Linear classification

Damien Garreau, Rémi Flamary

January 29, 2020
Course overview

Introduction
   Learning problem
   Training data

Linear Discriminant Analysis
   Bayesian decision
   Regularized LDA

Logistic regression
   Optimization problem
   Gradient descent
   Newton’s descent
   Regularization

Rosenblatt’s perceptron
   Perceptron
   Optimization problem

Support vector machines
   Optimization problems

Conclusion on linear prediction
   Data fitting
Goal of the course

Introduction
- binary classification
- convex optimization

Linear classification
- logistic regression
- Rosenblatt’s Perceptron
- Support Vector Machines

Convex optimization
- gradient descent
- Newton’s descent
- stochastic gradient

We will focus on binary classification: only two classes
Linear Prediction

Reminder: linear functions
Function \( f : \mathbb{R}^d \rightarrow \mathbb{R} \), can be expressed as

\[
f(x) = \sum_{i=1}^{d} w_i x_i + b = x^\top w + b = [x^\top 1] a .
\]

with \( w \in \mathbb{R}^d \) a vector defining an hyperplane in \( \mathbb{R}^d \) and \( b \in \mathbb{R} \) a bias. All parameters can be stored in a unique vector \( a = \begin{bmatrix} b \\ w \end{bmatrix} \) of dimension \( d + 1 \).

Goal of linear prediction
- regression setting: \( f(\cdot) \in \mathbb{R} \).
- classification setting: \( \text{sign}(f(\cdot)) \in \{-1, 1\} \).
Linear classification

Goal
▶ obtain a linear function $f(\cdot)$ from which we predict binary values $y \in \{-1, 1\}$ from observations $x \in \mathbb{R}^d$
▶ in practice, we seek to estimate the coefficients $(b, w)^\top$ of $f(\cdot)$ using training data $\{x_i, y_i\}_{i=1,\ldots,n}$
▶ predicted class is selected as the sign of function $f(\cdot)$

Examples
▶ optical character recognition
▶ computer-aided diagnosis
▶ quality inspection
How do we store training data?

\[ \mathbf{X} = \begin{bmatrix} 1 & \mathbf{x}_1^\top \\ 1 & \mathbf{x}_2^\top \\ \vdots & \vdots \\ 1 & \mathbf{x}_i^\top \\ \vdots & \vdots \\ 1 & \mathbf{x}_n^\top \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1j} & \cdots & x_{1d} \\ 1 & x_{21} & x_{22} & \cdots & x_{2j} & \cdots & x_{2d} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 1 & x_{i1} & x_{i2} & \cdots & x_{ij} & \cdots & x_{id} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{nj} & \cdots & x_{nd} \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_i \\ \vdots \\ y_n \end{bmatrix} \]

Training data

- \( \mathbf{x}_i \in \mathbb{R}^d \) observations for \( i = 1, \ldots, n \)
- \( y_i \in \mathbb{R} \) values to predict for \( i = 1, \ldots, n \)

Matrix form:

- \( \mathbf{X} \in \mathbb{R}^{n \times (d+1)} \) such that \( \mathbf{x} = [\mathbf{e}, \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]^\top \) with \( \mathbf{e} \in \mathbb{R}^d \) and \( e_i = 1, \forall i \)
- \( \mathbf{y} \in \mathbb{R}^n \) such that \( \mathbf{y} = [y_1, y_2, \ldots, y_n]^\top \).
- \( \mathbf{a} \in \mathbb{R}^{d+1} \) is a vector such that \( \mathbf{a} = \begin{bmatrix} b \\ \mathbf{w} \end{bmatrix} \)
Section

Introduction
  Learning problem
  Training data

Linear Discriminant Analysis
  Bayesian decision
  Regularized LDA

Logistic regression
  Optimization problem
  Gradient descent
  Newton’s descent
  Regularization

Rosenblatt’s perceptron
  Perceptron
  Optimization problem

Support vector machines
  Optimization problems

Conclusion on linear prediction
  Data fitting
Linear Discriminant Analysis (LDA)

Bayesian decision method using likelihood ratio for predicting a class
we suppose that samples are drawn from Gaussian distributions $\mathcal{N}(\mu_1, \Sigma)$ for class $\omega_1$ and $\mathcal{N}(\mu_2, \Sigma)$ for class $\omega_2$
$p_1$ and $p_2$ are the probability of a sample being positive and negative
the decision function is linear thanks to the shared covariance $\Sigma$ between the classes
Likelihood ratio

Decision function

▶ **Assumption:** conditional probabilities density functions for each class are given by

$$p(x|\omega_1) = \text{det}(2\pi \Sigma)^{-\frac{1}{2}} \exp \left( -\frac{1}{2} (x - \mu_1)^\top \Sigma^{-1} (x - \mu_1) \right).$$

▶ if $p(\omega_1|x) > p(\omega_2|x)$, then choose $\omega_1$ else $\omega_2$:

$$p(\omega_1|x) \geq p(\omega_2|x) \iff p(x|\omega_1)p(\omega_1) \geq p(x|\omega_2)p(\omega_2)$$

▶ take for decision function $f$ such that:

$$f(x) = \log \left( \frac{p(\omega_1|x)}{p(\omega_2|x)} \right) = \log \left( \frac{p(x|\omega_1)p(\omega_1)}{p(x|\omega_2)p(\omega_2)} \right)$$

The function will be positive if $p(\omega_1|x) > p(\omega_2|x)$, negative otherwise. Its sign recovers the likelihood ratio decision.
Decision function

The decision function is linear!

\[
f(x) = \log \left( \frac{p(\omega_1 | x)}{p(\omega_2 | x)} \right) = \log \left( \frac{p(x | \omega_1)p(\omega_1)}{p(x | \omega_2)p(\omega_2)} \right)
\]

\[
= -\frac{1}{2} (x - \mu_1)^\top \Sigma^{-1} (x - \mu_1) + \frac{1}{2} (x - \mu_2)^\top \Sigma^{-1} (x - \mu_2) + \log(p_1) - \log(p_2)
\]

\[
= x^\top \Sigma^{-1} \mu_1 - \frac{1}{2} \mu_1^\top \Sigma^{-1} \mu_1 - x^\top \Sigma^{-1} \mu_2 + \frac{1}{2} \mu_2^\top \Sigma^{-1} \mu_2 + \log(p_1) - \log(p_2)
\]

\[
= x^\top \Sigma^{-1} (\mu_1 - \mu_2) - \frac{1}{2} \mu_1^\top \Sigma^{-1} \mu_1 + \frac{1}{2} \mu_2^\top \Sigma^{-1} \mu_2 + \log(p_1) - \log(p_2)
\]

\[
= x^\top \Sigma^{-1} (\mu_1 - \mu_2) + \frac{1}{2} (\mu_1 + \mu_2)^\top \Sigma^{-1} (\mu_1 - \mu_2) + \log(p_1) - \log(p_2)
\]

\[
= x^\top w + b
\]

with

\[
w = \Sigma^{-1} (\mu_1 - \mu_2), \quad \text{and} \quad b = \frac{1}{2} w^\top (\mu_1 + \mu_2) + \log(p_1) - \log(p_2) .
\]
Regularized LDA

LDA parameter estimation

\[ \mathbf{w} = \Sigma^{-1}(\mu_1 - \mu_2), \quad \text{and} \quad b = \frac{1}{2} \mathbf{w}^\top(\mu_1 + \mu_2) + \log(p_1) - \log(p_2). \]

- Σ, µ₁, µ₂, p₁, p₂ often unknown ⇒ estimated from data
- Σ can be non-invertible (not enough samples)
- numerical problems in high dimension

Regularized LDA parameter estimation

\[ \mathbf{w} = (\Sigma + \lambda \mathbf{I})^{-1}(\mu_1 - \mu_2), \quad \text{and} \quad b = \frac{1}{2} \mathbf{w}^\top(\mu_1 + \mu_2) + \log(p_1) - \log(p_2). \]

- I is the identity matrix and λ ≥ 0 a regularization parameter
- makes the matrix invertible and the solution unique
Fisher Discriminant Analysis

Multiclass LDA

- **Generalization:** samples from class $k$ are drawn from $\mathcal{N}(\mu_k, \Sigma)$
- define $\Sigma_b$ such that for all classes $1, \ldots, C$

$$
\Sigma_b = \frac{1}{C} \sum_{k=1}^{C} (\mu_k - \mu)(\mu_k - \mu)^\top, \quad \text{where} \quad \mu = \frac{1}{C} \sum_{k=1}^{C} \mu_k.
$$

Fisher Discriminant Analysis

$$
\max_{w, \|w\|=1} \frac{w^\top \Sigma_b w}{w^\top \Sigma w}
$$

- we look for a projection $w$ that **maximizes the distance between classes** while **minimizing the variance of each class**
- solutions of the problem are the eigenvectors of $\Sigma^{-1} \Sigma_b$
- special case with two classes: binary LDA
Conclusion on LDA

Pros

▶ probabilistic model
▶ closed-form solution when the distribution parameters are known
▶ regularization helps avoiding overfitting
▶ extension to multiclass with Fisher Discriminant Analysis

Cons

▶ Gaussian distribution parameters have to be estimated
▶ all classes are supposed to have the same covariance matrix $\Sigma$
Section

Introduction
  Learning problem
  Training data

Linear Discriminant Analysis
  Bayesian decision
  Regularized LDA

Logistic regression
  Optimization problem
  Gradient descent
  Newton’s descent
  Regularization

Rosenblatt’s perceptron
  Perceptron
  Optimization problem

Support vector machines
  Optimization problems

Conclusion on linear prediction
  Data fitting
Logistic regression

Goal

▶ train a linear discriminant function
▶ model directly the conditional probabilities (predict probabilities)
▶ avoid parameter estimations for distributions such as Gaussians

Approach

▶ we suppose a conditional probability $P(\omega_1|\mathbf{x})$ of the form:

$$P(\omega_1|\mathbf{x}) = \frac{\exp(\mathbf{w}^\top \mathbf{x} + b)}{1 + \exp(\mathbf{w}^\top \mathbf{x} + b)} = \frac{1}{1 + \exp(-\mathbf{w}^\top \mathbf{x} - b)},$$

and thus

$$P(\omega_2|\mathbf{x}) = 1 - P(\omega_1|\mathbf{x}) = \frac{1}{1 + \exp(\mathbf{w}^\top \mathbf{x} + b)}.$$
Likelihood ratio

Decision function

- we take as decision function the result of the likelihood ratio
- if $P(\omega_1|x) > P(\omega_2|x)$ then choose $\omega_1$ else $\omega_2$:

$$P(\omega_1|x) \geq P(\omega_2|x)$$

- the decision function $f$ such that:

$$f(x) = \log \left( \frac{P(\omega_1|x)}{P(\omega_2|x)} \right) = \log(\exp(w^\top x + b)) = w^\top x + b$$

- its sign recovers the decision of the likelihood ratio
Optimization problem

Log-likelihood
We want to \textbf{maximize} the log-likelihood on the data, which means minimizing:

\begin{equation}
J(b, w) = -\log \left( \prod_{i} P(y_i|x_i) \right) = -\sum_{i \in \mathcal{I}_1} \log(P(\omega_1|x_i)) - \sum_{i \in \mathcal{I}_2} \log(P(\omega_2|x_i))
\end{equation}

where \(\mathcal{I}_1\) and \(\mathcal{I}_2\) are the set of examples from class \(\omega_1\) and \(\omega_2\) respectively.

Objective function
We define the following objective function:

\begin{equation}
J(b, w) = \sum_{i \in \mathcal{I}_1} \log(1 + \exp(-w^\top x_i - b)) + \sum_{i \in \mathcal{I}_2} \log(1 + \exp(w^\top x_i + b))
\end{equation}

\begin{equation}
= \sum_{i} \log(1 + \exp(-y_i(w^\top x_i + b)))
\end{equation}
Gradient computation

- the function $J(b, w)$ is convex and differentiable. In order to compute its gradient we reformulate it as:

$$J(a) = \sum_i \log(1 + \exp(-y_i a^\top \tilde{x}_i)) .$$

- partial derivative of $J(a)$ with respect to $a_j$ is

$$\frac{\partial J(a)}{\partial a_j} = \sum_i -y_i (\tilde{x}_i)_j \exp(-y_i a^\top \tilde{x}_i) \frac{1}{1 + \exp(-y_i a^\top \tilde{x}_i)} = \sum_i -y_i (\tilde{x}_i)_j p_i \frac{1}{1 + p_i}$$

with $p_i = \exp(-y_i a^\top \tilde{x}_i)$

- gradient formulation in matrix form:

$$\nabla J(a) = -X^\top P y ,$$

where $P$ is a diagonal matrix of diagonal elements $\frac{p_i}{1+p_i}$ that depends on $a$.

- $\nabla J(a) = 0$ defines a **nonlinear equation** that cannot be solved in closed-form $\rightarrow$ Iterative optimization method.
Steepest gradient descent

Iterative optimization method

▸ **Idea:** update an approximate solution at each iteration
▸ at iteration $t$ the solution is updated with:

$$a^{(t)} = a^{(t-1)} + \mu_t d_t,$$

where $d_t$ is a descent direction which means that $d_t^T \nabla_\alpha J(a^{(t-1)}) < 0$ and $\mu_t > 0$ is the stepsize

Steepest gradient descent

▸ we take $d_t = -\nabla J(a^{(t-1)})$, that is, the **steepest descent direction**
▸ stepsize $\mu_t$ has to be chosen small enough to ensure descent
▸ each iteration decreases the objective value, but this can converge slowly
Algorithm of steepest GD

Initialization of $a$, $\mu$

delete repeat

d $\leftarrow$ $-\nabla J(a)$
a $\leftarrow$ $a + \mu d$
until convergence

Discussion

- sensitive to initialization of $a$
- we can ensure the decrease of the objective function at each iteration with a linesearch:

Backtracking method

Initialization of $\mu$ and $0 < \rho < 1$

repeat

$\mu \leftarrow \rho \mu$

until $J(a + \mu d) < J(a)$

- convergence conditions discussed later
Example of steepest descent

Simulation

- regularized logistic regression
- steepest gradient descent
- data \( (x_i, y_i) \) with \( d = 1 \):
  \( (1, -1), (2, -1), (3, 1), (4, 1) \)
- \( \mu = 0.1, \lambda = 1 \)
- 1000 iterations
- initialization \( a_0 = [1, -0.5] \)
- problem solution : \( a^* = [1, -2.5] \)

Discussion

- slow convergence around the solution
- after 1000 iterations, still not converged
- complexity \( \mathcal{O}(nd) \) per iteration
Hessian matrix

- the Hessian matrix of a differentiable function is the matrix $H \in \mathbb{R}^{(d+1) \times (d+1)}$ such that

$$H_{u,v} = \frac{\partial^2 J(a)}{\partial a_u \partial a_v}.$$ 

- contains all the second order derivatives of the multivariate function $J$.

- for logistic regression, we have

$$\frac{\partial^2 J(a)}{\partial a_u \partial a_v} = \sum_i (\tilde{x}_i)_u (\tilde{x}_i)_v \exp(-y_i a^\top \tilde{x}_i)}{(1 + \exp(-y_i a^\top \tilde{x}_i))^2} = \sum_i (\tilde{x}_i)_u (\tilde{x}_i)_v p_i \frac{1 + p_i}{(1 + p_i)^2}.$$ 

- can be expressed in matrix form as:

$$H = X^\top \tilde{P} X,$$

where $\tilde{P}$ is a diagonal matrix of diagonal element $\frac{p_i}{(1+p_i)^2}$.
Newton’s gradient descent

Principle

► **Idea:** minimize a quadratic approximation \( \tilde{J}(a) \) of the function \( J(a) \) at each iteration

► minimizing this approximation is equivalent to taking
\[
d = -H^{-1} \nabla_{\alpha} J(a^{(t-1)})
\]
as direction

► if the function \( J \) is convex, \( H \) is positive definite and \( d \) is provably a descent direction

Newtons’s gradient descent

Initialization of \( a, \mu, P = I \) and \( \tilde{P} = I \)

repeat

Update \( P \) and \( \tilde{P} \)
\[
d \leftarrow (X^\top \tilde{P} X)^{-1} X^\top P y
\]

\( a \leftarrow a + \mu d \)

until convergence

Discussion

► better convergence speed

► needs computation and inverse of Hessian

► iteration much more complex than steepest descent

► if the problem is quadratic, algorithm converges in **one step**
Example for Newton descent

Simulation

- regularized logistic regression
- Newton’s descent
- data \((x_i, y_i)\) with \(d = 1\):
  \((1, -1), (2, -1), (3, 1), (4, 1)\)
- \(\mu = 0.1, \lambda = 1\)
- 1000 iterations
- initialization \(a_0 = [1, -0.5]\)
- problem solution: \(a^* = [1, -2.5]\)

Discussion

- converges quickly to the solution
- after 5 iterations, same position as 100 with steepest descent
- complexity \(O(nd^2 + d^3)\) per iteration
Convergence and stopping conditions

Convergence

- for an iterative method, when do we stop?
- stationary point is reached by the algorithm if:

\[ \nabla_\alpha J(a) = 0. \]

Stopping conditions

In practice, algo is stopped if one of these conditions is met:

- norm of the gradient below a threshold: \( \| \nabla_\alpha J(a) \| < \varepsilon \)
- relative variation of the objective value below a threshold:

\[ \frac{|J(a^{(t)}) - J(a^{(t-1)})|}{J(a^{(t-1)})} < \varepsilon. \]

- maximum number of iterations reached: \( t = t_{max} \)
Regularization

A priori on $w$

- an *a priori* $p(w)$ of the probability distribution of $w$ can be easily added to the log-likelihood.
- if we suppose that $w \sim \mathcal{N}(0, \sigma^2 I)$, then we have
  $$p(w) \propto e^{-\frac{\|w\|^2}{2\sigma^2}}.$$
- maximizing the log-likelihood is then equivalent to minimizing:
  $$J(b, w) = -\log \left( p(w) \prod_i p(y_i|x_i) \right) = -\log (p(w)) - \log \left( \prod_i p(y_i|x_i) \right)$$
  $$= \sum_i \log(1 + \exp(-y_i a^\top \tilde{x}_i)) + \frac{1}{2\sigma^2} \|w\|^2.$$

The additional term is exactly the ridge regularization with $\lambda = \frac{1}{\sigma^2}$.
- other *a priori* about $w$ would lead to a different optimization problem
Conclusions about logistic regression

Pros

▶ probabilistic model
▶ convex problem, *strictly* convex with regularization
▶ regularization helps avoiding overfitting
▶ less parameters to estimate than Bayesian approaches such as LDA

\[ d + 1 \ll \underbrace{d^2 + 2d + 2}_{\text{LDA}} \]

Cons

▶ nonlinear problem to optimize and interpret
▶ iterative methods such as gradient descent are necessary
Section

Introduction
  Learning problem
  Training data

Linear Discriminant Analysis
  Bayesian decision
  Regularized LDA

Logistic regression
  Optimization problem
  Gradient descent
  Newton’s descent
  Regularization

Rosenblatt’s perceptron
  Perceptron
  Optimization problem

Support vector machines
  Optimization problems

Conclusion on linear prediction
  Data fitting
Method of Perceptron

History

▸ perceptron was proposed in 1957 by Frank Rosenblatt
▸ first very simple neuron (linear), biological inspiration

\[ f(x) = \sum_k w_k x_k + b. \]

▸ able to train only on separable data

Principle

▸ seek for an hyperplane defined by \( f(x) = w^\top x + b = 0 \) separating the classes
▸ iterative method with very small complexity per iteration
▸ update \((w, b)\) on mis-classified examples
▸ stop the iterations when all examples are well classified
Optimization problem

Objective function

Training the perceptron is equivalent to minimizing:

\[
J(a) = J(w, b) = - \sum_{i \in M} y_i(x_i^T w + b) = \sum_{i} \max(0, -y_i(x_i^T w + b)),
\]

where \( M \) is the set of all misclassified examples

Perceptron algorithm

Initialization of \( a \) and \( \mu > 0 \)

repeat

\[
\text{for } i \in I \text{ do}
\]

\[
\text{if } y_i \tilde{x}_i^T a < 0 \text{ then}
\]

\[
a \leftarrow a + \mu y_i \tilde{x}_i
\]

\[
\text{end if}
\]

\[
\text{end for}
\]

until \( y_i \tilde{x}_i^T a \geq 0, \forall i \)

Discussion

▶ \( I \) define the order in which the examples are selected.

▶ each iteration compute the gradient for a unique example

▶ “Stochastic Gradient Descent” (SGD) algorithm

▶ converges in a finite number of iterations if the data is separable
Example of perceptron solutions

Discussion

- each green line is a solution
- final solution depends on initialization and order of sample update
- note that the hyperplane are often close to one class or the other
- **Question:** do these solutions generalize well?
Conclusion on the perceptron

Advantages

▶ historical method
▶ find a solution in a finite number of iteration for separable data
▶ iterations are very cheap (SGD is still used a lot)

Inconvenients

▶ no unique solution
▶ no convergence for non-separable data
▶ risk of overfitting since no regularization
▶ bad performances proved in multi-class
Section

Introduction
  Learning problem
  Training data

Linear Discriminant Analysis
  Bayesian decision
  Regularized LDA

Logistic regression
  Optimization problem
  Gradient descent
  Newton’s descent
  Regularization

Rosenblatt’s perceptron
  Perceptron
  Optimization problem

Support vector machines
  Optimization problems

Conclusion on linear prediction
  Data fitting
Support vector machines

Idea

- find the hyperplane that maximizes the margin between the classes
- we want the samples to be well-classified with a margin, leading to the following constraints:

\[ y_i (\mathbf{w}^\top \mathbf{x} + b) \geq 1 \quad \forall i. \]
Optimization problem

- distance of a point to the hyperplane is defined as
  \[ d(x) = \frac{|w^\top x + b|}{\|w\|} \] .

- constraints \( y_i (w^\top x + b) \geq 1 \) ensure that the minimal distance of the samples to the hyperplane is equal to \( \frac{1}{\|w\|} \). The margin is then equal to
  \[ m = \frac{2}{\|w\|} \] .

- **maximizing** the margin is then equivalent to **minimizing** \( \|w\|^2 \)

- final support vector machine optimization problem is:
  \[
  \min_{w,b} \quad \|w\|^2 \\
  \text{s.t.} \quad y_i (w^\top x + b) \geq 1 \quad \forall 1 \leq i \leq n
  \]

  samples **exactly** on the margin \((w^\top x_k + b = y_k)\) are called support vectors
Optimization methods

Non-separable data (primal formulation)
When the margin constraints are relaxed the optimization problem becomes:

$$\min_{w,b} \sum_{i=1}^{n} \max(0, 1 - y_i (w^\top x_i + b)) + \frac{\lambda}{2} \|w\|^2.$$ 

Direct solver in the primal with a gradient descent approach.
→ Non-differentiable problem of size $d + 1$.

Dual formulation

$$\max_{\beta} \sum_{i=1}^{n} \beta_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i \beta_i (x_i^\top x_j) y_j \beta_j,$$

subject to $\sum_{i=1}^{n} \beta_i y_i = 0$, and $0 \leq \beta_i \leq \frac{1}{2\lambda}$ for all $i$.

The hyperplane can be recovered with $w = \sum_{i=1}^{n} \beta_i y_i x_i$
→ Constrained Quadratic Program (QP) of size $n$. 
Kernel Trick

- **Kernel** = a positive definite function of two samples that can be expressed as a scalar product:
  \[ k(x_1, x_2) = \phi(x_1)^\top \phi(x_2). \]

- Dual formulation of the problem depends only on scalar product and can be expressed with kernels:
  \[
  \max_\beta \quad \sum_{i=1}^{n} \beta_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i \beta_i k(x_i, x_j) y_j \beta_j, \\
  \text{subject to} \quad \sum_{i=1}^{n} \beta_i y_i = 0, \text{ and } 0 \leq \beta_i \leq \frac{1}{2n\lambda} \text{ for all } i.
  \]

- Prediction is then of the form:
  \[ f(x) = \sum_{i=1}^{n} \beta_i k(x, x_i) + b, \]

where \( b = y_j - \sum_{i=1}^{n} \beta_i k(x_j, x_i) \) can be estimated from a sample \( j \) on the margin (that is, \( f(x_k) = y_k \)).
Conclusion for support vector machines

Avantages
▶ optimization problem is strictly convex
▶ consistent method: converges to the Bayes classifier when \( n \to +\infty \)
▶ can be extended to non-linear classifier thanks to the kernel trick
▶ very good performances in practice, even on small datasets

Cons
▶ regularization parameter \( \lambda \) needs to be validated
▶ non-differentiable objective function.
▶ do not scale well in the dual (necessary for kernel formulation)
Section

Introduction
  Learning problem
  Training data

Linear Discriminant Analysis
  Bayesian decision
  Regularized LDA

Logistic regression
  Optimization problem
  Gradient descent
  Newton’s descent
  Regularization

Rosenblatt’s perceptron
  Perceptron
  Optimization problem

Support vector machines
  Optimization problems

Conclusion on linear prediction
  Data fitting
Regularized linear regression

General problem formulation:

\[
\min_{w,b} \sum_{i=1}^{n} \ell(y_i, w^\top x_i + b) + \lambda \Omega(w) \tag{1}
\]

With

- \(\ell(\cdot, \cdot)\) a loss function
- \(\Omega(\cdot)\) a regularization term

Examples:

**Loss function** \(L(y, \hat{y})\)

- \((y - \hat{y})^2\), quadratic
- \(|y - \hat{y}|\), absolute value
- \(\min(0, |y - \hat{y}| - \varepsilon)\) epsilon insensitive
- \(\max(0, 1 - y\hat{y})\), Hinge loss
- \(\log(1 + e^{-y\hat{y}})\), logistic

**Regularizations** \(\Omega(w)\)

- \(\|w\|_2^2\), quadratic
- \(\|w\|_1\), \(\ell_1\) norm
- \(w^\top \Sigma w\), Mahalanobis
**Data fitting for regression**

<table>
<thead>
<tr>
<th>Loss</th>
<th>$\ell(y, \hat{y})$</th>
<th>Smooth</th>
<th>Convex</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Square</strong></td>
<td>$(y - \hat{y})^2$</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Absolute value</strong></td>
<td>$</td>
<td>y - \hat{y}</td>
<td>$</td>
</tr>
<tr>
<td><strong>$\varepsilon$ insensible</strong></td>
<td>$\max(0,</td>
<td>y - \hat{y}</td>
<td>- \varepsilon)$</td>
</tr>
</tbody>
</table>

**Regression problem**

- **Objective**: predict a real value
- **Error if $y \neq \hat{y}$.
- **Error measure**: $|y - \hat{y}|$
## Data fitting for classification

<table>
<thead>
<tr>
<th>Cost</th>
<th>ℓ(y, ̂y)</th>
<th>Smooth</th>
<th>Convex</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1 loss</td>
<td>(1 − sgn(ŷ)) / 2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Hinge</td>
<td>max(0, 1 − ŷ)</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>Squared Hinge</td>
<td>max(0, 1 − ŷ)^2</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Logistic</td>
<td>log(1 + exp(−ŷ))</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Sigmoid</td>
<td>(1 − tanh(ŷ)) / 2</td>
<td>✓</td>
<td>-</td>
</tr>
<tr>
<td>Perceptron</td>
<td>max(0, −ŷ)</td>
<td>-</td>
<td>✓</td>
</tr>
</tbody>
</table>

### Regression problem

- **Objective**: predict a binary value
- error when \( y \neq \text{sign}(\hat{y}) \) i.e. if \( y \) and \( \hat{y} \) have a different sign.
- **Error measure**: \( y\hat{y} \)
- non-symmetric loss