

# CS-E4710 Machine Learning: Supervised Methods

## Lecture 5: Linear classification

---

Juho Rousu

October 12, 2021

Department of Computer Science  
Aalto University

# Course topics

- Part I: Theory
  - Introduction
  - Generalization error analysis & PAC learning
  - Rademacher Complexity & VC dimension
  - Model selection
- Part II: Algorithms and models
  - **Linear models: perceptron, logistic regression**
  - Support vector machines
  - Kernel methods
  - Boosting
  - Neural networks (MLPs)
- Part III: Additional learning models
  - Feature learning, selection and sparsity
  - Multi-class classification
  - Preference learning, ranking, multi-output learning

# Linear classification

---

# Linear classification

- Input space  $X \subset \mathbb{R}^d$ , each  $\mathbf{x} \in X$  is a  $d$ -dimensional real-valued vector, output space:  $\mathcal{Y} = \{-1, +1\}$
- Target function or concept  $f : X \mapsto \mathcal{Y}$  assigns a (true) label to each example
- Training sample  $S = \{(x_1, y_1), \dots, (x_m, y_m)\}$ , with  $y_i = f(x_i)$  drawn from an unknown distribution  $D$
- Hypothesis class  
 $\mathcal{H} = \{\mathbf{x} \mapsto \text{sgn} \left( \sum_{j=1}^d w_j x_j + w_0 \right) \mid \mathbf{w} \in \mathbb{R}^d, w_0 \in \mathbb{R}\}$  consists of functions  $h(\mathbf{x}) = \text{sgn} \left( \sum_{j=1}^d w_j x_j + w_0 \right)$  that map each example in one of the two classes
- $\text{sgn}(a) = \begin{cases} +1, & a \geq 0 \\ -1 & a < 0 \end{cases}$  is the sign function

# Linear classifiers

Linear classifiers

$$h(\mathbf{x}) = \text{sgn} \left( \sum_{j=1}^d w_j x_j + w_0 \right) = \mathbf{w}^T \mathbf{x} + w_0$$

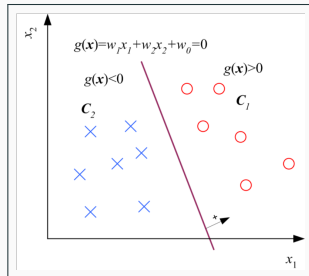
have several attractive properties

- They are fast to evaluate and takes small space to store ( $O(d)$  time and space)
- Easy to understand:  $|w_j|$  shows the importance of variable  $x_j$  and its sign tells if the effect is positive or negative
- Linear models have relatively low complexity (e.g.  $VCdim = d + 1$ ) so they can be reliably estimated from limited data

Good practise is to try a linear model before something more complicated

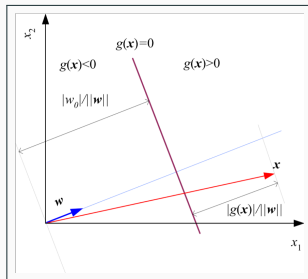
# The geometry of the linear classifier

- The points  $\{\mathbf{x} \in X | g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0 = 0\}$  define a hyperplane in  $\mathbb{R}^d$ , where  $d$  is the number of variables in  $\mathbf{x}$
- The hyperplane  $g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0 = 0$  splits the input space into two half-spaces. The linear classifier predicts +1 for points in the halfspace  $\{\mathbf{x} \in X | g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0 \geq 0\}$  and -1 for points in  $\{\mathbf{x} \in X | g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0 < 0\}$



# The geometry of the linear classifier

- $\mathbf{w}$  is the **normal vector** of the hyperplane  $\mathbf{w}^T \mathbf{x} + w_0 = 0$
- The distance of the hyperplane from the origin is  $|w_0| / \|\mathbf{w}\|$
- If  $w_0 < 0$  the hyperplane lies in the direction of  $\mathbf{w}$  from origin, otherwise it lies in the direction of  $-\mathbf{w}$
- The distance of a point  $\mathbf{x}$  from the hyperplane is  $|g(\mathbf{x})| / \|\mathbf{w}\|$
- If  $g(\mathbf{x}) > 0$ ,  $\mathbf{x}$  lies in the halfspace that is in the direction of  $\mathbf{w}$  from the hyperplane, otherwise it lies in the direction of  $-\mathbf{w}$  from the hyperplane



# Learning linear classifiers

---



# Change of representation

- Consider learning the parameters of the linear discriminant  
 $g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0$
- For presentation is is convenient to subsume term  $w_0$  into the weight vector

$$\mathbf{w} \Leftarrow \begin{bmatrix} \mathbf{w} \\ w_0 \end{bmatrix}$$

and augment all inputs with a constant 1:

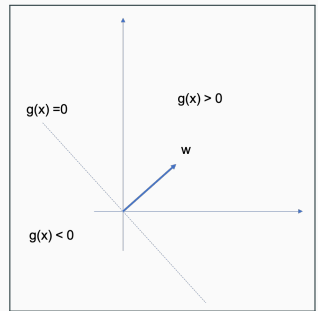
$$\mathbf{x} \Leftarrow \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}$$

- The models have the same value for the discriminant:

$$\begin{bmatrix} \mathbf{w} \\ w_0 \end{bmatrix}^T \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} = \mathbf{w}^T \mathbf{x} + w_0$$

# Geometric interpretation

- Geometrically, the hyperplane defined by the discriminant goes now through origin
- The positive points have an **acute angle** with  $\mathbf{w}$ :  $\mathbf{w}^T \mathbf{x} > 0$
- The negative points have an **obtuse angle** with  $\mathbf{w}$ :  $\mathbf{w}^T \mathbf{x} \leq 0$



## Checking for prediction errors

- When the labels are  $\mathcal{Y} = \{-1, +1\}$  for a training example  $(\mathbf{x}, y)$  we have for  $g(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$ ,

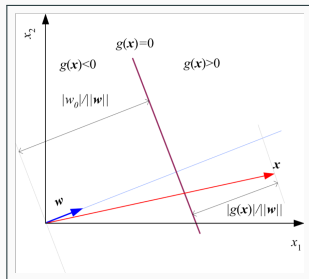
$$\text{sgn}(g(\mathbf{x})) = \begin{cases} y & \text{if } \mathbf{x} \text{ is correctly classified} \\ -y & \text{if } \mathbf{x} \text{ is incorrectly classified} \end{cases}$$

- Alternative we can just multiply with the correct label to check for misclassification:

$$yg(\mathbf{x}) = \begin{cases} \geq 0 & \text{if } \mathbf{x} \text{ is correctly classified} \\ < 0 & \text{if } \mathbf{x} \text{ is incorrectly classified} \end{cases}$$

# Margin

- The geometric margin of an example  $\mathbf{x}$  is given by  $\gamma(\mathbf{x}) = yg(\mathbf{x}) / \|\mathbf{w}\|$
- It takes into account both the distance  $|\mathbf{w}^T \mathbf{x}| / \|\mathbf{w}\|$  from the hyperplane, and whether  $\mathbf{x}$  is on the correct side of the hyperplane
- The unnormalized version of the margin is sometimes called the **functional margin**  $\gamma(\mathbf{x}) = yg(\mathbf{x})$
- Often the term **margin** is used for both variants, assuming the context makes clear which one is meant

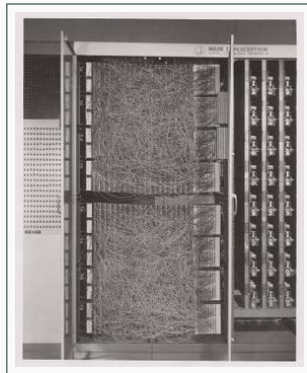


# Perceptron

---

# Perceptron

- Perceptron algorithm by Frank Rosenblatt (1956) is perhaps the first machine learning algorithm
- Its purpose was to learn a linear discriminant between two classes
- It was built in hardware and shown to be capable of performing rudimentary pattern recognition tasks
- New York Times in 1958: "the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence."  
(Source: Wikipedia)



Mark I perceptron ca. 1958 (Picture: Wikipedia)

# The perceptron algorithm

- The perceptron algorithm learns a hyperplane separating two classes

$$g(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$$

- It processes incrementally a set of training examples
  - At each step, it finds a training example  $\mathbf{x}_i$  that is incorrectly classified by the current model
  - It updates the model by adding the example to the current weight vector together with the label:  $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + y_i \mathbf{x}_i$
  - This process is continued until incorrectly predicted training examples are not found

# The perceptron algorithm

**Input:** Training set  $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m, \mathbf{x} \in \mathbb{R}^d, y \in \{-1, +1\}$   
Initialize  $\mathbf{w}^{(1)} \leftarrow (0, \dots, 0), t \leftarrow 1, stop \leftarrow FALSE$   
**repeat**  
    **if** exists  $i$ , s.t.  $y_i \mathbf{w}^{(t)T} \mathbf{x}_i \leq 0$  **then**  
         $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + y_i \mathbf{x}_i$   
    **else**  
         $stop \leftarrow TRUE$   
    **end if**  
     $t \leftarrow t + 1$   
**until**  $stop$



# Understanding the update rule

- Let us examine the update rule

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + y_i \mathbf{x}_i$$

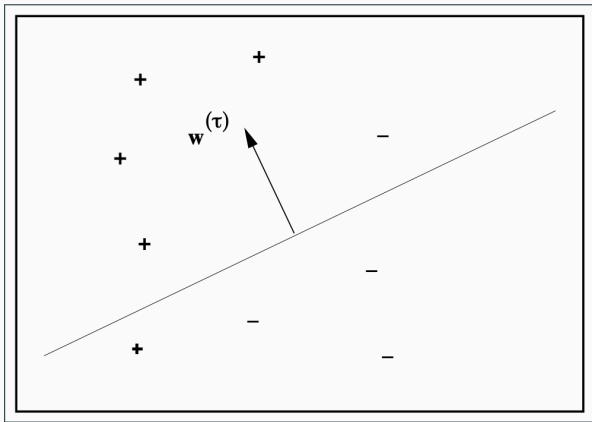
- We can see that the margin of the example  $(\mathbf{x}_i, y_i)$  increases after the update

$$\begin{aligned} y_i g^{(t+1)}(\mathbf{x}_i) &= y_i \mathbf{w}^{(t+1)T} \mathbf{x}_i = y_i (\mathbf{w}^{(t)} + y_i \mathbf{x}_i)^T \mathbf{x}_i \\ &= y_i \mathbf{w}^{(t)T} \mathbf{x}_i + y_i^2 \mathbf{x}_i^T \mathbf{x}_i = y_i g^{(t)}(\mathbf{x}_i) + \|\mathbf{x}_i\|^2 \\ &\geq y_i g^{(t)}(\mathbf{x}_i) \end{aligned}$$

- Note that this does not guarantee that  $y_i g^{(t+1)}(\mathbf{x}_i) > 0$  after the update, further updates may be required to achieve that

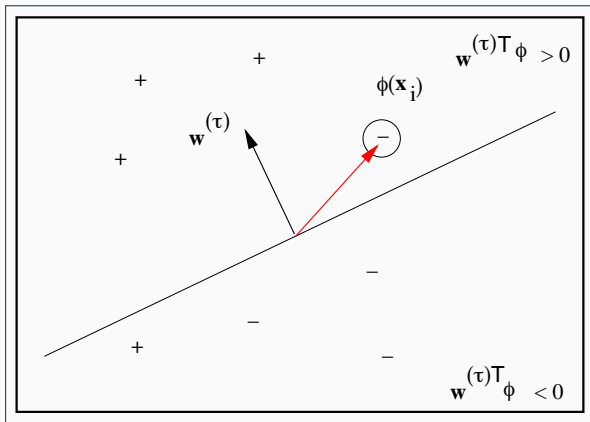
# Perceptron animation

- Assume  $\mathbf{w}^{(t)}$  has been found by running the algorithm for  $t$  steps
- We notice two misclassified examples



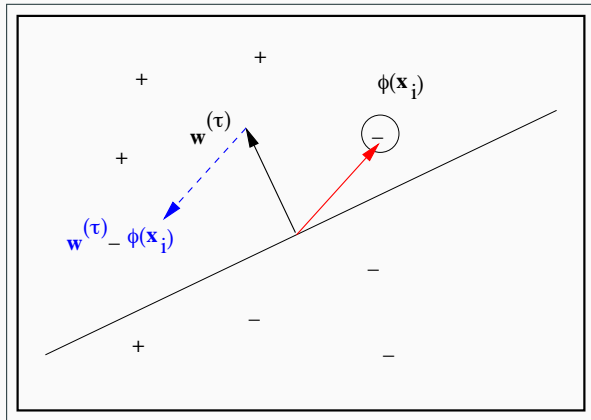
# Perceptron animation

- Select the misclassified example  $(\phi(\mathbf{x}_i), -1)$
- Note:  $\phi(\mathbf{x}_i)$  is here some transformation of  $\mathbf{x}_i$  e.g. with some basis functions but it could be identity  $\phi(\mathbf{x}) = \mathbf{x}$



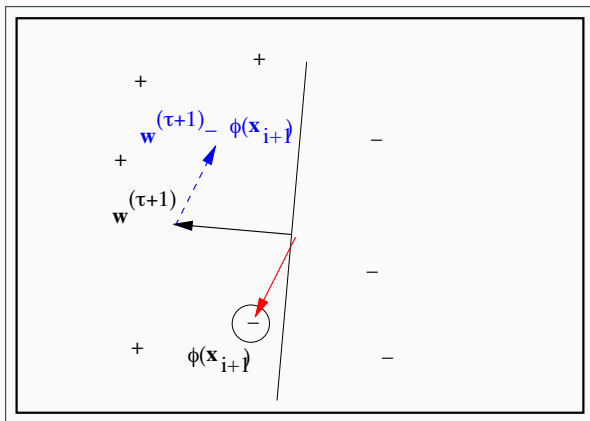
# Perceptron animation

- Update the weight vector:  $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + y_i \phi(\mathbf{x}_i)$



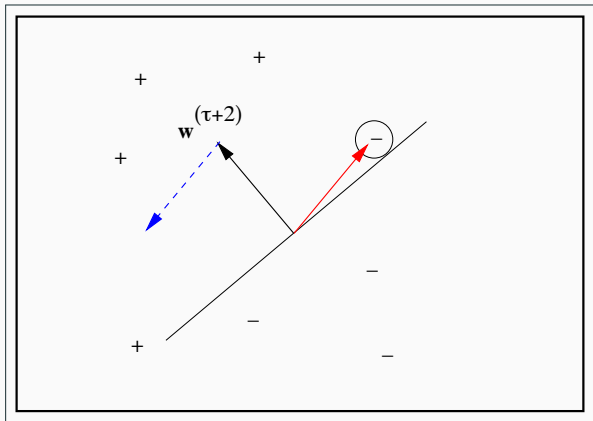
# Perceptron animation

- The update tilts the hyperplane to make the example "more correct", i.e. more negative
- We repeat the process by finding the next misclassified example  $\phi(\mathbf{x}_{i+1})$  and update:  $\mathbf{w}^{(t+2)} = \mathbf{w}^{(t+1)} + y_{i+1}\phi(\mathbf{x}_{i+1})$



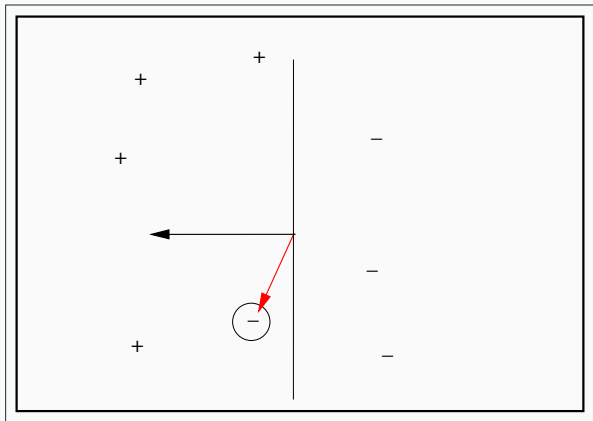
# Perceptron animation

- Next iteration



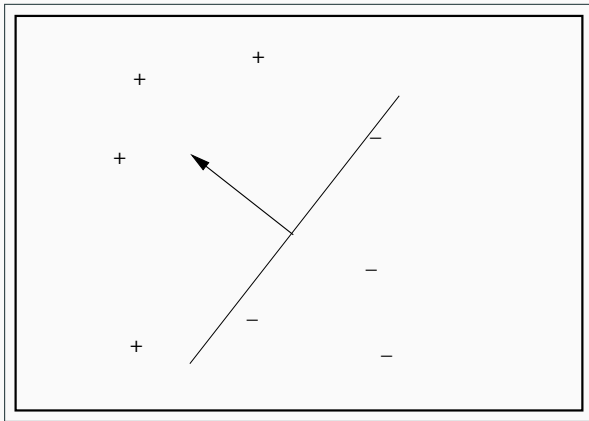
# Perceptron animation

- Next iteration



# Perceptron animation

- Finally we have found a hyperplane that correctly classify the training points
- We can stop the iteration and output the final weight vector





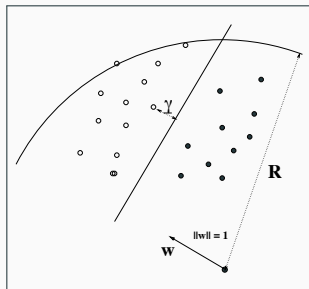
# Convergence of the perceptron algorithm

- The perceptron algorithm can be shown to eventually converge to a consistent hyperplane if the two classes are **linearly separable**, that is, if there exists a hyperplane that separates the two classes
- Theorem (Novikoff):
  - Let  $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$  be a linearly separable training set.
  - Let  $R = \max_{\mathbf{x}_i \in S} \|\mathbf{x}_i\|$ .
  - Let there exist a vector  $\mathbf{w}_*$  that satisfies  $\|\mathbf{w}_*\| = 1$  and  $y_i \mathbf{w}_*^T \mathbf{x}_i + b_{opt} \geq \gamma$  for  $i = 1 \dots, m$ .
  - Then the perceptron algorithm will stop after at most  $t \leq \left(\frac{2R}{\gamma}\right)^2$  iterations and output a weight vector  $\mathbf{w}^{(t)}$  for which  $y_i \mathbf{w}^{(t)} \mathbf{x}_i \geq 0$  for all  $i = 1 \dots, m$

# Convergence of the perceptron algorithm

The number of iterations in the bound  $t \leq (\frac{2R}{\gamma})^2$  depend on:

- $\gamma$ : The largest achievable geometric margin so that all training examples have at least that margin
- $R$ : The smallest radius of the  $d$ -dimensional ball that encloses the training data



Intuitively: how large the margin is relative to the distances of the training points

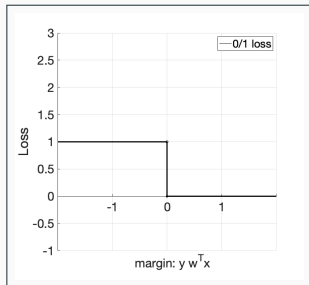
# The non-separable case

- Perceptron algorithm does not stop on a non-separable training set, since there will always be a misclassified example that causes an update
- In general, finding a hyperplane that minimizes the number of classification errors is computationally hard (NP-hard to minimize empirical error)

# The non-separable case

The main source of difficulty is the "step function" shape of the zero-one loss function

$$L(y, \mathbf{w}^T \mathbf{x}) = \begin{cases} 1 & \text{if } y\mathbf{w}^T \mathbf{x} < 0 \\ 0 & \text{otherwise} \end{cases}$$

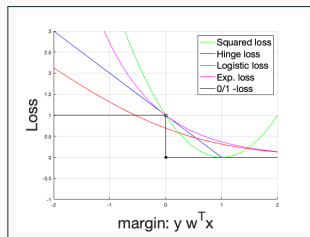


- It is non-differentiable, so cannot optimize using gradient approaches
- It is non-convex, so optimizer susceptible to fall in local minima

# Surrogate loss functions for classification

There are multiple **surrogate** losses that are convex and differentiable upper bounds to zero-one loss

- Squared loss - used for regression, not optimal for classification
- Hinge loss - used in Support vector machines (Lecture 6)
- Exponential loss - used in Boosting
- **Logistic loss - used in Logistic regression**



# Logistic regression

---

# Logistic regression

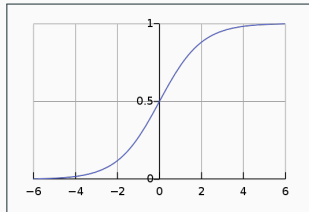
Logistic regression is a classification technique (despite the name)

- it gets its name from the logistic function

$$\phi_{\text{logistic}}(z) = \frac{1}{1 + \exp(-z)} = \frac{\exp(z)}{1 + \exp(z)}$$

that maps a real valued input  $z$  onto the interval  $0 < \phi_{\text{logistic}}(z) < 1$

- The function is an example of **sigmoid** ("S" shaped) functions



# Logistic function: a probabilistic interpretation

- The logistic function  $\phi_{\text{logistic}}(z)$  is the inverse of **logit function**
- The logit function is the logarithm of **odds ratio** of probability  $p$  of and event happening vs. the probability of the event not happening,  $1 - p$ ;

$$z = \text{logit}(p) = \log \frac{p}{1-p} = \log p - \log(1-p)$$

- Thus the logistic function

$$\phi_{\text{logistic}}(z) = \text{logit}^{-1}(z) = \frac{1}{1 + \exp(-z)}$$

answer the question "what is the probability  $p$  that gives the log odds ratio of  $z$ "



# Logistic regression

- Logistic regression model assumes a underlying conditional probability:

$$Pr(y|\mathbf{x}) = \frac{\exp(+\frac{1}{2}y\mathbf{w}^T\mathbf{x})}{\exp(+\frac{1}{2}y\mathbf{w}^T\mathbf{x}) + \exp(-\frac{1}{2}y\mathbf{w}^T\mathbf{x})}$$

where the denominator normalizes the right-hand side to be between zero and one.

- Dividing the numerator and denominator by  $\exp(+\frac{1}{2}y\mathbf{w}^T\mathbf{x})$  reveals the logistic function

$$Pr(y|\mathbf{x}) = \phi_{\text{logistic}}(y\mathbf{w}^T\mathbf{x}) = \frac{1}{1 + \exp(-y\mathbf{w}^T\mathbf{x})}$$

- The margin  $z = y\mathbf{w}^T\mathbf{x}$  is thus interpreted as the log odds ratio of label  $y$  vs. label  $-y$  given input  $\mathbf{x}$ :

$$y\mathbf{w}^T\mathbf{x} = \log \frac{Pr(y|\mathbf{x})}{Pr(-y|\mathbf{x})}$$

# Logistic loss

- Consider the maximization of the likelihood of the observed input-output in the training data:

$$\mathbf{w}^* = \operatorname{argmax}_{\mathbf{w}} \prod_{i=1}^m P(y_i | \mathbf{x}_i) = \operatorname{argmax}_{\mathbf{w}} \prod_{i=1}^m \frac{1}{1 + \exp(-y_i \mathbf{w}^T \mathbf{x}_i)}$$

- Since the logarithm is monotonically increasing function, we can take the logarithm to obtain an equivalent objective:

$$\sum_{i=1}^m \log Pr(y_i | \mathbf{x}_i) = - \sum_{i=1}^m \log(1 + \exp(-y_i \mathbf{w}^T \mathbf{x}_i))$$

- The right-hand side is the **logistic loss**:

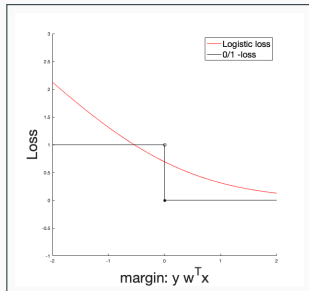
$$L_{\text{logistic}}(y, \mathbf{w}^T \mathbf{x}) = \log(1 + \exp(-y \mathbf{w}^T \mathbf{x}))$$

- Minimizing the logistic loss correspond maximizing the likelihood of the training data

# Geometric interpretation of Logistic loss

$$L_{\text{logistic}}(y, \mathbf{w}^T \mathbf{x}) = \log(1 + \exp(-y\mathbf{w}^T \mathbf{x}))$$

- Logistic loss is convex and differentiable
- It is a monotonically decreasing function of the margin  $y\mathbf{w}^T \mathbf{x}$
- The loss changes fast when the margin is highly negative  $\implies$  penalization of examples far in the incorrect halfspace
- It changes slowly for highly positive margins  $\implies$  does not give extra bonus for being very far in the correct halfspace



# Logistic regression optimization problem

- To train a logistic regression model, we need to find the  $\mathbf{w}$  that minimizes the average logistic loss  $J(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^m L_{\text{logistic}}(y_i, \mathbf{w}^T \mathbf{x}_i)$  over the training set:

$$\min J(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^m \log(1 + \exp(-y_i \mathbf{w}^T \mathbf{x}_i))$$

*w.r.t* parameters  $\mathbf{w} \in \mathbb{R}^d$

- The function to be minimized is continuous and differentiable
- However, it is a non-linear function so it is not easy to find the optimum directly (e.g. unlike in linear regression)
- We will use **stochastic gradient descent** to incrementally step towards the direction where the objective decreases fastest, the **negative gradient**

- The gradient is the vector of partial derivatives of the objective function  $J(\mathbf{w})$  with respect to all parameters  $w_j$

$$\nabla J(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^m \nabla J_i(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^m \left[ \frac{\partial}{\partial w_1} J_i(\mathbf{w}), \dots, \frac{\partial}{\partial w_d} J_i(\mathbf{w}) \right]^T$$

- Compute the gradient by using the regular rules for differentiation.  
For the logistic loss we have

$$\begin{aligned} \frac{\partial}{\partial w_j} J_i(\mathbf{w}) &= \frac{\partial}{\partial w_j} \log(1 + \exp(-y_i \mathbf{w}^T x_i)) = \frac{\exp(-y_i \mathbf{w}^T x_i)}{1 + \exp(-y_i \mathbf{w}^T x_i)} \cdot (-y_i x_{ij}) \\ &= -\frac{1}{1 + \exp(y_i \mathbf{w}^T x_i)} y_i x_{ij} = -\phi_{\text{logistic}}(-y_i \mathbf{w}^T x_i) y_i x_{ij} \end{aligned}$$

# Stochastic gradient descent

- We collect the partial derivatives with respect to a single training example into a vector:

$$\nabla J_i(\mathbf{w}) = \begin{bmatrix} -(\phi_{\text{logistic}}(-y_i \mathbf{w}^T \mathbf{x}_i) y_i) \cdot x_{i1} \\ \vdots \\ -(\phi_{\text{logistic}}(-y_i \mathbf{w}^T \mathbf{x}_i) y_i) \cdot x_{ij} \\ \vdots \\ -(\phi_{\text{logistic}}(-y_i \mathbf{w}^T \mathbf{x}_i) y_i) \cdot x_{id} \end{bmatrix} = -\phi_{\text{logistic}}(-y_i \mathbf{w}^T \mathbf{x}_i) y_i \cdot \mathbf{x}_i$$

- The vector  $-\nabla J_i(\mathbf{w})$  gives the update direction that fastest decreases the loss on training example  $(\mathbf{x}_i, y_i)$

# Stochastic gradient descent

- Evaluating the full gradient

$$\nabla J(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^m \nabla J_i(\mathbf{w}) = -\frac{1}{m} \sum_{i=1}^m \phi_{\text{logistic}}(-y_i \mathbf{w}^T \mathbf{x}_i) y_i \cdot \mathbf{x}_i$$

is costly since we need to process all training examples

- Stochastic gradient descent instead uses a series of smaller updates that depend on single randomly drawn training example  $(\mathbf{x}_i, y_i)$  at a time
- The update direction is taken as  $-\nabla J_i(\mathbf{w})$
- Its expectation is the full negative gradient:

$$-\mathbb{E}_{i=1, \dots, m} [\nabla J_i(\mathbf{w})] = -\nabla J(\mathbf{w})$$

- Thus on average, the updates match that of using the full gradient

# Stochastic gradient descent algorithm

Initialize  $\mathbf{w} = 0$

**repeat**

    Draw a training example  $(x_i, y_i)$  uniformly at random

    Compute the update direction corresponding to the training example:

$$\Delta \mathbf{w} = -\nabla J_i(\mathbf{w})$$

    Determine a stepsize  $\eta$

    Update  $\mathbf{w} = \mathbf{w} - \eta \nabla J_i(\mathbf{w})$

**until** stopping criterion satisfied

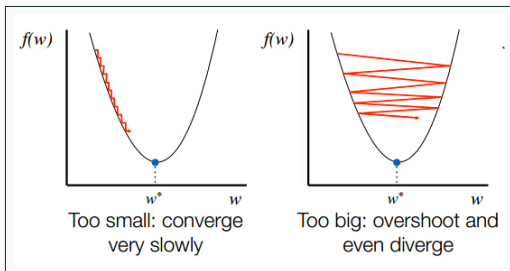
Output  $\mathbf{w}$



# Stepsize selection

Consider the SGD update:  $\mathbf{w} = \mathbf{w} - \eta \nabla J_i(\mathbf{w})$

- The stepsize parameter  $\eta$ , also called the **learning rate** is a critical one for convergence to the optimum value
- One uses small constant stepsize, the initial convergence may be unnecessarily slow
- Too large stepsize may cause the method to continually overshoot the optimum.



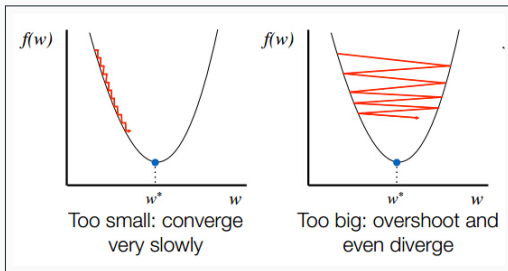
# Diminishing stepsize

- Initially larger but diminishing stepsize is one option:

$$\eta^{(t)} = \frac{1}{\alpha t}$$

for some  $\alpha > 0$ , where  $t$  is the iteration counter

- Caution: In practice, finding a good value for parameter  $\alpha$  requires experimenting with several values



Source: <https://dunglai.github.io/2017/12/21/gradient-descent/>

# Summary

- Linear classification models are an important class of machine learning models, they are used as standalone models and appear as building blocks of more complicated, non-linear models
- Perceptron is a simple algorithm to train linear classifiers on linearly separable data
- Logistic regression is a classification method that can be interpreted as maximizing odds ratios of conditional class probabilities
- Stochastic gradient descent is an efficient optimization method for large data that is nowadays very widely used