

## Discriminative learning

### Discriminative vs generative

- Generative learning assumes knowledge of the distribution governing the data
- Discriminative learning focuses on directly modeling the discriminant function
- E.g. for classification, directly modeling decision boundaries (rather than inferring them from the modelled data distributions)

## Discriminative learning

### PROS

- When data are complex, modeling their distribution can be very difficult
- If data discrimination is the goal, data distribution modeling is not needed
- Focuses parameters (and thus use of available training examples) on the desired goal

### CONS

- The learned model is less flexible in its usage
- It does not allow to perform arbitrary inference tasks
- E.g. it is not possible to efficiently generate new data from a certain class

## Linear discriminant functions

### Description

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0$$

- The discriminant function is a linear combination of example features
- $w_0$  is called *bias* or *threshold*
- it is the simplest possible discriminant function
- Depending on the complexity of the task and amount of data, it can be the best option available (at least it is the first to try)

## Linear binary classifier

### Description

$$f(\mathbf{x}) = \text{sign}(\mathbf{w}^T \mathbf{x} + w_0)$$

- It is obtained taking the sign of the linear function
- The decision boundary ( $f(\mathbf{x}) = 0$ ) is a hyperplane ( $H$ )
- The weight vector  $\mathbf{w}$  is orthogonal to the decision hyperplane:

$$\begin{aligned} \forall \mathbf{x}, \mathbf{x}' : f(\mathbf{x}) = f(\mathbf{x}') = 0 \\ \mathbf{w}^T \mathbf{x} + w_0 - \mathbf{w}^T \mathbf{x}' - w_0 = 0 \\ \mathbf{w}^T (\mathbf{x} - \mathbf{x}') = 0 \end{aligned}$$

## Linear binary classifier

### Functional margin

- The value  $f(\mathbf{x})$  of the function for a certain point  $\mathbf{x}$  is called *functional margin*
- It can be seen as a confidence in the prediction

### Geometric margin

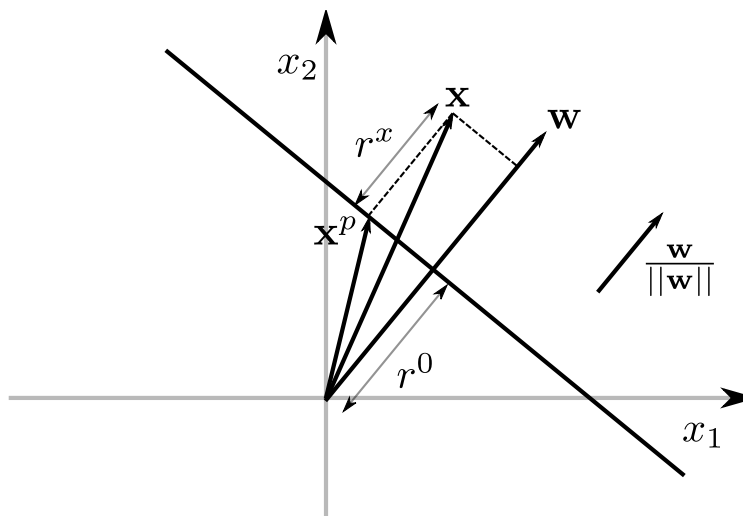
- The distance from  $\mathbf{x}$  to the hyperplane is called *geometric margin*

$$r^x = \frac{f(\mathbf{x})}{\|\mathbf{w}\|}$$

- It is a normalized version of the functional margin
- The distance from the origin to the hyperplane is:

$$r^0 = \frac{f(\mathbf{0})}{\|\mathbf{w}\|} = \frac{w_0}{\|\mathbf{w}\|}$$

## Linear binary classifier



### Geometric margin (cont)

- A point can be expressed by its projection on  $H$  plus its distance to  $H$  times the unit vector in that direction:

$$\mathbf{x} = \mathbf{x}^p + r^x \frac{\mathbf{w}}{\|\mathbf{w}\|}$$

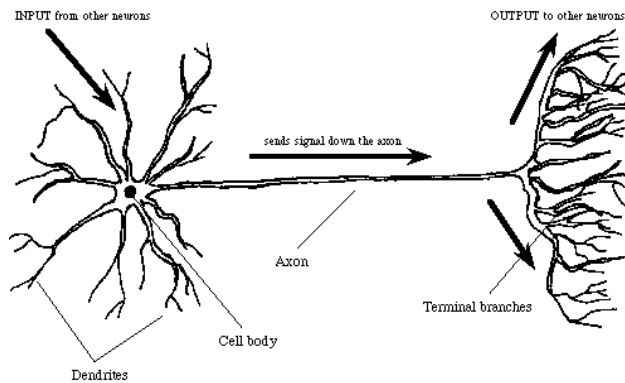
## Linear binary classifier

### Geometric margin (cont)

- Then as  $f(\mathbf{x}^p) = 0$ :

$$\begin{aligned}f(\mathbf{x}) &= \mathbf{w}^T \mathbf{x} + w_0 \\&= \mathbf{w}^T \left( \mathbf{x}^p + r^x \frac{\mathbf{w}}{\|\mathbf{w}\|} \right) + w_0 \\&= \underbrace{\mathbf{w}^T \mathbf{x}^p + w_0}_{f(\mathbf{x}^p)} + r^x \mathbf{w}^T \frac{\mathbf{w}}{\|\mathbf{w}\|} \\&= r^x \|\mathbf{w}\| \\ \frac{f(\mathbf{x})}{\|\mathbf{w}\|} &= r^x\end{aligned}$$

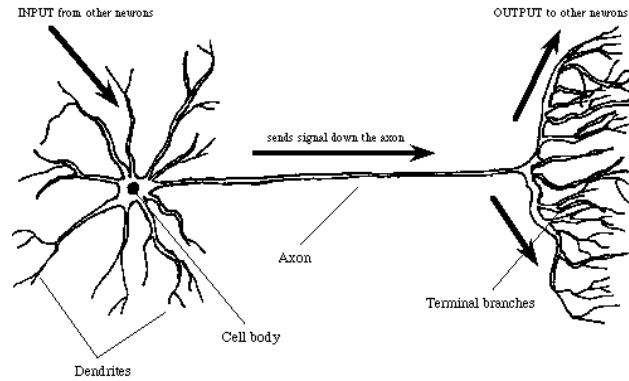
### Biological motivation



### Human Brain

- Composed of densely interconnected network of **neurons**
- A neuron is made of:
  - soma** A central body containing the nucleus
  - dendrites** A set of filaments departing from the body
  - axon** a longer filament (up to 100 times body diameter)
  - synapses** connections between dendrites and axons from other neurons

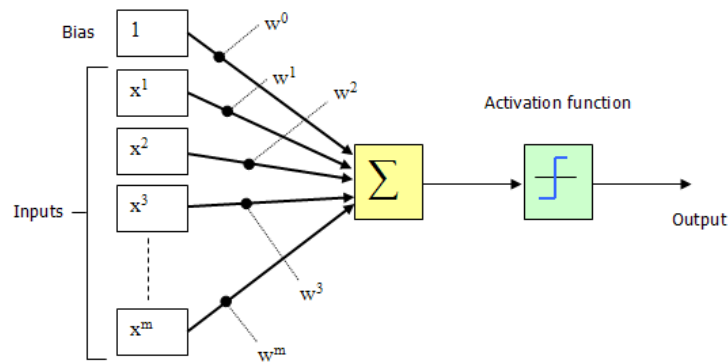
### Biological motivation



### Human Brain

- Electrochemical reactions allow signals to propagate along neurons via axons, synapses and dendrites
- Synapses can either *excite* or *inhibit* a neuron potential
- Once a neuron potential exceeds a certain *threshold*, a signal is generated and transmitted along the axon

### Perceptron



### Single neuron architecture

$$f(x) = \text{sign}(w^T x + w_0)$$

- Linear combination of input features
- Threshold activation function

### Perceptron

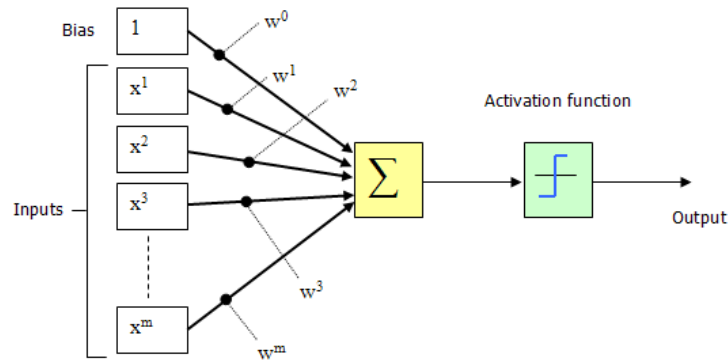
#### Representational power

- *Linearly separable* sets of examples
- E.g. primitive boolean functions (AND,OR,NAND,NOT)
- $\Rightarrow$  any logic formula can be represented by a network of two levels of perceptrons (in disjunctive or conjunctive normal form).

## Problem

- *non-linearly separable* sets of examples cannot be separated
- Representing complex logic formulas can require a number of perceptrons *exponential* in the size of the input

## Perceptron



### Augmented feature/weight vectors

$$f(x) = \text{sign}(\hat{\mathbf{w}}^T \hat{\mathbf{x}})$$

- Where bias is incorporated in augmented vectors:  
$$\hat{\mathbf{w}} = \begin{pmatrix} w_0 \\ \mathbf{w} \end{pmatrix}$$
$$\hat{\mathbf{x}} = \begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix}$$

- Search for weight vector + bias is replaced by search for augmented weight vector (we skip the “^” in the following)

## Parameter learning

### Error minimization

- Need to find a function of the parameters to be optimized (like in maximum likelihood for probability distributions)
- Reasonable function is measure of error on training set  $\mathcal{D}$  (i.e. the loss  $\ell$ ):

$$E(\mathbf{w}; \mathcal{D}) = \sum_{(\mathbf{x}, y) \in \mathcal{D}} \ell(y, f(\mathbf{x}))$$

- Problem of overfitting training data (less severe for linear classifier, we will discuss it)

## Parameter learning

### Gradient descent

1. Initialize  $\mathbf{w}$  (e.g.  $\mathbf{w} = 0$ )
2. Iterate until gradient is approx. zero:

$$\mathbf{w} = \mathbf{w} - \eta \nabla E(\mathbf{w}; \mathcal{D})$$

#### Note

- $\eta$  is called *learning rate* and controls the amount of movement at each gradient step
- The algorithm is guaranteed to converge to a local optimum of  $E(\mathbf{w}; \mathcal{D})$  (for small enough  $\eta$ )
- Too low  $\eta$  implies slow convergence
- Techniques exist to adaptively modify  $\eta$

## Parameter learning

### Problem

- The misclassification loss is piecewise constant
- Poor candidate for gradient descent

### Perceptron training rule

$$E(\mathbf{w}; \mathcal{D}) = \sum_{(\mathbf{x}, y) \in \mathcal{D}_E} -yf(\mathbf{x})$$

- $\mathcal{D}_E$  is the set of current training errors for which:

$$yf(\mathbf{x}) \leq 0$$

- The error is the sum of the functional margins of incorrectly classified examples

## Parameter learning

### Perceptron training rule

- The error gradient is:

$$\begin{aligned} \nabla E(\mathbf{w}; \mathcal{D}) &= \nabla \sum_{(\mathbf{x}, y) \in \mathcal{D}_E} -yf(\mathbf{x}) \\ &= \nabla \sum_{(\mathbf{x}, y) \in \mathcal{D}_E} -y(\mathbf{w}^T \mathbf{x}) \\ &= \sum_{(\mathbf{x}, y) \in \mathcal{D}_E} -y\mathbf{x} \end{aligned}$$

- the amount of update is:

$$-\eta \nabla E(\mathbf{w}; \mathcal{D}) = \eta \sum_{(\mathbf{x}, y) \in \mathcal{D}_E} y\mathbf{x}$$

## Perceptron learning

### Stochastic perceptron training rule

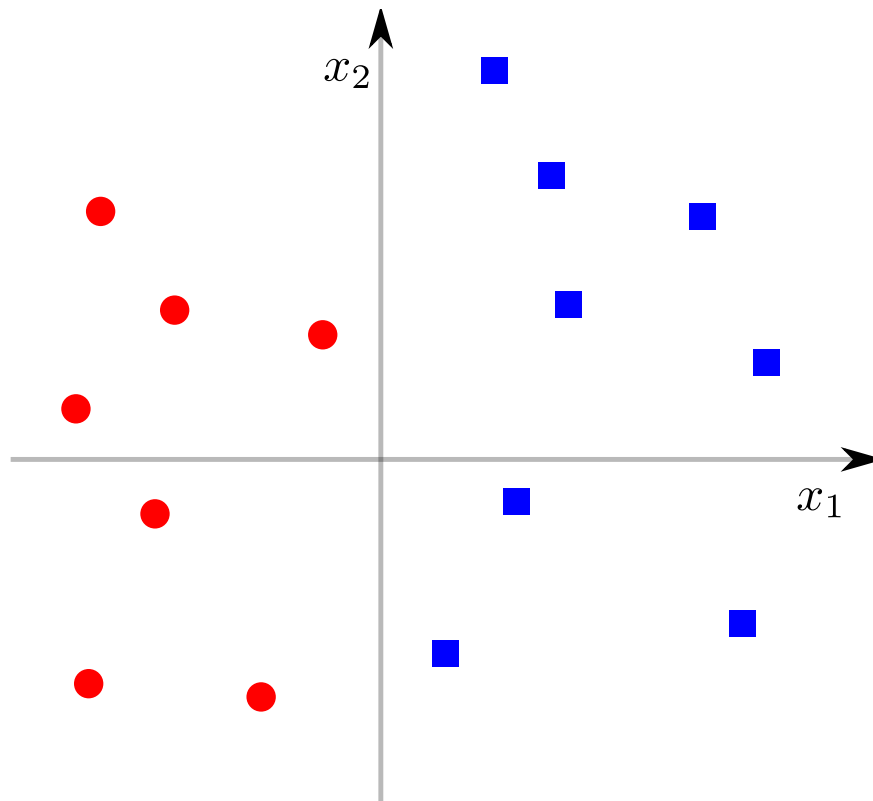
1. Initialize weights randomly
2. Iterate until all examples correctly classified:
  - (a) For each incorrectly classified training example  $(x, y)$  update weight vector:

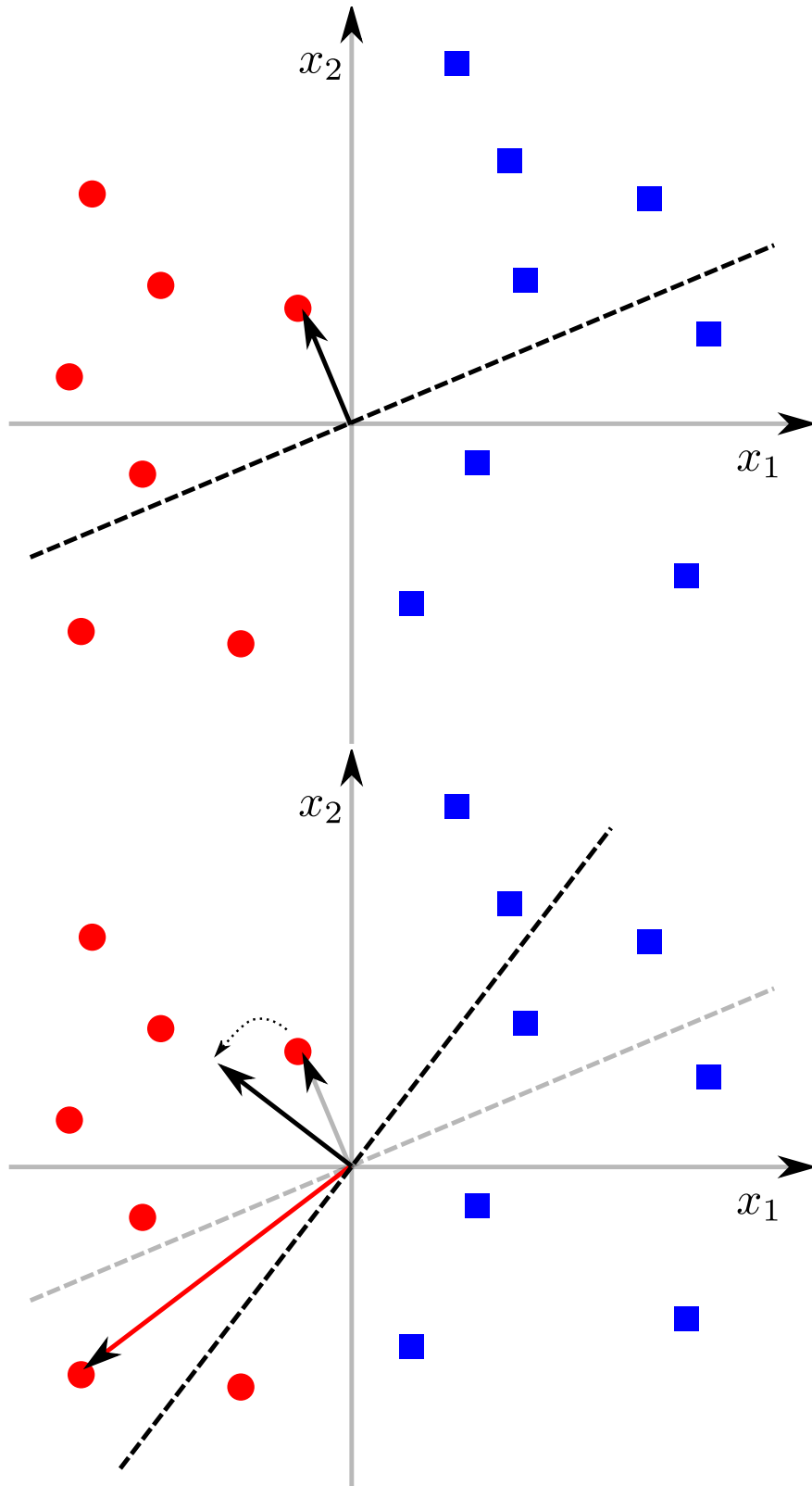
$$\mathbf{w} \leftarrow \mathbf{w} + \eta y \mathbf{x}$$

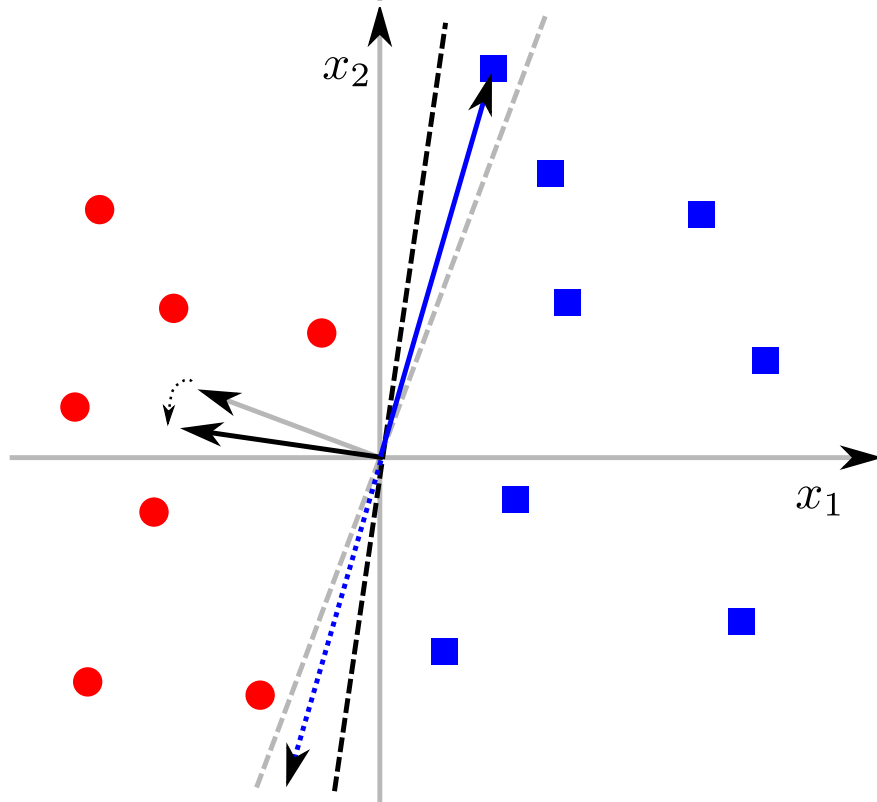
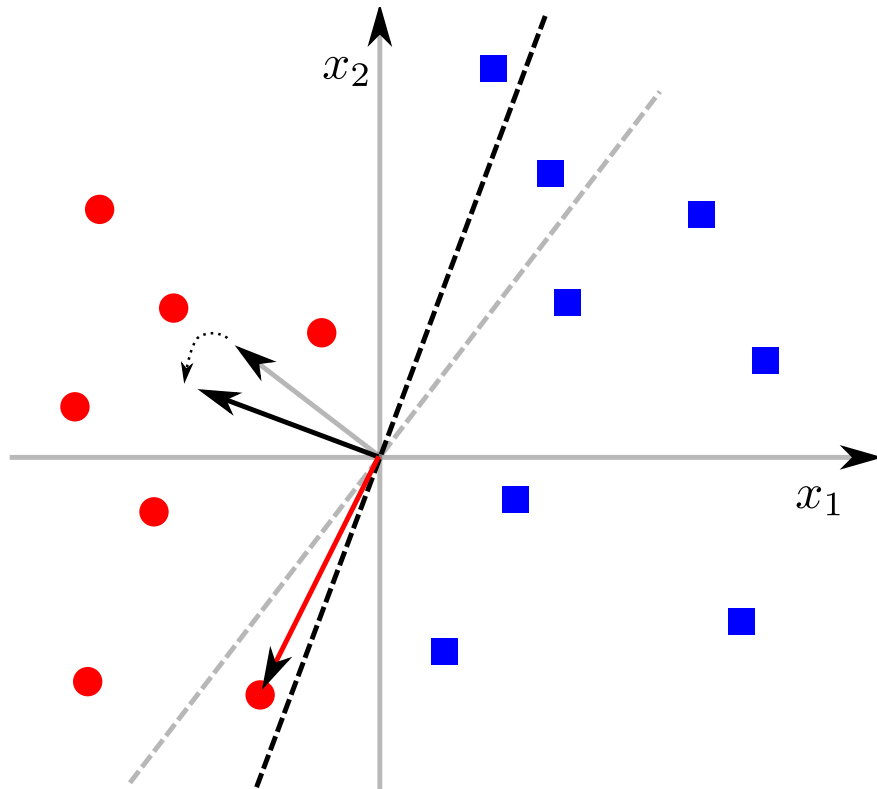
#### *Note on stochastic*

- we make a gradient step for each training error (rather than on the sum of them in *batch* learning)
- Each gradient step is very fast
- Stochasticity can sometimes help to avoid local minima, being guided by various gradients for each training example (which won't have the same local minima in general)

## Perceptron learning







## Perceptron regression

### Exact solution

- Let  $X \in \mathbb{R}^n \times \mathbb{R}^d$  be the input training matrix (i.e.  $X = [\mathbf{x}^1 \cdots \mathbf{x}^n]^T$  for  $n = |\mathcal{D}|$  and  $d = |\mathbf{x}|$ )
- Let  $\mathbf{y} \in \mathbb{R}^n$  be the output training matrix (i.e.  $y_i$  is output for example  $\mathbf{x}^i$ )
- Regression learning could be stated as a set of linear equations):

$$X\mathbf{w} = \mathbf{y}$$

- Giving as solution:

$$\mathbf{w} = X^{-1}\mathbf{y}$$

## Perceptron regression

### Problem

- Matrix  $X$  is rectangular, usually more rows than columns
- System of equations is overdetermined (more equations than unknowns)
- No exact solution typically exists

## Perceptron regression

### Mean squared error (MSE)

- Resort to loss minimization
- Standard loss for regression is the mean squared error:

$$E(\mathbf{w}; \mathcal{D}) = \sum_{(\mathbf{x}, y) \in \mathcal{D}} (y - f(\mathbf{x}))^2 = (\mathbf{y} - X\mathbf{w})^T (\mathbf{y} - X\mathbf{w})$$

- Closed form solution exists
- Can always be solved by gradient descent (can be faster)
- Can also be used as a classification loss

## Perceptron regression

### Closed form solution

$$\begin{aligned}\nabla E(\mathbf{w}; \mathcal{D}) &= \nabla (\mathbf{y} - X\mathbf{w})^T (\mathbf{y} - X\mathbf{w}) \\ &= 2(\mathbf{y} - X\mathbf{w})^T (-X) = 0 \\ &= -2\mathbf{y}^T X + 2\mathbf{w}^T X^T X = 0 \\ \mathbf{w}^T X^T X &= \mathbf{y}^T X \\ X^T X \mathbf{w} &= X^T \mathbf{y} \\ \mathbf{w} &= (X^T X)^{-1} X^T \mathbf{y}\end{aligned}$$

## Perceptron regression

$$\mathbf{w} = (X^T X)^{-1} X^T \mathbf{y}$$

Note

- $(X^T X)^{-1} X^T$  is called *left-inverse*
- If  $X$  is square and nonsingular, inverse and left-inverse coincide and the MSE solution corresponds to the exact one
- The left-inverse exists provided  $(X^T X) \in \mathbb{R}^{d \times d}$  is full rank  $\rightarrow$  features are linearly independent (if not, just remove the redundant ones!)

## Perceptron regression

### Gradient descent

- Closed-form solution requires computing  $(X^T X)^{-1}$  which can be expensive if  $d$  is large
- Gradient descent can be used instead:

$$\mathbf{w} = \mathbf{w} - \eta \nabla E(\mathbf{w}; \mathcal{D})$$

- where

$$\nabla E(\mathbf{w}; \mathcal{D}) = 2(\mathbf{y} - X\mathbf{w})^T (-X)$$

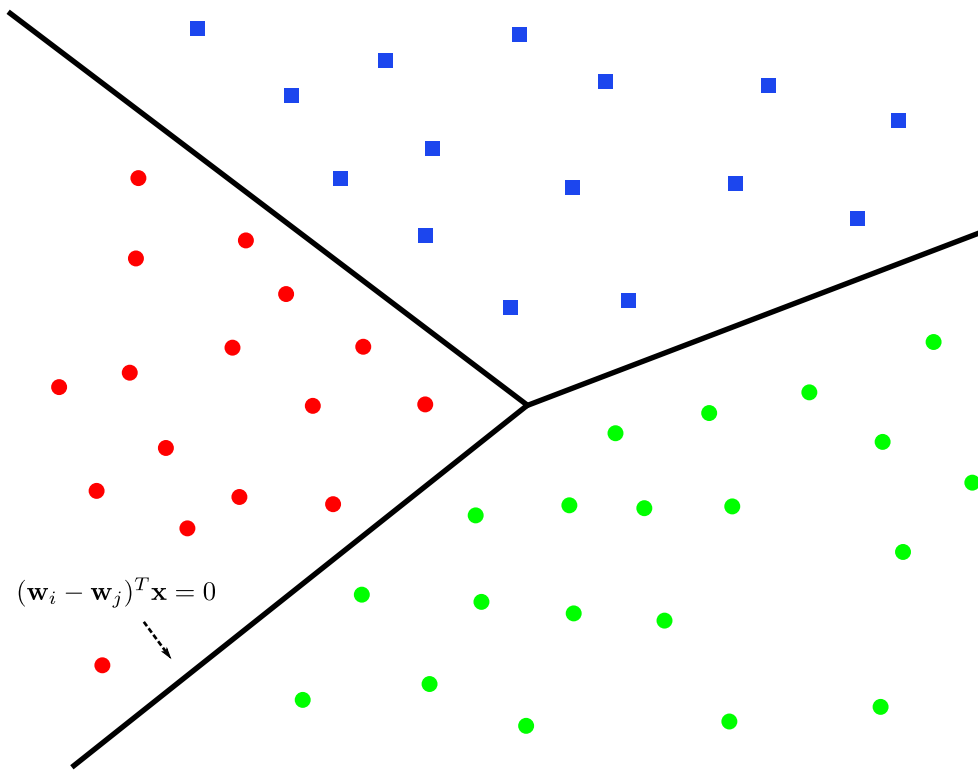
## Multiclass classification

### One-vs-all

- Learn one binary classifier for each class:
  - positive examples are examples of the class
  - negative examples are examples of all other classes
- Predict a new example in the class with maximum functional margin
- Decision boundaries for which  $f_i(\mathbf{x}) = f_j(\mathbf{x})$  are pieces of hyperplanes:

$$\begin{aligned} \mathbf{w}_i^T \mathbf{x} &= \mathbf{w}_j^T \mathbf{x} \\ (\mathbf{w}_i - \mathbf{w}_j)^T \mathbf{x} &= 0 \end{aligned}$$

## Multiclass classification



## Multiclass classification

### all-pairs

- Learn one binary classifier for each pair of classes:
  - positive examples from one class
  - negative examples from the other
- Predict a new example in the class winning the largest number of pairwise classifications

## Generative linear classifiers

### Gaussian distributions

- linear decision boundaries are obtained when covariance is shared among classes ( $\Sigma_i = \Sigma$ )

## Naive Bayes classifier

$$\begin{aligned} f_i(\mathbf{x}) = P(\mathbf{x}|y_i)P(y_i) &= \prod_{j=1}^{|\mathbf{x}|} \prod_{k=1}^K \theta_{ky_i}^{z_k(x[j])} \frac{|\mathcal{D}_i|}{|\mathcal{D}|} \\ &= \prod_{k=1}^K \theta_{ky_i}^{N_k \mathbf{x}} \frac{|\mathcal{D}_i|}{|\mathcal{D}|} \end{aligned}$$

- where  $N_{k\mathbf{x}}$  is the number of times feature  $k$  (e.g. a word) appears in  $\mathbf{x}$

### Generative linear classifiers

#### Naive Bayes classifier (cont)

$$\log f_i(\mathbf{x}) = \underbrace{\sum_{k=1}^K N_{k\mathbf{x}} \log \theta_{ky_i}}_{\mathbf{w}^T \mathbf{x}'} + \underbrace{\log\left(\frac{|\mathcal{D}_i|}{|\mathcal{D}|}\right)}_{w_0}$$

- $\mathbf{x}' = [N_{1\mathbf{x}} \cdots N_{K\mathbf{x}}]^T$
- $\mathbf{w} = [\log \theta_{1y_i} \cdots \log \theta_{Ky_i}]^T$
- Naive Bayes is a *log-linear* model (as Gaussian distributions with shared  $\Sigma$ )