Machine Learning

Regression

Regression model: Overview

Regression model:

1. A form of statistical modeling that attempts to evaluate the relationship between one variable (termed the dependent variable) and one or more other variables (termed the independent variables). It is a form of global analysis as it only produces a single equation for the relationship.

2. Regression model estimates the nature of the relationship between the independent and dependent variables.
   - Change in dependent variables that results from changes in independent variables, ie. size of the relationship.
   - Strength of the relationship.
   - Statistical significance of the relationship.

Examples:

- Dependent variable is employment income – independent variables might be hours of work, education, occupation, sex, age, region, years of experience, unionization status, etc.

- Price of a product and quantity produced or sold:
  - Quantity sold affected by price. Dependent variable is quantity of product sold – independent variable is price.
  - Price affected by quantity offered for sale. Dependent variable is price – independent variable is quantity sold

Regression Model: Linear regression

Regression: Output a scalar

- Stock Market Forecast

$f($  $) =$ Dow Jones Industrial Average at tomorrow

- Self-driving Car

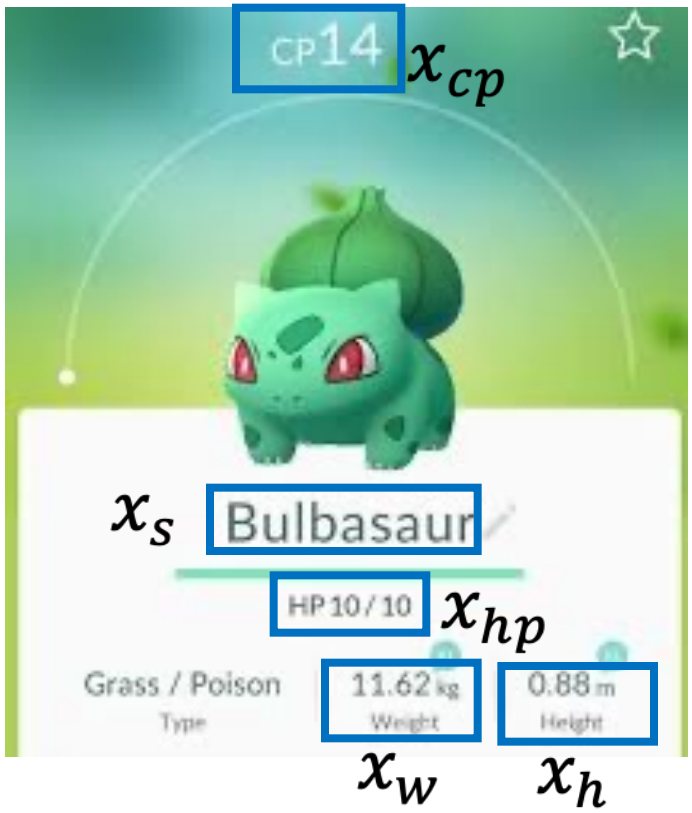
$f($  $) =$ Steering wheel angle

- Recommendation

$f($ User A Product B $) =$ Probability of Purchase

Example Application

- Estimating the Combat Power (CP) of a pokemon after evolution

$f($ 

$) =$ CP after evolution y

Step 1: Model

$y = b + w \cdot x_{cp}$

A set of function

Model

$f_1, f_2 \cdots$

w and b are parameters (can be any value)

f_1: $y = 10.0 + 9.0 \cdot x_{cp}$

f_2: $y = 9.8 + 9.2 \cdot x_{cp}$

f_3: $y = -0.8 - 1.2 \cdot x_{cp}$

…… infinite

$f($ 

$x) =$ CP after evolution y

Linear model: $y = b + \sum w_i x_i$

x_i: x_{cp}, x_{hp}, x_w, x_h …

feature

w_i: weight, b: bias

Step 2: Goodness of Function

$y = b + w \cdot x_{cp}$

A set of function

Model

$f_1, f_2 \cdots$

function input:

function Output (scalar):

Training Data

Step 2: Goodness of Function

Training Data:
10 pokemons

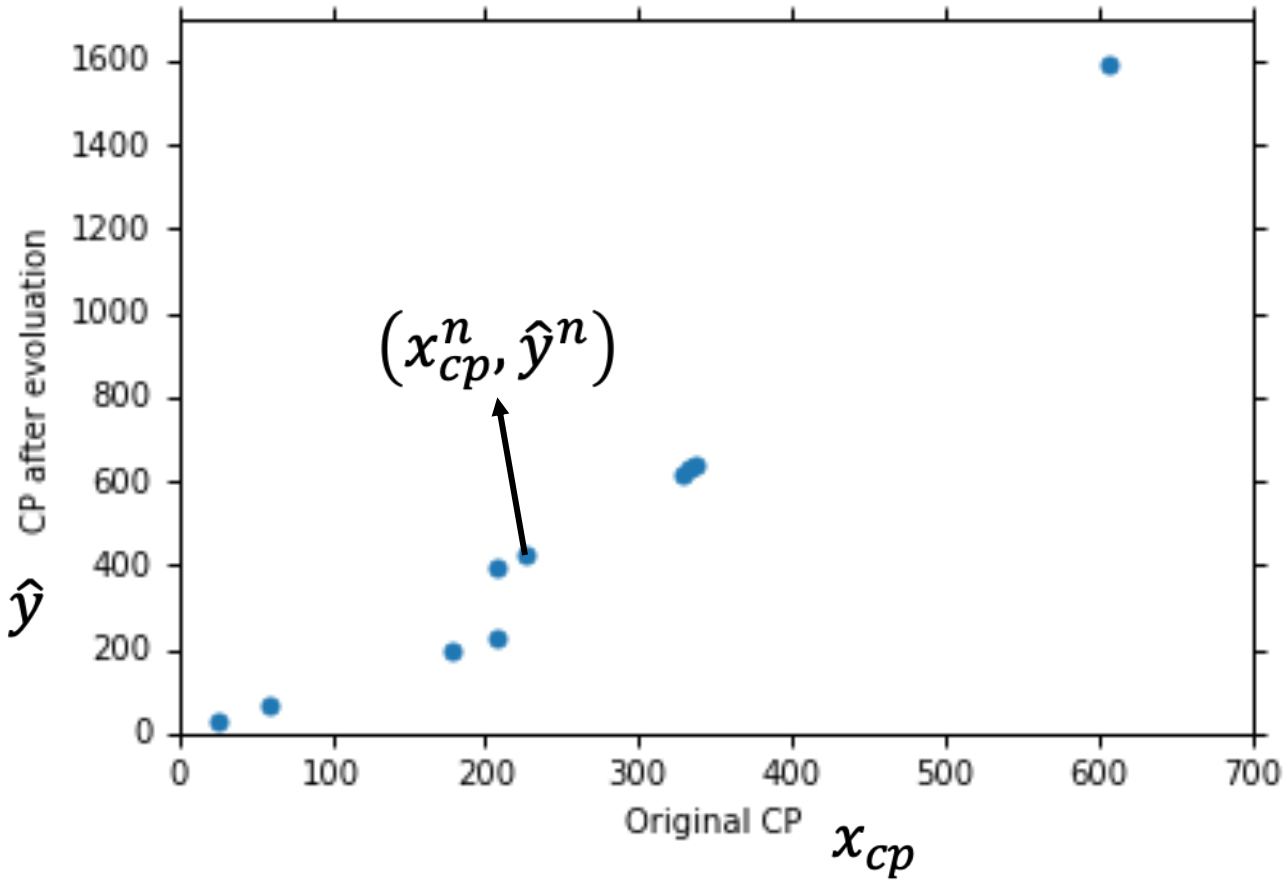
$(x^1, \hat{y}^1)$

$(x^2, \hat{y}^2)$

⋯ ⋯

$(x^{10}, \hat{y}^{10})$

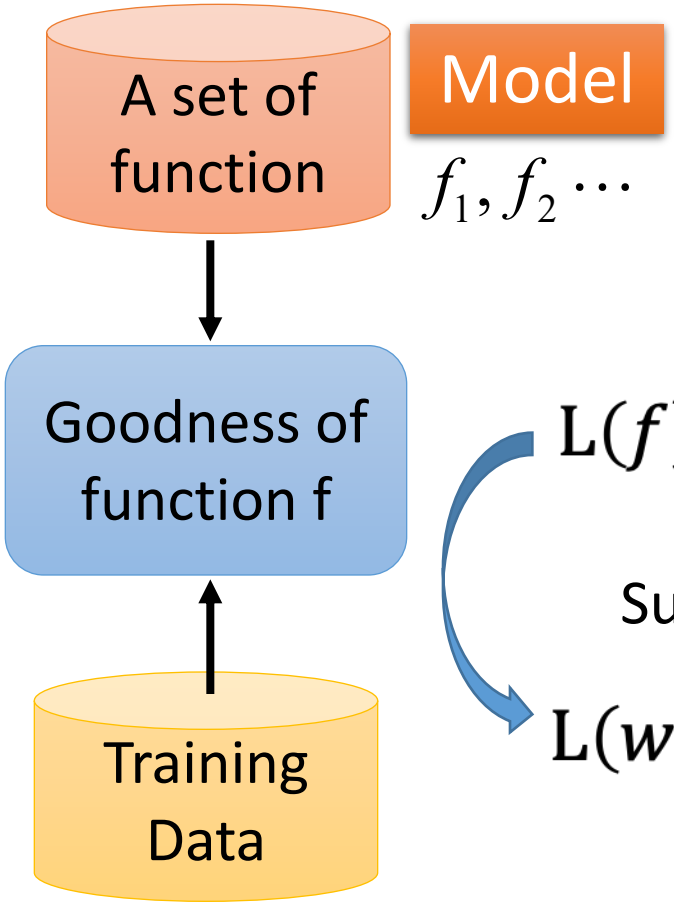
This is real data.



Source: https://www.openintro.org/stat/data/?data=pokemon

Step 2: Goodness of Function

$y = b + w \cdot x_{cp}$



Loss function L:

Input: a function, output: how bad it is

Estimation error

$$L(f) = \sum_{n=1}^{10} \left(\hat{y}^n - f\left(x_{cp}^n\right)\right)^2$$

Sum over examples

Estimated y based on input function

$$L(w, b) = \sum_{n=1}^{10} \left(\hat{y}^n - \left(b + w \cdot x_{cp}^n\right)\right)^2$$

Step 2: Goodness of Function

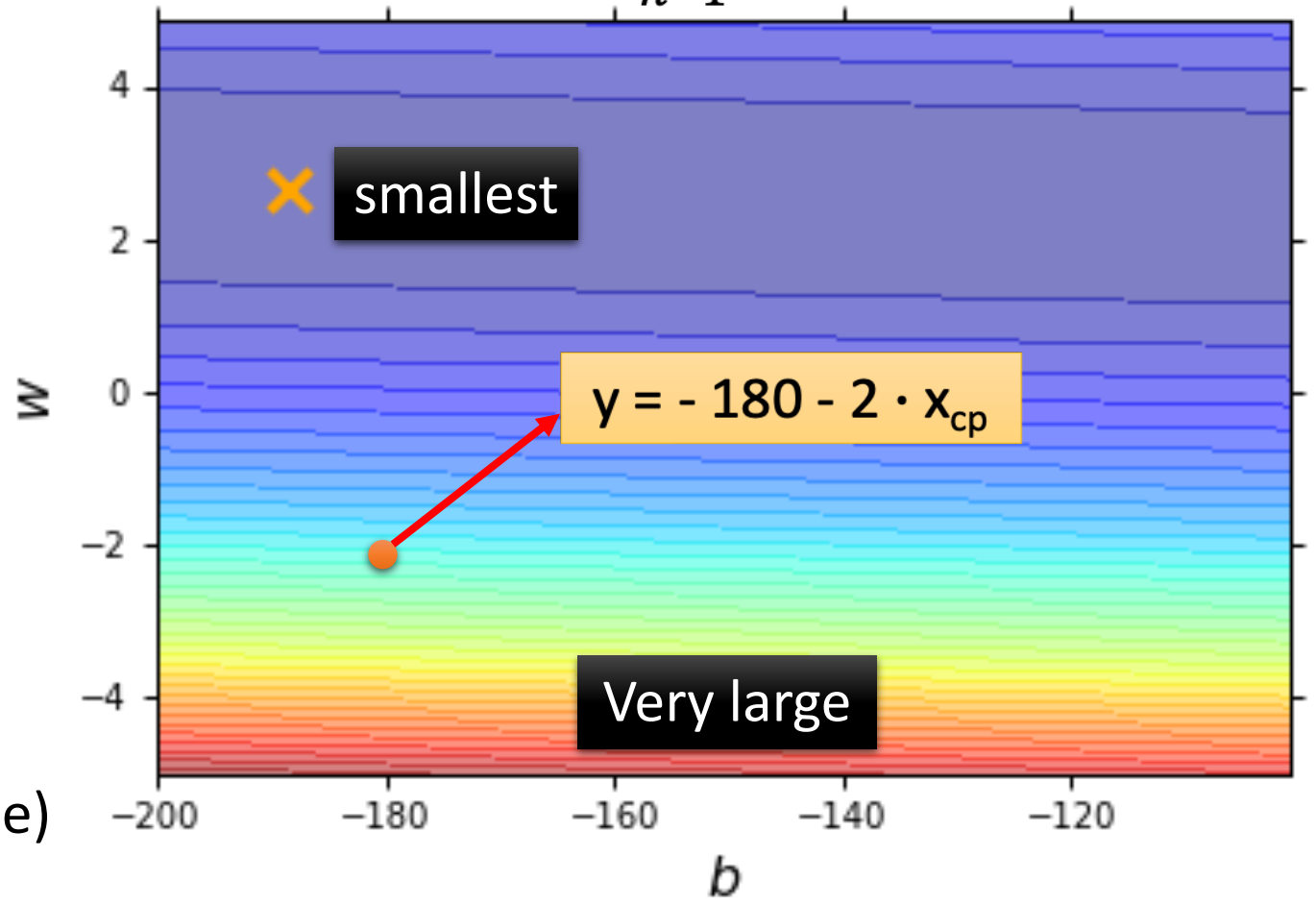
- Loss Function

$$L(w,b) = \sum_{n=1}^{10} \left(\hat{y}^n - \left(b + w \cdot x_{cp}^n\right)\right)^2$$

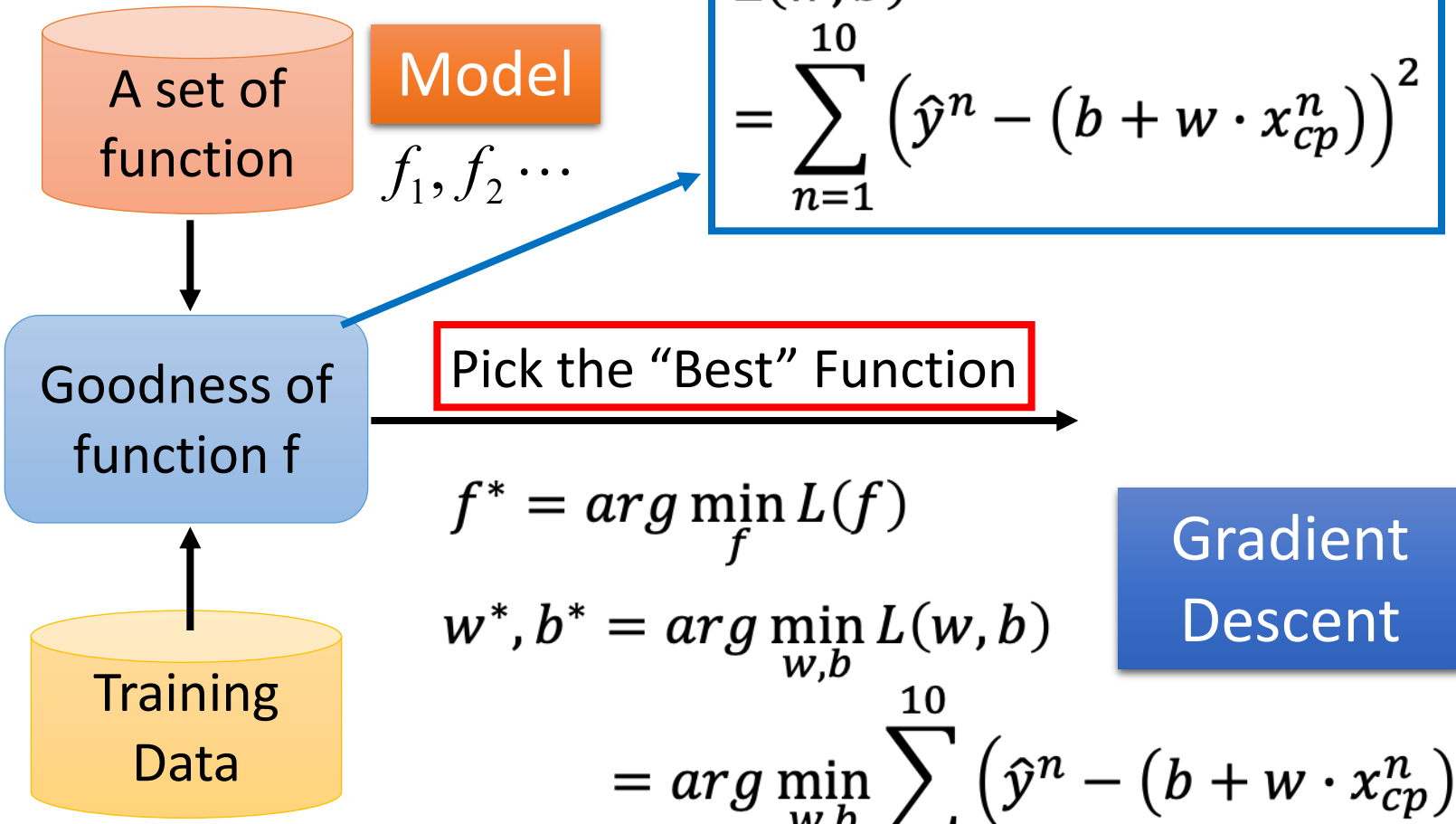
Each point in the figure is a function

The color represents $L(w,b)$.

(true example)



Step 3: Best Function

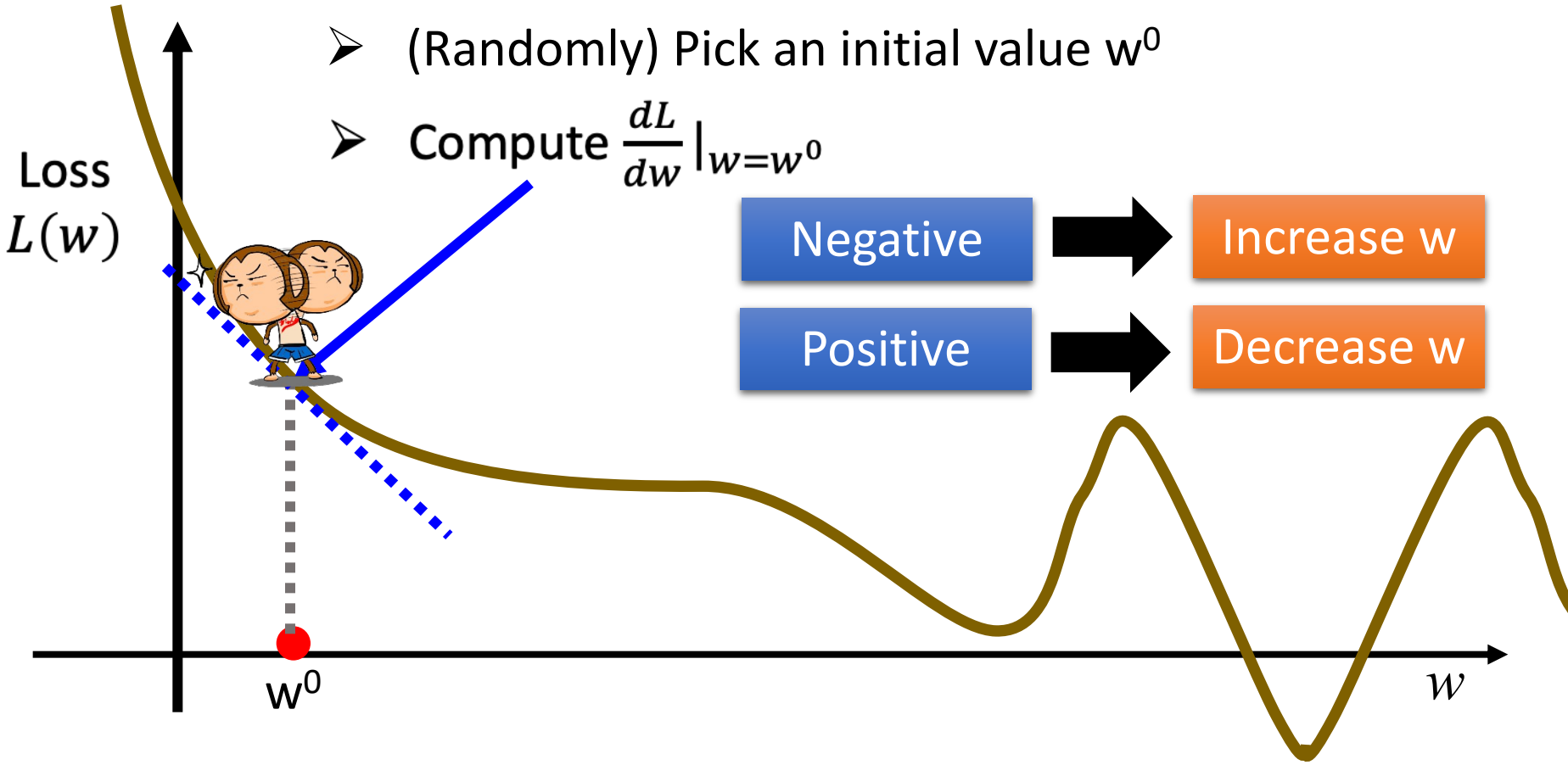


http://chico386.pixnet.net/album/photo/171572850

Step 3: Gradient Descent

$$w^* = arg \min_w L(w)$$

- Consider loss function $L(w)$ with one parameter w:

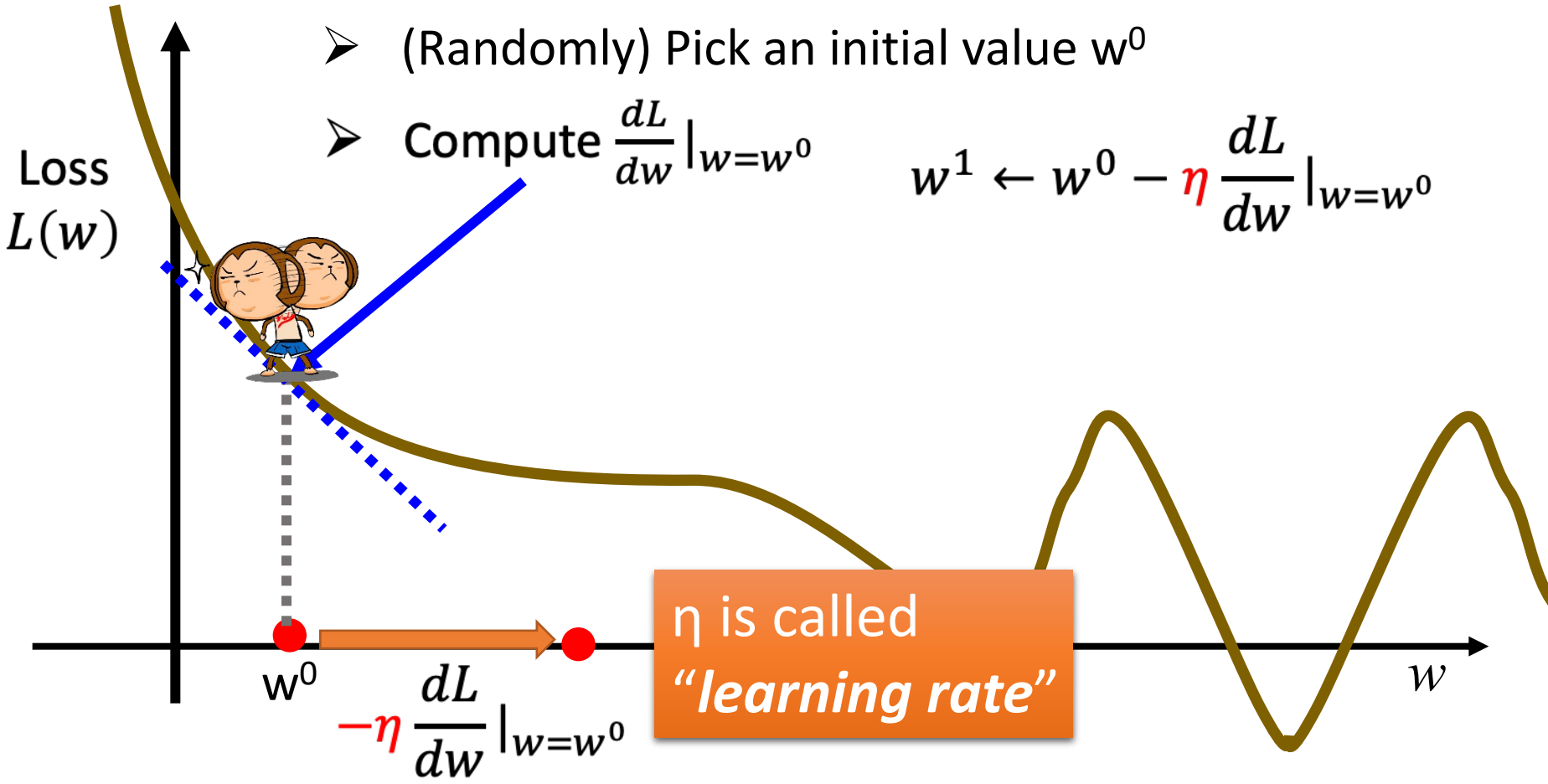


http://chico386.pixnet.net/album/photo/171572850

Step 3: Gradient Descent

$$w^* = arg \min_w L(w)$$

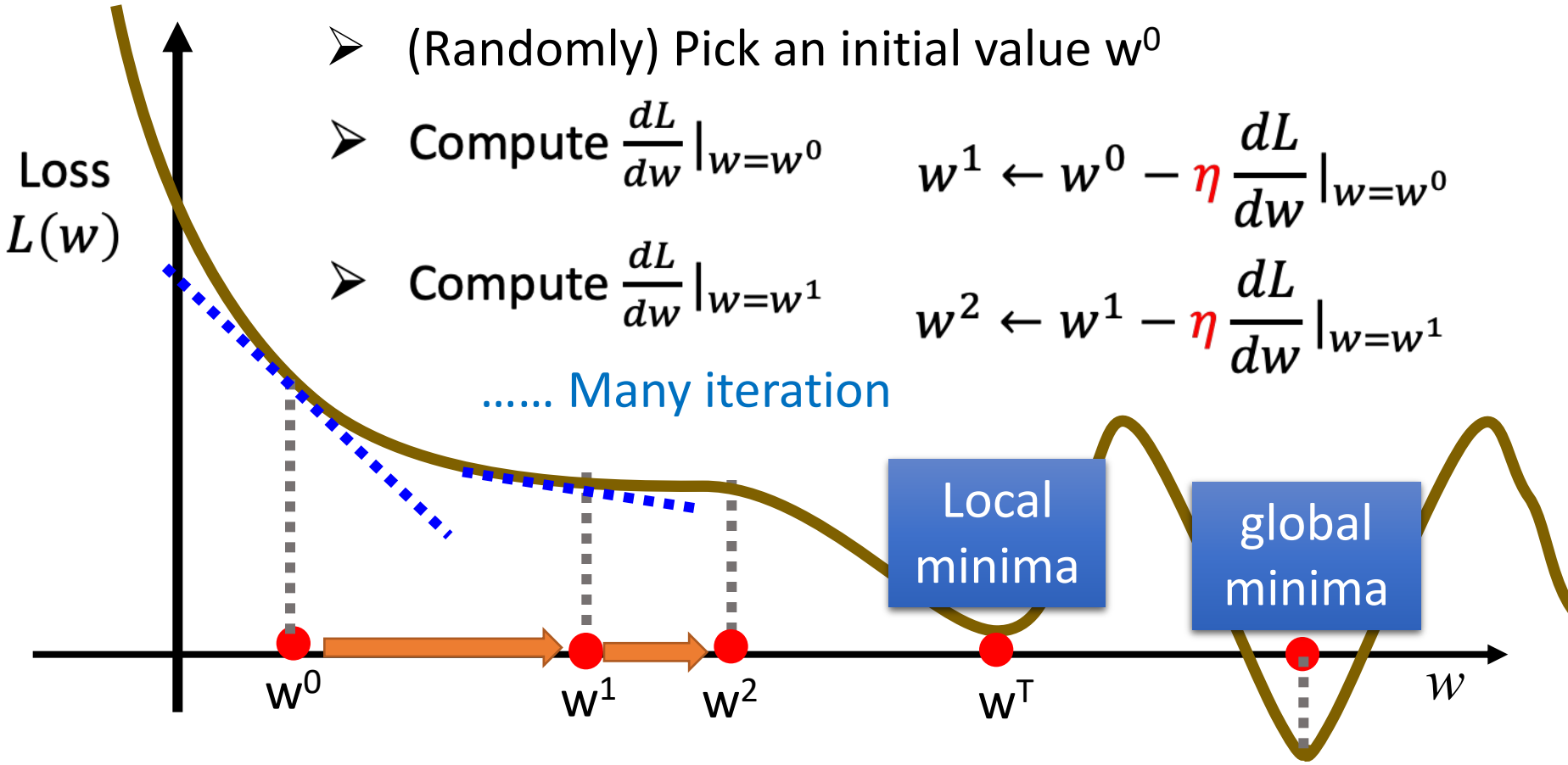
- Consider loss function $L(w)$ with one parameter w:



Step 3: Gradient Descent

$$w^* = arg\min_w L(w)$$

- Consider loss function $L(w)$ with one parameter w:



Step 3: Gradient Descent

$$\begin{bmatrix} \frac{\partial L}{\partial w} \\ \frac{\partial L}{\partial b} \end{bmatrix}$$ gradient

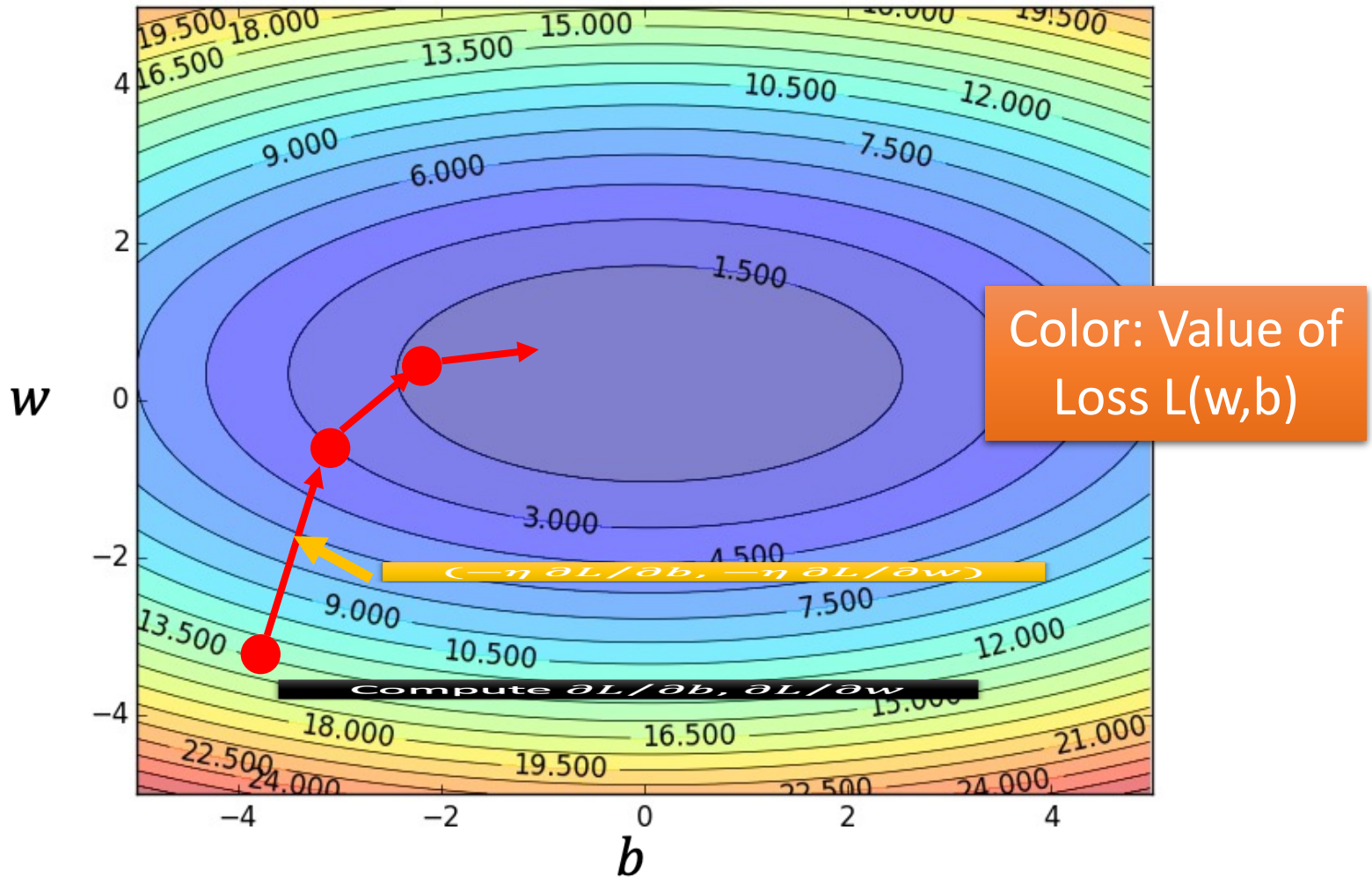
- How about two parameters? $w^*, b^* = arg \min_{w,b} L(w,b)$
  - (Randomly) Pick an initial value w^0, b^0
  - Compute $\frac{\partial L}{\partial w}|_{w=w^0,b=b^0}, \frac{\partial L}{\partial b}|_{w=w^0,b=b^0}$

$$w^1 \leftarrow w^0 - \eta \frac{\partial L}{\partial w}|_{w=w^0,b=b^0} \qquad b^1 \leftarrow b^0 - \eta \frac{\partial L}{\partial b}|_{w=w^0,b=b^0}$$

  - Compute $\frac{\partial L}{\partial w}|_{w=w^1,b=b^1}, \frac{\partial L}{\partial b}|_{w=w^1,b=b^1}$

$$w^2 \leftarrow w^1 - \eta \frac{\partial L}{\partial w}|_{w=w^1,b=b^1} \qquad b^2 \leftarrow b^1 - \eta \frac{\partial L}{\partial b}|_{w=w^1,b=b^1}$$

Step 3: Gradient Descent



Step 3: Gradient Descent

- When solving:

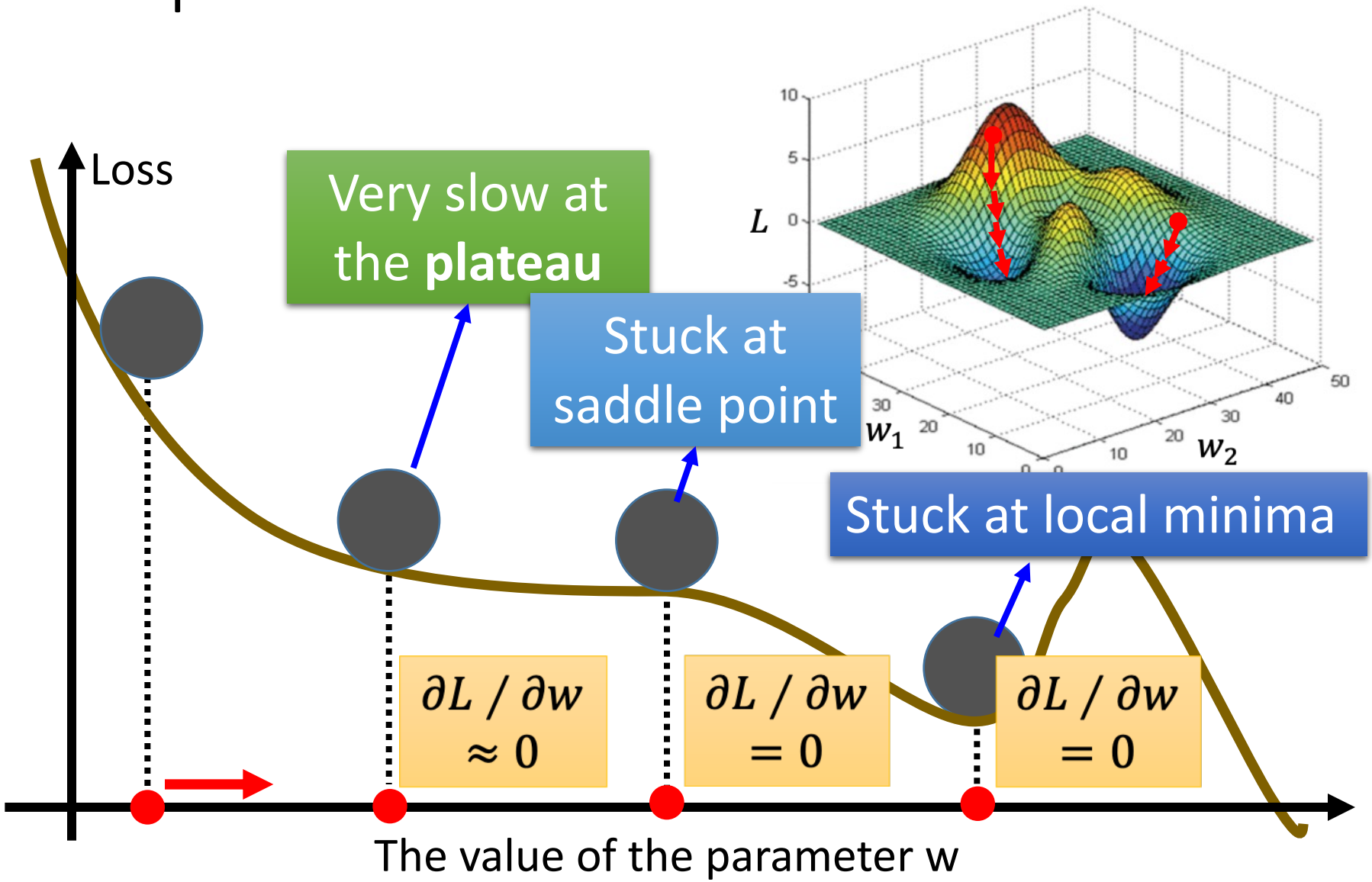
$$\theta^* = \arg\max_{\theta} L(\theta)$$ by gradient descent

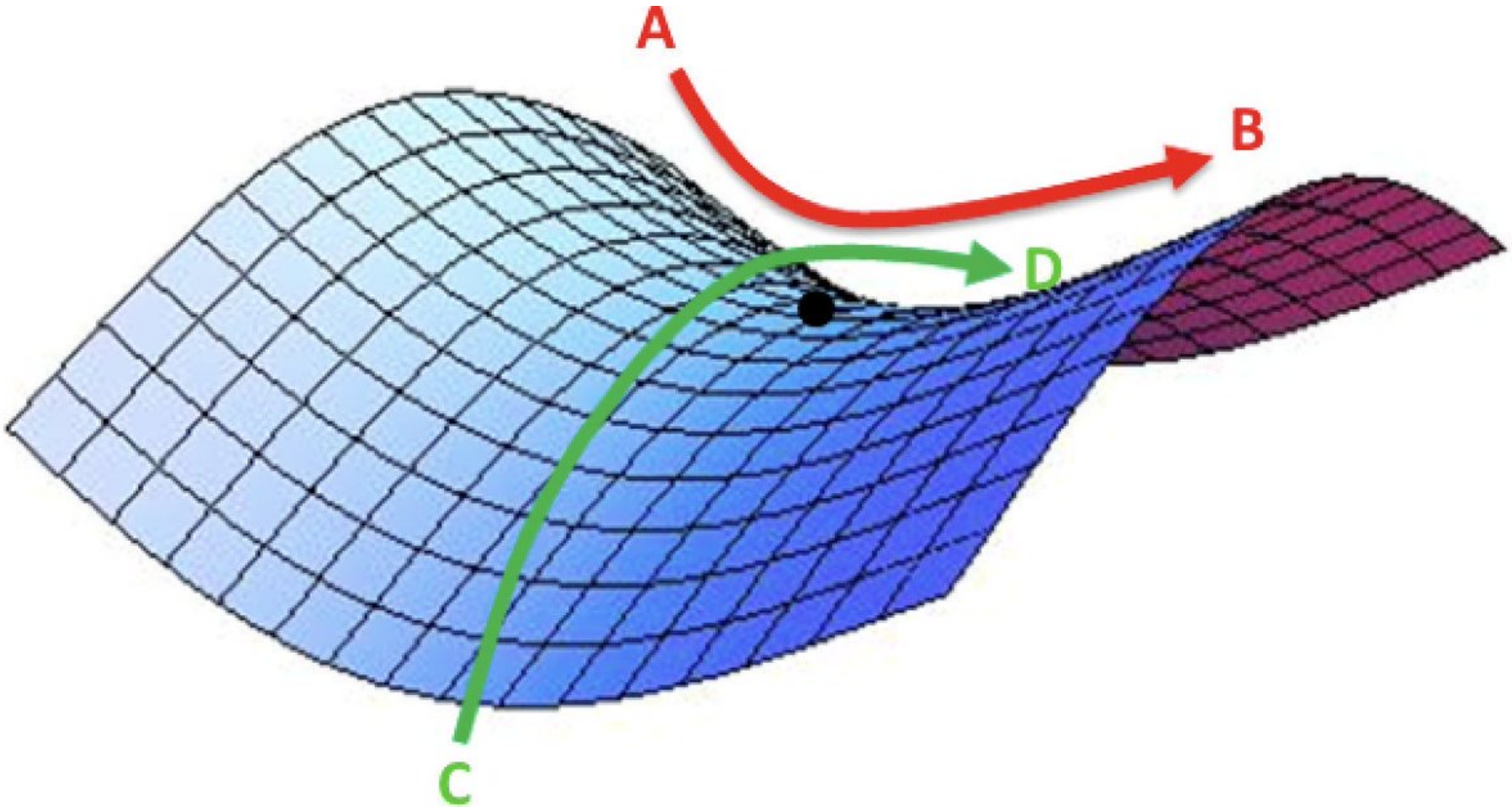
- Each time we update the parameters, we obtain θ that makes $L(\theta)$ smaller.

$$L(\theta^0) > L(\theta^1) > L(\theta^2) > \cdots$$

Is this statement correct?

Step 3: Gradient Descent




A
B
D
C

Step 3: Gradient Descent

- Formulation of $\partial L/\partial w$ and $\partial L/\partial b$

$$L(w,b) = \sum_{n=1}^{10} \left(\hat{y}^n - \left(b + w \cdot x_{cp}^n\right)\right)^2$$

$$\frac{\partial L}{\partial w} =? \sum_{n=1}^{10} 2\left(\hat{y}^n - \left(b + w \cdot x_{cp}^n\right)\right)$$

$$\frac{\partial L}{\partial b} =?$$

Step 3: Gradient Descent

- Formulation of $\partial L/\partial w$ and $\partial L/\partial b$

$$L(w,b) = \sum_{n=1}^{10} \left(\hat{y}^n - \left(b + w \cdot x_{cp}^n\right)\right)^2$$

$$\frac{\partial L}{\partial w} =? \sum_{n=1}^{10} 2\left(\hat{y}^n - \left(b + w \cdot x_{cp}^n\right)\right)\left(-x_{cp}^n\right)$$

$$\frac{\partial L}{\partial b} =? \sum_{n=1}^{10} 2\left(\hat{y}^n - \left(b + w \cdot x_{cp}^n\right)\right)$$

How's the results?

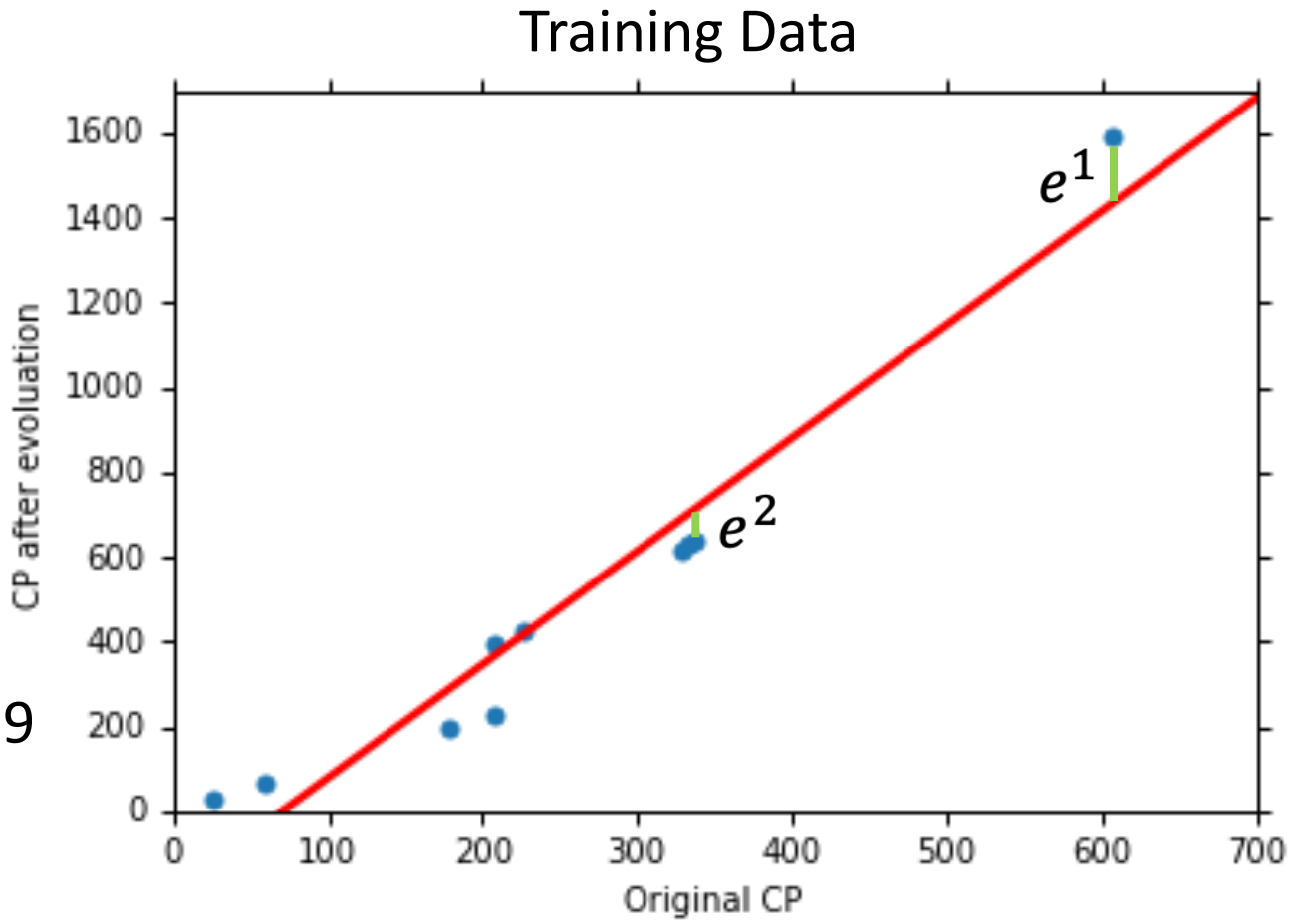
$y = b + w \cdot x_{cp}$

b = -188.4

w = 2.7

Average Error on Training Data

$$= \frac{1}{10}\sum_{n=1}^{10} e^n = 31.9$$



How's the results? - Generalization

What we really care about is the error on new data (testing data)

$y = b + w \cdot x_{cp}$

b = -188.4

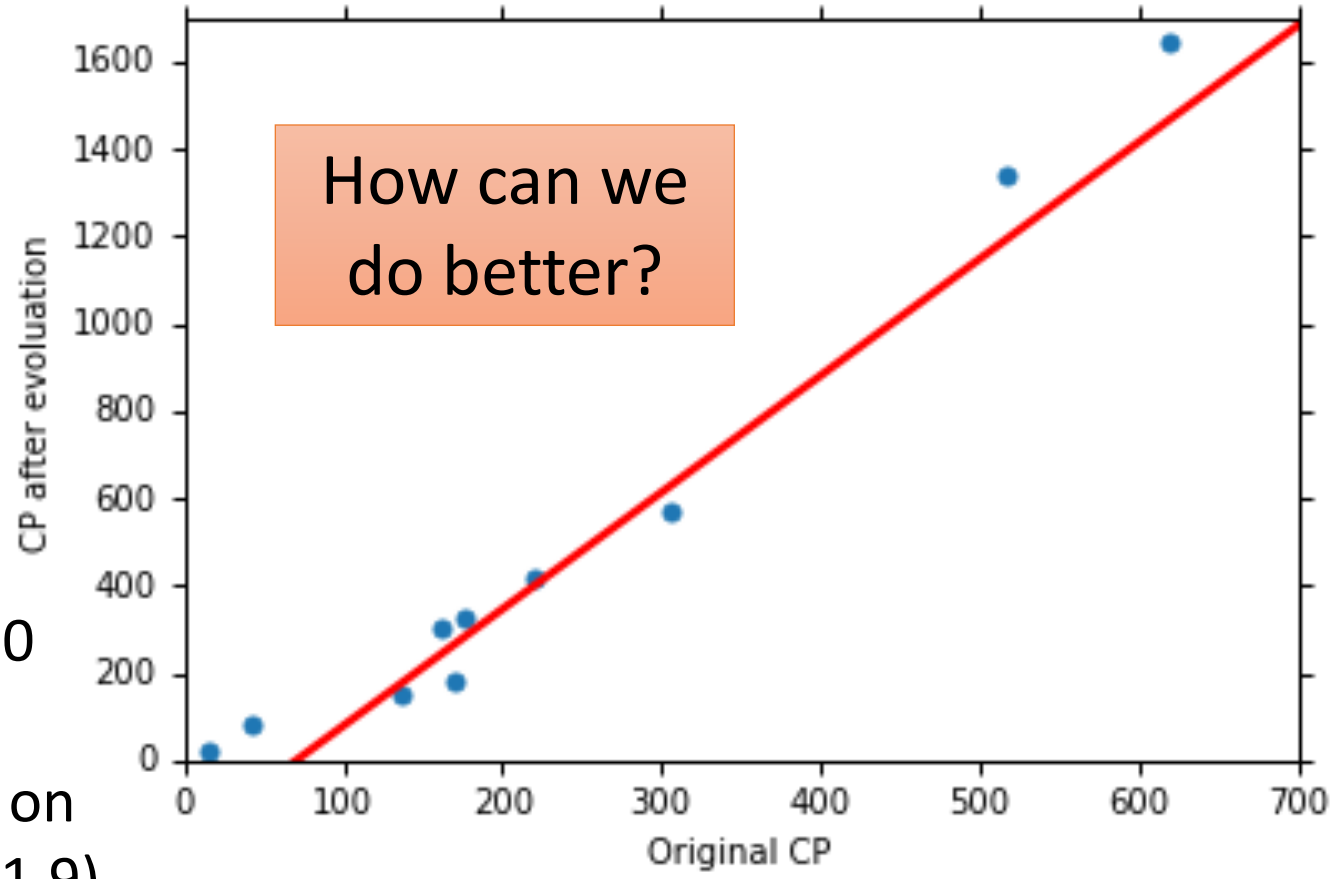
w = 2.7

Average Error on Testing Data

$$= \frac{1}{10}\sum_{n=1}^{10} e^n = 35.0$$

> Average Error on Training Data (31.9)

Another 10 pokemons as testing data



Selecting another Model

$$y = b + w_1 \cdot x_{cp} + w_2 \cdot (x_{cp})^2$$

Best Function

b = -10.3

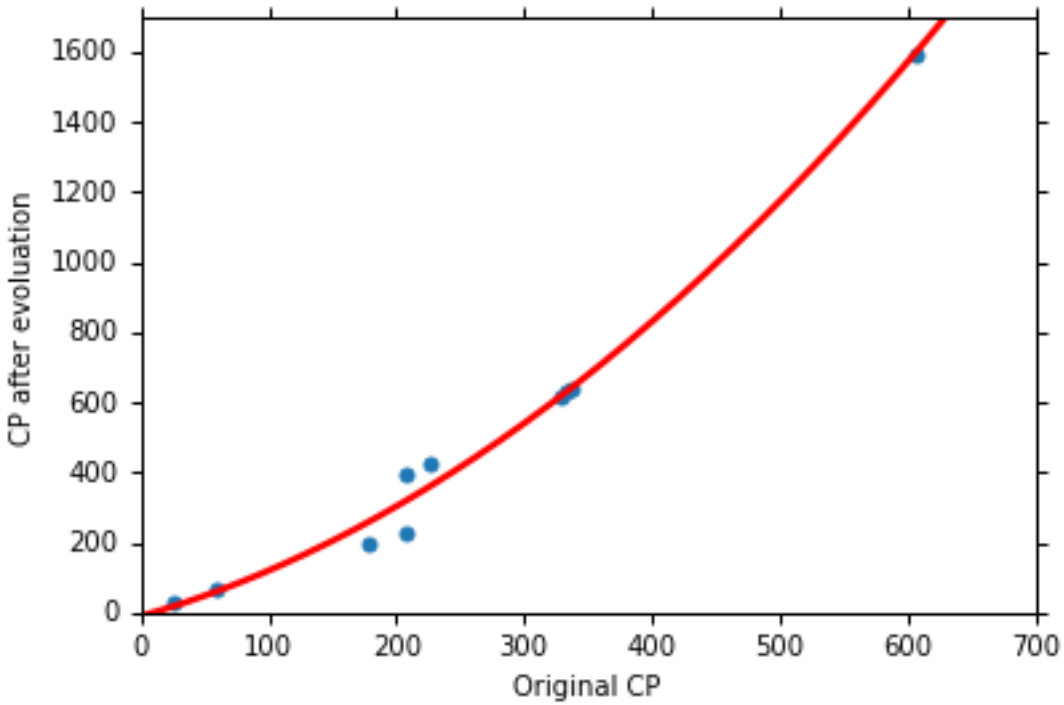
$w_1 = 1.0$, $w_2 = 2.7 \times 10^{-3}$

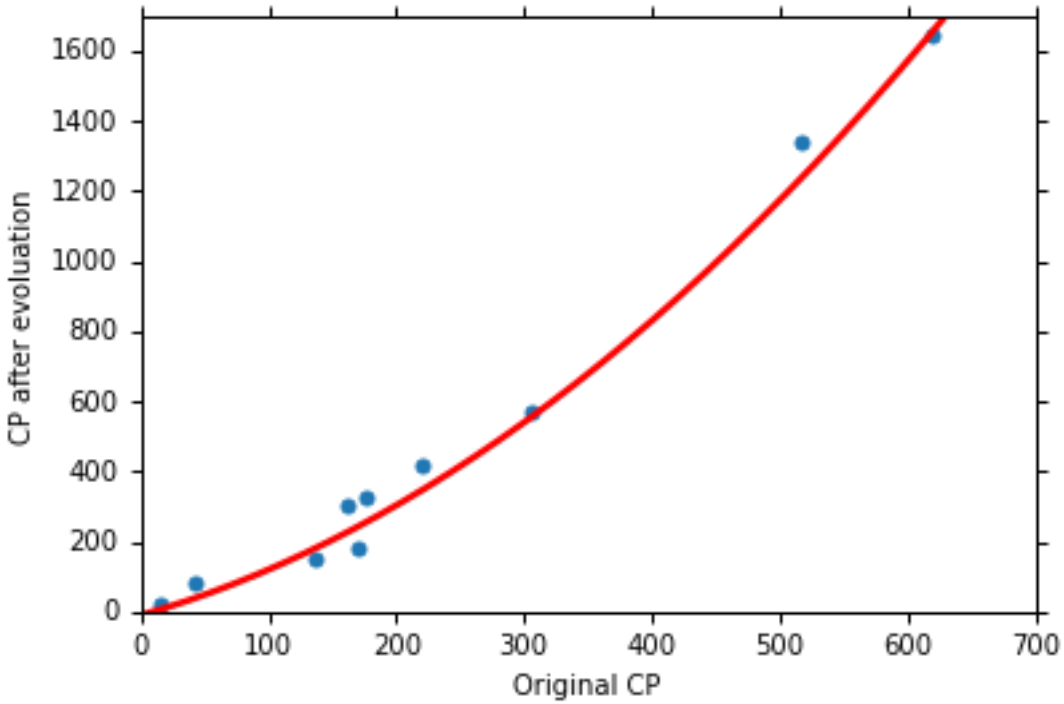
Average Error = 15.4

Testing:

Average Error = 18.4

Better! Could it be even better?





Selecting another Model

$$y = b + w_1 \cdot x_{cp} + w_2 \cdot (x_{cp})^2 + w_3 \cdot (x_{cp})^3$$

Best Function

$b = 6.4$, $w_1 = 0.66$

$w_2 = 4.3 \times 10^{-3}$

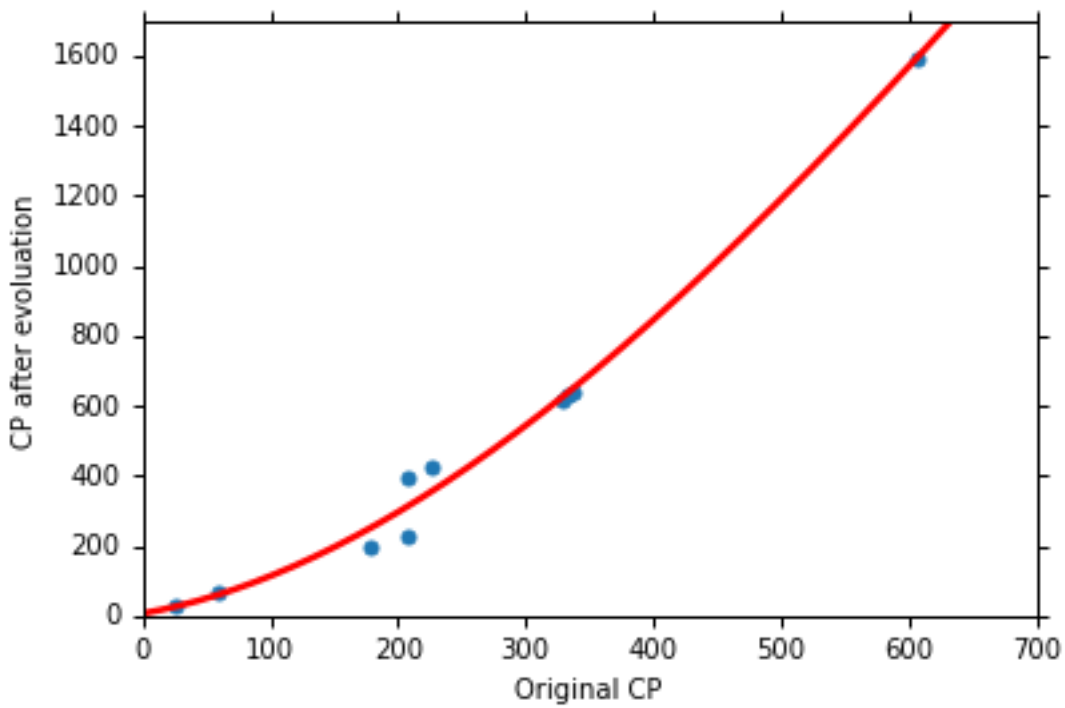
$w_3 = -1.8 \times 10^{-6}$

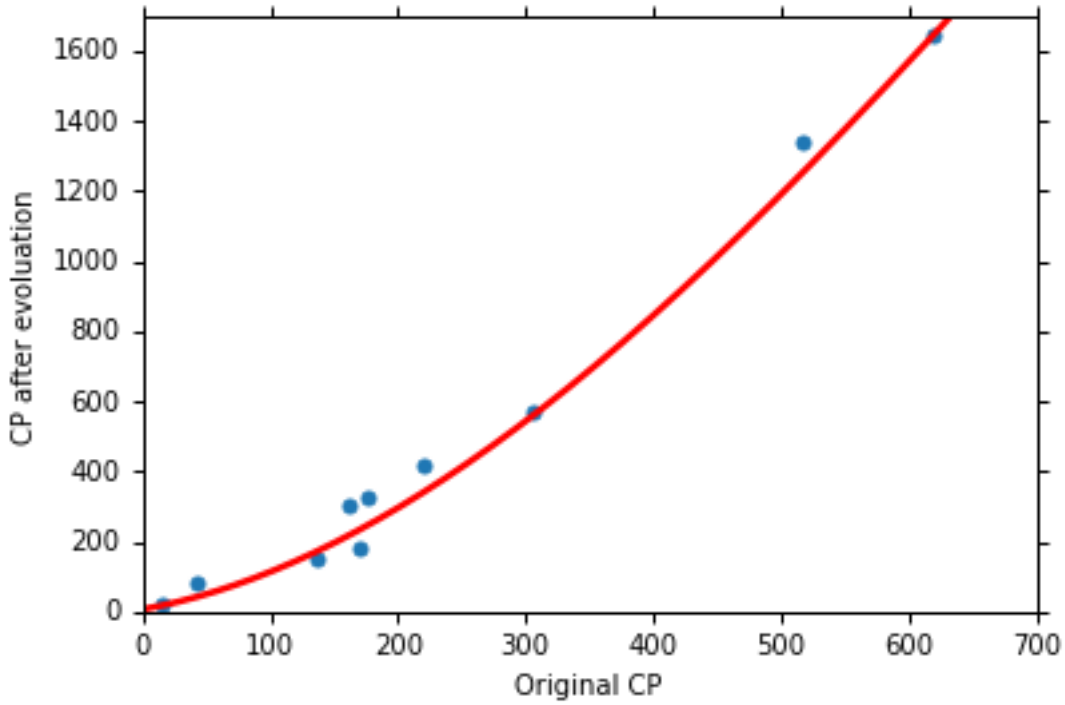
Average Error = 15.3

Testing:

Average Error = 18.1

Slightly better.
How about more complex model?





Selecting another Model

$$y = b + w_1 \cdot x_{cp} + w_2 \cdot (x_{cp})^2 + w_3 \cdot (x_{cp})^3 + w_4 \cdot (x_{cp})^4$$

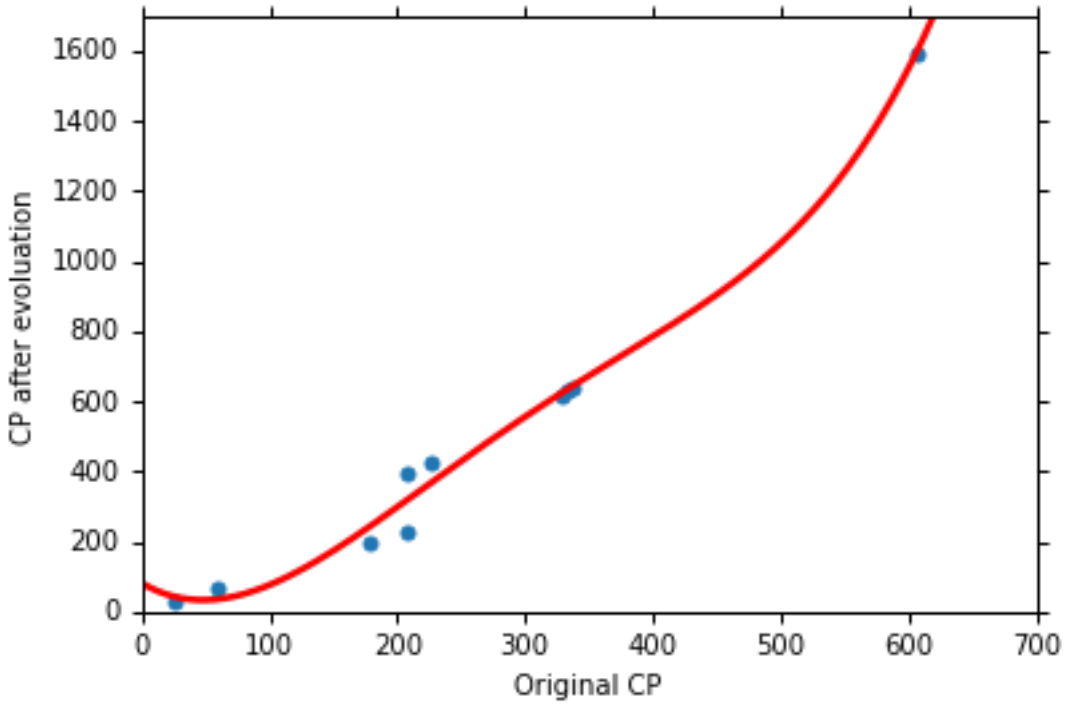
Best Function

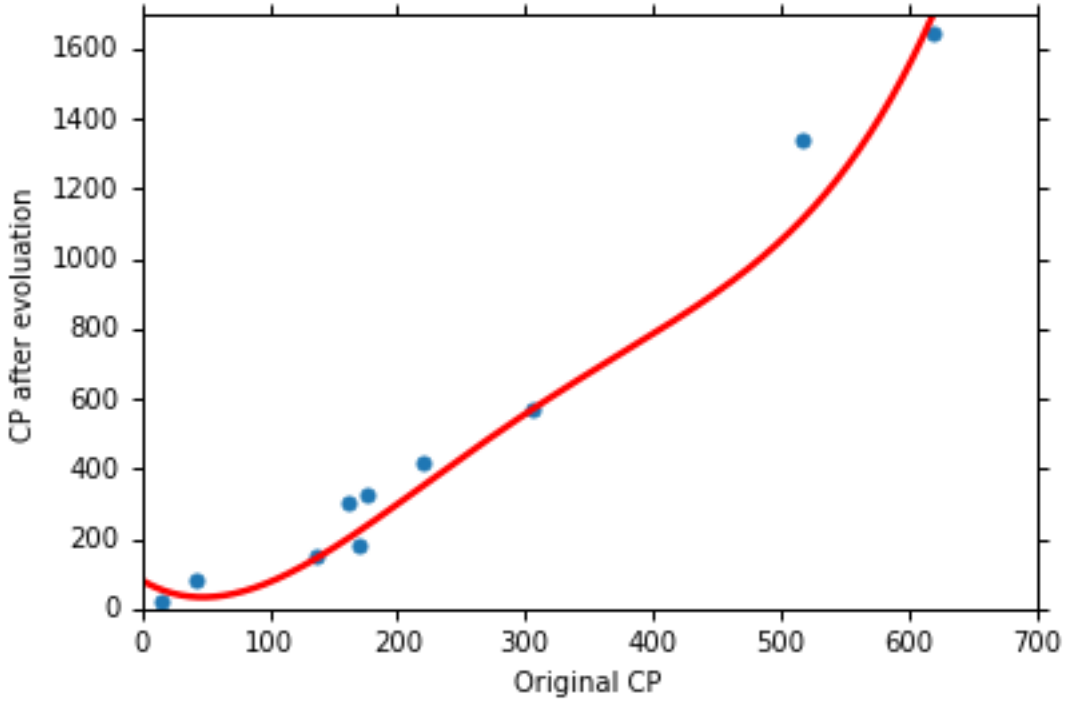
Average Error = 14.9

Testing:

Average Error = 28.8

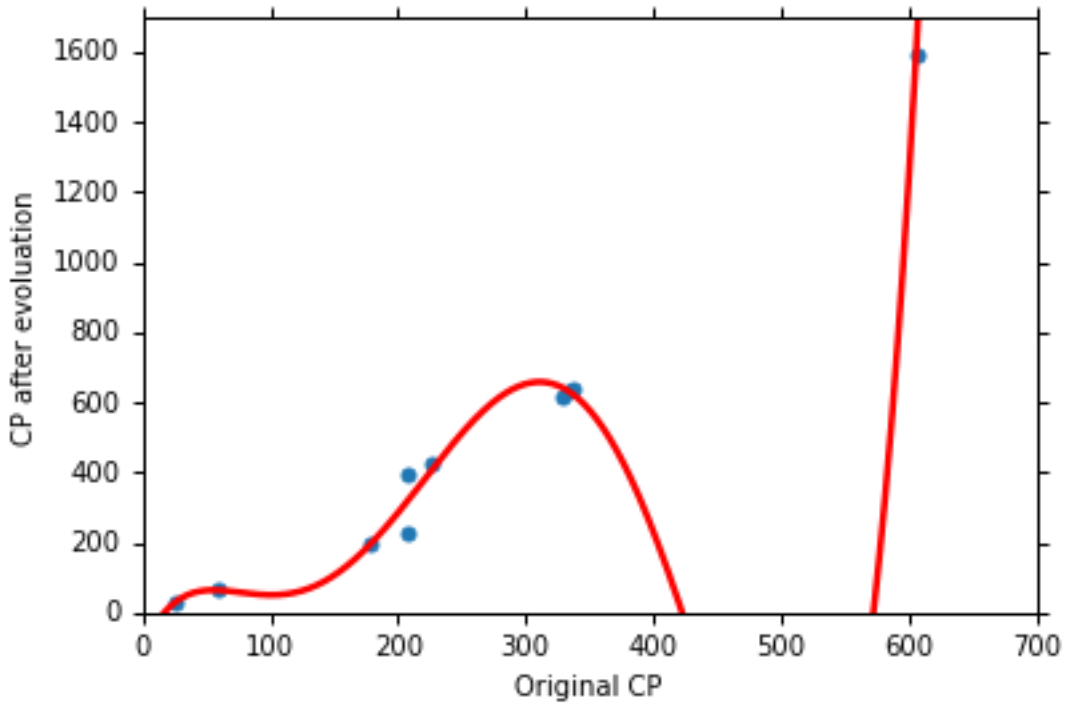
The results become worse ...





Selecting another Model

$$y = b + w_1 \cdot x_{cp} + w_2 \cdot (x_{cp})^2 + w_3 \cdot (x_{cp})^3 + w_4 \cdot (x_{cp})^4 + w_5 \cdot (x_{cp})^5$$



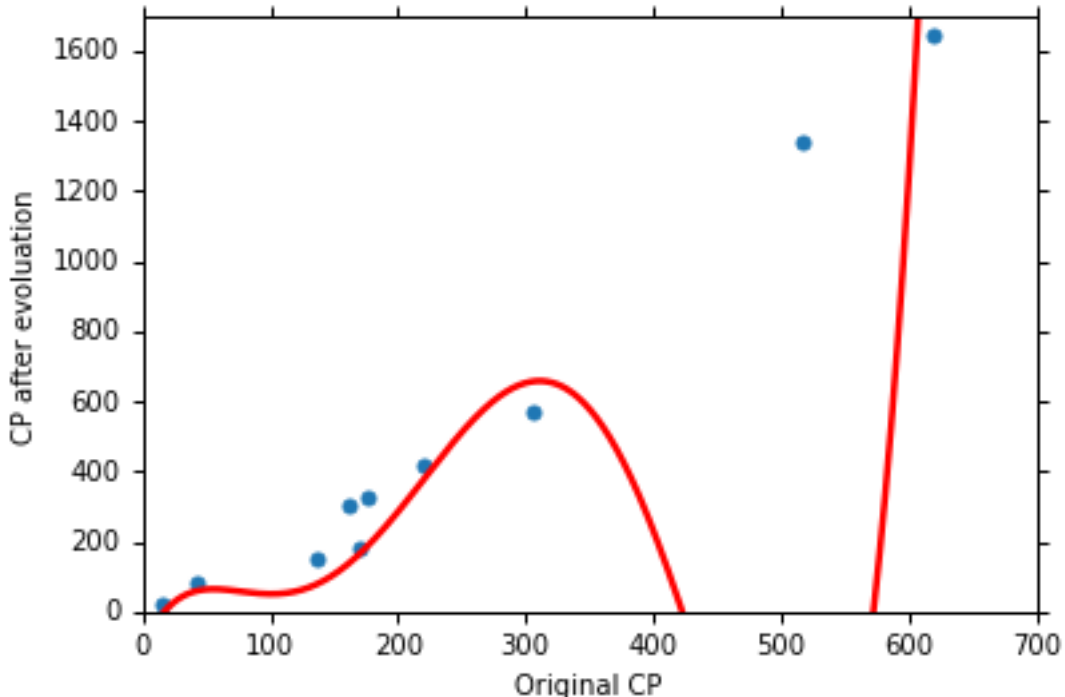
Best Function

Average Error = 12.8

Testing:

Average Error = 232.1

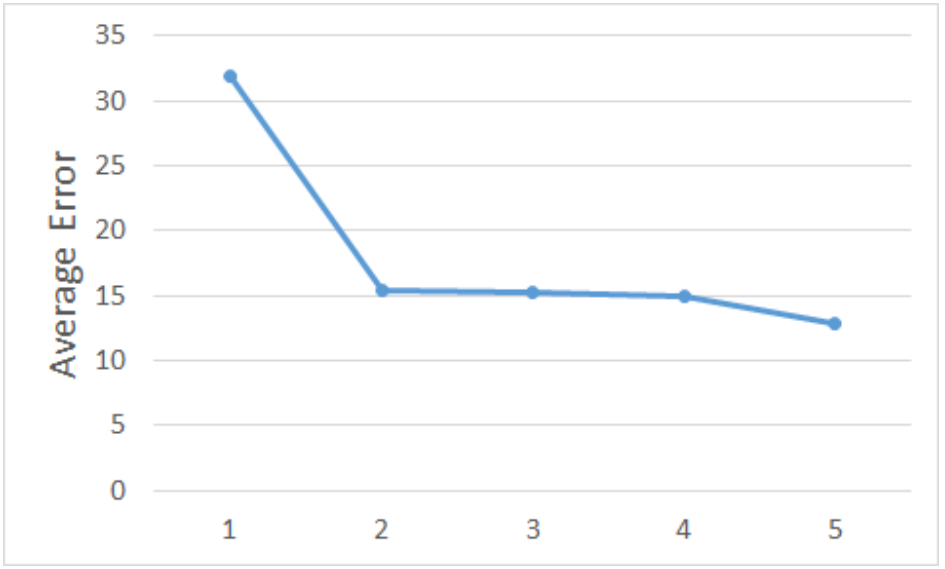
The results are so bad.

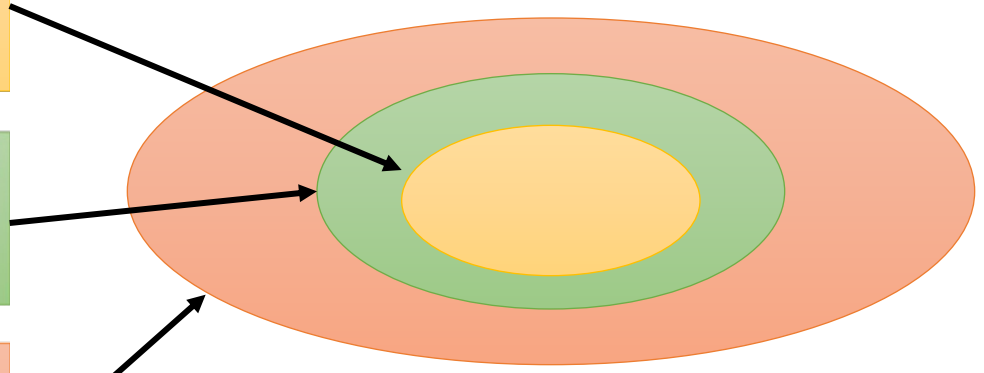


Model Selection

1. $y = b + w \cdot x_{cp}$
2. $y = b + w_1 \cdot x_{cp} + w_2 \cdot (x_{cp})^2$
3. $y = b + w_1 \cdot x_{cp} + w_2 \cdot (x_{cp})^2 + w_3 \cdot (x_{cp})^3$
4. $y = b + w_1 \cdot x_{cp} + w_2 \cdot (x_{cp})^2 + w_3 \cdot (x_{cp})^3 + w_4 \cdot (x_{cp})^4$
5. $y = b + w_1 \cdot x_{cp} + w_2 \cdot (x_{cp})^2 + w_3 \cdot (x_{cp})^3 + w_4 \cdot (x_{cp})^4 + w_5 \cdot (x_{cp})^5$

Training Data

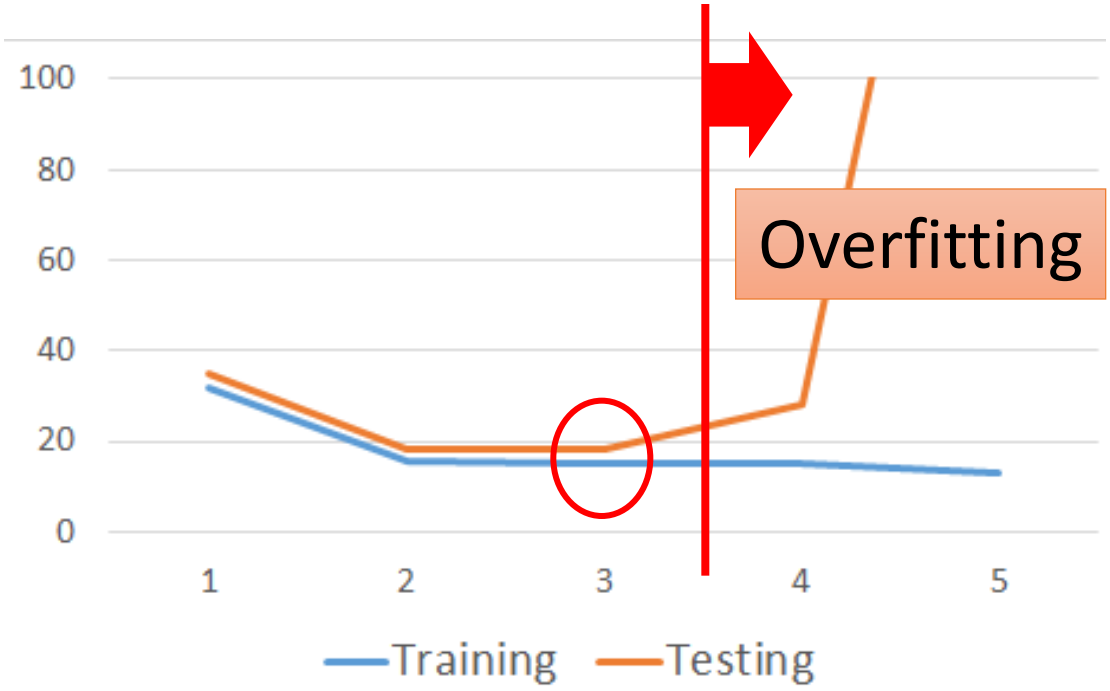




A more complex model yields lower error on training data.

If we can truly find the best function

Model Selection

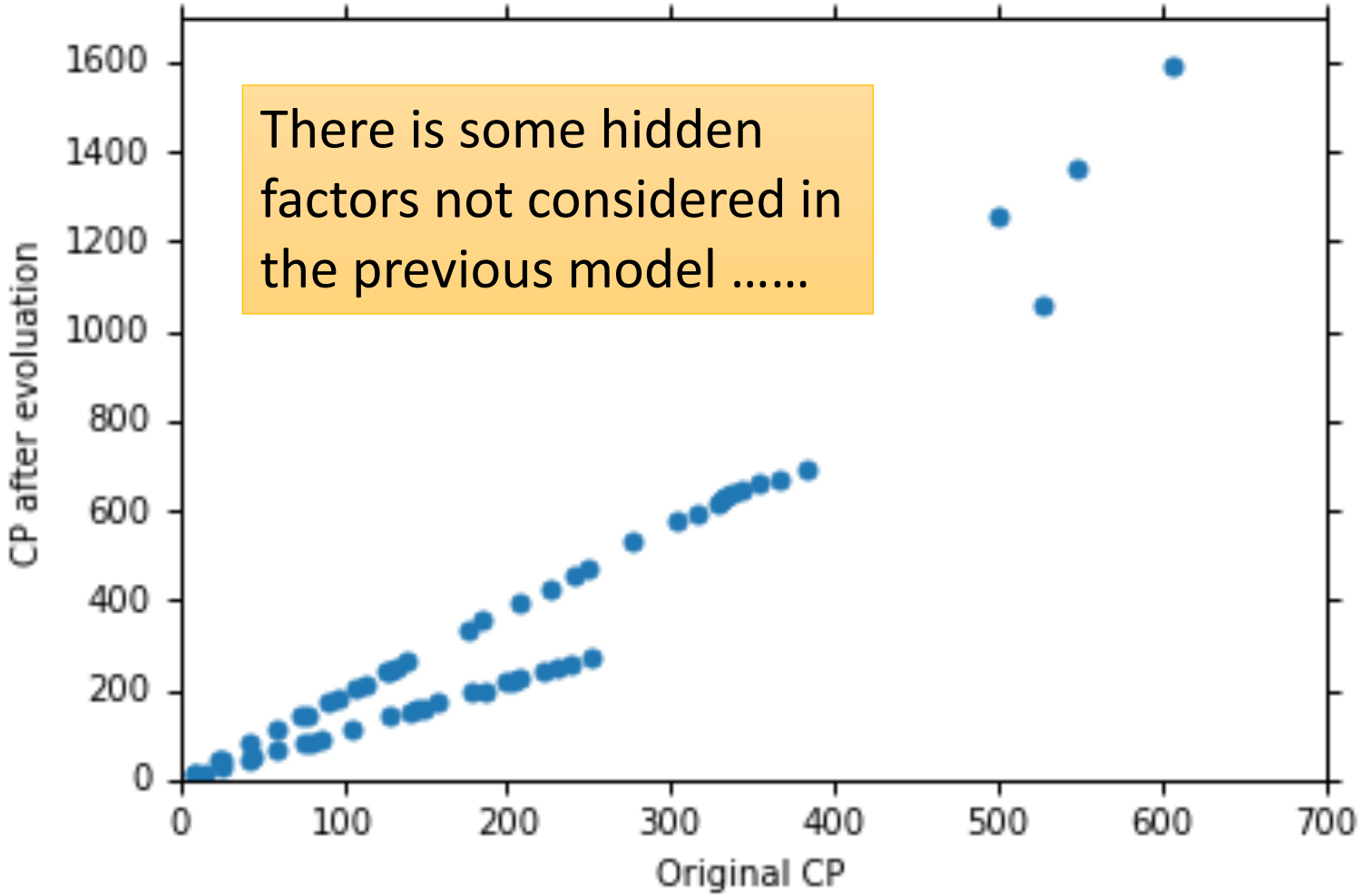


	Training	Testing
1	31.9	35.0
2	15.4	18.4
3	15.3	18.1
4	14.9	28.2
5	12.8	232.1

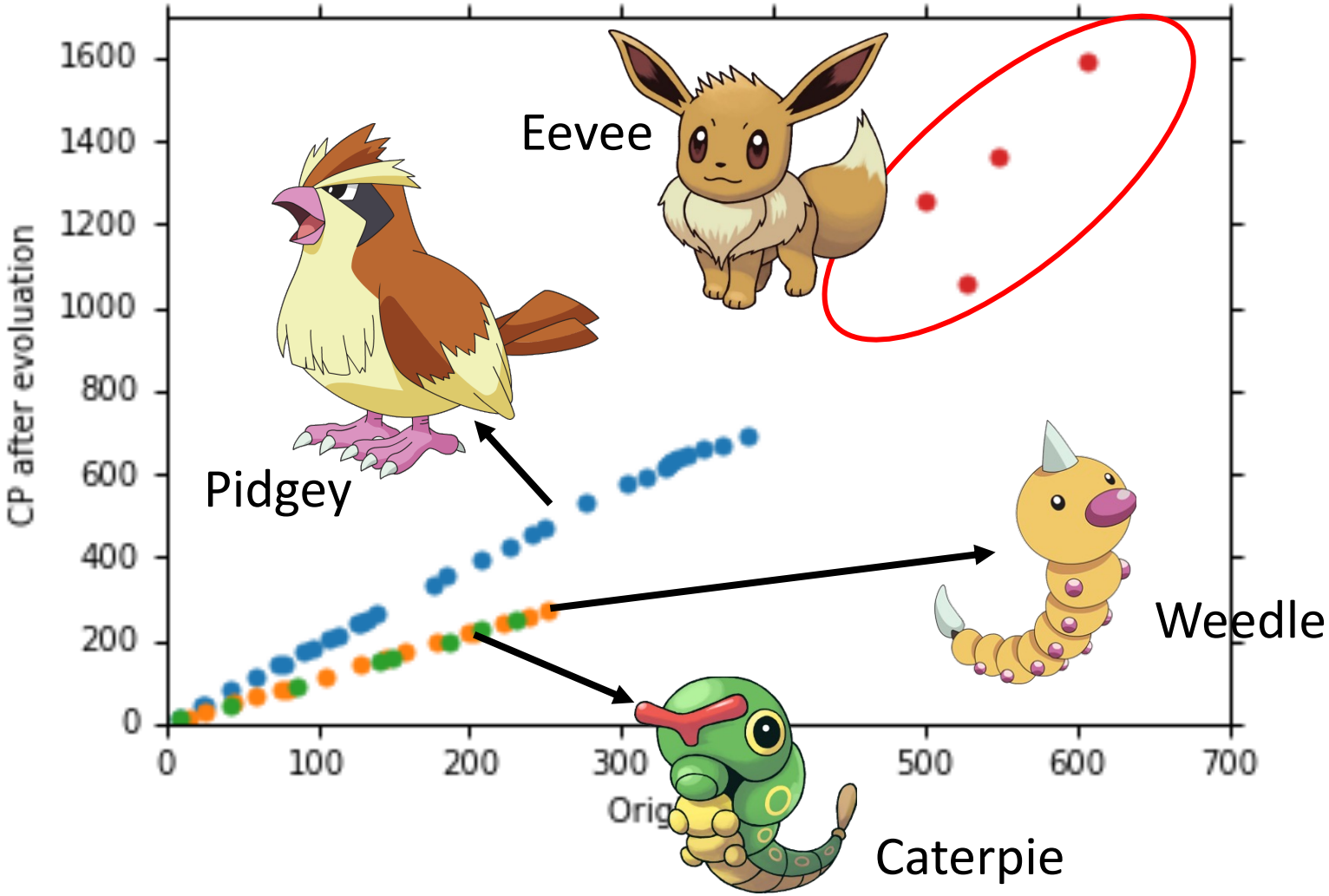
A more complex model does not always lead to better performance on ***testing data***.

This is ***Overfitting***. → Select suitable model

Let's collect more data



What are the hidden factors?



Back to step 1: Redesign the Model

$$y = b + \sum w_i x_i$$

Linear model?

x_s = species of x

x

If x_s = Pidgey: $y = b_1 + w_1 \cdot x_{cp}$

If x_s = Weedle: $y = b_2 + w_2 \cdot x_{cp}$

If x_s = Caterpie: $y = b_3 + w_3 \cdot x_{cp}$

If x_s = Eevee: $y = b_4 + w_4 \cdot x_{cp}$

y

Back to step 1: Redesign the Model

$$y = b + \sum w_i x_i$$

Linear model?

$$y = b_1 \cdot 1 + w_1 \cdot 1 \; x_{cp} + b_2 \cdot 0 + w_2 \cdot 0 + b_3 \cdot 0 + w_3 \cdot 0 + b_4 \cdot 0 + w_4 \cdot 0$$

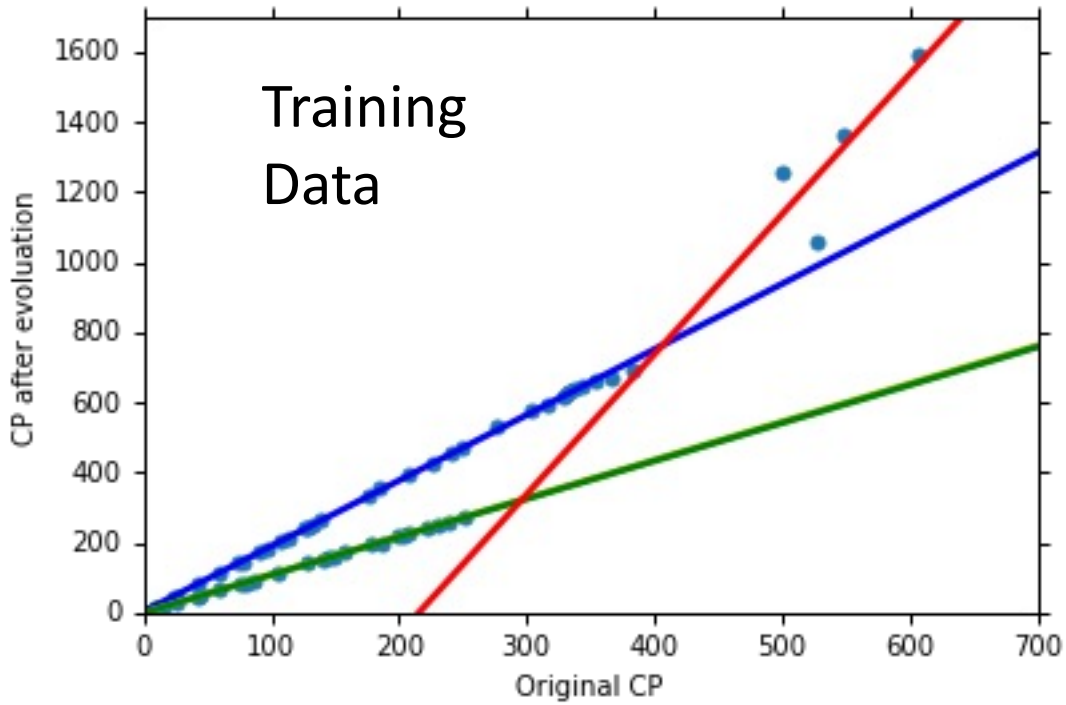
$\delta(x_s = \text{Pidgey})$

$$\begin{cases} =1 & \text{If } x_s = \text{Pidgey} \\ =0 & \text{otherwise} \end{cases}$$

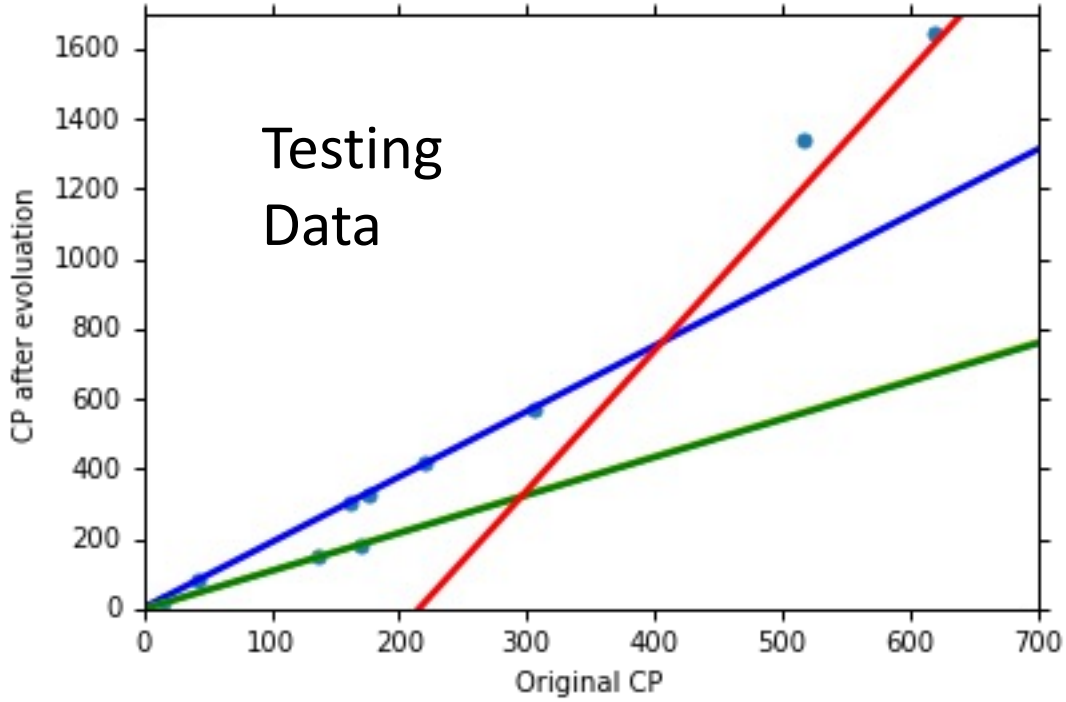
If $x_s = \text{Pidgey}$

$$y = b_1 + w_1 \cdot x_{cp}$$

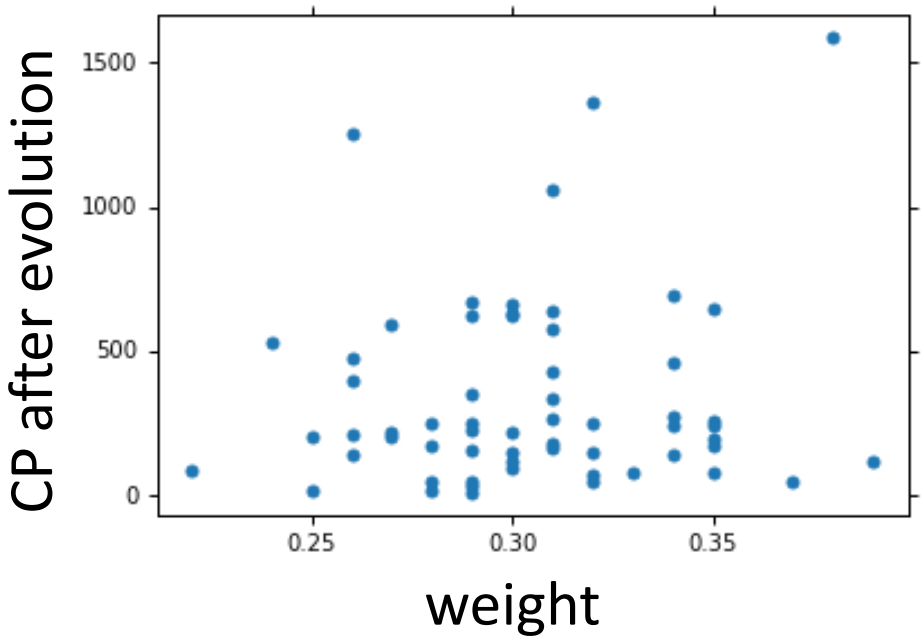
Average error
= 3.8

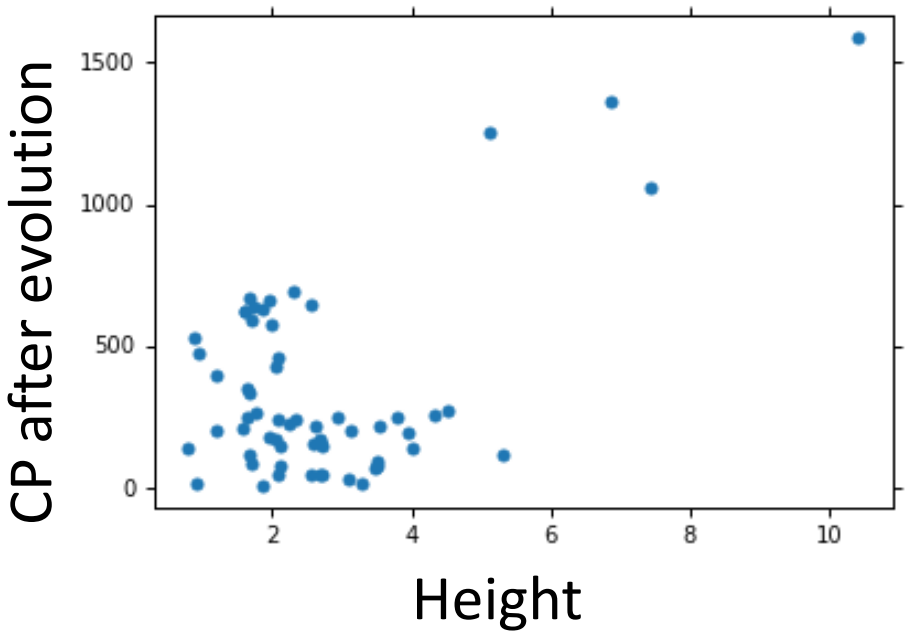


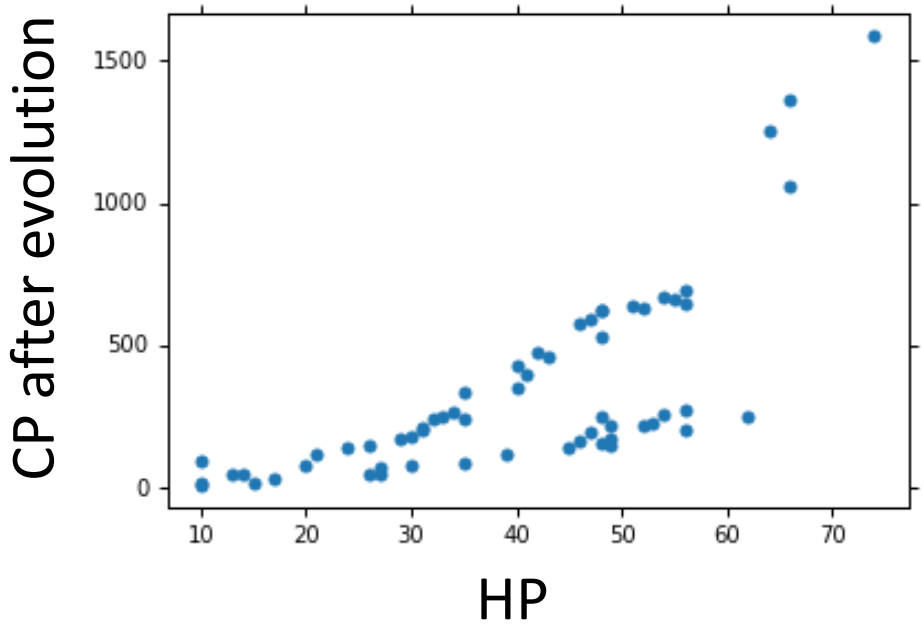
Average error
= 14.3



Are there any other hidden factors?







Back to step 1: Redesign the Model Again

x

If x_s = Pidgey: $y' = b_1 + w_1 \cdot x_{cp} + w_5 \cdot (x_{cp})^2$

If x_s = Weedle: $y' = b_2 + w_2 \cdot x_{cp} + w_6 \cdot (x_{cp})^2$

If x_s = Caterpie: $y' = b_3 + w_3 \cdot x_{cp} + w_7 \cdot (x_{cp})^2$

If x_s = Eevee: $y' = b_4 + w_4 \cdot x_{cp} + w_8 \cdot (x_{cp})^2$

$$y = y' + w_9 \cdot x_{hp} + w_{10} \cdot (x_{hp})^2 + w_{11} \cdot x_h + w_{12} \cdot (x_h)^2 + w_{13} \cdot x_w + w_{14} \cdot (x_w)^2$$

y

Training Error = 1.9

Testing Error = 102.3

Overfitting!

Back to step 2: Regularization

$$y = b + \sum w_i x_i$$

The functions with smaller w_i are better

$$L = \sum_n \left(\hat{y}^n - \left(b + \sum w_i x_i \right) \right)^2 \quad \boxed{+\lambda \sum (w_i)^2}$$

- Smaller w_i means ... smoother

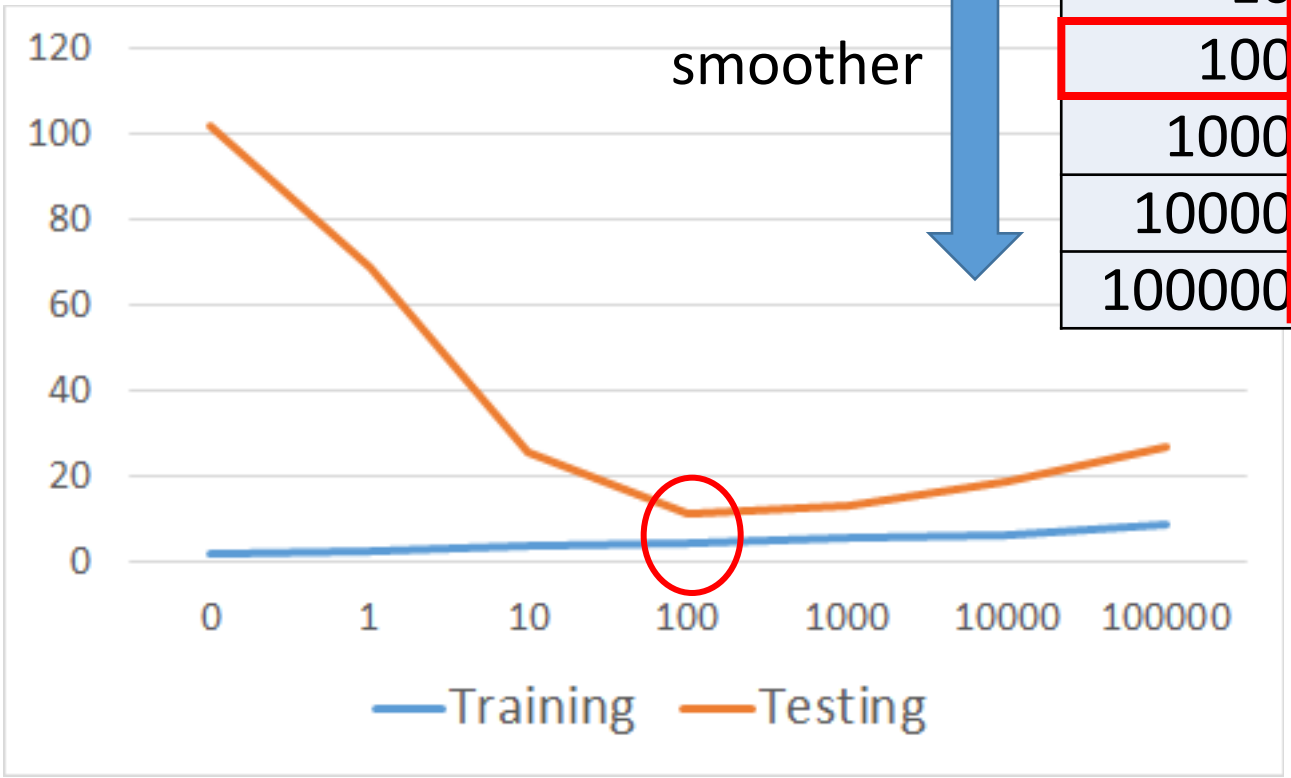
$$y = b + \sum w_i x_i$$

$$y + \sum w_i \Delta x_i = b + \sum w_i (x_i + \Delta x_i)$$

- We believe smoother function is more likely to be correct

Do you have to apply regularization on bias?

Regularization



λ	Training	Testing
0	1.9	102.3
1	2.3	68.7
10	3.5	25.7
100	4.1	11.1
1000	5.6	12.8
10000	6.3	18.7
100000	8.5	26.8

How smooth?

Select λ obtaining the best model

- Training error: larger λ, considering the training error less
- We prefer smooth function, but don't be too smooth.

Polynomial Linear Regression

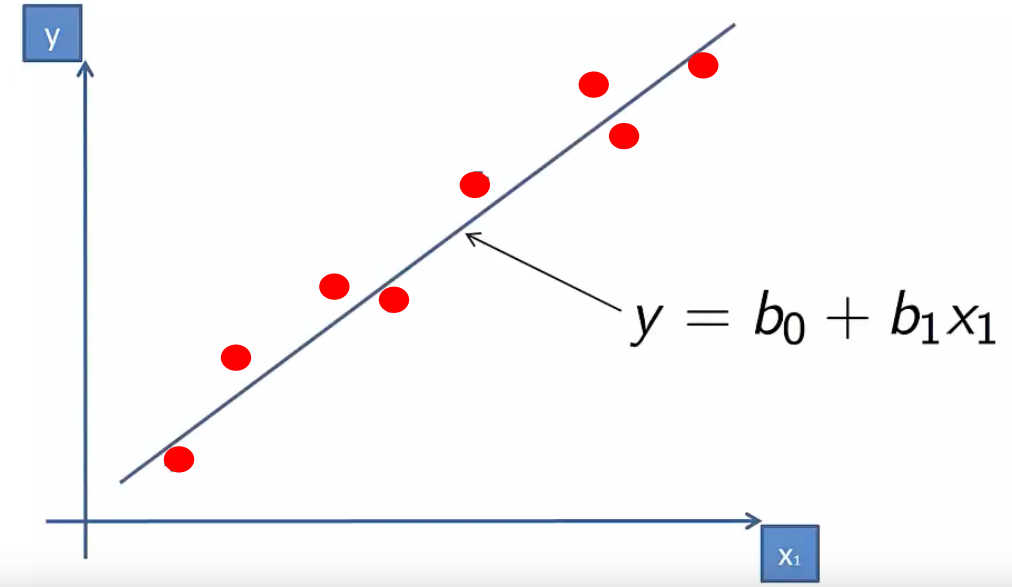
Linear Regression:

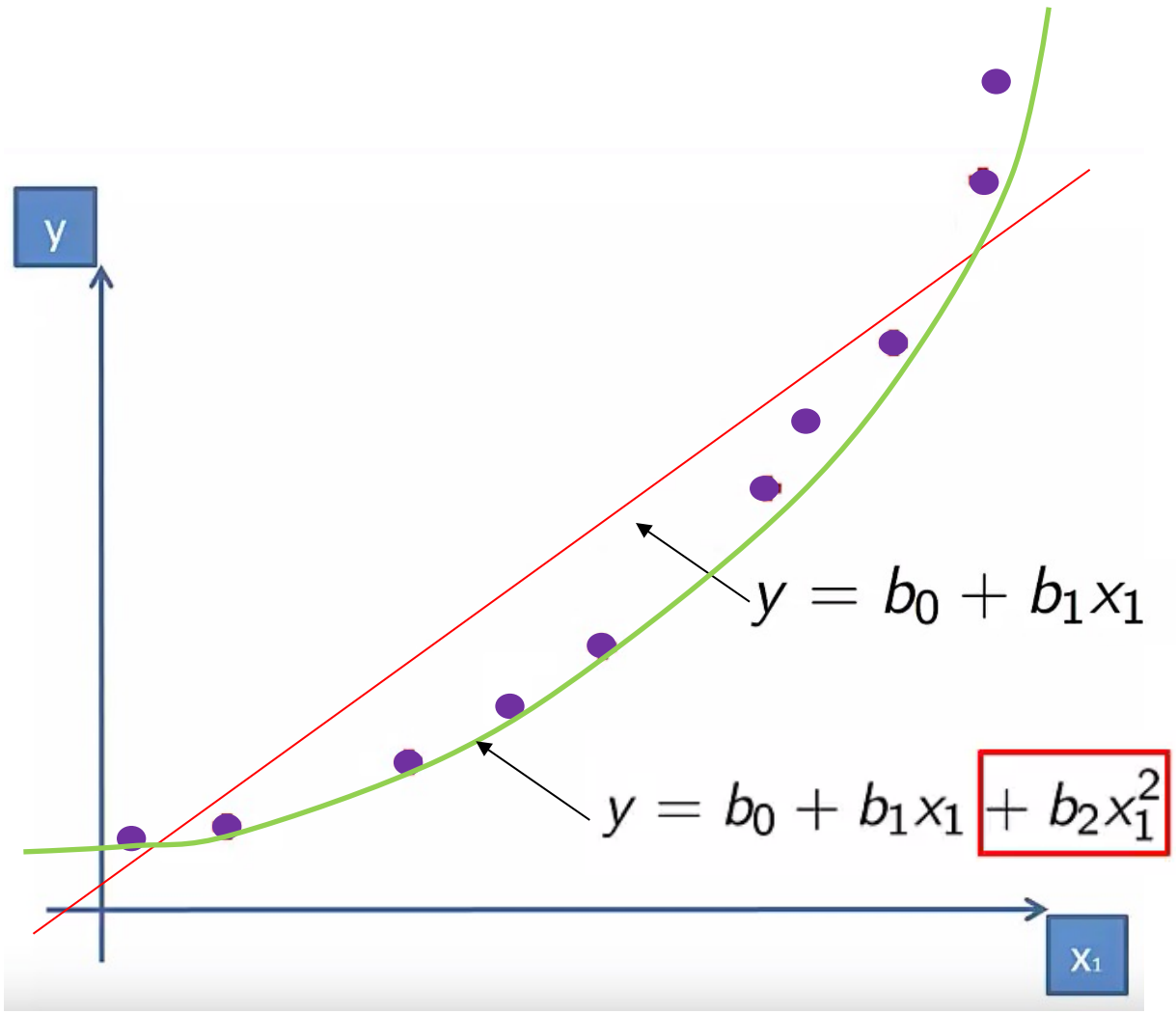
$$y = b_0 + b_1 x_1$$

Polynomial Linear Regression:

$$y = b_0 + b_1 x_1 + b_2 x_1^2 + \ldots + b_n x_1^n$$

Why polynomial regression




y
$y = b_0 + b_1 x_1$
$y = b_0 + b_1 x_1 + b_2 x_1^2$
x_1

Why still called “Linear”?

Polynomial Linear Regression:

$$y = b_0 + b_1 x_1 + b_2 x_1^2 + \ldots + b_n x_1^n$$

The function is expressed as the linear combination of unknowns (i.e. coefficients)?

Logistic regression

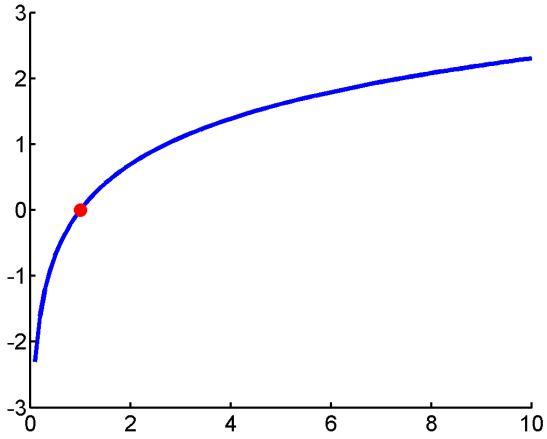
Logistic Regression

- Regression used to fit a curve to data in which the dependent variable is binary.
- Given some event with probability *p* of being 1, the **odds** of that event are given by:

$$\text{odds} = p / (1-p)$$

- The logit is the natural log of the odds
  logit(p) = ln(odds) = ln (p/(1-p))

Logit Transform



Logistic Regression

- In logistic regression, we seek a model:

$$\text{logit}(p) = \beta_0 + \beta_1 X$$

- That is, the log odds (logit) is assumed to be linearly related to the independent variable X
- So, now we can focus on solving an ordinary (linear) regression.

Logistic Regression-Recovering Probabilities

$$\ln\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 X$$

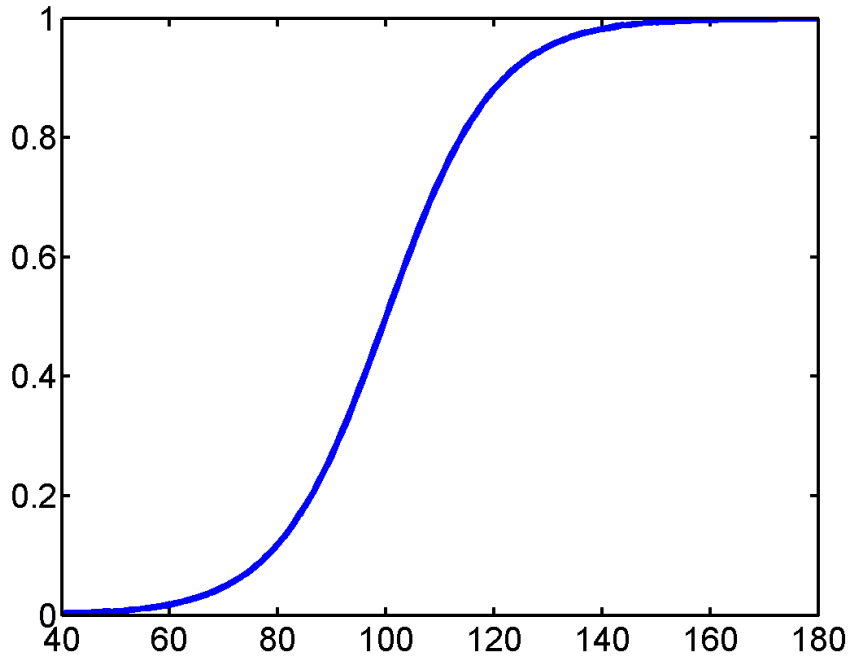
$$\Leftrightarrow \frac{p}{1-p} = e^{\beta_0 + \beta_1 X}$$

$$\Leftrightarrow p = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}} = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X)}}$$

which gives *p* as a sigmoid function!

Logistic Regression-Logistic Response Function

- When the response variable is binary, the shape of the response function is often sigmoidal:



Example Logistic Regression

- One application: The clinical research is interested in predicting response to treatment, where we might code survivors as 1 and those who don't survive as 0

10	20	30							90
8	5	6							0
0	0	1							10

-
+

Typical application: Linear regression results

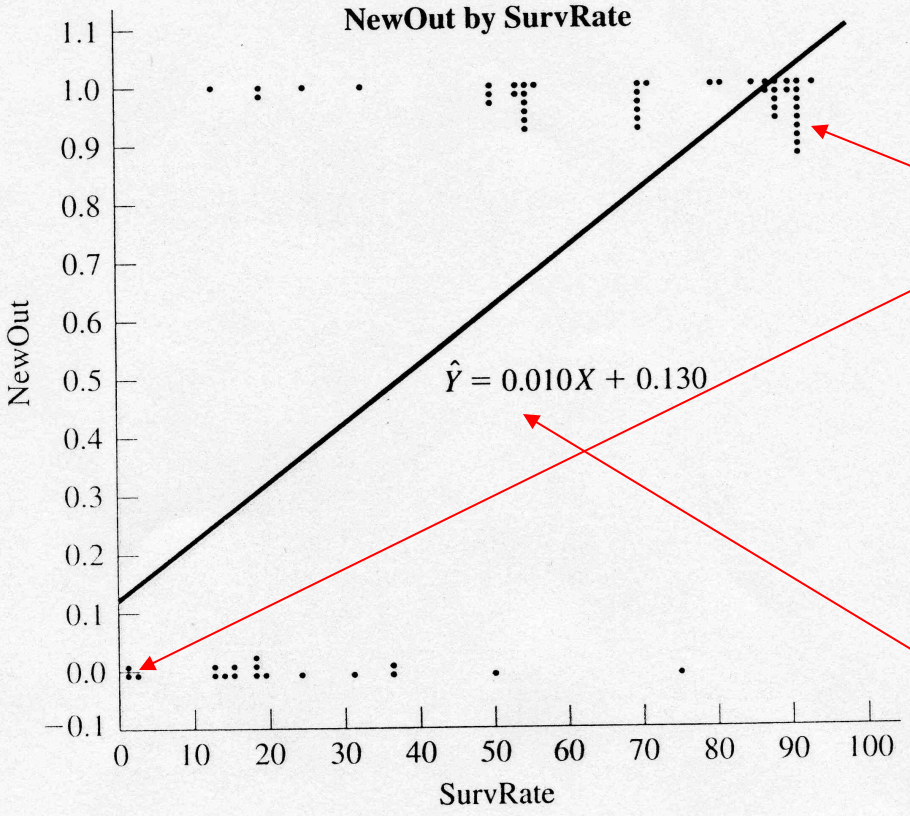


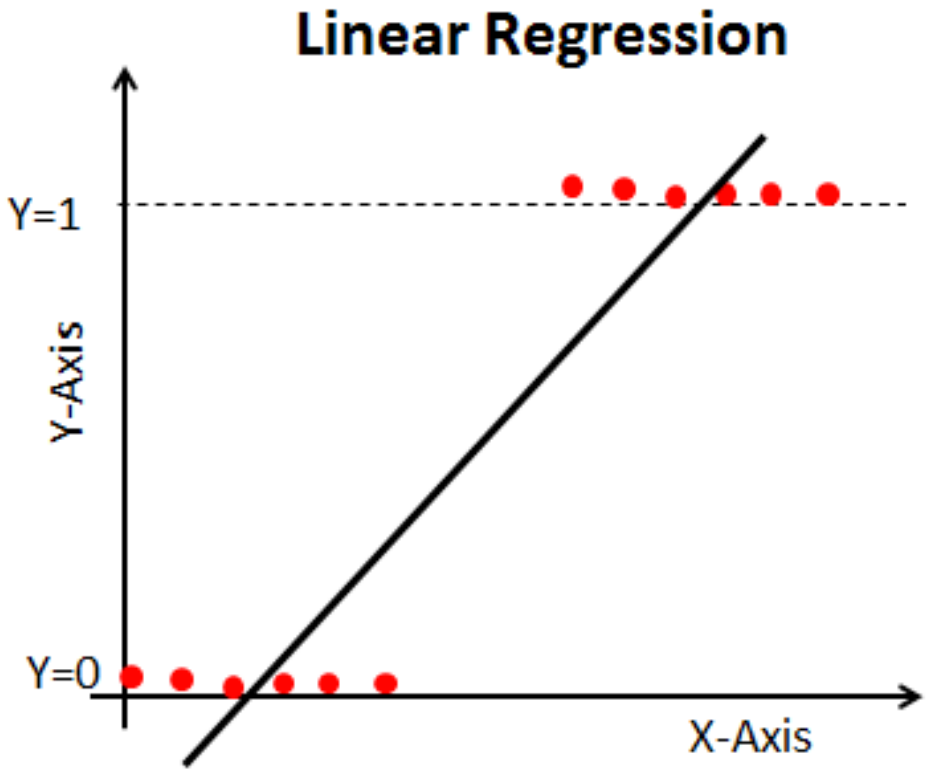
FIGURE 15.7 *Outcome as a function of SurvRate*

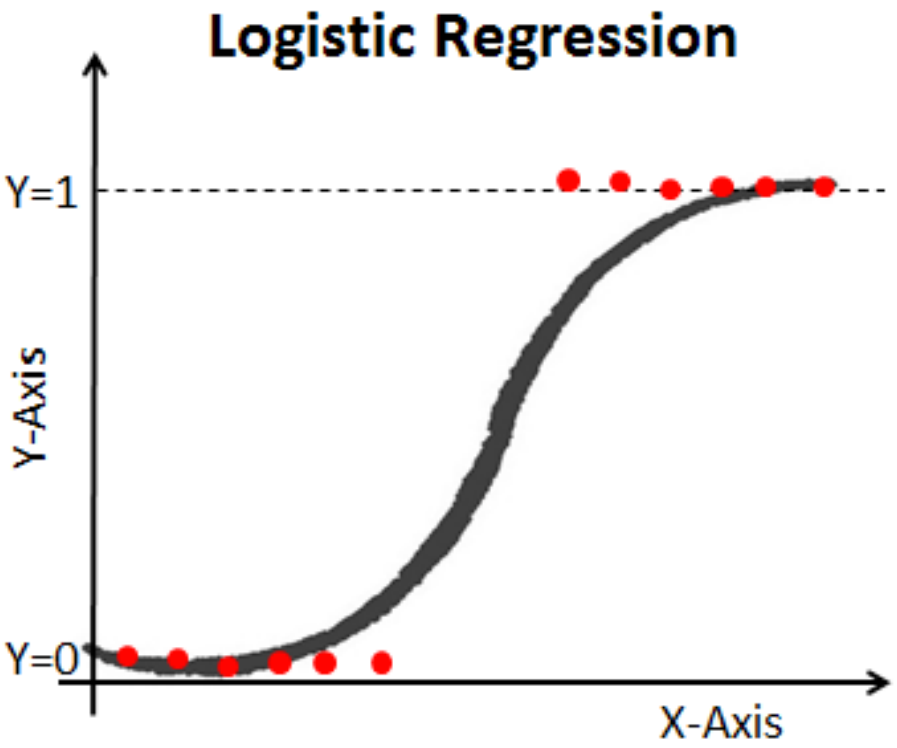
Observations:
For each value of SurvRate, the number of dots is the number of patients with that value of NewOut

Regression:
Standard linear regression

Problem: extending the regression line a few units left or right along the X axis produces predicted probabilities that fall outside of [0,1]

Comparison:


Linear Regression
Y=1
Y-Axis
Y=0
X-Axis


Logistic Regression
Y=1
Y-Axis
Y=0
X-Axis

Typical application: Medicine- A Better Solution

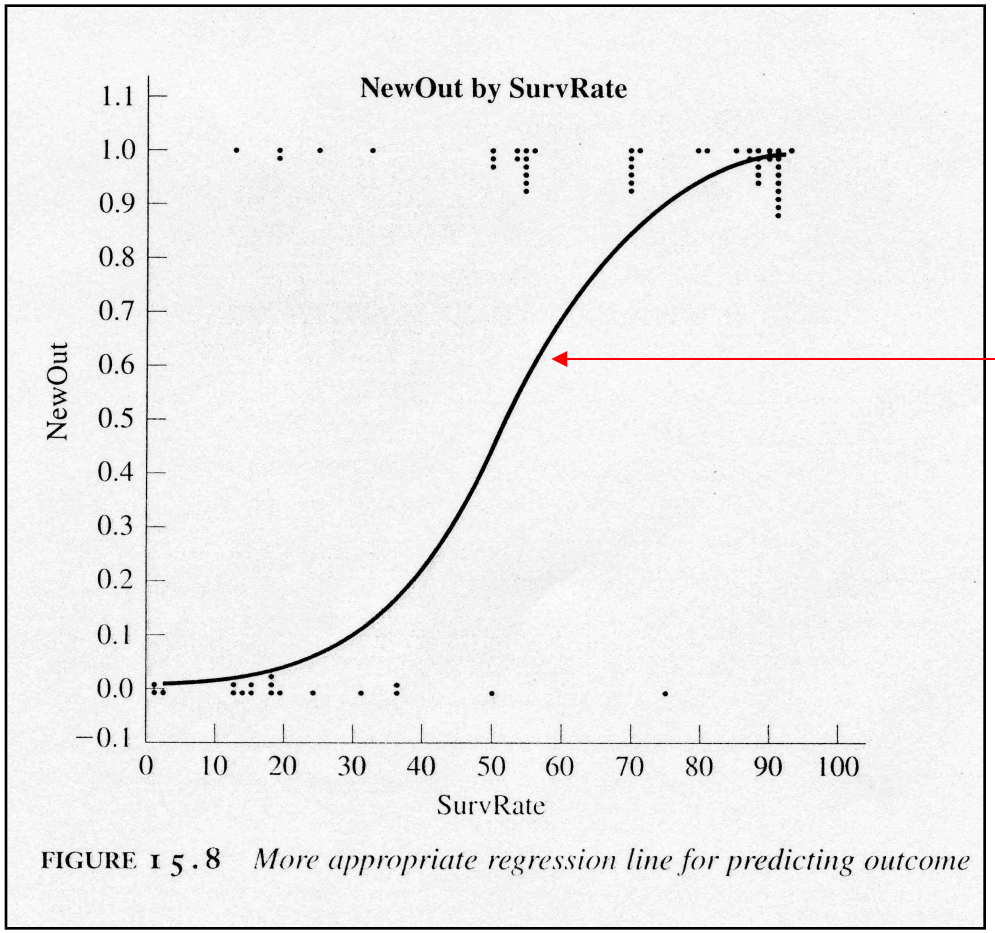


FIGURE 15.8 *More appropriate regression line for predicting outcome*

Regression Curve: Sigmoid function!

(bounded by asymptotes y=0 and y=1)

Introduce:

$$\sigma(z) = \frac{1}{1+e^{-z}}$$ sigmoid function

Then:

Logistic Regression is a classification algorithm (I know, terrible name) that works by trying to learn a function that approximates $P(Y|X)$. It makes the central assumption that $P(Y|X)$ can be approximated as a sigmoid function applied to a linear combination of input features. Mathematically, for a single training datapoint $(\mathbf{x}, y)$ Logistic Regression assumes:

$$P(Y = 1|\mathbf{X} = \mathbf{x}) = \sigma(z) \text{ where } z = \theta_0 + \sum_{i=1}^{m} \theta_i x_i$$

This assumption is often written in the equivalent forms:

$$P(Y = 1|\mathbf{X} = \mathbf{x}) = \sigma(\theta^T \mathbf{x}) \quad \text{where we always set } x_0 \text{ to be 1}$$

$$P(Y = 0|\mathbf{X} = \mathbf{x}) = 1 - \sigma(\theta^T \mathbf{x}) \quad \text{by total law of probability}$$

Training Logistic Regression Model

- Model:

$$P(Y = 1 | \mathbf{X} = \mathbf{x}) = \sigma(\theta^T \mathbf{x})$$

$$P(Y = 0 | \mathbf{X} = \mathbf{x}) = 1 - \sigma(\theta^T \mathbf{x})$$

Loss Function

- Loss is defined as the difference between the ground truth (actual values) and the predicted value
- So how we define loss function for logistic regression?

Likelihood Function

$$L(\theta) = \prod_{i=1}^{n} P(Y = y^{(i)} | X = \mathbf{x}^{(i)})$$

$$= \prod_{i=1}^{n} \sigma(\theta^T \mathbf{x}^{(i)})^{y^{(i)}} \cdot \left[1 - \sigma(\theta^T \mathbf{x}^{(i)})\right]^{(1-y^{(i)})}$$

Log Likelihood Function

$$LL(\theta) =$$

$$\sum_{i=0}^{n} y^{(i)} \log \sigma(\theta^T \mathbf{x}^{(i)}) + (1 - y^{(i)}) \log[1 - \sigma(\theta^T \mathbf{x}^{(i)})]$$

Gradient of the Function

$$LL(\theta) = \sum_{i=0}^{n} y^{(i)} \log \sigma(\theta^T \mathbf{x}^{(i)}) + (1 - y^{(i)}) \log[1 - \sigma(\theta^T \mathbf{x}^{(i)})]$$

Partial Derivative:

$$\frac{\partial LL(\theta)}{\partial \theta_j} = \sum_{i=0}^{n} \left[y^{(i)} - \sigma(\theta^T \mathbf{x}^{(i)}) \right] x_j^{(i)}$$

How 

Loss Function:

$$LL(\theta) = \sum_{i=0}^{n} y^{(i)} \log \sigma(\theta^T \mathbf{x}^{(i)}) + (1 - y^{(i)}) \log[1 - \sigma(\theta^T \mathbf{x}^{(i)})]$$

Derivative of sigma:

$$\frac{\partial}{\partial z} \sigma(z) = \sigma(z)[1 - \sigma(z)]$$

Then:

Derivative of gradient for one datapoint $(\mathbf{x}, y)$:

$$\frac{\partial LL(\theta)}{\partial \theta_j} = \frac{\partial}{\partial \theta_j} y \log \sigma(\theta^T \mathbf{x}) + \frac{\partial}{\partial \theta_j} (1-y) \log[1 - \sigma(\theta^T \mathbf{x}] \qquad \text{derivative of sum of terms}$$

$$= \left[\frac{y}{\sigma(\theta^T x)} - \frac{1-y}{1-\sigma(\theta^T x)} \right] \frac{\partial}{\partial \theta_j} \sigma(\theta^T x) \qquad \text{derivative of } \log f(x)$$

$$= \left[\frac{y}{\sigma(\theta^T x)} - \frac{1-y}{1-\sigma(\theta^T x)} \right] \sigma(\theta^T x)[1 - \sigma(\theta^T x)] x_j \qquad \text{chain rule + derivative of sigma}$$

$$= \left[\frac{y - \sigma(\theta^T x)}{\sigma(\theta^T x)[1 - \sigma(\theta^T x)]} \right] \sigma(\theta^T x)[1 - \sigma(\theta^T x)] x_j \qquad \text{algebraic manipulation}$$

$$= \left[y - \sigma(\theta^T x) \right] x_j \qquad \text{cancelling terms}$$

Loss Function

Loss Function: $-LL(\theta)$

$$-\left[\sum_{i=0}^{n} y^{(i)} \log \sigma(\theta^T \mathbf{x}^{(i)}) + (1 - y^{(i)}) \log[1 - \sigma(\theta^T \mathbf{x}^{(i)})]\right]$$

Training Method: Gradient Descent

- Gradient descent algorithm's main objective is to minimize the cost function. It is one of the best optimization algorithms to minimize errors (difference of actual value and predicted value).

Updating rule:

$$\theta_j^{\text{new}} = \theta_j^{\text{old}} + \eta \cdot \frac{\partial LL(\theta^{\text{old}})}{\partial \theta_j^{\text{old}}}$$

$$= \theta_j^{\text{old}} + \eta \cdot \sum_{i=0}^{n} \left[y^{(i)} - \sigma(\theta^T \mathbf{x}^{(i)}) \right] x_j^{(i)}$$

Python Function

```
def sigmoid(x):
    # Activation function used to map any real value between 0 and 1
    return 1 / (1 + np.exp(-x))

def net_input(theta, x):
    # Computes the weighted sum of inputs
    return np.dot(x, theta)

def probability(theta, x):
    # Returns the probability after passing through sigmoid
    return sigmoid(net_input(theta, x))
```

Next, we define the `cost` and the `gradient` function.

```
def cost_function(self, theta, x, y):
    # Computes the cost function for all the training samples
    m = x.shape[0]
    total_cost = -(1 / m) * np.sum(
        y * np.log(probability(theta, x)) + (1 - y) * np.log(
            1 - probability(theta, x)))
    return total_cost

def gradient(self, theta, x, y):
    # Computes the gradient of the cost function at the point theta
    m = x.shape[0]
    return (1 / m) * np.dot(x.T, sigmoid(net_input(theta,   x)) - y)
```

Fitting function:

Let's also define the `fit` function which will be used to find the model parameters that minimizes the cost function. In this blog, we coded the gradient descent approach to compute the model parameters. Here, we will use `fmin_tnc` function from the `scipy` library. It can be used to compute the minimum for any function. It takes arguments as

- func: the function to minimize
- x0: initial values for the parameters that we want to find
- fprime: gradient for the function defined by 'func'
- args: arguments that needs to be passed to the functions.

```
def fit(self, x, y, theta):
    opt_weights = fmin_tnc(func=cost_function, x0=theta,
                  fprime=gradient,args=(x, y.flatten()))
    return opt_weights[0]

parameters = fit(X, y, theta)
```

Reference

- https://ocw.mit.edu/courses/mathematics/18-05-introduction-to-probability-and-statistics-spring-2014/class-slides/MIT18_05S14_class25slides.pdf
- http://finance.wharton.upenn.edu/~mrrobert/resources/Teaching/CorpFinPhD/Linear-Regression-Slides.pdf
- http://www-hsc.usc.edu/~eckel/biostat2/slides/lecture13.pdf
- https://towardsdatascience.com/linear-regression-using-gradient-descent-97a6c8700931
- We have used huge amount of online resources for this course. All of them are the sole copyright holders of their material. Here we have referenced them with proper credits.
- https://towardsdatascience.com/building-a-logistic-regression-in-python-301d27367c24