Logistic Regression

MATH1900: Machine Learning

Location: http://people.sc.fsu.edu/~jburkardt/classes/ml_2019/logistic_regression/logistic_regression.pdf

How to separate the sheep from the goats

The Logistic Regression Problem

We need to make a Yes/No decision y based on the values of data x. We have n previous examples of such decisions. We would also like to estimate how likely our decision is to be correct.

A doctor must decide whether a baby should have a regular deliver or Caesarian section. While every patient is different, the doctor has many previous case histories and the decisions made there. The doctor wishes to make a decision based on this data. If the patient has concerns, the doctor wants to be able to say how strong the recommendation is.

This is an example of a **classification problem**. Our task is to look at a particular case, and decide which class it belongs to. In the *logistic regression* problem, there are only 2 such classes, which might be Yes/No, 0/1, Admit/Reject, Pass/Fail. Since our action will be based on previous examples for which the decision has already been made, this is an example of *supervised learning*.

1 A logistic function

To begin with, let us suppose that the output decision y is to made based on the value of just a single input x. Our decision will be simplified if we can somehow come up with a function $y(x)$ that automatically returns a reasonable Yes or No value. The simplest such function might simply apply some cutoff value x^*:

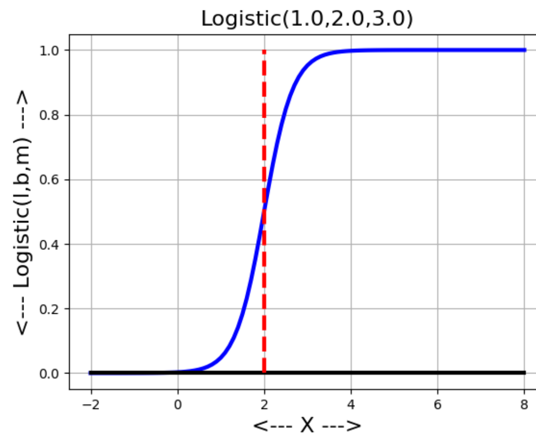
$$y(x) = \begin{cases} \text{No} & \text{if } x \leq x^* \\ \text{Yes} & \text{if } x^* < x \end{cases}$$

However, this approach does not give us the additional information of how sure we are of our classification, and it does not generalize easily to situations involving multiple input variables x.

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), m (slope) and b (cutoff). For our purposes, we will always set $l = 1$.



The logistic curve for $l = 1, b = 2, m = 3$.

The function $y(x)$ is near 0 for values to the left of the cutoff, hits $\frac{1}{2}$ at the cutoff value, and then rises to 1 on the right. The location of the cutoff is controlled by the value of b. The sharpness of the rise, and the width of the "uncertainty" region, depend on the slope m.

For a given problem, if we can determine good values of b and m, then we will have a way of classifying our data, as well as reporting how sure we are of our decision:

$$y(x) = \begin{cases} \text{No! (90\% sure)} & \text{if } y(x) <= 0.10 \\ \text{No} & \text{if } 0.10 < y(x) < \frac{1}{2} \\ \text{50/50} & \text{if } y(x) = \frac{1}{2} \\ \text{Yes} & \text{if } \frac{1}{2} < y(x) < 0.90 \\ \text{Yes! (90\% sure)} & \text{if } 0.90 < y(x) \end{cases}$$

2 Exercise: Plot the logistic function

Download the file *logistic.py*. Consider the logistic function, with parameters $l = 1, b = 2, m = 3$, over the range $-2.0 \leq x \leq 8.0$.

```
from logistic import logistic

l = 1.0
b = 2.0
m = 3.0
x = np.linspace ( -2, 8, 11 )
y = logistic ( l, b, m, x )
print ( y )
```

Listing 1: Evaluate the logistic function.

```
1           x     y=logistic(1,2,3,x)
2
3          -2     0.000006
4          -1     0.000123
5           0     0.002472
6           1     0.047425
7           2     0.500000
8           3     0.952574
9           4     0.997527
10          5     0.999877
11          6     0.999994
12          7     1.000000
13          8     1.000000
```

Listing 2: Sample logistic function values.

The file *logistic.py* includes the function `logistic(l,b,m,x1,x2)`, which plots the logistic function over the interval $[x1, x2]$:

```
1     from logistic import logistic_plot
2
3   #  Vary the cutoff b
4     logistic_plot ( 1.0, 1.0, 3.0, -2.0, 8.0 )
5     logistic_plot ( 1.0, 2.0, 3.0, -2.0, 8.0 )
6     logistic_plot ( 1.0, 3.0, 3.0, -2.0, 8.0 )
7
8   #  Vary the slope m
9     logistic_plot ( 1.0, 2.0, 1.0, -2.0, 8.0 )
10    logistic_plot ( 1.0, 2.0, 2.0, -2.0, 8.0 )
11    logistic_plot ( 1.0, 2.0, 3.0, -2.0, 8.0 )
```

Listing 3: Plot the logistic function over $-2 < x < 8$ with various values of b and m.

3 Example: Is this a counterfeit coin?

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 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.

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

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

where x is the data for a given case. In our example, x will be a vector $[1|g]$ where g is the (normalized) weight of the coin. The vector w is a set of weights (2 in this example), to be determined by minimizing the

classification error. This minimization can be done by first normalizing the data, and then calling a function that uses gradient descent.

For our work, we will be calling a function called `logistic_regression(x,y,alpha,kmax)`, where

- `x` is our normalized input data, 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

The function uses the gradient descent method, and returns its best estimate for the weights w. Here is how it would be used for our gold coin problem:

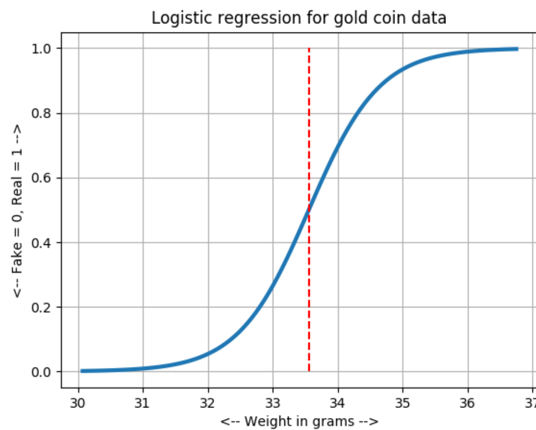
```
import numpy as np

data = np.loadtxt ( 'gold_data.txt' )

g = data[:,0]
y = data[:,1]
m = len ( g )

gmin = np.min(g)
gmax = np.max(g)
g = ( g - np.min ( g ) ) / ( np.max ( g ) - np.min ( g ) )

x = np.zeros ( [ m, 2 ] )
x[:,0] = 1
x[:,1] = g

alpha = 0.02
kmax = 100000

from logistic_regression import logistic regression

w = logistic_regression ( x, y, alpha, kmax )
```

Listing 4: Linear regression for gold coin data.

The weights $w = [-2.311, 4.092]$ are returned for the normalized data values. This means that, if we actually want to compute the value of y for the first data value $g[0]$, we need to do this by computing

```
gn = ( g[0] - gmin ) / ( gmax - gmin )
wtx = w[0] * 1.0 + w[1] * gn
y1 = 1.0 / ( 1.0 + np.exp ( - wtx ) )
```

In the following graph, we display the logistic function for our gold data, using the raw (un-normalized) values of the coin weights. This shows that the cutoff point is about 33.55 grams. Below this, a coin is more than 50% likely to be fake.



The logistic regression curve for the gold coin data.

4 Can we trust the logistic_regression() function?

We will treat the `logistic_regression()` function as a black box. It uses gradient descent, and we have already said that it is best for a gradient descent function that the data is normalized. Otherwise, badly scaled data can make it difficult to get convergence.

The variable **alpha** is a "learning rate", a kind of step control variable that we saw in the gradient descent lab. For our problems, using a value of around 0.01 to 0.1 may be reasonable.

The variable **kmax** controls the number of steps of gradient descent to be carried out. For our problems, a value of 10,000 might be reasonable.

Because this function does not include any convergence tests, we have to guess whether or not it has done its job. One way to check that **kmax** is about the right size is to compare the results when you double it or cut it in half. If the results change significantly, this can indicate that more iterations would improve the accuracy. Similarly, if double or halving **alpha** makes a noticeable change in the results, this may indicate an issue with the convergence.

```
1   alpha = 0.02
2   kmax = 100000
3   w = logistic_regression ( x, y, alpha, kmax )
4   print ( w )
5
6   alpha = 0.02
7   kmax = 10000    # Try 1/10 the number of iterations
8   w = logistic_regression ( x, y, alpha, kmax )
9   print ( w )
10
11  alpha = 0.02
12  kmax = 200000   # Try twice the number of iterations
13  w = logistic_regression ( x, y, alpha, kmax )
14  print ( w )
```

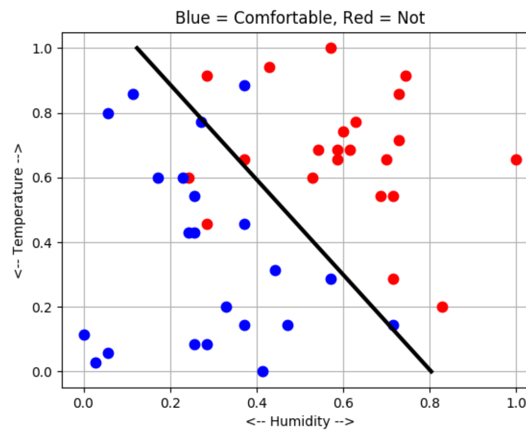
Listing 5: Is kmax about right?

5 Example: The comfort zone

The manager of an office complex has taken $m = 44$ measurements of humidity, h, and temperature t, and asked the office workers if they find the environmental comfortable ($y = 1$) or uncomfortable ($y = 0$).

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.

Now we have two input variables to work with. We normalize the input values, create an $m \times 3$ array x containing $[1|h|t]$ and call `logistic_regression()`.



The logistic regression dividing line for the comfort data.

Instead of a cutoff value, we now have a cutoff line, of the form $w'x = 0$ or $w_0\,1 + w_1\,\hat{h} + w_2\,\hat{t} = 0$. Values to the left of this line represent comfortable mixtures of humidity and temperature. This plot was made using the normalized data $\hat{h}, \hat{t}$.

```
 1  import matplotlib.pyplot as plt
 2  import numpy as np
 3  from logistic_regression import logistic_regression
 4
 5  data = np.loadtxt ( 'comfort_data.txt' )
 6
 7  h = data[:,0]
 8  t = data[:,1]
 9  y = data[:,2]
10  m = len ( h )
11
12  h = ( h - np.min ( h ) ) / ( np.max ( h ) - np.min ( h ) )
13  t = ( t - np.min ( t ) ) / ( np.max ( t ) - np.min ( t ) )
14
15  x = np.zeros ( [ m, 3 ] )
16  x[:,0] = 1
17  x[:,1] = h
18  x[:,2] = t
19
20  alpha = 1.0
21  kmax = 10000
22
23  w = logistic_regression ( x, y, alpha, kmax )
24
25  print ( '' )
26  print ( '  Estimated weights W = (%g,%g,%g)' % ( w[0], w[1], w[2] ) )
```

Listing 6: Linear regression for comfort data

6 Exercise: 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 (eng), on a math exam, (mat), and an admissions decision y, which is 0 (not admitted) or 1 (admitted). Our task is to find a logistic formula which, given eng and mat, can give a y value between 0 and 1, estimating the probability that the student will be admitted.

We will read the data, normalize it, pack the input values into an x array containing $[1|eng|mat]$, and call `logistic_regression()` to get weights w.

```
import numpy as np
from logistic_regression import logistic_regression
data = np.loadtxt ( 'admit_data.txt' )
eng = data[:,0]
mat = data[:,1]
admit = data[:,2]
m = len ( eng )

eng = ( eng - np.min ( eng ) ) / ( np.max ( eng ) - np.min ( eng ) )
mat = ( mat - np.min ( mat ) ) / ( np.max ( mat ) - np.min ( mat ) )

x = np.zeros ( [ m, 3 ] )
x[:,0] = 1
x[:,1] = eng[:]
x[:,2] = mat[:]

y = admit[:]

alpha = 0.01
kmax = 10000

w = logistic_regression ( x, y, alpha, kmax )
```

Listing 7: Linear regression for admissions data.

Three students apply late. We will use the weights w to decide whether to admit them. The raw data is:

```
eng  mat
 46   81
 26   75
 30   66
```

We need to use the normalized versions of these values, which are

```
  eng    mat
 0.66   0.88
 0.22   0.74
 0.31   0.55
```

We evaluate the exponent of the logistic function, and then the logistic function itself:

```
x1 = np.array ( [ 1.0, 0.66, 0.88 ] )
wx = np.dot ( w, x1 )
y1 = 1.0 / ( 1.0 + np.exp ( - wx ) )
print ( 'Student 1: wx = ', wx, '  Y = ', y1 )
```

Listing 8: Evaluate formula for student #1

A y value near 0.5 is hard to decide, but values above that will signify admission, and lower values rejection.

7 Computing Assignment #7

The file *caesarian_data.txt* contains 80 records; each record lists 6 values used by a 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);

Write a program *hw7.py* which:

1): Reads the caesarian data file, normalizes the first 5 data items, creates an 80×6 data matrix x containing $[1|age|num|tim|pre|hrt]$, calls `logistic_regression()` to estimate the weights w, and prints w.

2): Uses the weights w to evaluate three new cases. The raw values are:

```
1   age  num  tim  pre  hrt
2    29    2    2    0    1
3    33    2    0    2    0
4    22    0    1    1    0
```

but when you evaluate the formula, you must use the normalized data values:

```
1   age   num   tim   pre   hrt
2  0.52   0.5   1.0   0.0   1.0
3  0.70   0.5   0.0   1.0   0.0
4  0.22   0.0   0.5   0.5   0.0
```

For new cases #1, 2, 3, return the values of $w' x$ (the exponent in the logistic function), $y(x)$ (a number between 0 and 1), and your judgement of whether the patient should undergo a Caesarian operation: 'probably', 'unsure', 'probably not'.

Email a copy of your program, your results, and your decisions to Dr Schneier **mhs64@pitt.edu** by Wednesday, 30 October.