Kernel methods

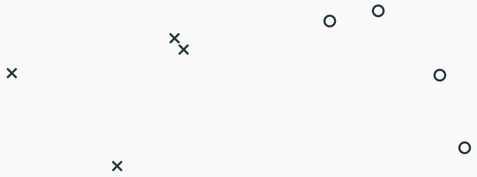
UE de Master 2, AOS1
Fall 2022

S. Rousseau

Introductory example

Let's look at one of the simplest binary classifier: the Euclidean classifier

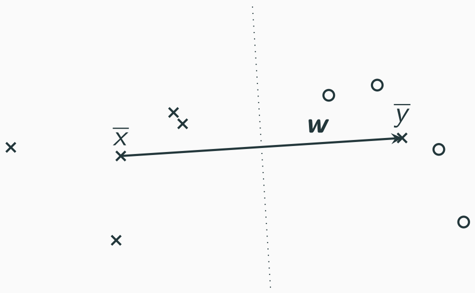
- Classification rule is: choose class whose class sample mean is the closest



Introductory example

Let's look at one of the simplest binary classifier: the Euclidean classifier

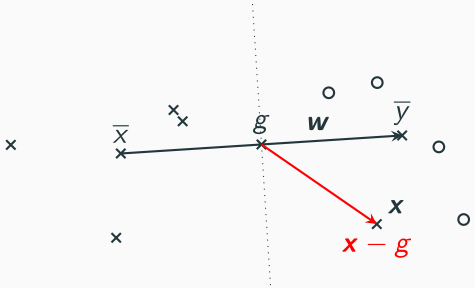
- Classification rule is: choose class whose class sample mean is the closest



Introductory example

Let's look at one of the simplest binary classifier: the Euclidean classifier

- Classification rule is: choose class whose class sample mean is the closest

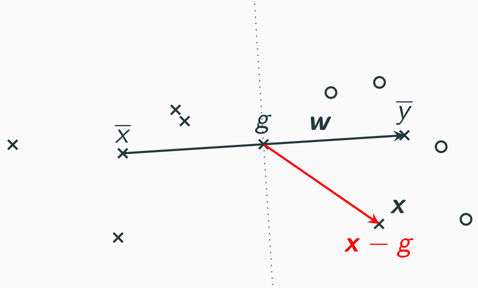


Decision rule

- Decision rule depends on the sign of an inner product

$$\langle \boldsymbol{x} - \boldsymbol{g}, \boldsymbol{w} \rangle$$

- Inner product leads to too simple decision boundary (hyperplane in n-d, line in 2-d)



How can we remedy to this?

Making my algorithm more powerful

Two solutions

1. Pragmatic but without guarantee solution: I replace the inner product by some other similarity
    - The similarity is easy to compute, but
    - I have no guarantee that it will work
2. Theoretical but hard in practice solution: I embed the samples in a possibly infinite dimensional feature space
    - My algorithm is the same but in a different space, but
    - Computations in feature space can be computationally intensive

Making my algorithm more powerful

Two solutions

1. Pragmatic but without guarantee solution: I replace the inner product by some other similarity
   - The similarity is easy to compute, but
   - I have no guarantee that it will work
2. Theoretical but hard in practice solution: I embed the samples in a possibly infinite dimensional feature space
   - My algorithm is the same but in a different space, but
   - Computations in feature space can be computationally intensive

The two solutions are actually the same!

(provided that the similarity is a **kernel**)

Kernel: formal definition

- Arbitrary input space $\mathcal{X}$ ($\mathbb{R}^p$, strings, graphs, ...)
- A **kernel** k defined on $\mathcal{X}$ is a map:

$$k : \mathcal{X} \times \mathcal{X} \to \mathbb{R},$$

  that verifies

  - **Symmetry**: $\forall \boldsymbol{x}, \boldsymbol{y} \in \mathcal{X}$, $k(\boldsymbol{x}, \boldsymbol{y}) = k(\boldsymbol{y}, \boldsymbol{x})$, matrix $(K)_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is symmetric
  - **Positive semidefiniteness**: For all $\boldsymbol{x}_1, \dots, \boldsymbol{x}_n \in \mathcal{X}$, matrix $(K)_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is positive semidefinite (PSD, $K \geqslant 0$):

$$\forall \boldsymbol{\alpha} \in \mathbb{R}^n, \boldsymbol{\alpha}^T K \boldsymbol{\alpha} = \sum_{ij} \alpha_i \alpha_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) \geqslant 0$$

Some examples

- Linear kernel (inner product is of course a kernel): $\mathcal{X} = \mathbb{R}^p$, $k(\boldsymbol{x}, \boldsymbol{y}) = \langle \boldsymbol{x}, \boldsymbol{y} \rangle$
- Polynomial kernel: let $\mathcal{X} = \mathbb{R}^p$, $d > 0, c \geqslant 0$

$$k_{\text{poly1}}(\boldsymbol{x}, \boldsymbol{y}) = (c + \langle \boldsymbol{x}, \boldsymbol{y} \rangle)^d$$

- Gaussian kernel or RBF (radial basis function): $k_{\text{gauss}}(\boldsymbol{x}, \boldsymbol{y}) = \exp\left\{-\frac{\|\boldsymbol{x}-\boldsymbol{y}\|_2^2}{\sigma^2}\right\}$
- Sigmoid kernel: $k(\boldsymbol{x}, \boldsymbol{y}) = \tanh\left(\gamma \langle \boldsymbol{x}, \boldsymbol{y} \rangle + c\right)$

From solution #2 to solution #1

Embedding actually defines a kernel

- Let $\Phi : \mathcal{X} \longmapsto \mathcal{F}$ some embedding of $\mathcal{X}$ in a **feature space** $\mathcal{F}$ that is equipped with an inner production $\langle \cdot, \cdot \rangle_{\mathcal{F}}$.
- Define k as follows

$$k(\boldsymbol{x}, \boldsymbol{y}) = \langle \Phi(\boldsymbol{x}), \Phi(\boldsymbol{y}) \rangle_{\mathcal{F}}$$

- Define that way, k is always a kernel!
    - Obviouly symmetric
    - Inner product in $\mathcal{F}$ makes k PSD

From solution #1 to solution #2: the polynomial kernel case

Suppose $\mathcal{X} = \mathbb{R}^2$, $n = 2$, $c = 1$ and $d = 2$

- Polynomial kernel is then

$$\begin{aligned} k_{\text{poly1}}(\boldsymbol{x}, \boldsymbol{y}) &= (1 + \langle \boldsymbol{x}, \boldsymbol{y} \rangle)^2 \\ &= (\langle \boldsymbol{x}, \boldsymbol{y} \rangle + 1)^2 = (1 + x_1 y_1 + x_2 y_2)^2 \\ &= 1 + 2x_1 y_1 + 2x_2 y_2 + x_1^2 y_1^2 + x_2^2 y_2^2 + 2x_1 x_2 y_1 y_2 \end{aligned}$$

- Define $\Phi(\boldsymbol{x}) = \left(1, x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x_2\right)$, we have

$$k_{\text{poly1}}(\boldsymbol{x}, \boldsymbol{y}) = \langle \Phi(\boldsymbol{x}), \Phi(\boldsymbol{y}) \rangle$$

- The kernel k_{poly1} is can be interpreted as an inner product

From solution #1 to solution #2: when $\mathcal{X}$ is finite

From the definition, K is a kernel iff K is symetric PSD

- K is diagonalizable in an orthonormal basis $K = UDU^T$ with U orthogonal and D diagonal

$$\begin{aligned} k(\boldsymbol{x}_i, \boldsymbol{x}_j) &= K_{ij} \\ &= (UDU^T)_{ij} \\ &= \mathrm{u}_i^T D \mathrm{u}_j \\ &= \mathrm{u}_i^T \sqrt{D}\sqrt{D}\mathrm{u}_j \\ &= \left\langle \sqrt{D}\mathrm{u}_i, \sqrt{D}\mathrm{u}_j \right\rangle \end{aligned}$$

Define the function Φ

$$\begin{aligned} \Phi \colon \mathcal{X} &\to \mathbb{R}^n \\ \boldsymbol{x}_i &\mapsto \sqrt{D}\mathrm{u}_i. \end{aligned}$$

we have $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \Phi(\boldsymbol{x}_i), \Phi(\boldsymbol{x}_j) \rangle$

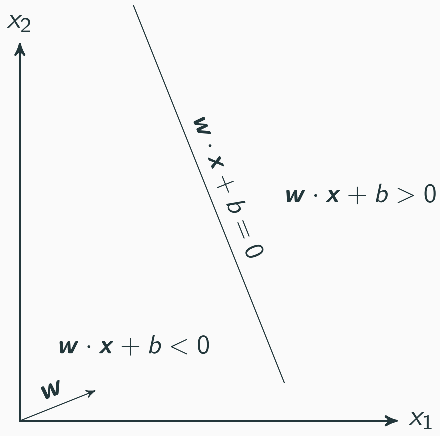
- The kernel k is can also be interpreted as an inner product

Kernel trick

- Kernel trick
  $$k(\boldsymbol{x}, \boldsymbol{y}) = \langle \Phi(\boldsymbol{x}), \Phi(\boldsymbol{y}) \rangle_{\mathcal{F}}$$
  No need to compute the embedding and then the inner product, the kernel is exactly doing this.
- Any algorithm using **only inner products** between training samples can be "kernelized"
- Most famous kernel method: SVM (support vector machines)

Separating hyperplane

- In $\mathbb{R}^p$, a hyperplane is defined by $h(\boldsymbol{x}) = 0$ with $h(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} + b$.


x_2
$\boldsymbol{w} \cdot \boldsymbol{x} + b = 0$
$\boldsymbol{w} \cdot \boldsymbol{x} + b > 0$
$\boldsymbol{w} \cdot \boldsymbol{x} + b < 0$
$\boldsymbol{w}$
x_1

- $\boldsymbol{w}$ is orthogonal to the hyperplane
- $d(\boldsymbol{x}, H) = \frac{|\boldsymbol{w} \cdot \boldsymbol{x} + b|}{\|\boldsymbol{w}\|_2}$
- Distance from the origin is $\frac{|b|}{\|\boldsymbol{w}\|_2}$

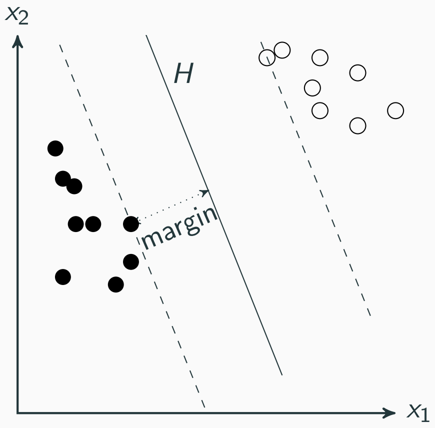
Margin

- H a separating hyperplane
- Distance from one $\boldsymbol{x}_i$ to H is

$$d(\boldsymbol{x}_i, H) = \frac{|\boldsymbol{w} \cdot \boldsymbol{x}_i + b|}{\|\boldsymbol{w}\|_2}$$

- Margin of a separating hyperplan is the smallest distance from H to the $\boldsymbol{x}_i$'s

$$M = \min_{i=1,\dots,n} d(\boldsymbol{x}_i, H)$$

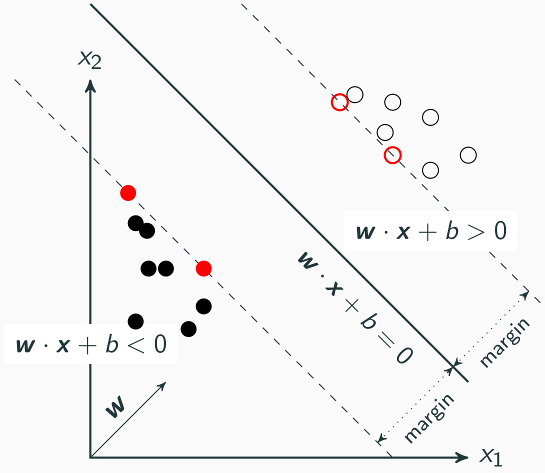


Optimal separating hyperplane

- Optimal separating hyperplane maximizes the margin
- Unique solution
- Vectors $\boldsymbol{x}_i$ supporting the margin, *i.e.* such that

$$M = d(\boldsymbol{x}_i, H)$$

  are called **support vectors**
- $H = \arg\max_{H \in \mathcal{H}} \min_{i=1,\ldots,n} d(\boldsymbol{x}_i, H)$


x_2
x_1
$\boldsymbol{w} \cdot \boldsymbol{x} + b > 0$
$\boldsymbol{w} \cdot \boldsymbol{x} + b = 0$
$\boldsymbol{w} \cdot \boldsymbol{x} + b < 0$
$\boldsymbol{w}$
margin
margin

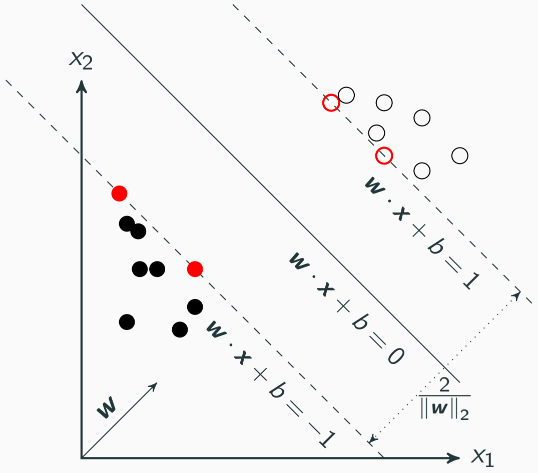
One-to-one representation

- Hyperplane $\Longleftrightarrow$ $(\boldsymbol{w}, b)$ is not one-to-one
  If $(\boldsymbol{w}, b)$ represents H so is $(c\boldsymbol{w}, cb)$ with $c \neq 0$
- Possible choice for c
  1. Data independant: imposing $\|\boldsymbol{w}\|_2 = 1$ (taking $c = \frac{1}{\|\boldsymbol{w}\|_2}$)
  $$(\boldsymbol{w}, b) \quad \text{with} \quad \|\boldsymbol{w}\|_2 = 1 \tag{normalization 1}$$
  2. Data dependant: $\boldsymbol{w} \cdot \boldsymbol{x} + b$ is ± 1 at margin lines
  $$(\boldsymbol{w}, b) \quad \text{with} \quad \|\boldsymbol{w}\|_2 = \frac{1}{M} \tag{normalization 2}$$
  When H is an optimal separating hyperplane, support vectors are ± 1

Optimal separating hyperplane

Choosing data-dependent normalization

- Margin is $\frac{1}{\|\boldsymbol{w}\|_2}$
- Takes the value ± 1 for support vectors



x_2
x_1
$\boldsymbol{w}$
$\boldsymbol{w} \cdot \boldsymbol{x} + b = 1$
$\boldsymbol{w} \cdot \boldsymbol{x} + b = 0$
$\boldsymbol{w} \cdot \boldsymbol{x} + b = -1$
$\frac{2}{\|\boldsymbol{w}\|_2}$

Maximizing the margin

- Maximizing $\frac{1}{\|\boldsymbol{w}\|_2} \iff$ Minimizing $\|\boldsymbol{w}\|_2^2$
- Proper labelling:

$$\left.\begin{array}{l} \bullet\ \boldsymbol{w} \cdot \boldsymbol{x}_i + b \geqslant 1 \text{ when } y_i = 1 \\ \bullet\ \boldsymbol{w} \cdot \boldsymbol{x}_i + b \leqslant -1 \text{ when } y_i = -1 \end{array}\right\} y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \geqslant 1$$

$$\boxed{\underset{(\boldsymbol{w},b)}{\arg\min}\ \frac{1}{2}\|\boldsymbol{w}\|_2^2 \quad \text{such that} \quad \forall i,\ y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \geqslant 1} \qquad (1)$$

- Quadratic programming problem: quadratic objective, linear constraints

Lagrangian formulation

- We use the Lagrangian, n Lagrange multipliers $\mu_1, \dots, \mu_n$

$$\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\mu}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2 - \sum_{i=1}^{n} \mu_i \left(y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) - 1\right)$$

- Karush–Kuhn–Tucker (KKT) conditions because we have inequalities

$$\begin{cases} \dfrac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = \boldsymbol{w} - \displaystyle\sum_{i=1}^{n} \mu_i y_i \boldsymbol{x}_i = 0 \\ \dfrac{\partial \mathcal{L}}{\partial b} = -\displaystyle\sum_{i=1}^{n} \mu_i y_i = 0 \end{cases} \implies \begin{cases} \boldsymbol{w} = \displaystyle\sum_{i=1}^{n} \mu_i y_i \boldsymbol{x}_i \\ \displaystyle\sum_{i=1}^{n} \mu_i y_i = 0 \end{cases}$$

$$\mu_i \geqslant 0 \quad \text{et} \quad \mu_i \left(y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) - 1\right) = 0$$

Dual formulation

- Plugging the KKT conditions back in the objective function we have

$$\mathcal{L}(\boldsymbol{\mu}, b) = \frac{1}{2}\left\|\sum_{j=1}^{n}\mu_j y_j \boldsymbol{x}_j\right\|_2^2 - \sum_{i=1}^{n}\mu_i\left(y_i\left(\sum_{j=1}^{n}\mu_j y_j \boldsymbol{x}_j \cdot \boldsymbol{x}_i + b\right) - 1\right)$$

$$\mathcal{L}(\boldsymbol{\mu}) = \frac{1}{2}\left\|\sum_{j=1}^{n}\mu_j y_j \boldsymbol{x}_j\right\|_2^2 - \sum_{i,j=1}^{n}\mu_i\mu_j y_i y_j \boldsymbol{x}_i \cdot \boldsymbol{x}_j + \sum_{i=1}^{n}\mu_i$$

$$= -\frac{1}{2}\left\|\sum_{j=1}^{n}\mu_j y_j \boldsymbol{x}_j\right\|_2^2 + \sum_{i=1}^{n}\mu_i$$

Dual formulation

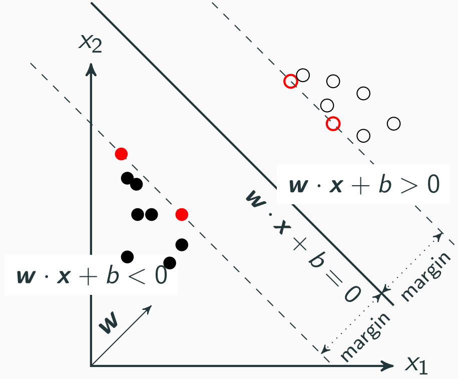
SVM minimization problem (dual form)

$$\boxed{\arg\min_{\boldsymbol{\mu}} \frac{1}{2}\sum_{i=1}^{n} \mu_i\mu_j y_i y_j \boldsymbol{x}_i \cdot \boldsymbol{x}_j - \sum_{i=1}^{n} \mu_i \quad \text{such that} \quad \begin{cases} \forall i,\ \mu_i \geqslant 0 \\ \sum_{i=1}^{n} \mu_i y_i = 0 \end{cases}}$$

- Only depends on $\boldsymbol{x}_i \cdot \boldsymbol{x}_j$: SVM algorithm is kernelizable

Support vectors

- $y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) > 1$: $\mu_i = 0$, $\boldsymbol{x}_i$ is not used
- $y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) = 1$: $\mu_i > 0$, $\boldsymbol{x}_i$ is on the margin, it is a *support vector*



x_2
$\boldsymbol{w} \cdot \boldsymbol{x} + b > 0$
$\boldsymbol{w} \cdot \boldsymbol{x} + b = 0$
$\boldsymbol{w} \cdot \boldsymbol{x} + b < 0$
$\boldsymbol{w}$
margin
margin
x_1

- Only support vectors are used to classify

Classification

- Classification of a new example $\boldsymbol{x}$:

$$\begin{aligned} y(\boldsymbol{x}) &= \operatorname{sgn}(\boldsymbol{w} \cdot \boldsymbol{x} + b) \\ &= \operatorname{sgn}\left(\sum_{i=1}^{n} \mu_i y_i \boldsymbol{x}_i \cdot \boldsymbol{x} + b\right) \end{aligned}$$

- Only the support vectors are used to classify
  - We only need to compute $\boldsymbol{x}_i \cdot \boldsymbol{x}$ with $\boldsymbol{x}_i$ a support vector
  - Robust to outliers in training data
- How to determine b:
  - If $\boldsymbol{x}_i$ is a support vector, $y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) = 1$ hence

$$b = y_i - \boldsymbol{w} \cdot \boldsymbol{x}_i$$

  - More stable numerically

$$\frac{1}{\#\{i, \mu_i > 0\}} \sum_{i,\, \mu_i > 0} y_i - \boldsymbol{w} \cdot \boldsymbol{x}_i$$

Extension to the non-separable case

- Until now, classes were was supposed to be linearly separable but:
  - it is never the case in practice
  - minimization problem (1) or dual form do not even have a solution: set of admissible solution is empty
- Typical solution: relaxing the constraints

Slack variables

- Original minimization problem

$$\boxed{\arg\min_{(\boldsymbol{w},b)} \frac{1}{2}\|\boldsymbol{w}\|_2^2 \quad \text{such that} \quad \forall i,\ y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i + b) \geqslant 1}$$

- Some constraint might be violated; we give them some slack by introducing nonnegative **slack variables** $\xi_1, \ldots, \xi_n$ so that

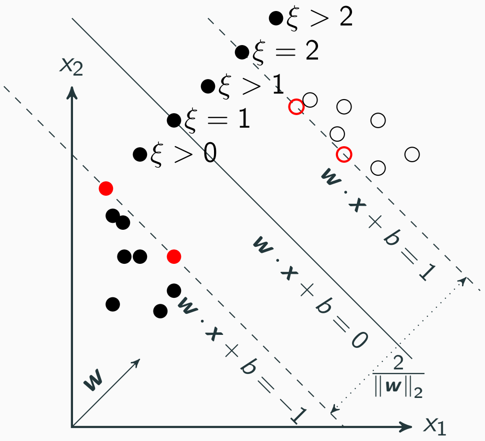
$$\forall i,\ y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i + b) \geqslant 1 \quad \text{becomes} \quad \forall i,\ y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i + b) \geqslant 1 - \xi_i$$

- The ξ_i's catch up on mistakes

Slack variables: geometric interpretation

New constraints: $y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \geqslant 1 - \xi_i$

- If $\xi_i = 0$ the original constraint is met
- If $0 < \xi_i < 2$ the original constraint is violated and the sample $\boldsymbol{x}_i$ belongs to the margin
- If $\xi_i \geqslant 2$ the original constraint is violated and the sample $\boldsymbol{x}_i$ belong to the wrong side



$\xi > 2$
$\xi = 2$
$\xi > 1$
$\xi = 1$
$\xi > 0$
x_2
x_1
$\boldsymbol{w}$
$\boldsymbol{w} \cdot \boldsymbol{x} + b = 1$
$\boldsymbol{w} \cdot \boldsymbol{x} + b = 0$
$\boldsymbol{w} \cdot \boldsymbol{x} + b = -1$
$\frac{2}{\|\boldsymbol{w}\|_2}$

Soft margin minimization problem

- Still maximizing the margin but regularizing by the amount of slack $\sum_{i=1}^{n} \xi_i$

$$\underset{(\boldsymbol{w},b)}{\arg\min} \frac{1}{2} \|\boldsymbol{w}\|_2^2 + C \sum_{i=1}^{n} \xi_i \quad \text{such that} \quad \begin{cases} \forall i,\ y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \geqslant 1 - \xi_i \\ \xi_i \geqslant 0 \end{cases}$$

- We introduce a new hyperparameter C: cost of adding some slack
    - Hard margin if $C \to +\infty$: no slack

Hinge loss

- Equivalence of constraints

$$\begin{cases} \forall i,\ y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \geqslant 1 - \xi_i \\ \xi_i \geqslant 0 \end{cases} \iff \quad \xi_i \geqslant \max\left(0, 1 - y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b)\right)$$

- Smaller objective function if $\xi_i = \max\left(0, 1 - y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b)\right)$
- If we plug the ξ_i's in the objective function we have

$$\operatorname*{arg\,min}_{(\boldsymbol{w},b)} \frac{1}{2} \|\boldsymbol{w}\|_2^2 + C \sum_{i=1}^{n} \max\left(0, 1 - y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b)\right)$$

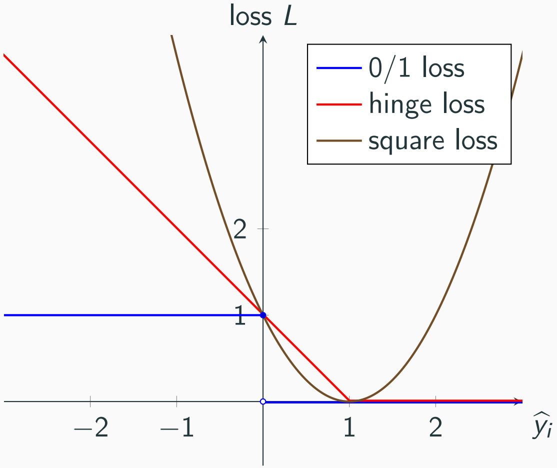
- Defining hinge loss function as: $L_{\text{hinge}}(x, y) = \max\left(0, 1 - xy\right)$

$$\operatorname*{arg\,min}_{(\boldsymbol{w},b)} \sum_{i=1}^{n} L_{\text{hinge}}(y_i, \boldsymbol{w} \cdot \boldsymbol{x}_i + b) + \frac{1}{2C} \|\boldsymbol{w}\|_2^2$$

Other losses

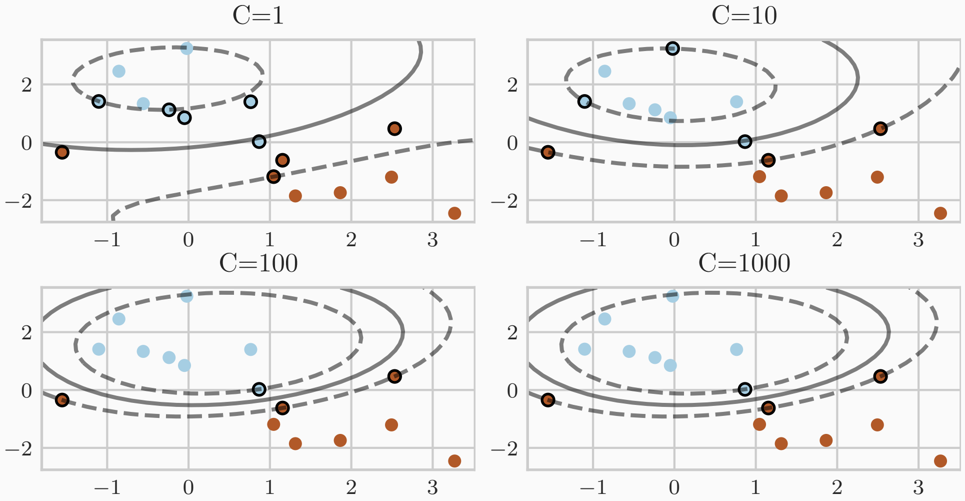
Losses when true class is $y_i = 1$

- 0/1 loss: penalises wrong sign of prediction $\widehat{y}_i$
- Square loss penalises the gap whatever the direction
- hinge loss: relaxing 0/1 loss

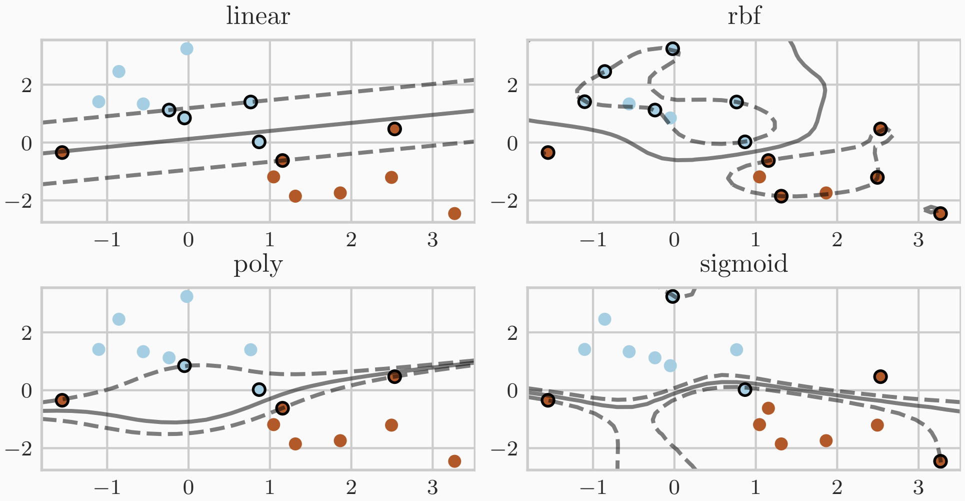


Illustrations: increasing the tuning parameter C

Gaussian kernel with varying C



Illustrations: changing the kernel



Kernel PCA

The idea is to apply the principal component analysis **on the feature space**

- New points are $\boldsymbol{y}_i = \Phi(\boldsymbol{x}_i)$
- Design matrix changes from

$$X = \begin{bmatrix} \boldsymbol{x}_1^T \\ \vdots \\ \boldsymbol{x}_n^T \end{bmatrix} \quad \text{to} \quad X_\Phi = \begin{bmatrix} \boldsymbol{y}_1^T \\ \vdots \\ \boldsymbol{y}_n^T \end{bmatrix}$$

- Suppose for now that X_Φ is **centered**
- The kernel matrix K gathers the inner products in feature space: $K = X_\Phi X_\Phi^T$

Kernel PCA

Kernel PCA on feature space

- Let $V_\Phi = \frac{1}{n} X_\Phi^T X_\Phi$ the sample variance–covariance matrix, $\mathrm{v}_1, \ldots, \mathrm{v}_q$ the eigen vectors and $\lambda_1 \geqslant \ldots \geqslant \lambda_q$ the corresponding eigenvalues
- We have seen that the i-th principal component is $X_\Phi \mathrm{v}_i$

We don't want to compute $X_\Phi \mathrm{v}_i$

- It is easy to see that $K(X_\Phi \mathrm{v}_i) = n\lambda_i (X_\Phi \mathrm{v}_i)$; PC are just eigenvectors (properly rescaled) of the kernel matrix K

Centering the kernel

We supposed that X_Φ was centered but what if it's not?

- Define the centering operator

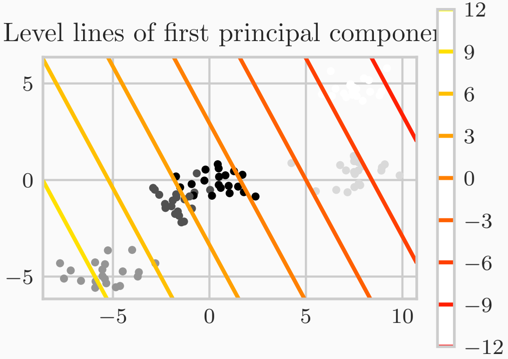
$$Q_n = I_n - \frac{1}{n}\mathbb{1}_n = \begin{pmatrix} 1-\frac{1}{n} & -\frac{1}{n} & \dots & -\frac{1}{n} \\ -\frac{1}{n} & 1-\frac{1}{n} & \ddots & \vdots \\ \vdots & \ddots & \ddots & -\frac{1}{n} \\ -\frac{1}{n} & \dots & -\frac{1}{n} & 1-\frac{1}{n} \end{pmatrix}.$$

  so that $X_\Phi^0 = Q_n X_\Phi$ is the centering of X_Φ
- The corresponding kernel is $K^0 = Q_n X_\Phi (Q_n X_\Phi)^T = Q_n X_\Phi X_\Phi^T Q_n = Q_n K Q_n$
- No need to compute X_Φ, just center column-wise and row-wise the kernel

Examples

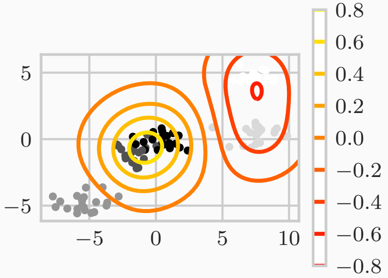
Classic PCA (linear kernel)

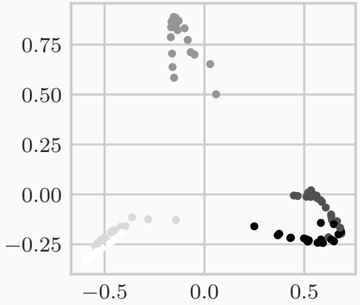
- Level lines of first principal component are straight lines
- And they are orthogonal to first principal direction



Examples

Kernel PCA with an RBF kernel (gaussian kernel)





Kernel ridge regression

- Matrix inversion trick: from $X\left(I + X^T X\right) = \left(I + XX^T\right) X$ we get

$$\left(I + XX^T\right)^{-1} X = X \left(I + X^T X\right)^{-1}$$

  where XX^T is $n \times n$ and $X^T X$ is $p \times p$

- Using this in $\widehat{\boldsymbol{y}}^{\text{ridge}} = X\left(X^T X + \lambda I_p\right)^{-1} X^T \boldsymbol{y}$ we get

$$\widehat{\boldsymbol{y}}^{\text{ridge}} = \left(\lambda I_p + XX^T\right)^{-1} XX^T \boldsymbol{y}$$

  and then $\widehat{\boldsymbol{y}}^{\text{ridge}} = (\lambda I_p + K)^{-1} K \boldsymbol{y}$

Kernel k-means / kernel k-nearest neighbors

Those algorithms only depend on distances between samples

- Since interdistance can be expressed as inner products

$$\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2 = \langle \boldsymbol{x}_i, \boldsymbol{x}_i \rangle - 2 \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle + \langle \boldsymbol{x}_j, \boldsymbol{x}_j \rangle$$

- K-means is kernelizable. The "kernel distance" is then

$$d(\boldsymbol{x}_i, \boldsymbol{x}_j)^2 = k(\boldsymbol{x}_i, \boldsymbol{x}_i) - 2k(\boldsymbol{x}_i, \boldsymbol{x}_j) + k(\boldsymbol{x}_j, \boldsymbol{x}_j)$$