K. To make the notation simpler, let's define a vector, beta_1, of coefficients for this model and denote the linear combination as negative beta_1 dot x. Then we can do the same for classes 2 through K minus 1. From each of these, we can express the probability of any class kappa as the probability of class K times the exponential of negative beta dot x. Here, kappa is any class, from 1 to K minus 1.

Next, we observe that because the class probabilities must add up to 1, the probability of class K equals 1 minus the sum of probabilities from 1 to K minus 1. We can now use the expression for the probability of class kappa that we just derived and obtain that the probability of class K equals 1 over 1 plus the sum of the odds of all other classes. Using this expression, we get the probability of the other classes, 1 through K minus 1, as the odds of the class divided by the same denominator. These are the final formulas for multinomial logistic regression. They may seem complicated at first, but comparing them to the original logistic regression, you will see that they are actually very reasonable as a generalization. Furthermore, they have solved the problem of multiclass classification in a much more elegant way for logistic regression than was achieved with the generic approaches of one-vs-rest and one-vs-one, since we have obtained explicit formulas for the probabilities with only K minus 1 binary models.

That's it for multiclass regression. In the next video, we will learn how to do all of this in code.

Video 7: Multiclass in Code

Alright, so now let's go on to logistic regression with three classes and two inputs. This is now considered a multinomial logistic regression because it has more than two classes. So, we're going back to our original dataset in which we had all of the flowers. We're adding in the setosa flowers. And we're going to try to classify them based on petal length and petal width. And there are two possible tactics for doing this. One is called one-vs-rest and the other's called multinomial. But you'll see that the code is very similar. All we have to do is pass this parameter, multi_class, into the constructor for LogisticRegression and give it a value of OVR, if we want one-vs-rest, and multinomial, if we want multinomial regression. But the rest is the same. Then we call the fit method and supply it with the data. Let's see what happens.

This is what the one-vs-rest method gave us. That is, it created these three regions for separating the virginica, versicolor, and setosa flowers. The result for multinomial looks a little bit different. Now, it's hard to say which one of these is better. If you wanted to do that, you would have to evaluate them using some of the metrics that you learned in the k-nearest neighbors part of the course—like precision, accuracy and so forth. And you would also want to split the data into a training dataset and a testing dataset. But we're not going to do that here. This is only about logistic regression. So, let's move on.

Finally, I want to show you how to use logistic regression to do feature selection via L1 regularization. So, this is useful if, say, you have a large dataset with many, many features—say hundreds of features—and you don't know which ones to use in your model. In this case, we have four features. So, let's say that we wanted, we didn't know which ones were best and we decided to use all of them, all four. And again, we're going to use them to separate only two species. So, virginica and versicolor. To do that, we are going to run LogisticRegression. But this time we are going to add the L1 regularization penalty. And the strength of the penalty, C, we are going to vary from ten to the minus 1.2 to ten to the minus 0.5. Now, we, when we loop over this, each time we solve it, we are going to store the resulting coefficients in this array. Let's see what we get.

This is the plot that we get. I've flipped the x-axis so that C gets smaller from left to right. As C gets smaller, the regularization actually gets stronger. And what this means, is that the… as the regularization gets stronger, it's becoming more and more expensive to keep coefficients that are not doing much work in the model. And so first, the model is going to sacrifice the sepal width coefficient and then the petal width, and then the sepal length, and then the petal length.

So, we can conclude from this that if you wanted to build a model, but you only could include one feature, that feature should be the petal length, because that's the most valuable feature. If you could include two features, then you would add to it the sepal length. If you can include three, you should add the petal width. So, this you can apply to models with hundreds of features and, in this way, rank them according to their value in the task of modeling.