

Classification

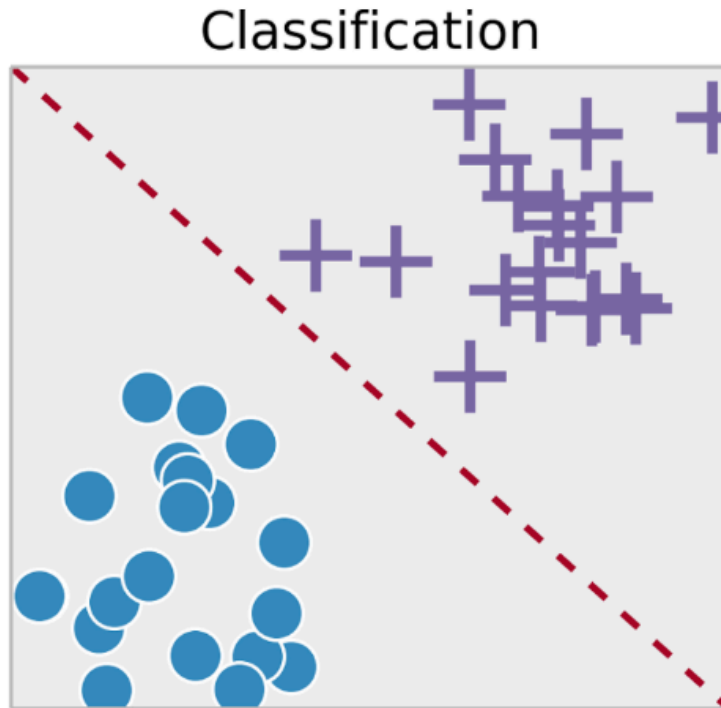
Liang Zheng

Australian National University

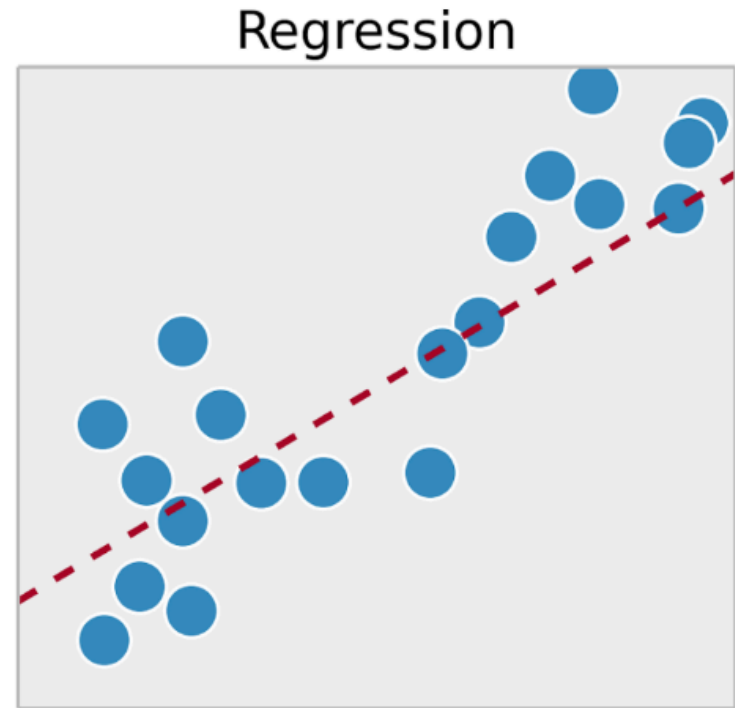
liang.zheng@anu.edu.au

Reference: Bishop, Christopher M. *Pattern recognition and machine learning*. Springer, 2006.

Classification problem



Goal: The goal in classification is to take an input vector \mathbf{x} and to assign it to one of K discrete classes $\mathcal{C}_k, k = 1, \dots, K$.



Goal: The goal in regression is to take an input vector \mathbf{x} and predict the value of a real number.

Regression

Training data

$$\mathcal{D} = \{ (\mathbf{x}_n, y_n) \mid n = 1, \dots, N \}$$

- Features/Inputs $\mathbf{x}_n \in \mathbb{R}^D$
- Response/Output $y_n \in \mathbb{R}$

Classification

Training data

$$\mathcal{D} = \{ (\mathbf{x}_n, y_n) \mid n = 1, \dots, N \}$$

- Features/Inputs $\mathbf{x}_n \in \mathbb{R}^D$
- Labels/Output $y_n \in$ a set of discrete numbers

4.1.1 Two classes

- Linear classifiers $h: \mathbb{R}^D \rightarrow \{-1, +1\}$

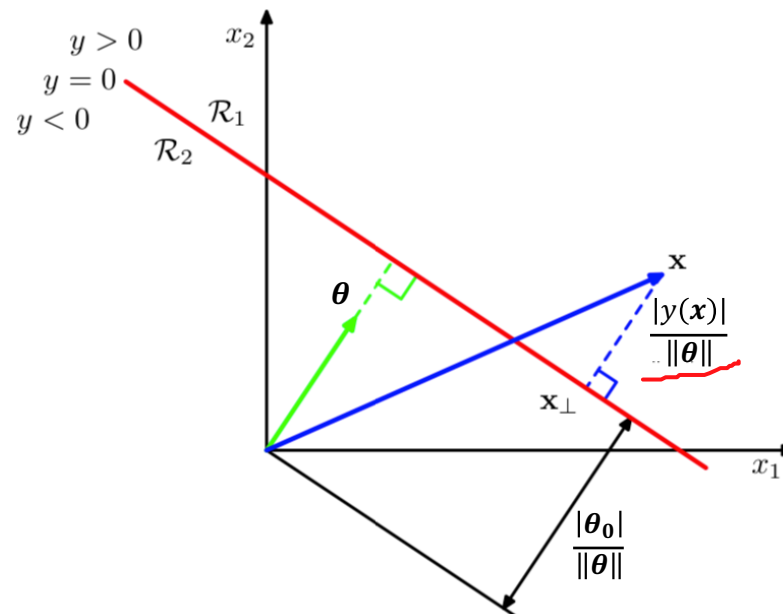
$$h(\mathbf{x}; \underline{\boldsymbol{\theta}}, \theta_0) = \text{sign}(y(\mathbf{x})) = \text{sign}(\underline{\boldsymbol{\theta}}^T \mathbf{x} + \theta_0)$$

where $\boldsymbol{\theta}$ is called a weight vector, and θ_0 is a **bias** or **offset**. $y(\mathbf{x})$ is the **discriminant function**.

$$\text{sign}(y) = \begin{cases} +1 & \text{if } y \geq 0, \\ -1 & \text{if } y < 0. \end{cases}$$

An input vector \mathbf{x} is assigned to class $+1$ if $y \geq 0$ and to class -1 otherwise.

- The **decision boundary** is defined by the hyperplane $y(\mathbf{x}) = 0$, a $(D - 1)$ -dimensional hyperplane within the D -dimensional input space.

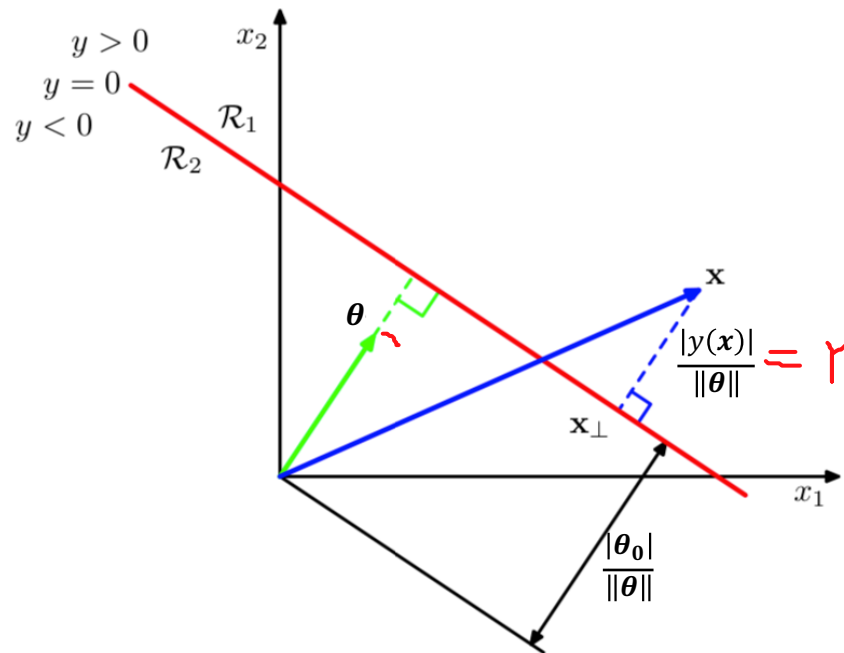


- Two points \mathbf{x}_A and \mathbf{x}_B both lie on the decision boundary $y(\mathbf{x}) = 0$. Then, we have $\boldsymbol{\theta}^T(\mathbf{x}_A - \mathbf{x}_B) = 0$ and hence the vector $\boldsymbol{\theta}$ is orthogonal to the decision boundary.
- Consider a point \mathbf{x} and let \mathbf{x}_\perp be its orthogonal projection onto the decision boundary. The perpendicular distance between \mathbf{x} and the decision boundary is given by,

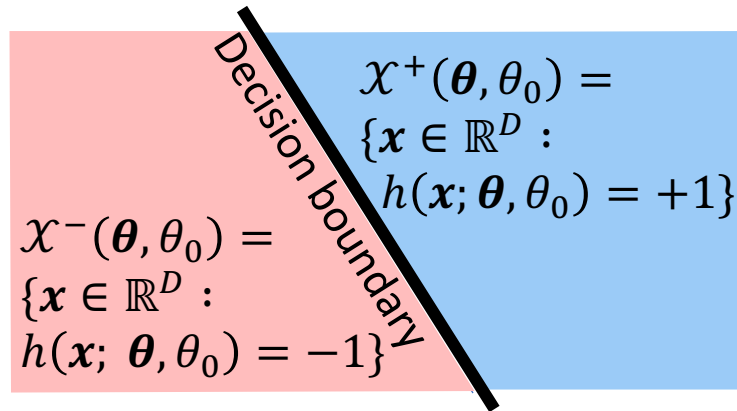
$$r = \frac{|y(\mathbf{x})|}{\|\boldsymbol{\theta}\|}$$

- The distance from the origin to the decision surface is given by

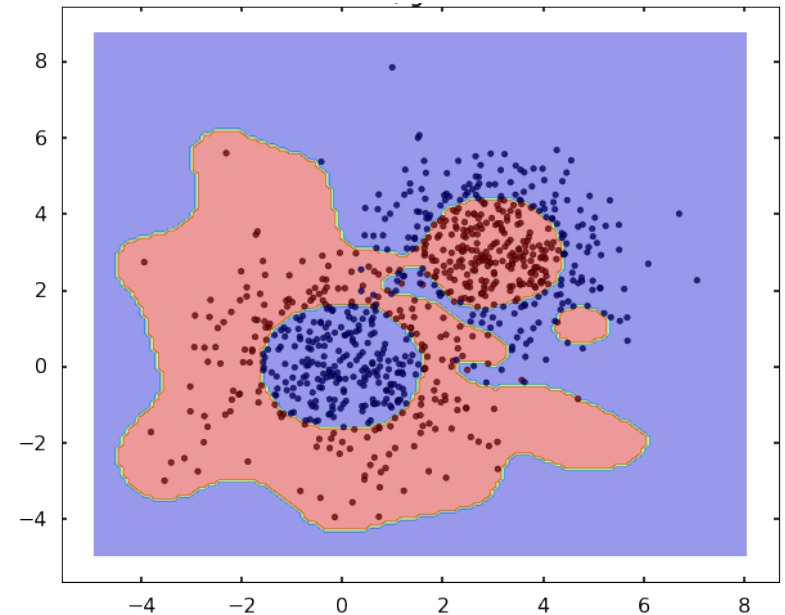
$$\frac{|\boldsymbol{\theta}_0|}{\|\boldsymbol{\theta}\|}$$



Decision Regions



linear classifier

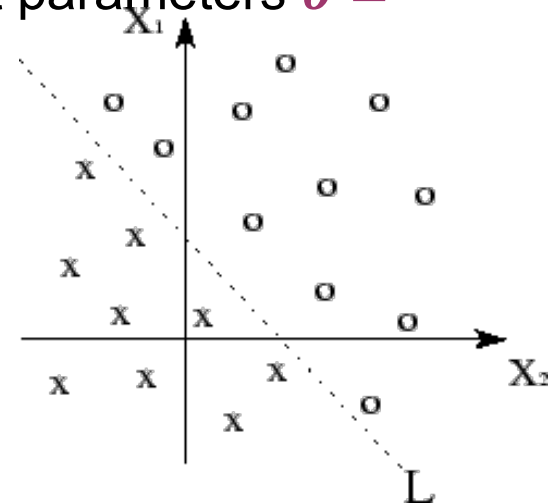


non-linear classifier

A classifier h partitions the space into **decision regions** that are separated by **decision boundaries**. In each region, all the points map to the same label. Many regions could have the same label.

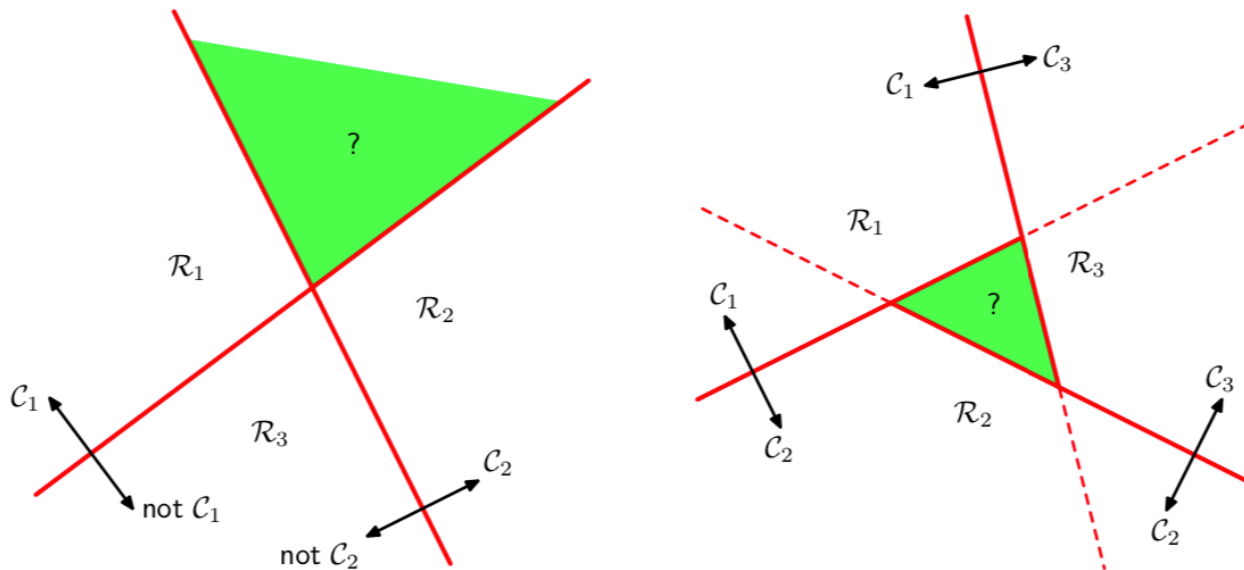
4.1.1 Two classes

- Compact representation
- Similar to linear regression, it is convenient to use a more compact notation in which we introduce an additional dummy 'input' value $x_0 = 1$ and then define $\theta = [\theta_0, \theta_1, \theta_2, \dots, \theta_D]^T$ and $x = [x_0, x_1, x_2, \dots, x_D]^T$ so that
$$y(x) = \theta^T x$$
- In this case, the decision surfaces are D -dimensional hyperplanes passing through the origin of the $D + 1$ -dimensional expanded input space.
- The training data \mathcal{D} is linearly ~~x~~ separable if there exist parameters $\theta = [\theta_0, \theta_1, \theta_2, \dots, \theta_D]^T$ such that for all $(x, y) \in \mathcal{D}$,
$$\underline{y(\theta^T x) > 0}$$



4.1.2 Multiple classes

- Now we consider a linear classifier for $K > 2$ classes.



对于多分类有两种策略：

- One-versus-the-rest classifier. Use $K - 1$ classifiers. Each classifier solves a two-class problem: separating points in a particular class \mathcal{C}_k from points not in that class.
- One-versus-one classifier. Use $K(K - 1)/2$ classifiers, one for every possible pair of classes. Each point is classified according to a majority vote amongst the classifiers.
- However, both methods suffer from the problem of ambiguous regions.

4.1.2 Multiple classes

- To avoid these difficulties, we consider a single K -class classifier comprising K linear functions of the form

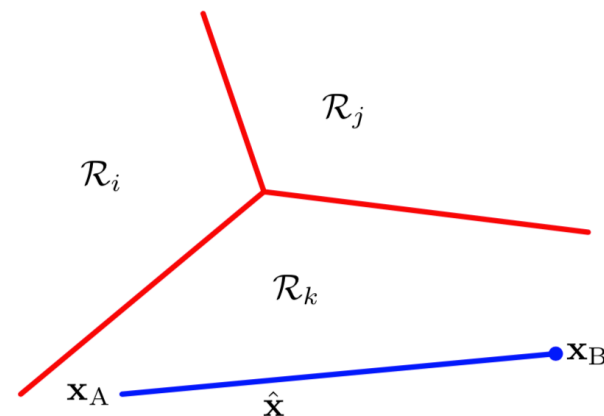
$$\underline{y_k(\mathbf{x})} = \underline{\boldsymbol{\theta}_k^T \mathbf{x} + \theta_{k0}} \quad \text{将x归类为使y最大的k}$$

- we then assign a point \mathbf{x} to class \mathcal{C}_k if $y_k(\mathbf{x}) > y_j(\mathbf{x})$ for all $j \neq k$.
- The decision boundary between class \mathcal{C}_k and class \mathcal{C}_j is therefore given by $y_k(\mathbf{x}) = y_j(\mathbf{x})$ and hence corresponds to a $(D - 1)$ -dimensional hyperplane defined by

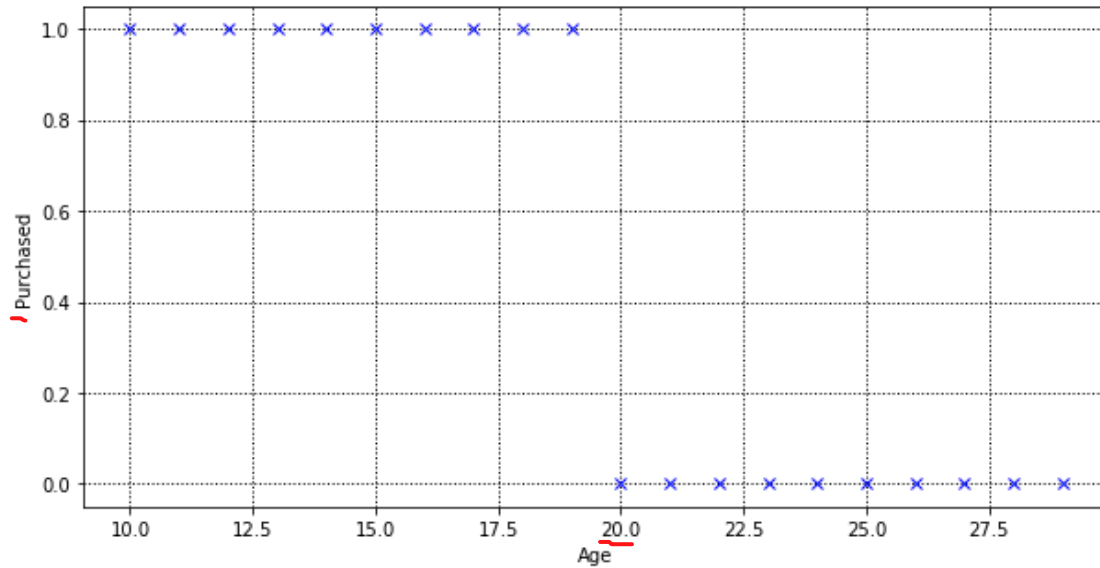
$$(\boldsymbol{\theta}_k - \boldsymbol{\theta}_j)^T \mathbf{x} + (\theta_{k0} - \theta_{j0}) = 0$$

- This has the same form as the decision boundary for the **two-class** case discussed in Section 4.1.1, and so analogous geometrical properties apply.
- The decision regions of this classifier are singly connected and convex

“Singly connected and convex”: Two points \mathbf{x}_A and \mathbf{x}_B both lie inside decision region \mathcal{R}_k . Any point $\hat{\mathbf{x}}$ that lies on the line connecting \mathbf{x}_A and \mathbf{x}_B must also lie in \mathcal{R}_k

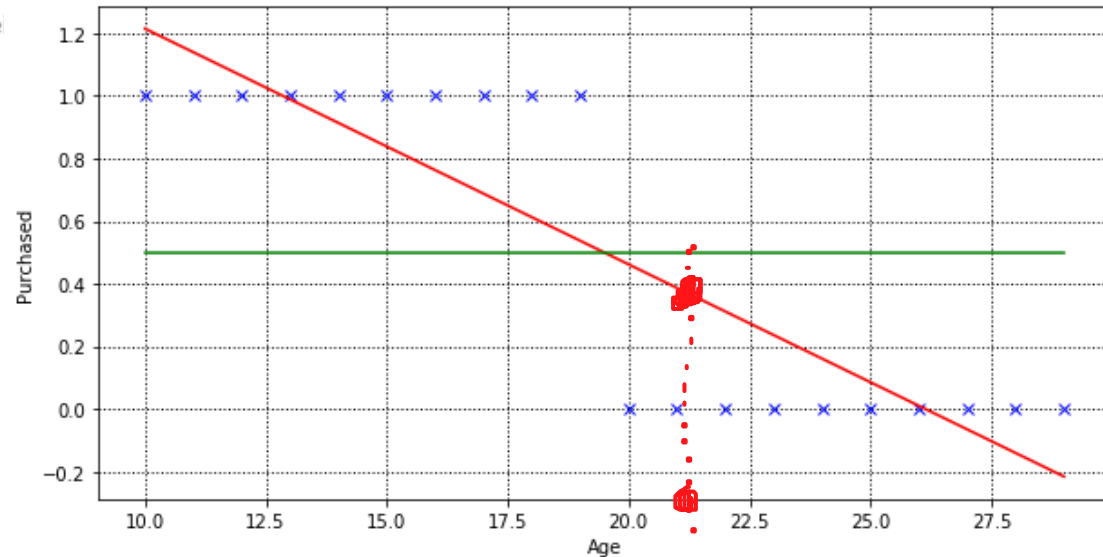


4.1.3 Least squares for classification



Our sample training dataset of 20 customers and the

You can use 0.5 as threshold to tell which class a data point belongs to



4.1.3 Least squares for classification

- We use the tools from least squares to solve classification.
- Problem setting
- Each class \mathcal{C}_k is described by its own compact linear model

$$y_k(\mathbf{x}) = \underline{\theta_k^T \mathbf{x}}$$

where $k = 1, \dots, K$. We group these together using vector notation

$$\underline{\mathbf{y}(\mathbf{x})} = \underline{\Theta^T \mathbf{x}}$$

- Where Θ is a matrix whose k^{th} column comprises the $D + 1$ -dimensional vector $\theta_k = (\theta_{k0}, \theta_{k1}, \dots, \theta_{kD})^T$ and $\mathbf{x} = (x_0, x_1, x_2, \dots, x_D)^T$ is the corresponding augmented input vector with a dummy input $x_0 = 1$.
- A new input \mathbf{x} is assigned to the class for which $y_k = \theta_k^T \mathbf{x}$ is largest.

4.1.3 Least squares for classification

- We determine the parameter matrix Θ by minimizing the square error function
- Consider a training data set $\{\mathbf{x}_n, \mathbf{y}_n\}$ where $n = 1, \dots, N$, and define a matrix \mathbf{Y} whose n^{th} row is the vector \mathbf{y}_n^T , together with a matrix \mathbf{X} whose n^{th} row is \mathbf{x}_n^T . The square error function is written as Tr:迹, 即方阵对角线上元素的和

$$L(\Theta) = \frac{1}{2} \text{Tr}\{(\mathbf{X}\Theta - \mathbf{Y})^T(\mathbf{X}\Theta - \mathbf{Y})\} \quad \text{使用最小二乘的迹表示error}$$

- Setting the derivative with respect to Θ to zero, and rearranging, we obtain the closed-form solution for Θ :

$$\Theta = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y} = \mathbf{X}^\dagger \mathbf{Y}$$

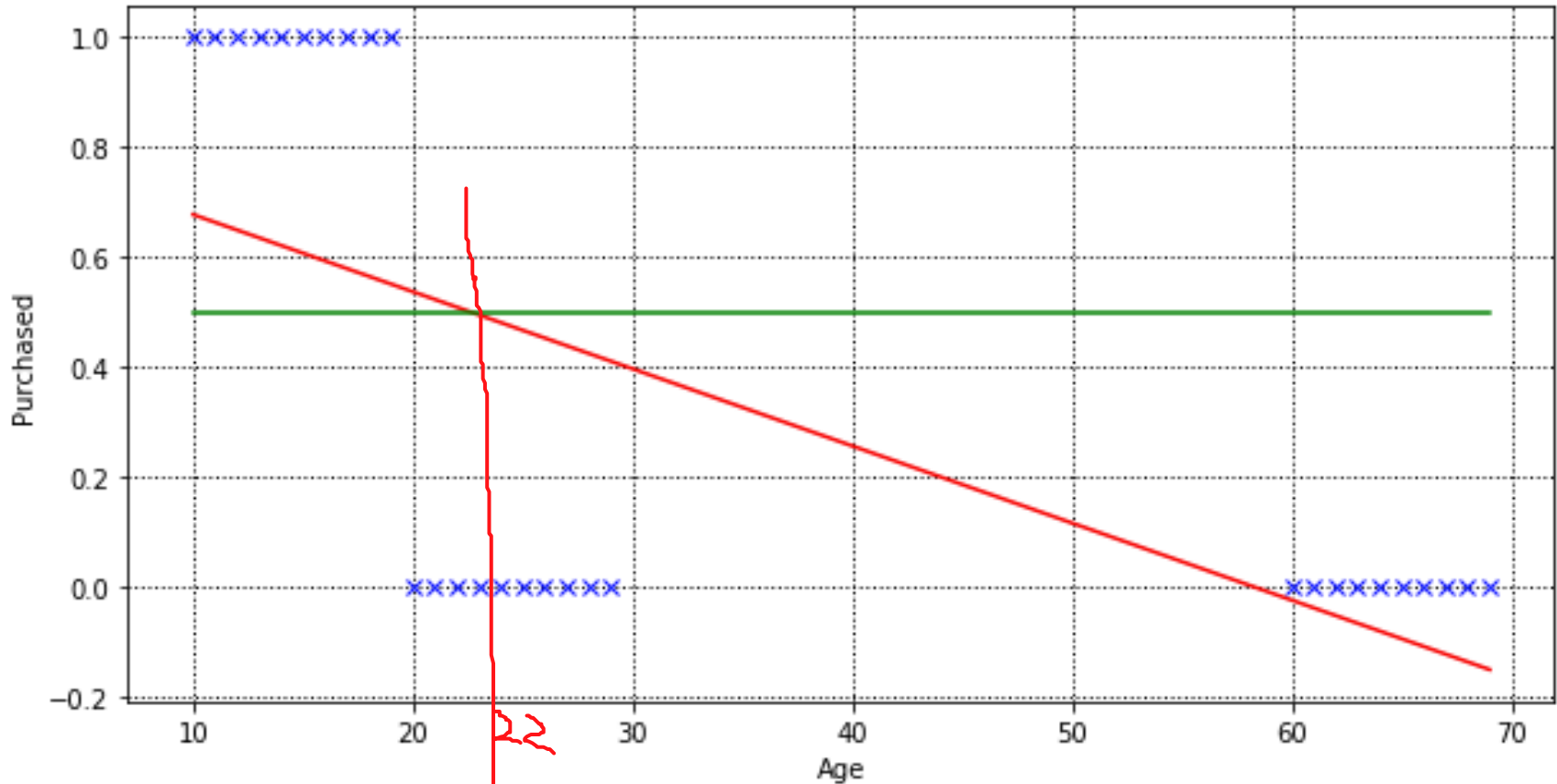
- where \mathbf{X}^\dagger is the pseudo-inverse of \mathbf{X} . We obtain

$$\mathbf{y}(\mathbf{x}) = \Theta^T \mathbf{x} = \mathbf{Y}^T (\mathbf{X}^\dagger)^T \mathbf{x}$$

- $\mathbf{y}(\mathbf{x})$ is a $K \times 1$ -dim vector. The predicted class label corresponds to the largest value in $\mathbf{y}(\mathbf{x})$.

4.1.3 Least squares for classification

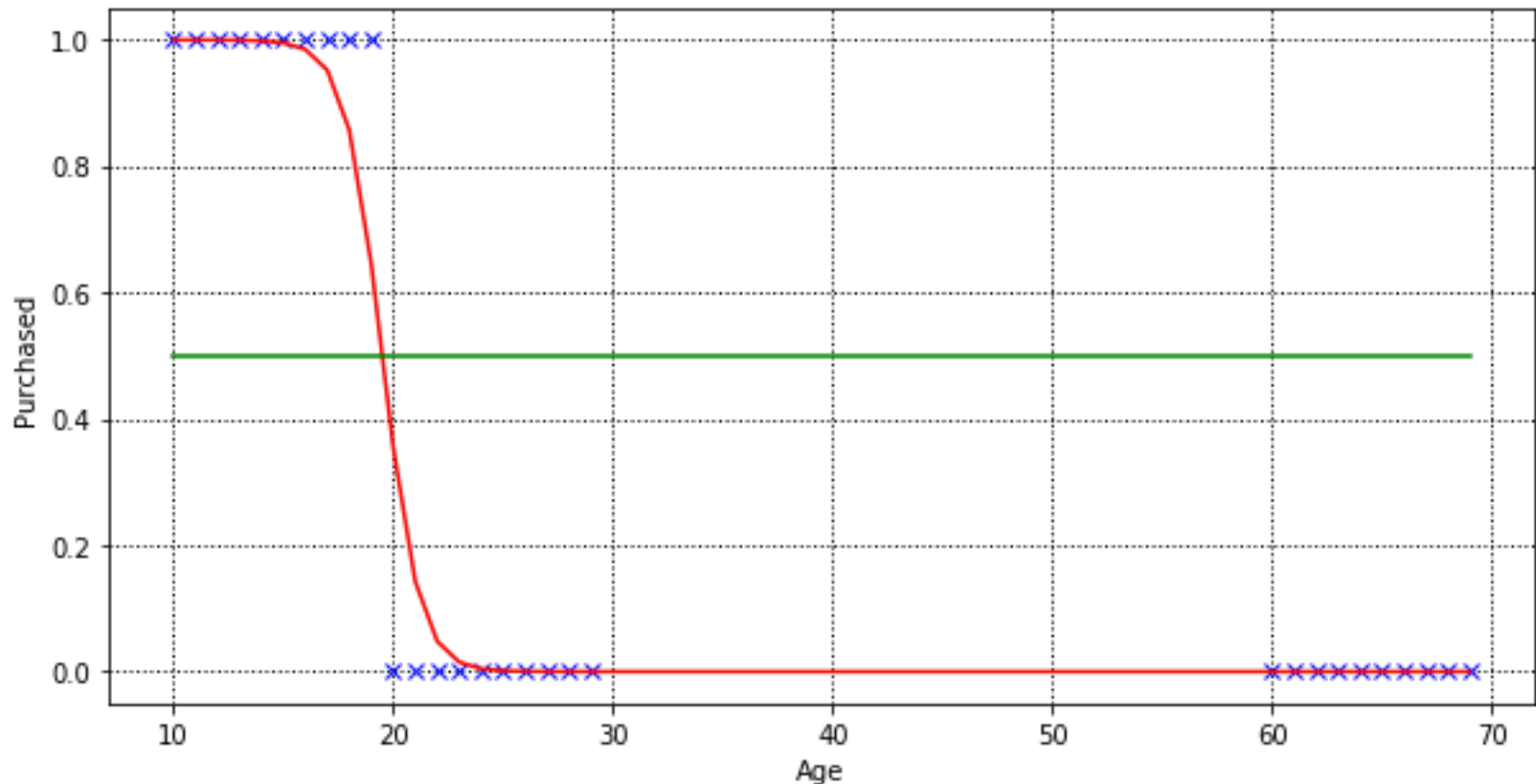
- Problems with Least squares



- If you still use 0.5 as the threshold, people between 20 and 22 will be misclassified.

4.1.3 Least squares for classification

- We can use Logistic regression to solve this problem
- Note: logistic regression is a classification method, not a linear regression method.



4.1.7 The Perceptron Algorithm

Rosenblatt (1962)

$\mathcal{L}_1(\boldsymbol{\theta}; \mathbf{x}, y)$ is the loss for one training sample (\mathbf{x}, y)

Classifier:

$$h(\mathbf{x}; \boldsymbol{\theta}) = \text{sign}(\boldsymbol{\theta}^T \mathbf{x})$$

Let $\mathcal{L}_1(\boldsymbol{\theta}; \mathbf{x}, y) = 1$ (0 otherwise) if

- $y \neq h(\mathbf{x}; \boldsymbol{\theta})$, or ← [misclassified]
- (\mathbf{x}, y) is on decision boundary ← [boundary]

Note that $y(\boldsymbol{\theta}^T \mathbf{x}) \leq 0$ if

- $\boldsymbol{\theta}^T \mathbf{x}$ and y differ in sign, or [misclassified]
- $\boldsymbol{\theta}^T \mathbf{x}$ is zero [boundary]

$$\mathcal{L}_1(\boldsymbol{\theta}; \mathbf{x}, y) = \mathbb{I}[y(\boldsymbol{\theta}^T \mathbf{x}) \leq 0] = \text{Loss}(y(\boldsymbol{\theta}^T \mathbf{x}))$$

where $\text{Loss}(z) = \mathbb{I}[z \leq 0]$ is the **zero-one loss**.

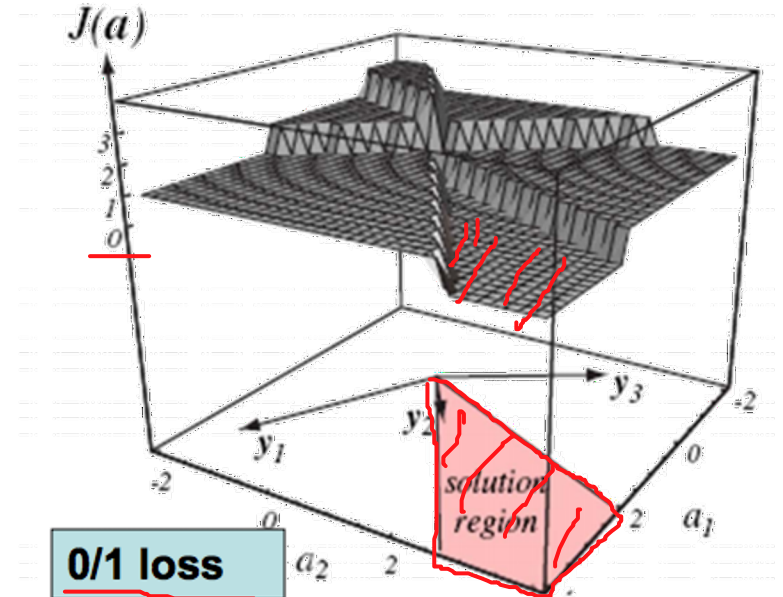
Training Loss

$$\text{Loss}(z) = \mathbb{I}[z \leq 0]$$

$$\mathcal{L}_1(\boldsymbol{\theta}; \mathbf{x}, y) = \text{Loss}(y(\boldsymbol{\theta}^T \mathbf{x}))$$

$$\mathcal{L}_n(\boldsymbol{\theta}; \mathcal{D}) = \frac{1}{n} \sum_{(\mathbf{x}, y) \in \mathcal{D}} \mathcal{L}_1(\boldsymbol{\theta}; \mathbf{x}, y)$$

数据集的loss是每个数据点loss的均值



Gradient is zero almost everywhere!

Gradient descent not possible.

Perceptron – a Mistake-Driven Algorithm

1. Initialize $\theta = 0$.
2. For each data $(x, y) \in \mathcal{D}$,
 - a. Check if $h(x; \theta) = y$.
 - b. If not, update θ to correct the mistake.
3. Repeat Step (2) until no mistakes are found.

Perceptron Algorithm

Weights may be initialized to $\mathbf{0}$ or to a small random value

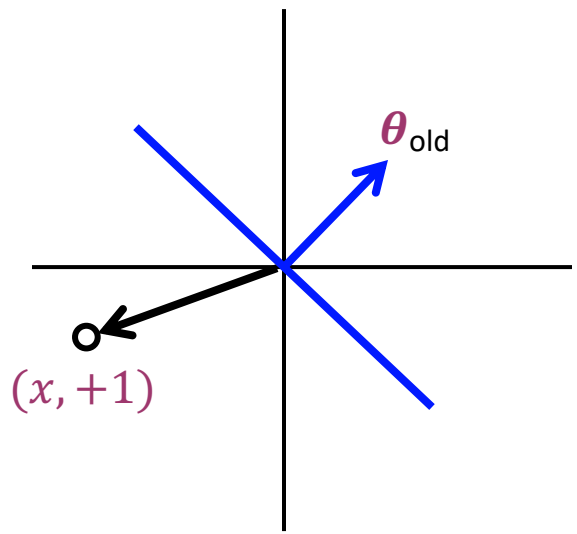
1. Initialize $\theta = \mathbf{0}$.
2. For each data $(x, y) \in \mathcal{S}_n$,
 - a. If $y(\theta^\top x) \leq 0$,
 - i. $\theta \leftarrow \theta + yx$.
3. Repeat Step (2) until no mistakes are found.

Due to the constant feature trick
 $x_0 = 1$, update for θ_0 will be
 $\theta_0 \leftarrow \theta_0 + yx_0 = \theta_0 + y$.

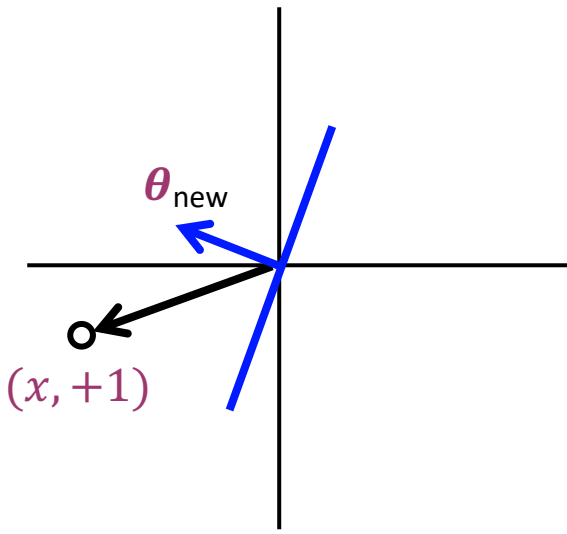
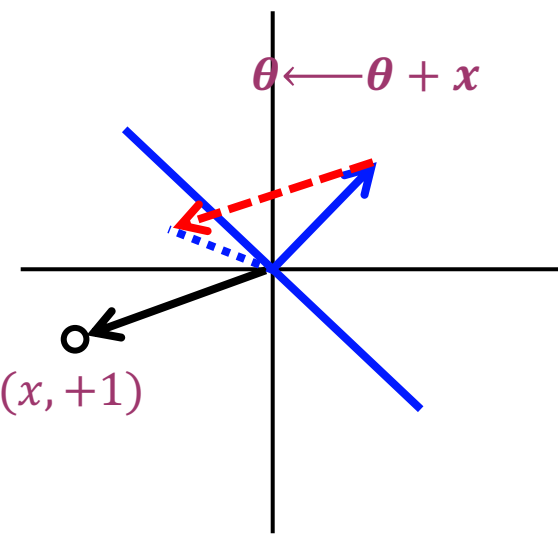
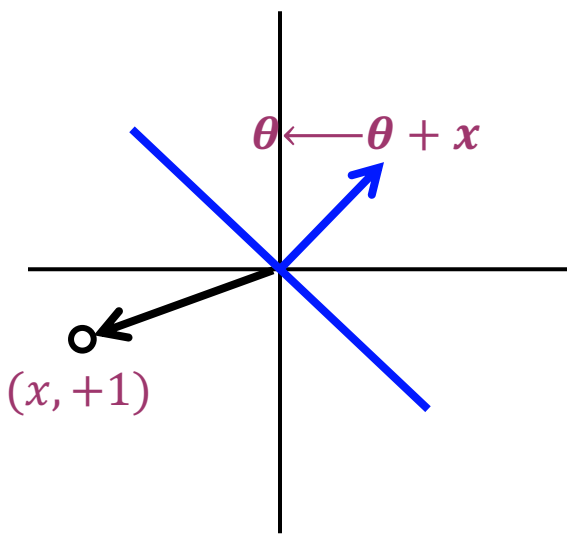
Intuition behind $\theta \leftarrow \theta + yx$.

- We are making a mistake on a ⁺ positive sample
- That is, $y = +1$, but $\theta^\top x \leq 0$
- According to this update, the new vector $\theta_{new} = \theta_{old} + yx = \theta_{old} + x$
- The new prediction will be
$$\theta_{new}^\top x = (\theta_{old} + x)^\top x = \theta_{old}^\top x + x^\top x > \theta_{old}^\top x$$
- For a positive example, the Perceptron update will increase the score assigned to the same input

Intuition behind $\theta \leftarrow \theta + yx$.



A mistake on a positive sample



Example

Training data

- $(\mathbf{x}_1, y_1) = ((2, 2)^\top, +1)$
- $(\mathbf{x}_2, y_2) = ((2, -1)^\top, -1)$

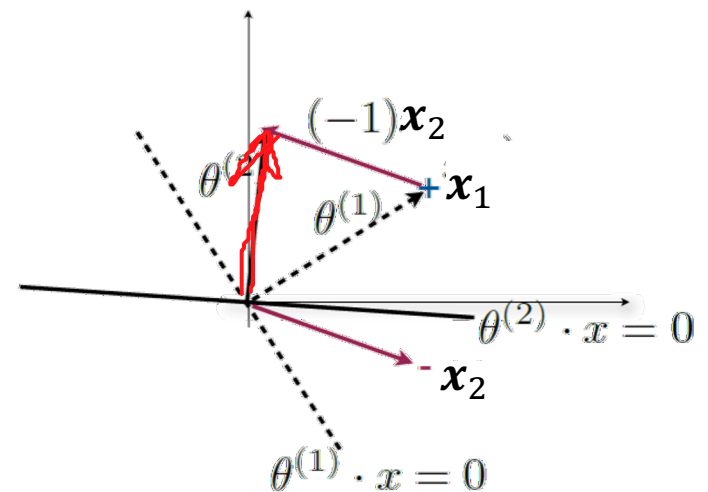
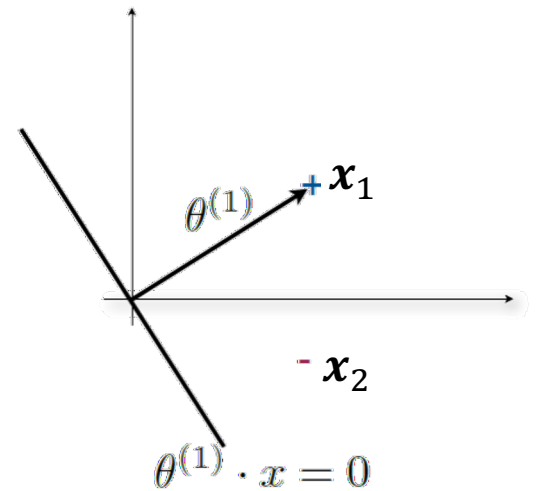
Apply the perceptron algorithm to the data to find a classifier $h(\mathbf{x}; \boldsymbol{\theta})$, $\boldsymbol{\theta} = (\theta_1, \theta_2)$, that separates the data.

Example

Training data

- $(\mathbf{x}_1, y_1) = ((2, 2)^\top, +1)$
- $(\mathbf{x}_2, y_2) = ((2, -1)^\top, -1)$

- Initialize $\boldsymbol{\theta} = (0, 0)^\top$.
- Since $y_1 \boldsymbol{\theta}^\top \mathbf{x}_1 = 0$,
set $\boldsymbol{\theta} = (0, 0) + (2, 2)^\top = (2, 2)^\top$.
- Since $y_2 \boldsymbol{\theta}^\top \mathbf{x}_2 = -2$,
set $\boldsymbol{\theta} = (2, 2)^\top - (2, -1)^\top = (0, 3)^\top$.
- $y_1 \boldsymbol{\theta}^\top \mathbf{x}_1 = 6 > 0$.
- $y_2 \boldsymbol{\theta}^\top \mathbf{x}_2 = 3 > 0$.
- No more mistakes, so we are done.



Perceptron Algorithm

1. Training Set (Linearly Separable)

$$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)$$

2. Model (Set of Perceptrons)

$$h(\mathbf{x}; \boldsymbol{\theta}) = \text{sign}(\theta_0 x_0 + \theta_1 x_1 + \dots + \theta_D x_D)$$

3. Training Loss (Fraction of Misclassified/Boundary Points)

$$\mathcal{L}_n(\boldsymbol{\theta}) = \frac{1}{N} \sum_{(\mathbf{x}, y) \in \mathcal{D}} \mathbb{I}[y(\boldsymbol{\theta}^\top \mathbf{x}) \leq 0]$$

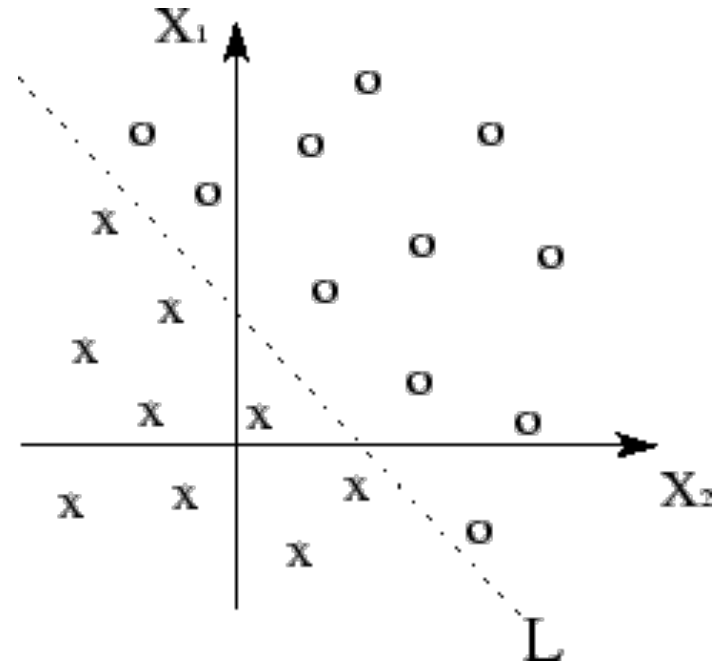
3. Algorithm (Mistake-Driven Algorithm)

Linearly Separable

The training data \mathcal{D} is
linearly separable
if there exists a
parameters θ and θ_0 such
that for all $(x, y) \in \mathcal{D}$,

$$\underline{y(\theta^\top x + \theta_0) > 0.}$$

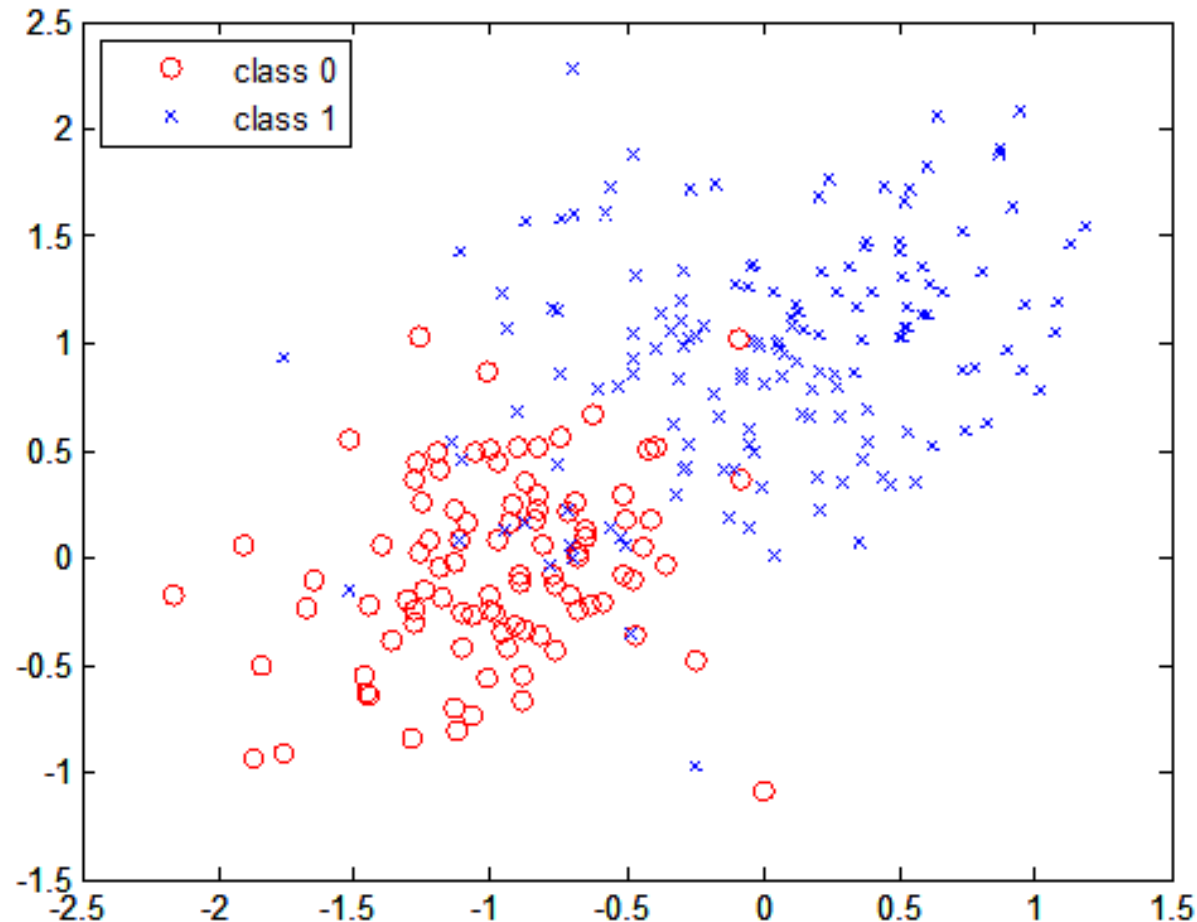
Perceptron algorithm can only be
applied to dataset that is linearly
separable



Check your understanding

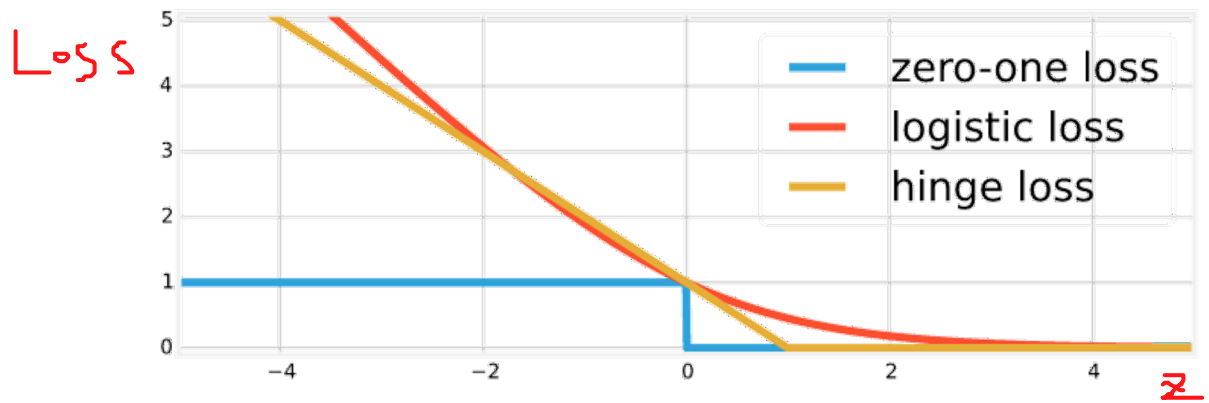
- Classification deals with discrete label space. ✓
- We can do multi-way classification using a single classifier. ✓
- In multi-way classification $y(x) = \Theta^T x$, the classifier weight θ_k can be shared by different classes. ✓
- We can devise a classifier based on linear regression. ✓ 技术上可行，但是效果会很差
- In Perceptron algorithm, whenever a sample is misclassified, ~~one~~ step of θ update $\theta \leftarrow \theta + yx$ will correct it. ✗
- Trained on a linearly separable training set, a perceptron classifier will make no mistake on both training and test sets. ✗

Hinge Loss



Perceptron algorithm does not converge for training sets that are not linearly separable.

Loss Functions



Training Loss

$$\mathcal{L}_n(\boldsymbol{\theta}) = \frac{1}{N} \sum_{(x,y) \in \mathcal{D}} \text{Loss}(\underbrace{y(\boldsymbol{\theta}^\top \mathbf{x})}_{\mathbf{z}})$$

Zero-One Loss

$$\text{Loss}_{01}(z) = \mathbb{I}_{\underline{z \leq 0}}$$

Hinge Loss

$$\underline{\text{Loss}_H(z) = \max\{1 - z, 0\}}$$

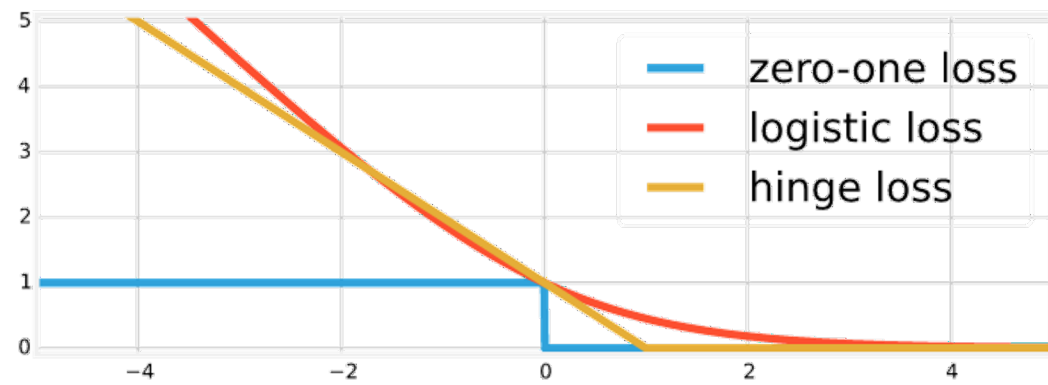
CONVEX!

Penalize large mistakes more.

Penalize near-mistakes, i.e. $0 \leq z \leq 1$.

错误越大惩罚越大

Hinge Loss



Find θ that minimizes


$$\begin{aligned}\mathcal{L}_n(\theta) &= \frac{1}{N} \sum_{(x,y) \in \mathcal{D}} \text{Loss}_H(z) \\ &= \frac{1}{N} \sum_{(x,y) \in \mathcal{D}} \max\{1 - z, 0\} \\ &= \frac{1}{N} \sum_{(x,y) \in \mathcal{D}} \max\{1 - y(\theta^\top x), 0\}\end{aligned}$$

Gradient

$$\nabla_z \text{Loss}_H(z) = \begin{cases} 0 & \text{if } z > 1, \\ -1 & \text{otherwise.} \end{cases}$$

$$\nabla_{\theta} \text{Loss}_H(y(\theta^\top x)) = \begin{cases} 0 & \text{if } y(\theta^\top x) > 1, \\ -yx & \text{otherwise.} \end{cases}$$

Stochastic Gradient Descent

1. Initialize $\theta = 0$.
2. Select data $(\mathbf{x}, y) \in \mathcal{D}$ at random.
 - a. If $y(\theta^\top \mathbf{x}) \leq 1$, then
 - i. $\theta \leftarrow \theta + \eta_k y \mathbf{x}$.
3. Repeat Step (2) until convergence.  learning rate
(e.g. when improvement in $\mathcal{L}(\theta)$ is small enough)

Differences from Perceptron Algorithm

- Check $z \leq 1$ rather than $z \leq 0$
- η_k rather than $\eta = 1$

Hinge Loss Algorithm

1. Training Set (Not Necessarily Linearly Separable)

$$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)$$

2. Model (Set of Perceptrons)

$$h(\mathbf{x}; \boldsymbol{\theta}) = \text{sign}(\theta_0 x_0 + \theta_1 x_1 + \dots + \theta_D x_D)$$

3. Training Loss (Hinge Loss)

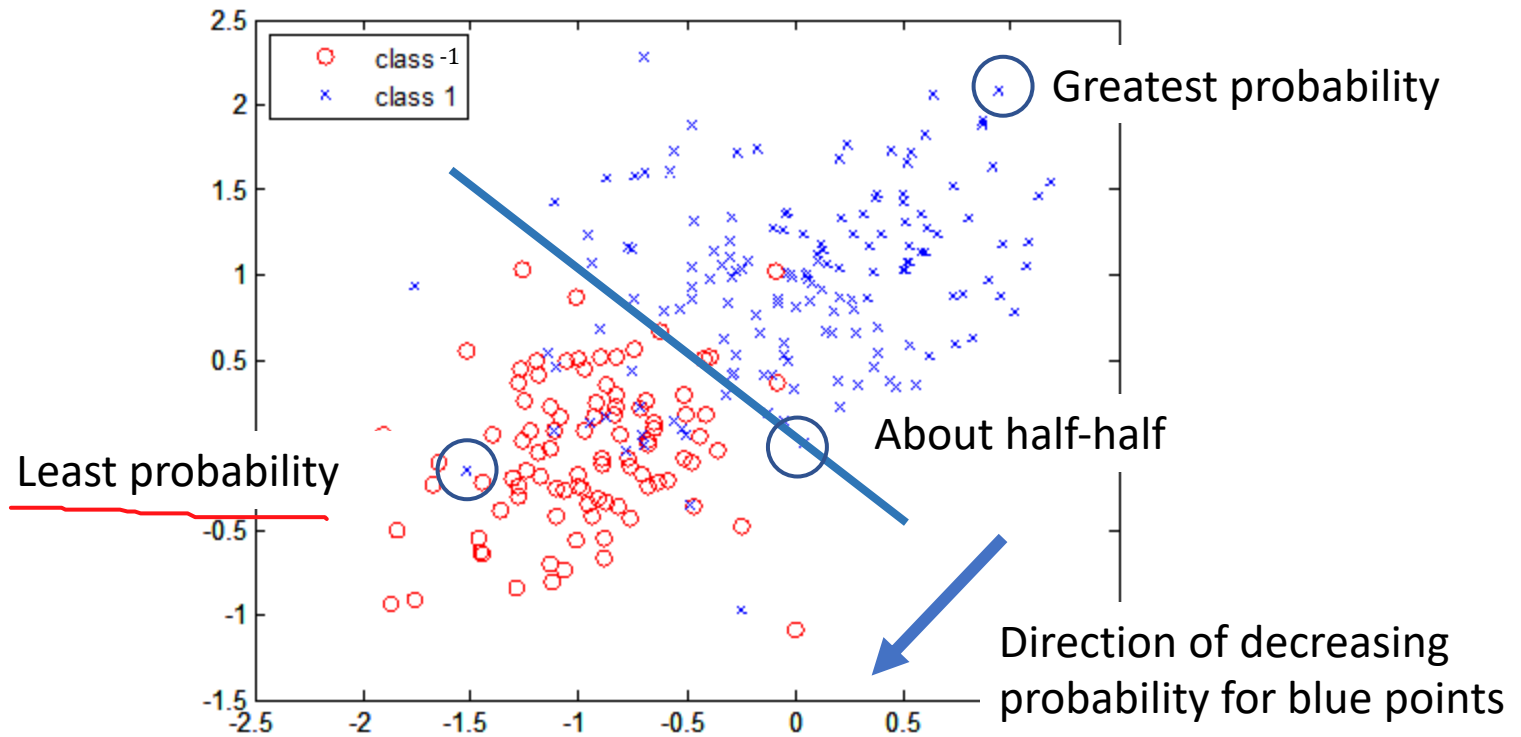
$$\mathcal{L}_n(\boldsymbol{\theta}) = \frac{1}{N} \sum_{(\mathbf{x}, y) \in \mathcal{D}} \max\{1 - y(\boldsymbol{\theta}^\top \mathbf{x}), 0\}$$

3. Algorithm (Gradient Descent)

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \frac{\eta_k}{N} \sum_{(\mathbf{x}, y) \in \mathcal{D}} y \mathbf{x}$$

N: usually use a mini batch,
so can be replaced by M

Logistic Regression



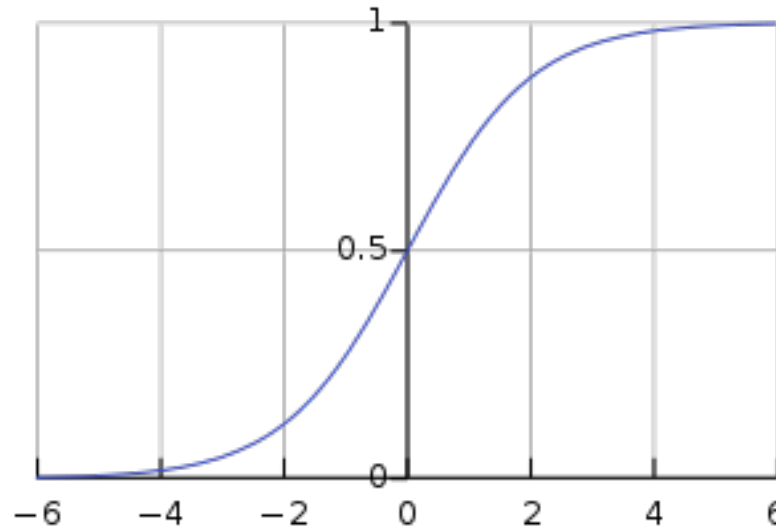
Probabilistic Model

Model the probability that the label y is $+1$ given the feature is \mathbf{x} .

$$h: \mathbb{R}^D \rightarrow [0, 1] \quad \text{即概率}$$

$$\underline{h(\mathbf{x}; \boldsymbol{\theta}) = \mathbb{P}(y = +1 | \mathbf{x}) = \text{sigmoid}(\boldsymbol{\theta}^\top \mathbf{x})}$$

For small $\boldsymbol{\theta}^\top \mathbf{x}$, the label is very likely to be -1 .



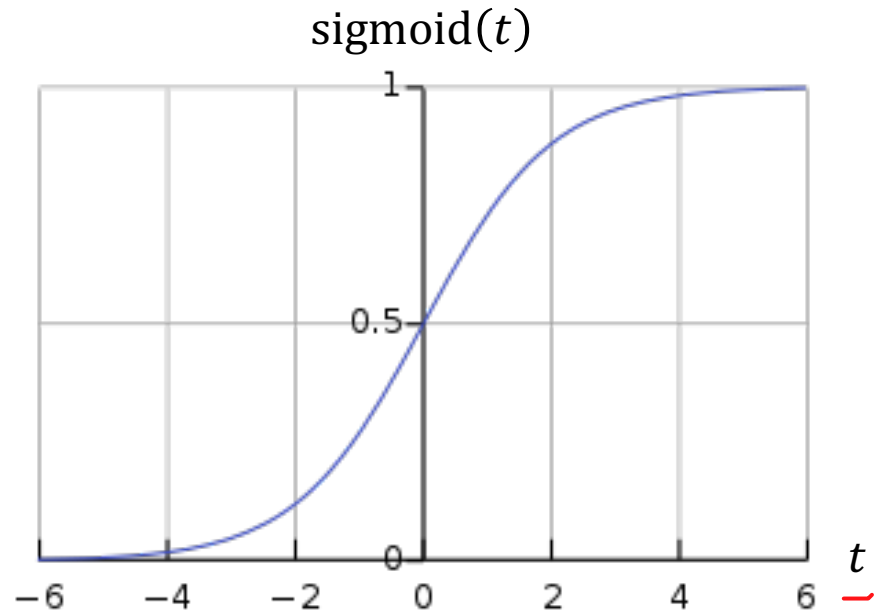
For large $\boldsymbol{\theta}^\top \mathbf{x}$, the label is very likely to be $+1$.

Sigmoid function

$$\text{sigmoid}(t) = \frac{1}{1 + e^{-t}}$$

$$\text{sigmoid}: \mathbb{R} \rightarrow [0, 1]$$

Sometimes also known as the **logistic** function.



Super useful formula

$$\text{sigmoid}(-t) = \frac{1}{1+e^t} = \frac{e^{-t}}{e^{-t}+1} = 1 - \frac{1}{e^{-t}+1} = 1 - \text{sigmoid}(t)$$

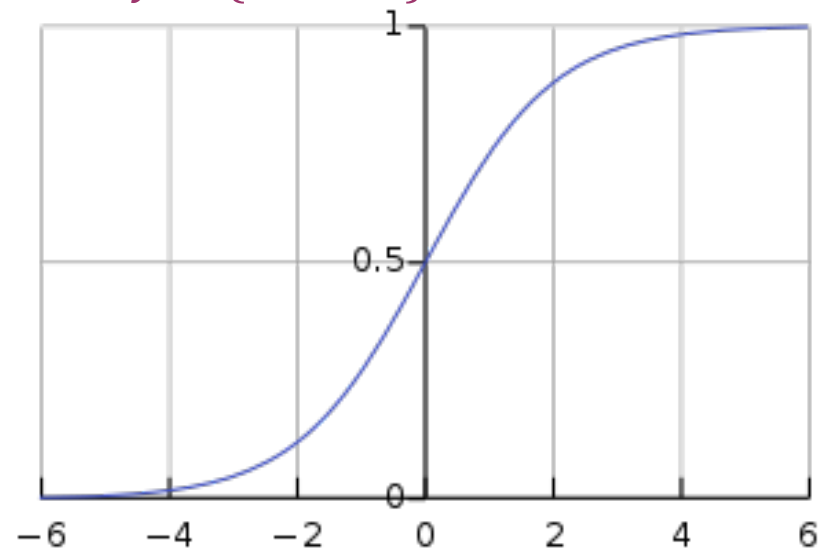
Label Probabilities

$$\mathbb{P}(y = +1 | \mathbf{x}) = \text{sigmoid}(\boldsymbol{\theta}^\top \mathbf{x}) = \text{sigmoid}(y(\boldsymbol{\theta}^\top \mathbf{x}))$$

To get the other label probability,

$$\begin{aligned}\mathbb{P}(y = -1 | \mathbf{x}) &= 1 - \mathbb{P}(y = +1 | \mathbf{x}) \\ &= 1 - \text{sigmoid}(\boldsymbol{\theta}^\top \mathbf{x}) \\ &= \text{sigmoid}(-\boldsymbol{\theta}^\top \mathbf{x}) = \text{sigmoid}(y(\boldsymbol{\theta}^\top \mathbf{x}))\end{aligned}$$

Thus, $\mathbb{P}(y|\mathbf{x}) = \text{sigmoid}(y(\boldsymbol{\theta}^\top \mathbf{x}))$ for both $y \in \{+1, -1\}$.



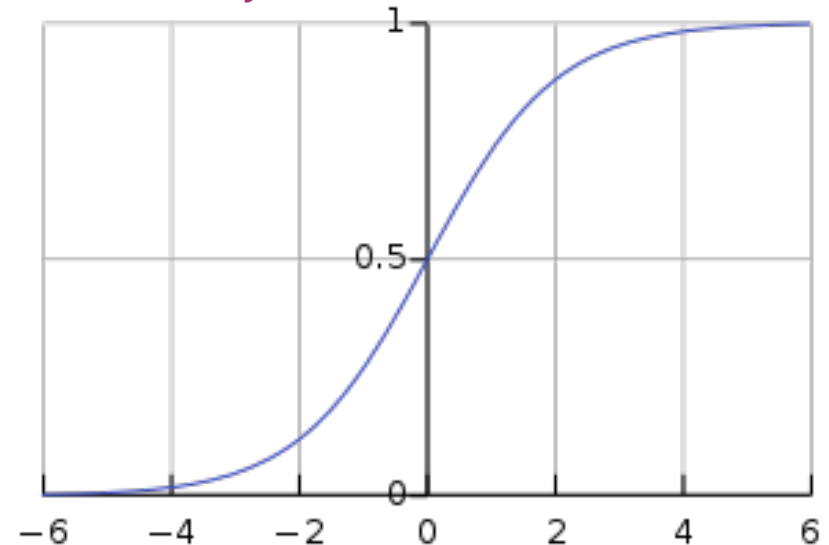
Label Predictions

Find out which label is more probable.

$$\mathbb{P}(y = +1 \mid \mathbf{x}) \geq \mathbb{P}(y = -1 \mid \mathbf{x}) \iff \underline{h(\mathbf{x}; \boldsymbol{\theta}) \geq \frac{1}{2}}$$

If $h(\mathbf{x}; \boldsymbol{\theta}) \geq \frac{1}{2}$, then we predict the label of \mathbf{x} is $y = +1$.

If $h(\mathbf{x}; \boldsymbol{\theta}) < \frac{1}{2}$, then we predict the label of \mathbf{x} is $y = -1$.

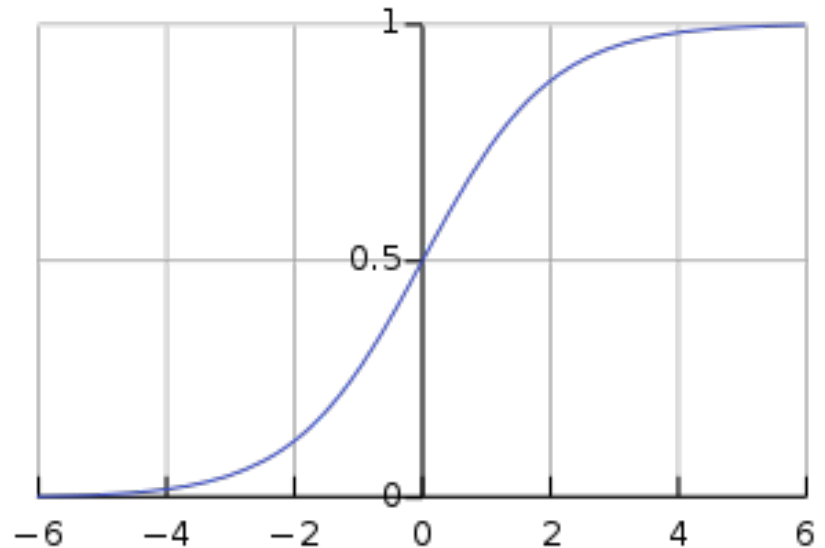
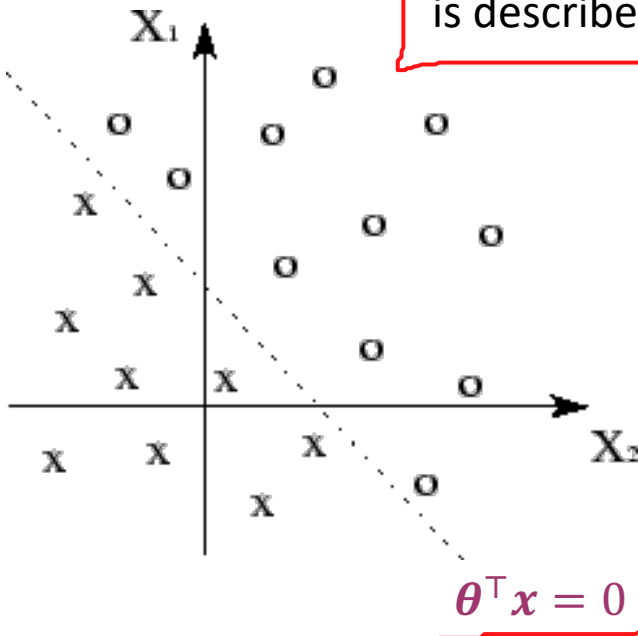


Decision Boundary

$$h(\mathbf{x}; \boldsymbol{\theta}) \geq \frac{1}{2} \iff \text{sigmoid}(\boldsymbol{\theta}^\top \mathbf{x}) \geq \frac{1}{2} \iff \boldsymbol{\theta}^\top \mathbf{x} \geq 0$$

$$h(\mathbf{x}; \boldsymbol{\theta}) < \frac{1}{2} \iff \text{sigmoid}(\boldsymbol{\theta}^\top \mathbf{x}) < \frac{1}{2} \iff \boldsymbol{\theta}^\top \mathbf{x} < 0$$

The decision boundary
is described by $\boldsymbol{\theta}^\top \mathbf{x} = 0$.



Likelihood

Probability of label y given feature x

$$\mathbb{P}(y|x) = \text{sigmoid}(y(\boldsymbol{\theta}^\top \mathbf{x})).$$

$L(\boldsymbol{\theta}; \mathcal{D})$ is called the
likelihood of the dataset \mathcal{D} .

Probability of labels y_1, \dots, y_N given features x_1, \dots, x_N

$$\begin{aligned} \underline{L(\boldsymbol{\theta}; \mathcal{D})} &= \underline{\mathbb{P}(y_1, \dots, y_N | x_1, \dots, x_N)} \\ &= \underline{\mathbb{P}(y_1 | x_1) \times \dots \times \mathbb{P}(y_N | x_N)} \\ &= \underline{\prod_{(x,y) \in \mathcal{D}} \mathbb{P}(y|x)} \end{aligned}$$

Maximizing $L(\boldsymbol{\theta}; \mathcal{D})$ is the same as **maximizing** $\log L(\boldsymbol{\theta}; \mathcal{D})$,
which is the same as **minimizing** $-\frac{1}{N} \log L(\boldsymbol{\theta}; \mathcal{D})$.

Logistic Loss

Minimize the training loss

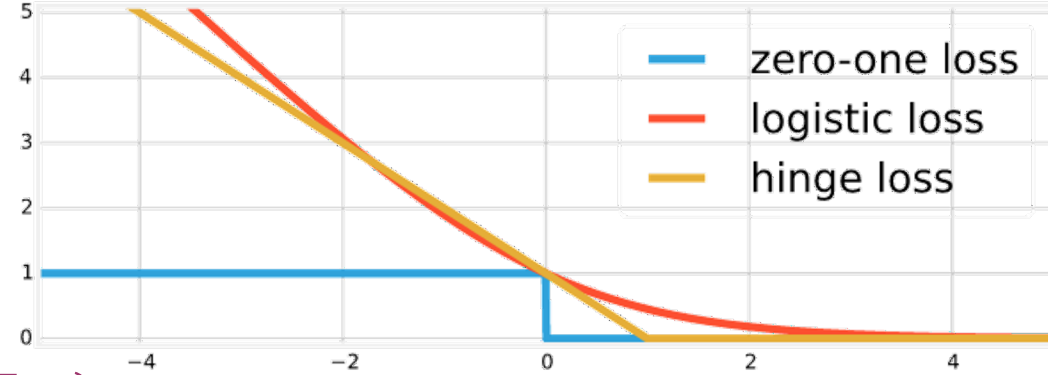
$$\begin{aligned}\underline{\mathcal{L}(\boldsymbol{\theta})} &= -\frac{1}{N} \log L(\boldsymbol{\theta}; \mathcal{D}) \\ &= -\frac{1}{N} \log \prod_{(x,y) \in \mathcal{D}} \mathbb{P}(y|\mathbf{x}) \\ &= -\frac{1}{N} \sum_{(x,y) \in \mathcal{D}} \log \mathbb{P}(y|\mathbf{x}) \\ &= -\frac{1}{N} \sum_{(x,y) \in \mathcal{D}} \log \frac{1}{1+e^{-y(\boldsymbol{\theta}^\top \mathbf{x})}} \\ &= \frac{1}{N} \sum_{(x,y) \in \mathcal{D}} \log(1 + e^{-y(\boldsymbol{\theta}^\top \mathbf{x})}) \\ &= \frac{1}{N} \sum_{(x,y) \in \mathcal{D}} \text{Loss}(y(\boldsymbol{\theta}^\top \mathbf{x}))\end{aligned}$$

$\text{Loss}(z) = \log(1 + e^{-z})$
is the logistic loss.

Loss Functions

Training Loss

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{(x,y) \in \mathcal{D}} \text{Loss}(y(\theta^\top x))$$



z

$$z = y(\theta^\top x)$$

Zero-One Loss

$$\text{Loss}_{01}(z) = \mathbb{I}[z \leq 0]$$

Perceptron Algorithm

Hinge Loss

$$\text{Loss}_H(z) = \max\{1 - z, 0\}$$

Hinge Loss Algorithm

Logistic Loss

$$\text{Loss}_L(z) = \log(1 + e^{-z})$$

Logistic Regression

Some folks use \log_2 instead of \log_e so that $\text{Loss}_L(0) = 1$.

Gradient

$$\text{Loss}_L(z) = \log(1 + e^{-z})$$

$$\nabla_z \text{Loss}_L(z) = \frac{-e^{-z}}{1+e^{-z}} = \frac{-1}{e^z+1} = -\text{sigmoid}(-z) = \text{sigmoid}(z) - 1$$

By chain rule, the gradient for 1 training sample $\nabla_{\theta} \mathcal{L}_1(\theta; \mathbf{x}, y)$ is

$$\begin{aligned} \nabla_{\theta} \text{Loss}_L(y(\theta^T \mathbf{x})) &= y \mathbf{x} (\text{sigmoid}(z) - 1) \\ &= \begin{cases} \mathbf{x}(h(\mathbf{x}; \theta) - 1) & \text{if } y = +1, \\ \mathbf{x}(h(\mathbf{x}; \theta) - 0) & \text{if } y = -1. \end{cases} \\ &= \mathbf{x} (h(\mathbf{x}; \theta) - \mathbb{I}[y = 1]) \end{aligned}$$

Training Gradient

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}; \mathcal{D}) &= \frac{1}{N} \sum_{(\mathbf{x}, y) \in \mathcal{D}} \nabla_{\boldsymbol{\theta}} \mathcal{L}_1(\boldsymbol{\theta}; \mathbf{x}, y) \\ &= \frac{1}{N} \sum_{(\mathbf{x}, y) \in \mathcal{D}} \mathbf{x} (h(\mathbf{x}; \boldsymbol{\theta}) - \mathbb{I}[y = 1])\end{aligned}$$

Compare this with the training gradient for least squares

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}; \mathcal{D}) = \frac{1}{N} \sum_{(\mathbf{x}, y) \in \mathcal{D}} \mathbf{x} (f(\mathbf{x}; \boldsymbol{\theta}) - y)$$

In regression, we have:

$$L(\boldsymbol{\theta}, \mathbf{X}, \mathbf{y}) = \frac{1}{N} \sum_{n=1}^N (y_n - f(\mathbf{x}_n, \boldsymbol{\theta}))^2$$

$f(\mathbf{x}_n, \boldsymbol{\theta}) = \boldsymbol{\theta}^T \mathbf{x}_n$

Gradient Descent

1. Initialize θ randomly.
2. Update $\theta \leftarrow \theta - \frac{\eta_k}{N} \sum_{(x,y) \in \mathcal{D}} \mathbf{x}(h(\mathbf{x}; \theta) - \mathbb{I}[y = 1])$
3. Repeat (2) until convergence.
(e.g. when improvement in $\mathcal{L}(\theta; \mathcal{D})$ is small enough)

Logistic Regression

1. Training Set (Not Necessarily Linearly Separable)

$$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)$$

2. Model (Set of Sigmoid Neurons)

$$h(\mathbf{x}; \boldsymbol{\theta}) = \text{sigmoid}(\theta_0 x_0 + \theta_1 x_1 + \dots + \theta_D x_D)$$

3. Training Loss (Logistic Loss)

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{(x,y) \in \mathcal{D}} \log(1 + e^{-y(\boldsymbol{\theta}^\top \mathbf{x})})$$

3. Algorithm (Gradient Descent)

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \frac{\eta_k}{N} \sum_{(x,y) \in \mathcal{D}} \mathbf{x} (h(\mathbf{x}; \boldsymbol{\theta}) - \mathbb{I}[y = 1])$$

Regularized Logistic Regression

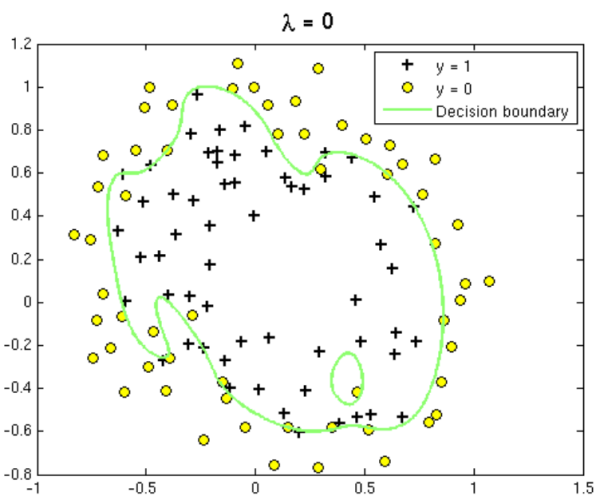
- When your data has a high-dimensional feature, or your training set is small, you might have the over-fitting problem.

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{(x,y) \in \mathcal{D}} \log(1 + e^{-y(\boldsymbol{\theta}^\top x)}) + \frac{\lambda}{2N} \sum_{j=1}^D \theta_j^2$$

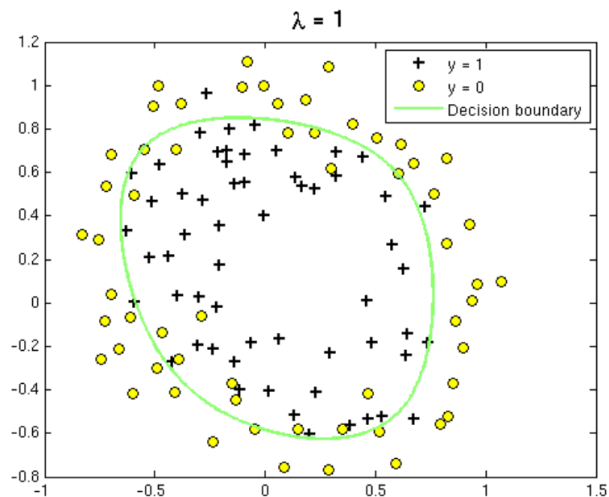
- We are doing regularization on $\theta_1, \theta_2, \dots, \theta_D$
- When using gradient descent, we have

$$\begin{aligned} \underline{\theta_0} &\leftarrow \theta_0 - \frac{\eta_k}{N} \sum_{(x,y) \in \mathcal{D}} x^{(0)} (h(\mathbf{x}; \boldsymbol{\theta}) - \mathbb{I}[y = 1]) \\ \underline{\theta_j} &\leftarrow \theta_j - \frac{\eta_k}{N} \left[\sum_{(x,y) \in \mathcal{D}} x^{(j)} (h(\mathbf{x}; \boldsymbol{\theta}) - \mathbb{I}[y = 1]) + \underline{\lambda \theta_j} \right] \\ &\quad j = 1, 2, \dots, D \end{aligned}$$

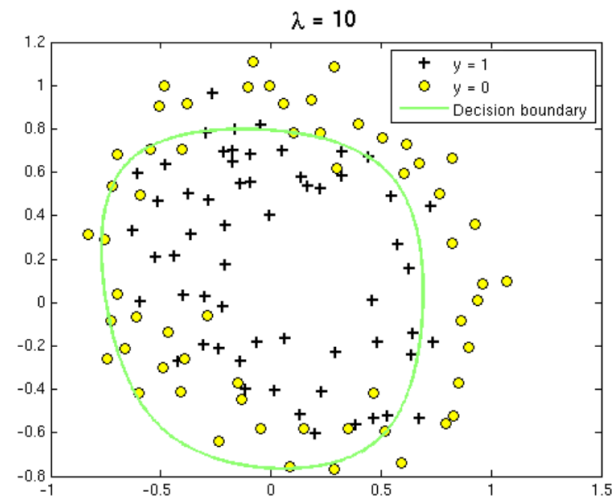
Regularized Logistic Regression



overfitting



relatively good



underfitting

Lots of research in classification architecture / loss function design

- Softmax loss (softmax cross-entropy loss)
- **AM-Softmax** F. Wang, J. Cheng, W. Liu, and H. Liu. Additive margin softmax for face verification. IEEE Signal Processing Letters, 2018.
- **NormFace** F. Wang, X. Xiang, J. Cheng, and A. L. Yuille. Normface: L2 hypersphere embedding for face verification. ACM MM, 2017
- **Triplet loss with hard mining** F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. CVPR 2015
- **Lifted structure loss** H. Oh Song, Y. Xiang, S. Jegelka, and S. Savarese. Deep metric learning via lifted structured feature embedding. CVPR 2016
- etc
- **Circle Loss** Sun, Yifan, et al. "Circle loss: A unified perspective of pair similarity optimization." CVPR 2020.