Sapienza University of Rome

Master in Engineering in Computer Science

Artificial Intelligence & Machine Learning

A.Y. 2024/2025

Prof. Fabio Patrizi

3. Linear classification

Fabio Patrizi

Overview

- Linearly separable data
- Least squares
- Perceptron

References

- Lecture notes and slides
- [AIMA] 19.6.4, 19.6.5, 19.7.5
- T. Mitchell. Machine Learning. Section 4.4

Linear Models for Classification

Classification problem:

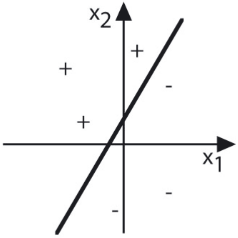
- Target function: $f : X \to Y$, where:
- $X \subseteq \mathbb{R}$
- $Y = \{c_1, \ldots, c_k\}$

Linearly Separable Data

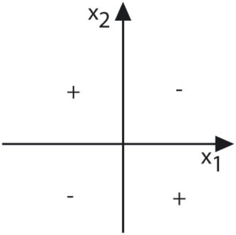
Linearly Separable Dataset:

- there exists a hyperplane separating instances from different classes

Example: $X \subseteq \mathbb{R}^2$, $Y = \{+, -\}$



(a)



(b)

Linear Discriminant Functions

- Linear Discriminant Function:
  - linear function $y(\mathbf{x})$ that defines separating hyperplanes
- Two classes ($\mathbf{x} = \langle x_1, x_2 \rangle$):
  - $y(x_1, x_2) = w_1 x_1 + w_2 x_2 + w_0$
  - prediction $h(\mathbf{x}) = \begin{cases} c_1, & \text{if } y(\mathbf{x}) \geq 0 \\ c_2, & \text{otherwise} \end{cases}$
  - separating hyperplane: $\{\langle x_1, x_2 \rangle \mid y(x_1, x_2) = 0\}$
- k classes ($\mathbf{x} = \langle x_1, \ldots, x_m \rangle$, k hyperplanes):
  - $y_1(x_1, \ldots, x_m) = w_{11} x_1 + \cdots + w_{1m} x_m + w_{10}$
  - $\cdots$
  - $y_k(x_1, \ldots, x_m) = w_{k1} x_1 + \cdots + w_{km} x_m + w_{k0}$
  - prediction: $h(\mathbf{x}) = c_i$, for $i = \text{argmax}_{i=1,\ldots,k}\, y_i(\mathbf{x})$
  - separating hyperplanes: $\{\langle x_1, \ldots, x_m \rangle \mid y_i(x_1, \ldots, x_m) = y_j(x_1, \ldots, x_m)\}$

Compact notation

- Two classes: $\tilde{\mathbf{x}} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$, $\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}$, $y(\mathbf{x}) = \mathbf{w}^T \tilde{\mathbf{x}}$
- k classes:
  - $\mathbf{y}(\mathbf{x}) = \begin{bmatrix} y_1(\mathbf{x}) \\ \cdots \\ y_k(\mathbf{x}) \end{bmatrix} = \mathbf{W}^T \tilde{\mathbf{x}}, \mathbf{W}^T = \begin{bmatrix} \mathbf{w}_1^T \\ \cdots \\ \mathbf{w}_k^T \end{bmatrix}, w_i = \begin{bmatrix} w_{i0} \\ w_{i1} \\ \cdots \\ w_{im} \end{bmatrix}, \tilde{\mathbf{x}} = \begin{bmatrix} 1 \\ x_1 \\ \cdots \\ x_m \end{bmatrix}$

Learning Linear Discriminants

- Given a dataset $D = \{\langle \mathbf{x}_1, t_1 \rangle, \ldots, \langle \mathbf{x}_N, t_N \rangle\}$
- with linearly separable data over k classes
- Determine $\mathbf{W}$ such that $\mathbf{y}(\mathbf{x}) = \mathbf{W}^T \tilde{\mathbf{x}}$ is the k-class discriminant

- Many learning approaches
- We see:
  - Least Squares
  - Perceptron

Least squares

Given $D = \{(\mathbf{x}_i, \mathbf{t}_n)_{n=1}^{N}\}$, find the linear discriminant

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

1-of-K coding scheme for $\mathbf{t}$: $\mathbf{x} \in C_k \rightarrow t_k = 1, t_j = 0$ for all $j \neq k$.
E.g., $\mathbf{t}_n = (0, \ldots, 1, \ldots, 0)^T$

$$\tilde{\mathbf{X}} = \begin{pmatrix} \tilde{\mathbf{x}}_1^T \\ \cdots \\ \tilde{\mathbf{x}}_i^t \end{pmatrix} \qquad \mathbf{T} = \begin{pmatrix} \mathbf{t}_1^T \\ \cdots \\ \mathbf{t}_i^t \end{pmatrix}$$

Least squares

Minimize sum-of-squares error function

$$E(\tilde{\mathbf{W}}) = \frac{1}{2}\, Tr\left\{ (\tilde{\mathbf{X}}\tilde{\mathbf{W}} - \mathbf{T})^T (\tilde{\mathbf{X}}\tilde{\mathbf{W}} - \mathbf{T}) \right\}$$

Closed-form solution:

$$\tilde{\mathbf{W}} = \underbrace{(\tilde{\mathbf{X}}^T \tilde{\mathbf{X}})^{-1} \tilde{\mathbf{X}}^T}_{\tilde{\mathbf{X}}^\dagger} \mathbf{T}$$

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

Least squares

Classification of new instance $\mathbf{x}$ not in dataset:

Use learnt $\tilde{\mathbf{W}}$ to compute:

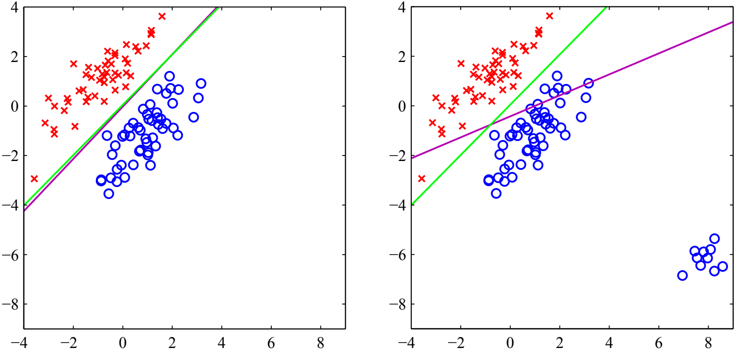
$$\mathbf{y}(\mathbf{x}) = \tilde{\mathbf{W}}^T \tilde{\mathbf{x}} = \begin{pmatrix} y_1(\mathbf{x}) \\ \cdots \\ y_K(\mathbf{x}) \end{pmatrix}$$

Assign class C_k to $\mathbf{x}$, where:

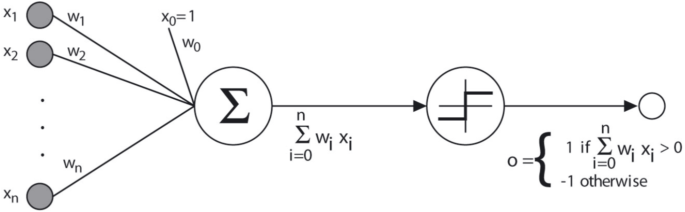
$$k = \underset{i \in \{1,\ldots,k\}}{\operatorname{argmax}} \{y_i(\mathbf{x})\}$$

Issues with least squares

Assume Gaussian conditional distributions. Not robust to outliers!



Perceptron



- For simplicity: $Y = \{+1, -1\}$, i.e., $k = 2$

$$h(x_1, \ldots, x_m) = \begin{cases} +1 & \text{if } w_0 + w_1 x_1 + \cdots + w_m x_m > 0 \\ -1 & \text{otherwise} \end{cases}$$

$$h(\mathbf{x}) = \begin{cases} +1 & \text{if } \mathbf{w}^T \tilde{\mathbf{x}} > 0 \\ -1 & \text{otherwise} \end{cases} = sign(\mathbf{w}^T \tilde{\mathbf{x}})$$

Perceptron training rule

Consider the *unthresholded* linear unit:

$$h(\mathbf{x}) = w_0 + w_1 x_1 + \cdots + w_m x_m = \mathbf{w}^T \tilde{\mathbf{x}}$$

- Learn w_i from training examples $D = \{\langle \mathbf{x}_1, t_1 \rangle, \ldots, \langle \mathbf{x}_N, t_N \rangle\}$
- minimizing the squared error (*Loss Function*)

$$E(\mathbf{w}) \equiv \frac{1}{2} \sum_{\langle \mathbf{x}, t \rangle \in D} (t - y(\mathbf{x}))^2 = \frac{1}{2} \sum_{\langle \mathbf{x}, t \rangle \in D} (t - \mathbf{w}^T \tilde{\mathbf{x}})^2$$

Perceptron training rule

$$
\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{\langle \mathbf{x}, t \rangle \in D} (t - \mathbf{w}^T \tilde{\mathbf{x}})^2 = \frac{1}{2} \sum_{\langle \mathbf{x}, t \rangle \in D} \frac{\partial}{\partial w_i} (t - \mathbf{w}^T \tilde{\mathbf{x}})^2 \\
&= \frac{1}{2} \sum_{\langle \mathbf{x}, t \rangle \in D} 2(t - \mathbf{w}^T \tilde{\mathbf{x}}) \frac{\partial}{\partial w_i} (t - \mathbf{w}^T \tilde{\mathbf{x}}) \\
&= \sum_{\langle \mathbf{x}, t \rangle \in D} (t - \mathbf{w}^T \tilde{\mathbf{x}}) \frac{\partial}{\partial w_i} (t - \mathbf{w}^T \tilde{\mathbf{x}}) \\
&= \sum_{\langle \mathbf{x}, t \rangle \in D} (t - \mathbf{w}^T \tilde{\mathbf{x}}) (-x_i)
\end{aligned}
$$

Perceptron training rule

Unthresholded unit:

Update of weights **w**

$$
\begin{aligned}
w_i &\leftarrow w_i + \Delta w_i \\
\Delta w_i &= -\eta \frac{\partial E}{\partial w_i} = \eta \sum_{\langle \mathbf{x}, t \rangle \in D} (t - \mathbf{w}^T \tilde{\mathbf{x}})\, x_i
\end{aligned}
$$

η is a small constant (e.g., 0.05) called *learning rate*

Perceptron training rule

Thresholded unit:

Update of weights **w**

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = \eta \sum_{\langle \mathbf{x}, t \rangle \in D} (t - sign(\mathbf{w}^T \tilde{\mathbf{x}}))\, x_i$$

Perceptron algorithm

Algorithm:

- Input: parameters of $h(\mathbf{x})$ ($\mathbf{w}$); dataset D
- Output: $\mathbf{w}$ assignment minimizing classification error (wrt to $h(\mathbf{x})$)

1. Initialize $\mathbf{w}$ (e.g., small random values)
2. Repeat until *termination condition*:
   - $w_i \leftarrow w_i + \Delta w_i$
3. Return $\mathbf{w}$

Typical termination conditions:

- number of iterations
- threshold on changes in $E(\mathbf{w})$

Perceptron algorithm

Weight-update variants (for every iteration):

- *Batch mode*: Consider entire dataset D
  - $\Delta w_i = \eta \sum_{\langle \mathbf{x}, t \rangle \in D} (t - h(\mathbf{x})) x_i$
- *Mini-Batch mode*: Choose a small subset $S \subset D$
  - $\Delta w_i = \eta \sum_{\langle \mathbf{x}, t \rangle \in S} (t - h(\mathbf{x})) x_i$
- *Incremental mode*: Choose one sample $\langle \mathbf{x}, t \rangle \in D$
  - $\Delta w_i = \eta (t - h(\mathbf{x})) x_i$

Incremental/mini-batch:

- speed up convergence
- less sensitive to local minima

Perceptron training rule

Recall:

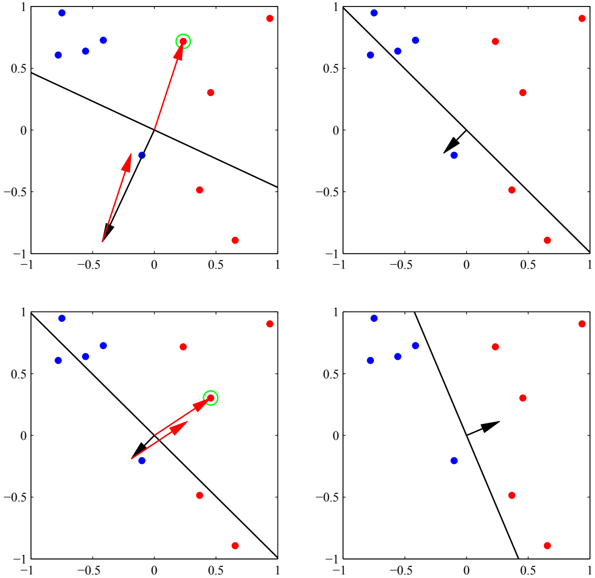
- $\mathbf{x} = \langle x_1, \ldots, x_m \rangle$
- $h(\mathbf{x}) = sign(y(\mathbf{x})) = sign(\mathbf{w}^T \tilde{\mathbf{x}}) = sign(w_0 + w_1 x_1 + \cdots + w_m x_m)$
- $\Delta w_i = \eta(t - h(\mathbf{x}))x_i$

Example:

$\eta = 0.1$, $x_i = 0.8$

- $t = h(\mathbf{x}) \Rightarrow \Delta w_i = 0$: keep w_i to keep y
- $t = 1$ and $h(\mathbf{x}) = -1 \Rightarrow \Delta w_i = 0.16$: increase w_i to increase y
- $t = -1$ and $h(\mathbf{x}) = 1 \Rightarrow \Delta w_i = -0.16$: decrease w_i to decrease y

Perceptron training rule



Perceptron training rule

Convergence proven under:

- linear separability of training data
- sufficiently small learning rate η

Note: Smaller η yields slower convergence

Perceptron: Prediction

Classification of new instance $\mathbf{x}$ not in dataset:

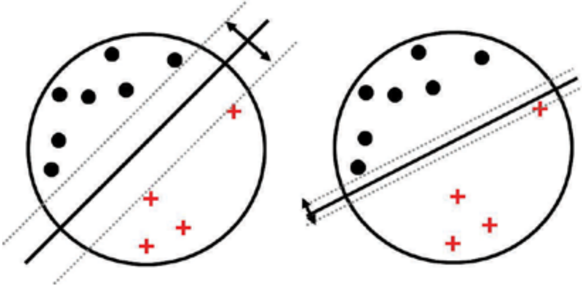
Classify $\mathbf{x}$ as $c_k = sign(\mathbf{w}^T \tilde{\mathbf{x}})$, using learnt $\mathbf{w}$

Can be generalized to k classes:

- train k linear units $y_i(\mathbf{x})$, one per class c_i
- $h(\mathbf{x}) = c_i$, for $i = \operatorname{argmax}_{i=1,\dots,k} y_i(\mathbf{x})$

Support Vector Machines

- Support Vector Machines (SVM) for Classification
  - maximum *margin* for better generalization



Support Vector Machines

Consider

- Consider binary classification $f : X \rightarrow \{+1, -1\}$
- Dataset $D = \{\langle \mathbf{x}_1, t_1 \rangle, \ldots, \langle \mathbf{x}_N, t_N \rangle\}$ $(t_i \in \{+1, -1\})$
- Linear model: $y(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0$ (we isolate w_0)

Assume D linearly separable:

- there exist $\bar{\mathbf{w}}, \bar{w}_0$ s.t., for $\bar{y}(\mathbf{x}) = \bar{\mathbf{w}}^T \mathbf{x} + \bar{w}_0$:
  - $t_i = \bar{h}(\mathbf{x}_i) = sign(\bar{y}(\mathbf{x}_i))$

Support Vector Machines: Margin

- Let $\mathbf{x}_k$ be point of D closest to separating hyperplane $\bar{y}(x) = 0$
- *Margin*: smallest distance between $\mathbf{x}_k$ and separator:

$$\text{margin} \; = \frac{|\bar{y}(\mathbf{x}_k)|}{||\bar{\mathbf{w}}||}$$

- Computed as (using property $|\bar{y}(\mathbf{x}_i)| = t_i \bar{y}(\mathbf{x}_i)$):

$$margin = \min_{i=1,\ldots,N} \frac{|\bar{y}(\mathbf{x}_i)|}{||\bar{\mathbf{w}}||} = \frac{1}{||\bar{\mathbf{w}}||} \min_{i=1,\ldots,N} t_i(\bar{\mathbf{w}}^T \mathbf{x}_i + \bar{w}_0)$$

Support Vector Machines: Maximum-margin Separator

- Given D, *Maximum-margin separator* is

$$y^*(\mathbf{x}) = {\mathbf{w}^*}^T \mathbf{x} + {w_0}^*, \text{ s.t.:}$$

$$\mathbf{w}^*, {w_0}^* = \underset{\mathbf{w}, w_0}{\operatorname{argmax}} \frac{1}{||\mathbf{w}||} \min_{i=1,\dots,N} t_i(\mathbf{w}^T \mathbf{x}_i + w_0)$$

- How to find maximum-margin separator?

Support Vector Machines: Maximum-margin Separator

- Maximum-margin hyperplane obtained as solution to optimization problem (proof omitted):

$$\mathbf{w}^*, w_0^* = \operatorname{argmax} \frac{1}{||\mathbf{w}||} \; = \; \operatorname{argmin} \frac{1}{2} ||\mathbf{w}||^2$$

subject to

$$t_i(\mathbf{w}^T \mathbf{x}_i + w_0) \geq 1 \;\; \forall n = 1, \ldots, N$$

- Quadratic programming problem solved with Lagrangian method

Support Vector Machines: Maximum-margin Separator

- Solution (proof omitted): $\mathbf{w}^* = \sum_{i=1}^{N} a_i^* t_i \mathbf{x}_i$
- a_i^* (Lagrange multipliers) results of optimization problem:

$$\mathbf{a}^* = \langle a_1^*, \ldots, a_N^* \rangle = \underset{\mathbf{a}}{\operatorname{argmax}} \sum_{i=1}^{N} a_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} a_i \, a_j \, t_i \, t_j \, \mathbf{x}_i^T \mathbf{x}_j$$

subject to:

- $a_i \geq 0$, for $i = 1, \ldots, N$
- $\sum_{i=1}^{N} a_i t_i = 0$

Support Vectors

- *Support vectors* (SV): points closest to separating hyperplane
- $a_i = 0$ except for support vectors $\mathbf{x}_i \in SV$ (proof omitted)
  - $\mathbf{w}^* = \sum_{i=1}^{N} a_i^* t_i \mathbf{x}_i = \sum_{\mathbf{x}_i \in SV}^{N} a_i^* t_i \mathbf{x}_i$
- Thus, hyperplane depends on support vectors only:
  - $y(\mathbf{x}) = \mathbf{w}^{*T}\mathbf{x} + w_0^* = \left(\sum_{\mathbf{x}_i \in SV} a_i^* t_i \mathbf{x}_i\right)^T \mathbf{x} + w_0^* = 0$
- Other points $\mathbf{x}_i \notin SV$ do not contribute ($a_i^* = 0$)
- $w_0^* = \frac{1}{|SV|} \sum_{\mathbf{x}_j \in SV} \left(t_j - \sum_{\mathbf{x}_i \in SV} a_i^* t_i \mathbf{x}_j^T \mathbf{x}_i \right)$

Support Vector Machines

- Given maximum-margin hyperplane determined by $\mathbf{w}^*$, w_0^*
- Classification of instance $\mathbf{x}$:

$$h(\mathbf{x}) = sign\left(\sum_{\mathbf{x}_i \in SV} a_i^* \, t_i \, \mathbf{x}^T \mathbf{x}_i + w_0^*\right)$$

- Observe: data points occur only as dot products, i.e., $\mathbf{x}_i^T \mathbf{x}_j$ (Importance of this discussed later)

Support Vector Machines

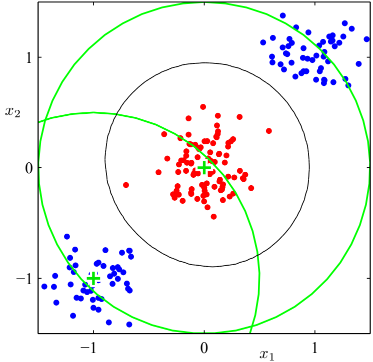
Observations:

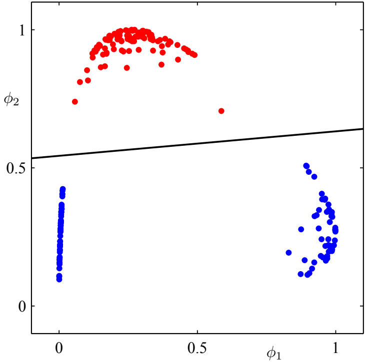
- Very powerful: maximum margin improves generalization
- No local minima (globally optimal solution)
- Analytical solution (software packages exist)
- Memory efficient: uses only a fraction of data points
- No parameters to tune (e.g., no learning rate)
- Effective also if data non-linearly separable (discussed later)

Non Linearly Separable Data

- So far models work directly on $\mathbf{x}$
- All results hold if apply non-linear transformation on input $\phi(\mathbf{x})$ (*basis functions*)
- Decision boundaries may be linear in feature space ϕ and non-linear in original space $\mathbf{x}$
- Classes linearly separable in feature space ϕ may be not in input space $\mathbf{x}$

Basis functions example





Basis functions examples

- Linear
- Polynomial
- Radial Basis Function (RBF)
- Sigmoid
- ...

Linear models for non-linear functions

Learning non-linear function

$$f : X \to \{c_1, \ldots, c_k\}$$

from data set D non-linearly separable.

Find a non-linear transformation ϕ and learn a linear model

$$y(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) + w_0 \textit{ (two classes)}$$

$$y_i(\mathbf{x}) = \mathbf{w}_i^T \phi(\mathbf{x}) + w_{i0} \textit{ (multiple classes)}$$

Summary

- Basic methods for learning linear classification functions
- Based on solving optimization problems
- Iterative vs. Analytical (closed form) solution
- Learning non-linear functions with linear models using basis functions
- Further developed as kernel methods