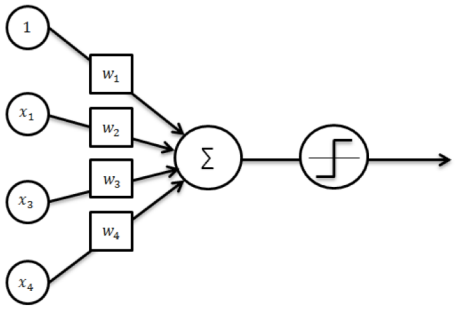
Everything you've ever wanted to know about linear classifiers (Part 1)



Slides courtesy L. Lazebnik (Univ. of Illinois CS498) and others

Image source

Outline

- Formalization of statistical learning of classifiers
- Ways to train linear classifiers
  - Linear regression
  - Perceptron training algorithm
  - Logistic regression
- Gradient descent and stochastic gradient descent

Formalization

- Let's focus on training of a *parametric model* in a supervised scenario

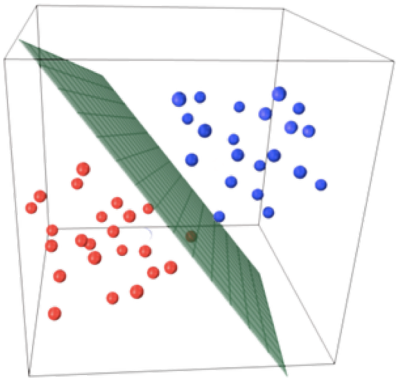


Image source

Formalization

- Given: training data $\{(x_i, y_i), i = 1, \ldots, n\}$
- Find $y = f(x)$
- S.t. f works well on *test* data

Source: Y. Liang

Formalization

- Given: training data $\{(x_i, y_i), i = 1, \dots, n\}$
- Find $y = f(x)$
- S.t. f works well on *test* data

What kinds of functions?

Source: Y. Liang

Formalization

- Given: training data $\{(x_i, y_i), i = 1, \dots, n\}$
- Find $y = f(x) \in \mathcal{H}$
- S.t. f works well on *test* data

Hypothesis class

Source: Y. Liang

Formalization

- Given: training data $\{(x_i, y_i), i = 1, \ldots, n\}$
- Find $y = f(x) \in \mathcal{H}$
- S.t. f works well 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 $y = f(x) \in \mathcal{H}$
- S.t. f works well on *test* data i.i.d. from distribution D

Source: Y. Liang

Formalization

- Given: training data $\{(x_i, y_i), i = 1, \dots, n\}$ i.i.d. from distribution D
- Find $y = f(x) \in \mathcal{H}$
- S.t. f works well on *test* data i.i.d. from distribution D

What kind of performance measure?

Source: Y. Liang

Formalization

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

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

  - 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 $\{(x_i, y_i), i = 1, \dots, n\}$ i.i.d. from distribution D
- Find $y = f(x) \in \mathcal{H}$
- 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 $y = f(x) \in \mathcal{H}$ that minimizes

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

Empirical loss

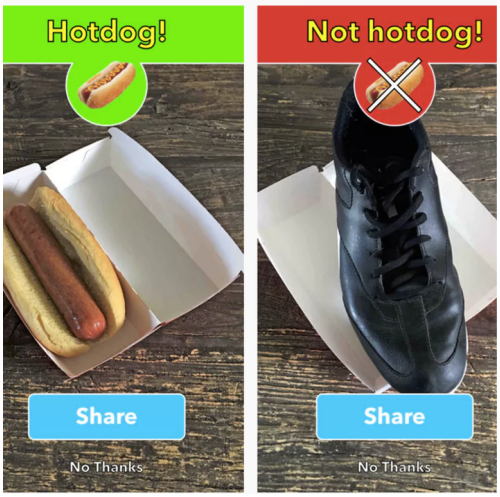
Source: Y. Liang

Supervised learning in a nutshell

1. **Collect 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 $\{(x_i, y_i), i = 1, \dots, n\}$, $y_i \in \{-1, 1\}$



Training linear classifiers

- Given: i.i.d. training data $\{(x_i, y_i), i = 1, \dots, n\}$, $y_i \in \{-1, 1\}$
- Hypothesis class: $f_w(x) = \text{sgn}(w^T x)$
- Classification with *bias*, i.e. $f_w(x) = \text{sgn}(w^T x + b)$, 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: $f_w(x) = \text{sgn}(w^T x)$
- Loss: minimizing the number of mistakes

$$\hat{L}(f_w) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}[\text{sgn}(w^T x_i) \neq y_i]$$

- Difficult to optimize directly!

Source: Y. Liang

Linear regression

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

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

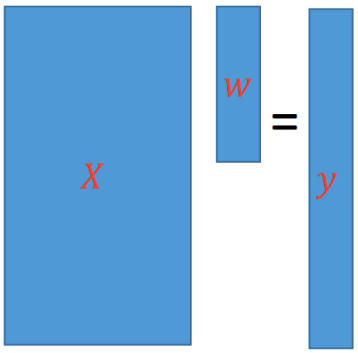
- Ignores the fact that $y \in \{-1, 1\}$ but is easy to optimize

Source: Y. Liang

Linear regression: Optimization

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

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



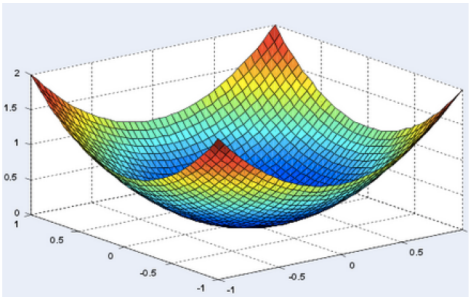
Source: Y. Liang

Linear regression: Optimization

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

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

- 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:

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

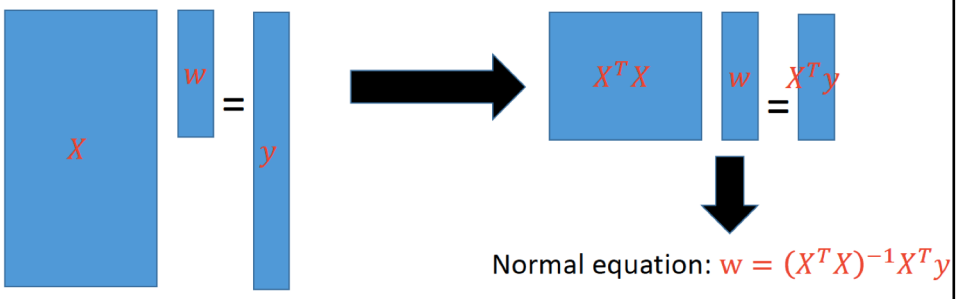
- Set gradient to zero to get the minimizer:

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

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 a least squares solution to an over-constrained system

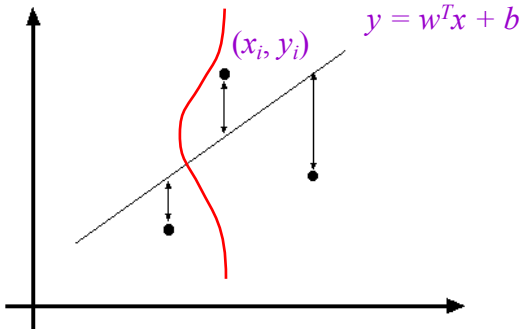


Normal equation: $w = (X^T X)^{-1} X^T y$

Source: Y. Liang

Maximum likelihood estimation

- **Interpretation of l_2 loss:** *negative log likelihood* assuming y is normally distributed with mean $f_w(x) = w^T x + b$



Maximum likelihood estimation

- Given: i.i.d. training data $\{(x_i, y_i), i = 1, \ldots, n\}$
- Let $\{P_\theta(y|x), \theta \in \Theta\}$ be a family of distributions parameterized by θ
- Maximum (conditional) likelihood estimate:

$$\theta_{ML} = \text{argmax}_\theta \prod_i P_\theta(y_i|x_i)$$
$$= \text{argmin}_\theta - \sum_i \log P_\theta(y_i|x_i)$$

Maximum likelihood estimation

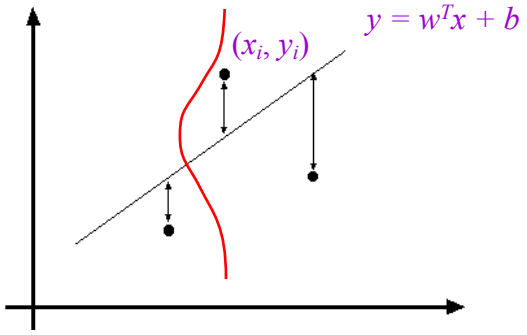
$$\theta_{ML} = \text{argmin}_\theta - \sum_i \log P_\theta(y_i|x_i)$$

- Assume $P_\theta(y|x) = \text{Normal}(y; f_\theta(x), \sigma^2)$

$$\log P_\theta(y_i|x_i) = \log\left[\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{\left(y - f_\theta(x)\right)^2}{2\sigma^2}\right]\right]$$
$$= -\frac{1}{2\sigma^2}\left(y - f_\theta(x)\right)^2 - \log\sigma - \frac{1}{2}\log(2\pi)$$
$$\theta_{ML} = \text{argmin}_\theta \sum_i \left(y_i - f_\theta(x_i)\right)^2$$

Problem with linear regression

- Interpretation of l_2 loss: *negative log likelihood* assuming y is normally distributed with mean $f_w(x) = w^T x + b$



- Does this make sense for binary classification?

Problem with linear regression

- In practice, very sensitive to outliers

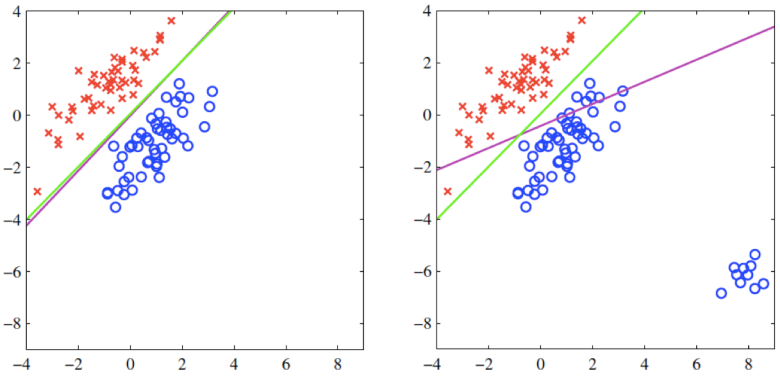
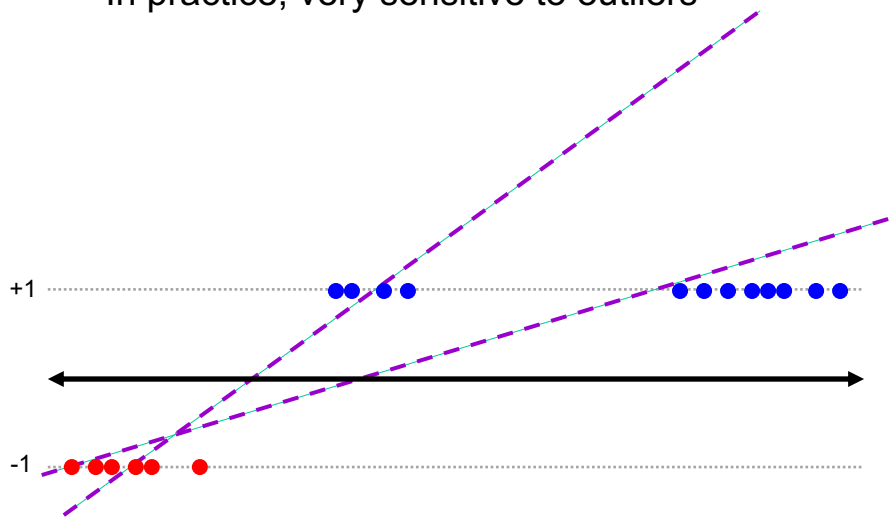


Figure 4.4 The left plot shows data from two classes, denoted by red crosses and blue circles, together with the decision boundary found by least squares (magenta curve) and also by the logistic regression model (green curve), which is discussed later in Section 4.3.2. The right-hand plot shows the corresponding 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

Problem with linear regression

- In practice, very sensitive to outliers



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 $f_w(x) = \text{sgn}(w^T x)$ that minimizes the number of mistakes

$$\hat{L}(f_w) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}[\text{sgn}(w^T x_i) \neq y_i]$$

Source: Y. Liang

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 $\text{sgn}(w^T x_i)$ does not match y_i then update weights:

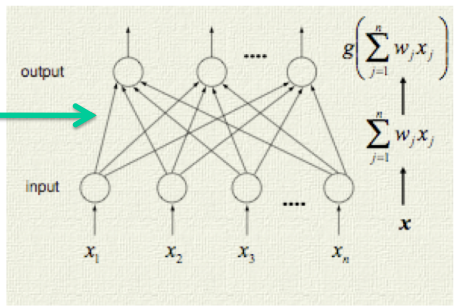
$$w \leftarrow w + \eta\, y_i x_i$$

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

Perceptron (Frank Rosenblatt, 1957)

- Consider one output unit at the time – the others are independent
- As g use a simple threshold function if $w^T x > 0$

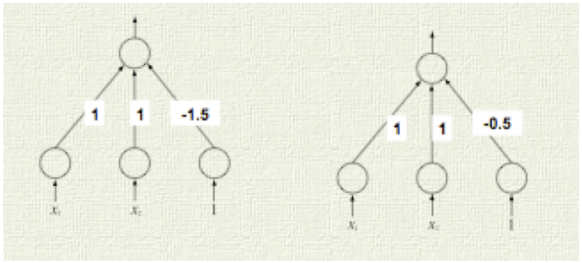
weights



$$\sum_{j=1}^{n} w_j x_j$$

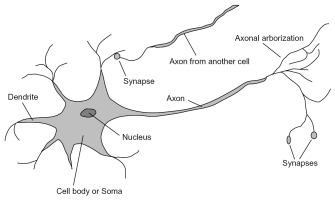
Perceptron (Frank Rosenblatt, 1957)

- Compare individual units with logic gates



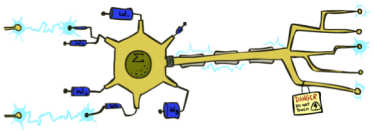
AND OR

Very loose inspiration: human neurons



Linear Classifiers

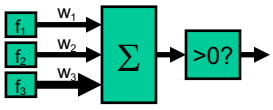
Inputs are feature values
Each feature has a weight
Sum is the activation



$$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$$

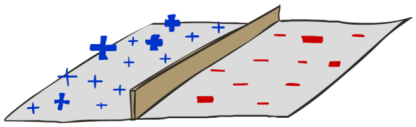
If the activation is:

- Positive, output +1
- Negative, output -1



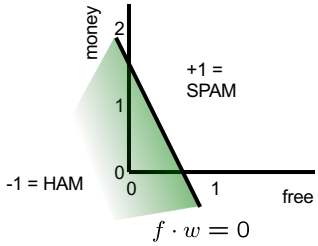
Binary Decision Rule

- Examples (features) are points
- Any weight vector is a hyperplane
- One side corresponds to Y=+1
- Other corresponds to Y=-1



w

```
BIAS  : -3
free  :  4
money :  2
...
```

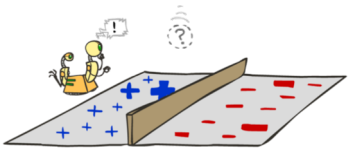


$f \cdot w = 0$

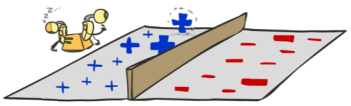
Learning: Binary Perceptron

Start with weights = 0
For each training instance:

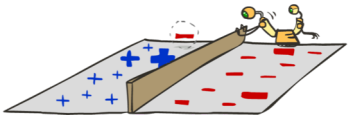
- Classify with current weights



- If correct (i.e., y=y*), no change!



- If wrong: adjust the weight vector



Learning: Binary Perceptron

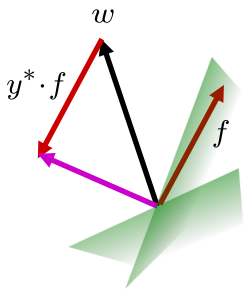
Start with weights = 0
For each training instance:

- Classify with current weights

$$y = \begin{cases} +1 & \text{if } w \cdot f(x) \geq 0 \\ -1 & \text{if } w \cdot f(x) < 0 \end{cases}$$

- If correct (i.e., y=y*), no change!
- If wrong: adjust the weight vector by adding or subtracting the feature vector. Subtract if y* is -1.

$$w = w + y^* \cdot f$$



Understanding the update rule

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

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

- The raw response of the classifier changes to

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

- If $y_i = 1$, the response $w^T x_i$ is initially *negative* and will be *increased*
- If $y_i = -1$, the response $w^T x_i$ is initially *positive* and will be *decreased*

Convergence of perceptron update rule

- **Linearly separable data:** converges to a perfect solution
  - Is the solution unique?
- **Non-separable data:** converges to a minimum-error solution assuming
  a) Examples are presented in random sequence
  b) The learning rate decays as $O(1/t)$ where t is the number of iterations and examples are presented in random sequence

Source: AIMA 3rd ed., sec. 18.6.3

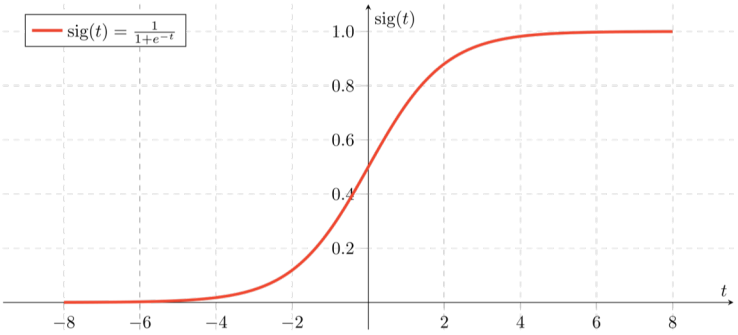
Recap so far

- We want to learn a linear classifier that minimizes the number of mistakes
- **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*
  - Gives good convergence guarantees, 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:

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

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

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

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

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

Logistic regression

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

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

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

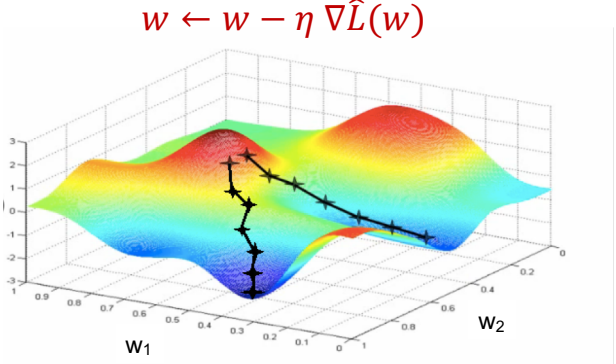
$$= -\frac{1}{n}\sum_{i=1}^{n} \log \sigma(y_i w^T x_i)$$

- 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

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

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

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

- 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*: $[g(f(x))]' = g'(f(x))f'(x)$

Gradient descent for logistic regression

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

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

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

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

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

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

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

- Full gradient update for w:

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

- Loss for a single sample:

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

- SGD update for w:

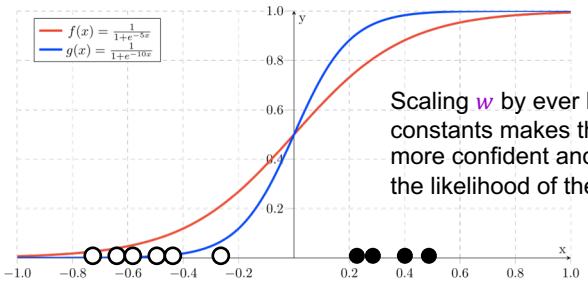
$$w \leftarrow w + \eta\, \sigma(-y_i w^T x_i) y_i x_i$$

SGD for logistic regression

- 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
  - Note, 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

SGD for logistic regression

- 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

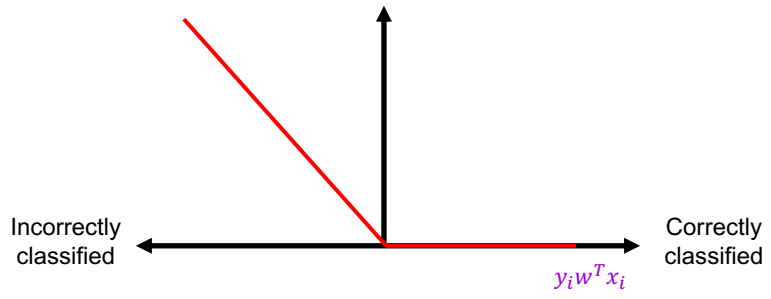
- Let's compare with perceptron update rule:

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

SGD and perceptron training algorithm

- Define *hinge loss*

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



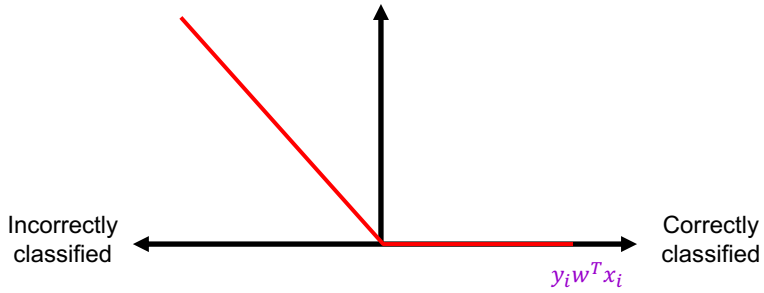
SGD and perceptron training algorithm

- Define *hinge loss*

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

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

- Note: strictly speaking the hinge loss is not differentiable, so this is called a *sub-gradient*



SGD and 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!

Linear regression update

- 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?