Logistic Regression
ML_2022: Machine Learning

https://people.sc.fsu.edu/∼jburkardt/classes/ml_2022/logistic_lab/logistic_lab.pdf

Separate the sheep from the goats

The Logistic Regression Problem

Logistic regression makes a Yes/No decision for new data, based on information from old data;

- *Our old data allows us to create a yes/no formula for future use;*
- *The formula actually gives a value $y(x)$ between 0 and 1;*
- *Meaning:* $y(x) \begin{cases} < 1/2 & \textit{answer is no} \\ > 1/2 & \textit{answeris yes} \end{cases}$
- *If $y(x)$ is near 0 or 0, our formula is confident of the answer;*
- *Results close to 1/2 indicate less confidence.*

1 Things to copy:

Each of the exercises will be carried out on a particular datafile. These datafiles are available on the *datasets* page at the class website:
https://people.sc.fsu.edu/∼jburkardt/classes/ml_2022/datasets/datasets.html

You might go ahead now and download them all:

- *caesarean_data.txt*
- *comfort_data.txt*
- *gold_data.txt*

You will also need a copy of the following Python function, which you can download from the class website, or else try to construct via cut-and-paste from this text:

- *logistic_regression.py*

2 Prepare for Exercise 1:

In the lecture, we discussed the logistic function, which is given a value x and returns a result $y(x)$ between 0 and 1.

Consider the formula known as the *logistic* or *sigmoid function*:

$$y(x) = \frac{l}{1 + e^{-m(x-b)}}$$

which has parameters l (maximum value), b (break point), and m (slope).

We want to "train" this formula, using a set of data values x. If the data naturally divides into two separate groups based on the value of x, then our formula can be used to represent this behavior.

To start with, we will simply demonstrate how the parameters `l`, `b` and `m` affect the shape of the graph of $y(x)$.

3 Exercise 1:

Create a `exercise1.py`.

1. Inside *exercise1.py*, create the following function, with the signature

```
def logistic ( l, b, m, x ):
  ...
  return value
```

   Your function should evaluate the logistic function, which can be defined by:

$$y(x) = \frac{l}{1 + e^{-m(x-b)}}$$

   The logistic function has parameters l (maximum value), b (break point), and m (slope).
2. Inside *exercise1.py*, include the following function, which will test `logistic()`:

```
def logistic_test ( l, b, m ):
  x = np.linspace ( b - 2.0, b + 2.0, 21 )
  y = logistic ( l, b, m, x )
  print ( np.c_ [ x, y ] )
  return
```

3. Inside *exercise1.py*, include the following function, which will plot `logistic()`:

```
def logistic_plot ( l, b, m ):
  x = np.linspace ( b - 2.0, b + 2.0, 101 )
  y = logistic ( l, b, m, x )
  plt.plot ( x, y )
  plt.grid ( True )
  plt.show ( )
  plt.close ( )
  return
```

4. Test your functions by making the following calls:

```
      logistic ( 1, 2, 3 )
      logistic_plot ( 1, 2, 3 )
      logistic_plot ( 10, 2, 3 )
      logistic_plot ( 1, 10, 3 )
      logistic_plot ( 1, 2, 10 )
```

4 Prepare for Exercise 2:

An American Eagle gold coin is expected to weigh on average 33.9 grams, worth something like $1,7000 (it varies with the price of gold!); naturally, the weight may vary a small amount. But if the weight is off by too much, we need to worry. Suppose a counterfeiter has been producing lightweight versions of the coin. These also vary in weight, but on average weigh 33.8 grams.

Suppose that we have sent $n = 20$ gold coins to the Treasury Department, and they have returned a data file *gold_data.txt* which lists, for each coin, the weight, and a label of 0 for counterfeit and 1 for real. The Treasury Department used many tests to make this determination. We are going to try to "explain" or model their judgment using only the value of weight. We may get some false results this way, but we are interested in a quick, simple formula.

Our formula will return a number between 0 and 1, representing how strongly it thinks a given coin is fake or real. A value near 0.5 would be a coin where we can't decide, but lower values mean a fake is more likely, while higher values will indicate the coin is more likely to be real.

5 *logistic_regression()*

The logistic regression formula that will be used has the form:

$$y(x) = \frac{1}{1 + e^{-w'x}}$$

where x is an item of data for a given case. In our example, x will be a $d = 2$ vector $[1|g]$ where g is the (normalized) weight of the coin. The vector w is a set of weights.

We assume that for any weight vector we choose, we will not be able to perfectly match the data. We would like to choose the weight vector that is the least bad. It will make sense to measure the error by computing the error for each data item, squaring, summing, and averaging. In other words, we will be looking again at mean square error or MSE:

$$mse = \frac{1}{n} \sum_{i=0}^{i<n} (y(x_i) - \frac{1}{1 + e^{-\sum_{j=0}^{j<d} w_j x_i, j}})^2$$

Now we can use a version of gradient descent to seek the best weight vector `w`. As we did for the linear regression problems, we will include a learning rate `alpha` that controls how quickly we adjust the weights, and an iteration maximum `kmax`. Then we can try to compute the best weight vector by using the following function:

`logistic_regression(X,y,alpha,kmax)`, where

- `X` is our input data (preferable normalized!), with an initial column of 1's
- `y` is our output data, values of 0 or 1
- `alpha` is a learning rate, which might be 0.1 or 0.01
- `kmax` is the maximum number of iterations

```
def logistic_regression ( x, y, alpha, kmax ):

  import numpy as np

  n, d = x.shape
  w = np.zeros ( d )

  for k in range ( 0, kmax ):
    y2 = 1.0 / ( 1.0 + np.exp ( - np.matmul ( x, w ) ) )

    for j in range ( 0, d ):
      w[j] = w[j] - ( alpha / n ) * np.dot ( ( y2 - y ), x[:,j] )

  return w
```

6 Exercise 2:

Write a code `exercise2.py` for this experiment.

1. Set `data` by reading the file *gold_data.txt*.
2. Set `g` and `y` to be the values in columns 0 and 1 of `data`.
3. Compute and save the values `gmax` and `gmin`.
4. Use these values to create `gn`, a normalized copy of `g`;
5. Create the $n \times 2$ array `X`, so column 0 is equal to 1, and column 1 is equal to `gn`;
6. Choose a learning rate `alpha` and iteration maximum `kmax`;
7. Call `logistic_regression()` to get values for the weights `w`;
8. Make plot data `gp = np.linspace ( -1, 2, 101 )` and `yp = 1/(1+w[0]+w[1]*gp)`;
9. Plot original data `(gn,y)` as dots, and plot data `(gp,yp)` as a red line;
10. If your red line does not approximately separate the real and fake coins, try a smaller `alpha` or a larger `kmax`;
11. The cutoff for your normalized data is `cn = -w[0] / w[1]`; Print this value;
12. The cutoff for your original data is `c = gmin - (gmax - gmin) * w[0] / w[1]`; Print this value;
13. Looking at the cutoff `c`, is a coin weighing $g = 33.3$ grams likely to be fake or legitimate?
14. Our logistic formula is evaluated using `gn`, that is, the normalized coin weight. What is `gn`, the normalized coin weight for your coin of $g = 33.3$ grams? What is the value of y(gn)?;

7 Prepare for Exercise 3:

The manager of an office complex has taken $n = 44$ measurements of humidity, h, and temperature t, and asked the office workers if they find the environmental comfortable ($y = 1$) or uncomfortable ($y = 0$), storing this information as rows of the dataset *comfort_data.txt*.

In the future, the manager wants a way to be able to predict, in advance, whether a given combination of humidity and temperature will make the employees uncomfortable. For that purpose, a formula is needed. We would like to use logistic regression, creating a formula that is given values (h, t) and returning a value y which we can regard as the "probability" that the employees are happy.

Because we have two variables to worry about, there will not be a single cutoff value. Instead, we expect to see a dividing line which will try to separate the data into comfortable and uncomfortable zones. And of course, we don't expect the result to be perfect. But we would be satisfied if we got a simple formula that pretty well summarized the situation.

We assume that an array X is created, with n rows and 3 columns. Each row is a sample data item $x = [1|h|t]$. Just as with our linear regression problem, we have a vector y, which in this case will only include the values 0 and 1. But instead of the formula $y = x'c$, our logistic regression formula will be

$$y(x) = \frac{1}{1 + e^{-w_0 * x_0 - w_1 * x_1 - w_2 * x_2}} = \frac{1}{1 + e^{-w_0 * 1 - w_1 * h - w_2 * t}}$$

As we said, instead of a cutoff value, we have a dividing line, the values of h and t for which $y(x) = \frac{1}{2}$. Interestingly enough, this implies

$$\begin{aligned} y(x) &= \frac{1}{2} \\ \frac{1}{2} &= \frac{1}{1+e^{-w_0*x_0-w_1*x_1-w_2*x_2}} \\ 2 &= 1+e^{-w_0*x_0-w_1*x_1-w_2*x_2} \\ 1 &= e^{-w_0*x_0-w_1*x_1-w_2*x_2} \\ 0 &= -w_0*x_0-w_1*x_1-w_2*x_2 \end{aligned}$$

And this final expression is actually a formula for a straight line that divides the data. If we think of t as the dependent variable:

$$t = \frac{-w_0 - w_1 * h}{w_2} \tag{Dividing line}$$

For this exercise, instead of calling our function `logistic_regression()`, we will work with a more powerful function from Scikit-Learn.

8 Exercise 3:

Create a file *exercise3.py* for this experiment.

1. Set `data` by reading the file *comfort_data.txt*;
2. Think of the three columns as representing `[h|t|y]`;
3. Create the three columns of `X` as `[1|h|t]`;
4. Plot your `(h,t)` data using the commands

```
plt.plot ( X[y==0,1], X[y==0,2], 'r.', markersize = 15 )
plt.plot ( X[y==1,1], X[y==1,2], 'b.', markersize = 15 )'
```

5. Import the logistic regression classifier and use it on `X, y`:

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state=0).fit ( X, y )
```

6. Use the classifier formula to reclassify the data we already have:

```
yp = classifier.predict ( X )
```

7. Collect the regression coefficients:

```
w = classifier.coef_[0].copy()
w[0] = classifier.intercept_
```

8. Compute the endpoints of the dividing line as follows:

```
tmin = np.min ( t )
tmax = np.max ( t )
hmin = ( - w[0] - w[2] * tmin ) / w[1]
hmax = ( - w[0] - w[2] * tmax ) / w[1]
```

9. Replot your original `(h,t)` data using the commands

```
plt.plot ( X[y==0,1], X[y==0,2], 'r.', markersize = 15 )
plt.plot ( X[y==1,1], X[y==1,2], 'b.', markersize = 15 )
plt.plot ( [hmin, hmax], [tmin,tmax], 'k-', linewidth = 3 )
```

10. Plot your fitted `(h,t)` data using the commands

```
plt.plot ( X[yp==0,1], X[yp==0,2], 'r.', markersize = 15 )
plt.plot ( X[yp==1,1], X[yp==1,2], 'b.', markersize = 15 )
plt.plot ( [hmin, hmax], [tmin,tmax], 'k-', linewidth = 3 )
```

11. What is the difference between your last two plots?
12. Looking at your final plot, would employees be comfortable at a setting of $h = 50, t = 80$? What is the value of $y(h,t)$ in this case?

9 Prepare for Exercise 4:

The file *caesarian_data.txt* contains $n = 80$ records; each record lists 6 values used by a senior doctor to determine whether or not to recommend that a baby be delivered by Caesarian section:

1. **age**, patient's age in years;
2. **num**, number of previous deliveries by patient;
3. **tim**, delivery date (0=timely, 1 = premature, 2 = late);
4. **pre**: blood pressure (0=low, 1=normal, 2=high);
5. **hrt**: heart (0=healthy, 1=unhealthy);
6. **cae**: decision (0=normal delivery, 1=use Caesarian);

This doctor has left the hospital, and it is now up to you to make these decisions. In other words, given values of `x=(1, age, num, tim, pre, hrt)`, will you recommend normal delivery `(cae=0)` or Caesarian delivery `(cae=1)`?

We plan to use logistic regression, to find a formula that will look like

$$y(x) = \frac{1}{1 + e^{-w_0*1-w_1*age-w_2*num-w_3*tim-w_4*pre-w_5*hrt}}$$

so that we will recommend a Caesarian delivery whenever $\frac{1}{2} \leq y(x)$.

Given so many independent variables, it will not be possible to create a simple plot that illustrates our formula versus the senior doctor's original recommendations. Instead, we may try to come up with some other measures of error.

10 Exercise 4:

Write a code *exercise4.py* for this experiment.

1. Set `data` by reading the file *caesarian_data.txt*;
2. Set `X` as the array of the form `[1, age, num, tim, pre, hrt]`;
3. Set `y` as the vector of `cae` values;
4. Import the logistic regression classifier and use it on `X, y`;
5. Use the classifier formula to reclassify the data we already have, creating a vector `yp`;
6. Compute the MSE error by as the average of the sum of the squares of `(y-yp)`;
7. Report this MSE value;

8. Collect the regression coefficients in the vector `w`;
9. Print the coefficients `w`;
10. Use the coefficient weights to evaluate three new cases, and print your recommendations:

```
age num tim pre hrt cae
 29   2   2   0   1  ?
 33   2   0   2   0  ?
 22   0   1   1   0  ?
```