MATH 728D: Machine Learning Lab #8: Logistic Regression

John Burkardt

December 6, 2018

What is a good model for data that can only have two values?

Previous labs looked at linear models in which the input x was used to predict an output y that had a real number value. Now suppose that the output we are expecting is a logical or yes/no result. This is a version of the *classification problem*, in which we want a formula that assigns each set of input to a given class.

We will actually be content with a rule that returns values in the range between 0 and 1, representing the probability that x belongs to class 0 or class 1.

Our model for this problem will use the **sigmoid function**, for which we will seek appropriate parameter values θ and μ:

$$y(x) = \frac{1}{1 + e^{-\theta(x-\mu)}}$$

1 Is This Gold Coin Too Light?

An American Eagle gold coin is expected to weigh on average 33.9 grams, worth something like \$1,2000; naturally, the weight may vary a small amount. But if the weight is off by too much, we need to worry. Let's say we want the formula $y(x)$ to return the probability that we should trust a coin whose weight is x grams, and that we expect in particular to be 50% unsure of such a coin if it weighs a half gram less than the expected value. It turns out the corresponding model has $\mu = 33.9 - 0.5 = 33.4$, while θ, the steepness of the curve, is still arbitrary:

$$y(x) = \frac{1}{1 + e^{-\theta(x-33.4)}}$$

Exercise 1:

1. Make 3 plots of the gold coin sigmoid function $y(x)$ over the range $31 \leq x \leq 36$, for $\theta = 0.5, 1.0, 5.0$
2. From the plots, it seems like $\theta = 5.0$ is a good choice. Use that value for the remaining questions.
3. What is the value of $y(x)$ for $x = 31, 32.9, 33.4, 33.9, 35$?
4. Using the formula for $y(x)$, what value of x will cause $y(x)$ to return the value $\frac{1}{4}$?

2 Standardization and Normalization

When dealing with data, the units used to measure each component can vary greatly in scale. Many algorithms work best when the all the data has a moderate scale; methods involving exponentiation can fail if numeric values are too large.

There are two techniques for rescaling data to avoid such issues.

In **standardization**, for each column j of data array x, we compute the mean μ_j and standard deviation σ_j and instead of working with x we work with the standardized variable xs:

$$xs_j = \frac{x_j - \mu_j}{\sigma_j}$$

or, as expressed in MATLAB:

```
mu = mean ( x );
sigma = std ( x );
for j = 1 : n
  xs(:,j) = ( x(:,j) - mu(j) ) / sigma(j);
end
```

Now each column of xs has mean 0 and standard deviation 1.

In **normalization**, for each column j of data array x, we compute the minimum and maximum and instead of working with x we work with the normalized variable xn:

$$xn_j = \frac{x_j - \min(x_j)}{\max(x_j) - \min(x_j j)}$$

or, as expressed in MATLAB:

```
xmin = min ( x );
xmax = max ( x );
for j = 1 : n
  xn(:,j) = ( x(:,j) - xmin(j) ) / ( xmax(j) - xmin(j) );
end
```

In this case, each column of xn lies in the range [0,1].

Exercise 2:

1. Read the data *aircon_data.txt*, which contains 44 records of humidity, temperature, and comfort (1/0), storing it as array `x`;
2. Create `xs`, a standardized copy of `x`; Report the minimum and maximum for each column;
3. Create `xn`, a normalized copy of `x`; Report the mean and standard deviation of each column.

3 Heat and Humidity

Because the sigmoid function only returns values in the range $[0, 1]$, we can use it to classify data, once we have figured out a procedure for choosing the parameters θ.

The manager of an office complex has taken measuredments of humidity, (x_1), and temperature (x_2), and asked the office workers if they find the environmental comfortable $(y = 1)$ or uncomfortable $(y = 0)$.

Rather than keeping all this data, the manage would like a plot, or formula, or some simplified model that will generally suggest the comfortableness of any pair of humidity and temperature values.

Exercise 3:

1. Read the data *aircon_data.txt*, which contains $m = 44$ records of humidity, temperature, and comfort (1/0), storing it as array `x`;
2. Create `xn`, a normalized copy of columns 1 and 2 of `x`;
3. Add a final column of m ones to `xn`, and increase the number of variables n to 3;

4. Create `yn`, a copy of column 3 of `x`;
5. Choose a small learning rate α, perhaps 0.05;
6. Initialize the weight vector w to a column vector of n zeros;
7. Carry out the gradient descent method to reduce the cost J:

```
for k = 1 : kmax
  y = 1 ./ ( 1 + exp ( -xn * w ) );
  for j = 1 : n
    w(j) = w(j) - ( alpha / m ) * ( y - yn )' * xn(:,j);
  end
  J = ( 1 / m ) * sum ( - yn' * log ( y ) - ( 1 - yn )' * log ( 1- y ) )
end
```

8. Once the iteration is completed, make a single plot that shows:
   (a) the "comfortable data" with blue dots;
   (b) the "uncomfortable data" in red dots;
   (c) and the separation line $y(x) = 0.5$:

```
plot ( [-w(3)/w(1), (-w(3)-w(2))/w(1)], [0,1] );
```

4 Admission Statistics

The file *admit_data.txt* contains 100 records relating to college admission applications. Each record contains a student's scores on an English exam (x_1), on a math exam, (x_2), and an admissions decision y, which is 0 (not admitted) or 1 (admitted).

Our task is to find a logistic formula which, given x_1 and x_2, can give a y value between 0 and 1, indicating our estimate of the probability that the student will be admitted.

We will develop our formula using the first 80 values in the data set, and then test it by comparing our predictions to the actual results for the final 20 data items.

Developing the formula will be very similar to Exercise 3, since we again start with $n = 2$ x variables (and add on a final x that is always 1). We have to remember, however, to only work with the first 80 rows of data.

Exercise 4:

1. Store the contents of *admit_data.txt* as array `data`;
2. Normalize `data`, by subtracting the column min, and dividing by the difference of column max and column min;
3. Create `xtrain`, a copy of rows 1 through 80, and columns 1 and 2 of `data`;
4. Add a final column of 80 ones to `xtrain`;
5. Create `ytrain`, a copy of rows 1 through 80, column 3 of `data`;
6. Create `xtest`, a copy of rows 81 through 100, and columns 1 and 2 of `data`;
7. Add a final column of 20 ones to `xtest`;
8. Create `ytest`, a copy of rows 81 through 100, column 3 of `data`;
9. Choose a small learning rate α, perhaps 0.05;
10. Initialize the weight vector w to a column vector of $n = 3$ zeros;
11. Carry out the gradient descent method, seeking w that minimizes the error measure:

```
for k = 1 : kmax
  y = 1 ./ ( 1 + exp ( -xtrain * w ) );
  for j = 1 : n
    w(j) = w(j) - ( alpha / m ) * ( y - ytrain )' * xtrain(:,j);
  end
end
```

12. Evalute the formula for `y` using your testing data `xtest`;
13. Compute and print your average percentage error:

```
y = 1 ./ ( 1 + exp ( -xtest * w ) );
err = 100 * sum ( abs ( y - ytest ) ) / 20;
```

14. Once the iteration is completed, set `hold on` and make a single plot that shows:
    (a) the training "admitted" with blue dots;
    (b) the training "not admitted' in red dots;
    (c) the testing "admitted" with cyan dots;
    (d) the testing "not admitted' in magenta dots;
    (e) and the separation line $y(x) = 0.5$:

```
plot ( [-w(3)/w(1), (-w(3)-w(2))/w(1)], [0,1] );
```