Logistic Regression

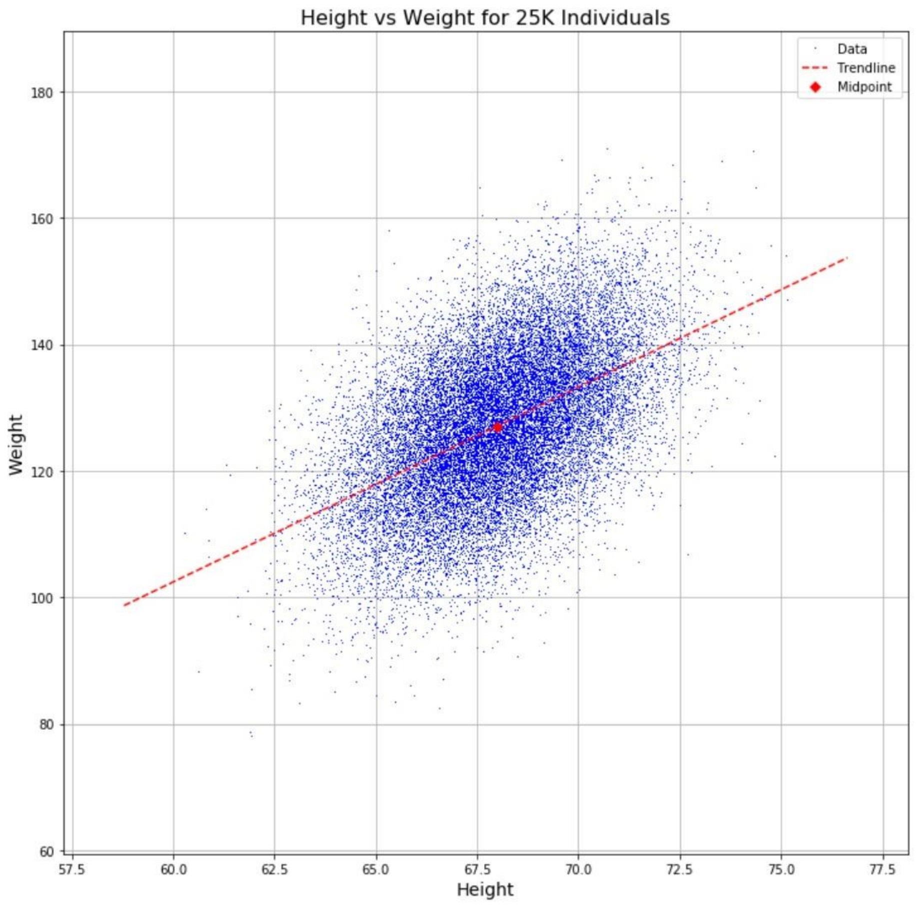
- Motivation: Why Logistic Regression?
- Sigmoid functions and the logit transformation
- Cost functions
- Optimization by Gradient Descent
- Using Logistic Regression to do Machine Learning

Linear Regression and Data Types

Linear Regression relates some number of independent variables

$X_1, X_2, \ldots, X_n$

with a dependent or response variable Y. All are assumed to be **real numbers**:



Linear Regression and Data Types

But probability and statistics deal with many different kinds of data, continuous and discrete, and we have dealt with at least the following during the semester:

- Real Numbers -- Height, weight, time, X ~ N(μ, σ^2), … .

Continuous

- Integers -- Number of emails, Cards, X ~ B(N,p), ….
- Binary – Heads/Tails, Red/Black, X ~ Bern(p), …

And there are other kinds of (discrete) data which statisticians must consider:

Categorical -- A finite list of unordered categories or labels, e.g.,

- Political parties (Republican, Democrat, Independent, Green ….)
- Blood type (A, B, AB, O,….)
- State lived in (MA, VT, ….)

Discrete

Ordinal – Like categorical, but with an explicit ordering, e.g.,

- Class year (freshman, sophomore, …)
- Grades (A, A-, B+, …)
- Likert Scale (disagree strongly, disagree, neutral, agree, agree strongly)

Logistic Regression: The Basic Idea

The regression framework has been adapted to all these kinds of statistical information, but we will consider only the simplest: **binary or Bernoulli data**.

Logistic Regression is a modification of linear regression to deal with binary categories or binary outcomes. It relates some number of independent variables

$$X_1, X_2, \ldots, X_n$$

with a Bernoulli dependent or response variable Y , i.e., $R_Y = \{ 0, 1 \}$. It returns the probability p for $Y \sim \text{Bernoulli}(p)$, i.e., the probability $P(Y = 1)$.

Logistic regression can be used to provide the following kinds of binary outcomes:

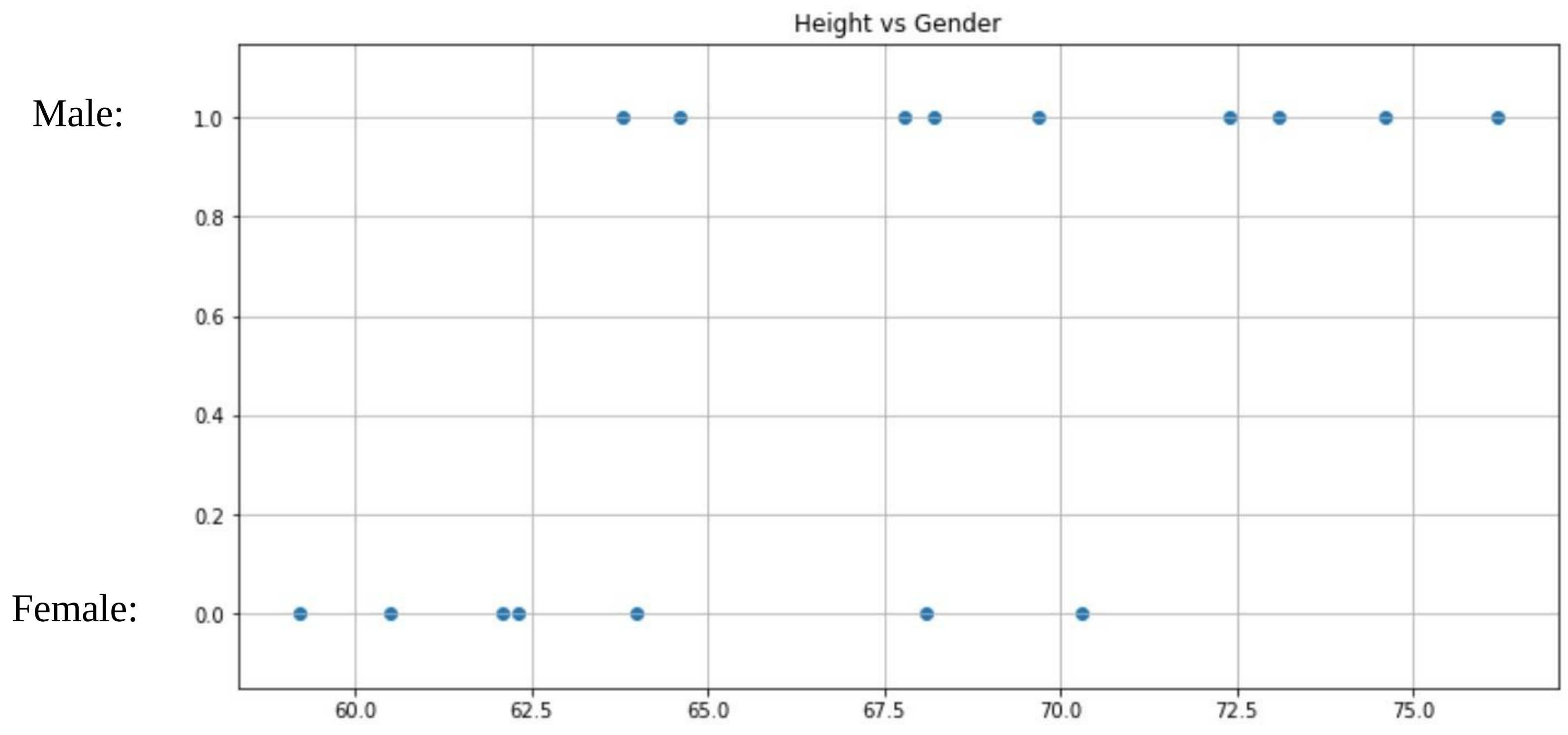
- Given the results of medical tests A, B, C and D, does this patient have cancer?
- Given the credit score, annual salary, gender, and state of residency, will this customer default on the loan he/she is applying for?
- Is this email spam or not?
- Given a student’s homework and midterm grades, will he/she get an A in CS 237?
- Given a person’s height, what is their gender?

Logistic Regression: A Motivating Example

Suppose we consider the last example. Men in general are taller than women (the average height of an American man is 5’ 9” and for women 5’ 4”), but can we use this datum to predict a person’s gender, or give the probability they are one or the other?

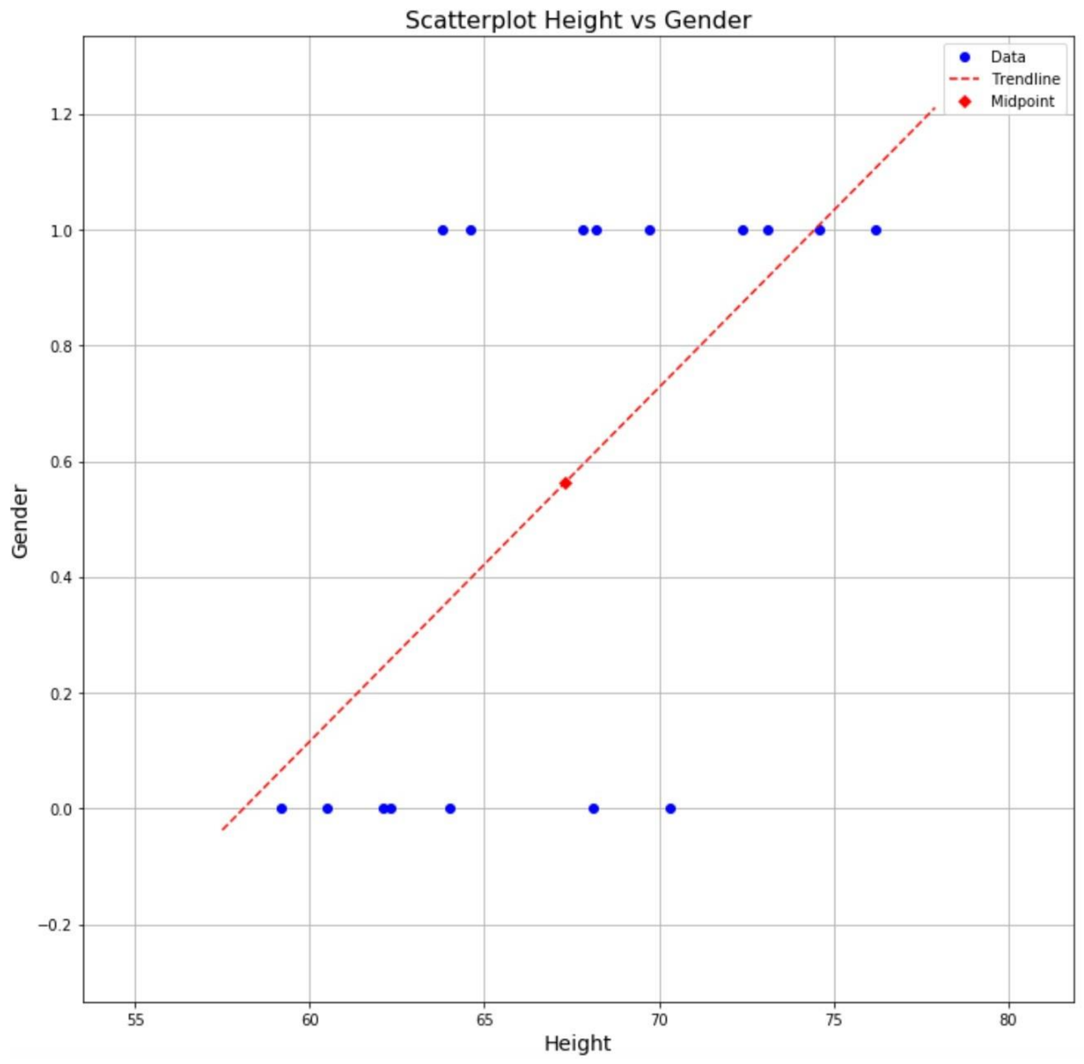
Let’s try it: we proceed to collect data from a random sample of 16 people, and plot X = height against Y = gender (1 for male, 0 for female):

```
Heights: [59.2, 60.5, 62.1, 62.3, 63.8, 64.0, 64.6, 67.8, 68.1, 68.2, 69.7, 70.3, 72.4, 73.1, 74.6, 76.2]
Gender: [0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1]
```



Logistic Regression: Motivating Example

If we plug this into the linear regression algorithm, we get the following:



```
rho:   0.6187   r^2:   0.3828

Residual SS:   2.4303   Explained SS: 1.5072    Total SS:   3.9375

Regression Line: y = 0.0612 * x - 3.554
```

There are many issues with this:

How can we use this to predict someone's gender from their height?

How to give the probability of their gender?

The R^2 value is bound to be very low (here, 0.3828), so how useful is this?

There is clearly no linear trend, so what does the line even mean?

In addition, there are technical and mathematical reasons why linear regression is not appropriate here.

Logistic Regression: The Logit Transformation

In order to solve this, we will use the same idea that we used for non-linear models: we will transform the scale of Y into a new domain, in this case into the real interval [0..1].

This is called the **Logit Transformation**, and is based on the notion of a **sigmoid function** $s : \mathcal{R} \rightarrow [0..1]$ the form

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

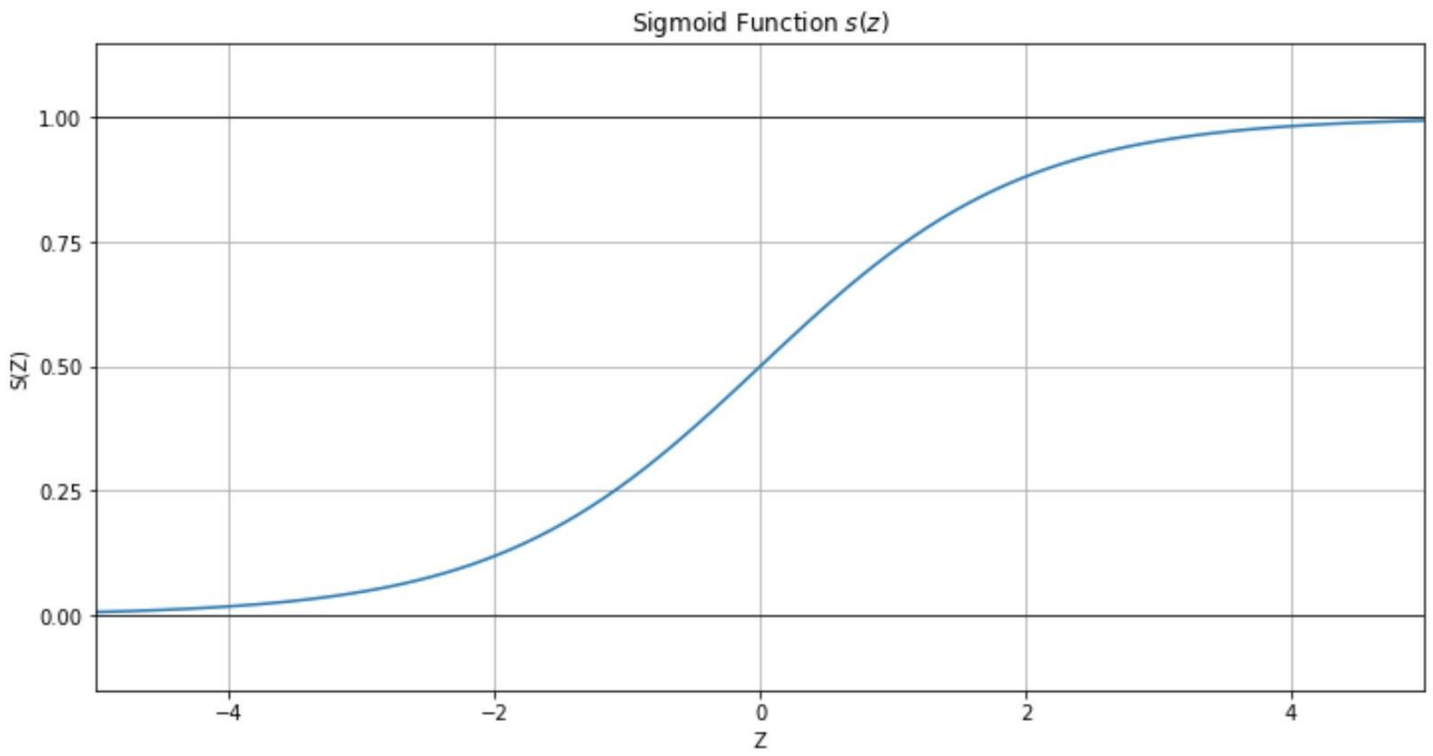
```python
def s(z):
    return 1/(1+np.exp(-z))
```

Notice that:

$$\lim_{z \to \infty} \frac{1}{1 + e^{-z}} = 1$$

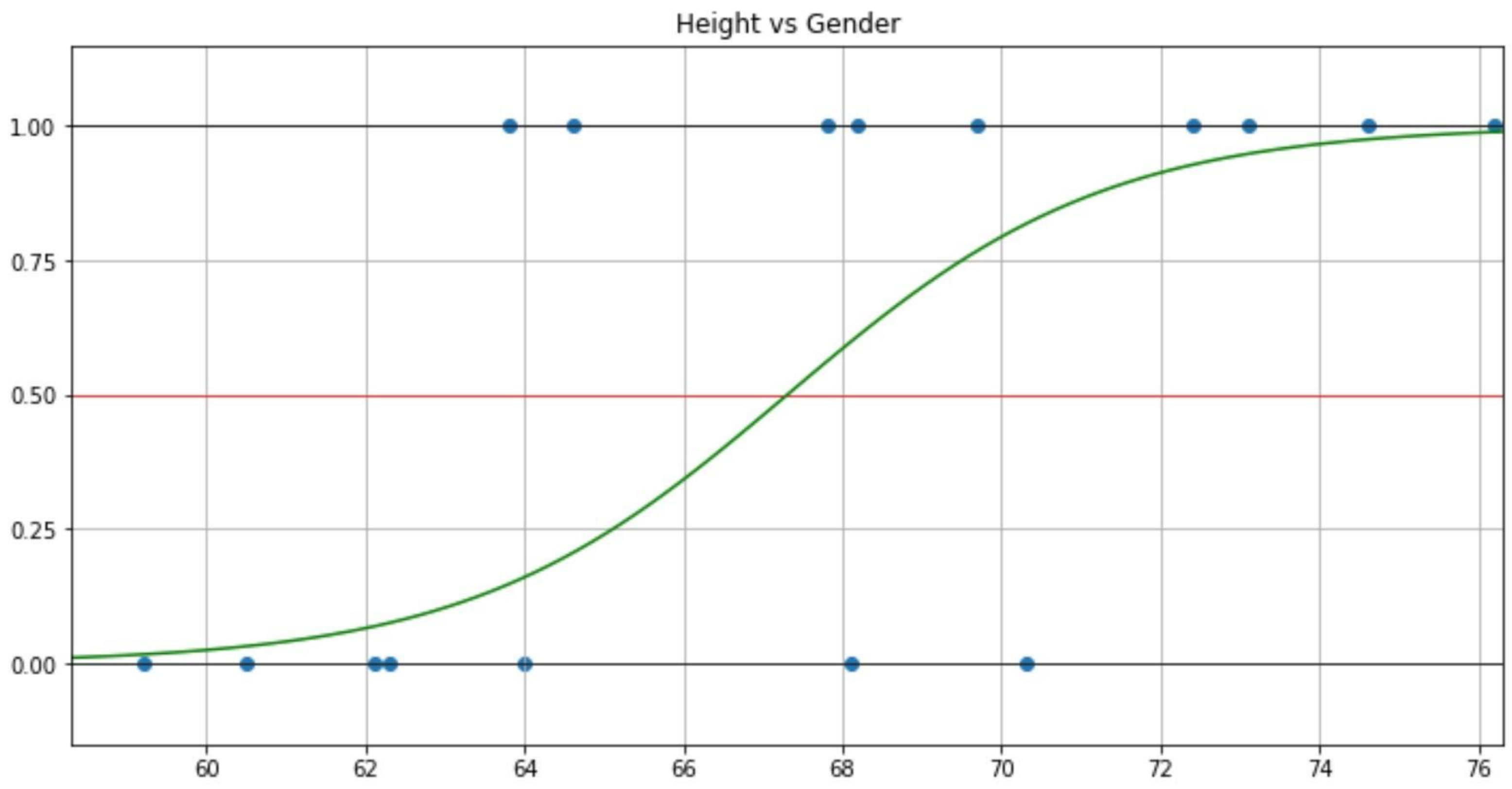
$$\lim_{z \to -\infty} \frac{1}{1 + e^{-z}} = 0$$

$$s(0) = \frac{1}{1 + e^0} = \frac{1}{1 + 1} = 0.5$$



Logistic Regression: The Logit Transformation

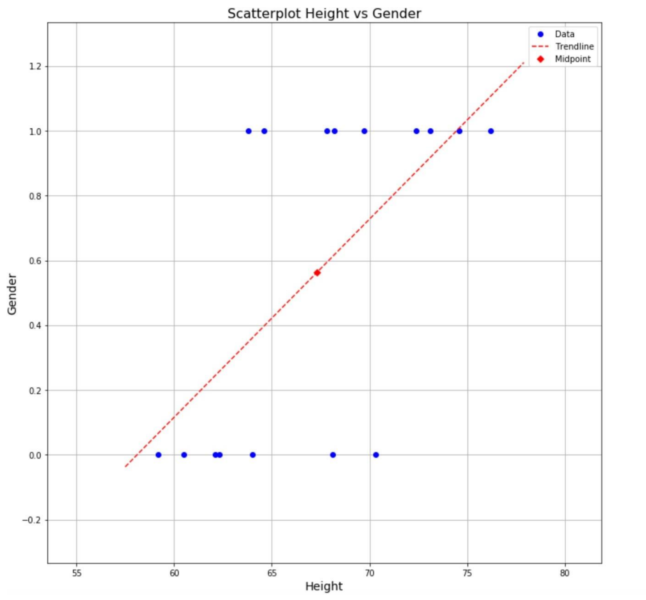
The punchline here is that we will transform the regression line into a sigmoid, and use it to give us the probability that a given individual is male, and then define as a **decision boundary** a threshold (typically 0.5) by which we will decide if the binary output is 1 or 0:



But in fact it is not that simple, because the **least squares technique does not work** any more, and we will have to recast the regression framework around the sigmoid function.....

Logistic Regression: The Logit Transformation

Linear Regression: $-\infty < Y < \infty$



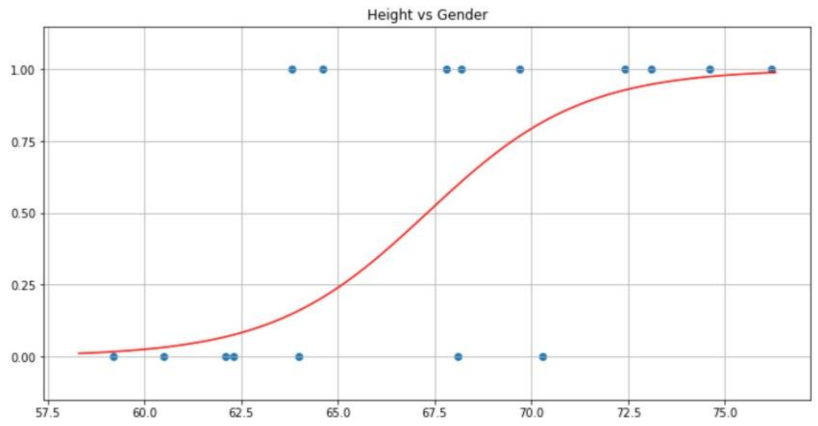
$s : \mathcal{R} \rightarrow [0..1]$

$s^{-1} : [0..1] \rightarrow \mathcal{R}$

$$s^{-1}(p) = ln\left(\frac{p}{1-p}\right)$$

This is called the "logit" or the "log odds ratio."

Logistic Regression: $0 \leq Y \leq 1$



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

$$\begin{aligned}
& y = \frac{e^x}{1+e^x} \\
\Leftrightarrow\ & y + ye^x = e^x \\
\Leftrightarrow\ & y = e^x - ye^x \\
\Leftrightarrow\ & y = (1-y)e^x \\
\Leftrightarrow\ & \frac{y}{(1-y)} = e^x \\
\Leftrightarrow\ & ln\left(\frac{y}{1-y}\right) = x \\
\Leftrightarrow\ & s^{-1}(p) = ln\left(\frac{p}{1-p}\right)
\end{aligned}$$

Linear Regression Redux: Gradient Descent to find $\hat{\theta}_0$nd $\hat{\theta}_1$

In linear regression, we have explicit formulae for finding the parameters for the slope and y-intercept of the regression line which minimizes the mean square error (MSE):

$$\hat{\theta}_1 = \rho(X, Y)\,\frac{\sigma^Y}{\sigma_X}$$

$$\hat{\theta}_0 = \mu_Y - \hat{\theta}_1 \mu_X$$

But what if we didn't? We could then use an iterative approximation algorithm called **Gradient Descent** to find an approximation of the values which minimize the MSE.

Basic idea: Define a **cost or loss function J(...)** which gives the cost or penalty measuring how well the model parameters fit the actual data (high cost = bad fit), and then search for the parameters which minimize this cost.

So let's pretend we didn't have the formulae at the upper right, and suppose we needed to find them by gradient descent. In linear regression this would mean finding values for $\hat{\theta}_0$ and $\hat{\theta}_1$ which minimize the MSE, in other words minimize the cost function:

$$J(\hat{\theta}_0, \hat{\theta}_1) = \underbrace{\frac{1}{N}\sum_{i=1}^{N}(y_i - (\hat{\theta}_0 + \hat{\theta}_1 x_i))^2}_{\text{Cost Function} \\ = \text{MSE}}$$

The **J** in the cost function is used in machine learning and refers to the **J**acobian Matrix.

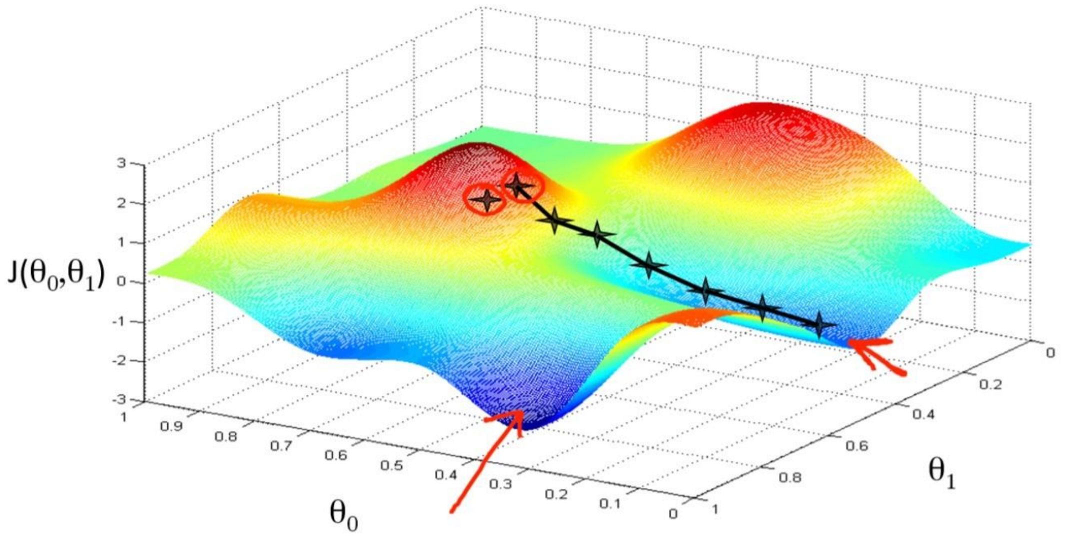
Reference: https://ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html

Linear Regression Redux: Gradient Descent to find $\hat{\theta}_0$nd $\hat{\theta}_1$

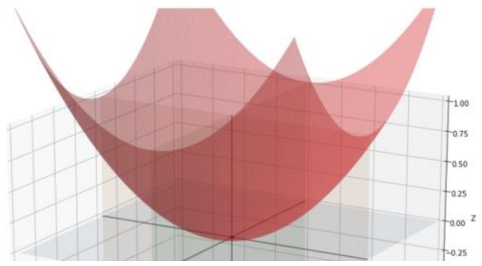
The Gradient Descent Algorithm: A **gradient** is a generalization of a derivative to functions of more than one variable:

> "Like the derivative, the gradient represents the slope of the tangent of the graph of the function. More precisely, the gradient points in the direction of the greatest rate of increase of the function, and its magnitude is the slope of the graph in that direction." - Wikipedia

In gradient descent, we pick a place to start, and move **down** the gradient until we find a minimum point:



When the search space is convex, such as a paraboloid, there will be a single minimum!



Another nice summary: https://hackernoon.com/gradient-descent-aynk-7cbe95a778da

Linear Regression Redux: Gradient Descent to find $\hat{\theta}_0$ nd $\hat{\theta}_1$

To find the minimum value along one axis we will work with only one of the partial derivatives as a time, say the y-intercept:

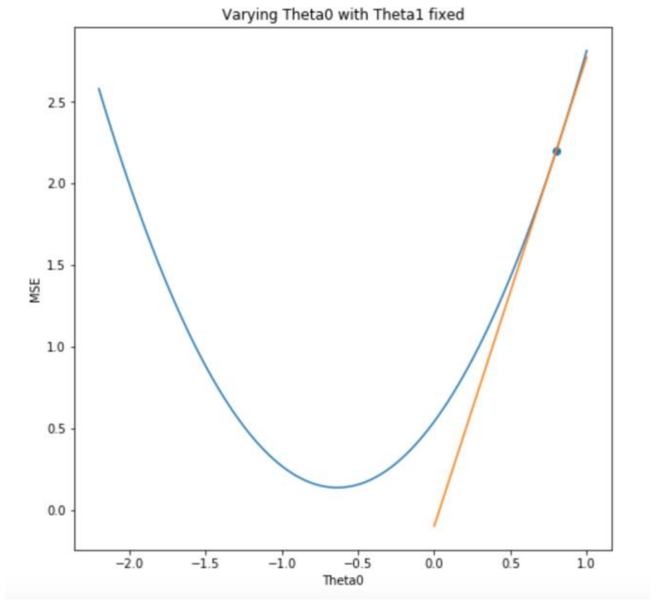
$$J'(b, m) = \begin{bmatrix} \frac{1}{N}\sum_{i=1}^{N} -2(y_i - (b + mx_i)) \\ \frac{1}{N}\sum_{i=1}^{N} -2x_i(y_i - (b + mx_i)) \end{bmatrix}$$

Step One: Choose an initial point b_0.

Step Two: Choose a step size or learning rate λ and threshold of accuracy ε.

Step Three: Move that distance along the axis, in the decreasing direction (the negative of the slope), and repeat until the distance moved is less than ε.

Step Four: Output b_{n+1} as the minimum.



1. Choose b_0;
2. Choose λ;
3. Repeat $b_{n+1} = b_n - J'(b_n \quad) \cdot \lambda$
   Until $| b_{n+1} - b_n | < \epsilon$
4. Output b_{n+1}.

Linear Regression Redux: Gradient Descent to find $\hat{\theta}_0$ ıd $\hat{\theta}_1$

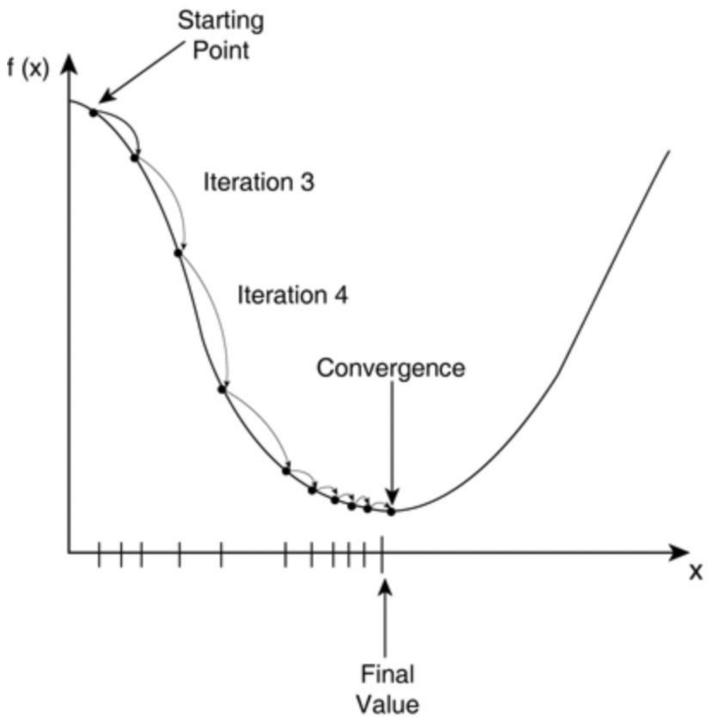
Gradient Descent for Linear Regression:

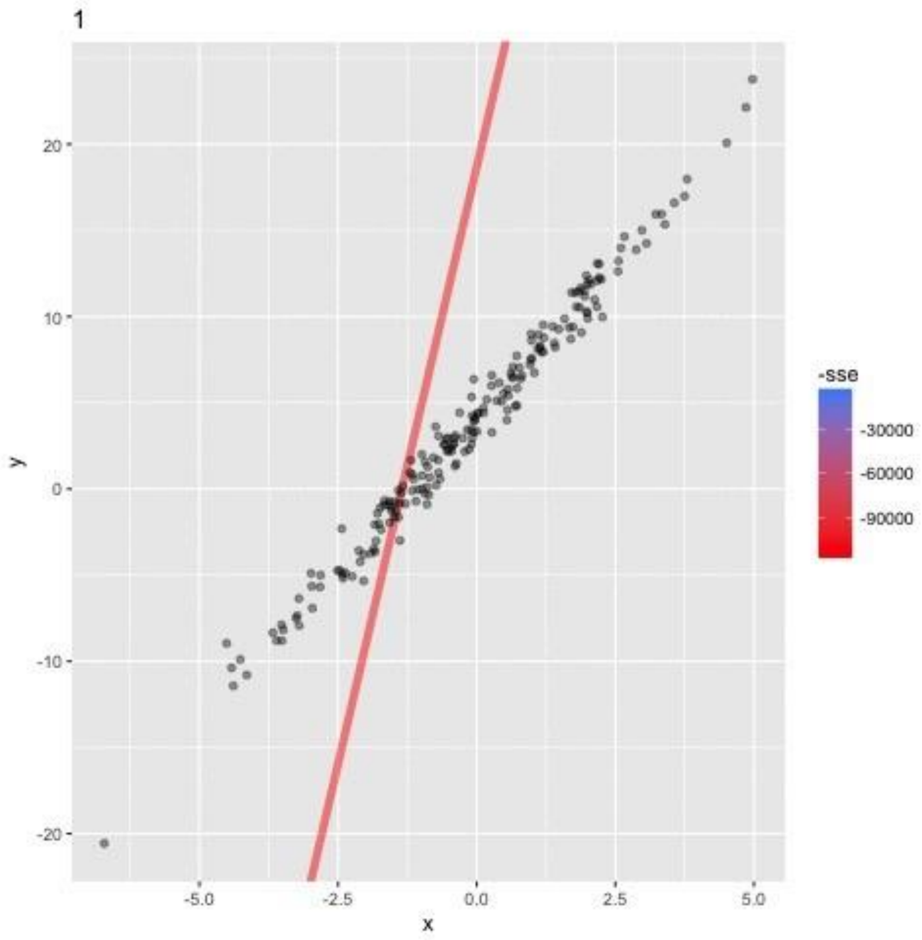
To find a point in multiple dimensions, we simply do all dimensions in the same way at the same time. Here is the algorithm from the reading:

```
def update_weights(m, b, X, Y, learning_rate):
    m_deriv = 0
    b_deriv = 0
    N = len(X)
    for i in range(N):
        # Calculate partial derivatives
        # -2x(y - (mx + b))
        m_deriv += -2*X[i] * (Y[i] - (m*X[i] + b))

        # -2(y - (mx + b))
        b_deriv += -2*(Y[i] - (m*X[i] + b))

    # We subtract because the derivatives point in direction of steepest ascent
    m -= (m_deriv / float(N)) * learning_rate
    b -= (b_deriv / float(N)) * learning_rate

    return m, b
```

Linear Regression Redux: Gradient Descent to find $\hat{\theta}_0$ ıd $\hat{\theta}_1$

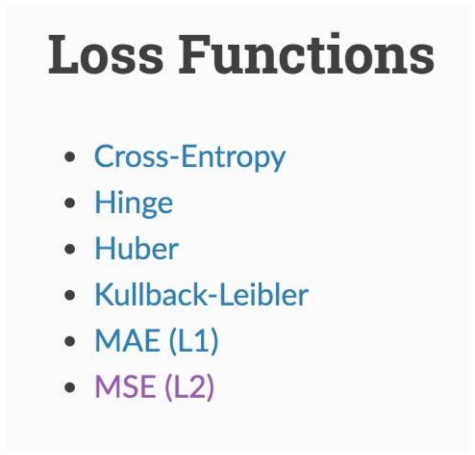
As the parameters are "tuned" to minimize the cost (= measuring how well the parameters fit the model) you get a better and better fit between the model and the data. You can run the gradient descent model as long as you wish to get a better fit. Obviously, defining the cost function and picking the learning rate and threshold are critical decisions, and much research has been devoted to different cost models and different approaches to gradient descent.





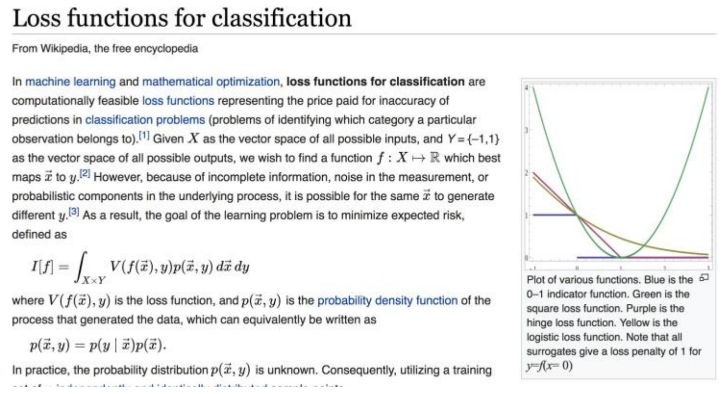
Linear Regression Redux: Gradient Descent to find $\hat{\theta}_0$ ıd $\hat{\theta}_1$

There are dozens of different cost functions that have been defined and even more varients of algorithms that perform minimization of a given cost function available in standard Python libraries:



Loss Functions

- Cross-Entropy
- Hinge
- Huber
- Kullback-Leibler
- MAE (L1)
- MSE (L2)



Loss functions for classification

From Wikipedia, the free encyclopedia

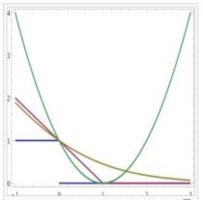
In machine learning and mathematical optimization, **loss functions for classification** are computationally feasible loss functions representing the price paid for inaccuracy of predictions in classification problems (problems of identifying which category a particular observation belongs to).[1] Given X as the vector space of all possible inputs, and $Y = \{-1,1\}$ as the vector space of all possible outputs, we wish to find a function $f : X \mapsto \mathbb{R}$ which best maps $\vec{x}$ to y.[2] However, because of incomplete information, noise in the measurement, or probabilistic components in the underlying process, it is possible for the same $\vec{x}$ to generate different y.[3] As a result, the goal of the learning problem is to minimize expected risk, defined as

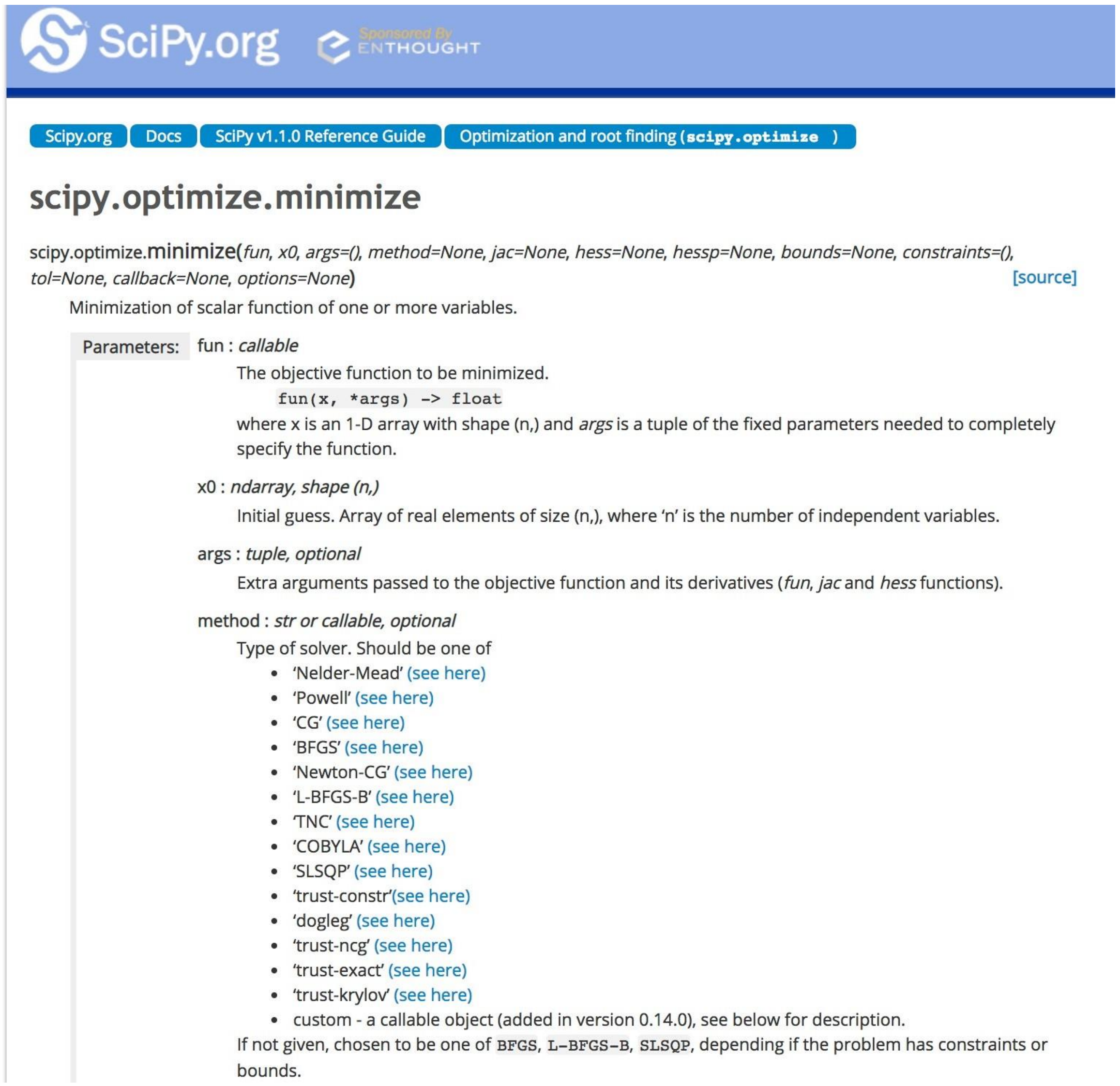
$$I[f] = \int_{X \times Y} V(f(\vec{x}), y) p(\vec{x}, y)\, d\vec{x}\, dy$$

where $V(f(\vec{x}), y)$ is the loss function, and $p(\vec{x}, y)$ is the probability density function of the process that generated the data, which can equivalently be written as

$$p(\vec{x}, y) = p(y \mid \vec{x}) p(\vec{x}).$$

In practice, the probability distribution $p(\vec{x}, y)$ is unknown. Consequently, utilizing a training


Plot of various functions. Blue is the 0–1 indicator function. Green is the square loss function. Purple is the hinge loss function. Yellow is the logistic loss function. Note that all surrogates give a loss penalty of 1 for $y = f(x = 0)$


SciPy.org
Sponsored By ENTHOUGHT
Scipy.org
Docs
SciPy v1.1.0 Reference Guide
Optimization and root finding (scipy.optimize)

scipy.optimize.minimize

scipy.optimize.**minimize**(*fun, x0, args=(), method=None, jac=None, hess=None, hessp=None, bounds=None, constraints=(), tol=None, callback=None, options=None*) [source]

Minimization of scalar function of one or more variables.

Parameters: fun : *callable*

The objective function to be minimized.

```
fun(x, *args) -> float
```

where x is an 1-D array with shape (n,) and *args* is a tuple of the fixed parameters needed to completely specify the function.

x0 : *ndarray, shape (n,)*

Initial guess. Array of real elements of size (n,), where 'n' is the number of independent variables.

args : *tuple, optional*

Extra arguments passed to the objective function and its derivatives (*fun*, *jac* and *hess* functions).

method : *str or callable, optional*

Type of solver. Should be one of

- 'Nelder-Mead' (see here)
- 'Powell' (see here)
- 'CG' (see here)
- 'BFGS' (see here)
- 'Newton-CG' (see here)
- 'L-BFGS-B' (see here)
- 'TNC' (see here)
- 'COBYLA' (see here)
- 'SLSQP' (see here)
- 'trust-constr'(see here)
- 'dogleg' (see here)
- 'trust-ncg' (see here)
- 'trust-exact' (see here)
- 'trust-krylov' (see here)
- custom - a callable object (added in version 0.14.0), see below for description.

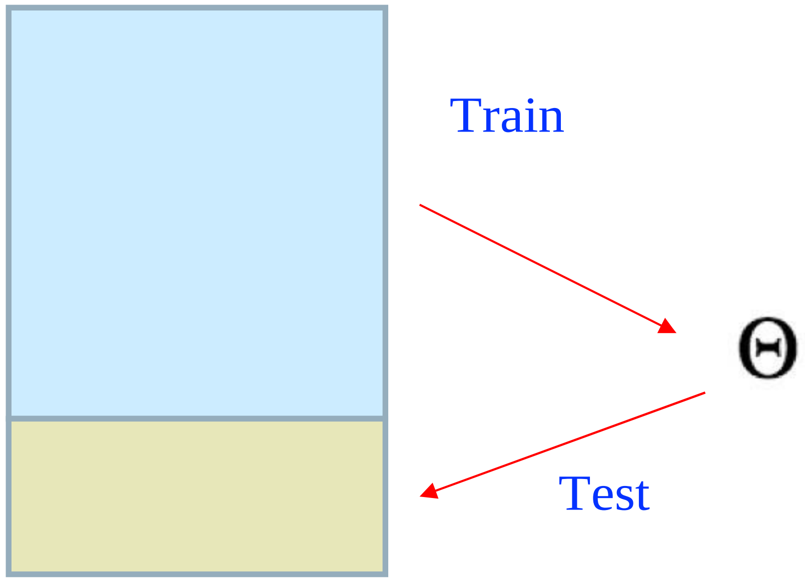
If not given, chosen to be one of `BFGS`, `L-BFGS-B`, `SLSQP`, depending if the problem has constraints or bounds.

Logistic Regression: Supervised Learning

Even if we have successfully minimized the cost function, surely the cost will not be 0, which means that our model will not be perfect. **How do we evaluate our model?**

We can think of this algorithm as trying to learn the categories (0 or 1) that the independent variables belong to, and use our data itself to test the results.

The basic idea is to take a random selection (say 80%) of our data set, find the best fitting parameters (called "training") and then test it on the remaining 20%. The percentage of the remaining data that is successfully classified is the accuracy of our model:


Train
Θ
Test