Linear Classification

Zhiyao Duan

Associate Professor of ECE and CS

University of Rochester

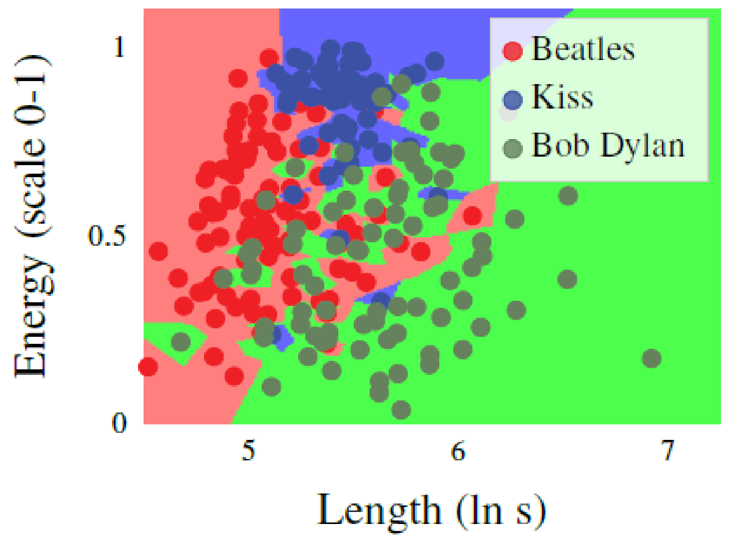
Some figures are copied from the following books

- **LWLS** - Andreas Lindholm, Niklas Wahlström, Fredrik Lindsten, Thomas B. Schön, *Machine Learning: A First Course for Engineers and Scientists*, Cambridge University Press, 2022.
- **Mitchell** - Tom M. Mitchell, *Machine Learning*, McGraw-Hill Education, 1997.

Classification Boundary

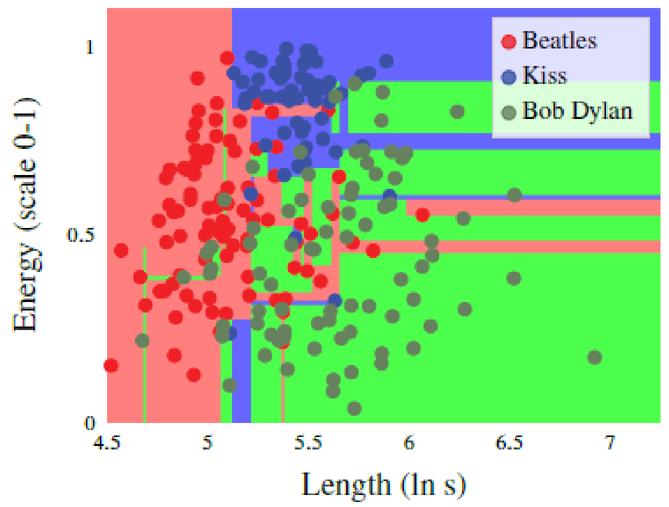
- Classification: given training examples $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$, learn $f\colon \boldsymbol{x} \mapsto y$, where y is categorical
- If $\boldsymbol{x}$ is numerical and has d attributes, then it is a d-dimension vector

Nearest Neighbor



(Fig. 2.5(a) in LWLS)

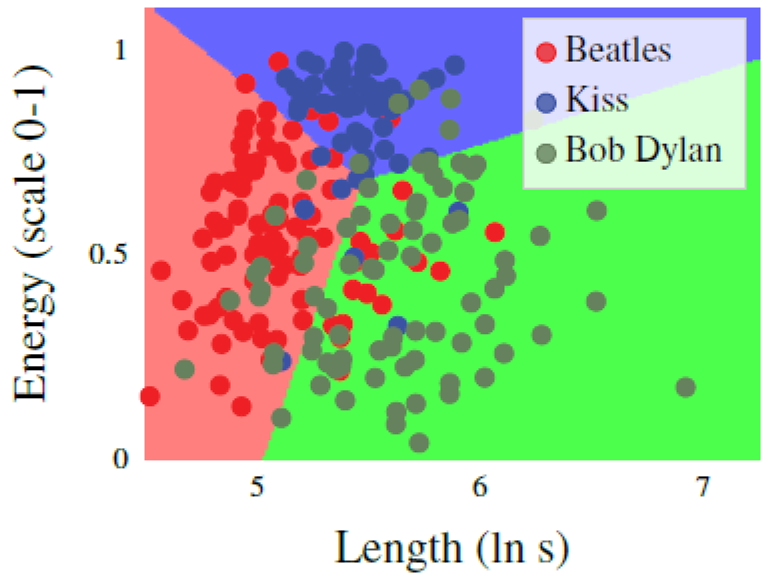
Decision Tree (fully grown)



(Fig. 2.11(a) in LWLS)

Linear Classification Boundary

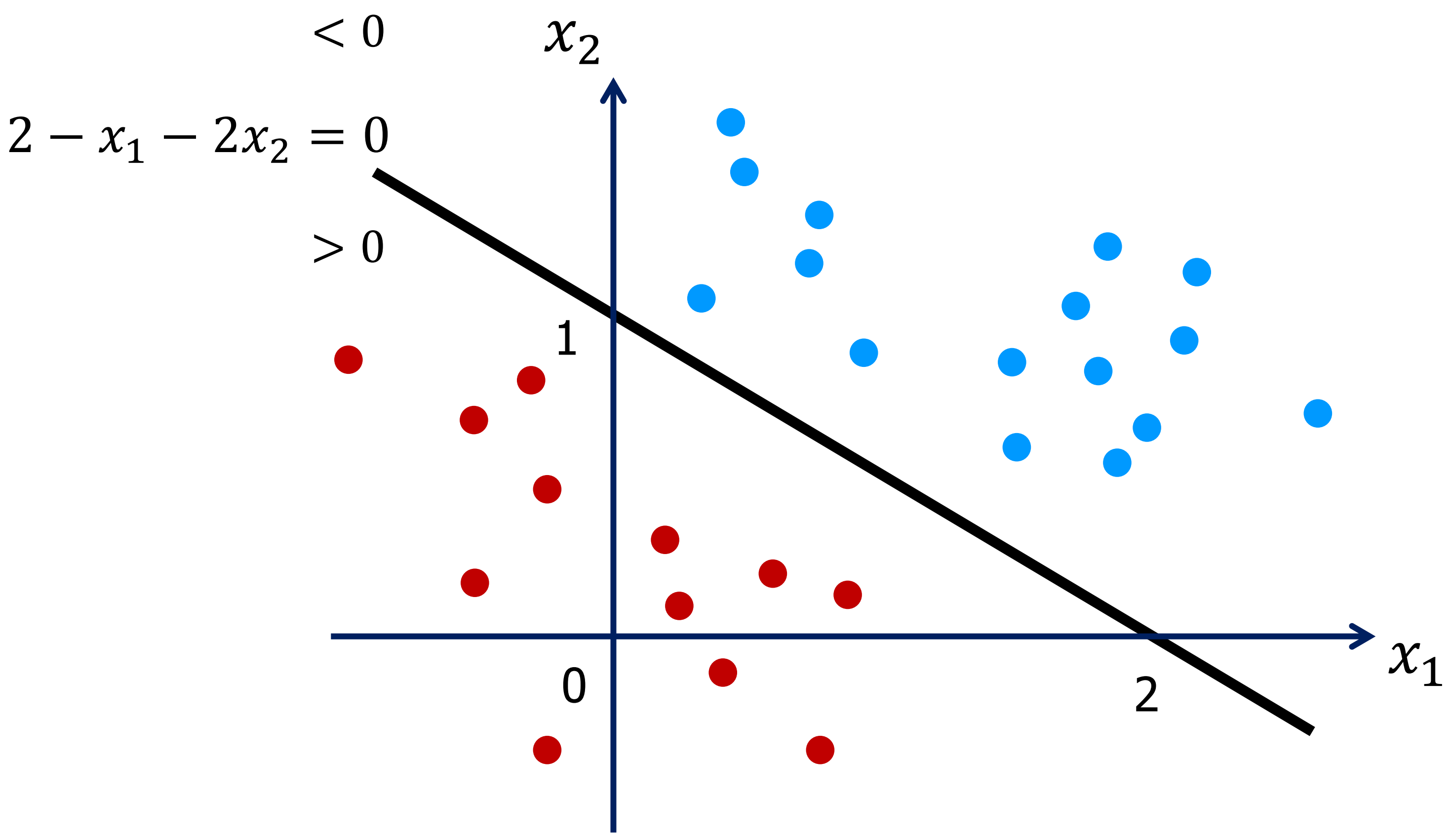
Logistic Regression



(Fig. 3.6 in LWLS)

Note: Logistic regression is in fact a classification method, not a regression method

Let's draw a linear boundary



- A linear boundary is a hyperplane, represented as a linear equation

$$w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_d x_d = 0$$

- It separates the space into two half spaces

Linear Boundary

- A linear boundary is a linear equation:

$$w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_d x_d = 0$$

$$\begin{bmatrix} w_0 & w_1 & \dots & w_d \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix} = 0, \text{ or } \boldsymbol{w}^T \begin{bmatrix} 1 \\ \boldsymbol{x} \end{bmatrix} = 0$$

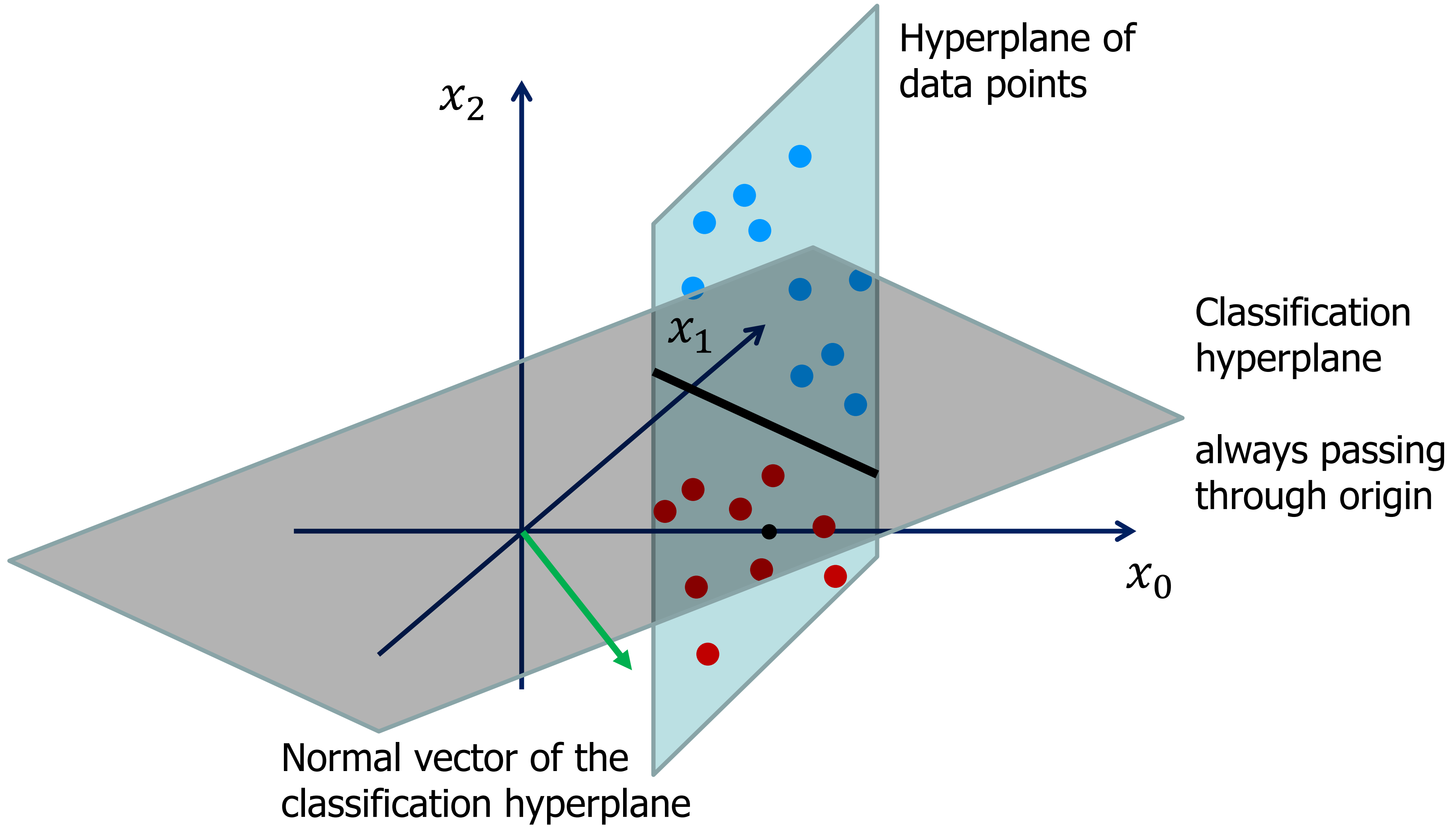
- $\boldsymbol{w}$ is the weight vector
- $\begin{bmatrix} 1 \\ \boldsymbol{x} \end{bmatrix}$ is the extended feature vector ($d + 1$ dimensional)

- **For convenience, we will view x as the extended feature vector**

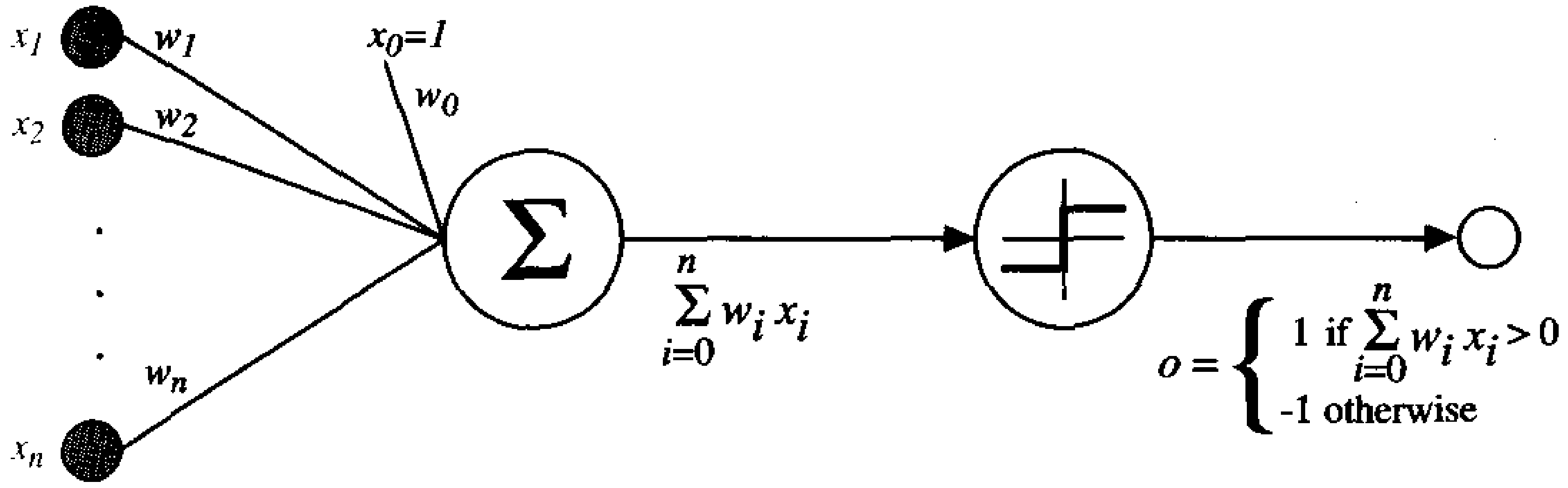
$$\boldsymbol{x} = (x_0, x_1, \dots, x_d)^T \text{ where } x_0 = 1$$

- So $\boldsymbol{w}^T \boldsymbol{x} = 0$ on the boundary (hyperplane), $\boldsymbol{w}$ is the normal vector
- $\boldsymbol{w}^T \boldsymbol{x} > 0$ on one side and $\boldsymbol{w}^T \boldsymbol{x} < 0$ on the other side

Geometry of the Extended Space


Hyperplane of data points
x_2
x_1
Classification hyperplane
always passing through origin
x_0
Normal vector of the classification hyperplane

Perceptron


x_1
x_2
x_n
w_1
w_2
w_n
$x_0=1$
w_0
Σ
$\sum_{i=0}^{n} w_i x_i$
$o = \begin{cases} 1 \text{ if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 \text{ otherwise} \end{cases}$

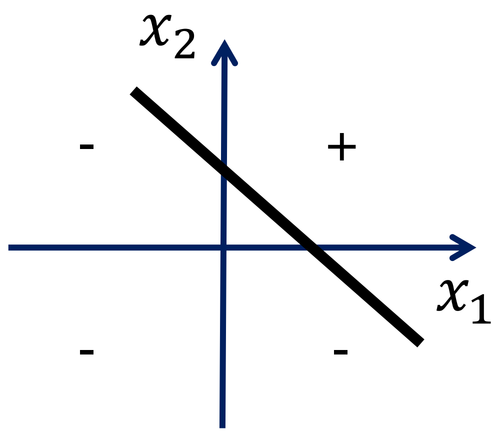
- Perceptron is a function that maps $\boldsymbol{x}$ to either 1 or -1

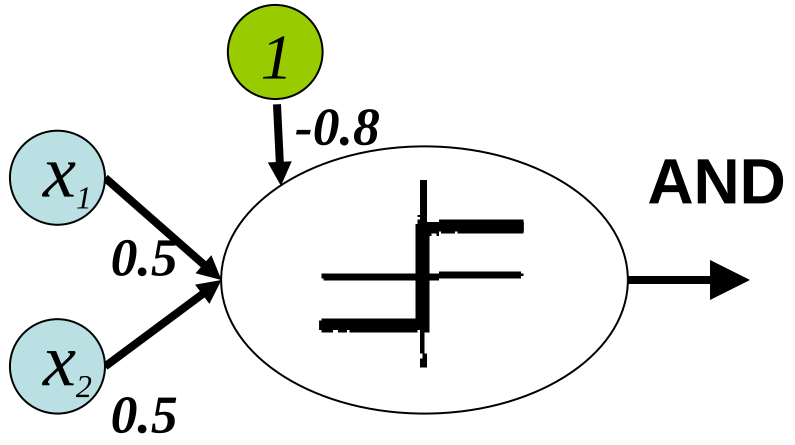
$$f(\boldsymbol{x}) = sign(\boldsymbol{w}^T \boldsymbol{x})$$

- This is a linear binary classifier where one side of the classification boundary is 1 and the other side is -1

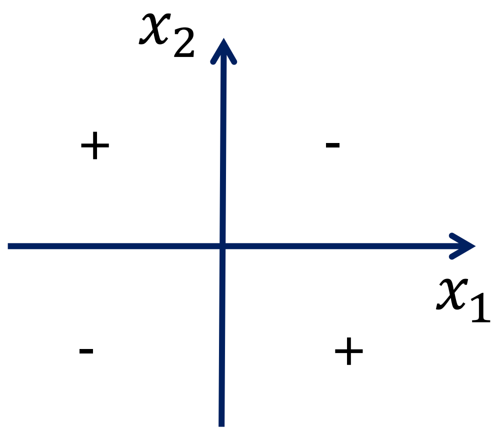
Perceptron's Representation Power

- Basic logic operations: AND, OR, NOT
  - They are all linearly separable





- How about XOR?
  - Not linearly separable



How to Learn a Perceptron?

- Perceptron is a function mapping $\boldsymbol{x}$ to either 1 or -1

$$f(\boldsymbol{x}) = sign(\boldsymbol{w}^T\boldsymbol{x})$$

- How to learn this function, i.e., the weights $\boldsymbol{w}$?

- One idea is to start from an initial vector $\boldsymbol{w}$, and then iteratively update its value till some condition

- This is an optimization procedure
  - Initialization: **Random?**
  - Iterative updates: **How?**
  - Termination condition: **What?**

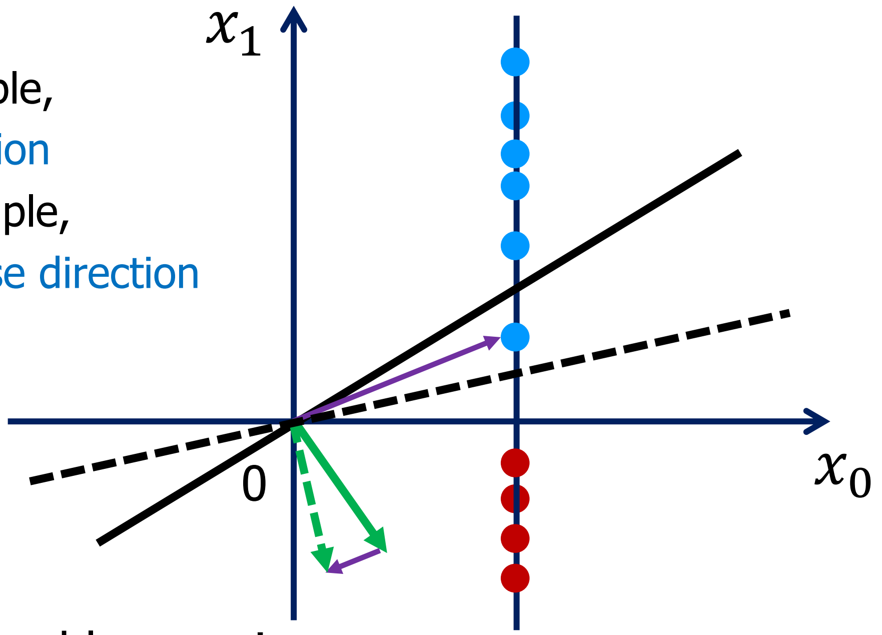
Perceptron Update Rule

- Initialization: **Random**
- Iterative updates: **Update w for each misclassified training example**
- Termination condition: **Till all examples are correctly classified**
  - Would not terminate if training examples are not linearly separable

- Idea
  - For a misclassified positive example, update w along the example's direction
  - For a misclassified negative example, update w along the example's reverse direction

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta \left(\mathrm{y}^{(\mathrm{i})} - f\left(\boldsymbol{x}^{(\mathrm{i})}\right) \right) \boldsymbol{x}^{(\mathrm{i})}$$

Step size $\eta > 0$, usually small



- Always converges for linearly separable cases!

How about non-linearly separable cases?

- Linear classification may still be desirable even if it misclassifies some training examples

- The perceptron update rule does not work in this case! Why?
  - Each update considers only one misclassified training example.

- How about we consider all misclassified training example together for each update?

- One thought: perhaps update $\boldsymbol{w}$ to reduce the number of misclassifications?
  - Doesn't work because this quantity is piecewise constant!

Perceptron Criterion Function

- Minimize perceptron criterion function

$$E(\boldsymbol{w}) = \sum_{i \in misclassified} -\boldsymbol{w}^T \boldsymbol{x}^{(i)} \underbrace{\left(\mathrm{y}^{(i)} - f\left(\boldsymbol{x}^{(i)}\right)\right)}_{\text{2 (pos example) or -2 (neg example)}}$$

- Computer gradient w.r.t. $\boldsymbol{w}$

$$\nabla_{\boldsymbol{w}} E(\boldsymbol{w}) = \sum_{i \in misclassified} -\boldsymbol{x}^{(i)} \left(\mathrm{y}^{(i)} - f\left(\boldsymbol{x}^{(i)}\right)\right)$$

- Update rule (via gradient descent)

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta \sum_{i \in misclassified} \boldsymbol{x}^{(i)} \left(\mathrm{y}^{(i)} - f\left(\boldsymbol{x}^{(i)}\right)\right)$$

- Terminate till classification error is acceptable

Comparing the Two Update Rules

- Perceptron update rule

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta \left(\mathrm{y}^{(i)} - f\left(\boldsymbol{x}^{(i)}\right)\right) \boldsymbol{x}^{(i)}, \quad \forall i \in misclassified$$

- Perceptron criterion function update rule

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta \sum_{i \in misclassified} \boldsymbol{x}^{(i)} \left(\mathrm{y}^{(i)} - f\left(\boldsymbol{x}^{(i)}\right)\right)$$

- The former can be viewed as a stochastic approximation of the true gradient of the perception criterion function
  - Also called stochastic gradient descent
- Using one example to approximate the gradient may be too stochastic. Let's use a few examples instead
  - Batch processing:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta \sum_{i \in a\ subset\ of\ misclassified} \boldsymbol{x}^{(i)} \left(\mathrm{y}^{(i)} - f\left(\boldsymbol{x}^{(i)}\right)\right)$$

Another Loss Function

- For non-linearly separable cases, the perceptron criterion update rule may still not converge if the termination condition is not set appropriately
  - Because correctly classified examples may be misclassified after update!

- How about we define a loss function that considers all training examples, including the correctly classified ones?
- We hope $\boldsymbol{w}^T\boldsymbol{x}$ is close to y for all training examples
- Let's use squared error (some definition divides it by N)

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{N}\left(y^{(i)} - \boldsymbol{w}^T\boldsymbol{x}^{(i)}\right)^2$$

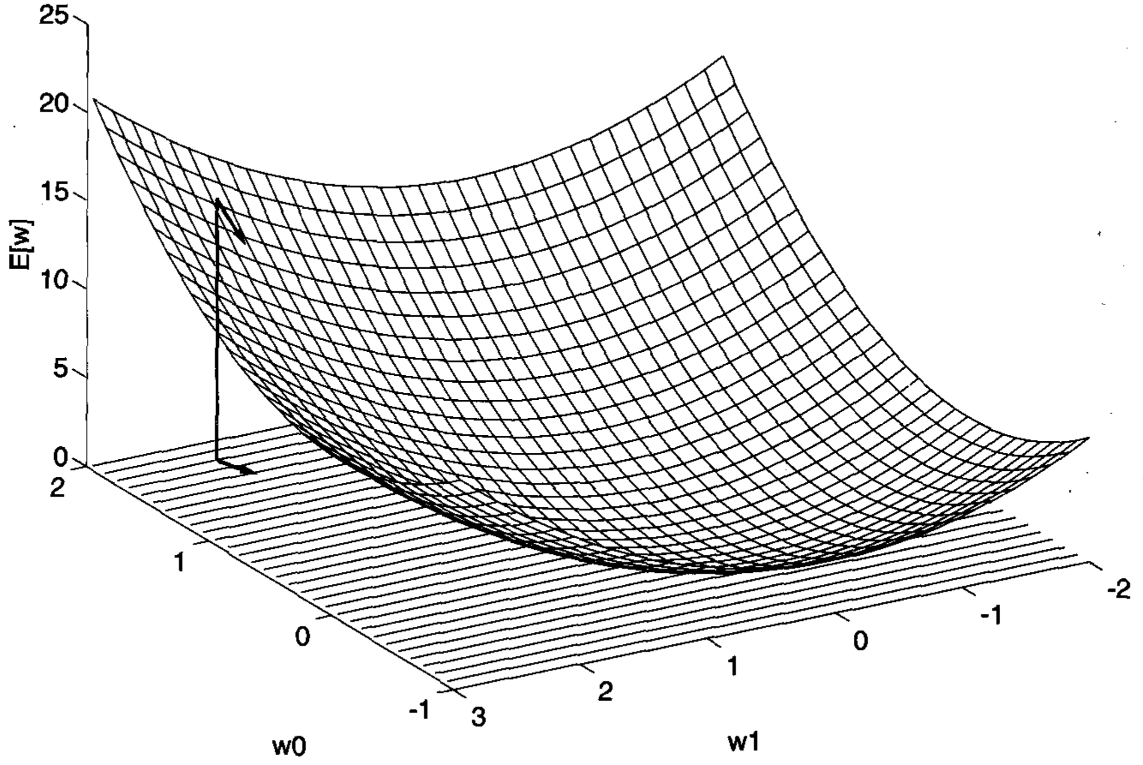
The Hypothesis Space

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{N}\left(y^{(i)} - \boldsymbol{w}^T\boldsymbol{x}^{(i)}\right)^2$$

- Quadratic function of $\boldsymbol{w}$
- Convex!
- Global minimum

(if it has a minimum)

- How to minimize $E(\boldsymbol{w})$?



(Fig. 4.4 in Mitchell)

Delta Rule

- Also called the LMS (least-mean-square) rule, Widrow-Hoff rule)
- Minimize square error loss

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{N}\left(y^{(i)} - \boldsymbol{w}^T\boldsymbol{x}^{(i)}\right)^2$$

- Computer gradient w.r.t. $\boldsymbol{w}$

$$\boldsymbol{\nabla}_{\boldsymbol{w}}E(\boldsymbol{w}) = \sum_{i=1}^{N} -\boldsymbol{x}^{(i)}\left(\mathrm{y}^{(i)} - \boldsymbol{w}^T\boldsymbol{x}^{(i)}\right)$$

- Update rule (via gradient descent)

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta\sum_{i=1}^{N}\boldsymbol{x}^{(i)}\left(\mathrm{y}^{(i)} - \boldsymbol{w}^T\boldsymbol{x}^{(i)}\right)$$

- Always converges, although may take a very long time

Delta Rule's Stochastic Version

- Similar to the perceptron update rule, we could approximate the true gradient of the squared error loss with the stochastic gradient computed from a single training example, then the update rule is:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta \boldsymbol{x}^{(i)}\left(\mathrm{y}^{(i)} - \boldsymbol{w}^T \boldsymbol{x}^{(i)}\right)$$

- We can also use a batch to have a better approximation

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta \sum_{i \in subset\ of\ training} \boldsymbol{x}^{(i)}\left(\mathrm{y}^{(i)} - \boldsymbol{w}^T \boldsymbol{x}^{(i)}\right)$$

Another Way to Minimize Squared Error

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{N}\left(y^{(i)} - \boldsymbol{w}^T\boldsymbol{x}^{(i)}\right)^2$$

$$= \frac{1}{2}\|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|^2$$

where $\boldsymbol{X} \in \mathbb{R}^{N\times(d+1)}$ is the extended training data matrix, $\boldsymbol{y} \in \{-1, 1\}^N$ is the label vector

- This is the approximation error (L2 norm) of the linear equation system

$$\boldsymbol{X}\boldsymbol{w} = \boldsymbol{y}$$

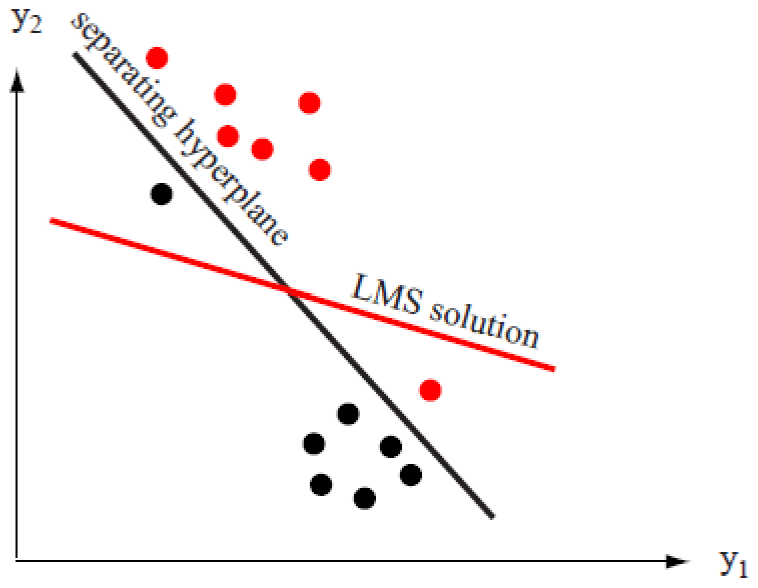
- This error is minimized by the least-square solution

$$\boldsymbol{w} = \underbrace{(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T}_{\text{Pseudo-inverse of } \boldsymbol{X}}\boldsymbol{y}$$

Remarks on Squared Error Minimization

- Delta rule is preferred over the least-square solution, because
  - $\boldsymbol{X}^T\boldsymbol{X}$ may be singular
  - It avoids large matrix inverse
  - Automatically copes with some computational problems due to roundoff or truncation

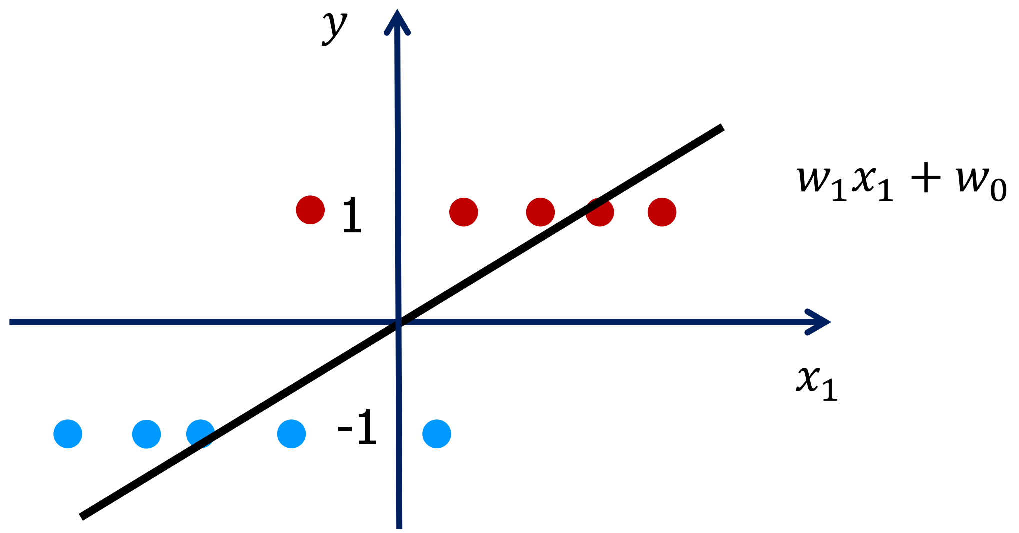
- The solution $\boldsymbol{w}$ found by minimizing the squared error may not be optimal in terms of minimizing the classification error among all linear classification methods, even for linearly separable cases!



(Fig. 5.17 in Duda, Hart & Stork, Pattern Classification, 2001, Wiley)

Geometric Interpretation of Squared Error

- Let's look at the 1d case

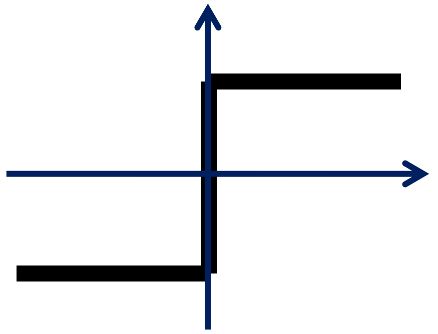


- $E(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{N}\left(y^{(i)} - \boldsymbol{w}^T\boldsymbol{x}^{(i)}\right)^2$ is fitting the training data using linear regression with squared error
- Correctly classified examples (points far from origin) may have a significant contribution to this loss!

Step Function → Smooth Function

- Perceptron

$$f(\boldsymbol{x}) = sign(\boldsymbol{w}^T\boldsymbol{x})$$



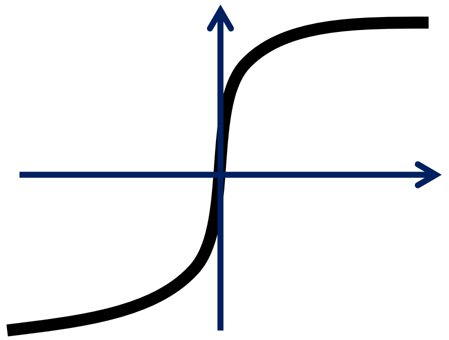
- Hyperbolic tangent activation

$$f(\boldsymbol{x}) = \tanh(\boldsymbol{w}^T\boldsymbol{x}) = \frac{1 - e^{-\boldsymbol{w}^T\boldsymbol{x}}}{1 + e^{-\boldsymbol{w}^T\boldsymbol{x}}}$$

- $\tanh()$ is a scaled logistic sigmoid function

$$\tanh(x) = 2\sigma(x) - 1$$

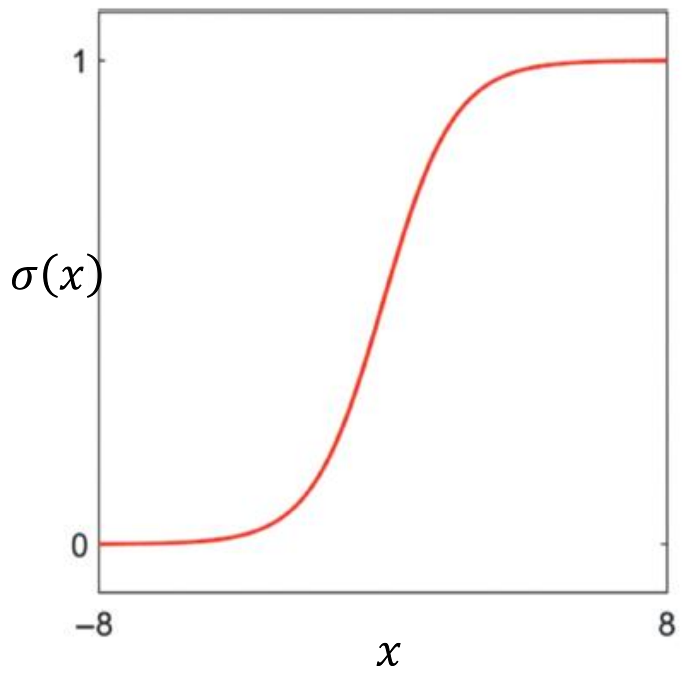
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

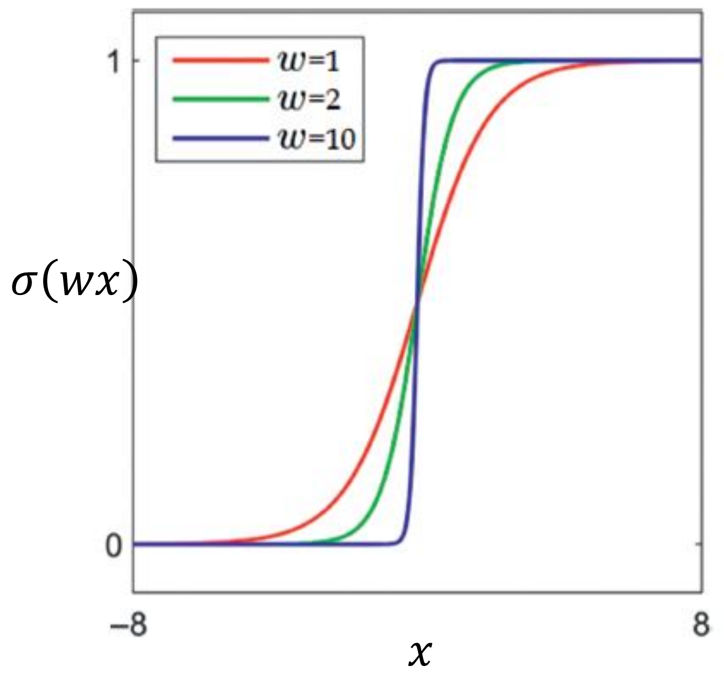


Logistic Sigmoid Function

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

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





- As $||\boldsymbol{w}||$ increases, it approximates the step function more closely

Logistic and Logit

- Logistic function

$$y = \sigma(x) = \frac{1}{1 + e^{-x}}, x \in \mathbb{R}$$

- Logit function: the inverse function

$$x = logit(y) = \log\left(\frac{y}{1 - y}\right), 0 < \mathrm{y} < 1$$

  - If y is viewed as probability of one class, then $1 - y$ is the probability of the other class
  - Logit is then the log of their ratio, or log of odds

- The word “logistic” is related to “logic”, and “logarithmic”

Logistic Regression with Squared Error

- Minimize squared error loss

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{N}\left(y^{(i)} - \tanh\left(\boldsymbol{w}^T\boldsymbol{x}^{(i)}\right)\right)^2$$

- This is essentially regression with squared error using the $\tanh(\cdot)$ function, which is a scaled logistic function; this is why it's called "logistic regression"

- This function is not convex w.r.t. $\boldsymbol{w}$, but we can still use gradient descent to find a local optimum

$$\nabla E(\boldsymbol{w}) = \sum_{i=1}^{N}\left(y^{(i)} - \frac{2}{1+e^{-\boldsymbol{w}^T\boldsymbol{x}^{(i)}}} + 1\right)\frac{2}{\left(1+e^{-\boldsymbol{w}^T\boldsymbol{x}^{(i)}}\right)^2} e^{-\boldsymbol{w}^T\boldsymbol{x}^{(i)}}\left(-\boldsymbol{x}^{(i)}\right)$$

Logistic Regression with Squared Error

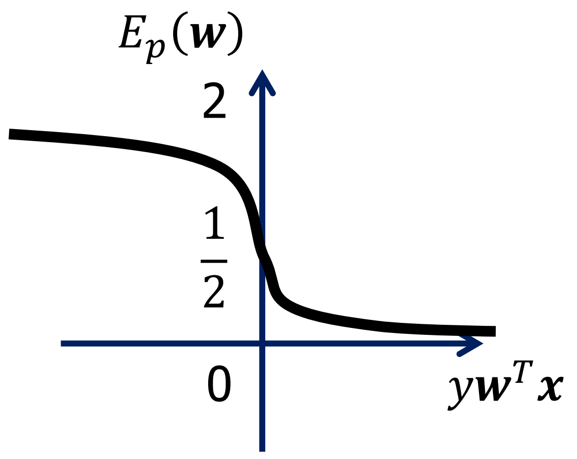
- Let's look at the error for each example

$$E_p(\boldsymbol{w}) = \frac{1}{2}(y - \tanh(\boldsymbol{w}^T\boldsymbol{x}))^2$$

$$= \frac{1}{2}\begin{cases}(1 - 2\sigma(\boldsymbol{w}^T\boldsymbol{x}) + 1)^2 & if\ y = +1\\(-1 - 2\sigma(\boldsymbol{w}^T\boldsymbol{x}) + 1)^2 & if\ y = -1\end{cases}$$

$$= 2\begin{cases}\sigma^2(-\boldsymbol{w}^T\boldsymbol{x}) & if\ y = +1\\\sigma^2(\boldsymbol{w}^T\boldsymbol{x}) & if\ y = -1\end{cases}$$

$$= 2\sigma^2(-y\boldsymbol{w}^T\boldsymbol{x})$$



- Bigger loss if y and $\boldsymbol{w}^T\boldsymbol{x}$ have different signs
  - But perhaps not big enough for very wrong $\boldsymbol{w}^T\boldsymbol{x}$
- This is not convex for $y\boldsymbol{w}^T\boldsymbol{x}$
  - So not convex for $\boldsymbol{w}$

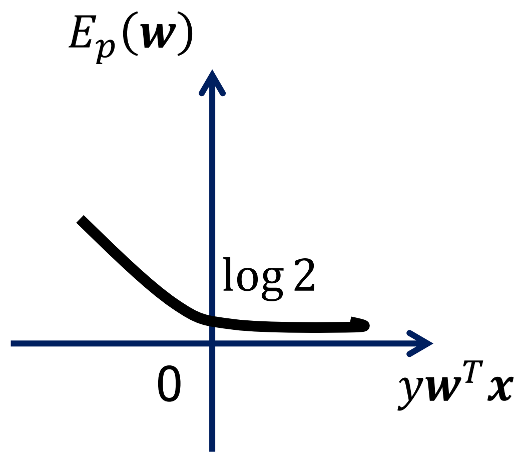
Logistic Regression with Log Error

- Define point-wise log error cost

$$E_p(\boldsymbol{w}) = \begin{cases} -\log\left(\sigma(\boldsymbol{w}^T\boldsymbol{x})\right) & if\ y = +1 \\ -\log\left(\sigma(-\boldsymbol{w}^T\boldsymbol{x})\right) & if\ y = -1 \end{cases}$$

$$= -\log\left(\sigma(y\boldsymbol{w}^T\boldsymbol{x})\right) = \log\left(1 + e^{-y\boldsymbol{w}^T\boldsymbol{x}}\right)$$

- Much bigger loss when y and $\boldsymbol{w}^T\boldsymbol{x}$ have different signs
- This is convex for $y\boldsymbol{w}^T\boldsymbol{x}$
  - So convex for $\boldsymbol{w}$



Logistic Regression with Log Error

- Sum all point-wise log error together → Softmax Loss

$$E(\boldsymbol{w}) = \sum_{i=1}^{N} \log\left(1 + e^{-y^{(i)}\boldsymbol{w}^T\boldsymbol{x}^{(i)}}\right)$$

- This is a convex function of $\boldsymbol{w}$!
- Compute gradient w.r.t. $\boldsymbol{w}$, then take gradient descent

$$\nabla E(\boldsymbol{w}) = -\sum_{i=1}^{N} \frac{e^{-y^{(i)}\boldsymbol{w}^T\boldsymbol{x}^{(i)}}}{1 + e^{-y^{(i)}\boldsymbol{w}^T\boldsymbol{x}^{(i)}}} y^{(i)}\boldsymbol{x}^{(i)}$$

- Or compute Hessian and then use Newton's method

$$\nabla^2 E(\boldsymbol{w}) = \sum_{i=1}^{N} \sigma\left(y^{(i)}\boldsymbol{w}^T\boldsymbol{x}^{(i)}\right)\left(\mathbf{1} - \sigma\left(y^{(i)}\boldsymbol{w}^T\boldsymbol{x}^{(i)}\right)\right)\boldsymbol{x}^{(i)}{\boldsymbol{x}^{(i)}}^{\mathrm{T}}$$

Why Called Softmax?

- Without loss of generality, assume $\mathrm{a} < \mathrm{b}$

$$\begin{aligned}\max\{a,b\} &= b\\ &= a + (b-a) = \log e^{a} + \log e^{b-a}\\ &< \log e^{a} + \log\left(1 + e^{b-a}\right)\\ &= \log\left(e^{a} + e^{b}\right)\\ &\doteq softmax\{a,b\}\end{aligned}$$

- Single-sample log error cost

$$\begin{aligned}\log\left(1 + e^{-y^{(i)}\boldsymbol{w}^{T}\boldsymbol{x}^{(i)}}\right) &= \log\left(e^{0} + e^{-y^{(i)}\boldsymbol{w}^{T}\boldsymbol{x}^{(i)}}\right)\\ &= softmax\left\{0, -y^{(i)}\boldsymbol{w}^{T}\boldsymbol{x}^{(i)}\right\}\end{aligned}$$

- The log error cost is thus the softmax between 0 and $-y^{(i)}\boldsymbol{w}^{T}\boldsymbol{x}^{(i)}$

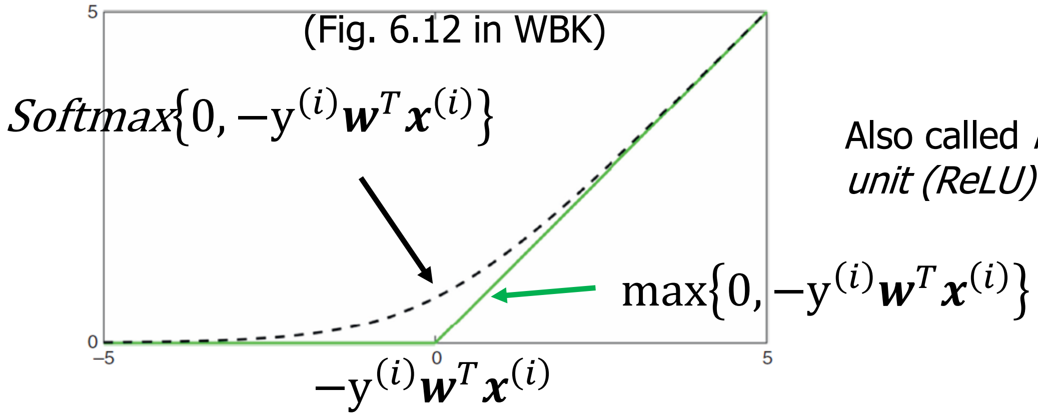
Softmax vs. Max

- Remember perceptron criterion function

$$E(\boldsymbol{w}) = \sum_{i \in misclassified} -\boldsymbol{w}^T \boldsymbol{x}^{(i)} \left(\mathrm{y}^{(i)} - f\left(\boldsymbol{x}^{(i)}\right)\right)$$

$$= \sum_{i \in misclassified} -2\mathrm{y}^{(i)} \boldsymbol{w}^T \boldsymbol{x}^{(i)} > 0$$

- For correctly classified examples, the loss is 0
- Therefore, its single-sample loss is $\max\{0, -2\mathrm{y}^{(i)} \boldsymbol{w}^T \boldsymbol{x}^{(i)}\}$
  - If we neglect the scale 2, then it's $\max\{0, -\mathrm{y}^{(i)} \boldsymbol{w}^T \boldsymbol{x}^{(i)}\}$



Also called *rectified linear unit (ReLU)* or *hinge loss*

Comparing Four Losses

- Perceptron criterion function (perceptron update rule)

$$E(\boldsymbol{w}) = \sum_{i \in misclassified} -\boldsymbol{w}^T \boldsymbol{x}^{(i)} \left(\mathrm{y}^{(i)} - f\left(\boldsymbol{x}^{(i)}\right)\right) = \sum_{i=1}^{N} \max\left\{0, -2\mathrm{y}^{(i)} \boldsymbol{w}^T \boldsymbol{x}^{(i)}\right\}$$

- Perceptron squared error (delta rule)

$$E(\boldsymbol{w}) = \frac{1}{2} \sum_{i=1}^{N} \left(y^{(i)} - \boldsymbol{w}^T \boldsymbol{x}^{(i)}\right)^2 = \frac{1}{2} \sum_{i=1}^{N} \left(1 - y^{(i)} \boldsymbol{w}^T \boldsymbol{x}^{(i)}\right)^2$$

- Logistic regression with squared error

$$E(\boldsymbol{w}) = \frac{1}{2} \sum_{i=1}^{N} \left(y^{(i)} - \tanh\left(\boldsymbol{w}^T \boldsymbol{x}^{(i)}\right)\right)^2 = 2 \sum_{i=1}^{N} \left(\frac{e^{-y^{(i)} \boldsymbol{w}^T \boldsymbol{x}^{(i)}}}{1 + e^{-y^{(i)} \boldsymbol{w}^T \boldsymbol{x}^{(i)}}}\right)^2$$

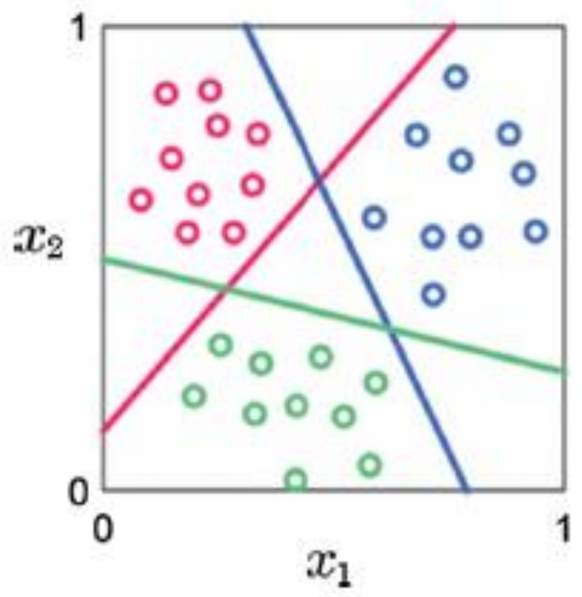
- Logistic regression with log error

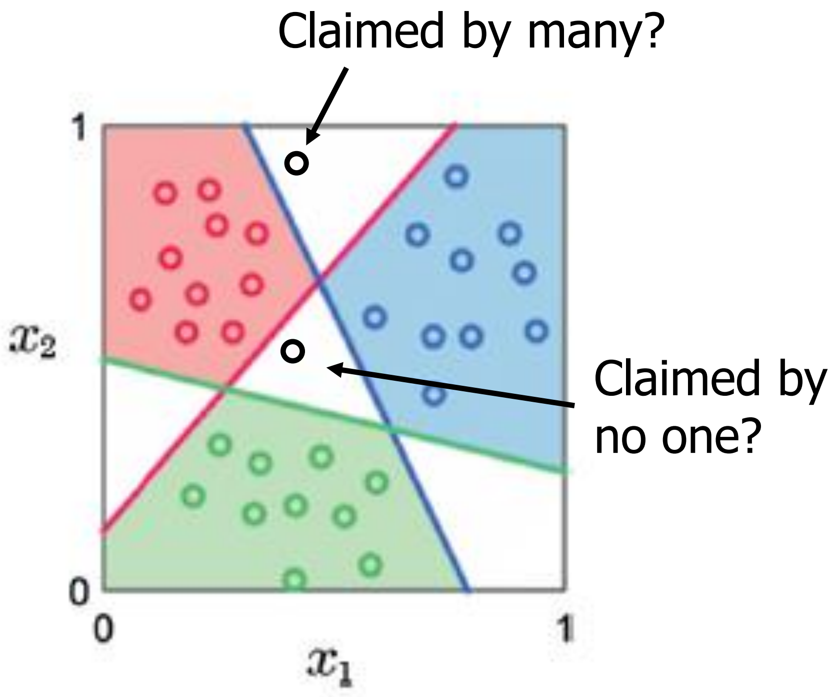
$$E(\boldsymbol{w}) = \sum_{i=1}^{N} \log\left(1 + e^{-y^{(i)} \boldsymbol{w}^T \boldsymbol{x}^{(i)}}\right)$$

- Let's draw them on whiteboard!

Multi-Class Classification

- How do we generalize binary linear classification to multiple (e.g., C) classes?
- First idea: train C one-versus-all classifiers



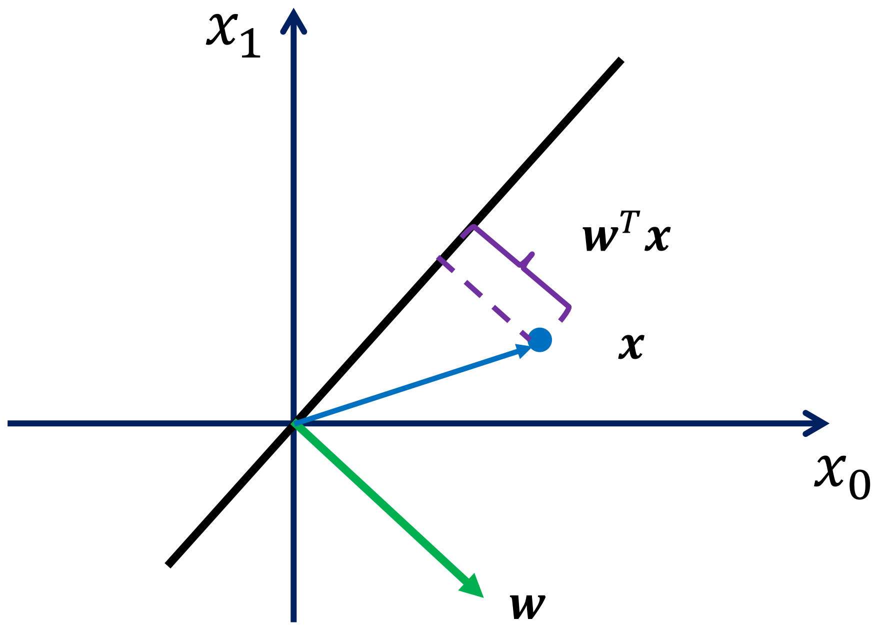


(Figs. 7.2 and 7.3 of WBK)

Distance from Classification Boundary

- If $\boldsymbol{x}$ is on the positive side
  - Distance is $\frac{\boldsymbol{w}^T\boldsymbol{x}}{\|\boldsymbol{w}\|_2}$
- If $\boldsymbol{x}$ is on the negative side
  - *Distance is* $-\frac{\boldsymbol{w}^T\boldsymbol{x}}{\|\boldsymbol{w}\|_2}$

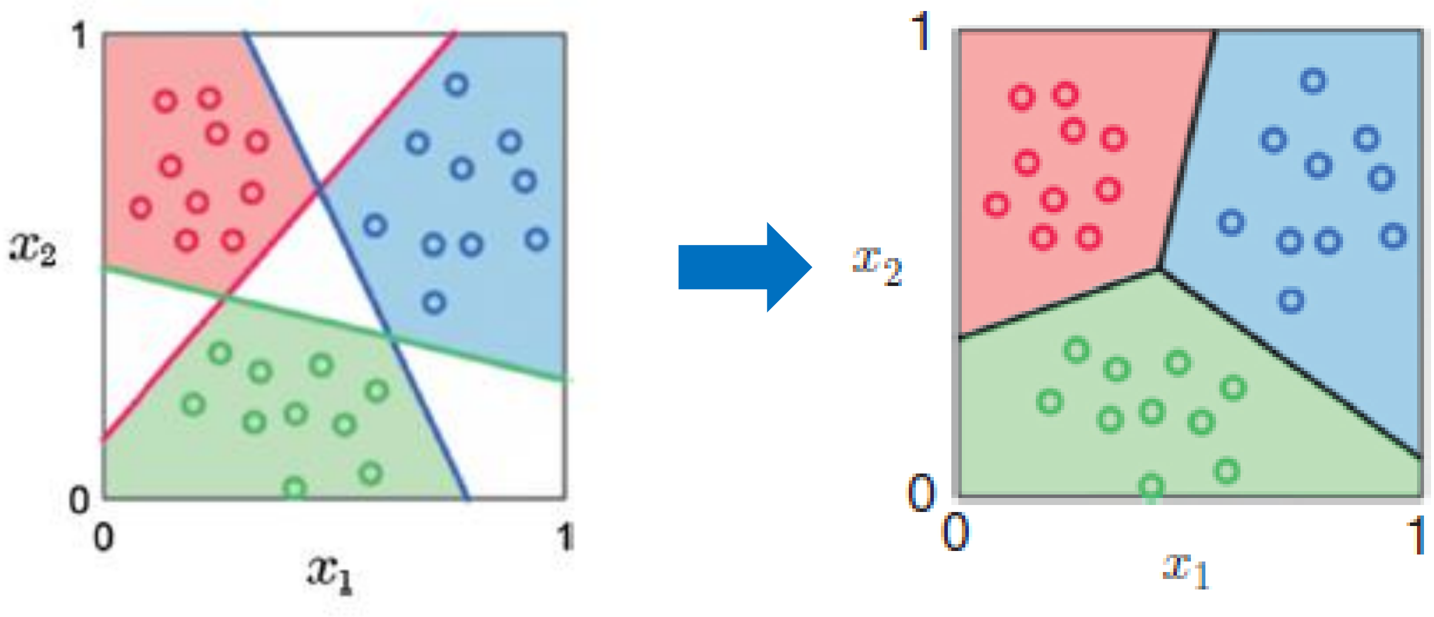
- If we assume $\boldsymbol{w}$ is normalized as $\|\boldsymbol{w}\|_2 = 1$, then the signed distance is simply $\boldsymbol{w}^T\boldsymbol{x}$
- This normalization can be achieved by regularization



Fusing all OvA Classifiers Together

- Output the class whose OvA classification boundary has the largest signed distance to the example

$$\hat{y} = \arg\max_{j=0,\cdots,C-1} \boldsymbol{w}_j^T \boldsymbol{x}, \qquad \text{s.t.} \ \|\boldsymbol{w}\|_2 = 1$$



Multi-Class Perceptron

- We hope that the signed distance between each training example and the binary classification boundary of its target class is the largest
- If it’s not, then this example is likely misclassified

- Based on this idea, we can define a loss function

$$E(\boldsymbol{w}_0, \cdots, \boldsymbol{w}_{C-1}) = \sum_{i=1}^{N} \left\{ \left(\max_{j=0,\cdots,C-1} \boldsymbol{w}_j^T \boldsymbol{x}^{(i)} \right) - \boldsymbol{w}_{y^{(i)}}^T \boldsymbol{x}^{(i)} \right\}$$

$$= \sum_{i=1}^{N} \left\{ \max_{\substack{j=0,\cdots,C-1 \\ j \neq y^{(i)}}} \left(0, \left(\boldsymbol{w}_j - \boldsymbol{w}_{y^{(i)}} \right)^T \boldsymbol{x}^{(i)} \right) \right\}$$

- This is similar to the binary Perceptron Criterion Function, convex but has non-differentiable points

Max → Softmax

- A new loss function

$$E(\boldsymbol{w}_0, \cdots, \boldsymbol{w}_{C-1}) = \sum_{i=1}^{N} \left\{ \left(\operatorname{softmax}_{j=0,\cdots,C-1} \boldsymbol{w}_j^T \boldsymbol{x}^{(i)} \right) - \boldsymbol{w}_{y^{(i)}}^T \boldsymbol{x}^{(i)} \right\}$$

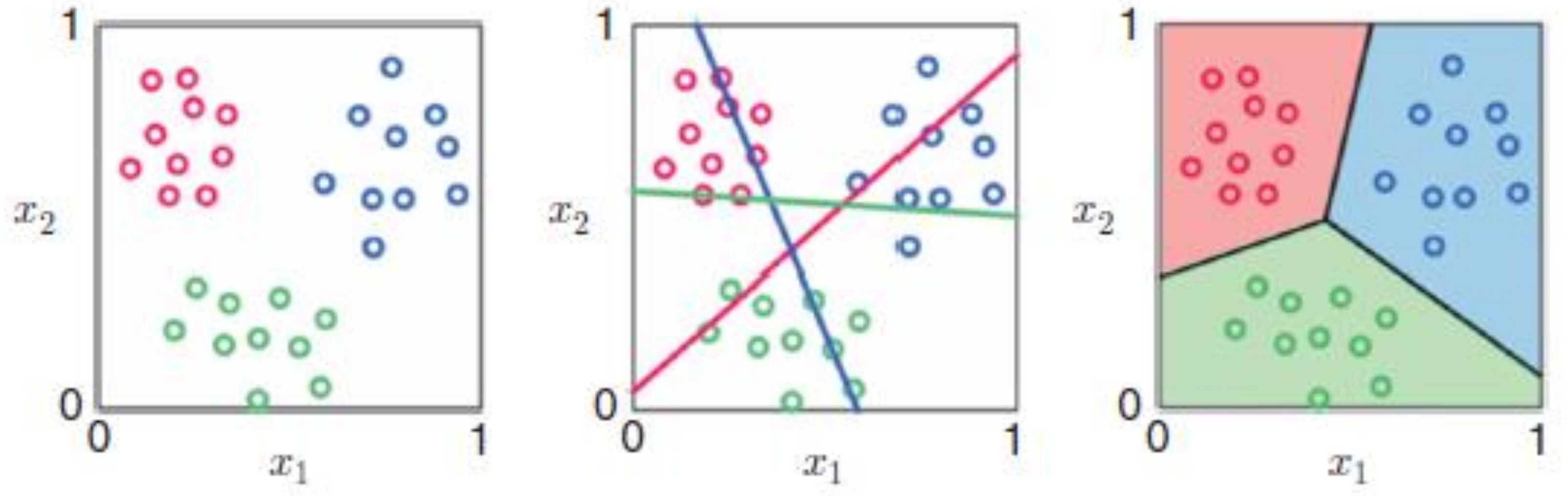
$$= \sum_{i=1}^{N} \left\{ \log \left(\sum_{j=0}^{C-1} e^{\boldsymbol{w}_j^T \boldsymbol{x}^{(i)}} \right) - \boldsymbol{w}_{y^{(i)}}^T \boldsymbol{x}^{(i)} \right\}$$

$$= -\sum_{i=1}^{N} \left\{ \log \left(\frac{e^{\boldsymbol{w}_{y^{(i)}}^T \boldsymbol{x}^{(i)}}}{\sum_{j=0}^{C-1} e^{\boldsymbol{w}_j^T \boldsymbol{x}^{(i)}}} \right) \right\}$$

- This is convex and smooth!

Classification Boundaries

- Multi-class perceptron learns all C binary classifiers simultaneously
  - Each single binary classifier may not be the best OvA classifier, but the fused classifier is usually better than the naïve fusion of C independently trained OvA classifiers



(Fig. 4.20 in WBK 1st ed.)

Summary

- Classification boundaries of linear classifiers are linear functions of the input features. They are hyperplanes.
- Perceptron uses a step function across the classification boundary. Its optimization has two main variants
  - Perceptron criterion function (max, hinge loss, ReLU) → perceptron update rule
  - Perceptron with squared error → delta rule
- Logistic regression uses hyperbolic tangent function to replace the step function. Its optimization has two main variants
  - Logistic regression with squared error (not convex)
  - Logistic regression with log error (softmax)
- Loss comparisons $E(\boldsymbol{w})$ vs. $-y\boldsymbol{w}^T\boldsymbol{x}$

Summary

- Multi-class classification
  - One-versus-All (OvA): train binary classifiers independently, then fuse based on signed distance from classification boundaries
  - Multi-class perceptron: train binary classifiers simultaneously; fusion is inside the model. Two variants of loss functions:
    - Max: extension of the binary-class perceptron criterion function (i.e., perceptron update rule)
    - Softmax: extension of the binary-class logistic regression with log error
    - The learned binary classifiers may not be the optimal OvA classifiers, but the learned multi-class classifier is usually better than the post fusion of independently trained OvA classifiers