Logistic Regression

The Method

Logistic regression extends the idea of linear regression to cases where the dependent variable, y, only has two possible outcomes, called classes. Examples of dependent variables that could be used with logistic regression are predicting whether a new business will succeed or fail, predicting the approval or disapproval of a loan, and predicting whether a stock will increase or decrease in value. These are all called classification problems, since the goal is to figure out which class each observation belongs to.

Similar to linear regression, logistic regression uses a set of independent variables to make predictions, but instead of predicting a continuous value for the dependent variable, it instead predicts the probability of each of the possible outcomes, or classes.

Logistic regression consists of two steps. The first step is to compute the probability that an observation belongs to class 1, using the **Logistic Response Function:**

$$P(y=1) = \frac{1}{1+e^{-(\beta_0+\beta_1 x_1+\beta_2 x_2+\ldots+\beta_k x_k)}}$$

The coefficients, or β values, are selected to maximize the likelihood of predicting a high probability for observations actually belonging to class 1, and predicting a low probability for observations actually belonging to class 0.

In the second step of logistic regression, a threshold value is used to classify each observation into one of the classes. A common choice is 0.5, meaning that if $P(y=1) \geq 0.5$, the observation is classified into class 1, and if $P(y=1) < 0.5$, the observation is classified into class 0. Simply stated, each observation is classified into the class with the highest probability.

However, other threshold values can be chosen, and in some cases are more appropriate. The threshold value that should be selected often depends on error preferences. When the probabilities are converted into class predictions, two types of errors can be made: false positives, and false negatives. A false positive error is made when the model predicts class 1, but the observation actually belongs to class 0. A false negative error is made when the model predicts class 0, but the observation actually belongs to class 1. If a higher threshold value is selected, more false negative errors will be made. If a lower threshold value is selected, more false positive errors will be made.

One application where decision-makers often have an error preference is in disease prediction. Suppose you built a model to predict whether or not someone will develop heart disease in the next 10 years (like the model we saw in the Framingham Heart Study lecture). We will consider class 1 to be the outcome in which the person does develop heart disease, and class 0 the outcome in which the person does not develop heart disease. If you pick a high threshold, you will tend to make more false negative errors, which means that you predicted that the person would not develop heart disease, but they actually did. If you pick a lower threshold, you will tend to make more false positive errors, which means that you predicted they would develop heart disease, but they actually did not. In this case, a false positive error is often preferred. Unnecessary resources might be spent treating a patient who did not need to worry, but you did not let as many patients go untreated (which is what a false negative error does).

Now, let's consider spam filters. Almost every email provider has a built in spam filter that tries to detect whether or not an email message is spam. Let's classify spam messages as class 1 and non-spam messages as class 0. Then if we build a logistic regression model to predict spam, we will probably want to select a high threshold. Why? In this case, a false positive error means that we predicted a message was spam, and sent it to the spam folder, when it actually was not spam. We might have just sent an important email to the junk folder! On the other hand, a false negative error means that we predicted a message was not spam, when it actually was. This creates a slight annoyance for the user (since they have to delete the message from the inbox themselves) but at least an important message was not missed.

This error trade-off can be formalized with a Confusion Matrix or a Receiver Operator Characteristic Curve (ROC curve). A confusion matrix compares predicted classes with actual classes for a particular threshold value, while an ROC curve plots the false positive rate versus the true positive rate for all possible threshold values. The ROC curve motivates an important metric for classification problems: the AUC, or Area Under the Curve. The AUC of a model gives the area under the ROC curve, and is a number between 0 and 1. The higher the AUC, the more area under the ROC curve, and the better the model. The AUC of a model can be interpreted as the model's ability to distinguish between the two different classes. If the model were handed two random observations from the dataset, one belonging to one class and one belonging to the other class, the AUC gives the proportion of the time when the observation from class 1 has a higher predicted probability of being in class 1. If you were to just guess which observation was which, this would be an AUC of 0.5. So a model with an AUC greater than 0.5 is doing something smarter than just guessing, but we want the AUC of a model to be as close to 1 as possible.

Logistic Regression in R

Suppose the training data for your model is in a data frame called "TrainingData", consisting of your dependent variable "DependentVar", and your two independent variables "IndependentVar1" and "IndependentVar2". (If you just have one dataset, you can randomly split your data frame into a training set and testing set with the sample.split function.) Then you can build a logistic regression model with the following command:

```
LogModel = glm(DependentVar ~ IndependentVar1 + IndependentVar2, data=TrainingData, family=binomial)
```

You can see the coefficients and other information about the model with the summary function:

```
summary(LogModel)
```

You can then create a vector of predictions for the training set and generate different confusion matrices with the predict() and table() functions:

```
TrainPredictions = predict(LogModel, type="response")
table(TrainingData$DependentVar, TrainPredictions >= 0.5)
table(TrainingData$DependentVar, TrainPredictions >= 0.3)
```

You can generate an ROC curve with the following commands (you first need to install and load the "ROCR" package):

```
ROC.Pred = prediction(TrainPredictions, TrainingData$DependentVar)
ROC.Perf = performance(ROC.Pred, "tpr", "fpr")
plot(ROC.Perf)
```

To add threshold labels and colors, replace the plot command with the following:

```
plot(ROC.Perf, colorize=TRUE, print.cutoffs.at=seq(0,1,by=0.1), text.adj=c(-0.2,1.7))
```

The AUC of the model can be computed with the following command:

```
as.numeric(performance(ROC.Pred, "auc")@y.values)
```

To make predictions on a test set called "TestData", you can use the predict() function:

```
TestPredictions = predict(LogModel, newdata=TestData, type="response")
```

You can then create confusion matrices, an ROC curve, and compute the AUC just like we did for the training set on the test set.