CS-E4710 Machine Learning: Supervised Methods

Lecture 5: Linear classification

Juho Rousu

October 3, 2023

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 regession**
  - Support vector machines
  - Kernel methods
  - Boosting
  - Neural networks (MLPs)
- Part III: Additional topics
  - Feature learning, selection and sparsity
  - Multi-class classification
  - Preference learning, ranking

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 $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$, with $y_i = f(x_i)$ drawn from an unknown distribution D
- Hypothesis class $\mathcal{H} = \{\mathbf{x} \mapsto \text{sgn}\left(\sum_{j=1}^{d} w_j x_j + w_0\right) | \mathbf{w} \in \mathbb{R}^d, w_0 \in \mathbb{R}\}$ 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

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

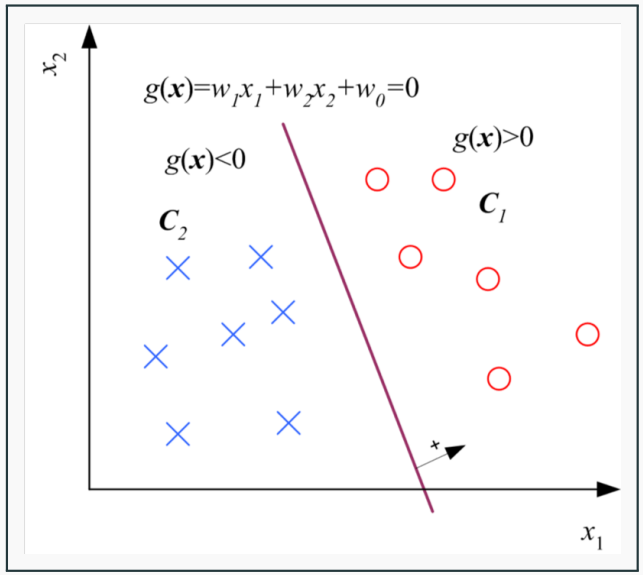
have several attractive properties

- 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

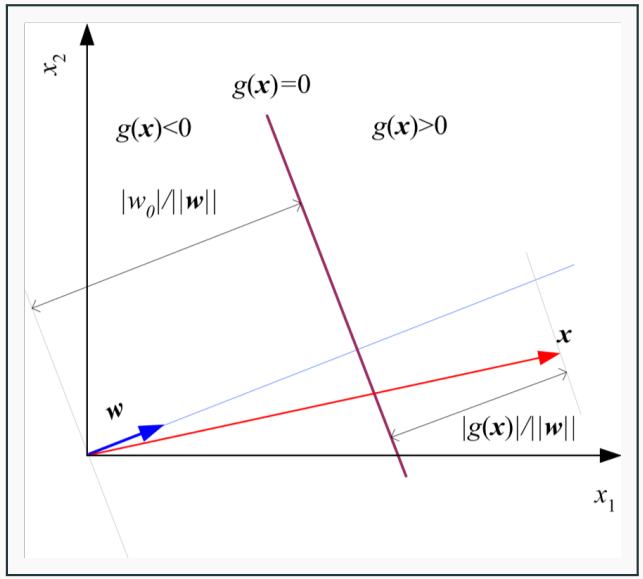
The geometry of the linear classifier

- 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 $\{\mathbf{x} \in X | g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0 < 0\}$



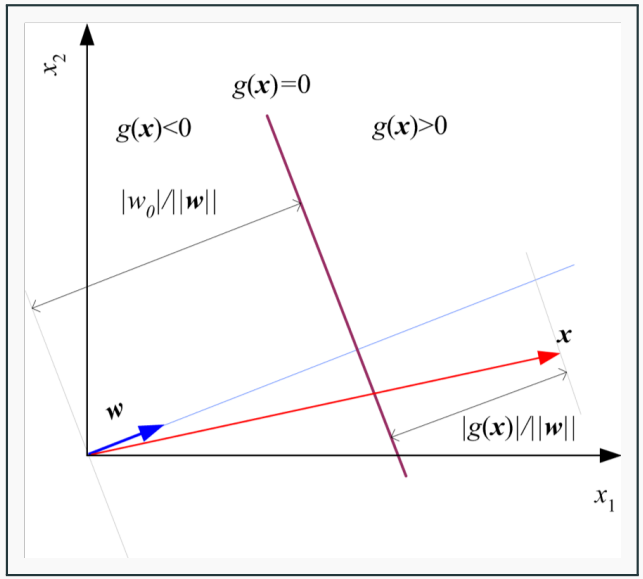
The geometry of the linear classifier

- $\mathbf{w}$ is the **normal vector** of the hyperplane $g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0 = 0$
- The distance of the hyperplane from the origin is $|w_0| / \|\mathbf{w}\|$, where $\|\mathbf{w}\| = \sqrt{\sum_j w_j^2}$ denotes the Euclidean norm
- 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 geometry of the linear classifier

- The value $g(\mathbf{x}')$ tells where $\mathbf{x}'$ lies in relation to the hyperplane:
  - $g(\mathbf{x}') > 0$: $\mathbf{x}'$ lies in the halfspace that is in the direction of $\mathbf{w}$ from the hyperplane
  - $g(\mathbf{x}) = 0$: $\mathbf{x}'$ lies on the hyperplane
  - $g(\mathbf{x}') < 0$: $\mathbf{x}'$ lies in the direction of $-\mathbf{w}$ from the hyperplane
- The distance of a point $\mathbf{x}'$ from the hyperplane $g(\mathbf{x}) = 0$ is $|g(\mathbf{x}')| / \|\mathbf{w}\|$



Learning linear classifiers

Change of representation

- Consider the parameters $(\mathbf{w}, w_0)$ of the linear function $g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0$
- 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:

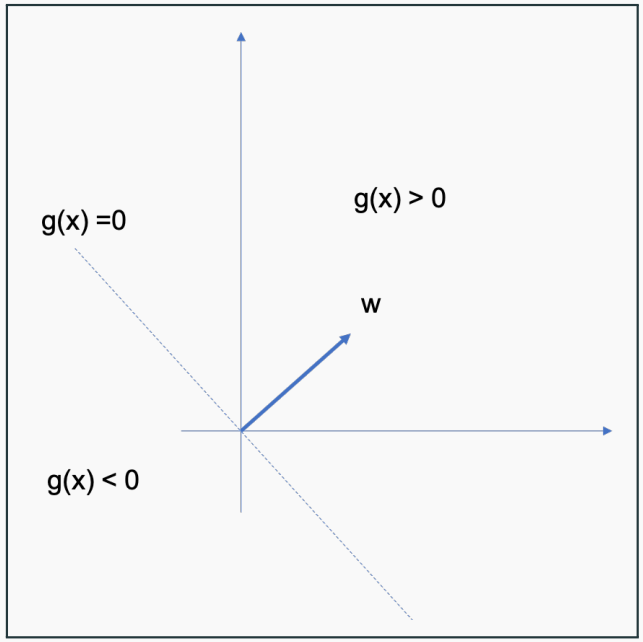
$$\mathbf{x} \Leftarrow \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}$$

- 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 $\mathbf{w}$: $\mathbf{w}^T\mathbf{x} <= 0$



Checking for prediction errors

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

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

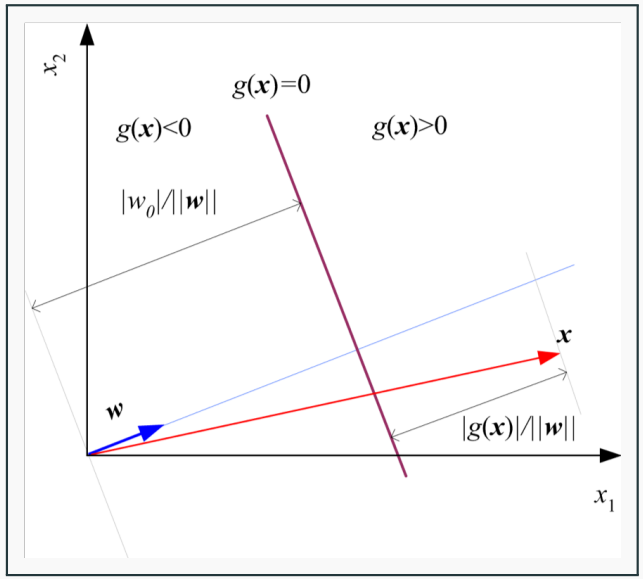
- Alternative we can just multiply with the correct label to check for misclassification:

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

Margin

一个好的超平面要尽可能远离两类别的样本，从而减少分类风险

- The geometric margin of a labeled example $(\mathbf{x}, y)$ is given by $\gamma(\mathbf{x}) = yg(\mathbf{x})/\left\|\mathbf{w}\right\|$
- It takes into account both the distance $|\mathbf{w}^T\mathbf{x}|/\left\|\mathbf{w}\right\|$ 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



这段话讲的是几何间隔的定义和含义。几何间隔是一种衡量一个有标签的样本 (x,y) 到超平面的距离和正确性的指标，它的公式是

$$\gamma(x) = \frac{yg(x)}{\|w\|}$$

其中，y 是样本的真实标签，g(x) 是超平面的线性函数，w 是超平面的权重向量，‖w‖ 是 w 的欧几里得范数。几何间隔的意义是，它既考虑了样本到超平面的距离 |wT x|/ ‖w‖，又考虑了样本是否在超平面的正确一侧。如果样本被正确分类，那么 y g(x) > 0，几何间隔为正；如果样本被错误分类，那么 y g(x) < 0，几何间隔为负；如果样本在超平面上，那么 y g(x) = 0，几何间隔为零。

几何间隔的一个变体是未归一化的间隔，有时也称为函数间隔，它的公式是

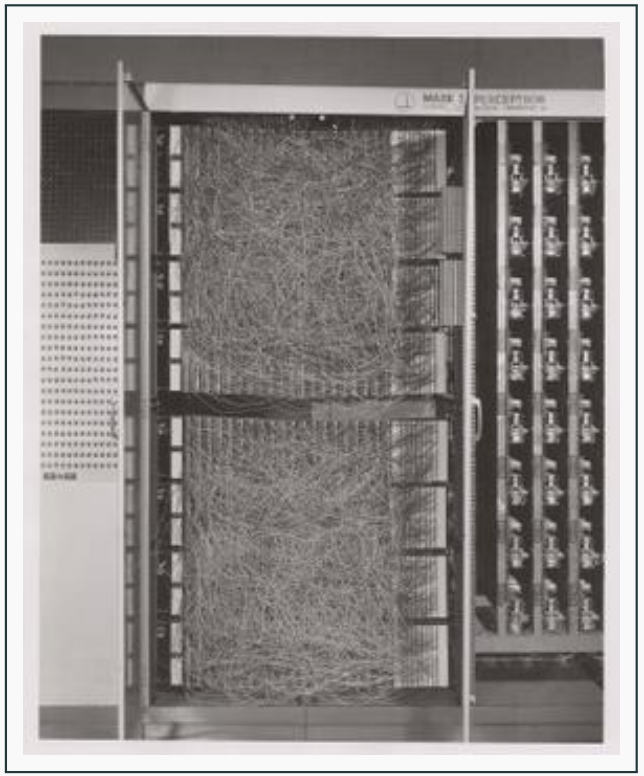
$$\gamma(x) = yg(x)$$

它只考虑了样本的正确性，而不考虑了样本到超平面的距离。函数间隔的缺点是，它依赖于超平面的参数的大小，如果将超平面的参数乘以一个常数，函数间隔也会相应地变化，但是超平面的位置和方向并没有改变。因此，函数间隔不能反映样本到超平面的真实距离，而几何间隔可以。

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 a 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

次随机选取一个错误点，然后根据它来更新权重和偏置。这样，权重和偏置就会不断地调整，直到没有错误点为止。这个过程可以用下面的伪代码来表示：

```
初始化权重和偏置为零
重复以下步骤，直到没有错误点：
    随机选取一个数据点(x, y)
    计算感知机的输出y_hat = sign(w * x + b)
    如果y_hat != y，说明是一个错误点，那么就更新权重和偏置：
        w = w + eta * y * x
        b = b + eta * y
    否则，说明是一个正确点，那么就不更新权重和偏置
```

其中，`sign`是符号函数，`eta`是学习率，`*`表示向量的点积。你可以把这个过程想象成一种试错的方法，每次找到一个错误点，就对权重和偏置做一些微小的改变，使得它更接近于正确的分类超平面。

如果你想了解更多关于感知机算法如何更新权重的内容，你可以参考以下链接：

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 $w^{(1)} \leftarrow (0, \ldots, 0)$, $t \leftarrow 1$, $stop \leftarrow FALSE$

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

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

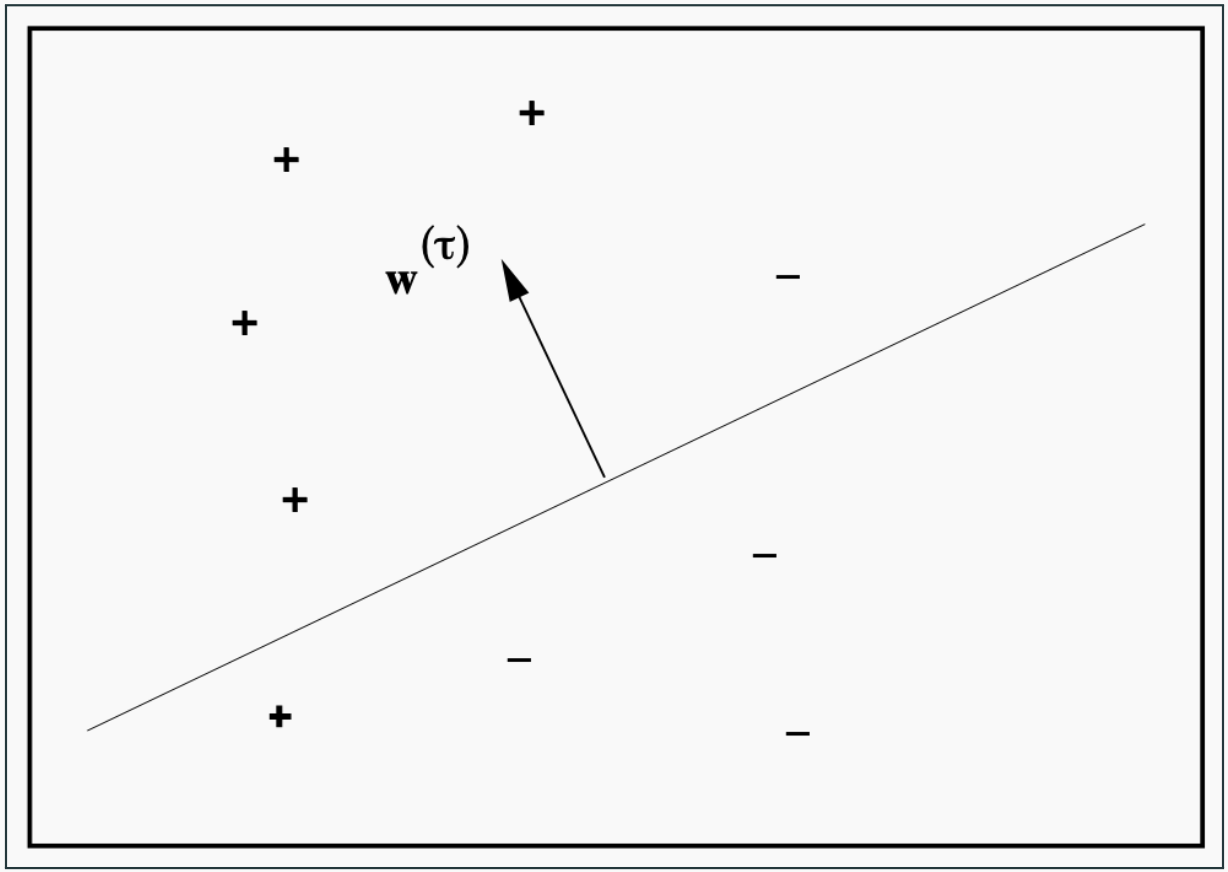
- We can see that the margin of the example $(\mathbf{x}_i, y_i)$ increases after the update

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

- 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

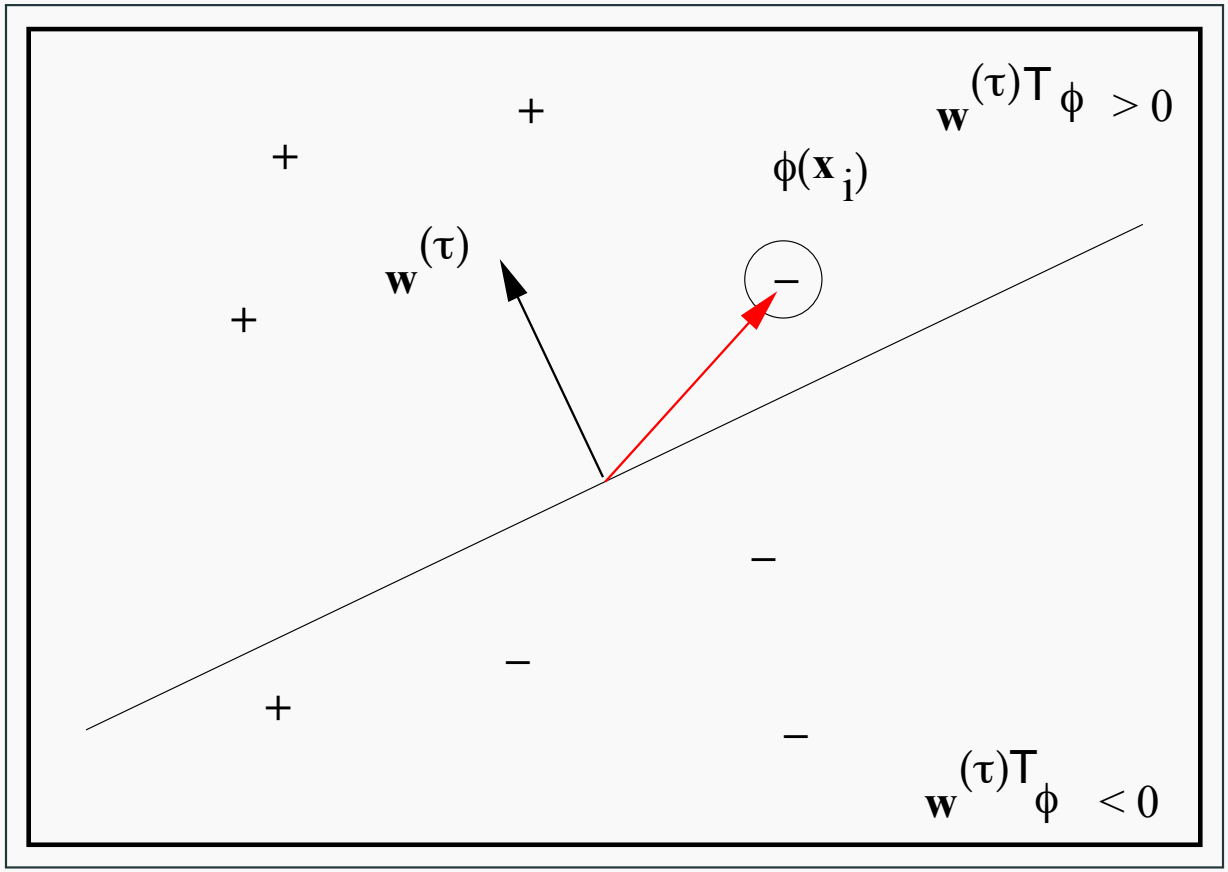
Perceptron animation

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



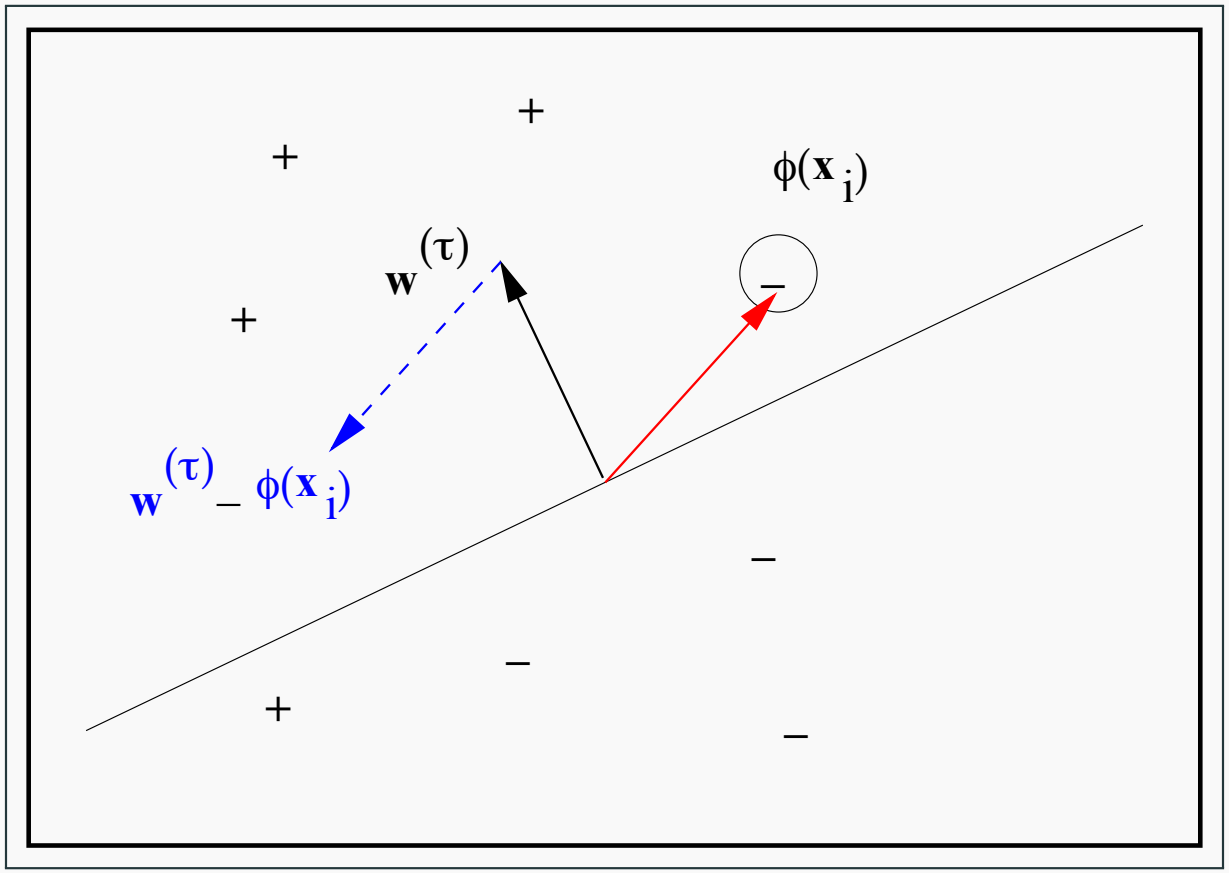
Perceptron animation

- 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 $\phi(\mathbf{x}) = \mathbf{x}$



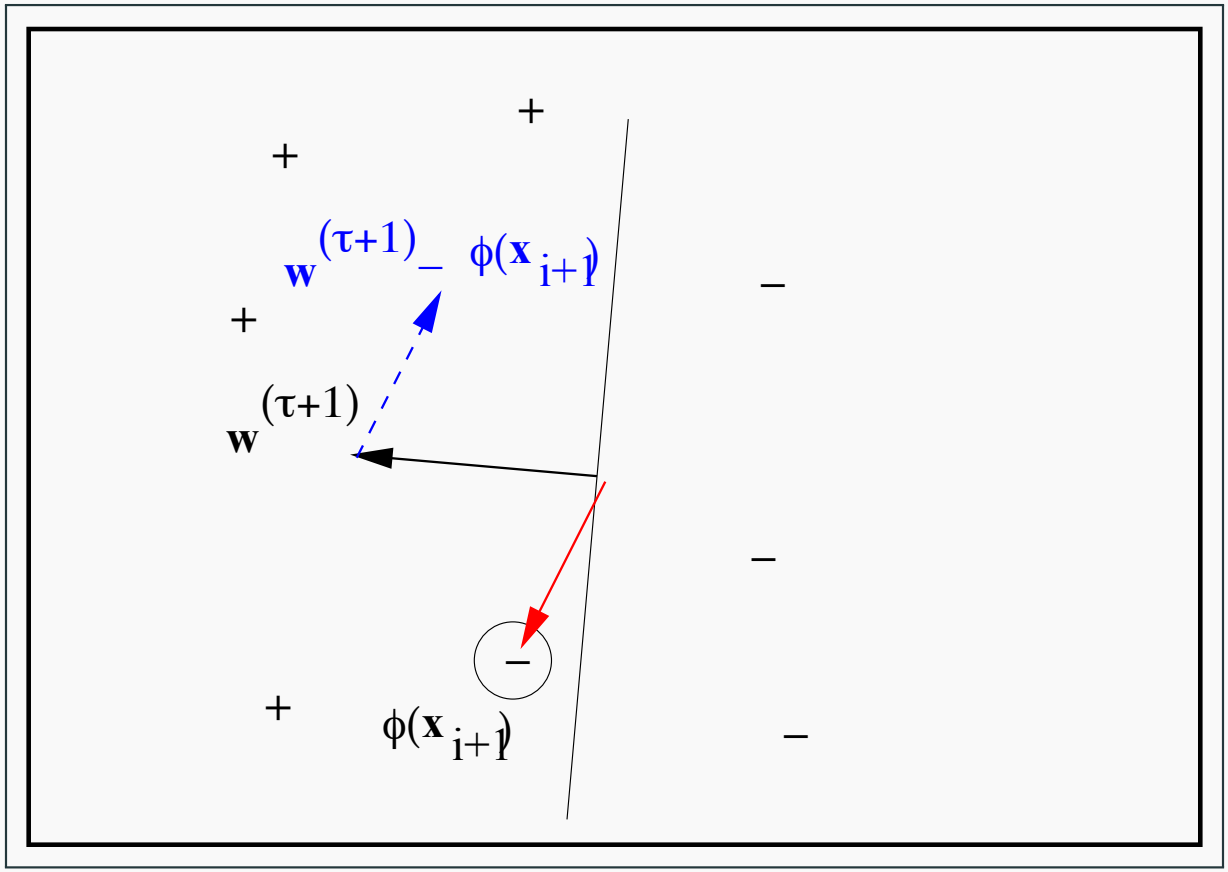
Perceptron animation

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



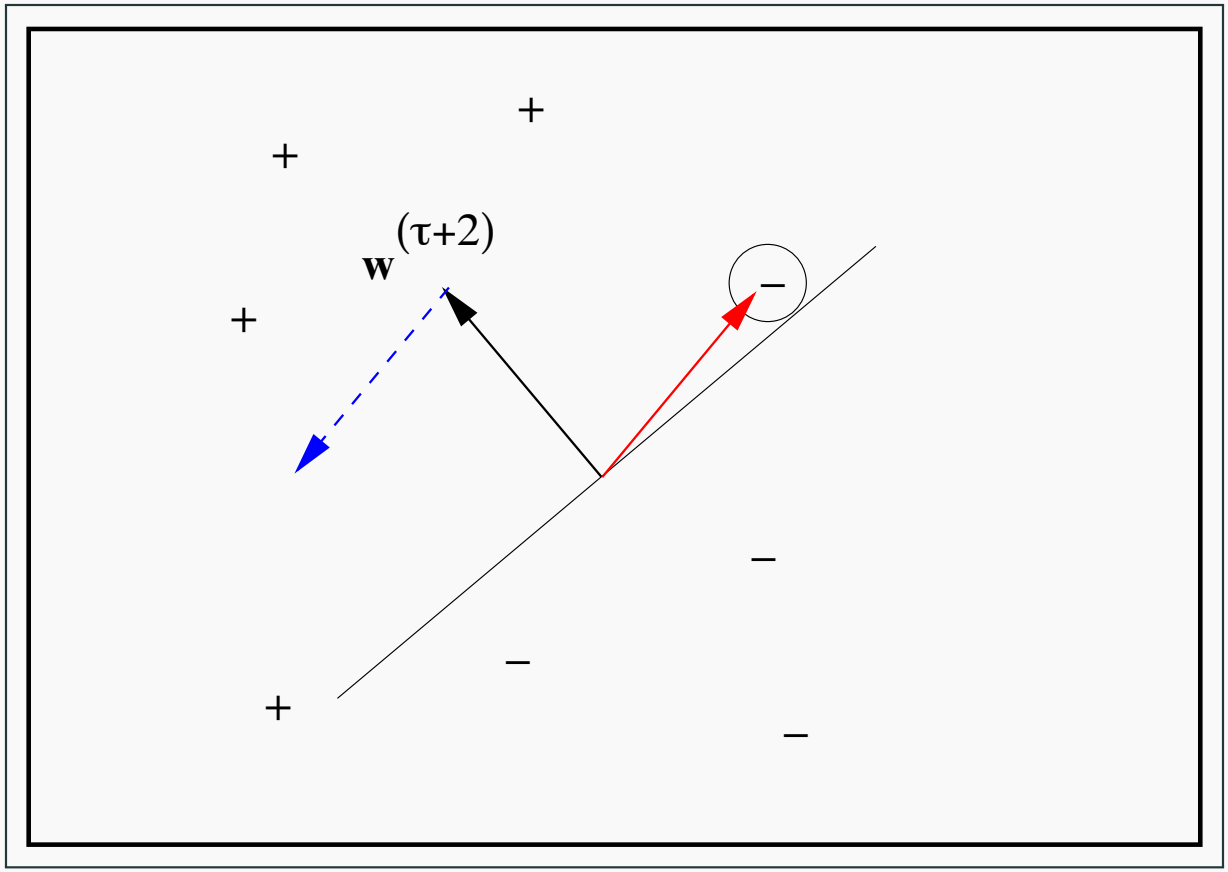
Perceptron animation

- 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: $\mathbf{w}^{(t+2)} = \mathbf{w}^{(t+1)} + y_{i+1}\phi(\mathbf{x}_{i+1})$



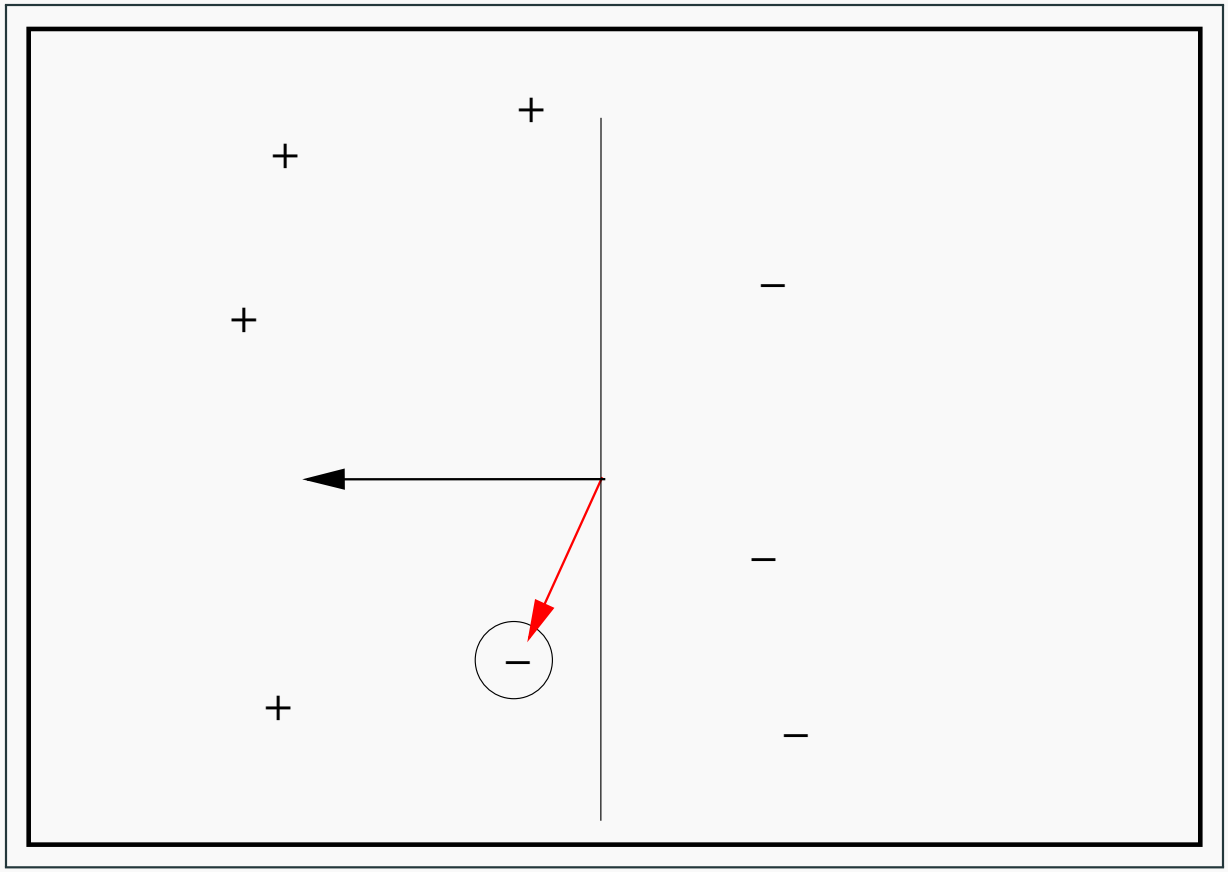
Perceptron animation

- Next iteration



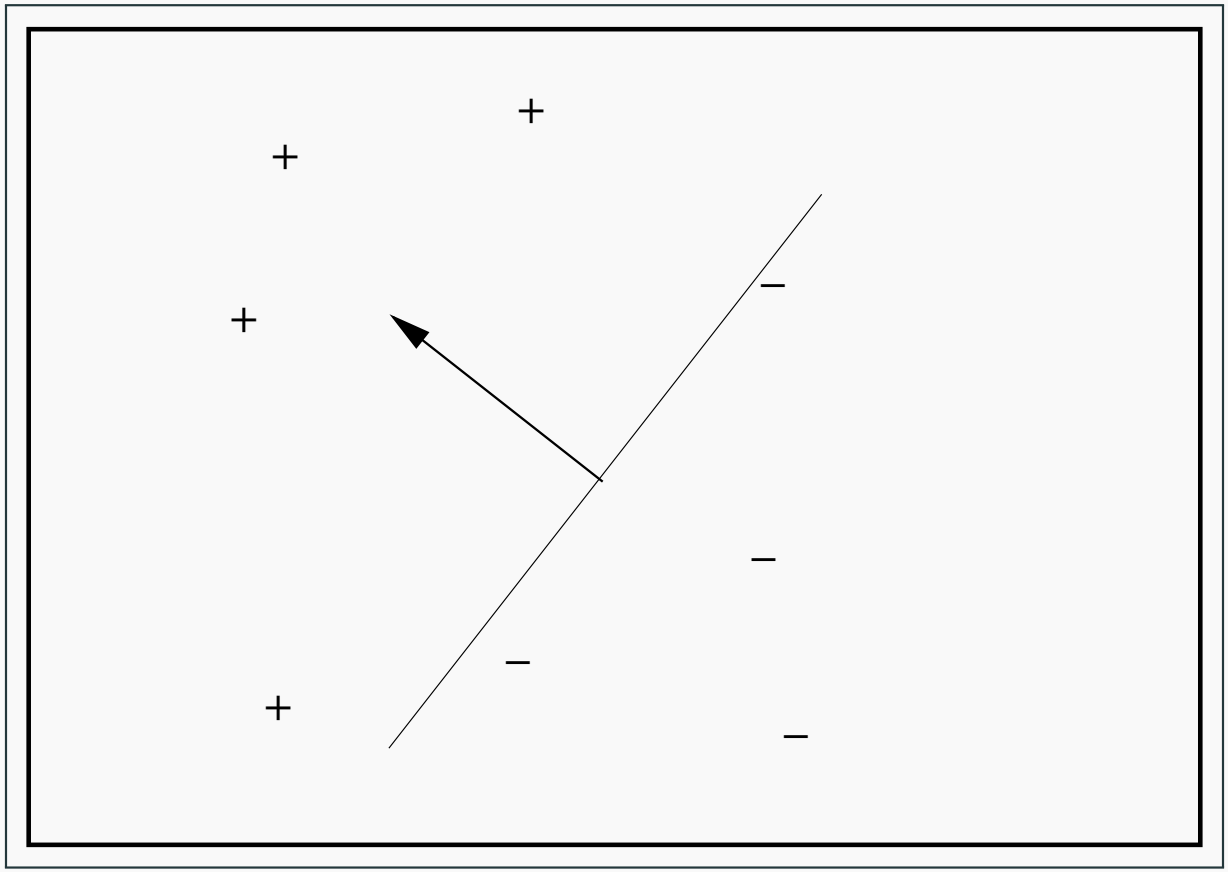
Perceptron animation

- 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 $i = 1 \ldots, m$.
  - Then the perceptron algorithm will stop after at most $t \leq (\frac{2R}{\gamma})^2$ iterations and output a weight vector $\mathbf{w}^{(t)}$ for which $y_i \mathbf{w}^{(t)} \mathbf{x}_i \geq 0$ for all $i = 1 \ldots, m$

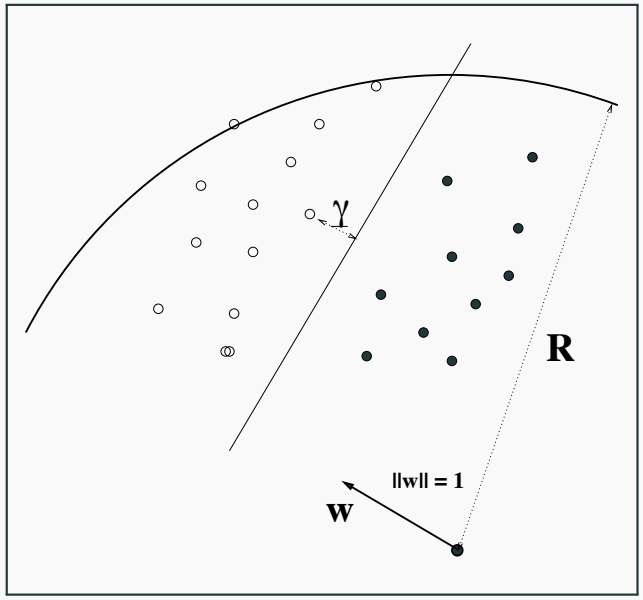
R是训练集中所有样本特征向量范数最大值

γ是一个正数，表示训练集中所有样本到超平面 $W^T X + b = 0$ 最小距离

Convergence of the perceptron algorithm

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

- γ: 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 in is relative to the distances of the training points



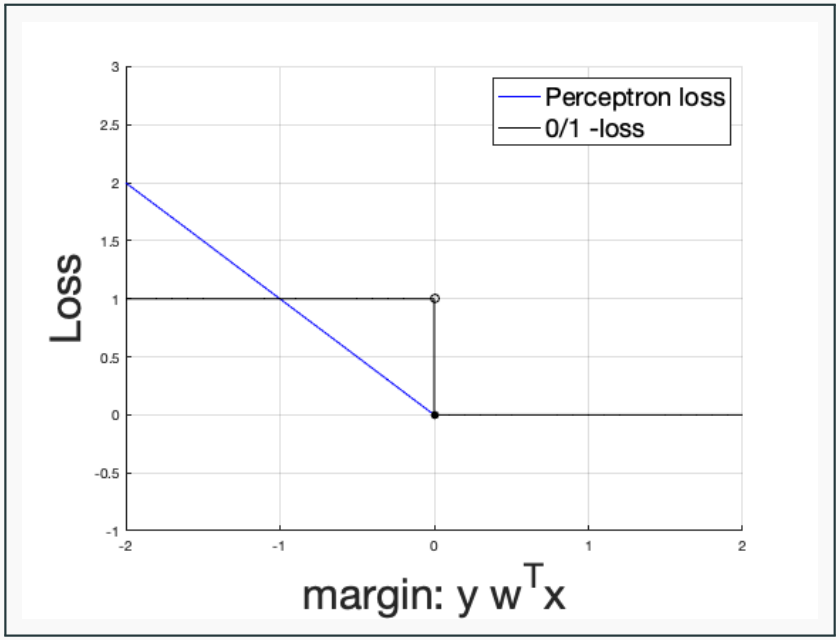
However, Perceptron algorithm does not stop on a non-separable training set, since there will always be a misclassified example that causes an update

The loss function of the Perceptron algorithm

It can be shown that the Perceptron algorithm is using the following loss:

$$L_{Perceptron}(y, \mathbf{w}^T\mathbf{x}) = \max(0, -y\mathbf{w}^T\mathbf{x})$$

- $y\mathbf{w}^T\mathbf{x}$ is the margin
- if $y\mathbf{w}^T\mathbf{x} < 0$, a loss of $-y\mathbf{w}^T\mathbf{x}$ is incurred, otherwise no loss is incurred



感知机的损失函数是一种衡量感知机模型分类错误程度的指标，它定义为所有误分类点到超平面的总距离[1][2]。具体来说，如果一个样本点$(\mathbf{x}_i, y_i)$被误分类了，那么它的函数间隔$y_i(\mathbf{w} \cdot \mathbf{x}_i + b)$就是负数，而它到超平面的距离就是$-\frac{1}{\|\mathbf{w}\|} y_i(\mathbf{w} \cdot \mathbf{x}_i + b)$。如果我们把所有误分类点的距离加起来，就得到了感知机的损失函数：

$$L(\mathbf{w}, b) = -\frac{1}{\|\mathbf{w}\|} \sum_{(\mathbf{x}_i, y_i) \in M} y_i(\mathbf{w} \cdot \mathbf{x}_i + b)$$

其中，M是所有误分类点的集合，$\|\mathbf{w}\|$是权重向量$\mathbf{w}$的范数。为了方便计算，我们可以忽略$\frac{1}{\|\mathbf{w}\|}$这个系数，因为它对于优化问题没有影响[3][4]。所以，我们可以简化感知机的损失函数为：

$$L(\mathbf{w}, b) = -\sum_{(\mathbf{x}_i, y_i) \in M} y_i(\mathbf{w} \cdot \mathbf{x}_i + b)$$

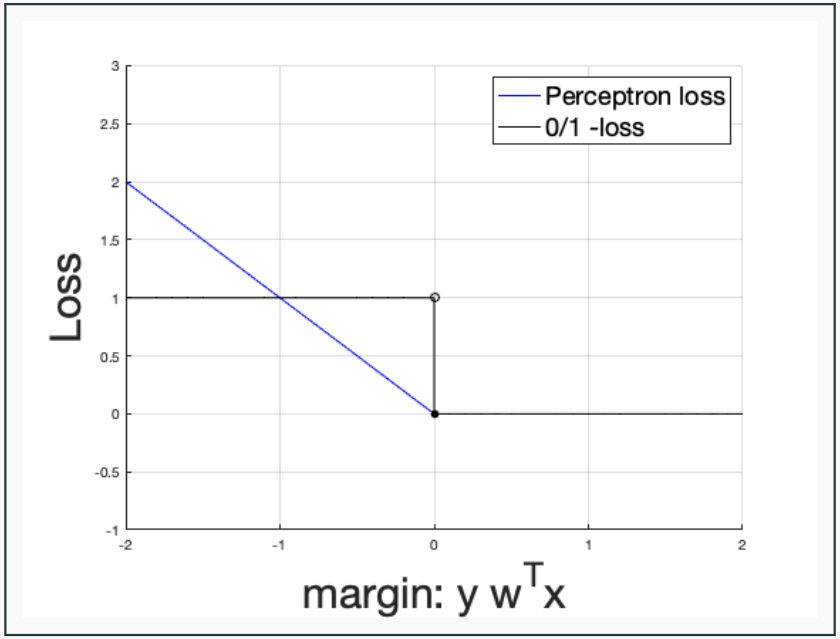
这个损失函数有一个很好的性质，就是它是参数$\mathbf{w}$和b的连续可导函数，这意味着我们可以用梯度下降法或者其他优化算法来求解它的最小值[5]。感知机的目标就是找到一组参数$\mathbf{w}$和b，使得损失函数最小，也就是使得误分类点的数量最少，或者说使得超平面能够尽可能地将不同类别的数据分开。

Convexity of Perceptron loss

A function $f : \mathbb{R}^n \mapsto \mathbb{R}$ is convex if for all x, y, and $0 \leq \theta \leq 1$, we have

$$f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta) f(y).$$

- Geometrical interpretation: the graph of a convex function lies below the line segment from $(x, f(x))$ to $(y, f(y))$
- It is easy to see that Perceptron loss is convex but zero-one loss is not convex



Convexity of Perceptron loss

- The convexity of the Perceptron loss has an important consequence: every local minimum is also the global minimum
- In principle we can minimize it with incremental updates that gradually decrease the loss
- In contrast, finding a hyperplane that minimizes the zero-one loss is computationally hard (NP-hard to minimize training error)
- However, we need better algorithms than the Perceptron, which terminate when we are close to the optimum

Logistic regression

逻辑回归（logistic regression）是一种常用的分类和预测分析的统计模型，它可以估计一个事件发生的概率，如投票与否、健康与否等，基于给定的一组自变量[1]。由于输出是一个概率，因此因变量的取值范围在0和1之间。在逻辑回归中，对于事件的几率（即事件发生的概率与不发生的概率之比）进行对数变换，使得几率成为一个或多个自变量的线性组合。这个对数变换也称为对数几率（log-odds），或者自然对数的几率（logit），因此逻辑回归也称为对数几率回归或者logit回归。这个逻辑函数可以用以下公式表示：

$$\operatorname{Logit}(p) = \frac{1}{1+e^{-p}}$$

$$\ln\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k$$

在这个逻辑回归方程中，$\operatorname{Logit}(p)$是因变量或者响应变量，x是自变量。β参数或者系数在这个模型中通常通过最大似然估计（MLE）来估计。这个方法通过多次迭代来测试不同的β值，以优化对数几率的最佳拟合。找到最优的系数后，就可以计算每个观测值的条件概率，然后取对数并求和，得到一个预测概率。对于二分类问题，如果概率小于0.5，就预测为0；如果概率大于0.5，就预测为1。这是一种常见的制作二元分类器的方法。当然，也可以使用其他的阈值来制作分类器，根据具体问题和目标来决定。

逻辑回归有很多应用领域，包括机器学习、医学、社会科学等[2]。例如，创伤和损伤严重程度评分（Trauma and Injury Severity Score），它广泛用于预测受伤患者的死亡率，最初就是由Boyd等人使用逻辑回归开发的[3]。

Logistic regression

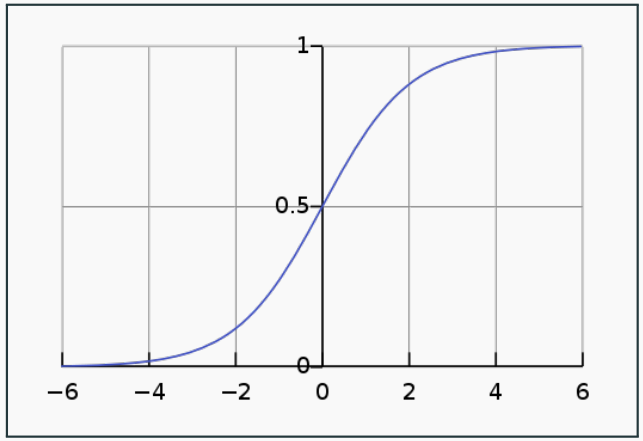
Logistic regression is a classification technique (despite the name)

- it gets its name from the logistic function

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

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

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



Logistic function: a probabilistic interpretation

- The logistic function $\phi_{logistic}(z)$ 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, $1 - p$;

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

- Thus the logistic function

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

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

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

- 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:

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

- 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**:

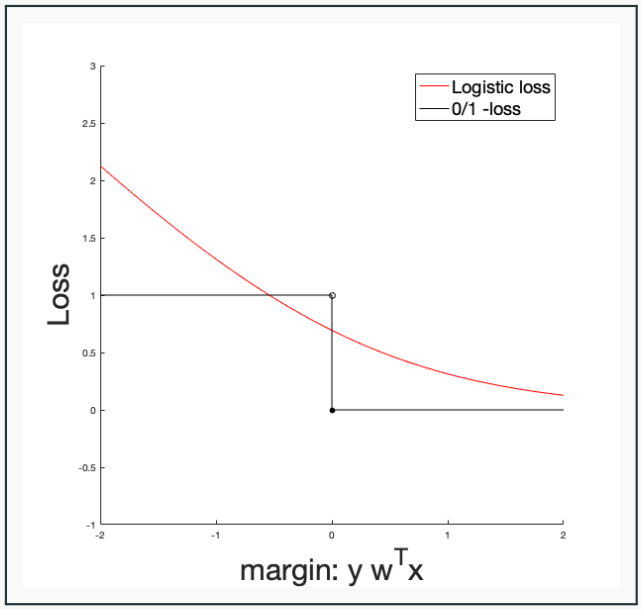
$$L_{logistic}(y, \mathbf{w}^T\mathbf{x}) = \log(1+\exp(-y\mathbf{w}^T\mathbf{x}))$$

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

Geometric interpretation of Logistic loss

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

- 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 $J(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^{m} L_{logistic}(y_i, \mathbf{w}^T\mathbf{x}_i)$ over the training set:

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

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

- 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**

Gradient

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

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

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

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

Stochastic gradient descent

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

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

- 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

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

  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:

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

- 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 η

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

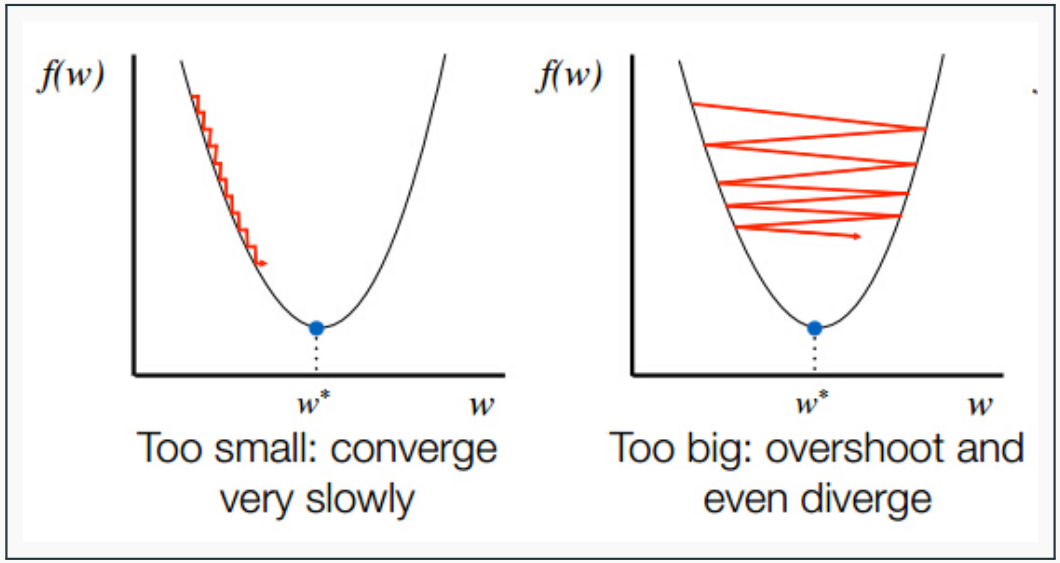
until stopping criterion statisfied

Output $\mathbf{w}$

Stepsize selection

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

- The stepsize parameter η, 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.



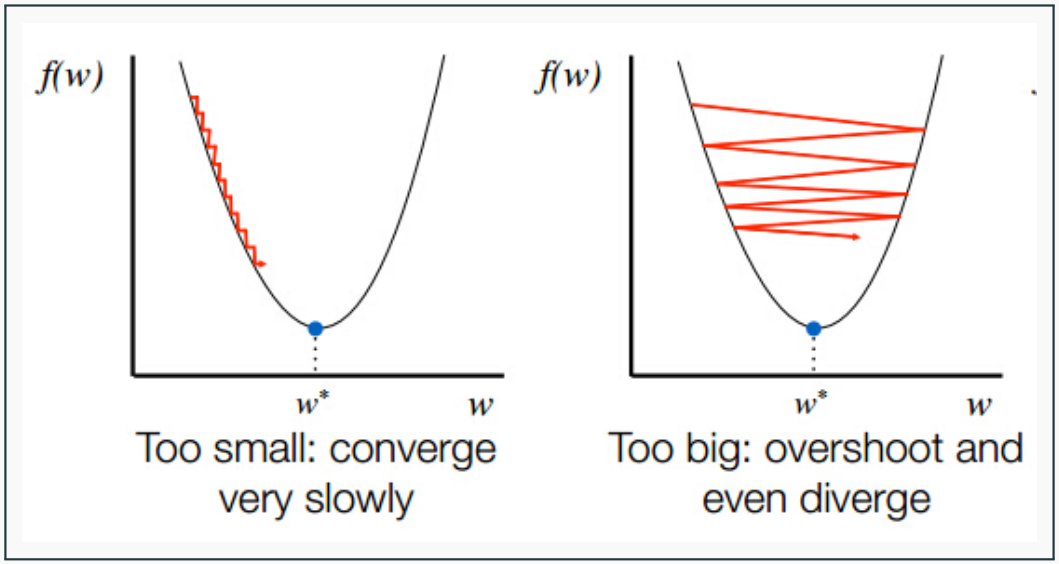
Source: https://dunglai.github.io/2017/12/21/gradient-descent/

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 α requires experimenting with several values



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

Stopping criterion

When should we stop the algorithm? Some possible choices:

1. Set a maximum number of iterations, after which the algorithm terminates
   - This needs to be separately calibrated for each dataset to avoid premature termination 需为每个数据集单独校准避免过早终止
2. Gradient of the objective: If we are at a optimum point $\mathbf{w}^*$ of $J(\mathbf{w})$, the gradient vanishes $\nabla J(\mathbf{w}^*) = 0$, so we can stop $\|J(\mathbf{w})\| < \gamma$ where γ is some user-defined parameter
3. It is usually sufficient to train until the **zero-one error** on training data does not change anymore
   - This usually happens before the logistic loss converges

Summary

- Linear classification model are and important class of machine learning models, they are used as standalone models and appear as building blocks of more complicated, non-liner 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