

# MSc Big Data for Business - *MAP 534*

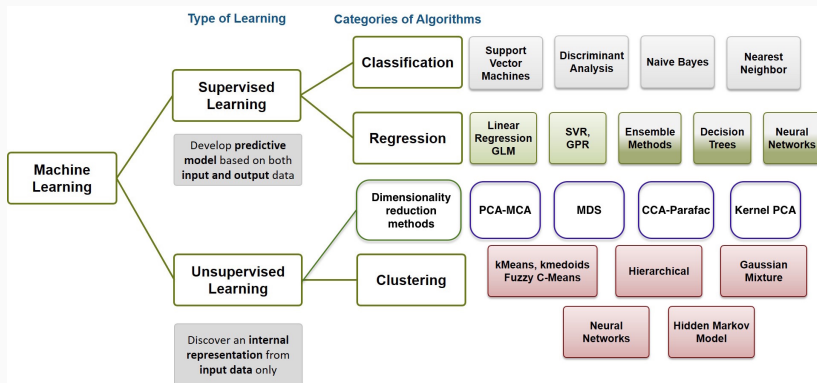
## Introduction to machine learning

---

Supervised classification

*Linear/Quadratic discriminant analysis (LDA/QDA) & Support Vector Machines (SVM)*

# Machine Learning



Introduction to supervised learning

Bayes and Plug-in classifiers

Naive Bayes

Discriminant analysis (linear and quadratic)

Support Vector Machine

## Supervised Learning Framework

- **Input** measurement  $\mathbf{X} \in \mathcal{X}$  (often  $\mathcal{X} \subset \mathbb{R}^d$ ), **output** measurement  $Y \in \mathcal{Y}$ .
- The joint distribution of  $(\mathbf{X}, Y)$  is **unknown**.
- $Y \in \{-1, 1\}$  (classification) or  $Y \in \mathbb{R}^m$  (regression).
- A **predictor** is a measurable function in  $\mathcal{F} = \{f : \mathcal{X} \rightarrow \mathcal{Y}\}$ .

## Training data

- $\mathcal{D}_n = \{(\mathbf{X}_1, Y_1), \dots, (\mathbf{X}_n, Y_n)\}$  i.i.d. with the same distribution as  $(\mathbf{X}, Y)$ .

## Goal

- Construct a **good** predictor  $\hat{f}_n$  from the training data.
- Need to specify the meaning of good.

## Loss function

- $\ell(Y, f(\mathbf{X}))$  measures the goodness of the prediction of  $Y$  by  $f(\mathbf{X})$ .
- **Prediction** loss:  $\ell(Y, f(\mathbf{X})) = \mathbf{1}_{Y \neq f(\mathbf{X})}$ .
- **Quadratic** loss:  $\ell(Y, \mathbf{X}) = \|Y - f(\mathbf{X})\|^2$ .

## Risk function

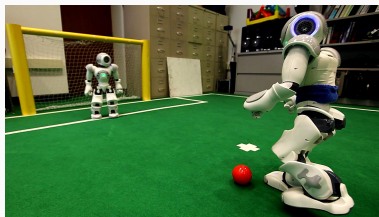
- Risk measured as the average loss:

$$\mathcal{R}(f) = \mathbb{E} [\ell(Y, f(\mathbf{X}))] .$$

- **Prediction** loss:  $\mathbb{E} [\ell(Y, f(\mathbf{X}))] = \mathbb{P} (Y \neq f(\mathbf{X}))$ .
- **Quadratic** loss:  $\mathbb{E} [\ell(Y, f(\mathbf{X}))] = \mathbb{E} [\|Y - f(\mathbf{X})\|^2]$ .
- **Beware:** As  $\hat{f}_n$  depends on  $\mathcal{D}_n$ ,  $\mathcal{R}(\hat{f}_n)$  is a random variable!

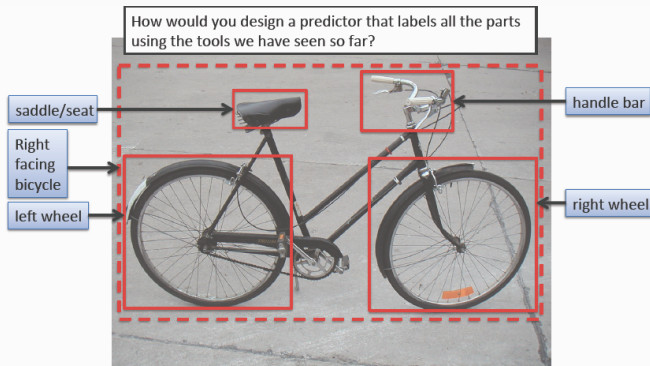
# A robot that learns

A robot endowed with a set of sensors and an online learning algorithm.



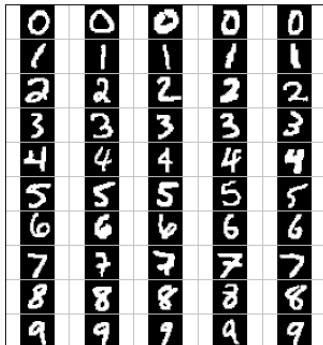
- **Task:** play football.
- **Performance:** score.
- **Experience:** current environment and outcome, past games...

# Object recognition in an image



- **Task**: say if an object is present or not in the image.
- **Performance**: number of errors.
- **Experience**: set of previously seen labeled image.

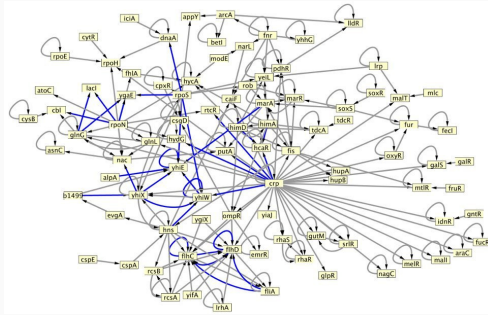
# Number



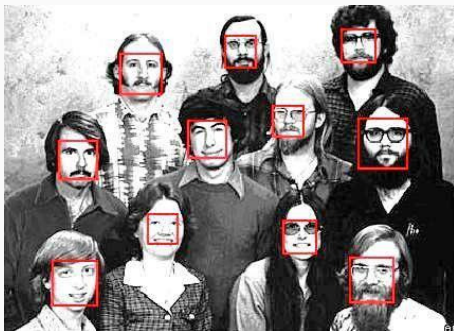
- **Task:** Read a ZIP code from an envelop.
- **Performance:** give a number from an image.
- **Prediction problem** with **X**: image and **Y**: corresponding number.



## Applications in biology



- **Task**: protein interaction network prediction.
- **Goal**: predict (unknown) interactions between proteins.
- **Prediction problem** with **X**: pair of proteins and **Y**: existence or no of interaction.



- **Goal**: detect the position of faces in an image.
- **X**: mask in the image and **Y**: presence or no of a face...

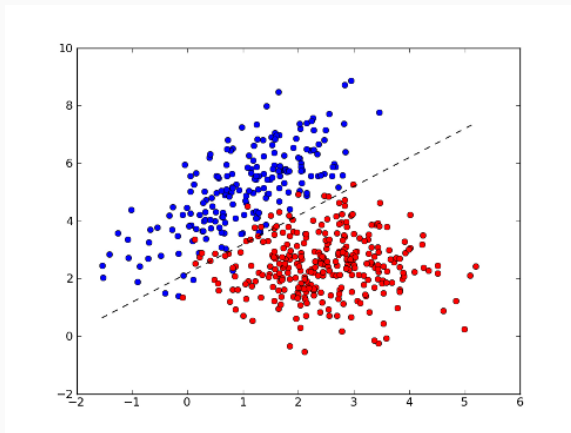
## Setting

- Historical data about **individuals**  $i = 1, \dots, n$ .
- **Features** vector  $\mathbf{X}_i \in \mathbb{R}^d$  for each individual  $i$ .
- For each  $i$ , the individual **belongs to a group** ( $Y_i = 1$ ) or not ( $Y_i = -1$ ).
- $Y_i \in \{-1, 1\}$  is the **label** of  $i$ .

## Aim

- Given a new  $\mathbf{X}$  (with no corresponding label), **predict a label in  $\{-1, 1\}$** .
- Use data  $\mathcal{D}_n = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$  **to construct a classifier**.

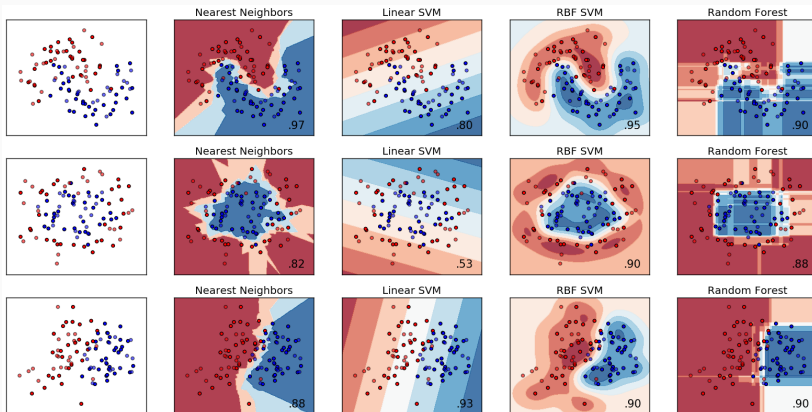
## Geometrically



Learn a **boundary** to separate two “groups” of points.

# Classification

...many ways to separate points!



Support Vector Machine

Linear Discriminant Analysis

Logistic Regression

Trees/ Random Forests

Kernel methods

Neural Networks

Many more...

Introduction to supervised learning

Bayes and Plug-in classifiers

Naive Bayes

Discriminant analysis (linear and quadratic)

Support Vector Machine

The best solution  $f^*$  (which is independent of  $\mathcal{D}_n$ ) is

$$f^* = \arg \min_{f \in \mathcal{F}} R(f) = \arg \min_{f \in \mathcal{F}} \mathbb{E} [\ell(Y, f(\mathbf{X}))] .$$

## Bayes Predictor (explicit solution)

→ Binary classification with 0 – 1 loss:

$$f^*(\mathbf{X}) = \begin{cases} +1 & \text{if } \mathbb{P}(Y = 1|\mathbf{X}) \geq \mathbb{P}(Y = -1|\mathbf{X}) \\ & \Leftrightarrow \mathbb{P}(Y = 1|\mathbf{X}) \geq 1/2, \\ -1 & \text{otherwise.} \end{cases}$$

→ Regression with the quadratic loss

$$f^*(\mathbf{X}) = \mathbb{E}[Y|\mathbf{X}] .$$

The explicit solution requires to know  $\mathbb{E}[Y|\mathbf{X}]$ ...



## Plugin Classifier

- In many cases, the conditional law of  $Y$  given  $\mathbf{X}$  is not known... or relies on parameters to be estimated.
- An empirical surrogate of the Bayes classifier is obtained from a possibly nonparametric estimator  $\hat{\eta}_n(\mathbf{X})$  of

$$\eta(\mathbf{x}) = \mathbb{P}(Y = 1|\mathbf{X})$$

using the training dataset.

- This surrogate is then plugged into the Bayes classifier.

### Plugin Bayes Classifier

- Binary classification with 0 – 1 loss:

$$\hat{f}_n(\mathbf{X}) = \begin{cases} +1 & \text{if } \hat{\eta}_n(\mathbf{X}) \geq 1/2, \\ -1 & \text{otherwise.} \end{cases}$$

**Input:** a data set  $\mathcal{D}_n$ .

Learn the distribution of  $Y$  given  $\mathbf{X}$  (using the data set) and plug this estimate in the Bayes classifier.

**Output:** a classifier  $\hat{f}_n : \mathbb{R}^d \rightarrow \{-1, 1\}$

$$\hat{f}_n(\mathbf{X}) = \begin{cases} +1 & \text{if } \hat{\eta}_n(\mathbf{X}) \geq 1/2, \\ -1 & \text{otherwise.} \end{cases}$$

→ Can we certify that the plug-in classifier is good ?

The missclassification error satisfies (see exercises):

$$0 \leq \mathbb{P}(\hat{f}_n(\mathbf{X}) \neq Y) - L^* \leq 2\mathbb{E} [|\eta(\mathbf{X}) - \hat{\eta}_n(\mathbf{X})|^2]^{1/2},$$

where

$$L^* = \mathbb{P}(f^*(\mathbf{X}) \neq Y)$$

and  $\hat{\eta}_n(\mathbf{x})$  is an empirical estimate based on the training dataset of

$$\eta(\mathbf{x}) = \mathbb{P}(Y = 1 | \mathbf{X} = \mathbf{x}).$$

# How to estimate the conditional law of $Y$ ?

## Fully parametric modeling.

Estimate the law of  $(\mathbf{X}, Y)$  and use the **Bayes formula** to deduce an estimate of the conditional law of  $Y$ : *LDA/QDA, Naive Bayes...*

## Parametric conditional modeling.

Estimate the conditional law of  $Y$  by a **parametric** law: *linear regression, logistic regression, Feed Forward Neural Networks...*

## Nonparametric conditional modeling.

Estimate the conditional law of  $Y$  by a **non parametric** estimate: *kernel methods, nearest neighbors...*

# Fully Generative Modeling

If the law of  $(\mathbf{X}, Y)$  is **known** everything can be easy!

## Bayes formula

With a slight abuse of notation, if the law of  $\mathbf{X}$  has a density  $g$  with respect to a reference measure,

$$\mathbb{P}(Y = k | \mathbf{X}) = \frac{g_k(\mathbf{X}) \mathbb{P}(Y=k)}{g(\mathbf{X})},$$

where  $g_k$  is the density of the distribution of  $\mathbf{X}$  given  $\{Y = k\}$ .

## Generative Modeling

Propose a model for  $(\mathbf{X}, Y)$ .

Plug the conditional law of  $Y$  given  $\mathbf{X}$  in the Bayes *classifier*.

**Remark:** require to model the joint law of  $(\mathbf{X}, Y)$  rather than only the conditional law of  $Y$ .

Great flexibility in the model design but may lead to complex computation.

Introduction to supervised learning

Bayes and Plug-in classifiers

**Naive Bayes**

Discriminant analysis (linear and quadratic)

Support Vector Machine

## Naive Bayes

Classical algorithm using a crude modeling for  $\mathbb{P}(\mathbf{X}|Y)$ :

→ **Feature independence** assumption:

$$\mathbb{P}(\mathbf{X}|Y) = \prod_{i=1}^d \mathbb{P}(X^{(i)}|Y) .$$

→ **Simple featurewise model**: binomial if binary, multinomial if finite and Gaussian if continuous.

If all features are continuous, the law of  $\mathbf{X}$  given  $Y$  is Gaussian with a **diagonal covariance matrix**!

Very simple learning even in **very high dimension**!

→ **Feature independence** assumption:

$$\mathbb{P}(\mathbf{X}|Y) = \prod_{j=1}^d \mathbb{P}(X^{(j)}|Y) .$$

For  $k \in \{-1, 1\}$ ,  $\mathbb{P}(Y = k) = \pi_k$  and the conditional density of  $X^{(j)}$  given  $\{Y = k\}$  is

$$g_k(\mathbf{x}^{(j)}) = (2\pi\sigma_{j,k}^2)^{-1/2} \exp \left\{ -(\mathbf{x}^{(j)} - \mu_{j,k})^2 / (2\sigma_{j,k}^2) \right\} .$$

The conditional distribution of  $\mathbf{X}$  given  $\{Y = k\}$  is then

$$g_k(\mathbf{x}) = (\det(2\pi\Sigma_k))^{-1/2} \exp \left\{ -(\mathbf{x} - \mu_k)^T \Sigma_k^{-1} (\mathbf{x} - \mu_k) / 2 \right\} ,$$

where  $\Sigma_k = \text{diag}(\sigma_{1,k}^2, \dots, \sigma_{d,k}^2)$  and  $\mu_k = (\mu_{1,k}, \dots, \mu_{d,k})^T$ .



In a two-classes problem, the optimal classifier is (see linear discriminant analysis below):

$$f^* : \mathbf{X} \mapsto 2\mathbb{1}\{\mathbb{P}(Y = 1|\mathbf{X}) > \mathbb{P}(Y = -1|\mathbf{X})\} - 1.$$

→ When the parameters are unknown, they may be replaced by their maximum likelihood estimates.

This yields, for  $k \in \{-1, 1\}$ ,

$$\hat{\pi}_k^n = \frac{1}{n} \sum_{i=1}^n \mathbb{1}_{Y_i=k},$$

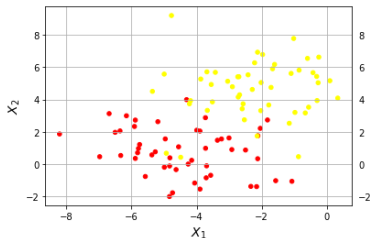
$$\hat{\mu}_k^n = \frac{1}{\sum_{i=1}^n \mathbb{1}_{Y_i=k}} \sum_{i=1}^n \mathbb{1}_{Y_i=k} X_i,$$

$$\hat{\Sigma}_k^n = \text{diag} \left( \frac{1}{n} \sum_{i=1}^n (X_i - \hat{\mu}_k^n)(X_i - \hat{\mu}_k^n)^T \mathbb{1}_{Y_i=k} \right).$$

# Gaussian Naive Bayes

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs

X, y = make_blobs(100, 2, centers = 2, cluster_std=1.5)
plt.scatter(X[:, 0], X[:, 1], c=y, s=20, cmap='autumn')
plt.ylabel(r"$X_2$", fontsize=14)
plt.xlabel(r"$X_1$", fontsize=14)
plt.tick_params(labelright=True)
plt.grid('True')
```



# Gaussian Naive Bayes

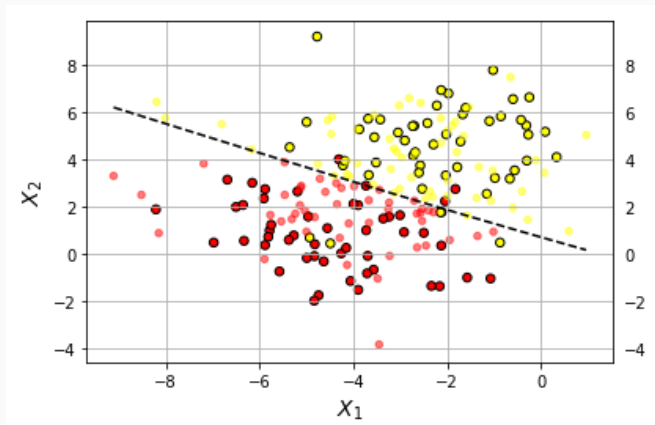
```
from sklearn.naive_bayes import GaussianNB
# Fit Naive Bayes to obtain the classifier
model = GaussianNB()
model.fit(X, y);

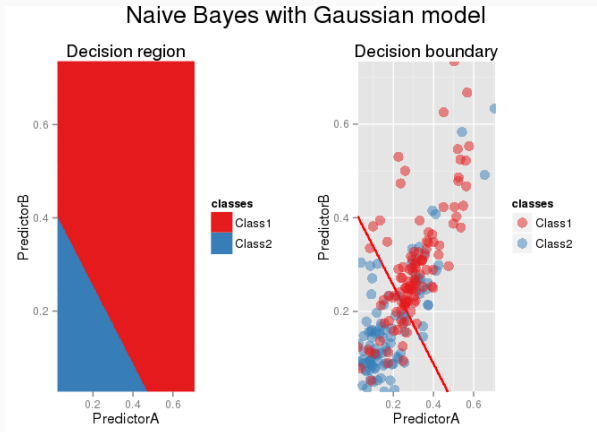
# Generate new data
Xnew = np.random.multivariate_normal([np.mean(X[:,0]),np.mean(X[:,1])], 4*np.eye(2), 100)
# Predict labels of new data from the classifier
ynew = model.predict(Xnew)

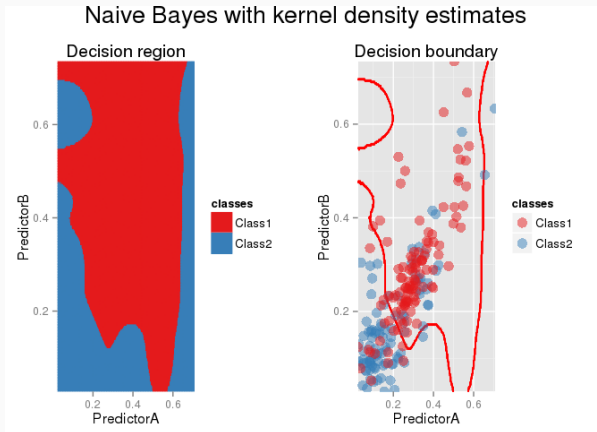
# Predict the classification probabilities on a grid
xgrid = (np.min(Xnew[:,0]), np.max(Xnew[:,0]))
ygrid = (np.min(Xnew[:,1]), np.max(Xnew[:,1]))
xx, yy = np.meshgrid(np.linspace(xgrid[0], xgrid[1], 100), np.linspace(ygrid[0], ygrid[1], 100))
Z = model.predict_proba(np.c_[xx.ravel(), yy.ravel()])
Z = Z[:, 1].reshape(xx.shape)

# Plot training data
plt.scatter(X[:, 0], X[:, 1], c = y, s = 30, edgecolors = 'black', cmap='autumn')
# Plot new data
plt.scatter(Xnew[:, 0], Xnew[:, 1], c = ynew, s = 20, cmap = 'autumn', alpha = 0.5)
# Plot classifier boundary
plt.contour(xx, yy, Z, [0.5], colors = 'k', linestyle = 'dashed')
plt.ylabel(r"$X_2$", fontsize=14)
plt.xlabel(r"$X_1$", fontsize=14)
plt.tick_params(labelright=True)
plt.grid(True)
```

## Gaussian Naive Bayes







Introduction to supervised learning

Bayes and Plug-in classifiers

Naive Bayes

Discriminant analysis (linear and quadratic)

Support Vector Machine

## Discriminant Analysis (Gaussian model)

The conditional densities are modeled as multivariate normal. For all class  $k$ , conditionnally on  $\{Y = k\}$ ,

$$\mathbf{X} \sim \mathcal{N}(\mu_k, \Sigma_k).$$

Discriminant functions:

$$g_k(\mathbf{X}) = \ln(\mathbb{P}\{\mathbf{X}|Y = k\}) + \ln(\mathbb{P}(Y = k)).$$

In a two-classes problem, the optimal classifier is (see exercises):

$$f^* : x \mapsto 2\mathbb{1}\{g_1(x) > g_{-1}(x)\} - 1.$$

QDA (different  $\Sigma_k$  in each class) and LDA ( $\Sigma_k = \Sigma$  for all  $k$ )

lightr: this model can be false but the methodology remains valid!



## Estimation

In practice,  $\mu_k$ ,  $\Sigma_k$  and  $\pi_k := \mathbb{P}(Y = k)$  have to be estimated.

→ **Estimated proportions**  $\hat{\pi}_k = \frac{n_k}{n} = \frac{1}{n} \sum_{i=1}^n \mathbf{1}_{\{Y_i=k\}}$ .

→ **Maximum likelihood estimate** of  $\hat{\mu}_k$  and  $\hat{\Sigma}_k$  (explicit formulas).

The DA classifier then becomes

$$\hat{f}_n(\mathbf{X}) = \begin{cases} +1 & \text{if } \hat{g}_1(\mathbf{X}) \geq \hat{g}_{-1}(\mathbf{X}), \\ -1 & \text{otherwise.} \end{cases}$$

If  $\Sigma_{-1} = \Sigma_1 = \Sigma$  then the **decision boundary is an affine hyperplane**.

The loglikelihood of the observations is given by

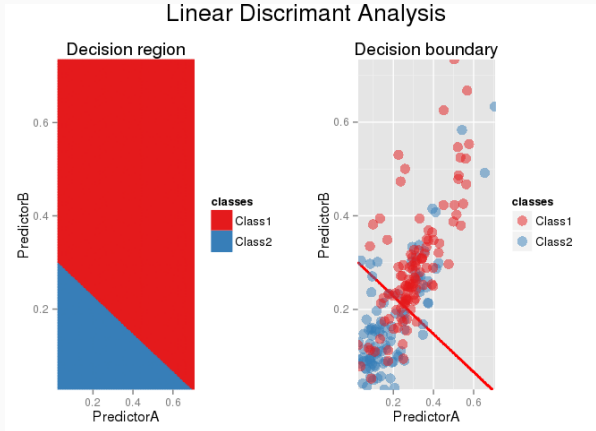
$$\begin{aligned}\log \mathbb{P}_{\theta}(X_{1:n}, Y_{1:n}) &= \sum_{i=1}^n \log \mathbb{P}_{\theta}(X_i, Y_i), \\ &= -\frac{nd}{2} \log(2\pi) - \frac{n}{2} \log \det(\Sigma) + \left( \sum_{i=1}^n \mathbb{1}_{Y_i=1} \right) \log \pi_1 + \left( \sum_{i=1}^n \mathbb{1}_{Y_i=-1} \right) \log(1 - \pi_1) \\ &\quad - \frac{1}{2} \sum_{i=1}^n \mathbb{1}_{Y_i=1} (X_i - \mu_1)^T \Sigma^{-1} (X_i - \mu_1) - \frac{1}{2} \sum_{i=1}^n \mathbb{1}_{Y_i=-1} (X_i - \mu_{-1})^T \Sigma^{-1} (X_i - \mu_{-1}).\end{aligned}$$

This yields, for  $k \in \{-1, 1\}$ ,

$$\begin{aligned}\hat{\pi}_k^n &= \frac{1}{n} \sum_{i=1}^n \mathbb{1}_{Y_i=k}, \\ \hat{\mu}_k^n &= \frac{1}{\sum_{i=1}^n \mathbb{1}_{Y_i=k}} \sum_{i=1}^n \mathbb{1}_{Y_i=k} X_i, \\ \hat{\Sigma}^n &= \frac{1}{n} \sum_{i=1}^n (X_i - \hat{\mu}_{Y_i}^n) (X_i - \hat{\mu}_{Y_i}^n)^T.\end{aligned}$$

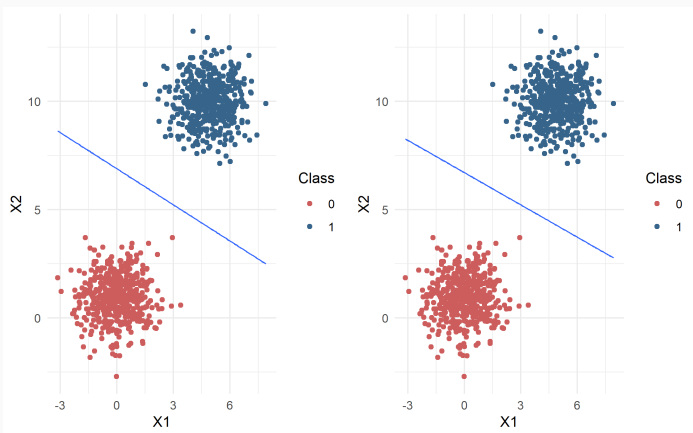
Remains to plug these estimates in the classification boundary.

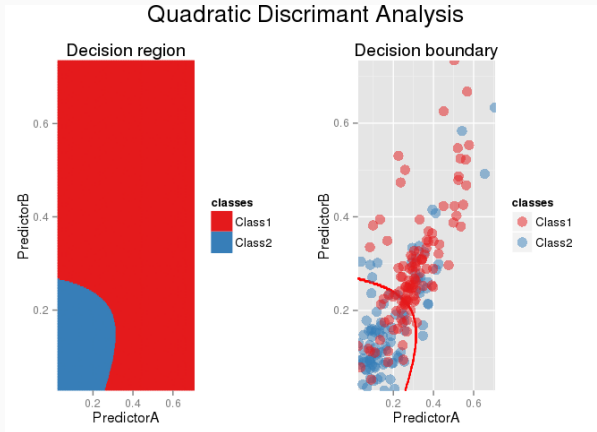
### Linear Discriminant Analysis



## Example: LDA

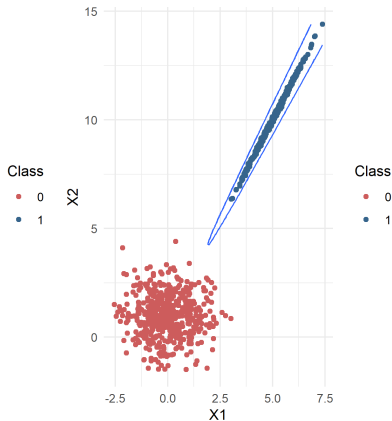
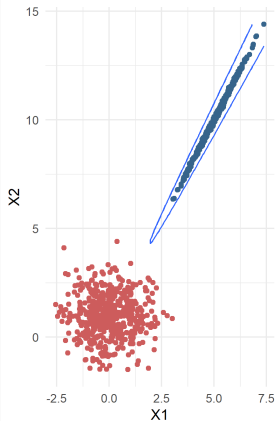
```
boundary_true_parameters = function(x, mu0, mu1, Sigma, pi0){  
  u = t(as.matrix(mu1-mu0)) %*% inv(Sigma)  
  v = (u %*% (matrix(x - ((mu1+mu0)/2)) )) - log(pi0/(1-pi0))  
  return(as.numeric(v))  
}
```





## Example: QDA

```
boundary_true_parameters_quadratic = function(x, mu0, mu1, Sigma0, Sigma1, pi0){  
  u1 = -0.5*(t(as.matrix(x-mu1))) %*% inv(Sigma1) %*% as.matrix((x - mu1))  
  u0 = 0.5*(t(as.matrix(x-mu0))) %*% inv(Sigma0) %*% as.matrix((x - mu0))  
  cste = - log(pi0/(1-pi0))  
  bonus = -0.5*log(abs(det(Sigma1))) + 0.5*log(abs(det(Sigma0)))  
  
  return(as.numeric(u1+u0+cste+bonus))  
}
```



Function `svm` in package `e1071`.

Function `lda` and `qda` in package `MASS`.

Function `naive_bayes` in package `naivebayes`.

Introduction to supervised learning

Bayes and Plug-in classifiers

Naive Bayes

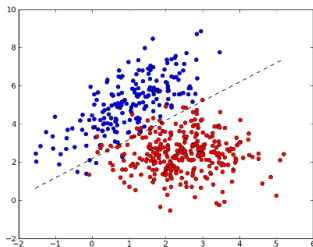
Discriminant analysis (linear and quadratic)

Support Vector Machine



# Linear classification

- Simple to interpret and to implement.
- On very large datasets ( $n \geq 10^6$ ), no other choice (training complexity).



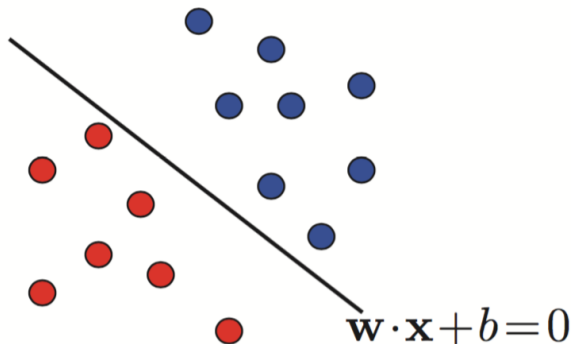
- Learn  $\hat{\mathbf{w}} \in \mathbb{R}^d$  and  $\hat{b}$  to build the classifier:

$$\hat{Y} = \text{sign}(\langle \mathbf{X}, \hat{\mathbf{w}} \rangle + \hat{b}).$$

## Linearly separable data

A dataset is **linearly separable** if there exists an hyperplane  $H$  (linear classification rule) such that the following assumptions hold.

- Points  $\mathbf{X}_i \in \mathbb{R}^d$  such that  $Y_i = 1$  are on one side of the hyperplane.
- Points  $\mathbf{X}_i \in \mathbb{R}^d$  such that  $Y_i = -1$  are on the other side.
- $H$  does not pass through any point  $\mathbf{X}_i$ .



A **hyperplane** is a translation of a set of vectors orthogonal to  $\mathbf{w}$ .

$$H_{\mathbf{w},b} = \{\mathbf{x} \in \mathbb{R}^d : \langle \mathbf{w}, \mathbf{x} \rangle + b = 0\}.$$

$\rightarrow \mathbf{w} \in \mathbb{R}^d$  is a **non-zero vector normal** to the hyperplane.  $\rightarrow b \in \mathbb{R}$  is a scalar.

Following for instance the results obtained for linear discriminant analysis and logistic regression, a **hyperplane  $H_{\mathbf{w},b}$  may be used as a classifier** by defining

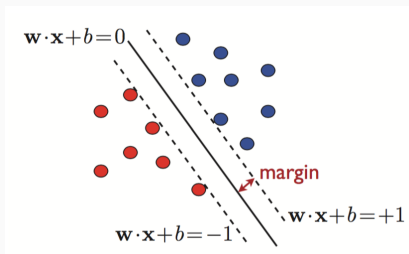
$$h_{\mathbf{w},b} : \mathbf{x} \mapsto \begin{cases} 1 & \text{if } \langle \mathbf{w}; \mathbf{x} \rangle + b > 0, \\ -1 & \text{otherwise.} \end{cases}$$

## Some geometry

Definition of  $H_{\mathbf{w},b}$  is invariant by multiplication of  $\mathbf{w}$  and  $b$  by a non-zero scalar.

If  $H_{\mathbf{w},b}$  does not pass through any sample point  $\mathbf{x}_i$ ,  $\mathbf{w}$  and  $b$  can be scaled so that

$$\min_{(\mathbf{x},y) \in \mathcal{D}_n} |\langle \mathbf{w}, \mathbf{x} \rangle + b| = 1.$$



For such  $\mathbf{w}$  and  $b$ , we call  $H$  the *canonical hyperplane*.

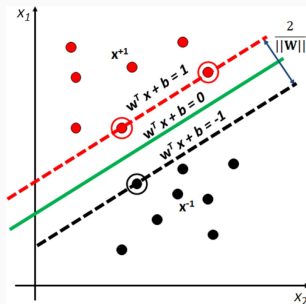
## Some geometry

The distance of any point  $\mathbf{x}' \in \mathbb{R}^d$  to  $H_{\mathbf{w},b}$  is

$$\frac{|\langle \mathbf{w}, \mathbf{x}' \rangle + b|}{\|\mathbf{w}\|}$$

If  $H_{\mathbf{w},b}$  is a canonical hyperplane, its margin is given by

$$\min_{(\mathbf{x},y) \in \mathcal{D}_n} \frac{|\langle \mathbf{w}, \mathbf{x} \rangle + b|}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|}.$$



## Linear separability and margin

If  $\mathcal{D}_n$  is **strictly linearly separable**, there exists a canonical separating hyperplane

$$H_{\mathbf{w},b} = \{\mathbf{x} \in \mathbb{R}^d : \langle \mathbf{w}, \mathbf{x} \rangle + b = 0\}$$

that satisfies

$$|\langle \mathbf{w}, \mathbf{X}_i \rangle + b| \geq 1 \text{ for any } i = 1, \dots, n.$$

An individual  $\mathbf{X}_i$  is **correctly classified** if

$$Y_i(\langle \mathbf{X}_i, \mathbf{w} \rangle + b) \geq 1.$$

The **margin of  $H_{\mathbf{w},b}$**  is equal to  $1/\|\mathbf{w}\|$ .

**Hard Support Vector Machines** is a classification procedure which aims at building a linear classifier with the largest possible margin, i.e. **the largest minimal distance between a point in the training set and the hyperplane**.

The hyperplane which **correctly separates all training data sets with the largest margin** is obtained with:

$$(\hat{w}_n, \hat{b}_n) \in \underset{\substack{(w,b) \in \mathbb{R}^d \times \mathbb{R}^d; \|w\|=1, \\ \forall i \in \{1, \dots, n\}, Y_i(\langle w; \mathbf{X}_i \rangle + b) > 0}}{\operatorname{argmax}} \left\{ \min_{1 \leq i \leq n} |\langle w; \mathbf{X}_i \rangle + b| \right\}.$$

The **hard Support Vector Machines** procedure is equivalent to solving the following optimization problem:

$$(\hat{w}_n, \hat{b}_n) \in \operatorname{argmax}_{(w,b) \in \mathbb{R}^d \times \mathbb{R}; \|w\|=1} \left\{ \min_{1 \leq i \leq n} Y_i (\langle w; X_i \rangle + b) \right\},$$

A **solution to the hard Support Vector Machines optimization** problem is obtained by setting  $(\hat{w}_n, \hat{b}_n) = (w_\star / \|w_\star\|, b_\star / \|w_\star\|)$  where

$$(w_\star, b_\star) \in \operatorname{argmin}_{\substack{(w,b) \in \mathbb{R}^d \times \mathbb{R} \\ \forall i \in \{1, \dots, n\}, Y_i(\langle w; X_i \rangle + b) \geq 1}} \|w\|^2.$$



A way of classifying  $\mathcal{D}_n$  with maximum margin is to solve the following problem:

$$\begin{aligned} (w_*, b_*) \in \operatorname{argmin}_{\substack{(w,b) \in \mathbb{R}^d \times \mathbb{R} \\ \forall i \in \{1, \dots, n\}, Y_i(\langle w; X_i \rangle + b) \geq 1}} \|w\|^2. \end{aligned}$$

- This problem **admits a unique solution**.
- It is a **quadratic programming** problem, which is easy to solve numerically.
- Dedicated optimization algorithms can **solve this on a large scale very efficiently**.

## Linear SVM: separable case

The optimization problem is solved using **Karush-Kuhn-Tucker's theorem**.

There are  $\alpha_i \geq 0$ ,  $i = 1, \dots, n$ , called **dual variables**, such that the solution  $(\mathbf{w}, b)$  of this problem satisfies:

$$\mathbf{w} = \sum_{i=1}^n \alpha_i Y_i \mathbf{X}_i \quad \text{and} \quad \alpha_i ((Y_i \langle \mathbf{w}, \mathbf{X}_i \rangle + b) - 1) = 0 \quad \text{for } i = 1, \dots, n.$$

→  $\alpha_i \neq 0$  if and only if  $Y_i \langle \mathbf{w}, \mathbf{X}_i \rangle + b = 1$ , meaning that  $\mathbf{X}_i$  is on the **marginal hyperplane**.

→ Weights vector  $\mathbf{w}$  is a **linear combination of the vectors  $\mathbf{x}_i$**  that belong to a marginal hyperplane.

→ Such points  $\mathbf{x}_i$  are called **support vectors**.

```

X, y = make_blobs(n_samples = 200, centers = 2, random_state = 0, cluster_std = 0.50)
simulated_data = pd.DataFrame(columns = ["X1", "X2", "Label"])

simulated_data["x"]      = X[:,0]
simulated_data["y"]      = X[:,1]
simulated_data["Label"] = y

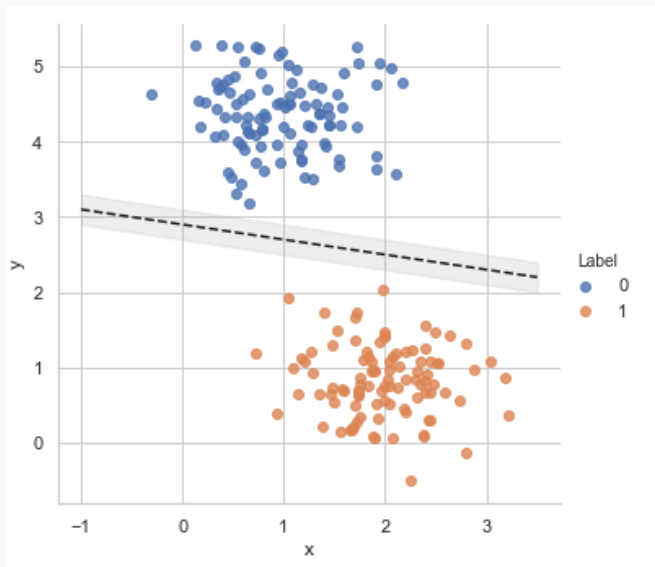
# Use the 'Label' argument to provide a factor variable
sns.set_style("whitegrid")
sns.lmplot(x = "x", y = "y", data = simulated_data, fit_reg = False, hue = 'Label', legend = True)

slope = 1.1
offset = 0.85
margin = 0.2

xfit = np.linspace(-1, 3.5)
yfit = m * xfit + b

plt.plot(xfit, yfit, '--k')
plt.fill_between(xfit, yfit - d, yfit + d, color = '#AAAAAA', alpha = 0.2)

```



```

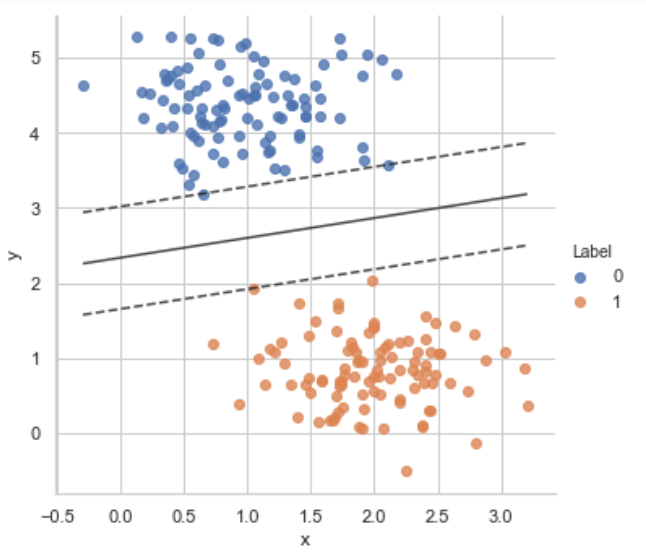
# Classification based on a support vector classifier
model = SVC(kernel='linear', C=10)
model.fit(X, y)
sns.set_style("whitegrid")
sns.lmplot(x = "x", y = "y", data = simulated_data, fit_reg = False, hue = 'Label', legend = True)

xlim = [np.min(X[:,0]), np.max(X[:,0])]
ylim = [np.min(X[:,1]), np.max(X[:,1])]
xplot = np.linspace(xlim[0], xlim[1], 30)
yplot = np.linspace(ylim[0], ylim[1], 30)

Yplot, Xplot = np.meshgrid(yplot, xplot)
xy          = np.vstack([Xplot.ravel(), Yplot.ravel()]).T
P           = model.decision_function(xy).reshape(Xplot.shape)

# plot decision boundary and margins
plt.contour(Xplot, Yplot, P, colors = 'k', levels = [-1, 0, 1], alpha = 0.8,
            linestyles = ['--', '-', '--'])

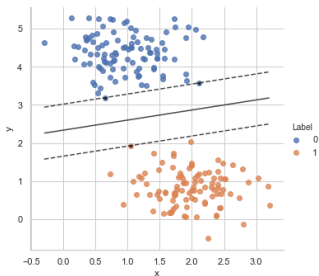
```



```

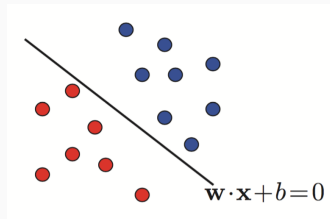
sns.set_style("whitegrid")
sns.lmplot(x = "x", y = "y", data = simulated_data, fit_reg = False, hue = 'Label', legend = True)
# plot decision boundary and margins
plt.contour(Xplot, Yplot, P, colors = 'k', levels = [-1, 0, 1], alpha = 0.8,
            linestyle = ['--', '-', '--'])
plt.scatter(model.support_vectors_[0], model.support_vectors_[1], s = 5, c = 'k');

```



## Linear SVM: non-separable case

Have you ever seen a dataset that looks that this?



→ Restricting the problem to linearly separable training data sets is a **somehow strong assumption**.

→ Inequality constraints in the quadratic optimization problem **can be relaxed**.

→ Introduction of nonnegative variables  $(\xi_i)_{1 \leq i \leq n}$  which **quantify the nonfeasibility of the constraint**  $Y_i(\langle \mathbf{w}; \mathbf{X}_i \rangle + b) \geq 1$ .

$$Y_i(\langle \mathbf{w}, \mathbf{X}_i \rangle + b) \geq 1 - \xi_i.$$



## Linear SVM: non-separable case

The original problem is then replaced by

$$(w_*, b_*, \xi_*) \in \underset{(w, b, \xi) \in \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}_+^n}{\operatorname{argmin}} \left\{ \lambda \|w\|^2 + \frac{1}{n} \sum_{i=1}^n \xi_i \right\},$$
$$\forall i \in \{1, \dots, n\}, Y_i(\langle w; X_i \rangle + b) \geq 1 - \xi_i$$

where  $\lambda > 0$ .

The **soft Support Vector Machines** algorithm minimizes simultaneously the margin of the linear classifier and the average value of these slack variables.

Note that, if  $(w_*, b_*)$  is solution to

$$(w_*, b_*) \in \underset{(w, b) \in \mathbb{R}^d \times \mathbb{R}}{\operatorname{argmin}} \left\{ \lambda \|w\|^2 + \frac{1}{n} \sum_{i=1}^n (1 - Y_i(\langle w; X_i \rangle + b))_+ \right\},$$

then  $(w_*/\|w_*\|, b_*, \xi_*/\|w_*\|)$  is solution to the soft SVM problem.

The **soft SVM problem boils down** to computing:

$$(w_*, b_*, \xi_*) \in \underset{\substack{(w, b, \xi) \in \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}_+^d \\ \forall i \in \{1, \dots, n\}, Y_i(\langle w; X_i \rangle + b) \geq 1 - \xi_i}}{\operatorname{argmin}} \left\{ \lambda \|w\|^2 + \frac{1}{n} \sum_{i=1}^n \xi_i \right\},$$

where  $\lambda > 0$ .

- This problem **admits a unique solution**.
- It is a **quadratic programming** problem, which is easy to solve numerically.
- Dedicated optimization algorithms can **solve this on a large scale very efficiently**.

# The hinge loss

This problem can be reformulated as follows

$$\operatorname{argmin}_{\mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}} \left\{ \frac{1}{2} \|\mathbf{w}\|_2^2 + C \sum_{i=1}^n \max \left( 0, 1 - y_i (\langle \mathbf{x}_i, \mathbf{w} \rangle + b) \right) \right\},$$

By introducing the **hinge loss**

$$\ell(y, y') = \max(0, 1 - yy') = (1 - yy')_+,$$

the problem can be written as

$$\operatorname{argmin}_{\mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}} \left\{ \frac{1}{2} \|\mathbf{w}\|_2^2 + C \sum_{i=1}^n \ell(y_i, \langle \mathbf{x}_i, \mathbf{w} \rangle + b) \right\}.$$

→ Specific loss functions  $\ell$  in a **general setting**.

The joint law of  $(X, Y)$  is **not assumed to belong to any parametric or semiparametric** family of models.

The classification risk **cannot be computed nor minimized**, it is instead estimated by the empirical classification risk defined as

$$\hat{L}_{\text{miss}}^n(f) = \frac{1}{n} \sum_{i=1}^n \mathbb{1}_{Y_i \neq f(\mathbf{x}_i)},$$

where  $(X_i, Y_i)_{1 \leq i \leq n}$  are independent with the same distribution as  $(X, Y)$ .

The classification problem then boils down to solving

$$\hat{f}^n \in \operatorname{argmin}_{f \in \mathcal{F}} \hat{L}_{\text{miss}}^n(f),$$

for a chosen class  $\mathcal{F}$  of classifiers.

Nonparametric classification based on the empirical risk minimization may seem appealing

It cannot be used to derive efficient practical classifiers due to **the computational cost of the optimization problem**.

The target loss function  $\hat{L}_{\text{miss}}^n$  is replaced by **a convex surrogate and its minimization is constrained to a convex set of classifiers**.

For any convex function  $\phi : \mathcal{X} \rightarrow \mathbb{R}$ , it is possible to build a classifier  $f$  given by  $f_\phi = \text{sign}(\phi)$ . The associated empirical classification is then

$$\hat{L}_{\text{miss}}^n(\phi) = \frac{1}{n} \sum_{i=1}^n \mathbb{1}_{Y_i \neq f_\phi(X_i)} = \frac{1}{n} \sum_{i=1}^n \mathbb{1}_{Y_i \phi(X_i) < 0}.$$

Replacing the indicator function by **any convex loss function  $\ell$  yields a convex surrogate**:

$$\hat{L}_{\text{miss}}^{n,\text{conv}}(\phi) = \frac{1}{n} \sum_{i=1}^n \ell(Y_i \phi(X_i)).$$

Penalizing the smoothness of the function  $\phi$  is penalized,  $\hat{L}_{\text{miss}}^{n,\text{conv}}$  may be replaced by

$$\hat{L}_{\text{miss}}^{n,\text{conv}}(\phi) = \frac{1}{n} \sum_{i=1}^n \ell(Y_i \phi(X_i)) + \lambda \|\phi\|^2,$$

where  $\lambda > 0$  and  $\|\cdot\|$  is a norm on the space  $\mathcal{H}$ .

**The soft Support Vector Machines** algorithm fits this framework with the affine base function  $\phi : \mathbf{x} \mapsto \langle \mathbf{w} ; \mathbf{x} \rangle + b$  and  $\ell$  chosen as the hinge loss  $\ell : x \mapsto (1 - x)_+$  when the target function is penalized by its margin  $\|\mathbf{w}\|^2$ .

A useful case in practice consists in choosing  $\mathcal{H}$  as a **Reproducing Kernel Hilbert Space** with positive definite kernel  $k$  on  $\mathcal{X} \times \mathcal{X}$ .

A function  $k$  on  $\mathcal{X} \times \mathcal{X}$  is said to be a **positive definite kernel** if and only if it is symmetric and if for all  $n \geq 1$ ,  $(x_1, \dots, x_n) \in \mathcal{X}^n$  and all  $(a_1, \dots, a_n) \in \mathbb{R}^n$ ,

$$\sum_{1 \leq i, j \leq n} a_i a_j k(x_i, x_j) \geq 0.$$

The **Reproducing Kernel Hilbert Space** with kernel  $k$  is the only Hilbert space  $\mathcal{H} \subset \mathbb{R}^{\mathcal{X}}$  such that for all  $x \in \mathcal{X}$ ,  $k(x, \cdot) \in \mathcal{H}$  and for all  $x \in \mathcal{X}$  and all  $f \in \mathcal{H}$ ,

$$f(x) = \langle f; k(x, \cdot) \rangle_{\mathcal{H}}.$$

$k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$  a positive definite kernel and  $\mathcal{H}$  the RKHS with kernel  $k$ .

$$\hat{\phi}_{\mathcal{H}}^n \in \operatorname{argmin}_{\phi \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^n \ell(Y_i \phi(X_i)) + \lambda \|\phi\|_{\mathcal{H}}^2,$$

where  $\|\phi\|_{\mathcal{H}}^2 = \langle \phi; \phi \rangle$ , is given by

$$\hat{\phi}_{\mathcal{H}}^n : x \mapsto \sum_{i=1}^n \hat{\alpha}_i k(X_i, x),$$

with

$$\hat{\alpha} \in \operatorname{argmin}_{\alpha \in \mathbb{R}^n} \left\{ \frac{1}{n} \sum_{i=1}^n \ell \left( \sum_{j=1}^n \alpha_j Y_j k(X_j, X_i) \right) + \lambda \sum_{1 \leq i, j \leq n} \alpha_i \alpha_j k(X_i, X_j) \right\}.$$