Linear Classification

Machine Learning

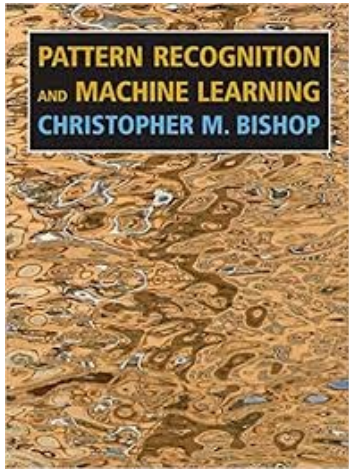
Daniele Loiacono

POLITECNICO
MILANO 1863

References

- This slides are based on material of prof. Marcello Restelli

- *Pattern Recognition and Machine Learning*, Bishop
  - Chapter 4 (4.1.1 - 4.1.3, 4.1.7, 4.3.1, 4.3.2)



Outline

- Linear Classification Models
- Direct Approaches
  - Least Squares for Classification
  - The Perceptron Algorithm
- Probabilistic Discriminative Approach
  - Logistic Regression

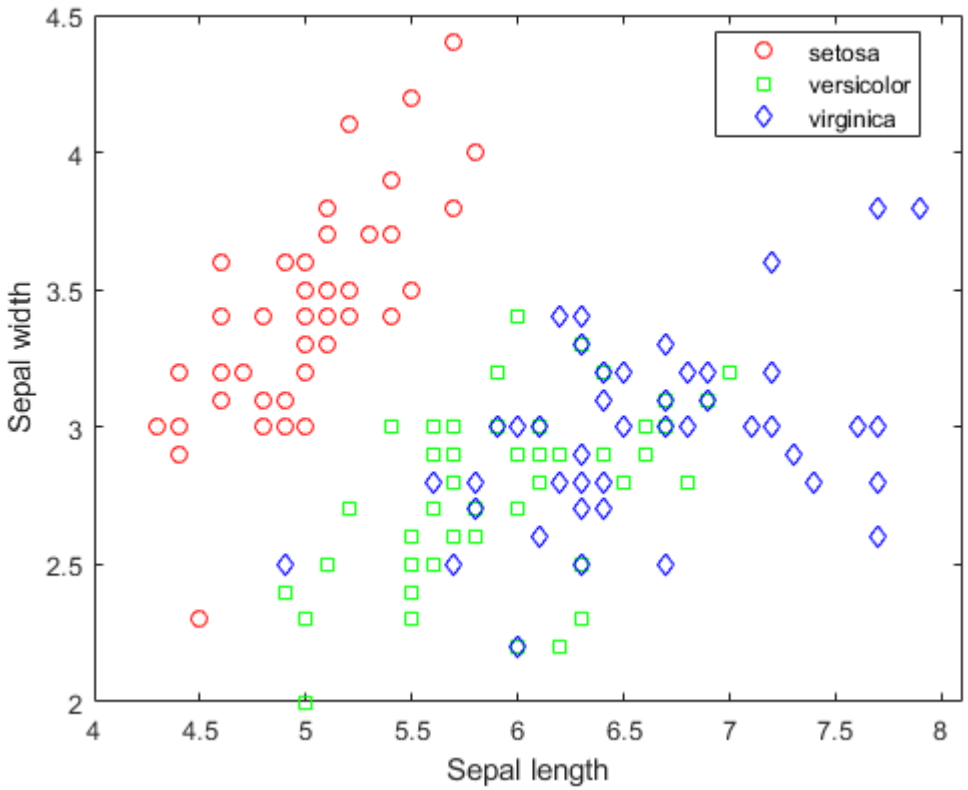
A quick recap

What is classification?

- Learn, from a dataset $\mathcal{D}$, an **approximation** of function $f(x)$ that maps input x to a discrete class C_k (with $\mathrm{k} = 1, \ldots, K$)

$$\mathcal{D} = \{\langle x, C_k \rangle\} \Rightarrow C_k = f(x)$$

  - How do we model f?
  - How do we encode C_k?
  - How do we evaluate our approximation?
  - How do we optimize our approximation?



Classification approaches

- Discriminant function
  - Model a parametric function that maps input to classes
  - Learn parameters from data
- Probablistic discriminative approach
  - Design a parametric model of $p(C_k|\mathbf{x})$
  - Learn model parameters from data
- Probabilistic generative approach
  - Model $p(\mathbf{x}|C_k)$ and class priors $p(C_k)$
  - Fit models to data
  - Infer **posterior** with Bayes' rule: $p(C_k|\mathbf{x}) = p(\mathbf{x}|C_k)p(C_k)/p(\mathbf{x})$

Classification approaches

- Discriminant function
  - Model a parametric function that maps input to classes
  - Learn parameters from data
- Probablistic discriminative approach
  - Design a parametric model of $p(C_k|\mathbf{x})$
  - Learn model parameters from data
- Probabilistic generative approach
  - Model $p(\mathbf{x}|C_k)$ and class priors $p(C_k)$
  - Fit models to data
  - Infer **posterior** with Bayes' rule: $p(C_k|\mathbf{x}) = p(\mathbf{x}|C_k)p(C_k)/p(\mathbf{x})$

Discriminant Function

Generalized Linear Models for classification

- In linear classification, we will use **generalized linear models**:

$$f(\mathbf{x}, \mathbf{w}) = f\left(w_0 + \sum_{j=1}^{D-1} w_j x_j\right) = f(\mathbf{x}^T \mathbf{w} + w_0)$$

  - **f**$(\cdot)$ is **not** linear in $\mathbf{w}$ due to the (non linear) **activation function** f, because its output is either a discrete label or a probability value
  - **f**$(\cdot)$ partitions the input space into **decision regions** whose boundaries are called **decision boundaries** or **decision surfaces**
  - these decision surfaces are linear function of $\mathbf{x}$ and $\mathbf{w}$, as they correspond to $\mathbf{x}^T \mathbf{w} + w_0 = \text{const}$
  - generalized linear models are more complex to use with respect to linear models (both from a computational and analytical perspective)

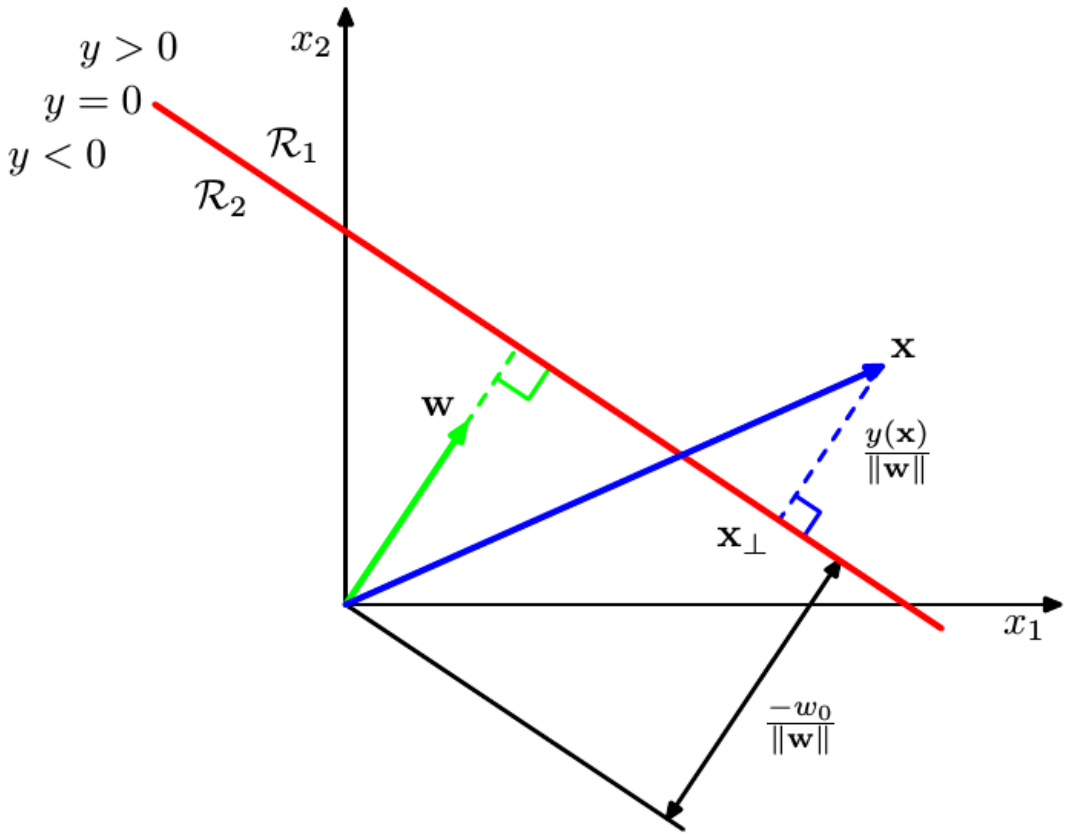
Label Encoding

- A common encoding for two-class problems is a binary encoding: $t \in \{0,1\}$
  - $t = 1$ encodes **positive** class and $t = 0$ encodes **negative** one
  - with this encoding, t and $f(\cdot)$ represent the **probability** of positive class
- A possible alternative encoding for two-class problems is $t \in \{-1,1\}$
  - this encoding is convenient for some algorithms
- When the problem has K classes, a typical choice is **1-of-K** encoding
  - t is a vector of length K, with a 1 in the position corresponding to the encoded class
  - with this encoding, t and $f(\cdot)$ represent the probability density over the classes
  - as an example, a data sample that belongs to class 4 of a problem with K=5, is encoded as $t = (0,0,0,1,0)^T$

Discriminant linear function for a two-class problem

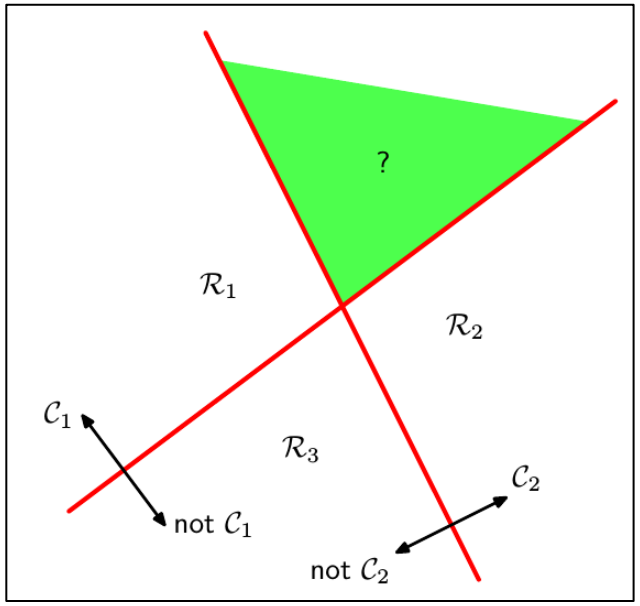
$$f(\mathbf{x}, \mathbf{w}) = \begin{cases} C_1, & \text{if } \mathbf{x}^T\mathbf{w} + w_0 \geq 0 \\ C_2, & \text{otherwise} \end{cases}$$

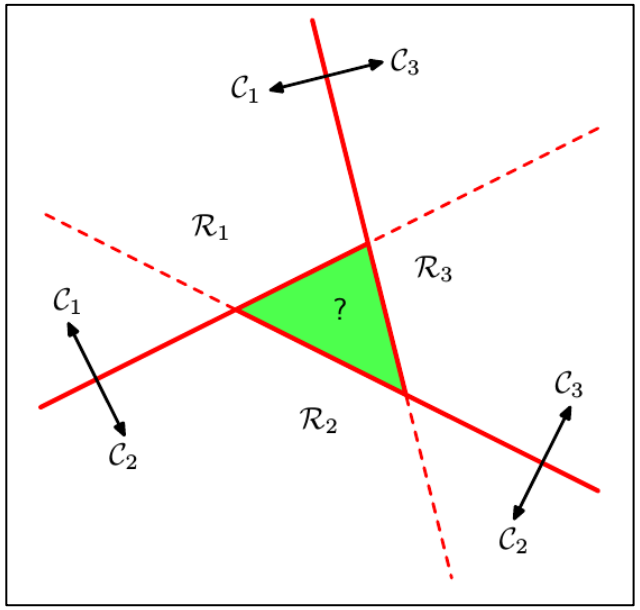
- Properties
  - DS is $y(\cdot) = \mathbf{x}^T\mathbf{w} + w_0 = 0$
  - DS is orthogonal to $\mathbf{w}$
  - distance of DS from origin is $-\frac{w_0}{\|\mathbf{w}\|_2}$
  - distance of $\mathbf{x}$ from DS is $\frac{y(\mathbf{x})}{\|\mathbf{w}\|_2}$



How to deal with multiple classes problems?

- In a multi-class problem we have K classes
- **One-versus-the-rest** approach uses K-1 **binary cassifiers** (i.e., that solve a two-class problem)
  - each classifier discriminates C_i and $\text{not } C_i$ regions
  - **ambiguity:** region mapped to several classes
- **One-versus-one** approach uses K(K-1)/2 class binary classifiers
  - each classifier discriminates between C_i and C_j
  - similary ambiguity of previous approach





A simple solution for multiple classes

- A possible solutions is to use K linear discriminant functions:

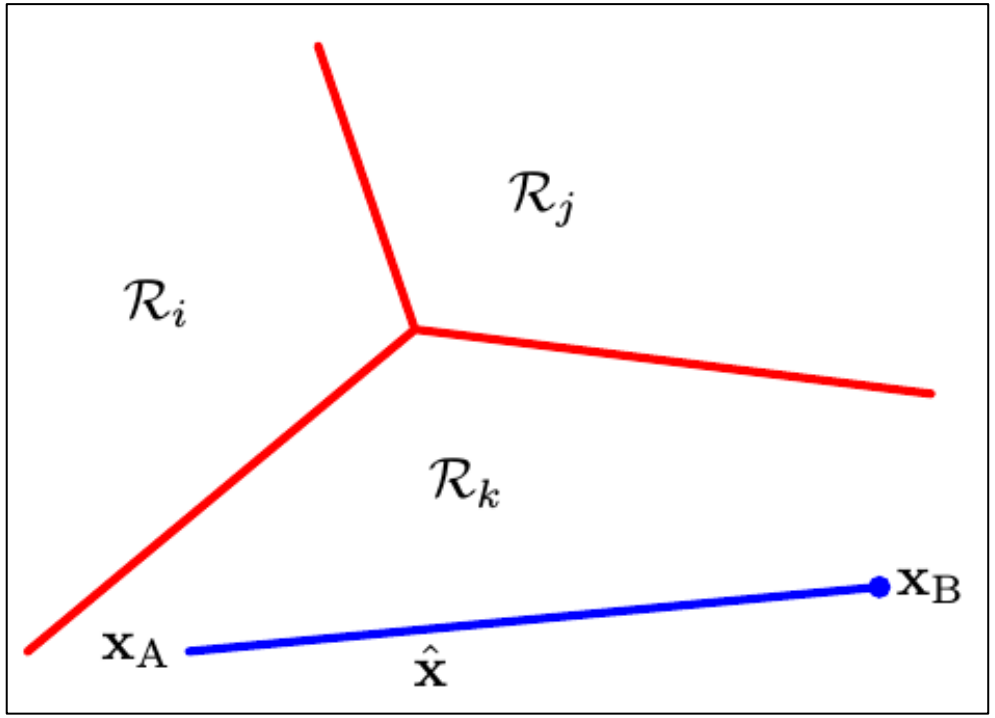
$$y_k(\mathbf{x}) = \mathbf{x}^T \mathbf{w}_k + w_{k0}, \text{ where } k = 1, \ldots, K$$

  - Map $\mathbf{x}$ to class C_k if $y_k > y_j \quad \forall j \neq k$
  - No ambiguity
  - DS are **singly connected** and **convex**

Let $\mathbf{x}_A, \mathbf{x}_b \in \mathcal{R}_k$
Thus, $y_k(\mathbf{x}_A) > y_j(\mathbf{x}_A)$ and $y_k(\mathbf{x}_B) > y_j(\mathbf{x}_B)$

$\Rightarrow \forall \alpha \ (0 < \alpha < 1)$

$y_k(\alpha \mathbf{x}_A + (1-\alpha)\mathbf{x}_B) > y_j(\alpha \mathbf{x}_A + (1-\alpha)\mathbf{x}_B)$

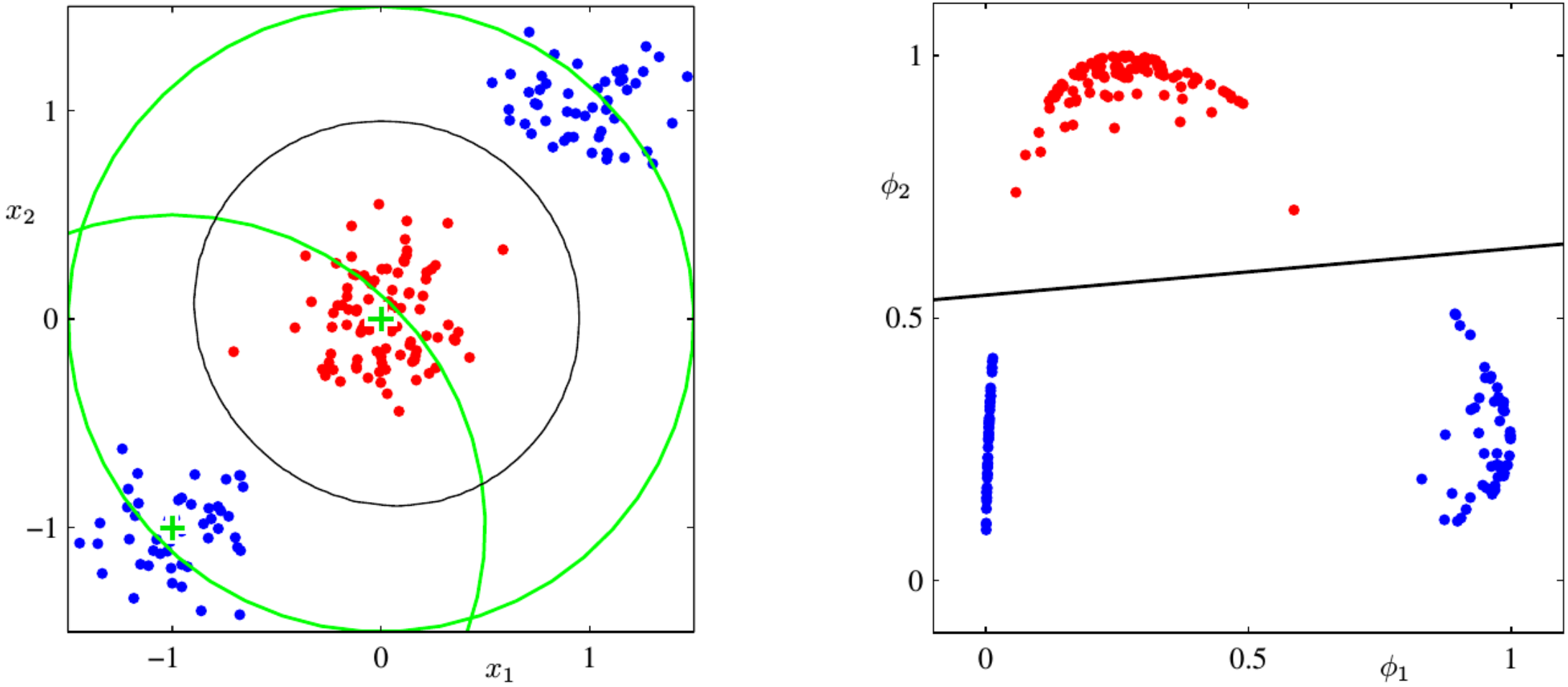


Linear Basis Function Models

- So far we considered models that work in the **input space** (i.e., with **x**)
- However, we can still extend models by using a fixed set of basis function $\phi(\mathbf{x})$
- Basically, we apply a non-linear transformation to map the **input space** into a **feature space**
- As a result, decision boundaries that are **linear** in the **feature space** would correspond to **nonlinear** boundaries in the **input space**
- This allows to apply linear classification models also to problem where samples are not **linearly separable**

Linear Basis Function Models: an example

- Assuming two Gaussian basis functions (in green)



Least Squares for Classification

- Let consider a K-class problem and use a 1-of-K encoding for target
- Each class is modeled with a linear function:

$$y_k(\mathbf{x}) = \mathbf{x}^T \mathbf{w}_k + w_{k0}, \text{ where } k = 1, \ldots, K$$

- In matrix notation:

$$\mathbf{y}(\mathbf{x}) = \tilde{\mathbf{W}}^T \tilde{\mathbf{x}}$$

  - $\tilde{\mathbf{W}}$ has size (D+1)xK
  - k-th column of $\tilde{\mathbf{W}}$ is $\widetilde{\mathbf{w}_k} = \left(w_{k0}, \mathbf{w}_k^T\right)^T$
  - $\tilde{\mathbf{x}} = (1, \mathbf{x}^T)^T$

Least Squares for Classification (2)

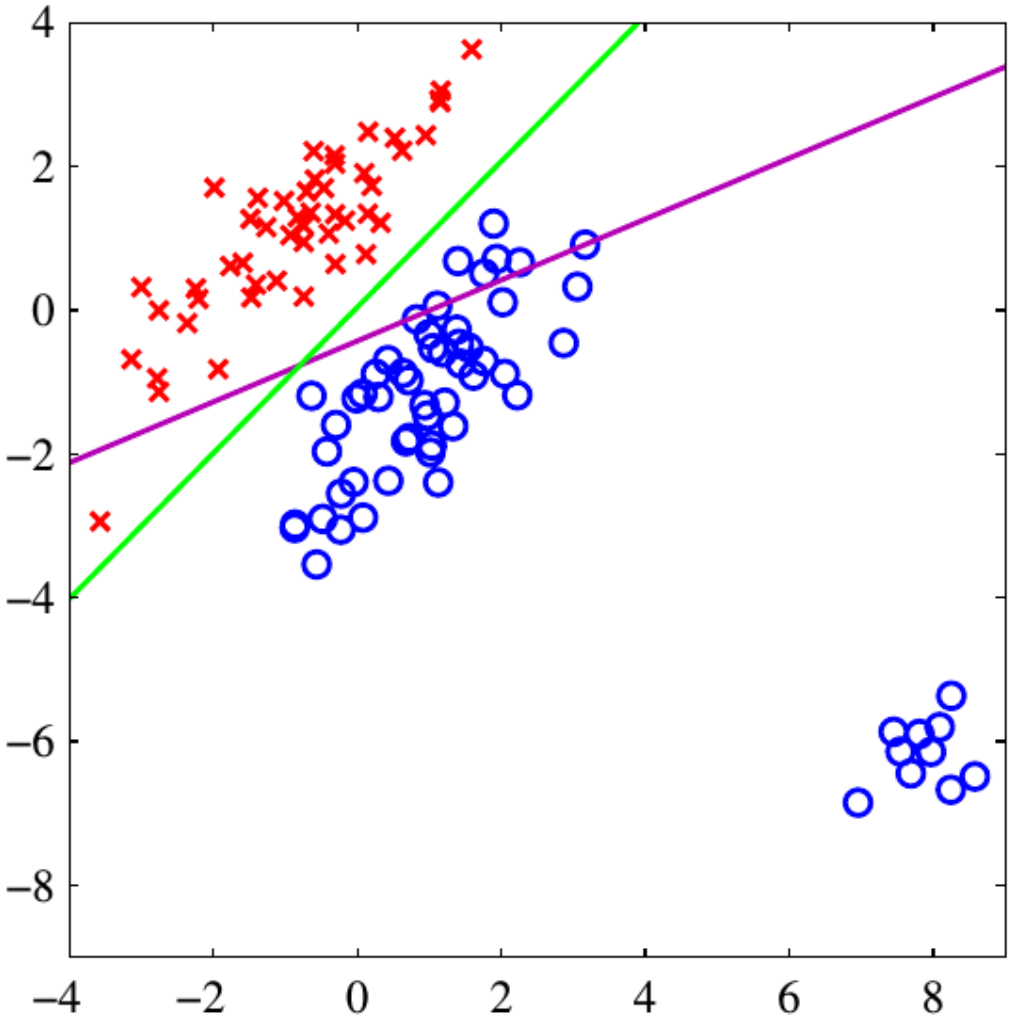
- Given a dataset $\mathcal{D} = \{\mathbf{x}_i, \mathbf{t}_i\}$, where i=1,...,N
- We can apply Least Squares to find the optimal value of $\widetilde{\mathbf{W}}$

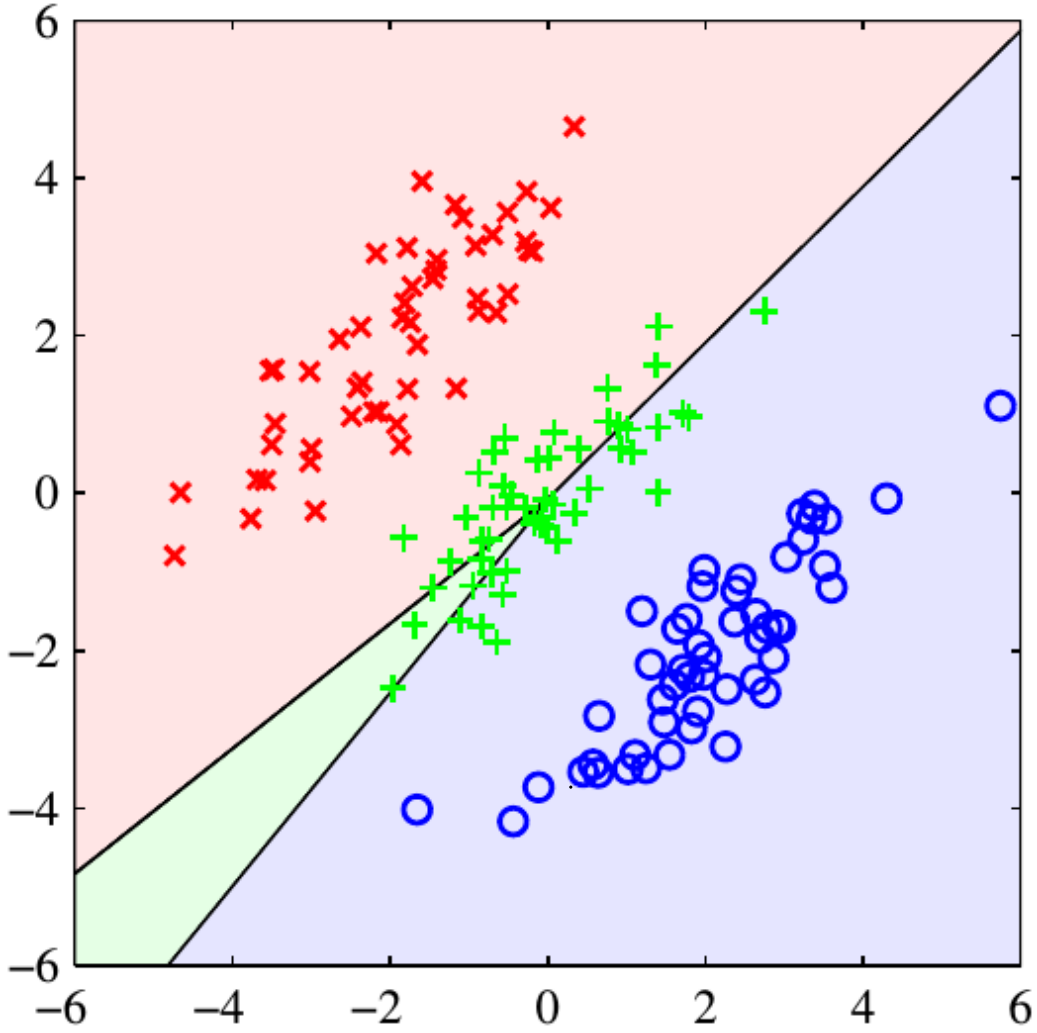
$$\tilde{\mathbf{W}} = \left(\tilde{\mathbf{X}}^T \tilde{\mathbf{X}}\right)^{-1} \tilde{\mathbf{X}}^T \mathbf{T}$$

  - $\tilde{\mathbf{X}}$ is a Nx(D+1) matrix whose i-th is $\tilde{\mathbf{x}}_i^T$
  - $\mathbf{T}$ is a NxK matrix whose i-th row is $\mathbf{t}_i^T$

- Any new sample $\tilde{\mathbf{x}}_{new}^T$ is mapped to class C_k if $t_k > t_j \quad \forall j$, where t_k is the k-th component of the model output, computed as $t_k = \tilde{\mathbf{x}}^T \widetilde{\mathbf{w}_k}$

Problems with Least Squares: Outliers and Output Distribution





Perceptron

- The **perceptron** is a linear discriminant model proposed by Rosenblatt in 1958 along with a **sequential learning** algorithm
- Perceptron is devised for a two-class problem, where classes encoding is {-1,1}

$$f(\mathbf{x}, \mathbf{w}) = \begin{cases} +1, & \text{if } \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}) \geq 0 \\ -1, & \text{otherwise} \end{cases}$$

- The **perceptron algorithm** aims at finding a **decision surface** (also called **separating hyperplane**) by minimizing the distance of **misclassified samples** to the boundary
- Minimization of this loss function can be performed using **stochastic gradient descent**
- Despite a simpler loss function could be used in principle (e.g., number of misclassified samples), this are more complex to minimize.

Perceptron Criterion

- We aim at finding $\mathbf{w}$ such that $\mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_i) \geq 0$ for $\mathbf{x}_i \in C_1$ and $\mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_i)<0$ otherwise
- The **Perceptron Criterion** is defined as:

$$L_P(\mathbf{w}) = -\sum_{n\in\mathcal{M}} \mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_n)t_n$$

  - samples clssified correctly do not contribute to L
  - each misclassified sample $\mathbf{x}_i \in \mathcal{M}$ contributes as $\mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_i)t_i$
- L_P can be minimized using **stochastic gradient descent**:

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \alpha\nabla L_P(\mathbf{w}) = \mathbf{w}^{(k)} + \alpha\boldsymbol{\phi}(\mathbf{x}_n)t_n$$

  - Since the scale of $\mathbf{w}$ does not change the perceptron function, usually the **learning rate** α is set to 1

Perceptron Algorithm

Given $\mathcal{D} = \{\mathbf{x}_i, \mathbf{t}_i\}$, where i=1,…,N

Initialize $\mathbf{w}_0$

k ← 0

repeat

  k ← k+1

  n ← k mod N

  if $\widehat{t_n} \neq t_n$ **then**

    $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + \boldsymbol{\phi}(\boldsymbol{x_n}) t_n$

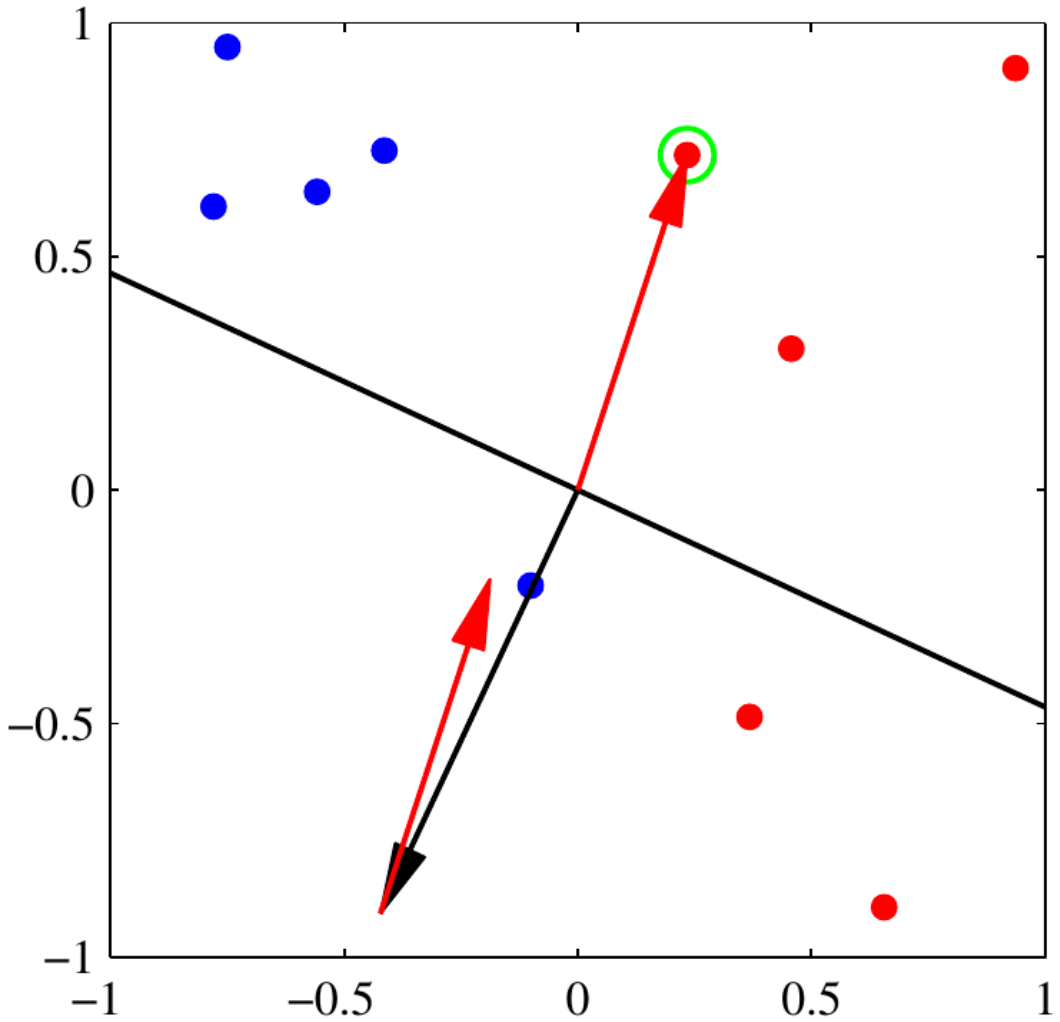
  endif

until convergence

Perceptron Algorithm

Given $\mathcal{D} = \{\mathbf{x}_i, \mathbf{t}_i\}$, where i=1,...,N

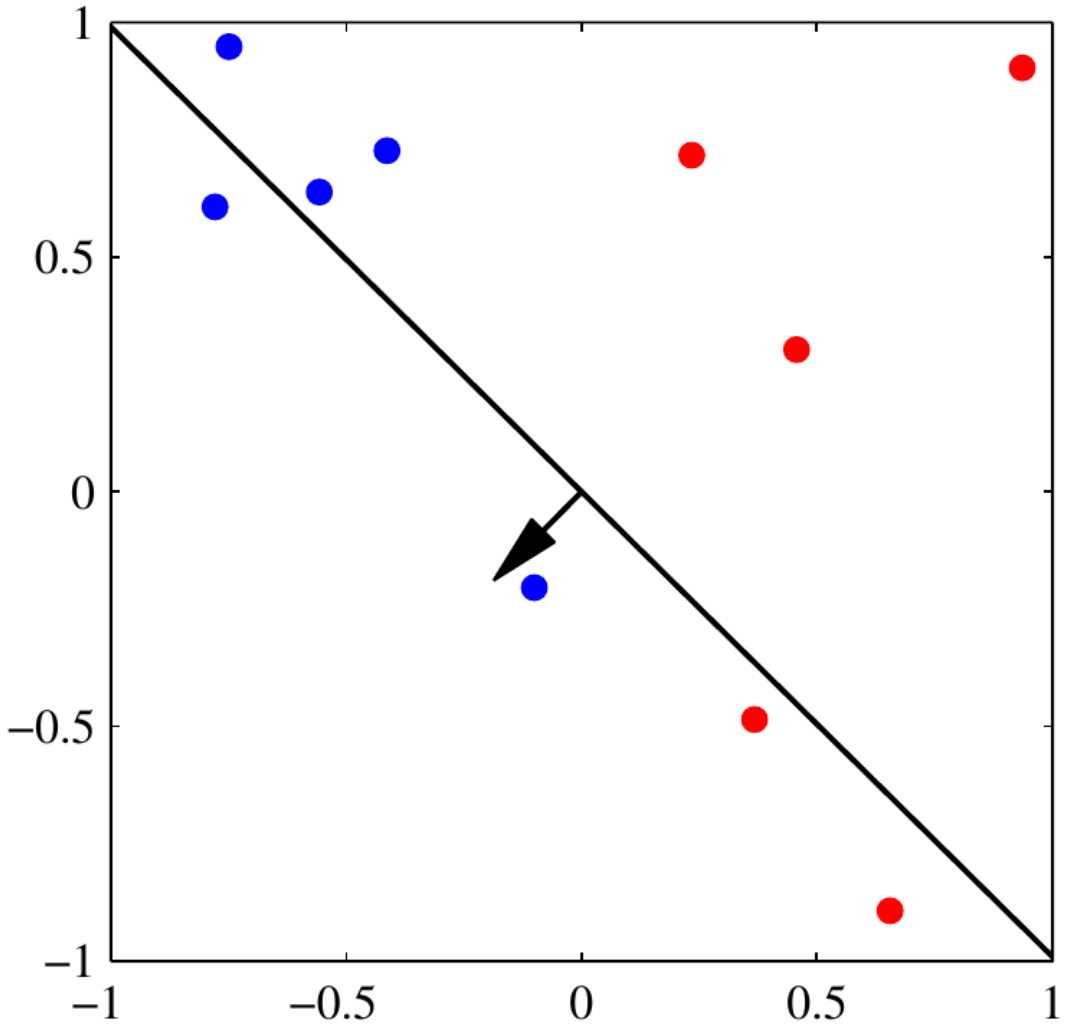
Initialize $\mathbf{w}_0$
k ← 0
repeat
  k ← k+1
  n ← k mod N
  if $\widehat{t_n} \neq t_n$ **then**
    $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + \boldsymbol{\phi}(\boldsymbol{x_n})t_n$
  endif
until convergence



Perceptron Algorithm

Given $\mathcal{D} = \{\mathbf{x}_i, \mathbf{t}_i\}$, where i=1,...,N

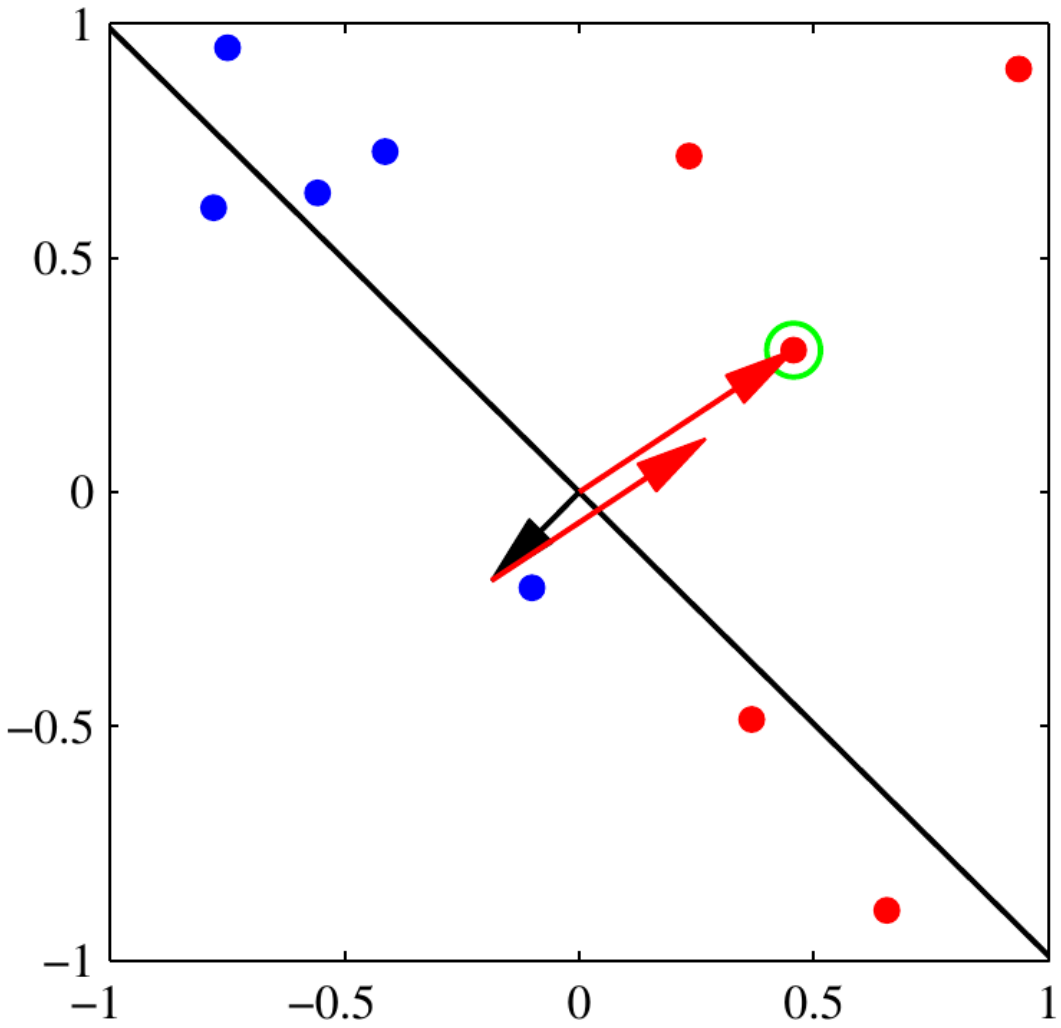
Initialize $\mathbf{w}_0$
k ← 0
repeat
  k ← k+1
  n ← k mod N
  if $\widehat{t_n} \neq t_n$ **then**
    $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + \boldsymbol{\phi}(\boldsymbol{x_n})t_n$
  endif
until convergence



Perceptron Algorithm

Given $\mathcal{D} = \{\mathbf{x}_i, \mathbf{t}_i\}$, where i=1,...,N

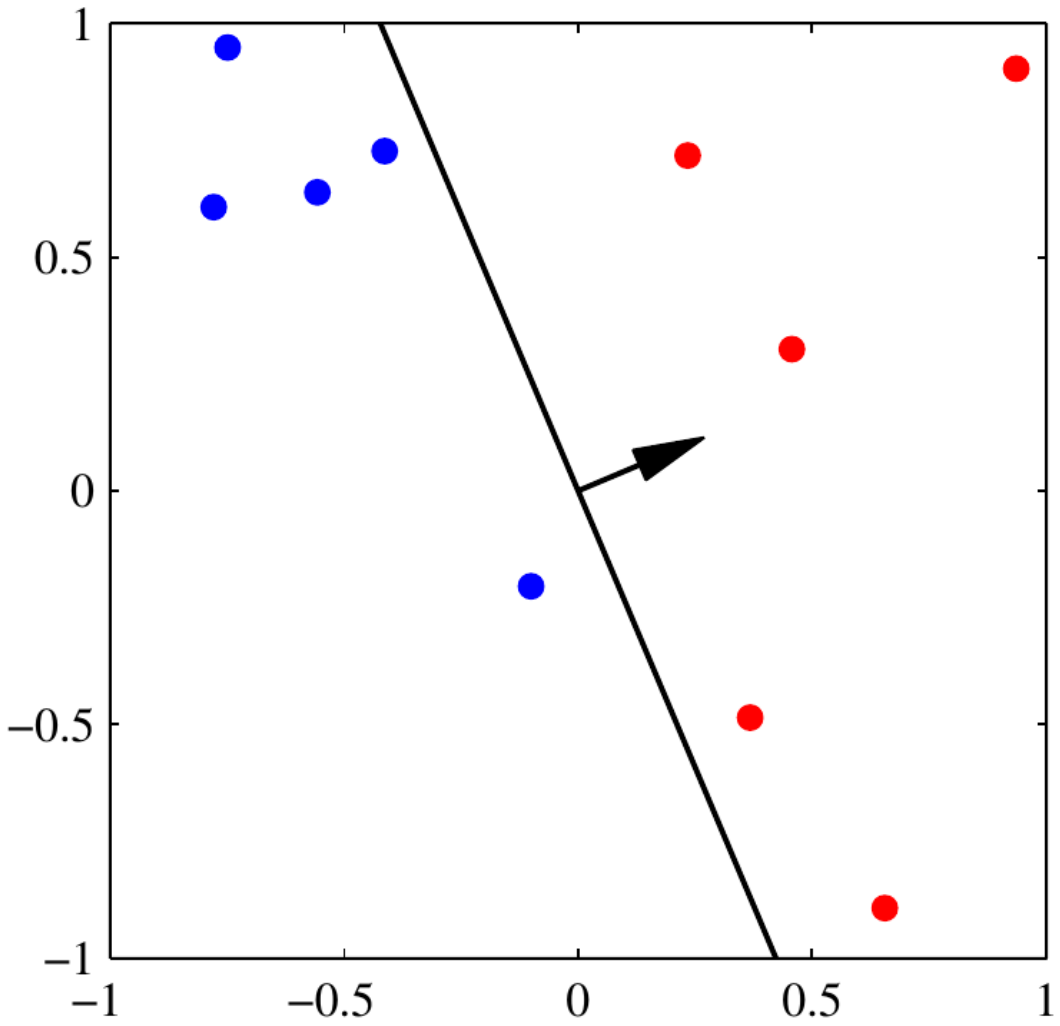
Initialize $\mathbf{w}_0$
k ← 0
repeat
  k ← k+1
  n ← k mod N
  if $\widehat{t_n} \neq t_n$ **then**
    $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + \boldsymbol{\phi}(\boldsymbol{x_n})t_n$
  endif
until convergence



Perceptron Algorithm

Given $\mathcal{D} = \{\mathbf{x}_i, \mathbf{t}_i\}$, where i=1,...,N

Initialize $\mathbf{w}_0$
k ← 0
repeat
  k ← k+1
  n ← k mod N
  if $\widehat{t_n} \neq t_n$ **then**
    $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + \boldsymbol{\phi}(\boldsymbol{x_n})t_n$
  endif
until convergence



Perceptron Convergence Theorem

- A single update **reduce the error** on the **single** misclassifed sample:

$$-\mathbf{w}^{(k+1)^T}\boldsymbol{\phi}(\mathbf{x}_n)t_n = -\mathbf{w}^{(k)^T}\boldsymbol{\phi}(\mathbf{x}_n)t_n - (\boldsymbol{\phi}(\mathbf{x}_n)t_n)^T\boldsymbol{\phi}(\mathbf{x}_n)t_n < -\mathbf{w}^{(k)^T}\boldsymbol{\phi}(\mathbf{x}_n)t_n$$

  - This does **not** imply that the entire loss is reduced after each update
- Perceptron Convergence Theorem

*If the training data set is **linearly separable** in the feature space $\boldsymbol{\Phi}$, then the perceptron learning algorithm is guaranteed to find an **exact solution** in a **finite number of steps***

  - How many steps? Several steps might be necessary, thus it might be difficult to distinguish between n**onseparable** problems and **slowly converging** ones
  - Which solution? If **multiple solutions** exist, the one found by the algorithms depends from i**nitialization** of parameters and the **order** of updates

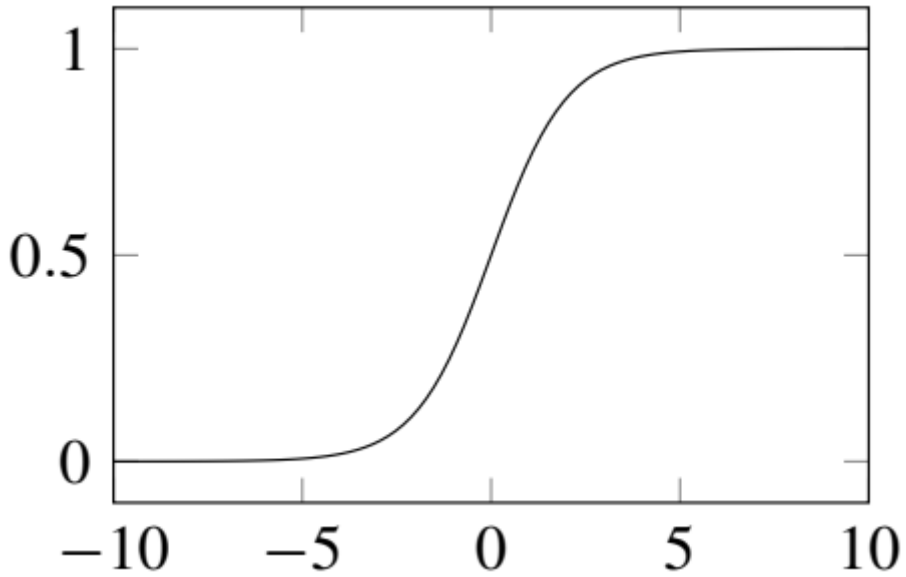
Probabilistic Discriminative Approaches

Two-Class Logistic Regression

- In a discriminative approach we model directly the conditioned class probability:

$$p(C_1|\boldsymbol{\phi}) = \frac{1}{1+\exp(-\mathbf{w}^T\boldsymbol{\phi})} = \sigma(\mathbf{w}^T\boldsymbol{\phi})$$

  - where $\sigma(a) = 1/(1+\exp(-a)$ is the sigmoidal function
  - $\mathrm{p}(C_2|\boldsymbol{\phi}) = 1 - p(C_1|\boldsymbol{\phi})$
  - this model is knonw as **logistic regression**



Maximum Likelihood for Logistic Regression

- Given dataset $\mathcal{D} = \{\mathbf{x}_i, t_i\}$, where i=1,...,N and $t_i \in \{0,1\}$ we want to maximize the likelihood, i.e., the probability to observe the targets given the inputs: $p(\mathbf{t}|\mathbf{X},\mathbf{w})$
- We model the likelihood of the single sample using a Bernoulli distribution, using the logistic regression model for conditioned class probability:

$$p(t_n|\mathbf{x}_n, \mathbf{w}) = y_n^{t_n}(1 - y_n)^{1-t_n} \quad \text{where} \quad y_n = p(t_n = 1|\mathbf{x}_n, \mathbf{w}) = \sigma(\mathbf{w}^T \boldsymbol{\phi}_n)$$

- Assuming that data in $\mathcal{D}$ have been independently sampled we get:

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}) = \prod_{n=1}^{N} y_n^{t_n}(1 - y_n)^{1-t_n}, \quad y_n = \sigma(\mathbf{w}^T \boldsymbol{\phi}_n)$$

Maximum Likelihood for Logistic Regression (2)

- A convenient loss function to **minimize** is the negative log-likelihood (also **known as cross-entropy error function**)

$$L(\mathbf{w}) = -\ln p(\mathbf{t}|\mathbf{X}, \mathbf{w}) = -\sum_{n=1}^{N} (t_n \ln y_n + (1 - t_n) \ln(1 - y_n)) = \sum_{n=1}^{N} L_n$$

- Now we have to compute the derivative of $L(\mathbf{w})$:

$$\frac{\partial L_n}{\partial y_n} = \frac{y_n - t_n}{y_n(1 - y_n)}, \quad \frac{\partial y_n}{\partial \mathbf{w}} = y_n(1 - y_n)\boldsymbol{\phi}_n, \Longrightarrow \frac{\partial L_n}{\partial \mathbf{w}} = \frac{\partial L_n}{\partial y_n}\frac{\partial y_n}{\partial \mathbf{w}} = (y_n - t_n)\boldsymbol{\phi}_n$$

Maximum Likelihood for Logistic Regression (3)

- Thus the gradient of the loss function is:

$$\nabla L(\mathbf{w}) = \sum_{n=1}^{N} (y_n - t_n)\boldsymbol{\phi}_n$$

  - due to the nonlinear logistic regression function it is not possible to find a closed-form solution
  - however, the error function is **convex** and **gradient-based optimization** can be applied (also in an **online learning setting**)

Multiclass Logistic Regression

- In multiclass problems, $p(C_k|\boldsymbol{\phi})$ is modeled by a **softmax** tranformation of the output of K linear functions (one for each class):

$$p(C_k|\boldsymbol{\phi}) = y_k(\boldsymbol{\phi}) = \frac{\exp(\mathbf{w}_k{}^T\boldsymbol{\phi})}{\sum_j \exp(\mathbf{w}_j{}^T\boldsymbol{\phi})}$$

- As for the two-class logistic regression and assuming 1-of-K encoding for the target, we can compute **likelihood** as:

$$p(\mathbf{T}|\boldsymbol{\Phi}, \mathbf{w}_1, \ldots, \mathbf{w}_K) = \prod_{n=1}^{N} \underbrace{\left(\prod_{k=1}^{K} p(C_k|\boldsymbol{\phi}_n)^{t_{nk}} \right)}_{\text{Only one term corresponding to correct class}} = \prod_{n=1}^{N} \left(\prod_{k=1}^{K} y_{nk}^{t_{nk}} \right)$$

Multiclass Logistic Regression

- In multiclass problems, $p(C_k|\boldsymbol{\phi})$ is modeled by a **softmax** tranformation of the output of K linear functions (one for each class):

$$p(C_k|\boldsymbol{\phi}) = y_k(\boldsymbol{\phi}) = \frac{\exp(\mathbf{w}_k{}^T\boldsymbol{\phi})}{\sum_j \exp(\mathbf{w}_j{}^T\boldsymbol{\phi})}$$

- As $y_{nk} = p(C_k|\boldsymbol{\phi}_n) = \frac{\exp(\mathbf{w}_k{}^T\boldsymbol{\phi}_n)}{\sum_j \exp(\mathbf{w}_j{}^T\boldsymbol{\phi}_n)}$ and assuming 1-of-K encoding for the target, we can compute **likelihood** as:

$$p(\mathbf{T}|\boldsymbol{\Phi}, \mathbf{w}_1, \ldots, \mathbf{w}_K) = \prod_{n=1}^{N} \underbrace{\left(\prod_{k=1}^{K} p(C_k|\boldsymbol{\phi}_n)^{t_{nk}} \right)}_{\text{Only one term corresponding to correct class}} = \prod_{n=1}^{N} \left(\prod_{k=1}^{K} y_{nk}^{t_{nk}} \right)$$

Multiclass Logistic Regression (2)

- As for the two-class problem, we can minimize the **cross-entropy error function**:

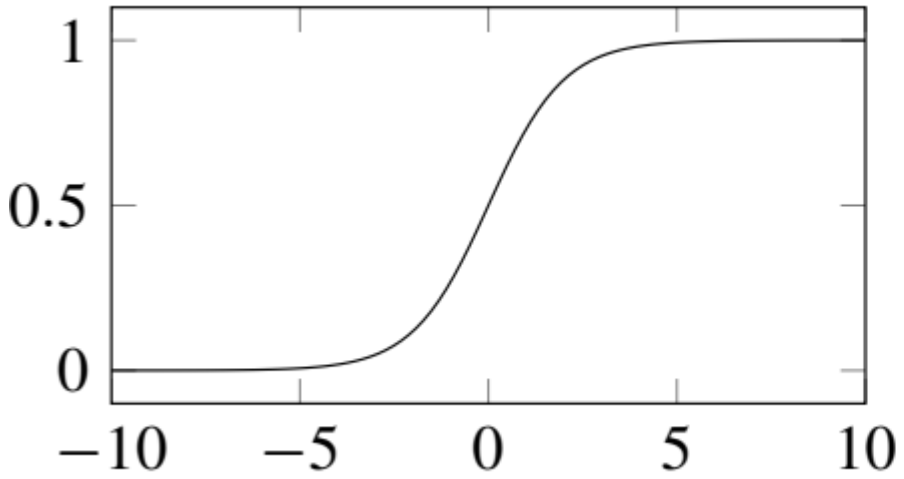
$$L(\mathbf{w}_1, \ldots, \mathbf{w}_K) = -\ln p(\mathbf{T}|\mathbf{\Phi}, \mathbf{w}_1, \ldots, \mathbf{w}_K) = -\sum_{n=1}^{N}\left(\sum_{k=1}^{K} t_{nk} \ln y_{nk}\right)$$

- Then, we compute the gradient for each weights vector:

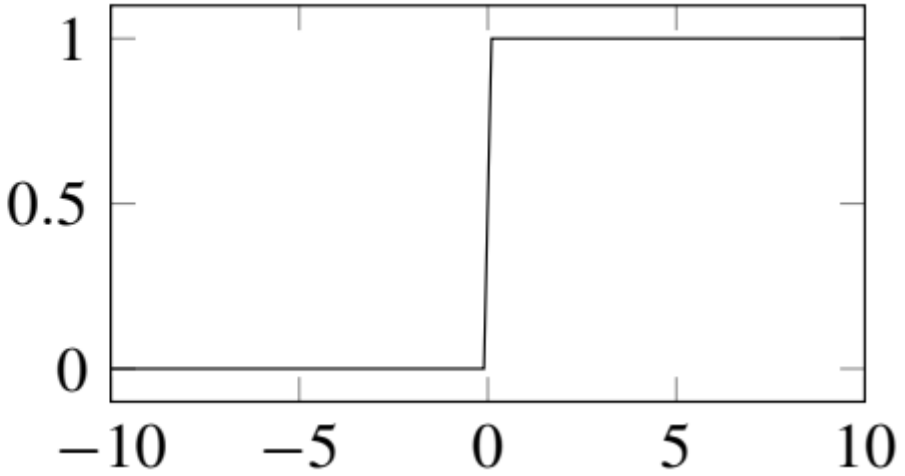
$$\nabla L_{\mathbf{w}_j}(\mathbf{w}_1, \ldots, \mathbf{w}_K) = \sum_{n=1}^{N}(y_{nj} - t_{nj})\boldsymbol{\phi}_n$$

Logistic Regression and Perceptron Algorithm

- If we replace the logistic function with a step function...



$$y(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-\mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x})}}$$



$$y(\mathbf{x}, \mathbf{w}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}) > 0 \\ 0 & \text{otherwise} \end{cases}$$

- ... logisitc regression leads to the same **updating rule** of perceptron algorithm:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \left(y(\mathbf{x}_n, \mathbf{w}) - t_n \right) \boldsymbol{\phi}_n$$