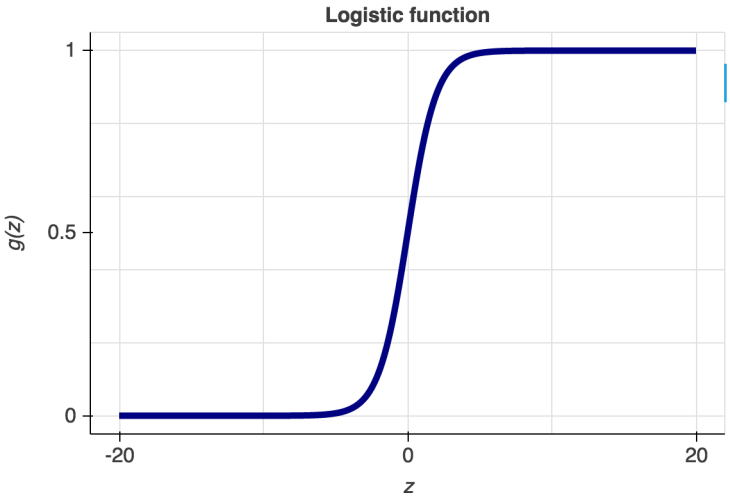
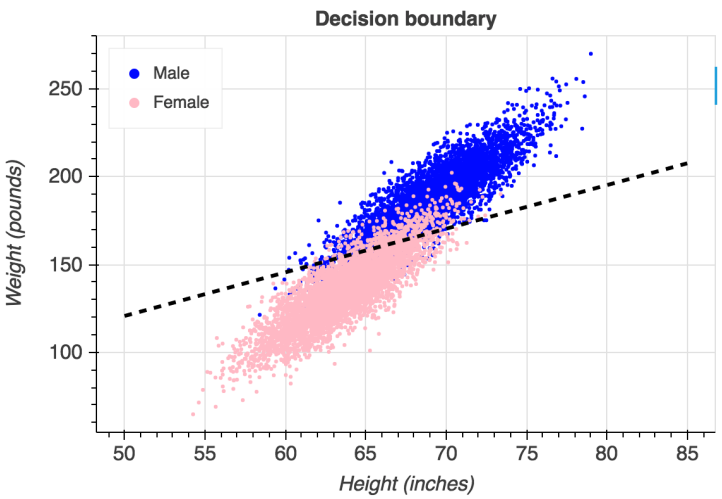
Understanding Logistic Regression step by step

Training a logistic regression classifier to predict people's gender based on their weight and height.

Gustavo Chávez Follow
Feb 21, 2019 · 6 min read





Logistic Regression is a popular statistical model used for binary classification, that is for predictions of the type *this or that*, *yes or no*, *A or B*, etc. Logistic regression can, however, be used for multiclass classification, but here we will focus on its simplest application.

As an example, consider the task of **predicting someone's gender** (Male/Female) based on their **Weight** and **Height**.

For this, we will *train* a machine learning model from a data set of 10,000 samples of people's weight and height. The data set is taken from the Conway & Myles Machine Learning for Hackers book, Chapter 2, and can it can be directly downloaded here.

This is a preview of what the data looks like:

```
// heights_weights.csv

"Height","Weight","Male"
73.847017017515,241.893563180437,1
68.7819040458903,162.3104725213,1
74.1101053917849,212.7408555565,1
71.7309784033377,220.042470303077,1
69.8817958611153,206.349800623871,1
67.2530156878065,152.212155757083,1
68.7850812516616,183.927888604031,1
68.3485155115879,167.971110489509,1
67.018949662883,175.92944039571,1
63.4564939783664,156.399676387112,1
...
63.1794982498071,141.266099582434,0
62.6366749337994,102.85356321483,0
62.0778316936514,138.691680275738,0
60.0304337715611,97.6874322554917,0
59.0982500313486,110.529685683049,0
66.1726521477708,136.777454183235,0
67.067154649054,170.867905890713,0
63.8679922137577,128.475318784122,0
69.0342431307346,163.852461346571,0
61.9442458795172,113.649102675312,0
```

Each sample contains three columns: *Height*, *Weight*, and *Male*.

- *Height* in inches
- *Weight* in pounds
- *Male:* 1 means that the measurement corresponds to a male person, and 0 means that the measurement corresponds to a female person.

There are 5,000 samples from males, and 5,000 samples for females, thus the data set is balanced and we can proceed to training.

The Python's scikit-learn code to *train* a logistic regression classifier and make a *prediction* is very straightforward:

```
import numpy as np
from sklearn import linear_model

# Load data
data = np.loadtxt('./heights_weights.csv', delimiter=',', skiprows=1)
X = data[:,0:2]
y = data[:,2]

# Fit (train) the Logistic Regression classifier
```

```
clf = linear_model.LogisticRegression(C=1e40, solver='newton-cg')
fitted_model = clf.fit(X, y)

# Predict
prediction_result = clf.predict([(70,180)])
```

LogisticRegression.py hosted with ♡ by GitHub view raw

The general workflow is:

1. get a dataset
2. train a classifier
3. make a prediction using such classifier

Logistic regression hypothesis

The logistic regression classifier can be derived by analogy to the ***linear regression hypothesis*** *which is*:

$$h_\theta(\mathbf{x}) = \theta^\top \mathbf{x}$$

Linear regression hypothesis.

However, the logistic regression hypothesis *generalizes* from the linear regression hypothesis in that it uses the ***logistic function***:

$$h_\theta(\mathbf{x}) = g(\theta^\top \mathbf{x})$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

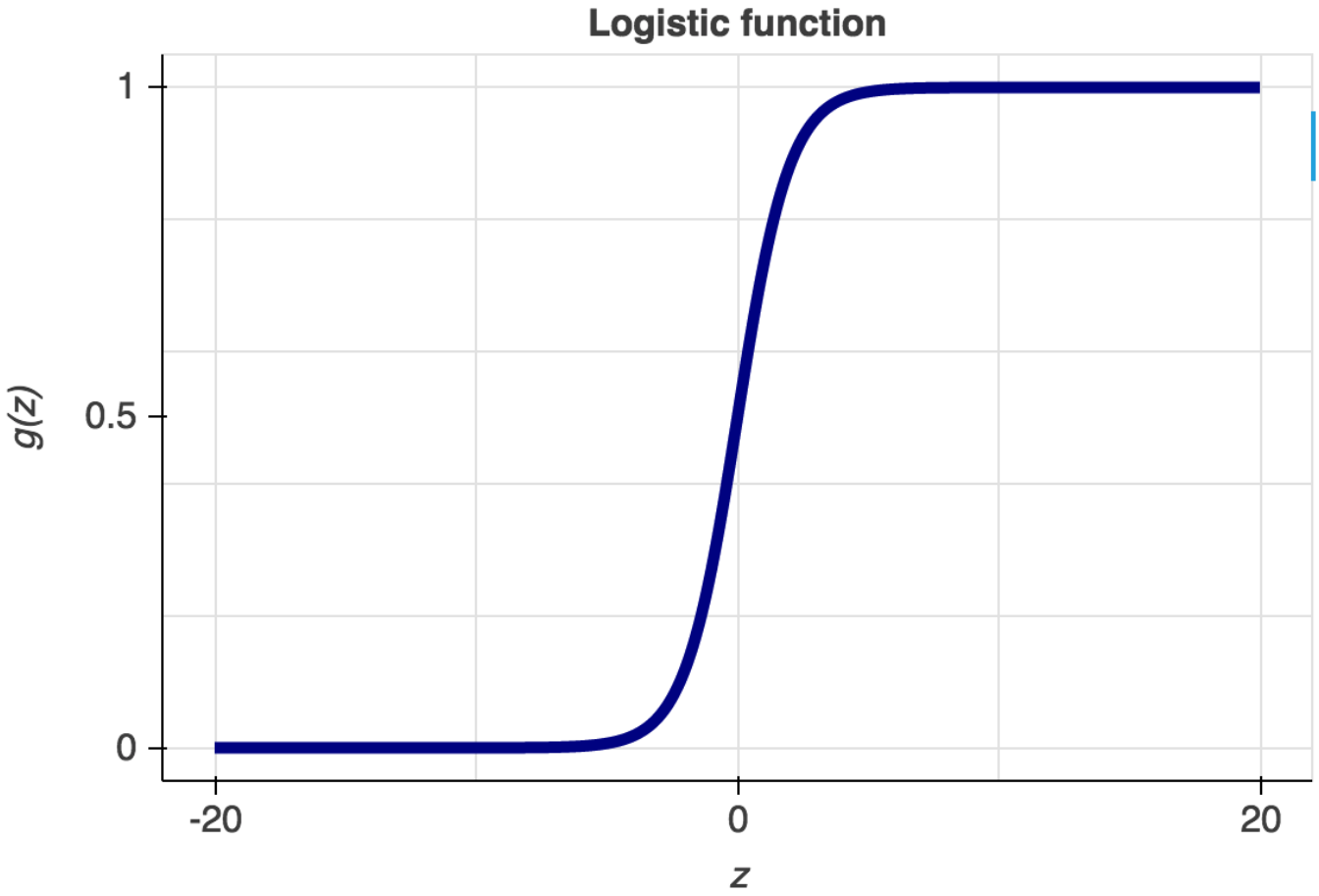
The result is the logistic regression hypothesis:

$$h_\theta(\mathbf{x}) = \frac{1}{1 + e^{-\theta^\top \mathbf{x}}}$$

Logistic regression hypothesis.

The function **g(z)** is the **logistic function**, also known as the *sigmoid function*.

The logistic function has asymptotes at 0 and 1, and it crosses the y-axis at 0.5.



Logistic function.

Logistic regression decision boundary

Since our data set has two features: height and weight, the logistic regression hypothesis is the following:

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

The logistic regression classifier will predict "Male" if:

$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 \geq 0$$

This is because the logistic regression "*threshold*" is set at g(z)=0.5, see the plot of the logistic regression function above for verification.

For our data set the values of θ are:

$$\theta = \begin{bmatrix} 0.69254 \\ -0.49269 \\ 0.19834 \end{bmatrix}$$

To get access to the θ parameters computed by scikit-learn one can do:

```
# For theta_0:
print( fitted_model.intercept_ )
# For theta_1 and theta_2:
print( fitted_model.coef_ )
```

With the coefficients at hand, a *manual* prediction (that is, without using the function **clf.predict()**) would simply require to compute the vector product

$$\theta^\top x$$

and to check if the resulting scalar is bigger than or equal to zero (to predict Male), or otherwise (to predict Female).

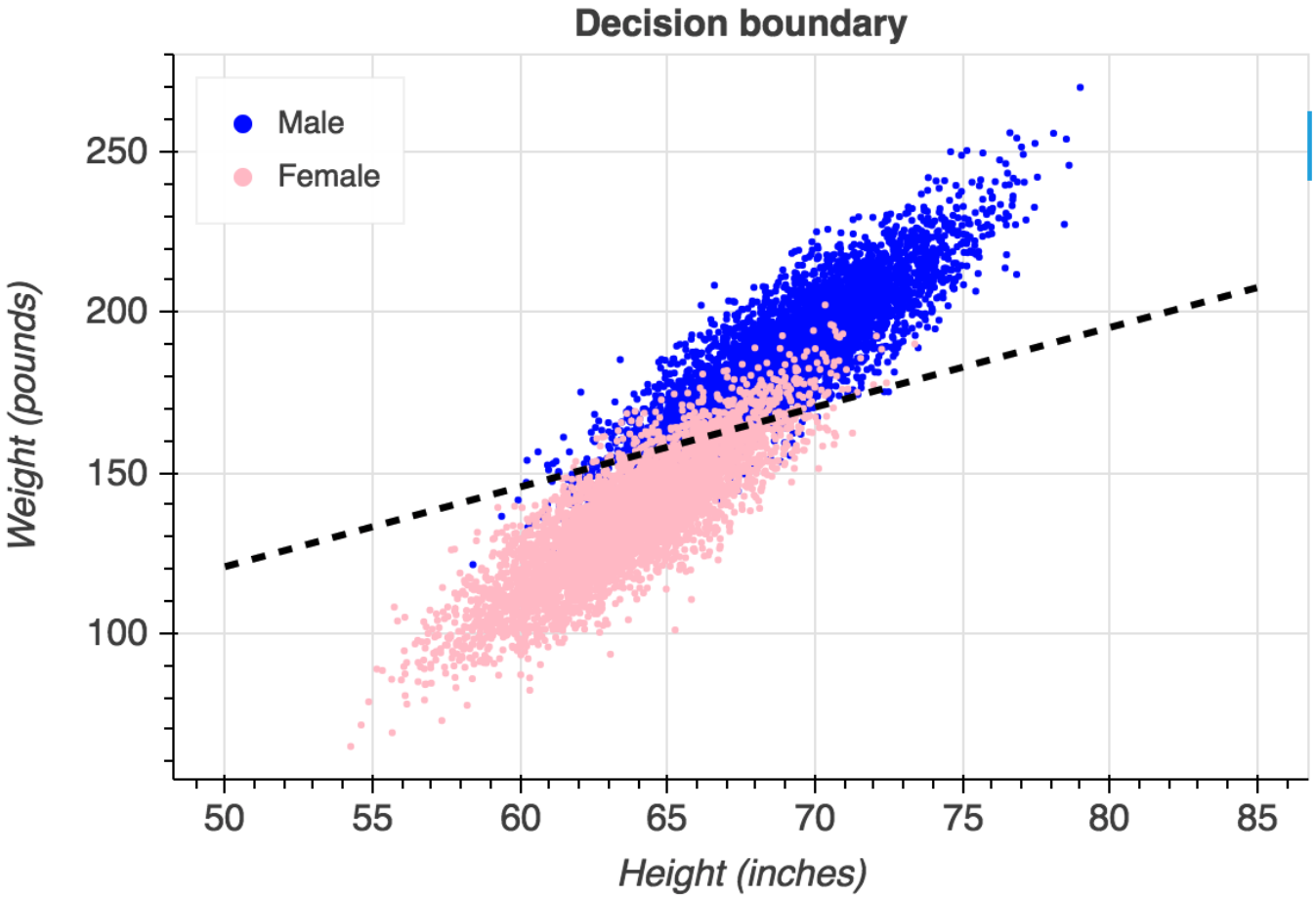
As an example, say we want to predict the gender of someone with ***Height=70 inches*** and ***Weight = 180 pounds***, like at line 14 at the script LogisticRegression.py above, one can simply do:

$$y = \begin{pmatrix} 0.69254 & -0.49269 & 0.19834 \end{pmatrix} \begin{pmatrix} 1 \\ 70 \\ 180 \end{pmatrix} = 1.91$$

Making a prediction using the Logistic Regression parameter θ.

Since the result of the product is bigger than zero, the classifier will predict Male.

A visualization of the ***decision boundary*** and the complete data set can be seen here:



As you can see, above the decision boundary lie most of the **blue points that correspond to the Male class**, and below it all the **pink points that correspond to the Female class**.

Also, from just looking at the data you can tell that the predictions won't be perfect. This can be improved by including more features (beyond weight and height), and by potentially using a different decision boundary.

Logistic regression decision boundaries can also be non-linear functions, such as higher degree polynomials.

Computing the logistic regression parameter

The scikit-learn library does a great job of abstracting the computation of the logistic regression parameter θ, and the way it is done is by solving an optimization problem.

Let's start by defining the logistic regression cost function for the two points of interest: y=1, and y=0, that is, when the hypothesis function predicts Male or Female.

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if} \quad y = 1 \\ -\log(1 - h_\theta(x)) & \text{if} \quad y = 0 \end{cases}$$

Then, we take a convex combination in **y** of these two terms to come up with the logistic regression cost function:

$$J(\theta) = -[y \log(h_\theta(x)) + (1 - y) \log(1 - h_\theta(x))]$$

$$J(\theta) = -[\, y \log(h_\theta(x)) + (1 - y) \log(1 - h_\theta(x)) \,]$$

Logistic regression cost function.

The logistic regression cost function is convex. Thus, in order to compute θ, one needs to solve the following (unconstrained) optimization problem:

$$\min_\theta J(\theta)$$

There is a variety of methods that can be used to solve this unconstrained optimization problem, such as the 1st order method ***gradient descent*** that requires the gradient of the logistic regression cost function, or a 2nd order method such as ***Newton's method*** that requires the gradient and the Hessian of the logistic regression cost function — this was the method prescribed in the scikit-learn script above.

For the case of gradient descent, the search direction is the negative partial derivative of the logistic regression cost function with respect to the parameter θ:

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{m}(h_\theta(x) - y)\, x$$

Partial derivative of the logistic regression cost function.

In its most basic form, gradient descent will iterate along the negative gradient direction of θ (known as a *minimizing sequence*) until reaching convergence.

$$\begin{aligned}&\text{Iterate until convergence}\\&\{\\&\quad \theta_j = \theta_j - \alpha(h_\theta(x) - y)\, x\\&\}\end{aligned}$$

Prototype of gradient descent.

Notice that the constant α is usually called the *learning rate* or the *search step* and that it has to be carefully tuned to reach convergence. Algorithms such as *backtracking line search* aid in the determination of α.

. . .

In summary, these are the three fundamental concepts that you should remember next time you are using, or implementing, a logistic regression classifier:

> *1. Logistic regression hypothesis*
>
> *2. Logistic regression decision boundary*
>
> *3. Logistic regression cost function*

For a discussion of the Logistic regression classifier applied to a data set with more features (using Python too) I recommend this Medium post of Susan Li.

References and further reading:

- https://github.com/gchavez2/code_machine_learning_algorithms
- Andrew Ng's lectures on Logistic Regression
- scikit-learn's logistic regression class

. . .

I am a postdoctoral fellow at the Lawrence Berkeley National Laboratory, where I work at the intersection of machine learning and high-performance computing.

If you find this article interesting, feel free to say hello over LinkedIn, I'm always happy to connect with other professionals in the field.

And as always: comments, questions, and shares are highly appreciated! ♡

Machine Learning Data Science Python Artificial Intelligence Logistic Regression