Supervised Machine Learning: Crash Course Statistics #36

Crash Course: Statistics
https://youtube.com/watch?v=jmLid2x9eKg
https://nerdfighteria.info/v/jmLid2x9eKg

Hi, I'm Adriene Hill, and welcome back to Crash Course Statistics.

We've covered a lot of statistical models, from the matched pairs t-test to linear regression. And for the most part, we've used them to model data that we already have so we can make inferences about it.

But sometimes we want to predict future data. A model that predicts whether someone will default on their loan could be very helpful to a bank employee. They're probably not writing scientific papers about why people default on loans, but they do care about accurately predicting who will.

Many types of Machine Learning (ML) do just that: build models to predict future outcomes. And this field has exploded over the past few decades. Supervised Machine Learning takes data that already has a correct answer, like images that have been labeled as "cat" or "not a cat", or the current salary of a company's CEO, and tries to learn how to predict it.

It's supervised because we can tell the model what it got wrong. It's called Machine Learning because instead of following strict rules and instructions from humans, the computers (or machines) learn how to do things from data. Today, we'll briefly cover a few types of supervised Machine Learning models, logistic regression, Linear Discriminant Analysis, and K Nearest Neighbors.

Intro Say you own a microloan company. Your goal is to give short term, low interest loans to people around the world, so they can invest in their small businesses. You have everyone fill out an application that asks them to specify things like their age, sex, annual income, and the number of years they've been in business.

The microloan is not a donation, the recipient is supposed to pay it back. So you need to figure out who is most likely to do that. During the early days of your company, you reviewed each application by hand and made that decision based on personal experience of who was likely to pay back the loan.

But now you have more money and applicants than you could possibly handle. You need a model--or algorithm--to help you make these decisions efficiently. Logistic regression is a simple twist on linear regression.

It gets its name from the fact that it is a regression that predicts what's called the log odds of an event occurring. While log odds can be difficult, once we have them, we can use some quick calculations to turn them into probabilities, which are a lot easier to work with. We can use these probabilities to predict whether an individual will default on their loan.

Usually the cutoff is 50%. If someone is less than 50% likely to default on their loan, we'll predict that they'll pay it off. Otherwise, we'll predict that they won't pay off their loan.

We need to be able to test whether our model will be good at predicting data it's never seen before. Data it doesn't have the correct answer for. So we need to pretend that some of our data is "future" data for which we don't know the outcome.

One simple way to do that is to split your data into two parts. The first portion of our data, called the training set, will be the data that we use to create--or train--our model. The other portion, called the testing set, is the data we're pretending is from the future.

We don't use it to train our model. Instead, to test how well our model works, we withhold the outcomes of the test set so that the model doesn't know whether someone paid off their loan or not, and ask it to make a prediction. Then, we can compare these with the real outcomes that we ignored before.

We can do this using a what's called a Confusion Matrix. A Confusion Matrix is a chart that tells us what actually happened--whether a person paid back a loan--and what the model predicted would happen. The diagonals of this matrix are times when the model got it right.

Cases where the model correctly predicted that the person will default on the loan is called a True Positive. "True" because it got it right. "Positive" because the person defaulted on their loan. Cases where the model correctly predicted that a person will pay back the loan are called True Negatives. Again "true" because it made the correct prediction, and "negative" because the person did not default.

Cases where the model was wrong are called False Negatives--if the model thought that they would not default--and False Positives--if the model thought they would default. Using current data and pretending it was future data allows us to see how this model performed with data it had never seen before. One simple way to measure how well the model did is to calculate its accuracy.

Accuracy is the total number of correct classifications--Our True Positives and True Negatives--divided by the total number of cases. It's the percent of cases our model got correct. Accuracy is important.

But it's also pretty simplistic. It doesn't take into account the fact that in different situations, we might care more about some mistakes than others. We won't touch on other methods of measuring a model's accuracy here, but it's important to recognize that in many situations, we want information above and beyond just an accuracy percentage.

Logistic regression isn't the only way predict the future. Another common model is Linear Discriminant Analysis or LDA for short. LDA uses Bayes' Theorem in order to help us make predictions about data.

Let's say we wanna predict whether someone would get into our local state college based on their high school GPA. The red dots represent people who did not get in, green are people who did. If we make a couple of assumptions, we can estimate the GPA distributions of people who did, and did not get their acceptance letter.

If we find a new student who wants to know if they will get in to your local state school, we use Bayes Rule and these distributions to calculate the probability of getting in or not. LDA just asks, "Which category is more likely?" If we draw a vertical line at their GPA, whichever distribution has a higher value at that line is the group we'd guess. Since this student, Analisa has a 3.2 GPA, we'd predict that she DOES get in.

Since it's more likely under the "got in" distribution. But we all know that GPA isn't everything. What if we looked at SAT Scores as well.

Looking at the distributions of both GPA and SAT scores together can get a little more complicated. And this is where LDA becomes really helpful. We want to create a score, we'll call it Score X, that's a linear combination of GPA and SAT scores.

Something like this: We, or rather the computer, want to make it so that the Score X value of the admitted students is as different as possible from the Score X value of the people who weren't admitted. This special way of combining variables to make a score that maximally separates the two groups is what makes LDA really

special. So, Score X is a pretty good indicator of whether or not a student got in.

AND that's just one number that we have to keep track of, instead of two: GPA and SAT score. For this sample, my computer told me that this is the correct formula: Which means we can take the scatter plot of both GPA and SAT score and change it into a one-dimensional graph of just Score X. Then we can plot the distributions and use Bayes Rule to predict whether a new student, Brad, is going to get into this school.

Brad's Score X is 8, so we predict that he won't get in, since with a score X of 8, it's more likely that you won't get in than that you will. Creating a score like Score X can simplify things a lot. Here, we looked at two variables, which we could have easily graphed.

But, that's not the case if we have 100 variables for each student. Trust me, you don't want your college admissions counselor making admissions decisions based on a graph like that. Using fewer numbers also means that on average, the computer can do faster calculations.

So if 5 million potential students ask you to predict whether they get in, using LDA to simplify will speed things up. Reducing the number of variables we have to deal with is called Dimensionality Reduction, and it's really important in the world of "Big Data". It makes working with millions of data points, each with thousands of variables, possible.

That's often the kind of data that companies like Google and Amazon have. The last machine learning model we'll talk about is K-Nearest Neighbors. K-Nearest Neighbors...or KNN for short...relies on the idea that data points will be similar to other data points that are near it.

For example, let's plot the height and weight of a group of Golden Retrievers, and a group of Huskies: If someone tells us a height and weight for a dog--named Chase--whose breed we don't know...we could plot it on our graph. The four points closest to Chase are Golden Retrievers, so we would guess he's a Golden Retriever. That's the basic idea behind K-Nearest Neighbors!

Whichever category--in this case dog breed--has the more data points near our new data point is the category we pick. In practice it is a tiny bit more complicated than that. One thing we need to do is decide how many "neighboring" data points to look at.

The K in KNN is a variable representing the number of neighbors we'll look at for each point--or dog--we want to classify. When we wanted to know whether Chase was a Husky or a Golden Retriever, we looked at the 4 closest data points. So K equals 4.

But we can set K to be any number. We could look at the 1 nearest neighbor. Or 15 nearest neighbors.

As K changes, our classifications can change. These graphs show how points in each area of the graph would be classified. There are many ways to choose which k to use.

One way is to split your data into two groups, a training set and a test set. I'm going to take 20% of the data, and ignore it for now. Then I'm going to take the other 80% of the data and use it to train a KNN classifier.

A classifier basically just predicts which group something will be in. It classifies it. We'll build it using k equals 5.

And we get this result: Where blue means Golden Retriever. And red means Husky. As you can see, the boundaries between classes don't have to be one straight line.

That's one benefit of KNN. It can fit all kinds of data. Now that we have trained our classifier using 80% of the data, we can test it using the other 20%.

We'll ask it to predict the classes of each of the data points in this 20% test set. And again, we can calculate an accuracy score. This model has 66.25% accuracy.

But we can also try out other K's and pick the one that has the best accuracy. It looks like using a k of 50 hits the sweet spot for us. Since the model with k equals 50 has the highest accuracy of predicting Husky vs.

Golden Retriever. So, if we want to build a KNN classifier to predict the breed of unknown dogs, we'd start with a K of 50. Choosing model parameters--variables like k that can be different numbers--can be done in much more complex ways than we showed here, or could be done using information about the specific data set you're working with .

We not going to get into alternative methods now, but if you're ever going to build models for real, you should look it up. Machine Learning focuses a lot on prediction. Instead of just accurately describing our current data, we want it to pretty accurately predict future data.

And these days, data is BIG. By one estimate, we produce 2.5 QUINTILLION bytes of data per day. And supervised machine learning can help us harness the strength of that data.

We can teach models or rather have the models teach themselves how to best distinguish between groups like will pay off a loan and those that won't. Or people who will love watching the new season of The Good Place `and those that won't. We're affected by these models all the time.

From online shopping, to streaming a new show on Hulu, to a new song recommendation on Spotify. Machine learning affects our lives everyday. And it doesn't always make it better we'll get to that.

Thanks for watching. I'll see you next time.