Machine Learning

Lecture 03: Logistic Regression and Gradient Descent

Nevin L. Zhang
lzhang@cse.ust.hk

Department of Computer Science and Engineering
The Hong Kong University of Science and Technology

This set of notes is based on internet resources and
KP Murphy (2012). Machine learning: a probabilistic perspective. MIT Press. (Chapter 8)
Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT press. www.deeplearningbook.org. (Chapter 4)
Andrew Ng. Lecture Notes on Machine Learning. Stanford.
Hal Daume. A Course on Machine Learning. http://ciml.info/

Outline

Recap of Probabilistic Linear Regression

- Training set $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^N$, where $y_i \in \mathbb{R}$.
- Probabilistic model:

$$p(y|\mathbf{x}, \theta) = \mathcal{N}(y|\mu(\mathbf{x}), \sigma^2) = \mathcal{N}(y|\mathbf{w}^\top \mathbf{x}, \sigma^2)$$

- Learning: Determining $\mathbf{w}$ by minimizing the cross entropy loss:

$$J(\mathbf{w}) = -\frac{1}{N}\sum_{i=1}^{N} \log p(y_i|\mathbf{x}_i, \mathbf{w})$$

- Point estimation of y:

$$\hat{y} = \mu(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$$

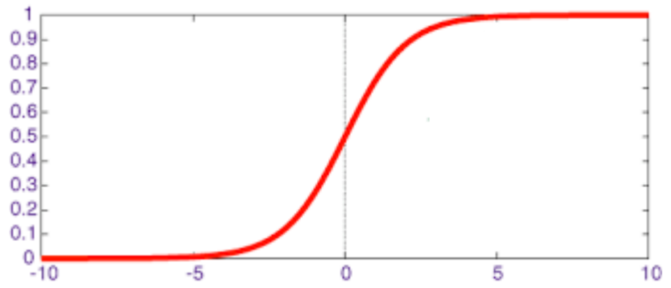
Logistic Regression (for Classification)

- Training set $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^N$, where $y_i \in \{0, 1\}$.
- Probabilistic model:

$$p(y|\mathbf{x}, \mathbf{w}) = Ber(y|\sigma(\mathbf{w}^\top \mathbf{x}))$$

  - $\sigma(z)$ is the **sigmoid/logistic/logit function**.

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



  - It maps the real line $\mathbb{R}$ to $[0, 1]$. *Not to be confused with the variance σ^2 in the Gaussian distribution*

Logistic Regression

- The model:

$$p(y|\mathbf{x}, \mathbf{w}) = Ber(y|\sigma(\mathbf{w}^\top \mathbf{x}))$$

$$\begin{aligned} p(y=1|\mathbf{x}, \mathbf{w}) &= \sigma(\mathbf{w}^\top \mathbf{x}) = \frac{1}{1+\exp(-\mathbf{w}^\top \mathbf{x})} \\ p(y=0|\mathbf{x}, \mathbf{w}) &= 1-\sigma(\mathbf{w}^\top \mathbf{x}) = \frac{\exp(-\mathbf{w}^\top \mathbf{x})}{1+\exp(-\mathbf{w}^\top \mathbf{x})} \end{aligned}$$

- Consider the **logit** of $p(y=1|\mathbf{x}, \mathbf{w})$

$$\begin{aligned} \log \frac{p(y=1|\mathbf{x}, \mathbf{w})}{1-p(y=1|\mathbf{x}, \mathbf{w})} &= \log \frac{p(y=1|\mathbf{x}, \mathbf{w})}{p(y=0|\mathbf{x}, \mathbf{w})} \\ &= \log \exp(\mathbf{w}^\top \mathbf{x}) = \mathbf{w}^\top \mathbf{x}. \end{aligned}$$

So, a linear model for the logit. Hence called **logistic regression**.

Logistic Regression: Decision Rule

- To classify instances, we obtain a point estimation of y: $\hat{y} = \arg\max_y p(y|\mathbf{x}, \mathbf{w})$
- In other words, the **decision/classification rule** is:

$$\hat{y} = 1 \text{ iff } p(y = 1|\mathbf{x}, \mathbf{w}) > 0.5$$

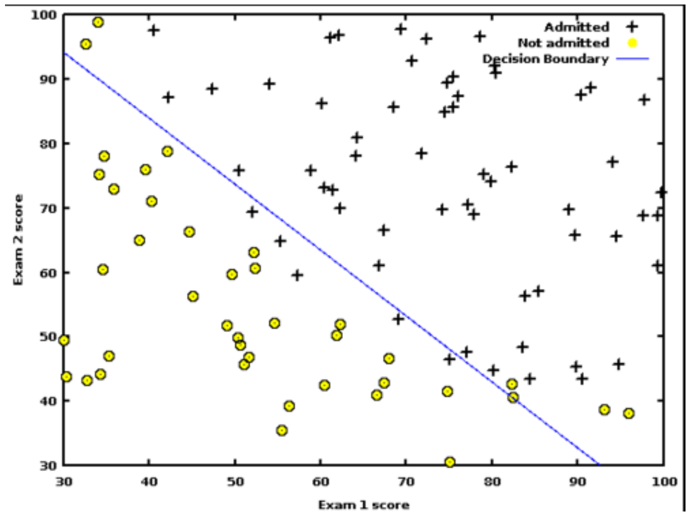
- This is called the **optimal Bayes classifier**:
  - Suppose the same situation is to occur many times. You will always make mistakes no matter what decision rule to use. The probability of mistakes is minimized if you use the above rule.

Logistic Regression is a Linear Classifier

- In fact, the decision/classification rule in logistic regression is equivalent to:

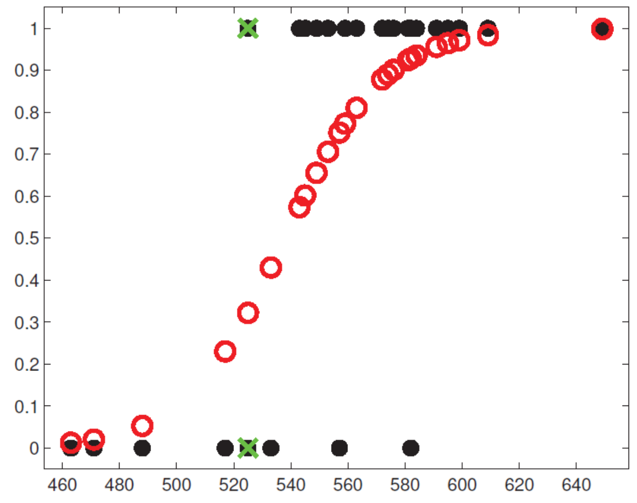
  $$\hat{y} = 1 \text{ iff } \mathbf{w}^\top \mathbf{x} > 0$$

  So,it is a **linear classifier** with a **decision boundary**.
- Example: Whether a students is admitted based on the results of two exams:

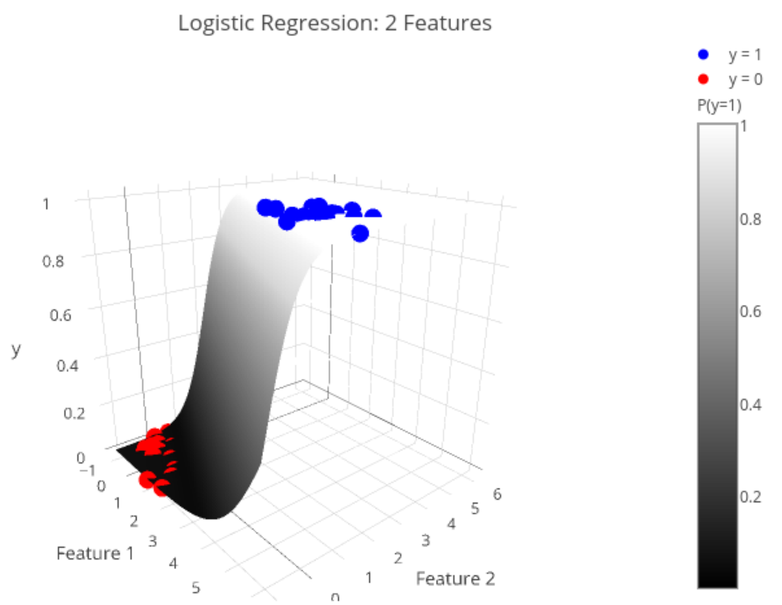


Logistic Regression: Example

- Solid black dots are the data. Those at the bottom are the SAT scores of applicants rejected by a university, and those at the top are the SAT scores of applicants accepted by a university.
- The red circles are the predicted probabilities that the applicants would be accepted.



Logistic Regression: 2D Example



The **decision boundary** $p(y = 1|\mathbf{x}, \mathbf{w}) = 0.5$ is a straight line in the feature space.

Parameter Learning

- We would like to find the MLE of $\mathbf{w}$, i.e, the values of $\mathbf{w}$ that minimizes the cross entropy loss: $J(\mathbf{w}) = -\frac{1}{N}\sum_{i=1}^{N} \log P(y_i|\mathbf{x}_i, \mathbf{w})$
- Consider a general training example $(\mathbf{x}, y)$. Because y is binary, we have

$$\begin{aligned} \log P(y|\mathbf{x}, \mathbf{w}) &= y \log \hat{y} + (1-y)\log(1-\hat{y}) \qquad (\hat{y} = P(y=1|\mathbf{x}, \mathbf{w})) \\ &= y \log \sigma(\mathbf{w}^\top \mathbf{x}) + (1-y)\log(1-\sigma(\mathbf{w}^\top \mathbf{x})). \end{aligned}$$

Hence,

$$J(\mathbf{w}) = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log \sigma(\mathbf{w}^\top \mathbf{x}_i) + (1-y_i)\log(1-\sigma(\mathbf{w}^\top \mathbf{x}_i))]$$

- Unlike linear regression, we can no longer write down the MLE in closed form. Instead, we need to use optimization algorithms to compute it.
  - Gradient descent
  - Newton's method

Outline

Gradient Descent

- Consider a function $y = J(w)$ of a scalar variable w. The derivative of $J(w)$ is defined as follows:

$$J'(w) = \frac{df(w)}{dw} = \lim_{\epsilon \to 0} \frac{J(w+\epsilon) - J(w)}{\epsilon}$$

- When ϵ is small, we have

$$J(w+\epsilon) \approx J(w) + \epsilon J'(w)$$

- This equation tells use how to reduce $J(w)$ by changing w in small steps:
  - If $J'(w) > 0$, make ϵ negative, i.e., decrease w;
  - If $J'(w) < 0$, make ϵ positive, i.e., increase w.

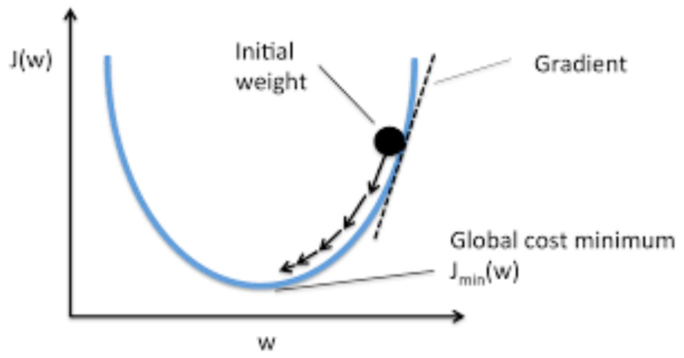
  In other words, **move in the opposite direction of the derivative (gradient)**

Gradient Descent

- To implement the idea, we update w as follows:

$$w \leftarrow w - \alpha J'(w)$$

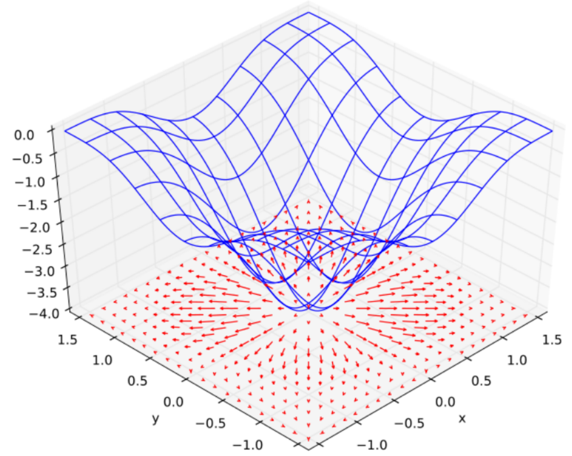
  The term $-J'(w)$ means that we move in the opposite direction of the derivative, and α determines how much we move in that direction. It is called the **step size** in optimization and the **learning rate** in machine learning



Gradient Descent

- Consider a function $y = J(\mathbf{w})$, where $\mathbf{w} = (w_0, w_1, \ldots, w_D)^\top$.
- The **gradient** of J at $\mathbf{w}$ is defined as

$$\nabla J = (\frac{dJ}{dw_0}, \frac{dJ}{dw_1}, \ldots, \frac{dJ}{dw_D})^\top$$



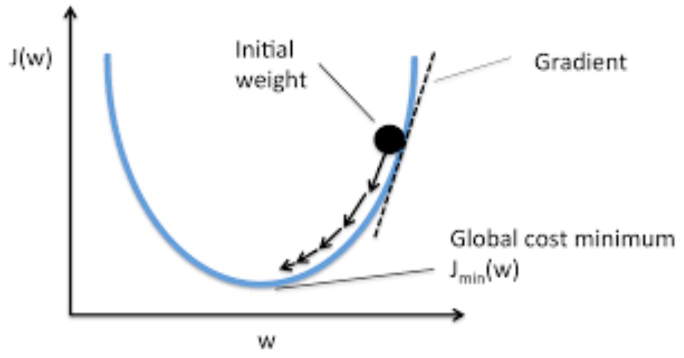
- The gradient is the direction along which J increases the fastest. If we want to reduce J as fast as possible, move in the opposite direction of the gradient.

Gradient Descent

- The method of **steepest descent/gradient descent** for minimizing $J(\mathbf{w})$
  1. Initialize $\mathbf{w}$
  2. Repeat until convergence

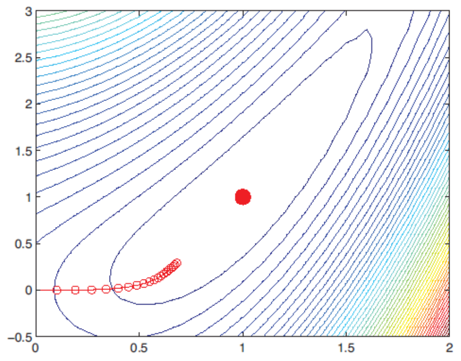
$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla J(\mathbf{w})$$

- The learning rate α usually changes from iteration to iteration.

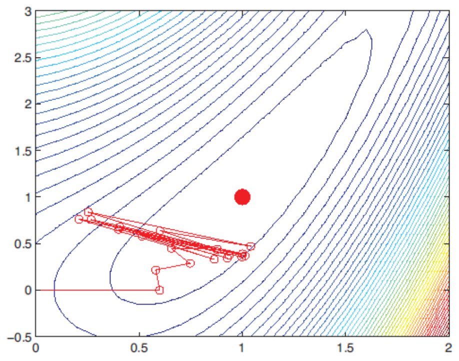


Choice of Learning Rate

- Constant learning rate is difficult to set:
  - Too small, convergence will be very slow
  - Too large, the method can fail to converge at all.



$\alpha = 0.1$

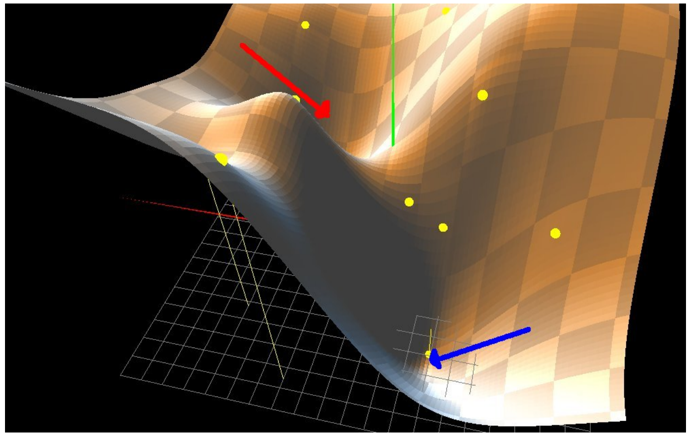


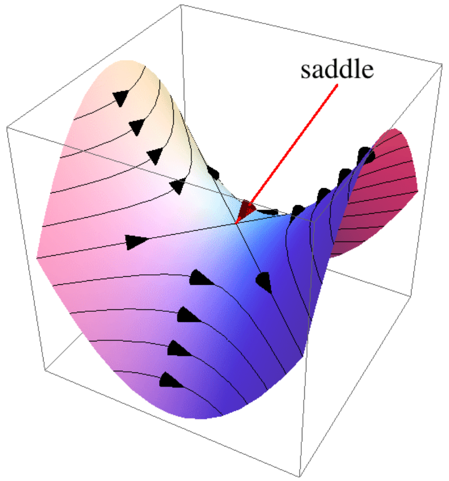
$\alpha = 0.6$

- Better to vary the learning rate. Will discuss this more later.

Gradient Descent

- Gradient descent can get stuck at local minima or saddle points





- Nonetheless, it usually works well.

Outline

Derivative of $\sigma(z)$

To apply gradient descent to logistic regression, we need to compute the partial derivative of $J(\mathbf{w})$ w.r.t each weight w_j. Before doing that, first consider the derivative of the sigma function:

$$\begin{aligned}
\sigma'(z) &= \frac{d\sigma(z)}{dz} = \frac{d}{dz}\frac{1}{1+e^{-z}} \\
&= -\frac{1}{(1+e^{-z})^2}\frac{d(1+e^{-z})}{dz} \\
&= \frac{1}{(1+e^{-z})^2}e^{-z} \\
&= \frac{1}{1+e^{-z}}(1-\frac{1}{1+e^{-z}}) \\
&= \sigma(z)(1-\sigma(z))
\end{aligned}$$

Derivative of $\log P(w|\mathbf{x}, \mathbf{w})$

- $z = \mathbf{w}^\top \mathbf{x}$, $\mathbf{x} = [x_0, x_1, \ldots, x_D]^\top$, $\mathbf{w} = [w_0, w_1, \ldots, w_D]^\top$.

$$\frac{\partial \sigma(z)}{\partial w_j} = \frac{d\sigma(z)}{dz} \frac{\partial z}{\partial w_j} = \sigma(z)(1 - \sigma(z))x_j$$

$$\begin{aligned}
\frac{\partial}{\partial w_j} \log P(y|\mathbf{x}, \mathbf{w}) &= \frac{\partial}{\partial w_j}[y \log \sigma(z) + (1-y)\log(1-\sigma(z))] \\
&= y \frac{1}{\sigma(z)} \frac{\partial \sigma(z)}{\partial w_j} - (1-y)\frac{1}{1-\sigma(z)} \frac{\partial \sigma(z)}{\partial w_j} \\
&= [y(1-\sigma(z)) - (1-y)\sigma(z)]x_j \\
&= [y - \sigma(z)]x_j.
\end{aligned}$$

Derivative of the Cross Entropy Loss

- The *i*-th training example: $\mathbf{x}_i = [x_{i,0}, x_{i,1}, \ldots, x_{i,D}]^\top$

$$\begin{aligned}\frac{\partial J(\mathbf{w})}{\partial w_j} &= -\frac{1}{N}\sum_{i=1}^{N}\frac{\partial}{\partial w_j}\log P(y_i|\mathbf{x}_i, \mathbf{w}) \\ &= -\frac{1}{N}\sum_{i=1}^{N}[y_i - \sigma(z_i)]x_{i,j}\end{aligned}$$

Batch Gradient Descent

- The **Batch Gradient Descent** algorithm for logistic regression

  Repeat until convergence

$$w_j \leftarrow w_j + \alpha \frac{1}{N} \sum_{i=1}^{N} [y_i - \sigma(\mathbf{w}^\top \mathbf{x}_i)] x_{i,j}$$

- Interpretation: (Assume $\mathbf{x}_i$ are positive vectors)
  - If predicted value $\sigma(\mathbf{w}^\top \mathbf{x}_i)$ is smaller than the actual value y_i, there is reason to increase w_j. The increment is proportional to $x_{i,j}$.
  - If predicted value $\sigma(\mathbf{w}^\top \mathbf{x}_i)$ is larger than the actual value y_i, there is reason to decrease w_j. The decrement is proportional to $x_{i,j}$.

Stochastic Gradient Descent

- Batch gradient descent is costly when N is large.
- In such case, we usually use **Stochastic Gradient Descent**:

```
Repeat until convergence
    Randomly choose $B \subset \{1, 2, \ldots, N\}$
    $w_j \leftarrow w_j + \alpha \frac{1}{|B|} \sum_{i \in B} [y_i - \sigma(\mathbf{w}^\top \mathbf{x}_i)] x_{i,j}$
```

- The randomly picked subset B is called a **minibatch**. Its size is called the **batch size**.

L_2 Regularization

- Just as we prefer ridge regression to linear regression, we prefer to add a regularization term in the objective function of logistic regression:

$$J(w) \leftarrow J(\mathbf{w}) + \frac{1}{2}\lambda ||\mathbf{w}||_2^2.$$

- In this case, we have

$$\frac{\partial J(\mathbf{w})}{\partial w_j} = -\frac{1}{N}\sum_{i=1}^{N}[y_i - \sigma(z_i)]x_{i,j} + \lambda w_j$$

- The weight update formula is:

$$w_j \leftarrow w_j + \alpha[-\lambda w_j + \frac{1}{|B|}\sum_{i \in B}(y_i - \sigma(\mathbf{w}^\top \mathbf{x}_i))x_{i,j}]$$

L_2 Regularization

- Update rule without regularization:

$$w_j \leftarrow w_j + \alpha \frac{1}{|B|} \sum_{i \in B} [y_i - \sigma(\mathbf{w}^\top \mathbf{x}_i)] x_{i,j}$$

- Update rule with regularization:

$$w_j \leftarrow (1 - \alpha\lambda) w_j + \alpha \frac{1}{|B|} \sum_{i \in B} (y_i - \sigma(\mathbf{w}^\top \mathbf{x}_i)) x_{i,j}$$

- Regularization forces the weights to be smaller:

$$|(1 - \alpha\lambda) w_j| < |w_j|$$

as $1 > \alpha\lambda > 0$.

Outline

Newton's Method

- Consider again a function $J(w)$ of a scalar variable w. We want to minimize $J(w)$. At a minimum point, the derivative $J'(w) = 0$.
- Let $f(w) = J'(w)$. We want to find points where $f(w) = 0$.
- **Newton's method** (aka **the Newton-Raphson method**) is a commonly used method to solve the problem:
  - Start some initial value w
  - Repeat the following update until convergence

$$w \leftarrow w - \frac{f(w)}{f'(w)}$$

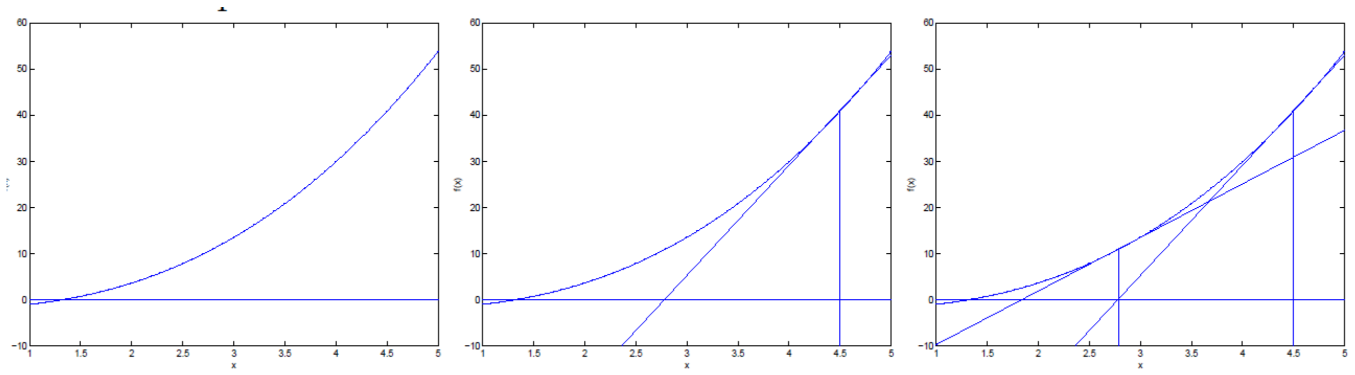
Newton's Method

Interpretation of Newton's method

- Approximate the function f via a linear function that is tangent to f at the current guess w_i: $y = f'(w_i)w + f(w_i) - f'(w_i)w_i$
- Solve for where that linear function equals to zero to get next w.

$$f'(w_i)w + f(w_i) - f'(w_i)w_i = 0$$

$$w_{i+1} = w_i - \frac{f(w_i)}{f'(w_i)}$$



Newton's Method

Minimize $J(w)$ using Newton's method:

- Start some initial value w
- Repeat the following update until convergence

$$w \leftarrow w - \frac{J'(w)}{J''(w)}$$

where f'' is the second derivative of f.

Newton's Method

New consider minimizing a $J(\mathbf{w})$ of a vector $\mathbf{w} = (w_0, w_1, \ldots, w_D)^\top$.

- The "first derivative" of $J(\mathbf{w})$ is the gradient vector $\nabla J(\mathbf{w})$.
- The "second derivative" of $J(\mathbf{w})$ is the **Hessian matrix** $H = [H_{ij}]$

$$H_{ij} = \frac{\partial^2 J(\mathbf{w})}{\partial w_i \partial w_j}$$

- Minimize $J(\mathbf{w})$ using Newton's method
  - Start some initial value $\mathbf{w}$
  - Repeat the following update until convergence

$$\mathbf{w} \leftarrow \mathbf{w} - H^{-1} \nabla J(\mathbf{w})$$

Newton's Method: Another Perspective

- By Taylor expansion, we have

$$J(w+\epsilon) \approx J(w) + J'(w)\epsilon + \frac{1}{2}J''(w)\epsilon^2$$

- We want $\frac{dJ(w+\epsilon)}{d\epsilon} = 0$. So,

$$J'(w) + J''(w)\epsilon = 0$$

- Solving the equation, we get

$$\epsilon = -\frac{J'(w)}{J''(w)}$$

Newton's Method vs Gradient Descent

- In **gradient descent**, we have only a first order term in the approximation (gradient).

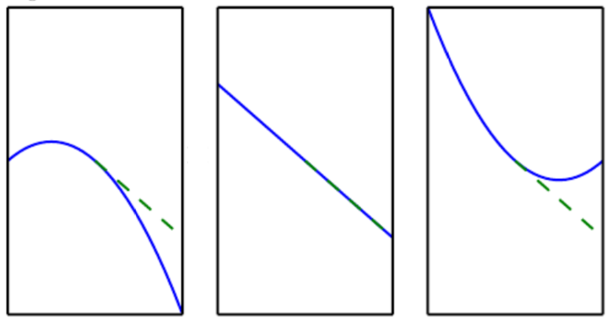
$$J(w+\epsilon) \approx J(w) + \epsilon J'(w)$$

- In **Newton's method**, we also have a second order term in the approximation (curvature)

$$J(w+\epsilon) \approx J(w) + J'(w)\epsilon + \frac{1}{2}J''(w)\epsilon^2$$

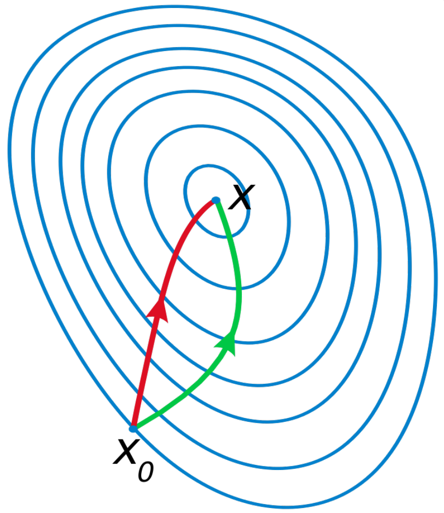
Newton's Method vs Gradient Descent

- Use of curvature allows us to better predict how $J(w)$ changes when we change w
  1. With negative curvature, J actually decreases faster than the gradient predicts.
  2. With no curvature, the gradient predicts the decrease correctly.
  3. With positive curvature, the function decreases more slowly than expected.
- The use of curvature mitigates with problems with case 1 and 3.



Newton's Method vs Gradient Descent

- Newton's method (red) typically enjoys faster convergence than (batch) gradient descent (green), and requires many fewer iterations to get very close to the minimum.



- One iteration of Newtons can, however, be more expensive than one iteration of gradient descent, since it requires finding and inverting an Hessian matrix.

Outline

Multi-Class Logistic Regression

- Training set $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^N$, where $y_i \in \{1, 2, \ldots, C\}$, and $C \geq 2$.
- **Multi-Class Logistic Regression** (aka **softmax regression**) uses the following probability model:
  - There is a weight vector $\mathbf{w}_c = (w_{c,0}, w_{c,1}, \ldots, w_{c,D})^\top$ for each class c. $\mathbf{W} = (\mathbf{w}_1, \ldots, \mathbf{w}_C)$ is the weight matrix.
  - The probability of a particular class c given $\mathbf{x}$ is:

$$p(y = c|\mathbf{x}, \mathbf{W}) = \frac{1}{Z(\mathbf{x}, \mathbf{W})} \exp(\mathbf{w}_c^\top \mathbf{x})$$

  where $Z(\mathbf{x}, \mathbf{W}) = \sum_{c'=1}^C \exp(\mathbf{w}_{c'}^\top \mathbf{x})$ is the normalization constant to ensure the probabilities of all classes sum to 1. It is called the **partition function**.
  - The **classification rule** is:

$$\hat{y} = \arg\max_y p(y|\mathbf{x}, \mathbf{W})$$

Relationship with Logistic Regression

When $C = 2$, name the two classes as $\{0, 1\}$ instead of $\{1, 2\}$:

$$p(y=0|\mathbf{x}, \mathbf{W}) = \frac{\exp(\mathbf{w}_0^\top \mathbf{x})}{\exp(\mathbf{w}_0^\top \mathbf{x}) + \exp(\mathbf{w}_1^\top \mathbf{x})}, \; p(y=1|\mathbf{x}, \mathbf{W}) = \frac{\exp(\mathbf{w}_1^\top \mathbf{x})}{\exp(\mathbf{w}_0^\top \mathbf{x}) + \exp(\mathbf{w}_1^\top \mathbf{x})}$$

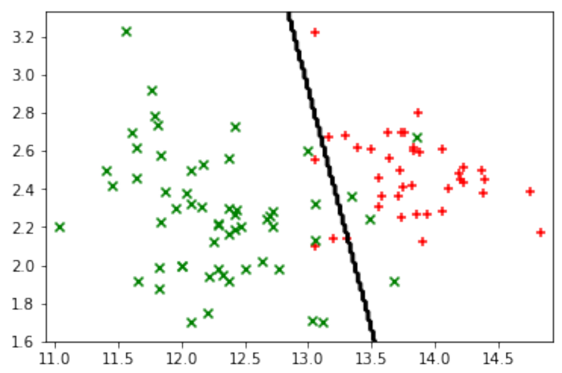
Dividing both numerator and denominator by $\exp(\mathbf{w}_1 \mathbf{x})$, we get

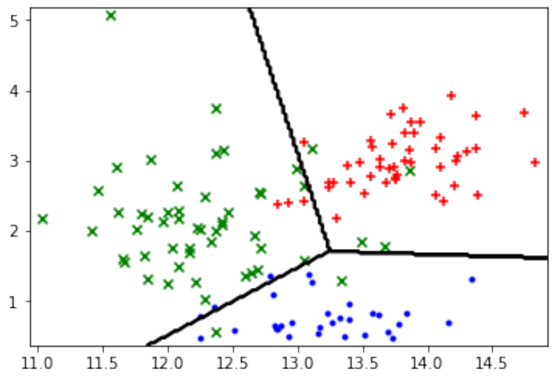
$$p(y=0|\mathbf{x}, \mathbf{W}) = \frac{\exp((\mathbf{w}_0^\top - \mathbf{w}_1^\top)\mathbf{x})}{\exp((\mathbf{w}_0^\top - \mathbf{w}_1^\top)\mathbf{x}) + 1}, \; p(y=1|\mathbf{x}, \mathbf{W}) = \frac{1}{\exp((\mathbf{w}_0^\top - \mathbf{w}_1^\top)\mathbf{x}) + 1}$$

Let $\mathbf{w} = \mathbf{w}_1 - \mathbf{w}_0$, we get the logistic regression model:

$$p(y=0|\mathbf{x}, \mathbf{w}) = \frac{\exp(-\mathbf{w}^\top \mathbf{x})}{1 + \exp(-\mathbf{w}^\top \mathbf{x})}, \; p(y=1|\mathbf{x}, \mathbf{w}) = \frac{1}{1 + \exp(-\mathbf{w}^\top \mathbf{x})}$$

Softmax Regression is a linear classifier





Cross Entropy Loss

- The cross entropy loss of the softmax regression model is:

$$
\begin{aligned}
J(\mathbf{w}) &= -\frac{1}{N}\sum_{i=1}^{N}\log P(y_i|\mathbf{x}_i,\mathbf{w}) \\
&= -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C}\mathbf{1}(y_i=c)\log P(y_i=c|\mathbf{x}_i,\mathbf{w}) \\
&= -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C}\mathbf{1}(y_i=c)\log\frac{\exp(\mathbf{w}_c^\top\mathbf{x}_i)}{\sum_{c'=1}^{C}\exp(\mathbf{w}_{c'}^\top\mathbf{x}_i)} \\
&= -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C}\mathbf{1}(y_i=c)[\mathbf{w}_c^\top\mathbf{x}_i-\log\sum_{c'=1}^{C}\exp(\mathbf{w}_{c'}^\top\mathbf{x}_i)] \\
&= -\frac{1}{N}\sum_{i=1}^{N}[\sum_{c=1}^{C}\mathbf{1}(y_i=c)\mathbf{w}_c^\top\mathbf{x}_i-\log\sum_{c'=1}^{C}\exp(\mathbf{w}_{c'}^\top\mathbf{x}_i)]
\end{aligned}
$$

Partial Derivative of Loss

- We would like to find the MLE of $\mathbf{W}$. To do so, we need to minimize the cross entropy loss. There is no closed-form solution. So, we resort to gradient descent.
- The partial derivative of cross entropy loss w.r.t each $\mathbf{w}_{c,j}$ is:

$$\begin{aligned}\frac{\partial J(\mathbf{w})}{\partial w_{c,j}} &= -\frac{1}{N}\sum_{i=1}^{N}[\frac{\partial}{\partial w_{c,j}}\sum_{c=1}^{C}\mathbf{1}(y_i=c)\mathbf{w}_c^\top\mathbf{x}_i - \frac{\partial}{\partial w_{c,j}}\log\sum_{c'=1}^{C}\exp(\mathbf{w}_{c'}^\top\mathbf{x}_i)] \\ &= -\frac{1}{N}\sum_{i=1}^{N}[\mathbf{1}(y_i=c)x_{i,j} - \frac{\exp(\mathbf{w}_c^\top\mathbf{x}_i)}{\sum_{c'=1}^{C}\exp(\mathbf{w}_{c'}^\top\mathbf{x}_i)}x_{i,j}] \\ &= -\frac{1}{N}\sum_{i=1}^{N}[\mathbf{1}(y_i=c) - p(y_i=c|\mathbf{x}_i,\mathbf{W})]x_{i,j}\end{aligned}$$

- Compare this to the gradient of logistic regression:

$$\frac{\partial J(\mathbf{w})}{\partial w_j} = -\frac{1}{N}\sum_{i=1}^{N}[y_i - \sigma(\mathbf{w}^\top\mathbf{x}_i)]x_{i,j}$$

Gradient of Loss

- The gradient of cross entropy loss w.r.t each $\mathbf{w}_c$ is:

$$\begin{aligned}\nabla_{\mathbf{w}_c} J(\mathbf{w}) &= \left(\frac{\partial J(\mathbf{w})}{\partial w_{c,0}}, \frac{\partial J(\mathbf{w})}{\partial w_{c,1}}, \ldots, \frac{\partial J(\mathbf{w})}{\partial w_{c,D}}\right)^\top \\ &= -\frac{1}{N}\sum_{i=1}^{N}[\mathbf{1}(y_i = c) - p(y_i = c|\mathbf{x}_i, \mathbf{W})]\mathbf{x}_i\end{aligned}$$

- Update rule in Gradient Descent:

$$\mathbf{w}_c \leftarrow \mathbf{w}_c + \alpha\frac{1}{N}\sum_{i=1}^{N}[\mathbf{1}(y_i = c) - p(y_i = c|\mathbf{x}_i, \mathbf{W})]\mathbf{x}_i$$

Outline

The Probabilistic Approach to Classification

So far, we have been talking about the probabilistic approach to classification:

- Start with training set $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^N$, where $y_i \in \{0, 1\}$.
- Learn a probabilistic model:

$$p(y|\mathbf{x}, \mathbf{w}) = Ber(y|\sigma(\mathbf{w}^\top \mathbf{x}))$$

  by minimizing the cross entropy loss:

$$J(\mathbf{w}) = -\frac{1}{N}\sum_{i=1}^{N} \log P(y_i|\mathbf{x}_i, \mathbf{w})$$

- Classify new instances using:

$$\hat{y} = 1 \text{ iff } p(y = 1|\mathbf{x}, \mathbf{w}) > 0.5$$

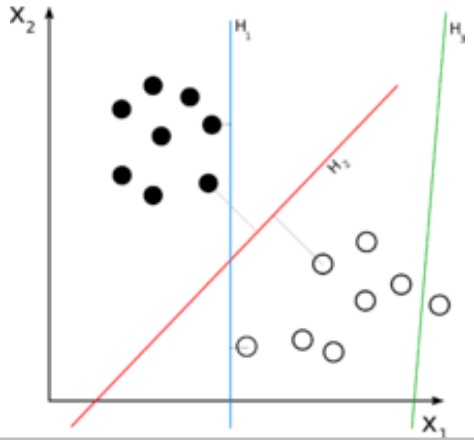
The Optimization Approach

Next, we will briefly talk about the optimization approach to classification:

- Start with training set $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^N$, where $y_i \in \{-\mathbf{1}, \mathbf{1}\}$,
- Learn a **linear classifier**

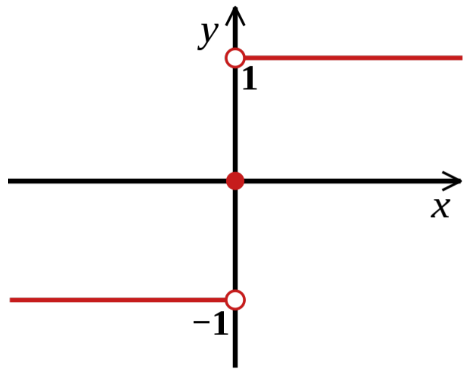
$$\hat{y} = \text{sign}(\mathbf{w}^\top \mathbf{x} + b)$$

where $\mathbf{x} = (x_1, \ldots, x_D)^\top$ and $\mathbf{w} = (w_1, \ldots, w_D)^\top$. We drop the convention $x_0 = 1$ and use b to replace w_0.



The Sign Function

- sign$x = 1$ if $x > 0$; sign$x = 0$ if $x = 0$; sign$x = -1$ if $x < 0$.



The Optimization Approach

- We determine $\mathbf{w}$ and b by minimizing the **training/empirical error**

$$L(\mathbf{w}, b) = \frac{1}{N}\sum_{i=1}^{N} L_{0/1}(y_i, \hat{y}_i)$$

where $L_{0/1}(y_i, \hat{y}_i) = \mathbf{1}(y_i \neq \hat{y}_i)$ is called the **zero/one loss function**.

- Let $z = \mathbf{w}^\top \mathbf{x} + b$. It is easy to see that

$$L_{0/1}(y, \hat{y}) = \mathbf{1}(y(\mathbf{w}^\top \mathbf{x} + b) \leq 0) = \mathbf{1}(yz \leq 0)$$

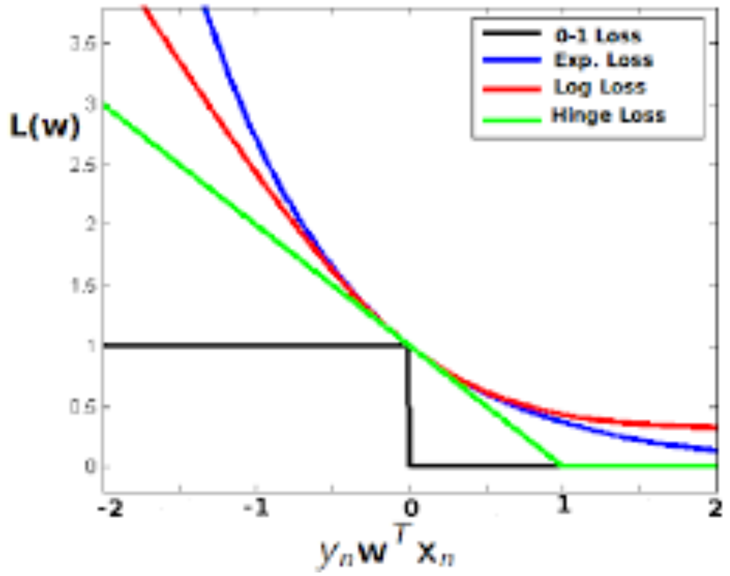
To overload the notation, we also write it as $L_{0/1}(y, z)$

- So, the loss function of linear classifiers is often written as

$$L(\mathbf{w}, b) = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}(y_i(\mathbf{w}^\top \mathbf{x}_i + b) \leq 0)$$

A problem the Zero/One Loss Function

- The zero/one loss function, as a function of **w** and *b*, has **zero gradient** almost everywhere and is not **convex**. Hence it is difficult to optimize



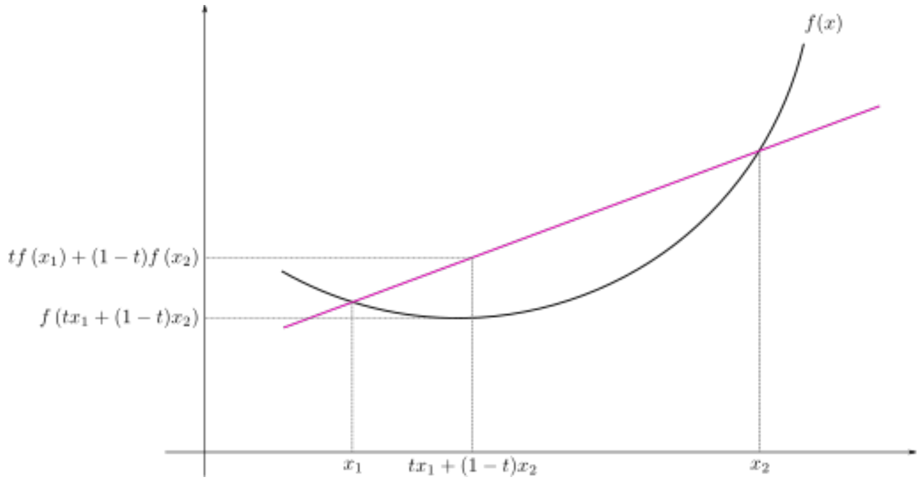
X-axis is yz.

Convex Function

- A real-valued function f is **convex** if the line segment between any two points on the graph of the function lies above or on the graph.
  - For any two points $x_1 < x_2$ and any $t \in [0, 1]$

$$f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2)$$

- f is **strictly convex** if the the equality holds only when $x_1 = x_2$.



Surrogate Loss Functions

- Convex functions are easy to minimize. Intuitively, if you drop a ball anywhere in a convex function, it will eventually get to the minimum. This is not true for non-convex functions.
- Since the zero/one loss is hard to optimize, we want to approximate it using a convex function and optimize that function.
- This approximating function will be called a **surrogate loss**.
- The surrogate losses need to be upper bounds on the true loss function: this guarantees that if you minimize the surrogate loss, you are also pushing down the real loss.

Logistic Loss

- One common surrogate loss function is the **logistic loss**:

$$\begin{aligned} L_{log}(y,z) &= \frac{1}{\log 2}\log(1+\exp(-yz)) \\ &= \begin{cases} \frac{1}{\log 2}\log(1+\exp(-(\mathbf{w}^\top\mathbf{x}+b))) & \text{if } y=1 \\ \frac{1}{\log 2}\log(1+\exp(\mathbf{w}^\top\mathbf{x}+b)) & \text{if } y=-1 \end{cases} \end{aligned}$$

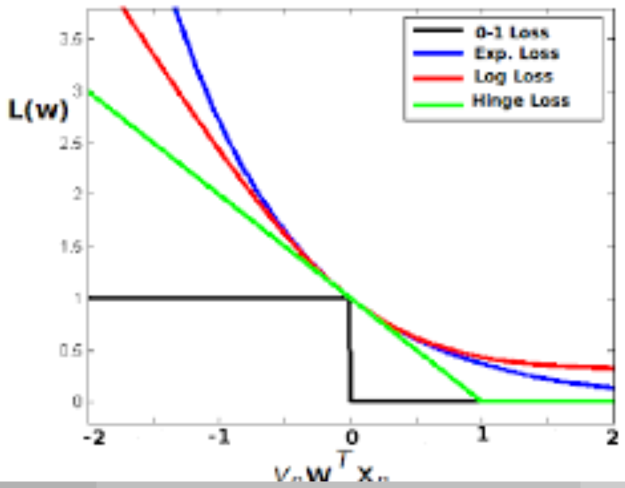
- On the other hand, each term in the NLL of logistic regression has the following form

$$\begin{aligned} -\log P(y|\mathbf{x},\mathbf{w}) &= -\{y\log\sigma(\mathbf{w}^\top\mathbf{x})+(1-y)\log(1-\sigma(\mathbf{w}^\top\mathbf{x}))\} \\ &= \begin{cases} -\log\frac{1}{1+\exp(-\mathbf{w}^\top\mathbf{x})} & \text{if } y=1 \\ -\log(1-\frac{1}{1+\exp(-\mathbf{w}^\top\mathbf{x})}) & \text{if } y=0 \end{cases} \\ &= \begin{cases} \log(1+\exp(-\mathbf{w}^\top\mathbf{x})) & \text{if } y=1 \\ \log(1+\exp(\mathbf{w}^\top\mathbf{x})) & \text{if } y=0 \end{cases} \end{aligned}$$

- So, *logistic regression is the same as linear classifier with logistic surrogate loss function*.

Other Surrogate Loss Functions

- **Zero/one loss**: $L_{0/1}(y, z) = \mathbf{1}(yz \leq 0)$.
- **Logistic loss**: $L_{log}(y, z) = \frac{1}{\log 2} \log(1 + \exp(-yz))$
- **Hinge loss**: $L_{hin}(y, z) = \max\{0, 1 - yz\}$
- **Exponential loss**: $L_{exp}(y, z) = \exp(-yz)$
- **Squared loss**: $L_{sqr}(y, z) = (y - z)^2$



Loss and Error

When a surrogate loss function is used,

- There are two ways to measure how a classifier performs on the training set: **training loss** and **training error**.
- There are also two ways to measure how a classifier performs on the set set: **test loss** and **test error**.
- As in the case of regression, test loss/error can be improved using regularization.