Linear Models Self-Check

605.645 – Artificial Intelligence - Butcher

The purpose of this self-check is to prepare you for the programming assignment. Check Blackboard for the due date and the Syllabus for the grading criteria.

You can reference the Module Pseudocode for this assignment. You might also want to grab a sheet of graph paper…

Data

Let's be a bit more explicit about what we mean by "data". Whatever form it might originally take ("raw data"), for most (almost all?) pattern recognition ("machine learning") algorithms, "data" refers to a rectangular array of data (in Python, a List of Lists). You might think of it as an Excel spreadsheet or a database table. Hadley Wickham calls it "tidy data".

Each row (Python: interior List; Excel: row) is a single observation. Every element is a measurement of a feature (Excel: column), usually numeric (like 2.34) or categorical (like "blue"). It can get a lot more complicated than that but for now, this suffices. There can be any number of features, it just depends on the problem.

The number of data points (usually *n*) is the length of the List or the number of rows in the file/Excel spreadsheet, etc.

If we have a *supervised* learning problem, one of these features is considered to be important because want to predict or estimate it. By at least one convention, we put that special "x" at the *end* of the row and call it "y" (the target variable). There are fancy Latin names for these things that need not concern us here.

For example, we might have a single observation as:

[3.41, 2.81, "a", 96.21, 0.001, "south", 2.3]

And the very last value is y (y=2.3). All the others are features. This is just a convention. Because hy is numeric, this observation would be for a regression problem.

There are a ton of ways of storing such data. A database, an Excel spreadsheet, a file using CSV (comma separated values), TSV (tab separated values), and even something like ARFF format. But that is what we mean by "data"…"tidy data" or "tabular data".

Linear Regression

Let's consider the simplest case. Remember this from grammar school?

$$y = mx + b$$

if we flip it:

$$y = b + mx$$

Using our notation, this is the same as:

$$\hat{y} = \theta_0 x_0 + \theta_1 x_1$$

where x_0 always equals 1 so we have:

$$\hat{y} = \theta_0 + \theta_1 x_1$$

we use the more general representation:

$$\hat{y} = \theta x$$

because the actual math during learning works out easier and it represents the most general case of any number of x's. There's nothing special about adding more x's except:

1. It is definitely harder to draw.
2. In some cases, you cannot use *exact* methods to solve the problem we're about to solve.

So let's assume some starting values for the thetas (taken "at random") like:

$$\theta = [1.3, 2.9]$$

(remember, the thetas are over the *features…the first theta is for x_0 and the second theta is for x_1)*.

1. Draw this line on a piece of graph paper.

Remember that in this case, 1.3 is the y-intercept (*b*) and that 2.9 is the "rise over the run" (*m* or dy/dx) so if we move x by 1, we move y by 2.9. In any case, this is the line we start with and the purpose of "learning" is to adjust those value so that it "fits" our data. How are we going to do that? Gradient Descent.

First, let's think about this a bit. If we only have one data point, we can move the line (thetas) to go through it exactly but there are an infinite number of such lines. If we have two data points, we can move the line to go through both of them exactly and there is only one such line.

If we have more than two data points, as long as they aren't all collinear (are all exactly "in" a line), we can't find a line that goes through them all.

In fact, in this case, there is no single unique line that will represents the data. So which line are we going to pick? We're going to pick the line that minimizes "error" (also referred to more generally as "loss"). In the case of linear regression, we're going to find the line that minimizes average squared error which means we end up with a line that goes through the data "averagely".

Assume our data points are:

[1.0, 2.0]

[3.0, 1.0]

Remember,

- What is our convention for data? The last value is y. The rest are features (x's).
- What is our convention for x_0? It is implied (this means it isn't in the actual observation but has to be added in). *This implicit x_0 is only for some models…linear regression, for example, and not for all of them (for example, it's not true of Decision Trees)*

This means that the first point above [1.0, 2.0] is actually xs = [1.0] and y = 2.0 and with the implicit x_0, we have xs = [1.0, 1.0] and y = 2.0.

What does the second point look like expressed as xs and y, with the implicit x_0?

2. **For each data point, calculate y hat.**
3. **For each data point, draw point—both the real location and the estimated location (using different symbols or colors).**
4. **Calculate the error for this data set (mean squared error).**
5. **Using the Module pseudocode, calculate one adjustment to the thetas, assuming alpha = 0.1**
6. **Draw the new line (thetas).**
7. **Draw the new estimates (y-hats).**

If you kept doing this, it should converge to a line that goes through both points as discussed above.

Logistic Regression

Everything, more or less, that we said about linear regression is true of logistic regression, ish.

The main difference here is that y (the real value) is always 0 or 1 and y-hat is always on the *range* (0, 1) exclusive. We simply treat values that are "close to 0" as 0 and values that are "close to 1" as 1, normally using 0.5 as a threshold.

Additionally, it's not obvious how to draw a logistic regression so geometric interpretations are sketchy and the projection (using the same thetas) doesn't even come close to having the same meaning. For example, if we use the same thetas from above:

$$\theta = [1.3, 2.9]$$

and calculate the value of the logistic regression for x_1 values = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}, they're almost all exactly 1. What gives?

Taking the 2-dimensional case again, in *linear* regression, when x_1 = 0, then y = theta_0.

In *logistic* regression, if x_1 = 0, then we have:

$$\hat{y} = \frac{1}{1+e^{-\theta_0}}$$

which is a constant. Using the thetas from above, theta_0 = 1.3, and y_hat is 0.78 which says, when x_1 = 0, then the probability of being “in the class” (a “1”) is 0.78. As x_1 changes up or down, this probability goes up or down. The logistic curve determined by the thetas will cross the y axis at 0.78 (kind of the same concept as the “y intercept”).

Assume we have a new vector of initial thetas:

$$\theta = [0.8, 1.1]$$

1. **Draw an approximation of the logistic curve represented by these parameters by plotting the values for x_1 = {-3, -2, -1, 0, 1, 2, 3}.**

Now what if we have some data:

[1.1, 0]

[2.7, 1]

We are still using the conventional representation! Make the same adjustments before you start working.

2. **Calculate the error (log loss) for this data set.**
3. **Plot these actual points (x_1, y) and estimated points (x_1, y_hat).**
4. **Using the Module pseudocode, calculate one adjustment to the thetas, assuming alpha = 0.1**
5. **Draw a new approximation of the logistic curve using {-3, -2, -1, 0, 1, 2, 3}.**
6. **Plot the new y_hats.**

Finis.

LaTeX: y

= mx + b y

= b + mx

\hat{y} = \theta_0 x_0 + \theta_1 x_1

\hat{y} = \theta x

\theta = [1.3, 2.9]

\hat{y} = \frac{1}{1+e^{-\theta_0}}

\theta = [0.8, 1.1]