Northeastern University
College *of* Engineering

IE6600-Workshop

Logistic Regression

Zhenyuan Lu

0. Introduction

Introduction *Some Denotations in Machine Learning*

A set of n×1 dimension $D(X,Y) = \{(x_i, y_i)\}_{i=1}^{n}, (x_i, y_i) \in \mathbb{R}^n \times \mathbb{R}^n$

Statistics:

$Y = \beta_0 + \beta_1 X * + \epsilon$

Machine Learning:

$Y = f(\mathrm{x}) = w_0 + w_1 x + \mathrm{b}$

Jurafsky 2019

Introduction *Generative and Discriminative Classifier*

1. Generative classifiers (e.g., Naïve Bayes):
   Imagine we're trying to distinguish dog images from cat images. A generative model would have the goal of understanding what dogs look like and what cats look like. You might literally ask such a model to 'generate', i.e. draw, a dog. Given a test image, the system then asks whether it's the cat model or the dog model that better fits (is less surprised by) the image, and chooses that as its label.

2. Discriminative classifiers (e.g., Logistic regression) :
   A discriminative model, by contrast, is only trying to learn to distinguish the classes (perhaps without learning much about them).

Jurafsky 2019

Introduction *Generative and Discriminative Classifier*

A set of n points X_i in a d- dimension space, y_i denote the class for each point, with $y_i \in \{c_1, c_2, \dots, c_k\}$. Training classifiers estimates: $P(c_i|X)$

1. Generative classifiers (e.g., Naïve Bayes):
   a) Assumptions on: $P(X|c_i)P(c_i)$ $\hat{y} = arg\max_i\{P(X|c_i)P(c_i)\}$
   b) Estimate parameters of $P(X|c_i)P(c_i)$ directly from training data
   c) Use Bayes theorem to calculate $P(c_i|X)$

2. Discriminative classifiers (e.g., Logistic regression) :
   a) Assumptions on: $P(c_i|X)$
   b) Estimate parameters of $P(c_i|X)$ directly from training data

Introduction *Components of a probabilistic machine learning classifier*

Like naive Bayes, logistic regression is a probabilistic classifier that makes use of supervised machine learning.

1. A feature representation of the input. For each input observation $x_i = \{x_1, \ldots, x_d\}$
2. A classification function that computes $\hat{y}$, the estimated class, via $P(c_i|X)$, we will introduce the **sigmoid**
3. An objective function for learning, usually involving minimizing error on training examples. We will introduce **the cross-entropy loss function**
4. An algorithm for optimizing the objective function. We introduce the **stochastic gradient descent algorithm.**

Logistic regression has two phases:

Training: we train the system (specifically the weights w and b) using stochastic gradient descent and the cross-entropy loss.

Test: Given a test example x we compute $P(c_i|X)$ and return the higher probability label $c = 1$ or $c = 0$.

1. Logistic Regression

Logistic Regression *Derivation*

Given:

1. Y is Boolean, governed by a Bernoulli distribution, with parameter $\pi = P(Y = 1)$
2. $X = \{X_1, \dots, X_n\}$, each X_i is a continuous random variable
3. For X_i, $P(X_i|Y = y_k)$ is a Gaussian distribution of the form $N(\mu_{ik}, \sigma_i)$, assume $\sigma_{ik} = \sigma_i$
4. $\forall i \text{ and } j{\neq}i, X_\mathrm{i}$ and X_j are conditionally independent given Y

Goal: Derive $P(Y = y_k|X)$

$$P(Y = 1|X) = \frac{P(Y = 1)P(X|Y = 1)}{P(Y = 1)P(X|Y = 1) + P(Y = 0)P(X|Y = 0)}$$

$$= \frac{1}{1 + \frac{P(Y = 0)P(X|Y = 0)}{P(Y = 1)P(X|Y = 1)}}$$

$$= \frac{1}{1 + \exp\left(\ln \frac{P(Y = 0)P(X|Y = 0)}{P(Y = 1)P(X|Y = 1)}\right)}$$

$$= \frac{1}{1 + \exp\left(\left(ln\frac{1-\pi}{\pi}\right) + \sum_i \ln\frac{P(X_i|Y = 0)}{P(X_i|Y = 1)}\right)}$$

Logistic Regression *Derivation*

$$P(Y=1|X)=\frac{1}{1+\exp\left(\left(ln\frac{1-\pi}{\pi}\right)+\sum_i \ln\frac{P(X_i|Y=0)}{P(X_i|Y=1)}\right)}$$

$$P(X_i|Y=y_k)=\frac{1}{\sqrt{2\pi}\sigma_{ik}}\exp\{-\frac{\left(x_j-\mu_{ik}\right)^2}{2\sigma_{ik}^2}\}$$

$$lnP(X_i|Y=y_k)=\frac{1}{\sqrt{2\pi}\sigma_{ik}}+\left(-\frac{\left(x_j-\mu_{ik}\right)^2}{2\sigma_{ik}^2}\right)$$

$$\ln\frac{P(X_i|Y=0)}{P(X_i|Y=1)}=\frac{1}{\sqrt{2\pi}\sigma_{i0}}+\left(-\frac{\left(x_j-\mu_{i0}\right)^2}{2\sigma_{i0}^2}\right)-\frac{1}{\sqrt{2\pi}\sigma_{i1}}+\left(-\frac{\left(x_j-\mu_{i1}\right)^2}{2\sigma_{i1}^2}\right)$$

$$=\frac{1}{\sqrt{2\pi}\sigma_{i0}}-\frac{1}{\sqrt{2\pi}\sigma_{i1}}-\frac{x_i^2-2x_i\mu_{i0}+\mu_{i0}^2}{2\sigma_{i0}^2}+\frac{x_i^2-2x_i\mu_{i1}+\mu_{i1}^2}{2\sigma_{i1}^2}$$

Assume $\sigma_{ik}=\sigma_i$

$$=\frac{\mu_{i0}-\mu_{i1}}{\sigma_i^2}X_i+\frac{\mu_{i1}^2-\mu_{i0}^2}{2\sigma_i^2}$$

Logistic Regression *Derivation*

$$P(Y=1|X) = \frac{1}{1+\exp\left(\left(ln\frac{1-\pi}{\pi}\right) + \sum_i \ln\frac{P(X_i|Y=0)}{P(X_i|Y=1)}\right)}$$

$$\ln\frac{P(X_i|Y=0)}{P(X_i|Y=1)} = \frac{\mu_{i0}-\mu_{i1}}{\sigma_i^2}X_i + \frac{\mu_{i1}^2-\mu_{i0}^2}{2\sigma_i^2}$$

$$P(Y=1|X) = \frac{1}{1+\exp\left(\left(ln\frac{1-\pi}{\pi}\right) + \sum_i\left(\frac{\mu_{i0}-\mu_{i1}}{\sigma_i^2}X_i + \frac{\mu_{i1}^2-\mu_{i0}^2}{2\sigma_i^2}\right)\right)}$$

Where weights are given by: $w_0 = ln\frac{1-\pi}{\pi} + \sum_i\left(\frac{\mu_{i1}^2-\mu_{i0}^2}{2\sigma_i^2}\right), w_1 = \frac{\mu_{i0}-\mu_{i1}}{\sigma_i^2}$

$$P(Y=1|X) = \frac{1}{1+\exp(w_0+\sum_{i=1}^n w_iX_i)}, P(Y=0|X) = \frac{\exp(w_0+\sum_{i=1}^n w_iX_i)}{1+\exp(w_0+\sum_{i=1}^n w_iX_i)}$$

Logistic Regression *Derivation*

$$P(Y=1|X)=\frac{1}{1+\exp(w_0+\sum_{i=1}^{n} w_iX_i)}$$

$$P(Y=0|X)=\frac{\exp(w_0+\sum_{i=1}^{n} w_iX_i)}{1+\exp(w_0+\sum_{i=1}^{n} w_iX_i)}$$

Then we get

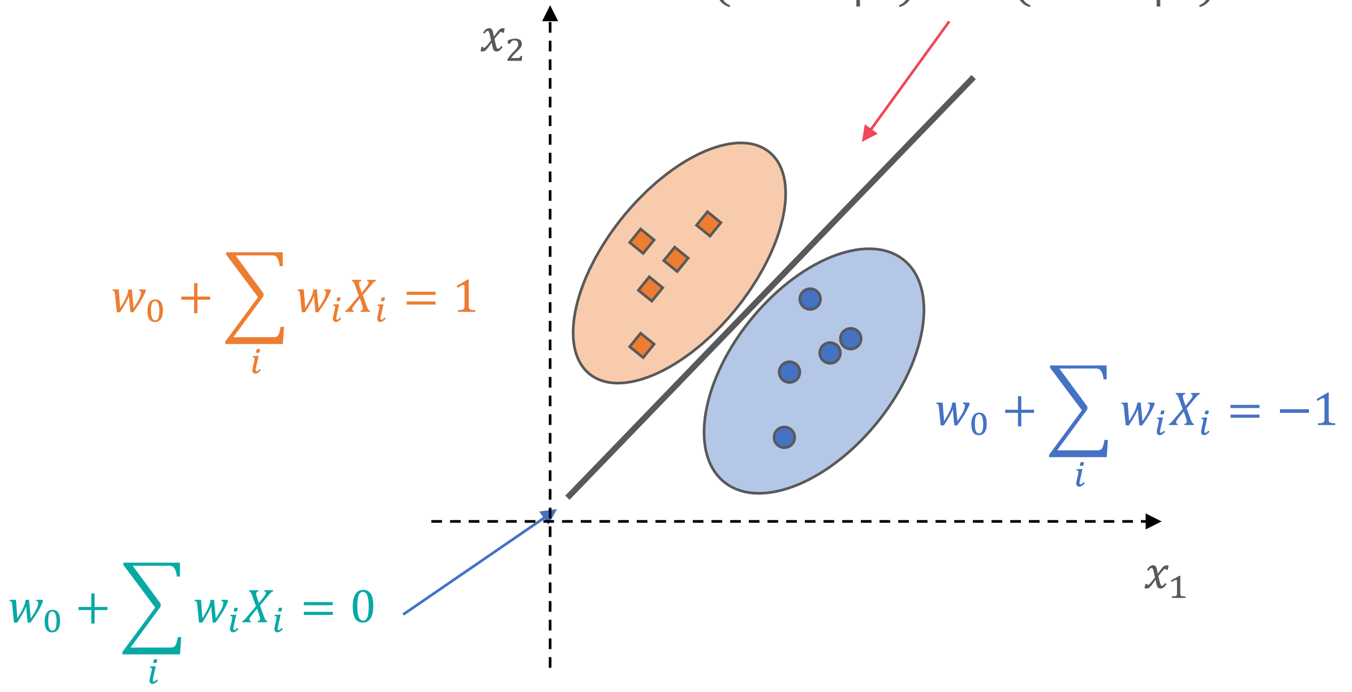
$$\frac{P(Y=0|X)}{P(Y=1|X)}=\exp\left(w_0+\sum_i w_iX_i\right)$$

$$ln\frac{P(Y=0|X)}{P(Y=1|X)}=w_0+\sum_i w_iX_i$$

Linear classification rule

$$z=ln\frac{P(Y=0|X)}{P(Y=1|X)}=0$$

$$P(Y=1|X)=P(Y=0|X)$$



Logistic Regression *Derivation*

$$P(Y=1|X) = \frac{1}{1+\exp(w_0 + \sum_{i=1}^{n} w_i X_i)}$$
$$P(Y=0|X) = \frac{\exp(w_0 + \sum_{i=1}^{n} w_i X_i)}{1+\exp(w_0 + \sum_{i=1}^{n} w_i X_i)}$$

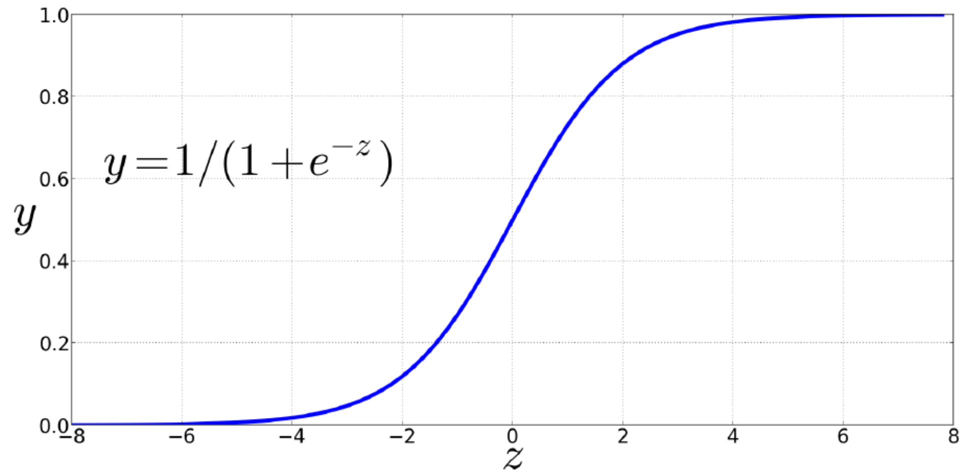
Then we get

$$\frac{P(Y=0|X)}{P(Y=1|X)} = \exp\left(w_0 + \sum_i w_i X_i\right)$$

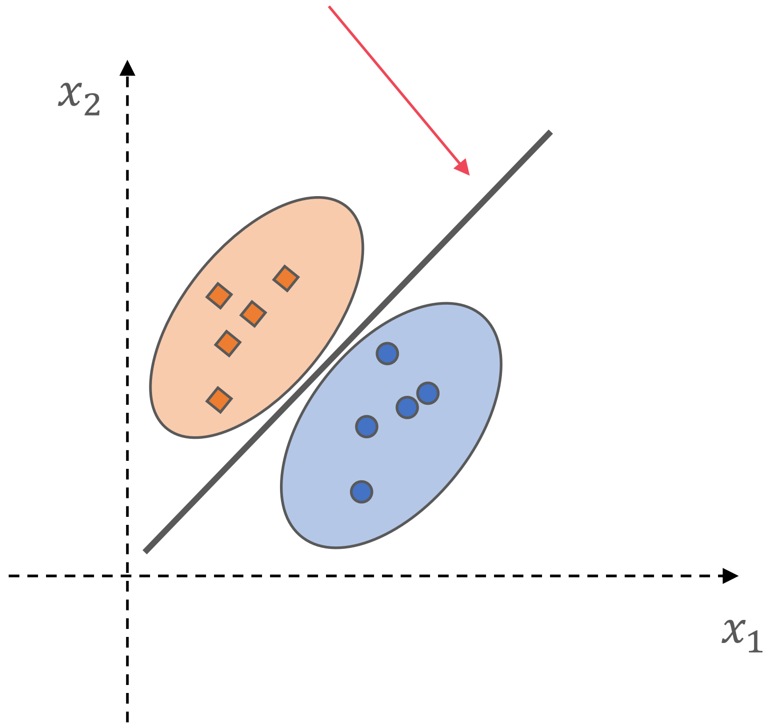
$$\underbrace{ln\frac{P(Y=0|X)}{P(Y=1|X)}}_{\text{Log-odds or logit transformations}} = w_0 + \sum_i w_i X_i \leftarrow \text{This reflects the constraint that the probabilities sum to one}$$

Logistic Regression *Derivation*

$$P(Y=0|X) = \frac{1}{1+\exp-(w_0+\sum_{i=1}^{n} w_i X_i)}$$



$$z = ln\frac{P(Y=0|X)}{P(Y=1|X)} = w_0 + \sum_i w_i X_i = 0$$



Logistic Regression *More general*

Given: Logistic regression when $Y = \{y_1, \dots y_K\}$

$$P(Y = y_K|X) = \frac{1}{1 + \sum_{j=1}^{K-1} \exp(w_{j0} + \sum_{i=1}^{n} w_{ji}X_i)}$$

$$P(Y = y_k|X) = \frac{\exp(w_{k0} + \sum_{i=1}^{n} w_{ki}X_i)}{1 + \sum_{j=1}^{K-1} \exp(w_{j0} + \sum_{i=1}^{n} w_{ji}X_i)}, k = 1, \dots, K-1$$

Logistic Regression *The sigmoid function*

$$z = w_0 + \sum_i w_i X_i = w \cdot x + b\ (equivalent\ to\ w^T x + b)$$

The sigmoid function $\hat{y} = \sigma(w \cdot x + b) = \frac{1}{1+e^{-z}}$

Logistic Regression *The cross-entropy loss function*

We need a loss function that expresses, for an observation x, how close the classifier output $\hat{y} = \sigma(w \cdot x + b)$ is to the correct output (y, which is 0 or 1). We'll call this:

$L(\hat{y}, y) =$How much $\hat{y}$ differs from the true y

We do this via a loss function that prefers the correct class labels of the training examples to be *more likely*. This is called conditional maximum likelihood estimation: we choose the parameters $\boldsymbol{w}, \boldsymbol{b}$ that maximize the log probability of the true $\boldsymbol{y}$ labels in the training data given the observations $\boldsymbol{x}$. The resulting loss function is the negative log likelihood loss, generally called the cross-entropy loss.

Logistic Regression *The cross-entropy loss function*

Given: $y = \{0, 1\}$, $\mathbf{Bernoulli\ distribution}$, which can be expressed as probability $p(y|x)$. If $y = 1$, simplifies to $\hat{y}$; if $y = 0$, simplifies to $1 - \hat{y}$
Goal: Maximize the probability of $\hat{y}$

$$p(y|x) = \hat{y}^{y}(1 - \hat{y})^{1-y}$$

Now we take the log of both sides. This will turn out to be handy mathematically, and doesn’t hurt us.

$$\begin{aligned} \log p(y|x) &= \log[\hat{y}^{y}(1 - \hat{y})^{1-y}] \\ &= y\log\hat{y} + (1 - y)\log(1 - \hat{y}) \end{aligned}$$

Logistic Regression *The cross-entropy loss function*

$$\log p(y|x) = y\log \hat{y} + (1-y)\log(1-\hat{y})$$

This shows a log likelihood that should be maximized. In order to turn this into loss function (something that we need to minimize), we'll just flip the sign. The result is the cross-entropy loss L_{CE}

$$L_{CE}(\hat{y}, y) = -\log p(y|x) = -[y\log \hat{y} + (1-y)\log(1-\hat{y})]$$

Finally, we can plug in the definition of $\hat{y} = \sigma(w \cdot x + b)$:

$$L_{CE}(w, b) = -[y\log \sigma(w \cdot x + b) + (1-y)\log(1-\sigma(w \cdot x + b))]$$

Logistic Regression *The cross-entropy loss function*

Why does minimizing this negative log probability do what we want?

A perfect classifier would assign probability 1 to the correct outcome (y=1 or y=0) and probability 0 to the incorrect outcome. That means the higher $\hat{y}$ (the closer it is to 1), the better the classifier; the lower $\hat{y}$ is (the closer it is to 0), the worse the classifier. The negative log of this probability is a convenient loss metric since it goes from 0 (negative log of 1, no loss) to infinity (negative log of 0, infinite loss). This loss function also ensures that as the probability of the correct answer is maximized, the probability of the incorrect answer is minimized; since the two sum to one, any increase in the probability of the correct answer is coming at the expense of the incorrect answer. It's called the cross-entropy loss,

Logistic Regression *Gradient descent*

Our goal with gradient descent is to find the optimal weights: minimize the loss Function. Below function represents the loss function $\boldsymbol{L}$ is parameterized by the weights with $\boldsymbol{m}$ training examples, which we refer to as $\boldsymbol{\theta}$. In the case of logistic regression $\theta = w, b$.

$$\hat{\theta} = \underset{\theta}{\arg\min} \frac{1}{m} \sum_{i=1}^{m} L_{CE}(y_i, x_i; \theta)$$

Gradient descent is a method that finds a minimum of a function by figuring out in which direction (in the space of the parameters q) the function's slope is rising the most steeply, and moving in the opposite direction.

Logistic Regression *Gradient descent*

This is You
Bottom
https://www.nps.gov/grca/planyourvisit/one-day-river-trip.htm

The intuition is that if you are hiking in a canyon and trying to descend most quickly down to the river at the bottom, you might look around yourself 360 degrees, find the direction where the ground is sloping the steepest, and walk downhill in that direction.

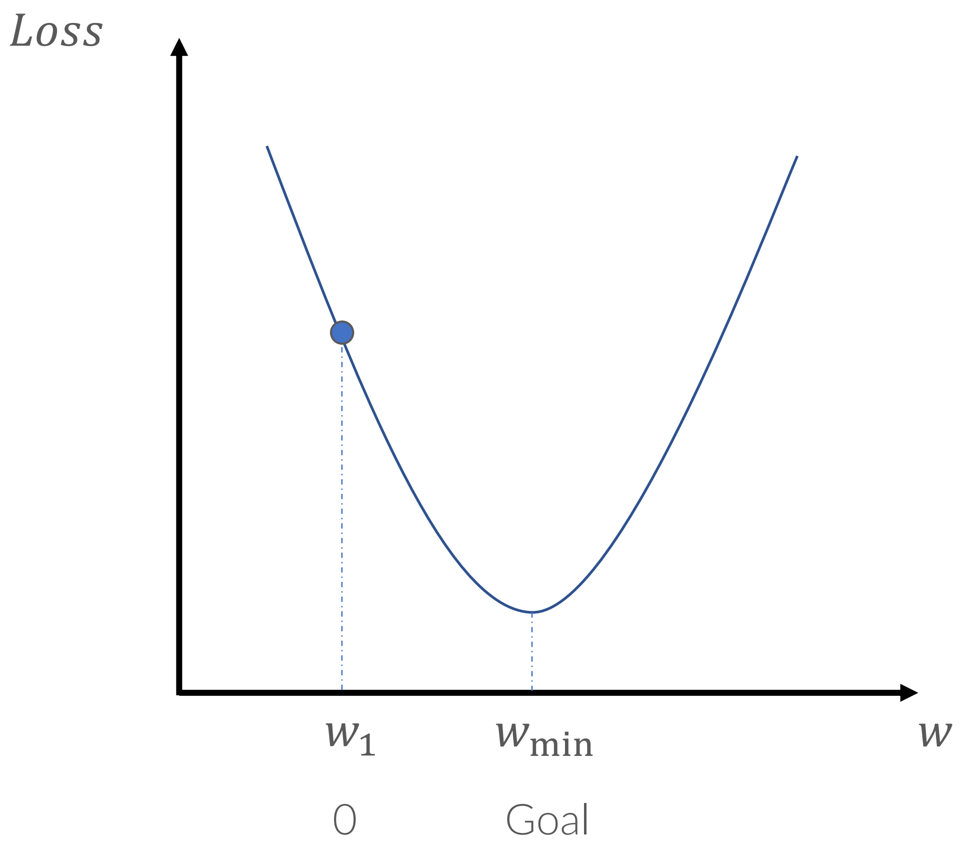
Logistic Regression *Gradient descent – single scalar w*

The algorithm (and the concept of gradient) are designed for *direction vectors*, let's first consider a visualization of the case where the parameter of our system is just a single scalar w

Update rule: $w_{t+1} = w_t - \eta \frac{d}{dw} f(x; w)$

Logistic Regression *Gradient descent – single scalar w*

Start at a random point



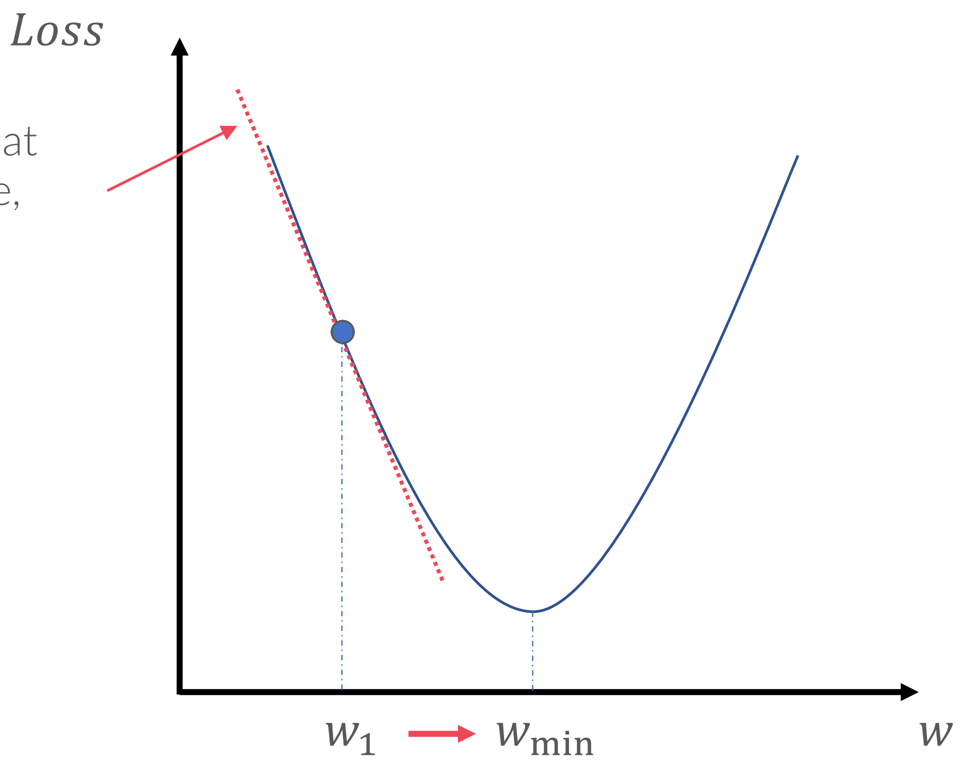
Update rule: $w_{t+1} = w_t - \eta \frac{d}{dw} f(x; w)$

Logistic Regression *Gradient descent – single scalar w*

Start at a random point

Determine a descent direction

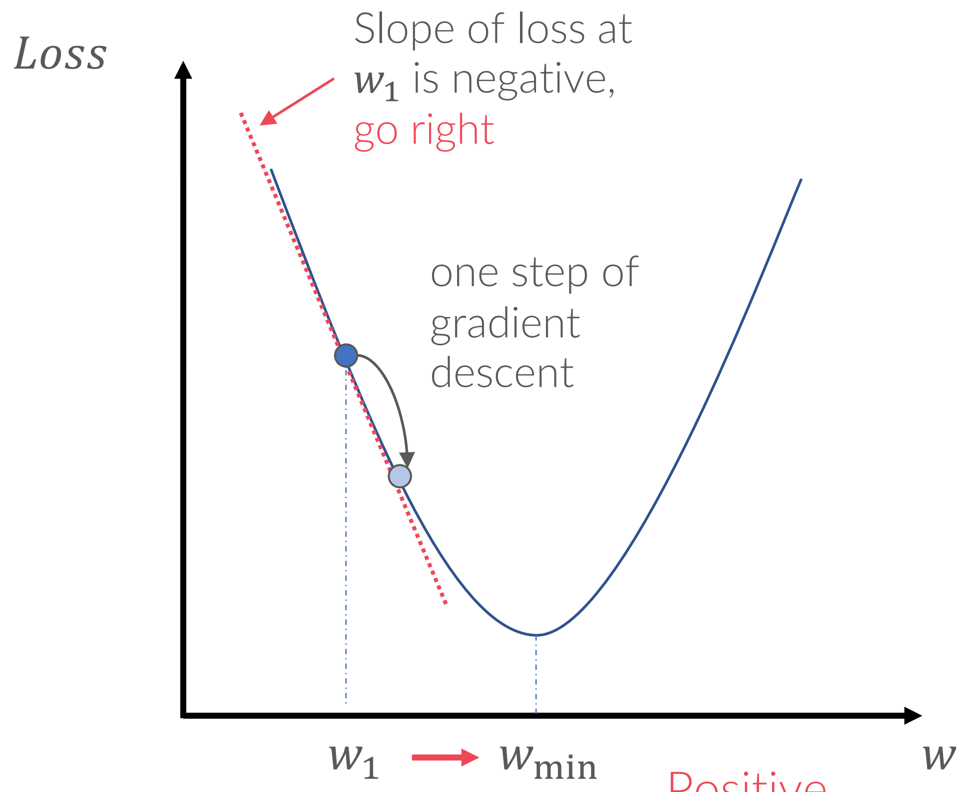
Slope of loss at w_1 is negative, go right



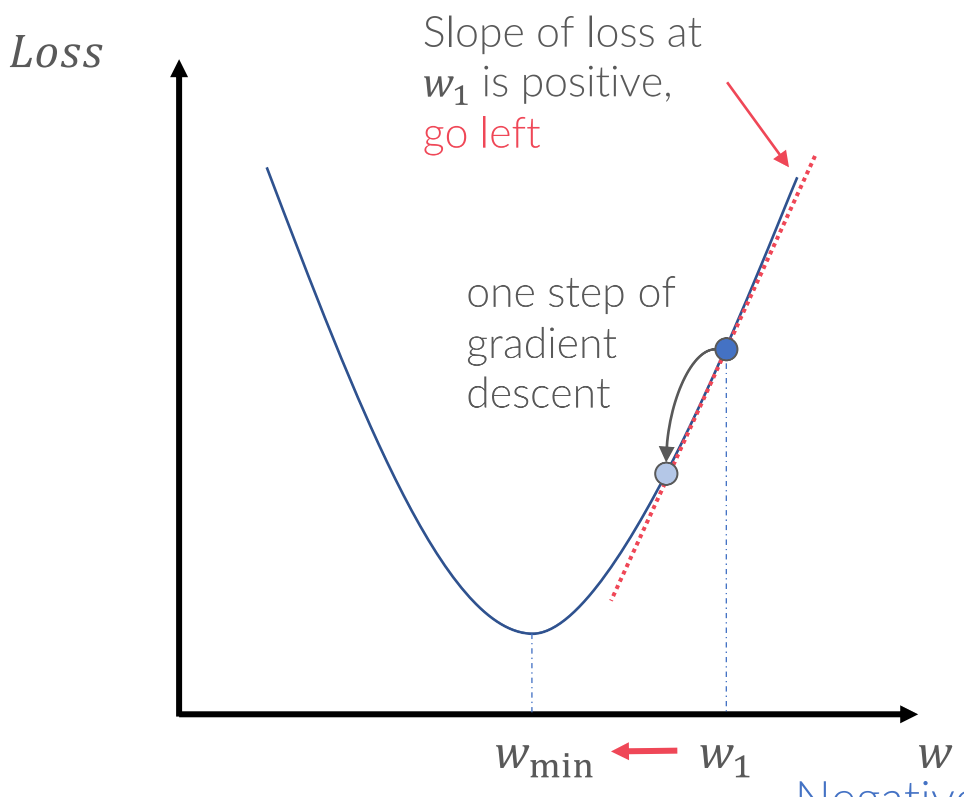
Update rule: $w_{t+1} = w_t \overbrace{- \eta \underbrace{\frac{d}{dw} f(x;w)}_{\text{Negative}}}^{\text{Positive}}$

Logistic Regression *Gradient descent – single scalar w*

Determine a descent direction



$Loss$
Slope of loss at w_1 is negative, go right
one step of gradient descent
$w_1 \rightarrow w_{\min}$
w



$Loss$
Slope of loss at w_1 is positive, go left
one step of gradient descent
$w_{\min} \leftarrow w_1$
w

$$\text{Update rule: } w_{t+1} = \overbrace{w_t - \eta \underbrace{\frac{d}{dw} f(x;w)}_{\text{Negative}}}^{\text{Positive}}$$

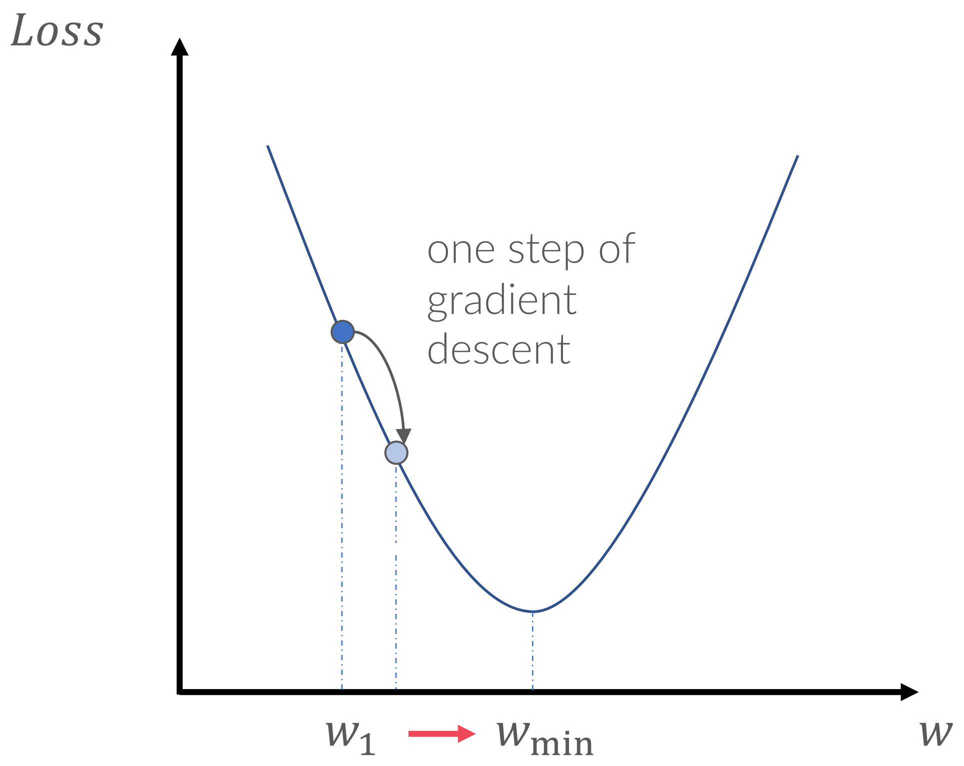
$$\text{Update rule: } w_{t+1} = \overbrace{w_t - \eta \underbrace{\frac{d}{dw} f(x;w)}_{\text{Positive}}}^{\text{Negative}}$$

Logistic Regression *Gradient descent – single scalar w*

Start at a random point

Determine a descent direction
Choose a step size

Update rule: $w_{t+1} = w_t - \eta \frac{d}{dw} f(x; w)$

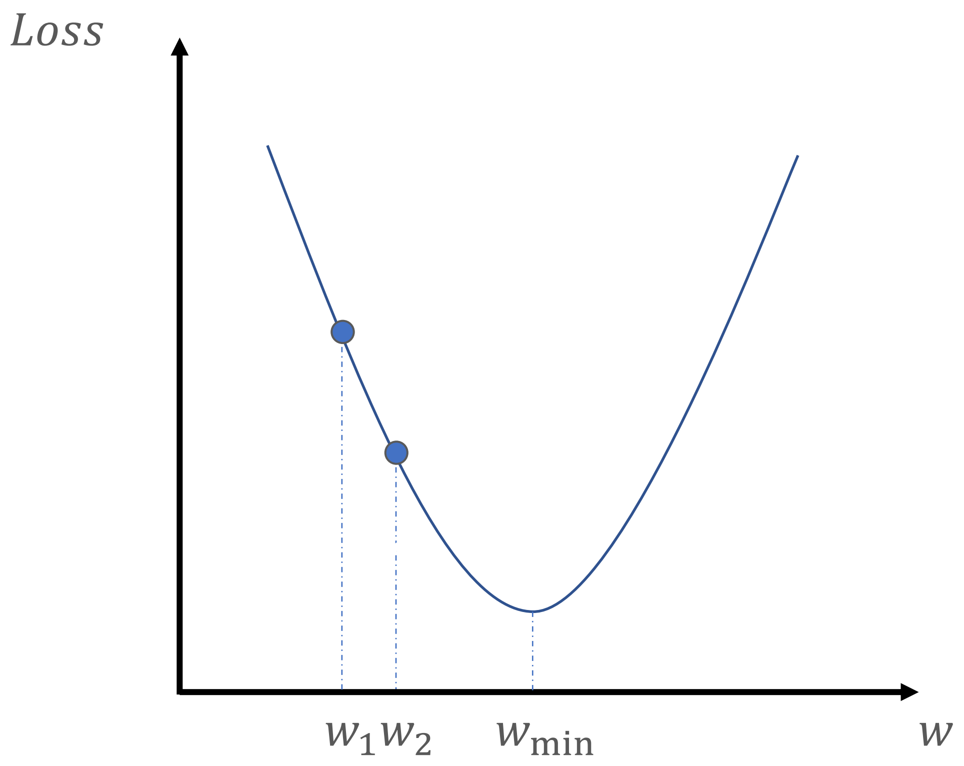


Logistic Regression *Gradient descent – single scalar w*

Start at a random point

- Determine a descent direction
- Choose a step size
- Update

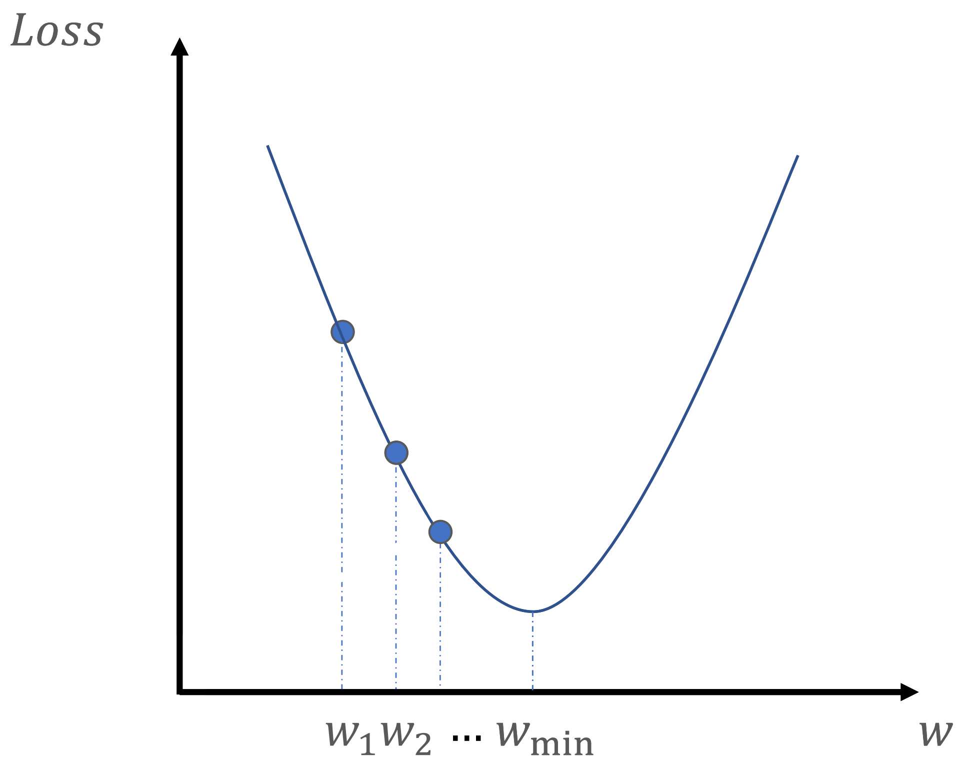
Update rule: $w_{t+1} = w_t - \eta \frac{d}{dw} f(x; w)$



Logistic Regression *Gradient descent – single scalar w*

```
Start at a random point
Repeat
    Determine a descent direction
    Choose a step size
    Update
Until
    Stopping criterion is satisfied
```

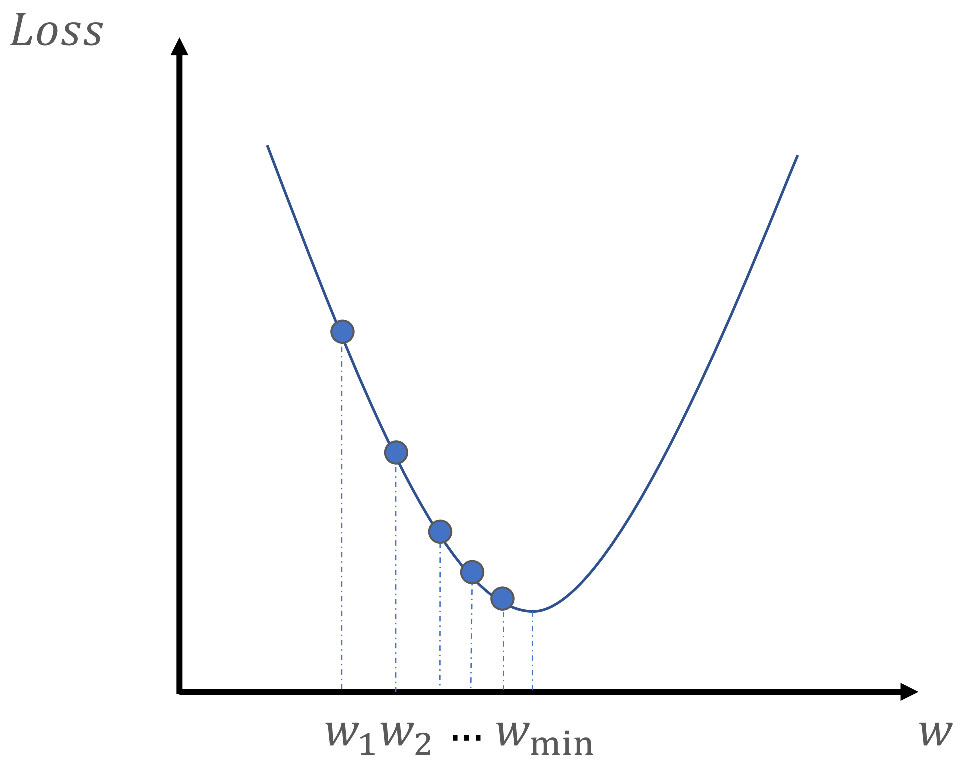
Update rule: $w_{t+1} = w_t - \eta \frac{d}{dw} f(x; w)$



Logistic Regression *Gradient descent – single scalar w*

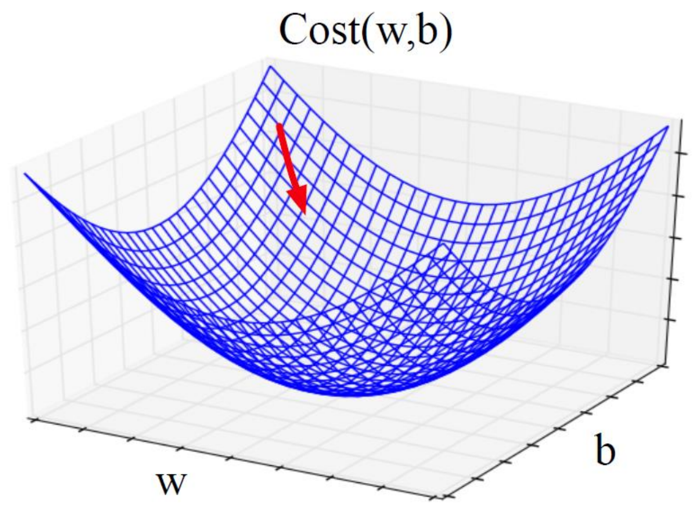
```
Start at a random point
Repeat
    Determine a descent direction
    Choose a step size
    Update
Until
    Stopping criterion is satisfied
```

Update rule: $w_{t+1} = w_t - \eta \frac{d}{dw} f(x; w)$



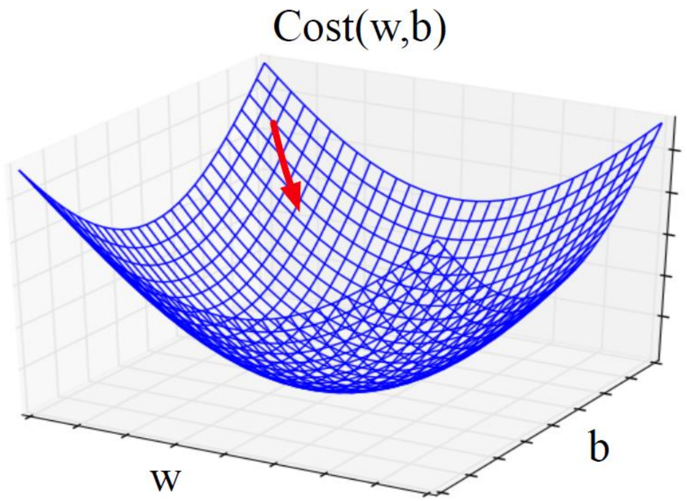
Logistic Regression *Gradient descent – multiple w*

Now let's extend the intuition from a function of one scalar variable w to many variables, because we don't just want to move left or right, we want to know where in the N-dimensional space (of the N parameters that make up θ) we should move. The gradient is just such a *vector*; it expresses the directional components of the sharpest slope along each of those N dimensions



Logistic Regression *Gradient descent – multiple w*

For each dimension/variable w_i in w (plus the bias b), the gradient will have a component that tells us the slope with respect to that variable.



In each dimension w_i , we express the slope as a partial derivative $\frac{\partial}{\partial w_i}$ of the loss function. The gradient is then defined as a vector of these partials. We'll represent w_i as $f(x;\theta)$ to make the dependence on θ more obvious:

$$\nabla_\theta L(f(x;\theta),y)) = \begin{bmatrix} \frac{\partial}{\partial w_1} L(f(x;\theta),y) \\ \frac{\partial}{\partial w_2} L(f(x;\theta),y) \\ \vdots \\ \frac{\partial}{\partial w_n} L(f(x;\theta),y) \end{bmatrix}$$

The final equation for updating q based on the gradient is thus

$$\theta_{t+1} = \theta_t - \eta \nabla L(f(x;\theta)y)$$

∇ denotes as standard derivative

Logistic Regression *Gradient descent for logistic regression*

Recall that the cross-entropy loss function for logistic regression is:

$$L_{CE}(w,b) = -[y\log\sigma(w\cdot x+b)+(1-y)\log(1-\sigma(w\cdot x+b))]$$

It turns out that the derivative of this function for one observation vector $\boldsymbol{x}$ is

$$\frac{\partial L_{CE}(w,b)}{\partial w_j} = [\sigma(w\cdot x+b)-y]x_j$$

Logistic Regression *Deriving the gradient equation*

Some basic calculus:

$$\frac{d}{dx}\ln(x) = \frac{1}{x}$$

The derivative of the sigmoid:

$$\frac{d\sigma(z)}{dz} = \sigma(z)\big(1 - \sigma(z)\big)$$

Chain rule of derivatives, suppose we are computing the derivative of a composite function $f\,(x) = u(v(x))$. The derivative of $f\,(x)$ is the derivative of $u(x)$ with respect to $v(x)$ times the derivative of $v(x)$ with respect to x:

$$\frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dx}$$

First, we want to know the derivative of the loss function with respect to a single weight w_j (we'll need to compute it for each weight, and for the bias):

$$\frac{\partial LL(w,b)}{\partial w_j} = \frac{\partial}{\partial w_j} - \left[y log \sigma(w \cdot x + b) + (1-y)\log\big(1-\sigma(w \cdot x + b)\big)\right]$$

$$= -\left[\frac{\partial}{\partial w_j} y log \sigma(w \cdot x + b) + \frac{\partial}{\partial w_j}(1-y)\log\big(1-\sigma(w \cdot x + b)\big)\right]$$

Next, using the chain rule, and relying on the derivative of log:

$$\frac{\partial LL(w,b)}{\partial w_j} = -\frac{y}{\sigma(w \cdot x + b)}\frac{\partial}{\partial w_j}\sigma(w \cdot x + b) - \frac{(1-y)}{\big(1-\sigma(w \cdot x + b)\big)}\frac{\partial}{\partial w_j}\big(1-\sigma(w \cdot x + b)\big)$$

Rearranging terms:

$$\frac{\partial LL(w,b)}{\partial w_j} = -\left[\frac{y}{\sigma(w \cdot x + b)} - \frac{(1-y)}{\big(1-\sigma(w \cdot x + b)\big)}\right]\frac{\partial}{\partial w_j}\sigma(w \cdot x + b)$$

And now plugging in the derivative of the sigmoid, and using the chain rule one more time

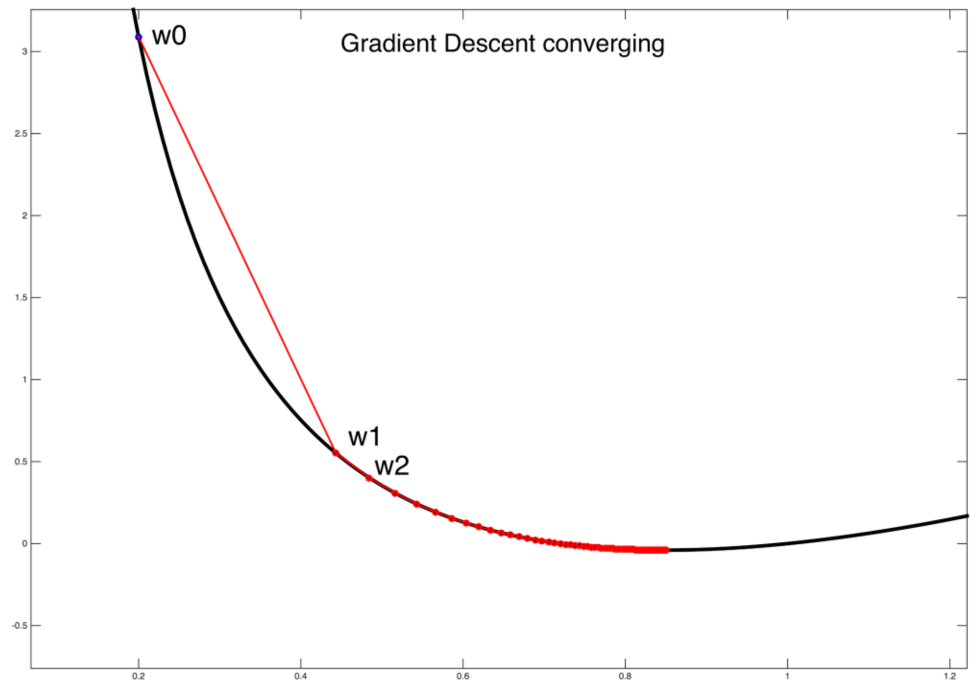
$$\frac{\partial LL(w,b)}{\partial w_j} = -\left[\frac{y-\sigma(w \cdot x + b)}{\sigma(w \cdot x + b)\big(1-\sigma(w \cdot x + b)\big)}\right]\sigma(w \cdot x + b)\big(1-\sigma(w \cdot x + b)\big)\frac{\partial(w \cdot x + b)}{\partial w_j}$$

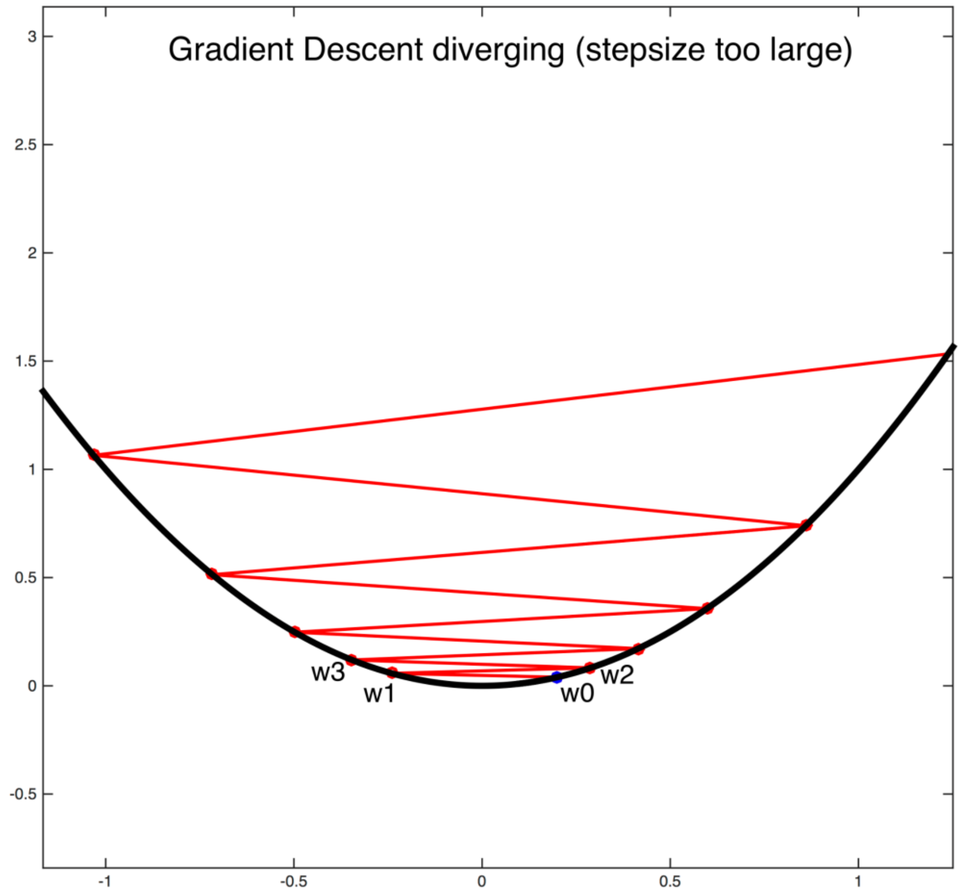
$$= -\left[\frac{y-\sigma(w \cdot x + b)}{\sigma(w \cdot x + b)\big(1-\sigma(w \cdot x + b)\big)}\right]\sigma(w \cdot x + b)\big(1-\sigma(w \cdot x + b)\big)x_j$$

$$= -[y-\sigma(w \cdot x + b)]x_j$$

$$= [\sigma(w \cdot x + b) - y]x_j$$

Logistic Regression *Gradient descent diverging and converging*





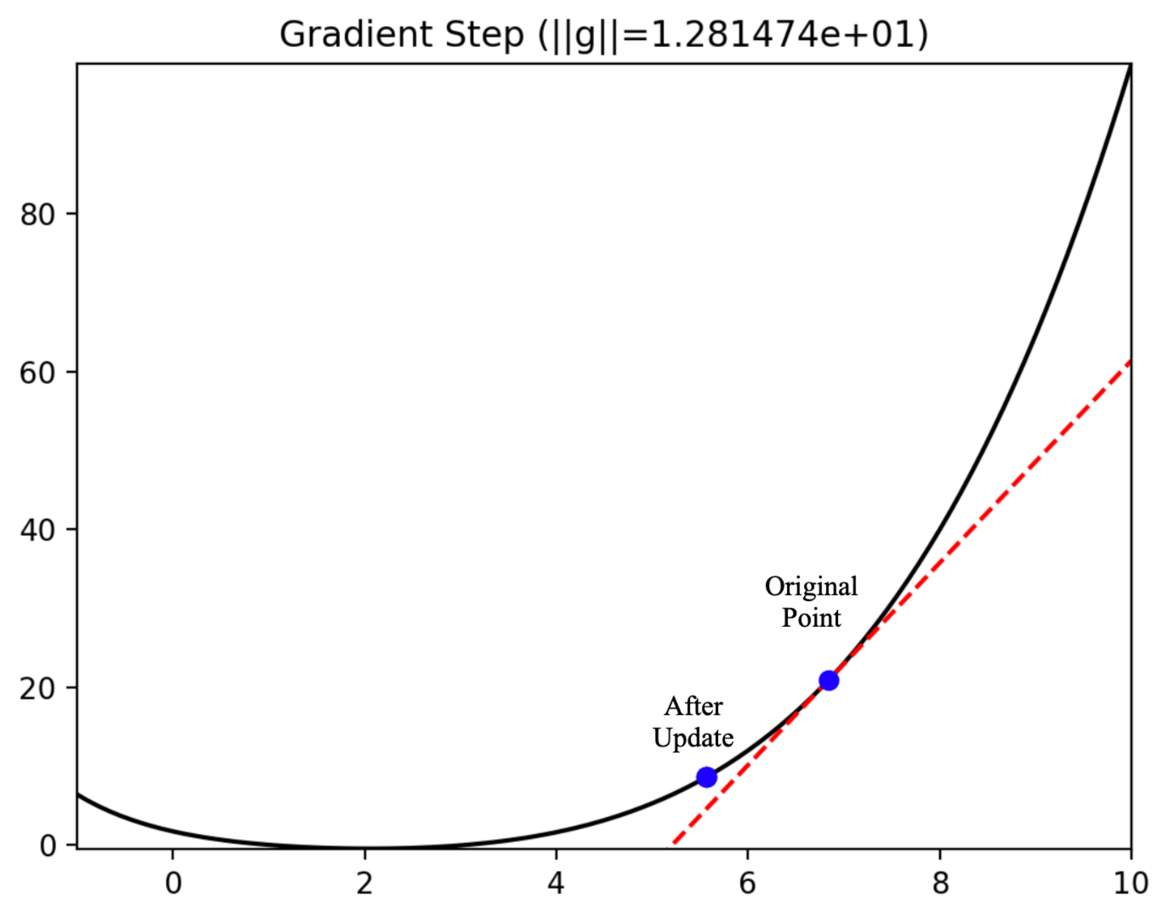
Logistic Regression *Gradient descent for logistic regression*

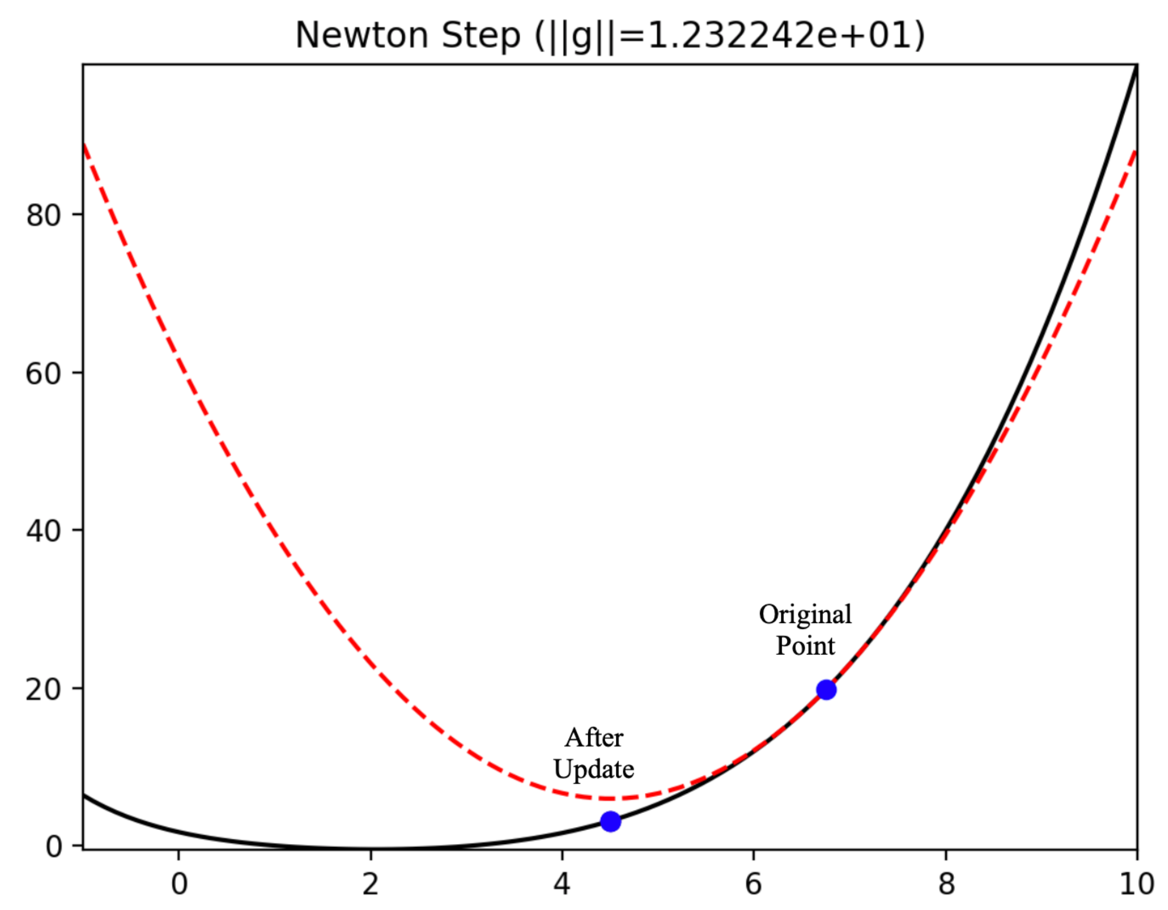
```
function STOCHASTIC GRADIENT DESCENT($L()$, $f()$, $x$, $y$) returns $\theta$
      # where: L is the loss function
      #      f is a function parameterized by $\theta$
      #      x is the set of training inputs $x^{(1)}, x^{(2)}, ..., x^{(n)}$
      #      y is the set of training outputs (labels) $y^{(1)}, y^{(2)}, ..., y^{(n)}$

$\theta \leftarrow 0$
repeat til done   # see caption
   For each training tuple $(x^{(i)}, y^{(i)})$ (in random order)
      1. Optional (for reporting):           # How are we doing on this tuple?
         Compute $\hat{y}^{(i)} = f(x^{(i)};\theta)$         # What is our estimated output $\hat{y}$?
         Compute the loss $L(\hat{y}^{(i)}, y^{(i)})$   # How far off is $\hat{y}^{(i)}$) from the true output $y^{(i)}$?
      2. $g \leftarrow \nabla_\theta L(f(x^{(i)};\theta), y^{(i)})$         # How should we move $\theta$ to maximize loss?
      3. $\theta \leftarrow \theta - \eta\, g$                    # Go the other way instead
return $\theta$
```

The stochastic gradient descent algorithm. Step 1 (computing the loss) is used to report how well we are doing on the current tuple. The algorithm can terminate when it converges, or when progress halts (for example when the loss starts going up on a held-out set).

Logistic Regression *Gradient descent vs. Newton Raphson*





$$\boxed{\boxed{l(\vec{w}+\vec{s}) = l(\vec{w}) + g(\vec{w})\vec{s}} + \frac{1}{2}s^T H(\vec{w})\vec{s}}$$

Logistic Regression *Gradient descent - Example*

One single observation, whose correct class is $y = 1$, and with only two features:

$$x_1 = 3\ (Biomarker_1\ level)$$

$$x_2 = 2\ (Biomarker_2\ level)$$

Assume the initial weights and bias in θ^0 are all set to $\mathbf{0}$, and the initial learning rate η is $\mathbf{0.1}$:

$$w_1 = w_2 = b = 0$$

$$\eta = 0.1$$

Then compute

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L(f(x_i;\theta)y_\text{i})$$

Logistic Regression *Gradient descent - Example*

In our mini example there are three parameters, so the gradient vector has 3 dimensions, for w_1, w_2, and b. We can compute the first gradient as follows:

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{CE}(w,b)}{\partial w_1} \\ \frac{\partial L_{CE}(w,b)}{\partial w_2} \\ \frac{\partial L_{CE}(w,b)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

$$\theta^2 = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} - \eta \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix} = \begin{bmatrix} .15 \\ .1 \\ .05 \end{bmatrix}$$

Logistic Regression *Batch training vs. Mini-batch training*

Batch gradient descent (batch training): compute $L_{CE}(w, b)$ over entire training set M to find the perfect direction

1. Compute the gradient: $\nabla L(f(x;\theta)y)$
2. Update the vector of parameters: $\theta_{t+1} = \theta_t - \eta \nabla L(f(x;\theta)y)$

Stochastic gradient descent (mini- batch): compute $L_{CE}(w, b)$ over single training example $m \in M$

1. Choose (with replacement) a random training example $\mathrm{d} \in D$
2. Compute the gradient: $\nabla L_d(f(x;\theta)y)$
3. Update the vector of parameters: $\theta_{t+1} = \theta_t - \eta \nabla L_d(f(x;\theta)y)$

When D is very large, Stochastic gradient is faster.
If m is the size of the dataset, then we are doing **batch gradient descent**; if m = 1, we are back to doing **stochastic gradient descent**.

Logistic Regression *Mini-batch training of m examples*

We make the assumption that the training examples are independent

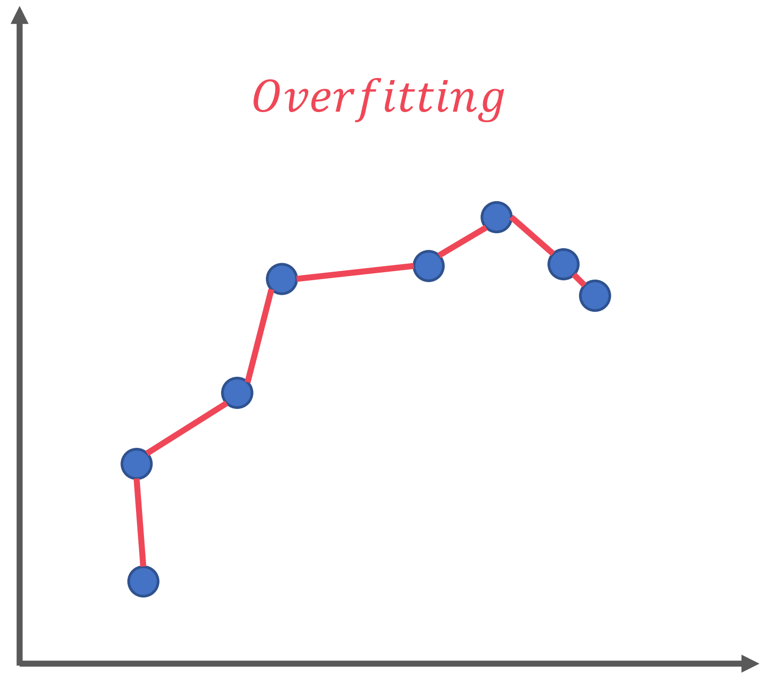
$$log\ p(training\ labels) = log \prod_{i=1}^{m} p(y_i|x_i) = \sum_{i=1}^{m} logp(y_i|x_i) = -\sum_{i=1}^{m} L_{CE}(\hat{y}, y)$$

Now the cost function for the mini-batch of m examples is the average loss for each example:

$$cost(w,b) = \frac{1}{m}\sum_{i=1}^{m} L_{CE}(\hat{y}, y)$$

$$= \frac{1}{m}\sum_{i=1}^{m} -[y_i \log \sigma(w \cdot x_i + b) + (1 - y_i) \log\big(1 - \sigma(w \cdot x_i + b)\big)]$$

Logistic Regression *Regularization*



A good model should be able to **generalize** well from the training data to the unseen test set, but a model that overfits will have poor generalization.

To avoid overfitting, a new regularization term $R(\theta)$ is added to the objective function $\hat{\theta} = \arg\min_{\theta} \frac{1}{m} \sum_{i=1}^{m} L_{CE}(y_i, x_i; \theta)$

Slightly rewritten to be maximizing log probability rather than minimizing loss, and removing the $\frac{1}{m}$ term

$$\hat{\theta} = \arg\max_{\theta} \sum_{i=1}^{m} logP(y_i|x_i) - \alpha R(\theta)$$

The new regularization term R(q) is used to penalize large weights.

Mathematics *Distance*

From the general $L_p - norm$ we can define the corresponding $L_p - distance$ function, given as follows

$$\sigma_p(a, b) = ||a - b||_{\mathrm{p}}$$

Mathematics *Distance*

1. Euclidean Distance*, L_2
2. Manhattan Distance, L_1
3. Hamming Distance, L_0
4. Minkowski Distance, L_p
5. Cosine Distance(similarity)

Euclidean Distance* is the most popular method

Mathematics *Euclidean distance*

Euclidean Distance

$$L_2 - norm: ||x_i - y_i||_2 = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Mathematics ***Manhattan distance***

Manhattan Distance

$$L_1 - norm: \left|\left|x_i - y_i\right|\right|_1 = \sum_{i=1}^{n} |x_i - y_i|$$

Logistic Regression *L2 Regularization*

$L2\ regularization$ is a quadratic function of Regularization the weight values, named because it uses the (square of the) L2 norm of the weight values. The L2 norm, $||\theta||_2$, is the same as the Euclidean distance of the vector θ from the origin. If θ consists of n weights, then:

$$R(\theta) = ||\theta||_2^2 = \sum_{j=1}^{n} \theta_j^2$$

Plug in

$$\hat{\theta} = \arg\max_{\theta} \sum_{i=1}^{m} logP(y_i|x_i) - \alpha \sum_{j=1}^{n} \theta_j^2$$

Logistic Regression *L1 Regularization*

$L1\ regularization$ is a linear function of the weight values, named after the L1 norm Regularization $||\theta||_1$, the sum of the absolute values of the weights, or Manhattan distance (the Manhattan distance is the distance you'd have to walk between two points in a city with a street grid like New York):

$$R(\theta) = ||\theta||_1 = \sum_{i=1}^{n} |\theta_i|$$

Plug in

$$\hat{\theta} = \arg\max_{\theta} \sum_{i=1}^{m} logP(y_i|x_i) - \alpha \sum_{i=1}^{n} |\theta_i|$$

Logistic Regression *Multinomial logistic regression*

For classes more than two, we use multinomial logistic regression, also called SoftMax regression (or, historically, the maxent classifier) using a generalization of the sigmoid. We want to know the probability of y being in each potential class $c \in C, p(y = c|x)$. The SoftMax function takes a vector $z = [z_1, \ldots, z_k]$ of k arbitrary values and maps them to a probability distribution, with each value in the range (0,1), and all the values summing to 1. Like the sigmoid, it is an exponential function.

$$softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}, 1 \leq i \leq k$$

Logistic Regression *SoftMax is an extension of the sigmoid*

Let’s say, we have two classes:

$$Softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} = \frac{e^{z_1}}{e^{z_1} + e^{z_2}} = \frac{e^{z_1 - z_2}}{e^{z_1 - z_2} + 1}$$

Sigmoid

Logistic Regression *Multinomial logistic regression*

The SoftMax of an input vector $z = [z_1, \dots, z_k]$ is thus a vector itself:

$$softmax(z) = \left[\frac{e^{z_1}}{\sum_{i=1}^{k} e^{z_i}}, \dots, \frac{e^{z_k}}{\sum_{i=1}^{k} e^{z_i}}\right]$$

The denominator $\sum_{i=1}^{k} e^{z_i}$ is used to normalize all the values into probabilities. Thus for example given a vector:

$$z = [0.6, 1.1, -1.5, 1.2, 3.2, -1.1]$$

The result $\boldsymbol{softmax(z)}$ is

$$[0.055, 0.090, 0.0067, 0.10, 0.74, 0.010]$$

Logistic Regression *Multinomial logistic regression*

Like the sigmoid, the SoftMax has the property of squashing values toward 0 or 1. Thus if one of the inputs is larger than the others, it will tend to push its probability toward 1, and suppress the probabilities of the smaller inputs

$$p(y = c|x) = \frac{e^{w_c \cdot x + b_c}}{\sum_{j=1}^{k} e^{w_j \cdot x + b_j}}$$

Logistic Regression *Learning in SoftMax*

Like the sigmoid, the SoftMax has the property of squashing values toward 0 or 1. Thus if one of the inputs is larger than the others, it will tend to push its probability toward 1, and suppress the probabilities of the smaller inputs

$$L_{CE}(\hat{y}, y) = -\sum_{k=1}^{K} 1\{y = k\} log p(y = k|x)$$

$$= -\sum_{k=1}^{K} 1\{y = k\} \log \frac{e^{w_c \cdot x + b_c}}{\sum_{j=1}^{k} e^{w_j \cdot x + b_j}}$$

This makes use of the function $1\{\,\}$ which evaluates to 1 if the condition in the brackets is true and to 0 otherwise.

Gradient

$$\frac{\partial L_{CE}}{\partial w_j} = -\big(1\{y = k\} - p(y = k|x)\big) x_k$$

$$= -\left(1\{y = k\} - \frac{e^{w_c \cdot x + b_c}}{\sum_{j=1}^{k} e^{w_j \cdot x + b_j}}\right) x_k$$

2. Implementation in R

```r
# South African heart disease ----
heart <-
  read.table(
    "http://www-
stat.stanford.edu/~tibs/ElemStatLearn/datasets/SAheart.data",
    sep = ",",
    head = T,
    row.names = 1
  )
# low-density lipoprotein - ldl; Coronary Heart Disease - chd ----
heart0 <- heart %>% select(ldl,age, chd)
```

```r
# Set the seed to be reproductive
set.seed(06032020)
ran <- sample(1:nrow(heart0), 0.8 * nrow(heart0))
```

```r
train_n <-
  as.data.frame(lapply(heart0[1:2], scale, scale =
T, center = T)) %>%
  cbind(heart0[3]) %>%
  slice(., ran)%>%
  mutate(chd=as.factor(chd))
test_n <-
  as.data.frame(lapply(heart0[1:2], scale, scale =
T, center = T)) %>%
  cbind(heart0[3]) %>%
  slice(., -ran) %>%
  mutate(chd=as.factor(chd))
```

```r
# Compared to the glm function
fit <- glm(chd ~ ldl + age, family = binomial,
data = train_n)
```

```r
pred <- predict(fit, test_n, type = "response")

test <- test_n %>%
mutate(pred_label=as.factor(ifelse(pred>0.5, 1,
0)))

confusionMatrix(test$pred_label, test$chd)
```

```
Reference
Prediction  0  1
         0 52 20
         1  8 13
```

Acknowledgements and Resources

Acknowledgements and Resources

All the lectures materials borrowed from the following resources with/without modifications:

Textbook:

Galit Shmueli, Peter C. Bruce, Inbal Yahav, Nitin R. Patel, Kenneth C. Lichtendahl Jr., Data Mining for Business Analytics: Concepts, Techniques, and Applications in R (DMBA), Wiley, 1st Edition, ISBN-10: 1118879368, ISBN-13: 978-1118879368.

Additional Textbooks:

R For Data Science (open license, R4DS), Wickham, Hadley, and Garrett Grolemund

R Markdown (open license, RMD), Xie, Yihui, et al.

James, Gareth, et al. An Introduction to Statistical Learning: with Applications in R. Springer, 2017. (open license, ISL)

Mohammed J. Zaki, Wagner Meira, Jr., Data Mining and Analysis: Fundamental Concepts and Algorithms (DMA), Cambridge University Press, May 2014 (NEU library link)

David Hand, Heikki Mannila, Padhraic Smyth. Principles of Data Mining (PDM), The MIT Press, 2001, ISBN-10: 026208290X, ISBN-13: 978-0262082907. (NEU library link)

Tan, Pang-Ning, et al. Introduction to Data Mining (DM). Pearson Education, 2006. (official link)

Hastie, T., Friedman, J., & Tisbshirani, R. (2017). The Elements of statistical learning: data mining, inference, and prediction (ESL). New York: Springer. (open license)