Everything you've ever wanted to know about linear classifiers

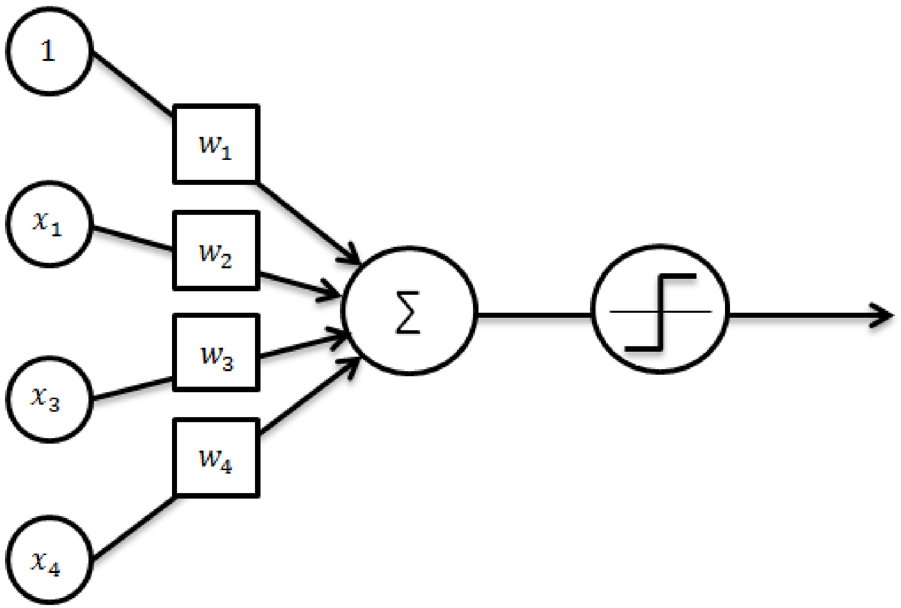


Image source

Outline

- Formalization of statistical learning of classifiers
- Ways to train linear classifiers
    1. Linear regression
    2. Perceptron training algorithm
    3. Logistic regression
    4. Support vector machines
- Gradient descent and stochastic gradient descent

Formalization

- Let’s focus on statistical learning of a *parametric model* in a supervised scenario

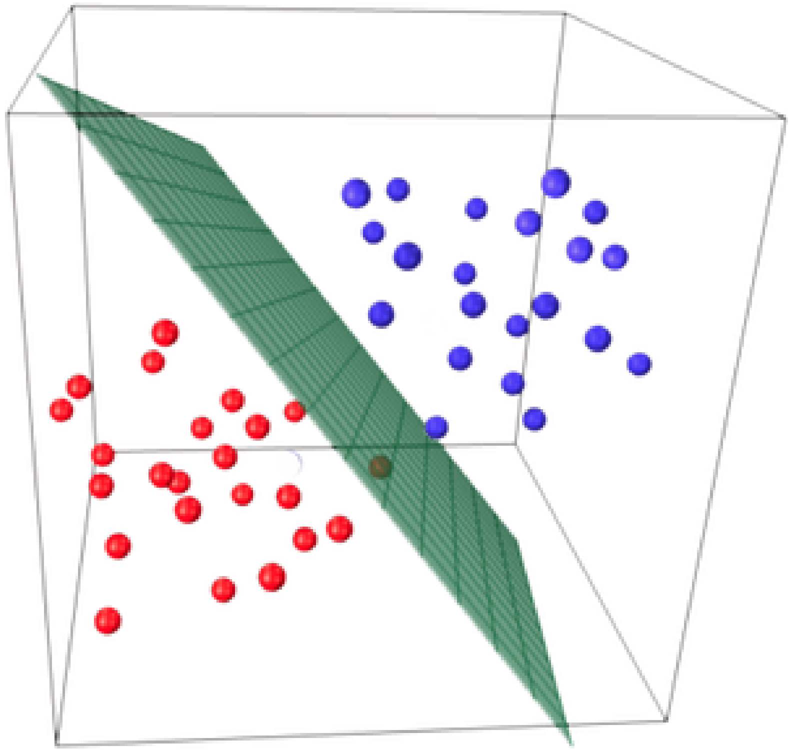


Image source

Formalization

- Given: training data $\{(x_i, y_i), i = 1, \dots, n\}$
- Find: predictor f
- Goal: make good predictions $\hat{y} = f(x)$ on *test* data

Source: Y. Liang

Formalization

- Given: training data $\{(x_i, y_i), i = 1, \ldots, n\}$
- Find: predictor f
- Goal: make good predictions $\hat{y} = f(x)$ on *test* data

What kinds of functions?

Source: [Y. Liang]

Formalization

- Given: training data $\{(x_i, y_i), i = 1, \ldots, n\}$
- Find: predictor $f \in \mathcal{H}$
- Goal: make good predictions $\hat{y} = f(x)$ on *test* data

Hypothesis class

The **hypothesis class** is the set of possible classification functions

Source: Y. Liang

Formalization

- Given: training data $\{(x_i, y_i), i = 1, \ldots, n\}$
- Find: predictor $f \in \mathcal{H}$
- Goal: make good predictions $\hat{y} = f(x)$ on *test* data

Connection between training and test data?

Source: [Y. Liang]

Formalization

- Given: training data $\{(x_i, y_i), i = 1, \ldots, n\}$ i.i.d. from distribution D
- Find: predictor $f \in \mathcal{H}$
- Goal: make good predictions $\hat{y} = f(x)$ on *test* data i.i.d. from distribution D

i.i.d. - independent and identically distributed

Source: [Y. Liang]

Formalization

- Given: training data $\{(x_i, y_i), i = 1, \ldots, n\}$ i.i.d. from distribution D
- Find: predictor $f \in \mathcal{H}$
- Goal: make good predictions $\hat{y} = f(x)$ on *test* data i.i.d. from distribution D

What kind of performance measure?

Source: Y. Liang

Formalization

- Given: training data $\{(x_i, y_i), i = 1, \ldots, n\}$ i.i.d. from distribution D
- Find: predictor $f \in \mathcal{H}$
- S.t. the *expected loss* is small:

$$L(f) = \mathbb{E}_{(x,y)\sim D}[l(f, x, y)]$$

Various loss functions

Source: Y. Liang

Formalization

- Given: training data $\{(x_i, y_i), i = 1, \ldots, n\}$ i.i.d. from distribution D
- Find: predictor $f \in \mathcal{H}$
- S.t. the *expected loss* is small:

$$L(f) = \mathbb{E}_{(x,y)\sim D}[l(f, x, y)]$$

- Example losses:

$0 - 1$ loss: $l(f, x, y) = \mathbb{I}[f(x) \neq y]$ and $L(f) = \Pr[f(x) \neq y]$

l_2 loss: $l(f, x, y) = [f(x) - y]^2$ and $L(f) = \mathbb{E}[f(x) - y]^2$

Source: Y. Liang

Formalization

- Given: training data $\{(x_i, y_i), i = 1, \ldots, n\}$ i.i.d. from distribution D
- Find: predictor $f \in \mathcal{H}$
- S.t. the *expected loss* is small:

$$L(f) = \mathbb{E}_{(x,y)\sim D}[l(f, x, y)]$$

Can't optimize this directly

Source: Y. Liang

Formalization

- Given: training data $\{(x_i, y_i), i = 1, \ldots, n\}$ i.i.d. from distribution D
- Find: predictor $f \in \mathcal{H}$ that minimizes

$$\hat{L}(f) = \frac{1}{n} \sum_{i=1}^{n} l(f, x_i, y_i)$$

Empirical loss

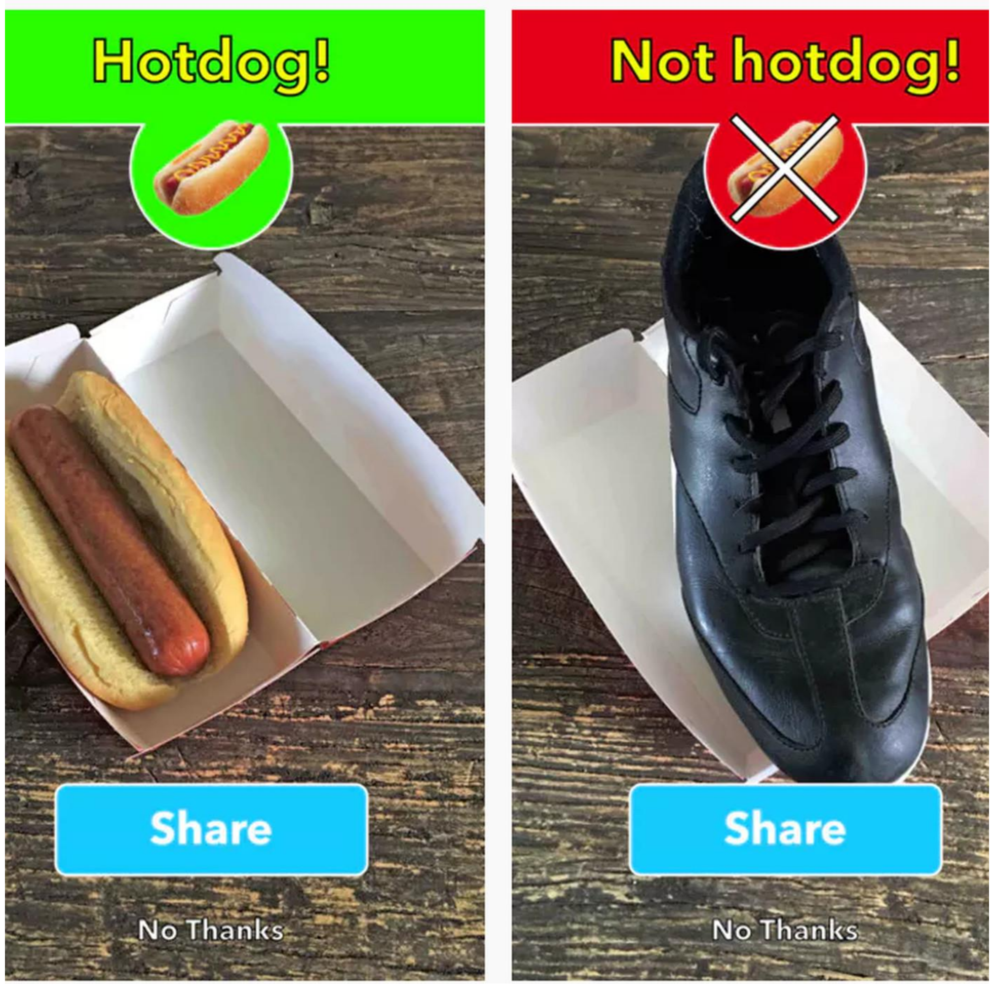
Source: Y. Liang

Supervised learning in a nutshell

1. **Collect *training data* and labels**
2. **Specify model:** select *hypothesis class* and *loss function*
3. **Train model:** find the function in the hypothesis class that minimizes the *empirical loss* on the training data

Training linear classifiers

- Given: i.i.d. training data $\{(x_i, y_i), i = 1, \ldots, n\}$, $y_i \in \{-1,1\}$



Training linear classifiers

- Given: i.i.d. training data $\{(x_i, y_i), i = 1, \dots, n\}$, $y_i \in \{-1,1\}$
- Hypothesis class: $f_w(x) = \mathrm{sgn}(w^T x)$
- Classification with *bias*, i.e. $f_w(x) = \mathrm{sgn}(w^T x + b)$, can be reduced to the case w/o bias by letting $w' = [w; b]$ and $x' = [x; 1]$

Source: Y. Liang

Training linear classifiers

- Given: i.i.d. training data $\{(x_i, y_i), i = 1, \dots, n\}$,
  $$y_i \in \{-1,1\}$$
- Hypothesis class: $f_w(x) = \mathrm{sgn}(w^T x)$
- Loss: minimizing the number of mistakes
  $$\hat{L}(f_w) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}[\mathrm{sgn}(w^T x_i) \neq y_i]$$
- Difficult to optimize directly!

Source: Y. Liang

Linear regression ("straw man" model)

- Find $f_w(x) = w^T x$ that minimizes l_2 *loss* or *mean squared error*

$$\hat{L}(f_w) = \frac{1}{n}\sum_{i=1}^{n}(w^T x_i - y_i)^2$$

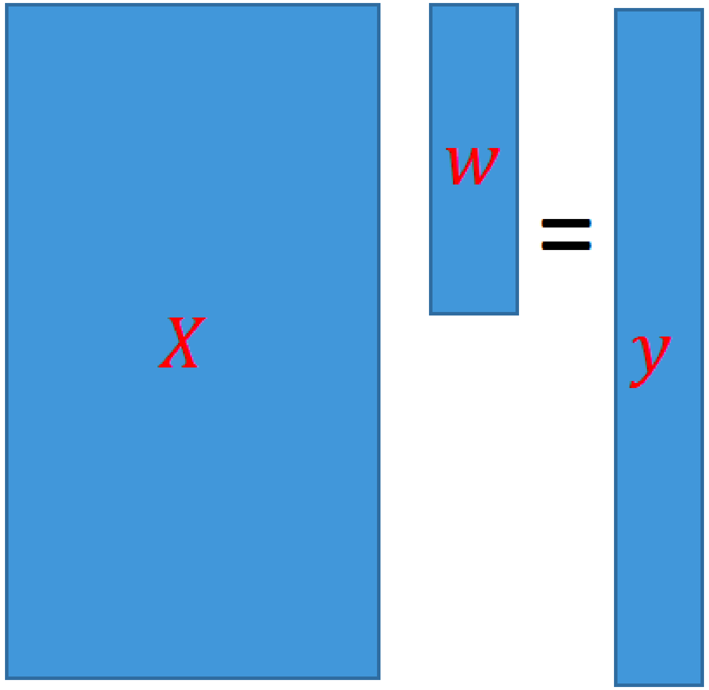
- Ignores the fact that $y \in \{-1,1\}$ but is easy to optimize

Source: Y. Liang

Linear regression: Optimization

- Let X be a matrix whose ith row is x_i^T, Y be the vector $(y_1, \dots, y_n)^T$

$$\hat{L}(f_w) = \frac{1}{n}\sum_{i=1}^{n}(w^T x_i - y_i)^2 = \|Xw - Y\|_2^2$$



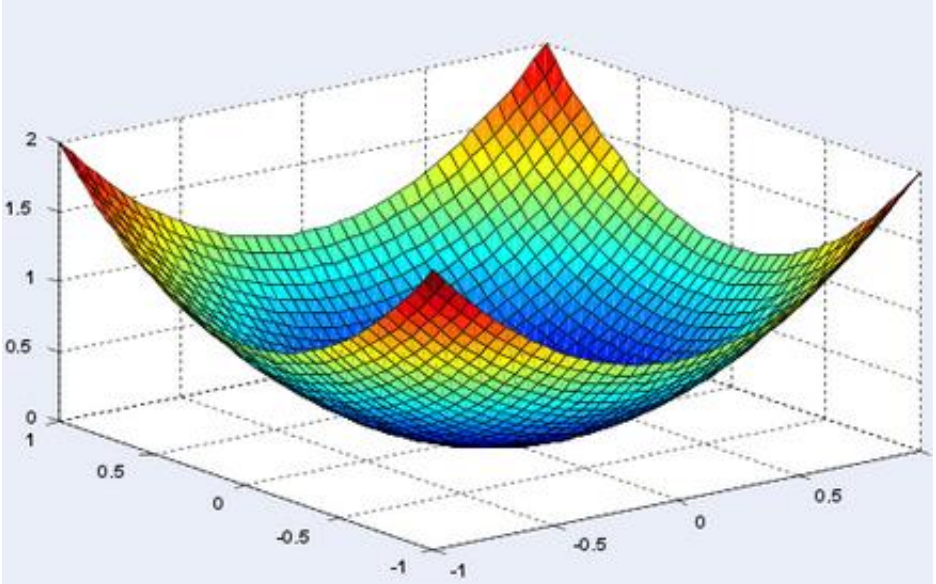
Source: [Y. Liang]

Linear regression: Optimization

- Let X be a matrix whose ith row is x_i^T, Y be the vector $(y_1, \dots, y_n)^T$

$$\hat{L}(f_w) = \frac{1}{n}\sum_{i=1}^{n}(w^T x_i - y_i)^2 = \|Xw - Y\|_2^2$$

- This is a *convex* function of the weights



Source: Y. Liang

Linear regression: Optimization

- Let X be a matrix whose ith row is x_i^T, Y be the vector $(y_1, \dots, y_n)^T$

$$\hat{L}(f_w) = \frac{1}{n}\sum_{i=1}^{n}(w^T x_i - y_i)^2 = \|Xw - Y\|_2^2$$

- Find the *gradient* w.r.t. w:

$$\nabla_w \|Xw - Y\|_2^2$$

Source: [Y. Liang]

Linear regression: Optimization

- Let X be a matrix whose ith row is x_i^T, y be the vector $(y_1, \dots, y_n)^T$

$$\hat{L}(f_w) = \frac{1}{n}\sum_{i=1}^{n}(w^T x_i - y_i)^2 = \|Xw - Y\|_2^2$$

- Find the *gradient* w.r.t. w:

$$\begin{aligned}\nabla_w \|Xw - Y\|_2^2 &= \nabla_w[(Xw - Y)^T(Xw - Y)] \\ &= \nabla_w[w^T X^T X w - 2w^T X^T Y + Y^T Y] \\ &= 2X^T X w - 2X^T Y\end{aligned}$$

- Set gradient to zero to get the minimizer:

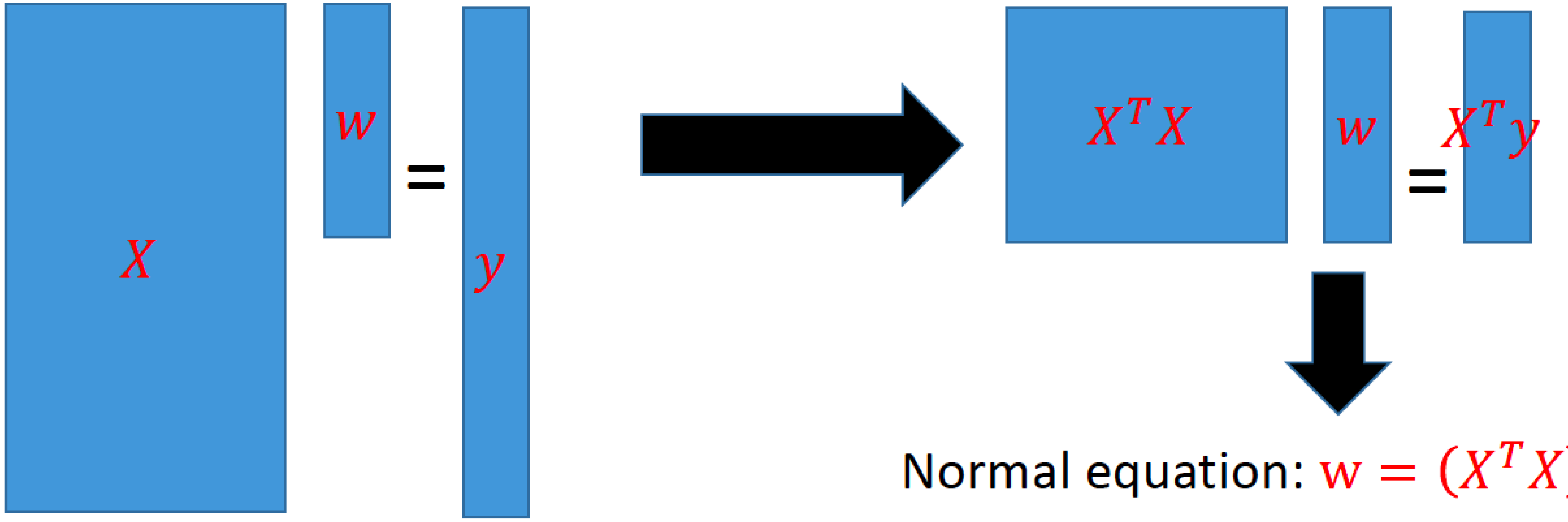
$$X^T X w = X^T Y$$

$$w = (X^T X)^{-1} X^T Y$$

Source: Y. Liang

Linear regression: Optimization

- Linear algebra view
  - If X is invertible, simply solve $Xw = Y$ and get $w = X^{-1}Y$
  - But typically X is a "tall" matrix so you need to find the *least squares solution* to an over-constrained system



Normal equation: $\mathrm{w} = (X^T X)^{-1} X^T y$

Source: [Y. Liang]

Problem with linear regression

- In practice, very sensitive to outliers

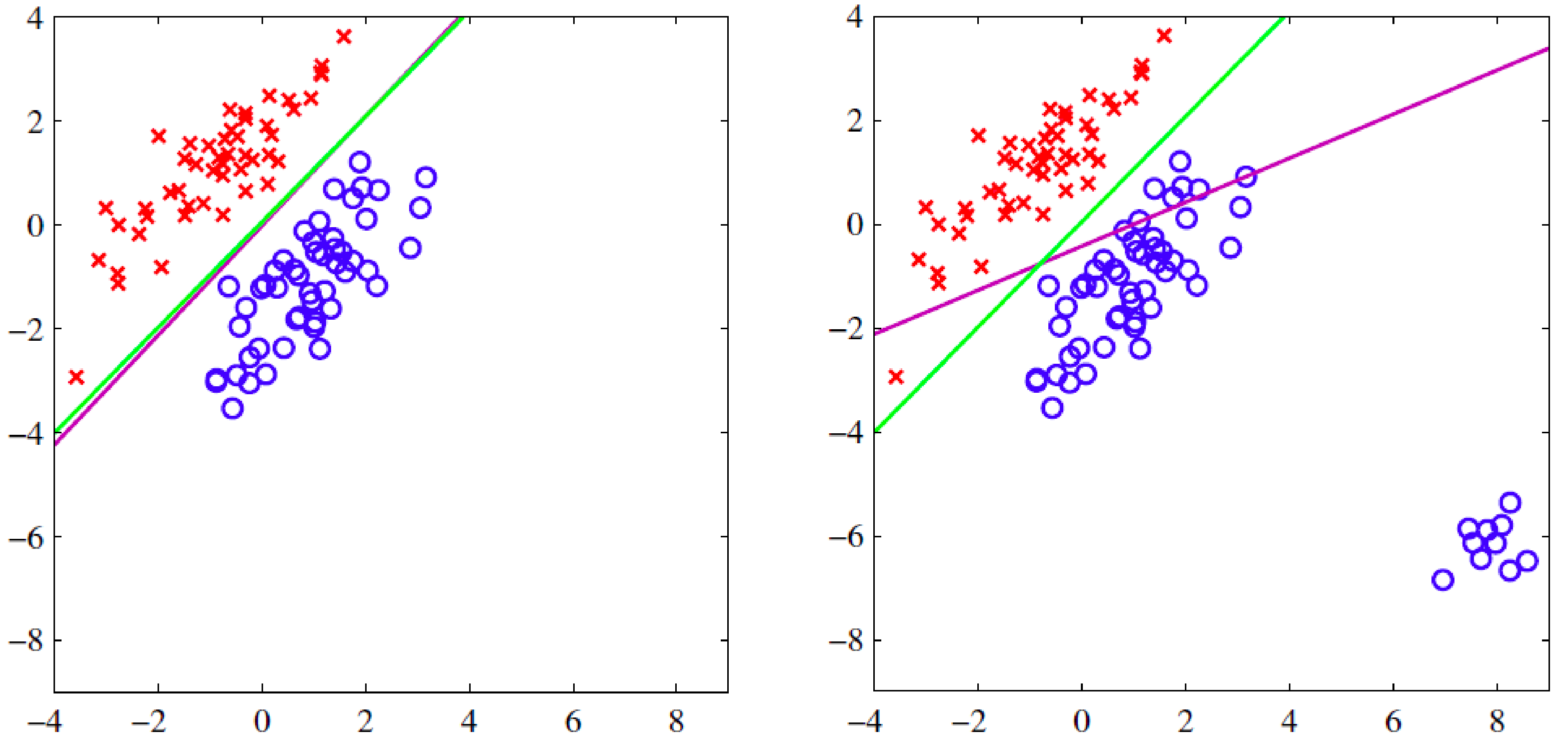


Figure 4.4 The left plot shows data from two classes, denoted by red crosses and blue circles, together with the decision boundary found by least squares (magenta curve) and also by the logistic regression model (green curve), which is discussed later in Section 4.3.2. The right-hand plot shows the corresponding results obtained when extra data points are added at the bottom left of the diagram, showing that least squares is highly sensitive to outliers, unlike logistic regression.

Figure from *Pattern Recognition and Machine Learning*, Bishop

Back to what we really want

- Given: i.i.d. training data $\{(x_i, y_i), i = 1, \dots, n\}$,
$$y_i \in \{-1,1\}$$
- Find $f_w(x) = \mathrm{sgn}(w^T x)$ that minimizes the number of mistakes
$$\hat{L}(f_w) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}[\mathrm{sgn}(w^T x_i) \neq y_i]$$

Source: Y. Liang

Linear classifiers: Outline

- Formalization of statistical learning of classifiers
- Linear classification models
  1. Linear regression (least squares)
  2. Perceptron training algorithm
  3. Logistic regression
  4. Support vector machines

Recap so far

- We want to learn a linear classifier that minimizes the number of mistakes on the training data (but this is NP-hard)
- **Attempt 1:** *relax* objective by dropping the sign function, using l_2 loss – *linear regression*
  - Easy to optimize, not really appropriate for binary classification, sensitive to outliers
- **Attempt 2:** *perceptron update rule*

Perceptron training algorithm

- Initialize weights randomly
- Cycle through training examples in multiple passes (*epochs*)
- For each training example (x_i, y_i):
- If current prediction $\mathrm{sgn}(w^T x_i)$ does not match y_i then update weights:

$$w \leftarrow w + \eta\, y_i x_i$$

where η is a *learning rate* that should decay slowly* over time

Understanding the update rule

- **Perceptron update rule:** If $y_i \neq \mathrm{sgn}(w^T x_i)$ then update weights:

$$w \leftarrow w + \eta\, y_i x_i$$

- The raw response of the classifier changes to

$$w^T x_i + \eta\, y_i \|x_i\|^2$$

- How does the response change if $y_i = 1$?
  - The response $w^T x_i$ is initially *negative* and will be *increased*
- How does the response change if $y_i = -1$?
  - The response $w^T x_i$ is initially *positive* and will be *decreased*

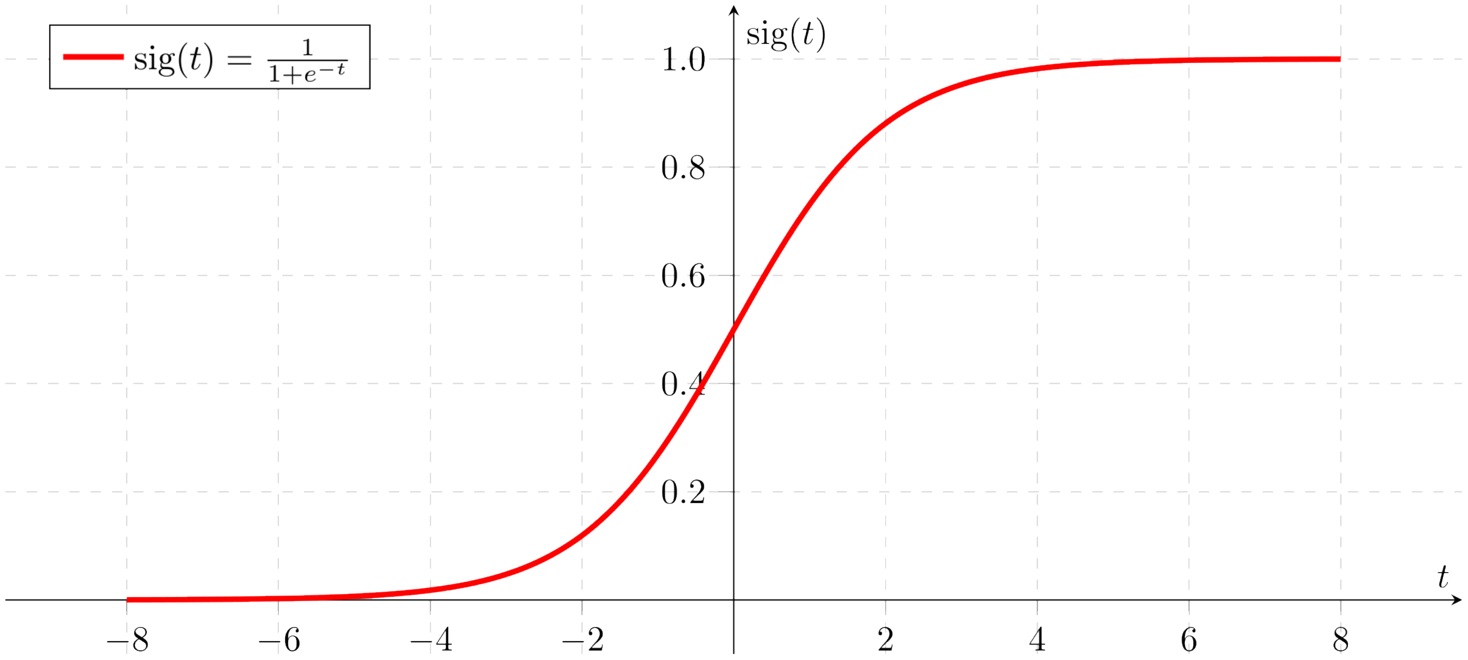
Recap so far

- We want to learn a linear classifier that minimizes the number of mistakes on the training data (but this is NP-hard)
- **Attempt 1:** *relax* objective by dropping the sign function, using l_2 loss – *linear regression*
  - Easy to optimize, not really appropriate for binary classification, sensitive to outliers
- **Attempt 2:** *perceptron update rule*
  - Converges for separable data, but hard to extend to optimizing more complicated models
- **Attempt 3:** relax objective by turning the sign function into a differentiable "soft threshold" – *logistic regression*

Sigmoid function

- Squash the linear response of the classifier to the interval [0,1]:

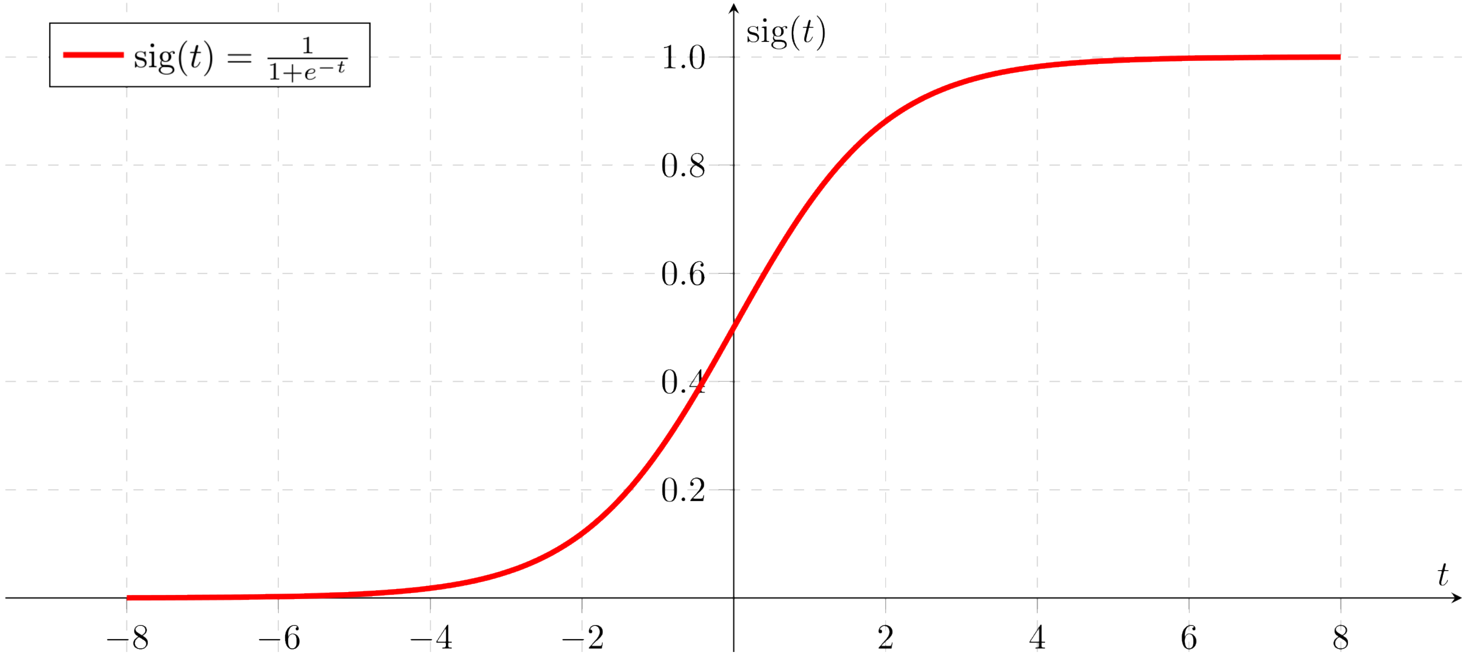
$$\sigma(w^T x) = \frac{1}{1 + \exp(-w^T x)}$$



Sigmoid function

- Output of sigmoid can be interpreted as posterior label probability or confidence returned by classifier:

$$P_w(y = 1|x) = \sigma(w^T x) = \frac{1}{1+\exp(-w^T x)}$$

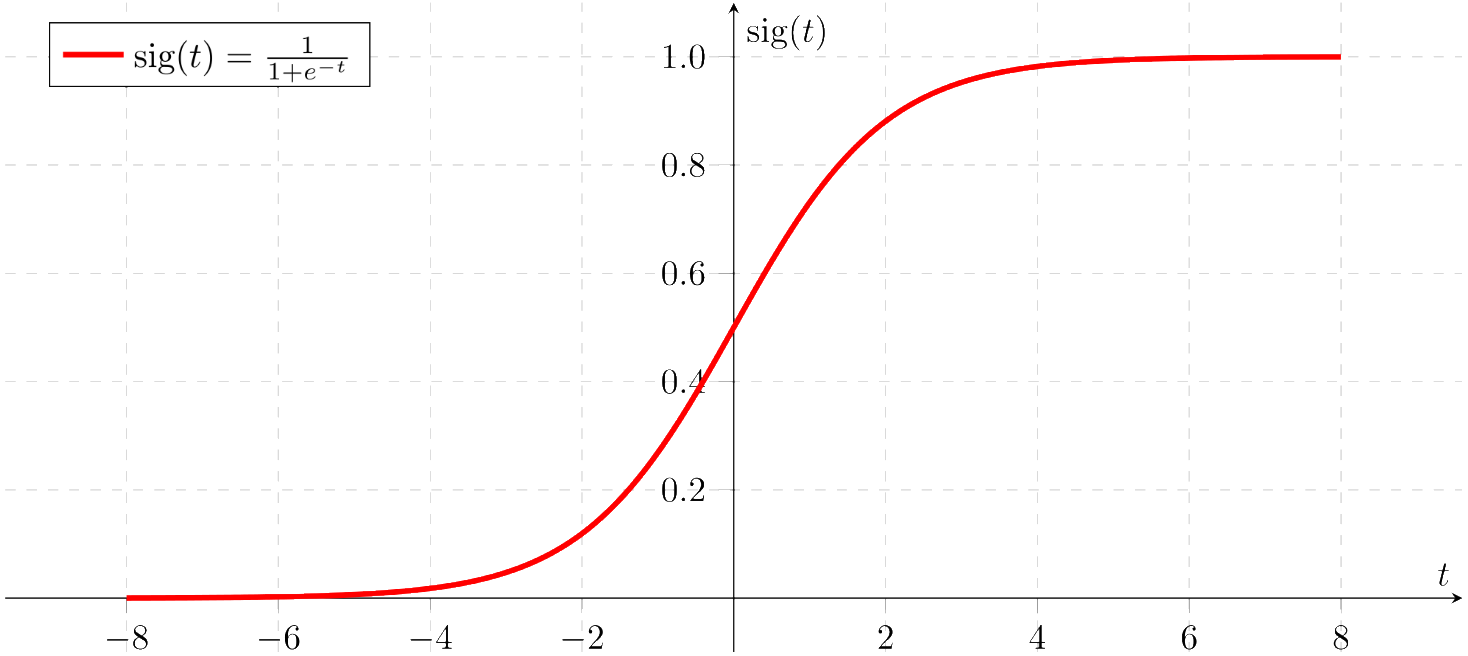


Sigmoid function

- Output of sigmoid can be interpreted as posterior label probability or confidence returned by classifier:

$$P_w(y = 1|x) = \sigma(w^T x) = \frac{1}{1+\exp(-w^T x)}$$

- What is $P_w(y = -1|x)$?



Sigmoid function

- Output of sigmoid can be interpreted as posterior label probability or confidence returned by classifier:

$$P_w(y = 1|x) = \sigma(w^T x) = \frac{1}{1+\exp(-w^T x)}$$

- What is $P_w(y = -1|x)$?

$$P_w(y = -1|x) = 1 - \sigma(w^T x)$$

$$= \frac{1+\exp(-w^T x)-1}{1+\exp(-w^T x)} = \frac{\exp(-w^T x)}{1+\exp(-w^T x)} = \frac{1}{\exp(w^T x)+1}$$

$$= \sigma(-w^T x)$$

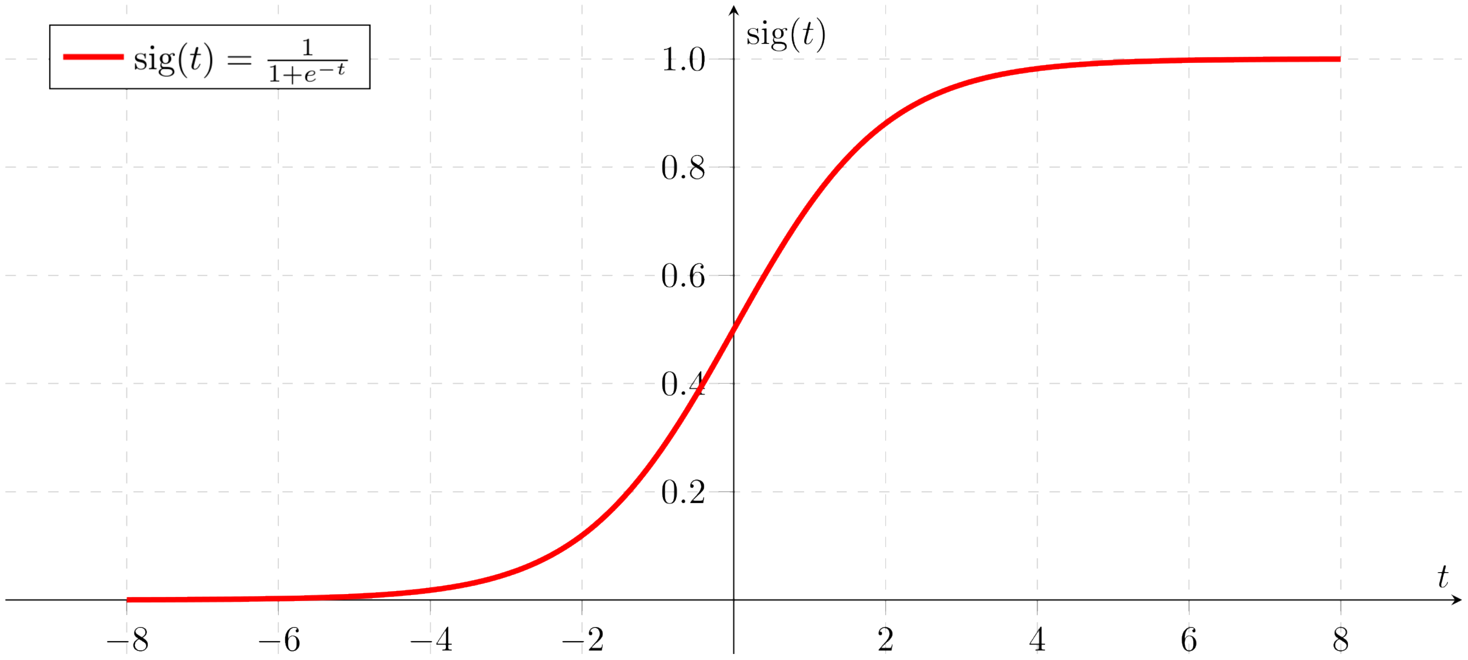
- Thus, sigmoid is *symmetric*: $1 - \sigma(t) = \sigma(-t)$

Sigmoid function

- Output of sigmoid can be interpreted as posterior label probability or confidence returned by classifier:

$$P_w(y = 1|x) = \sigma(w^T x) = \frac{1}{1+\exp(-w^T x)}$$

- Sigmoid is *symmetric*: $1 - \sigma(t) = \sigma(-t)$

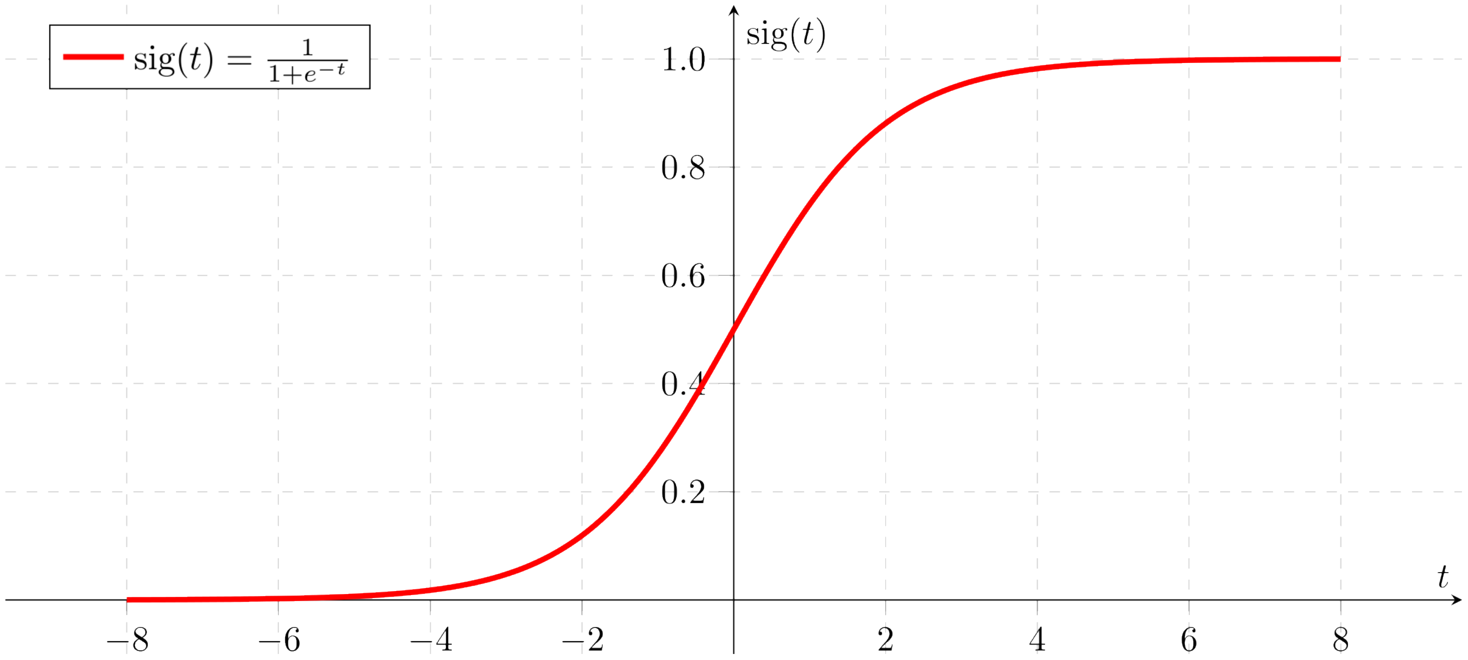


Sigmoid function

- Output of sigmoid can be interpreted as posterior label probability or confidence returned by classifier:

$$P_w(y=1|x) = \sigma(w^T x) = \frac{1}{1+\exp(-w^T x)}$$

- What happens if we scale w by a constant?



Sigmoid function

- Output of sigmoid can be interpreted as posterior label probability or confidence returned by classifier:

$$P_w(y = 1|x) = \sigma(w^T x) = \frac{1}{1+\exp(-w^T x)}$$

- What happens if we scale w by a constant?

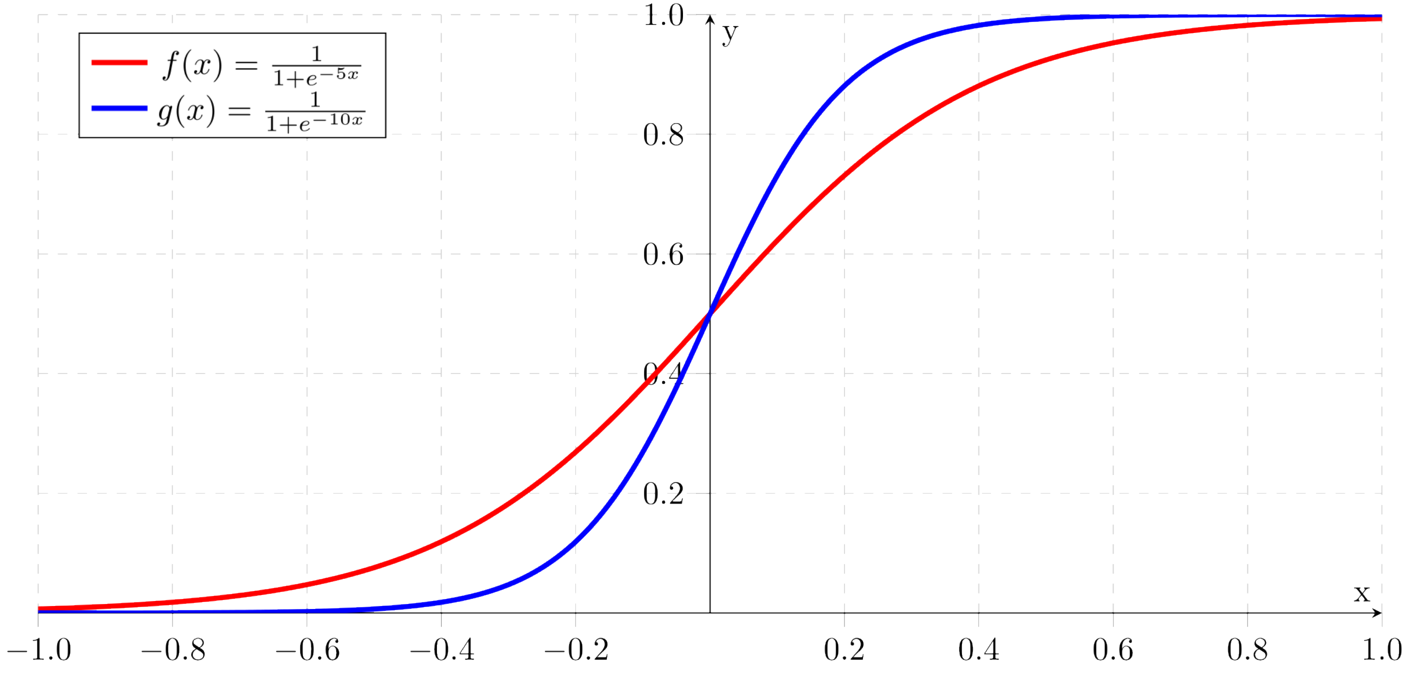


Image source

Sigmoid: Interpretation

- We can write out the connection between the *posteriors* $P(y|x)$ and the *class-conditional densities* $P(x|y)$:

posterior = $\dfrac{\text{likelihood x prior}}{\text{evidence}}$

$$P(y=1|x) = \frac{P(x|y=1)P(y=1)}{P(x)}$$

$$= \frac{P(x|y=1)P(y=1)}{P(x|y=1)P(y=1) + P(x|y=-1)P(y=-1)}$$

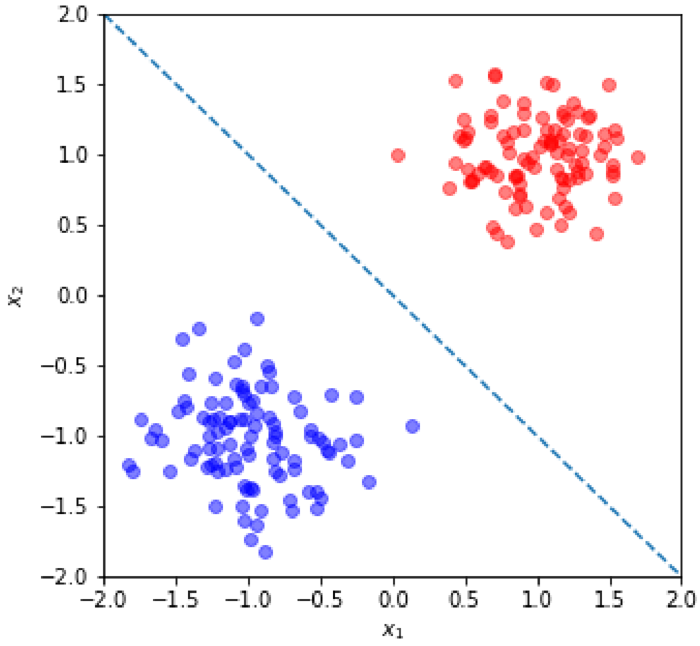
$$= \frac{1}{1+\exp(-a)} = \sigma(a), \qquad a = \log \frac{P(y=1|x)}{P(y=-1|x)}$$

Sigmoid: Interpretation

- Adopting a linear + sigmoid model is equivalent to assuming *linear log likelihood*:

$$\log \frac{P(y=1|x)}{P(y=-1|x)} = w^T x + b$$

- This happens when $P(x|y=1)$ and $P(x|y=-1)$ are Gaussians with different means and the same covariance matrices (w is related to the difference between the means)



Logistic regression: Training

- Given: $\{(x_i, y_i), i = 1, \dots, n\}$, $y_i \in \{-1,1\}$
- Find w that minimizes

$$\hat{L}(w) = -\frac{1}{n}\sum_{i=1}^{n} \log P_w(y_i|x_i)$$

$$= -\frac{1}{n}\sum_{i:y_i=1}^{n} \log \sigma(w^T x_i) - \frac{1}{n}\sum_{i:y_i=-1}^{n} \log[1 - \sigma(w^T x_i)]$$

$$= -\frac{1}{n}\sum_{i:y_i=1}^{n} \log \sigma(w^T x_i) - \frac{1}{n}\sum_{i:y_i=-1}^{n} \log[\sigma(-w^T x_i)]$$

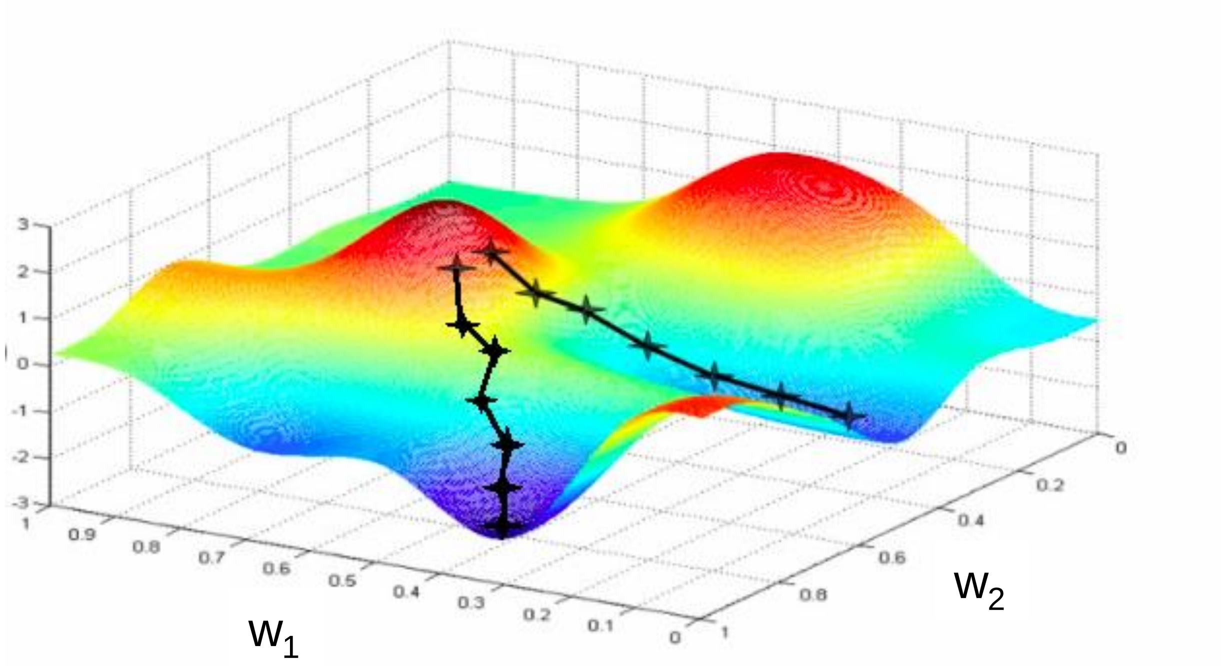
$$= -\frac{1}{n}\sum_{i=1}^{n} \log \sigma(y_i w^T x_i)$$

No closed-form solution, need to use *gradient descent*

Gradient descent

- Goal: find w to minimize loss $\hat{L}(w)$
- Start with some initial estimate of w
- At each step, find $\nabla \hat{L}(w)$, the *gradient* of the loss w.r.t. w, and take a small step in the *opposite* direction

$$w \leftarrow w - \eta \, \nabla \hat{L}(w)$$



Gradient descent

- Goal: find w to minimize loss $\hat{L}(w)$
- Start with some initial estimate of w
- At each step, find $\nabla\hat{L}(w)$, the *gradient* of the loss w.r.t. w, and take a small step in the *opposite* direction

$$w \leftarrow w - \eta\,\nabla\hat{L}(w)$$

- Note: step size plays a crucial role

Gradient descent for logistic regression

$$\hat{L}(w) = -\frac{1}{n}\sum_{i=1}^{n} \log \sigma(y_i w^T x_i)$$

$$\nabla \hat{L}(w) = -\frac{1}{n}\sum_{i=1}^{n} \nabla_w \log \sigma(y_i w^T x_i)$$

- Derivative of log:

$$\left[\log\left(f(x)\right)\right]' = \frac{f'(x)}{f(x)}$$

Gradient descent for logistic regression

$$\hat{L}(w) = -\frac{1}{n}\sum_{i=1}^{n} \log \sigma(y_i w^T x_i)$$

$$\nabla \hat{L}(w) = -\frac{1}{n}\sum_{i=1}^{n} \nabla_w \log \sigma(y_i w^T x_i)$$

$$= -\frac{1}{n}\sum_{i=1}^{n} \frac{\nabla_w \sigma(y_i w^T x_i)}{\sigma(y_i w^T x_i)}$$

- Derivative of sigmoid:

$$\sigma'(a) = \sigma(a)(1 - \sigma(a)) = \sigma(a)\sigma(-a)$$

Gradient descent for logistic regression

$$\hat{L}(w) = -\frac{1}{n}\sum_{i=1}^{n} \log \sigma(y_i w^T x_i)$$

$$\nabla \hat{L}(w) = -\frac{1}{n}\sum_{i=1}^{n} \nabla_w \log \sigma(y_i w^T x_i)$$

$$= -\frac{1}{n}\sum_{i=1}^{n} \frac{\nabla_w \sigma(y_i w^T x_i)}{\sigma(y_i w^T x_i)}$$

$$= -\frac{1}{n}\sum_{i=1}^{n} \frac{\sigma(y_i w^T x_i)\sigma(-y_i w^T x_i) y_i x_i}{\sigma(y_i w^T x_i)}$$

- We also used the *chain rule*: $\left[g\left(f(x)\right)\right]' = g'\left(f(x)\right)f'(x)$

Gradient descent for logistic regression

$$\hat{L}(w) = -\frac{1}{n}\sum_{i=1}^{n} \log \sigma(y_i w^T x_i)$$

$$\nabla \hat{L}(w) = -\frac{1}{n}\sum_{i=1}^{n} \nabla_w \log \sigma(y_i w^T x_i)$$

$$= -\frac{1}{n}\sum_{i=1}^{n} \frac{\nabla_w \sigma(y_i w^T x_i)}{\sigma(y_i w^T x_i)}$$

$$= -\frac{1}{n}\sum_{i=1}^{n} \frac{\sigma(y_i w^T x_i)\sigma(-y_i w^T x_i) y_i x_i}{\sigma(y_i w^T x_i)}$$

$$= -\frac{1}{n}\sum_{i=1}^{n} \sigma(-y_i w^T x_i) y_i x_i$$

Gradient descent for logistic regression

$$\nabla\hat{L}(w) = -\frac{1}{n}\sum_{i=1}^{n}\sigma(-y_i w^T x_i)y_i x_i$$

- Update for w:

$$w \leftarrow w + \eta\frac{1}{n}\sum_{i=1}^{n}\sigma(-y_i w^T x_i)y_i x_i$$

- For a single parameter update, need to cycle through the entire training set!
    - This is also called a *batch* update

Stochastic gradient descent (SGD)

- At each iteration, take a single data point (x_i, y_i) and perform a parameter update using $\nabla l(w, x_i, y_i)$, the gradient of the loss for that point:

$$w \leftarrow w - \eta \, \nabla l(w, x_i, y_i)$$

- This is called an *online* or *stochastic* update

SGD for logistic regression

- Full empirical loss:

$$\hat{L}(w) = -\frac{1}{n}\sum_{i=1}^{n} \log \sigma(y_i w^T x_i)$$

- Loss for a single sample:

$$l(w, x_i, y_i) = -\log \sigma(y_i w^T x_i)$$

- Full gradient update:

$$w \leftarrow w + \eta \frac{1}{n}\sum_{i=1}^{n} \sigma(-y_i w^T x_i) y_i x_i$$

- SGD update:

$$w \leftarrow w + \eta\, \sigma(-y_i w^T x_i) y_i x_i$$

SGD for logistic regression

- Let's take a closer look at the SGD update:

$$w \leftarrow w + \eta\, \sigma(-y_i w^T x_i) y_i x_i$$

  - For an *incorrectly* classified point, $-y_i w^T x_i$ is *positive*, and the update rule approaches the perceptron update rule:

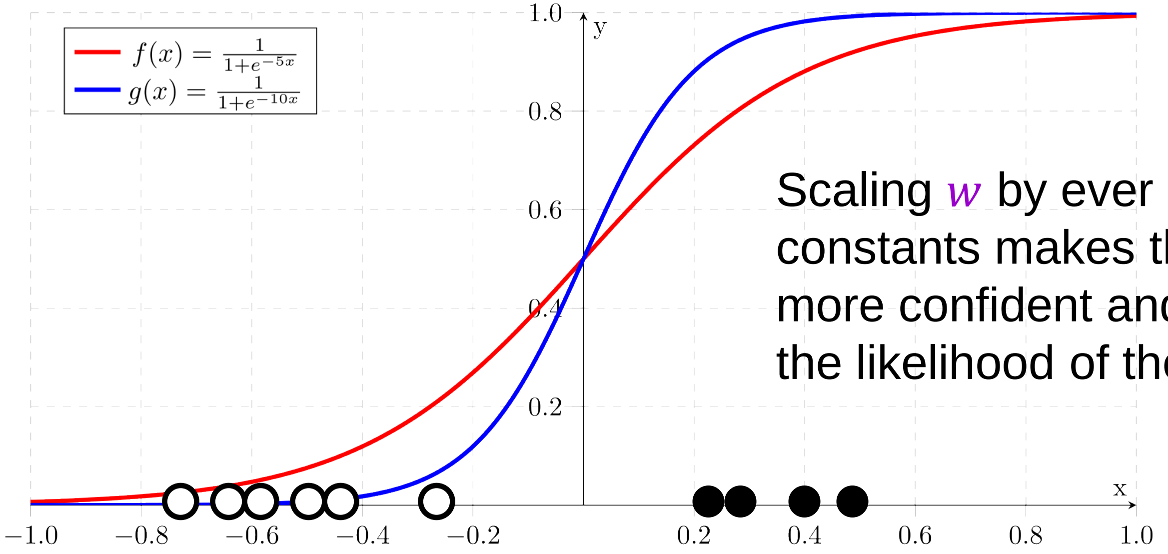
$$w \leftarrow w + \eta\, y_i x_i$$

SGD for logistic regression

- Let's take a closer look at the SGD update:

$$w \leftarrow w + \eta\, \sigma(-y_i w^T x_i) y_i x_i$$

  - For a *correctly* classified point, $-y_i w^T x_i$ is *negative*, so $\sigma(-y_i w^T x_i)$ is small
  - However, the update never reaches zero, and logistic regression actually *does not converge* for linearly separable data!



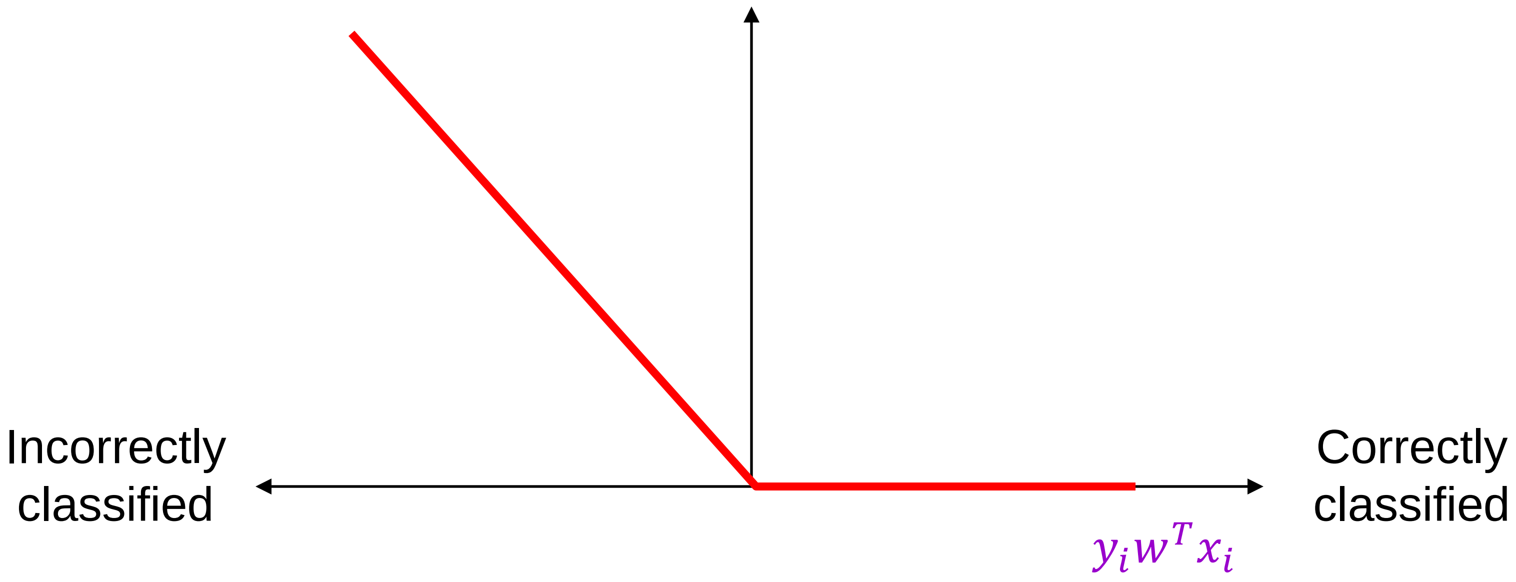
Scaling w by ever larger constants makes the classifier more confident and increases the likelihood of the data

Image source

Revisiting the perceptron training algorithm

- Define *hinge loss*

$$l(w, x_i, y_i) = \max(0, -y_i w^T x_i)$$

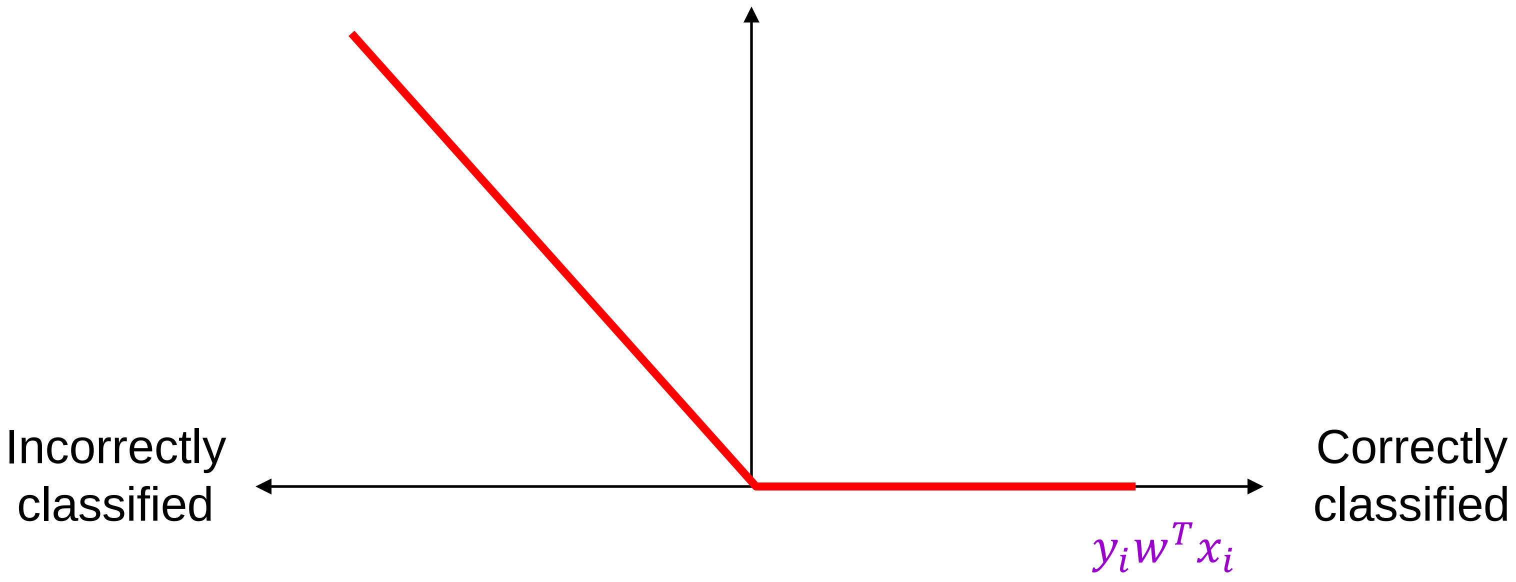


Revisiting the perceptron training algorithm

- Define *hinge loss*

$$l(w, x_i, y_i) = \max(0, -y_i w^T x_i)$$

- Let's find the gradient update
  - Note: strictly speaking the hinge loss is not differentiable, so this is called a *sub-gradient*

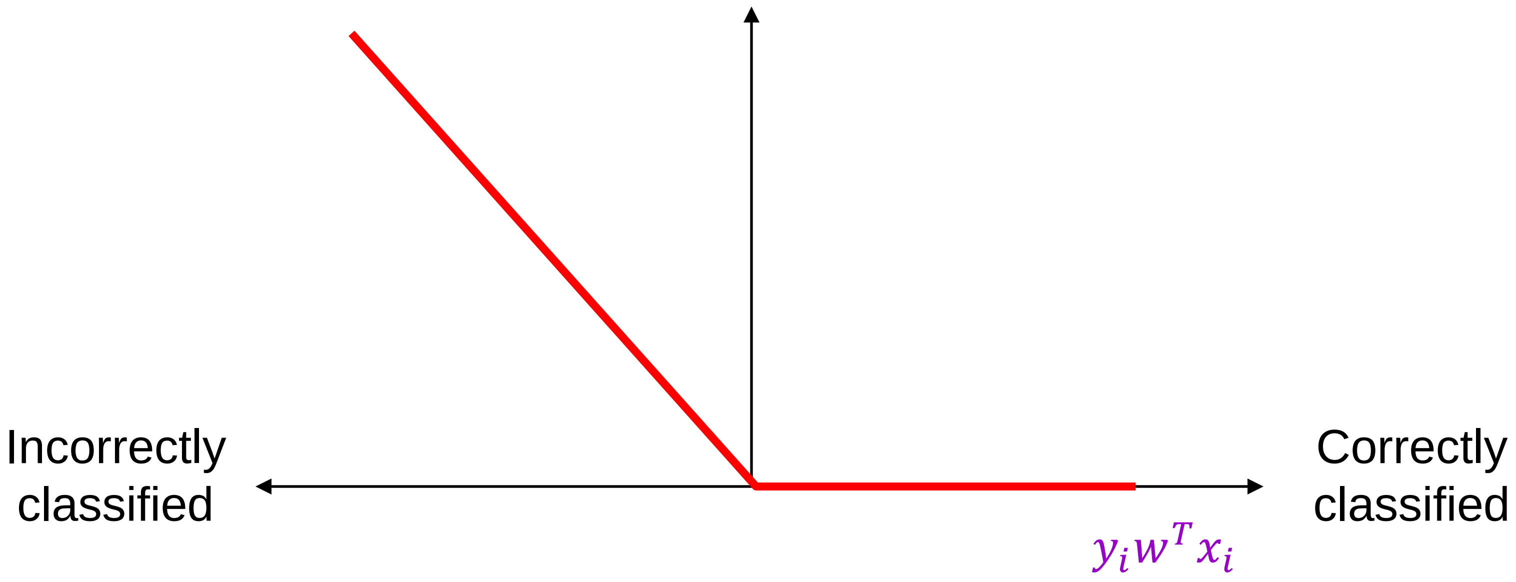


Revisiting the perceptron training algorithm

- Define *hinge loss*

$$l(w, x_i, y_i) = \max(0, -y_i w^T x_i)$$

$$\frac{\partial}{\partial a} \max(0, a) = \mathbb{I}[a > 0]$$



Revisiting the perceptron training algorithm

- Define *hinge loss*

$$l(w, x_i, y_i) = \max(0, -y_i w^T x_i)$$

$$\frac{\partial}{\partial a}\max(0, a) = \mathbb{I}[a > 0]$$

$$\nabla l(w, x_i, y_i) = -\mathbb{I}[y_i w^T x_i < 0] y_i x_i$$

- Corresponding SGD update:

$$w \leftarrow w + \eta\, \mathbb{I}[y_i w^T x_i < 0] y_i x_i$$

- This is the same as the perceptron update we originally introduced!

Revisiting linear regression

- For completeness: what is the SGD update for linear regression?

$$l(w, x_i, y_i) = (w^T x_i - y_i)^2$$

$$\nabla l(w, x_i, y_i) = 2(w^T x_i - y_i)x_i$$

- Update: $w \leftarrow w - \eta\,(w^T x_i - y_i)x_i$

- What will happen if x_i is correctly classified with high confidence?

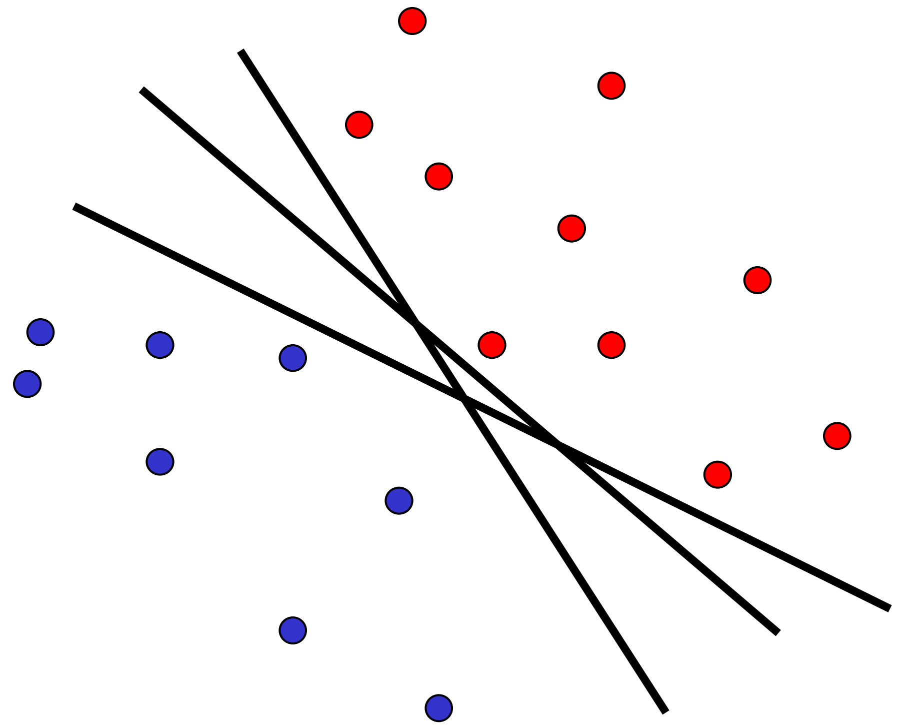
Linear classifiers: Outline

- Formalization of statistical learning of classifiers
- Linear classification models
  1. Linear regression (least squares)
  2. Perceptron training algorithm
  3. Logistic regression
  4. Support vector machines

Support vector machines

- When the data is linearly separable, which of the many possible solutions should we prefer?

  - **Perceptron training algorithm:** no special criterion, solution depends on initialization

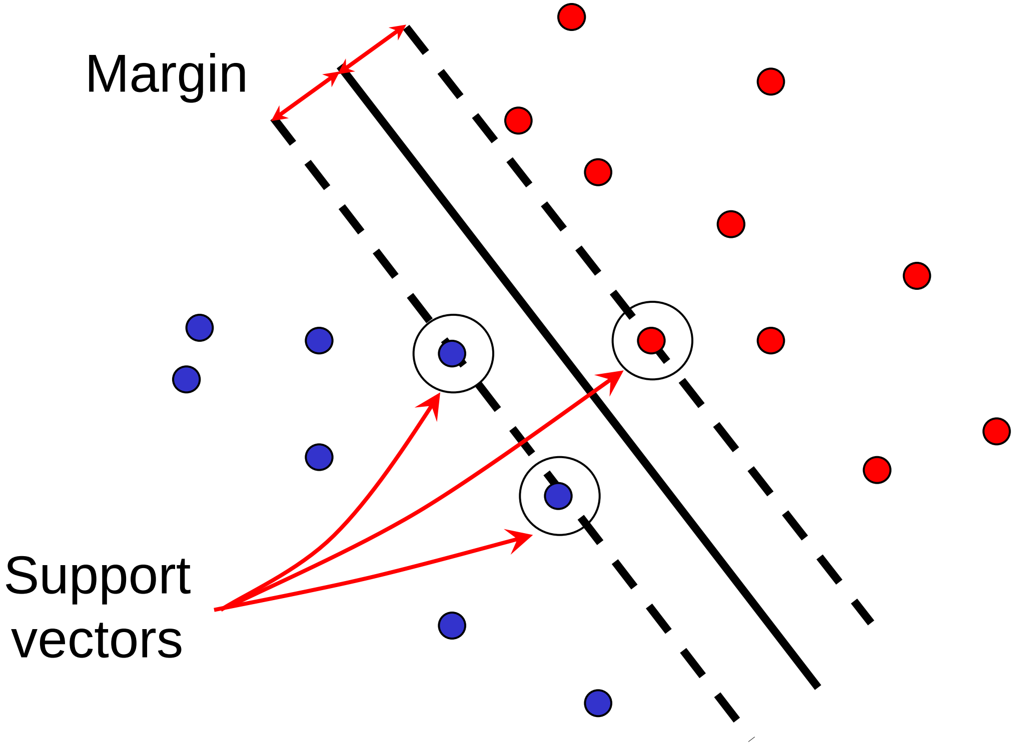


Support vector machines

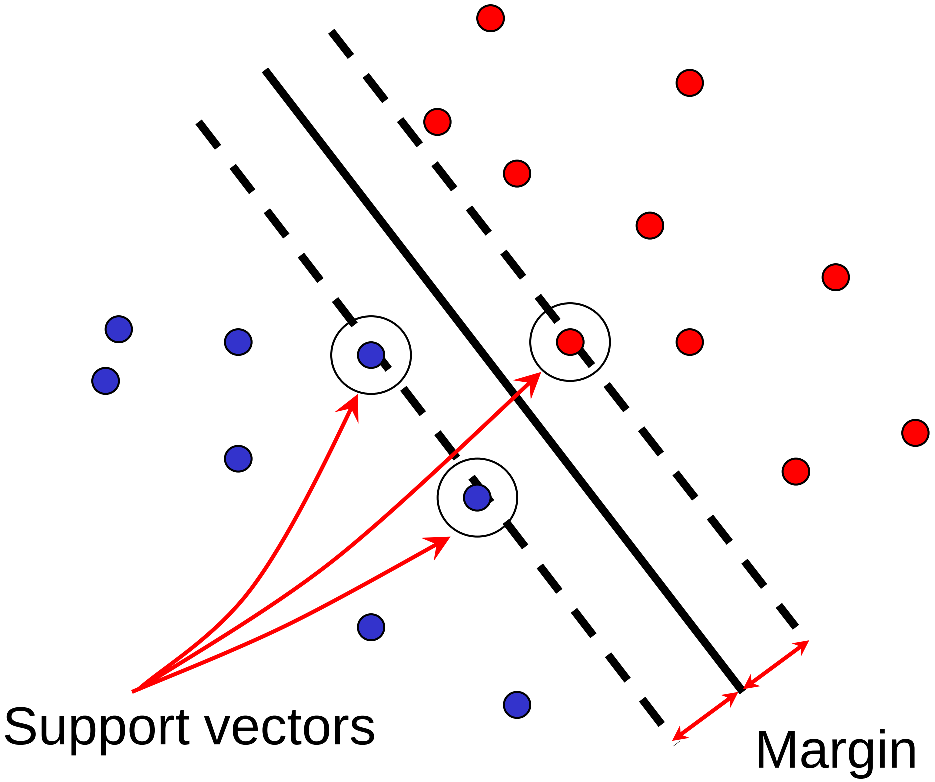
- When the data is linearly separable, which of the many possible solutions should we prefer?

  - **Perceptron training algorithm:** no special criterion, solution depends on initialization

  - **SVM criterion:** maximize the *margin*, or distance between the hyperplane and the closest training example



Finding the maximum margin hyperplane



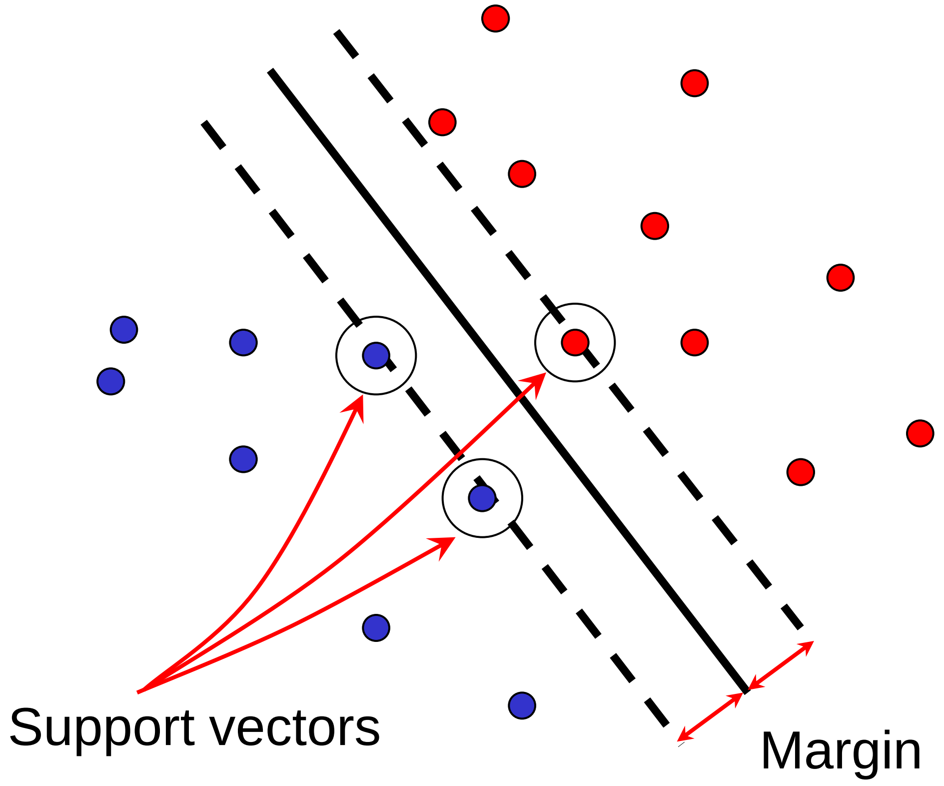
Positive examples: $w^T x_i \geq 1$
Negative examples: $w^T x_i \leq -1$

For support vectors, $w^T x_i = \pm 1$

The margin is $2 * \frac{|w^T x_i|}{\|w\|} = \frac{2}{\|w\|}$

Finding the maximum margin hyperplane

- We want to maximize margin $2/\|w\|$ while correctly classifying all training data: $y_i w^T x_i \geq 1$



Positive examples: $w^T x_i \geq 1$
Negative examples: $w^T x_i \leq -1$

For support vectors, $w^T x_i = \pm 1$

The margin is $2 * \frac{|w^T x_i|}{\|w\|} = \frac{2}{\|w\|}$

Finding the maximum margin hyperplane

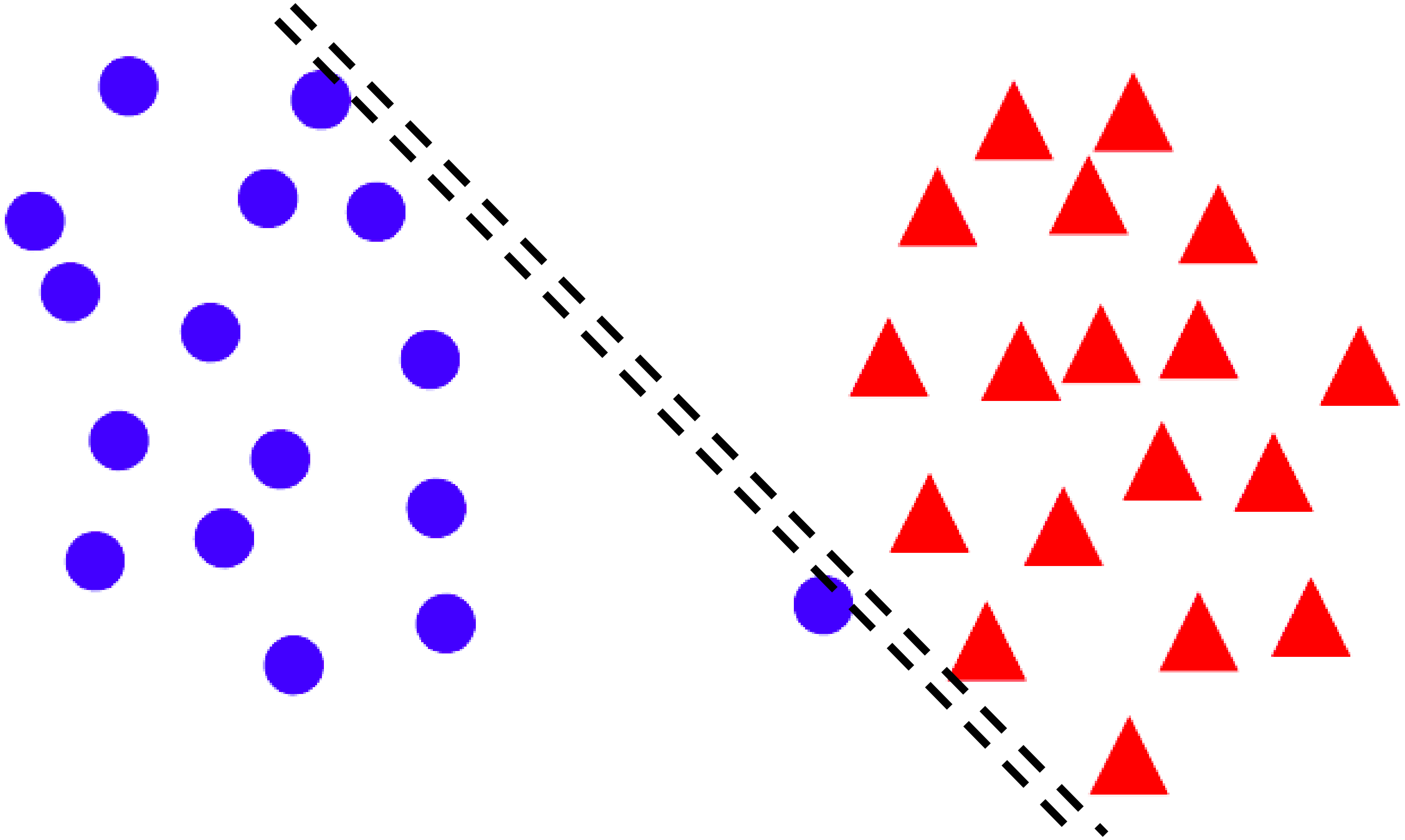
- We want to maximize margin $2/\|w\|$ while correctly classifying all training data: $y_i w^T x_i \geq 1$
- Equivalent problem:

$$\min_w \frac{1}{2}\|w\|^2 \quad \text{s.t.} \quad y_i w^T x_i \geq 1 \;\; \forall i$$

  - This is a quadratic objective with linear constraints: convex optimization problem, global optimum can be found using well-studied methods

“Soft margin” formulation

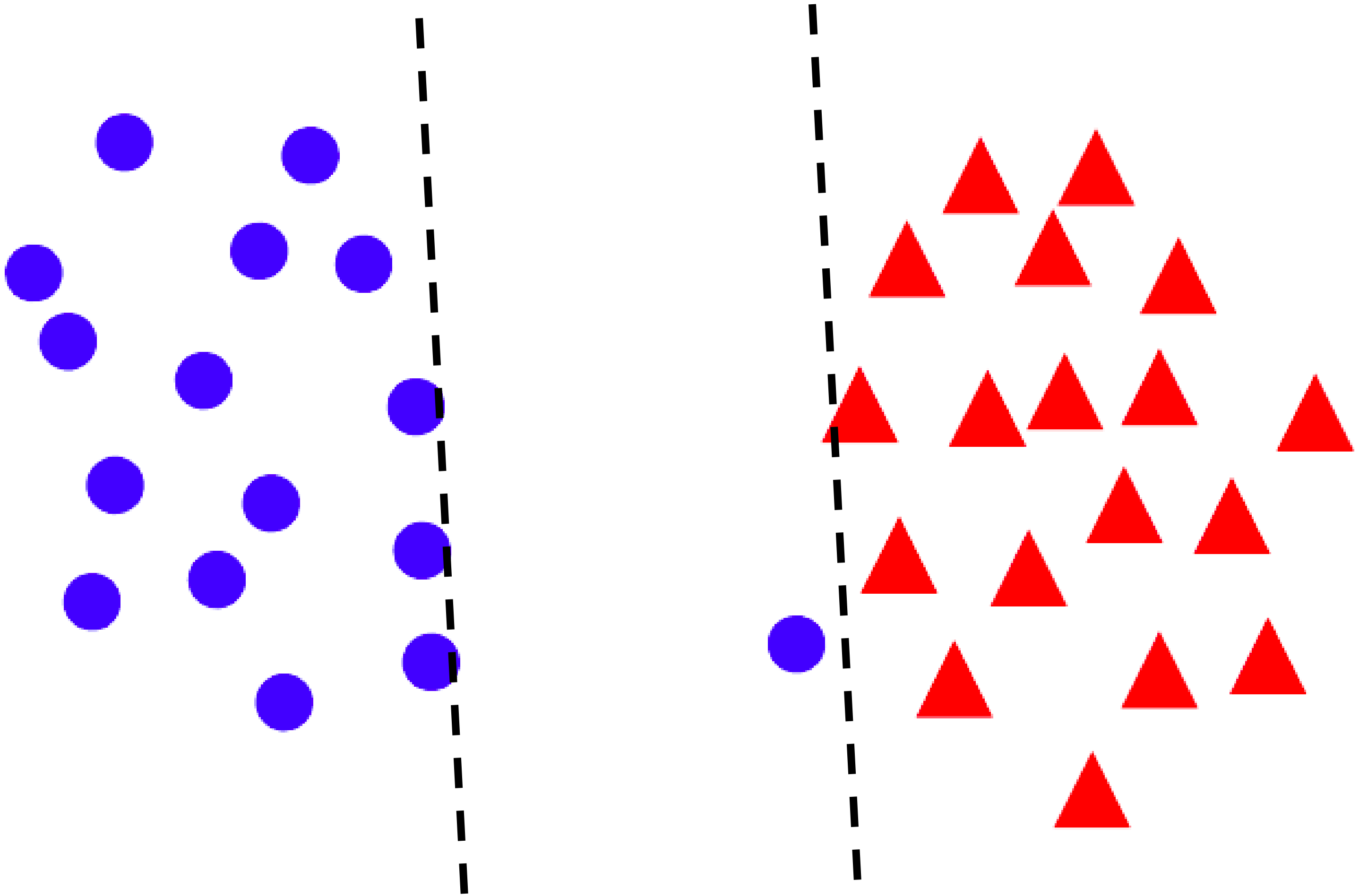
- What about non-separable data?
- And even for separable data, we may prefer a larger margin with a few constraints violated



[Source](#)

“Soft margin” formulation

- What about non-separable data?
- And even for separable data, we may prefer a larger margin with a few constraints violated

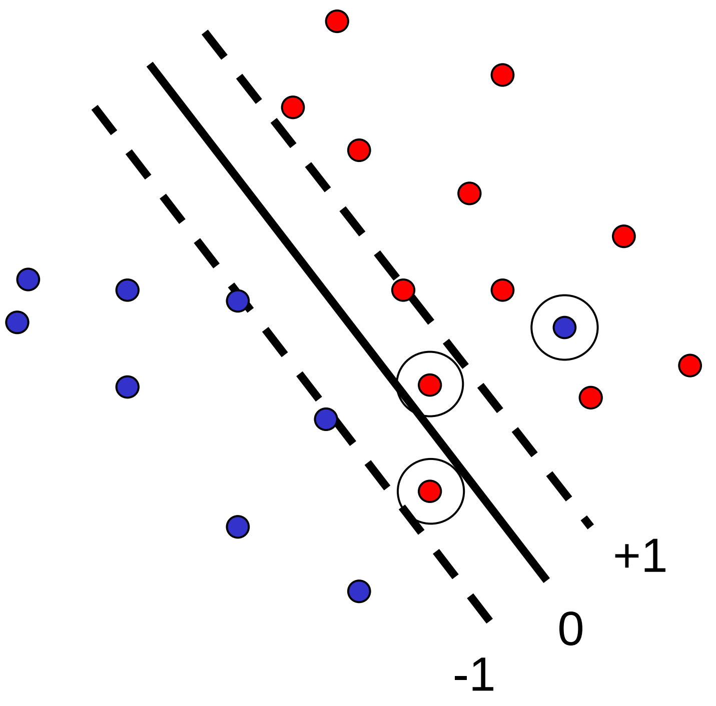


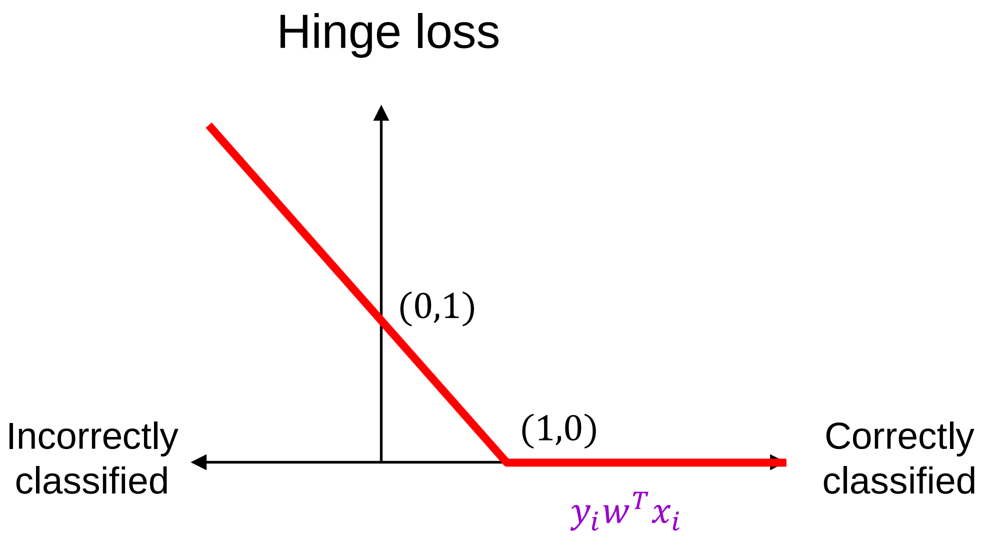
Source

“Soft margin” formulation

- Penalize margin violations using *hinge loss*:

$$\min_w \frac{\lambda}{2}\|w\|^2 \quad + \quad \sum_{i=1}^{n} \max[0, 1 - y_i w^T x_i]$$

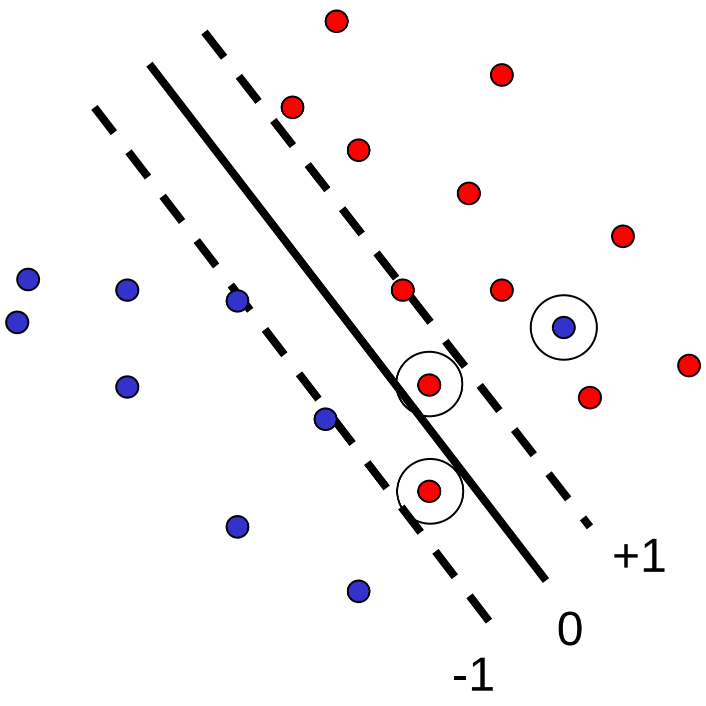


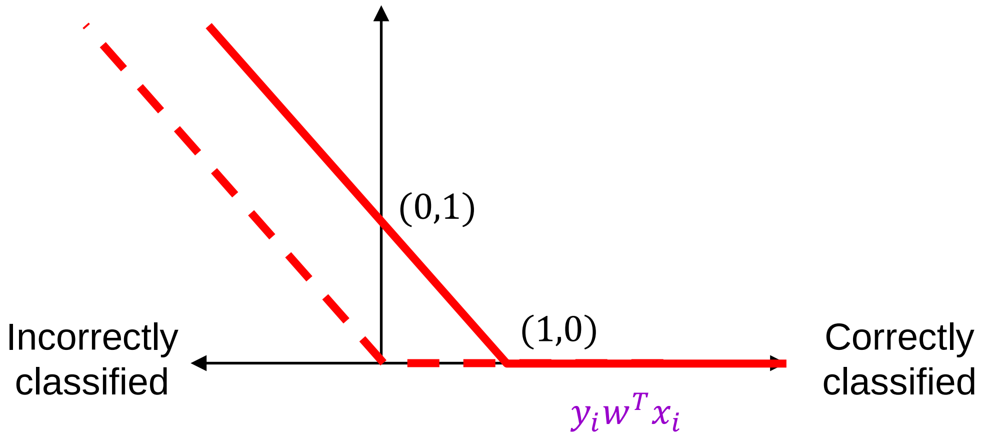


"Soft margin" formulation

- Penalize margin violations using *hinge loss*:

$$\min_w \frac{\lambda}{2}\|w\|^2 \quad + \quad \sum_{i=1}^{n} \max[0, 1 - y_i w^T x_i]$$





Recall hinge loss used by the perceptron update algorithm!

“Soft margin” formulation

- Penalize margin violations using *hinge loss*:

$$\min_w \underbrace{\frac{\lambda}{2}\|w\|^2}_{\text{Maximize margin – a.k.a. } \textit{regularization}} + \underbrace{\sum_{i=1}^{n} \max[0, 1 - y_i w^T x_i]}_{\text{Minimize misclassification loss}}$$

SGD update for SVM

$$l(w, x_i, y_i) = \frac{\lambda}{2n} ||w||^2 + \max[0, 1 - y_i w^T x_i]$$

$$\nabla l(w, x_i, y_i) = \frac{\lambda}{n} w - \mathbb{I}[y_i w^T x_i < 1] y_i x_i$$

Recall: $\frac{\partial}{\partial a} \max(0, a) = \mathbb{I}[a > 0]$

SGD update for SVM

$$l(w, x_i, y_i) = \frac{\lambda}{2n}\|w\|^2 + \max[0, 1 - y_i w^T x_i]$$

$$\nabla l(w, x_i, y_i) = \frac{\lambda}{n} w - \mathbb{I}[y_i w^T x_i < 1] y_i x_i$$

- SGD update:
  - If $y_i w^T x_i < 1$: $w \leftarrow w + \eta \left(y_i x_i - \frac{\lambda}{n} w \right)$
  - Otherwise: $w \leftarrow w - \eta \frac{\lambda}{n} w$

S. Shalev-Schwartz et al., [Pegasos: Primal Estimated sub-GrAdient SOlver for SVM](), *Mathematical Programming*, 2011

SVM vs. perceptron

- SVM loss: $l(w, x_i, y_i) = \frac{\lambda}{2n}\|w\|^2 + \max[0, 1 - y_i w^T x_i]$
- SVM update:
  - If $y_i w^T x_i < 1$: $w \leftarrow \left(1 - \eta\frac{\lambda}{n}\right) w + \eta\, y_i x_i$
  - Otherwise: $w \leftarrow \left(1 - \eta\frac{\lambda}{n}\right) w$

- Perceptron loss: $l(w, x_i, y_i) = \max[0, -y_i w^T x_i]$
- Perceptron update:
  - If $y_i w^T x_i < 0$: $w \leftarrow w + \eta\, y_i x_i$
  - Otherwise: do nothing

Acknowledgement

- Deep Learning, Stanford University
- Introduction to Deep Learning, University of Illinois at Urbana-Champaign
- Introduction to Deep Learning, Carnegie Mellon University
- Convolutional Neural Networks for Visual Recognition, Stanford University
- Natural Language Processing with Deep Learning, Stanford University