APPRENTISSAGE SUPERVISÉ (Le Plus Proche Voisin)

UN PROGRAMME QUI APPREND

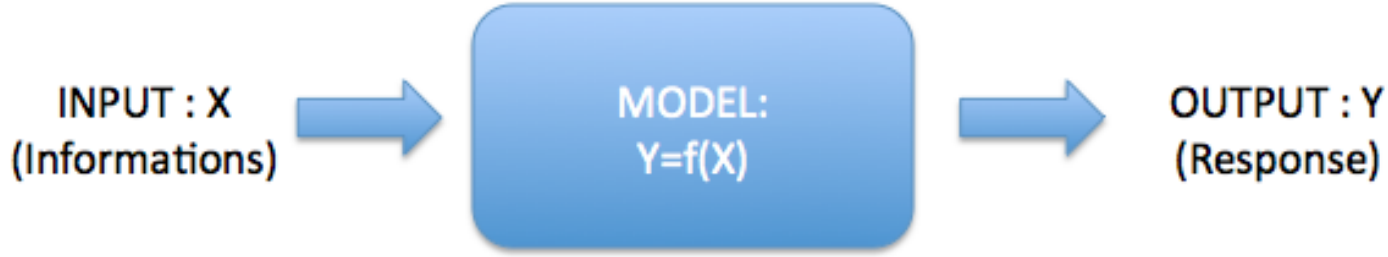
Objectif : induire la démarche de l'apprentissage par l'exemple.

- **Montrer** des exemples à votre programme en lui disant de quoi il s'agit.
- Lui faire **apprendre** une règle.
- Lui faire **appliquer** la règle à de nouveaux exemples.
- **Evaluer** si les prédictions sont bonnes en les comparant à la réalité.

Principes

ENTRÉES ET SORTIES

Le principe est toujours le même : X en entrée, Y en sortie. On cherche f tq Y = f (X).



	X	Y
Exemples :	emails	spam (oui on non)
	profil client	nombre de clics
	expression génique	état du patient
	profil électeur	vote
	âge, experience	salaire

TYPES DE PROBLÈMES

Il existe deux types de problèmes à résoudre:

Régression : la réponse est un nombre réel. Exemples : prédiction du salaire, nombre de clics, prix d'un appartement.

Classification : la réponse est une classe. Exemples : catégorie d'un article (classification multiple), spam (classification binaire).

EXEMPLE 1: CLASSIFICATION BINAIRE

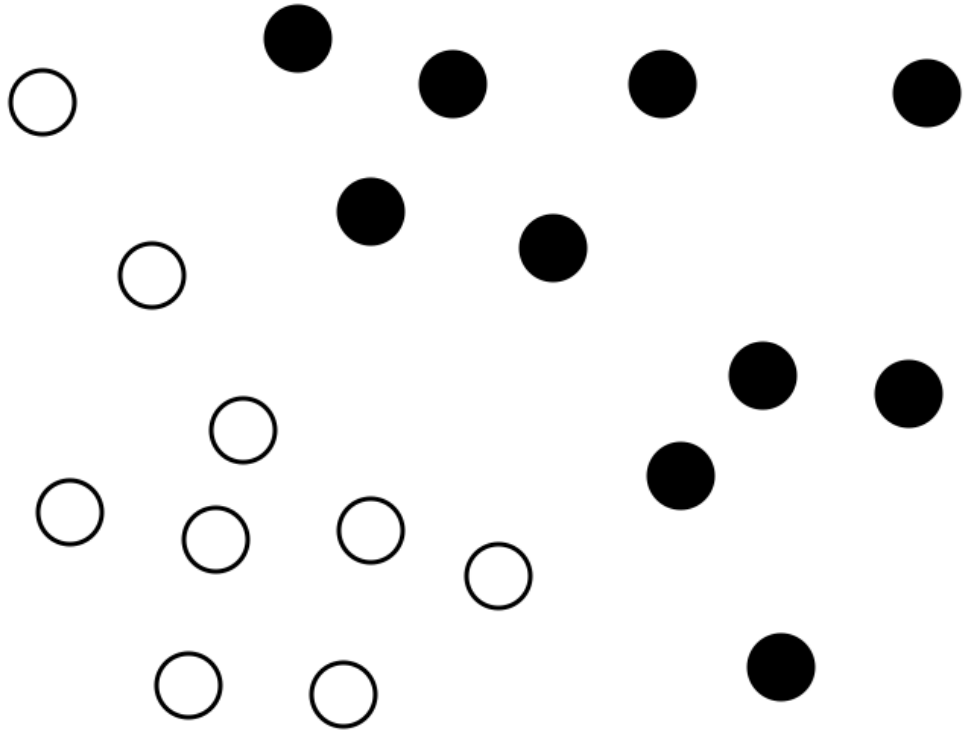


Image empruntée à Jean-Philippe Vert

EXEMPLE 1: CLASSIFICATION BINAIRE

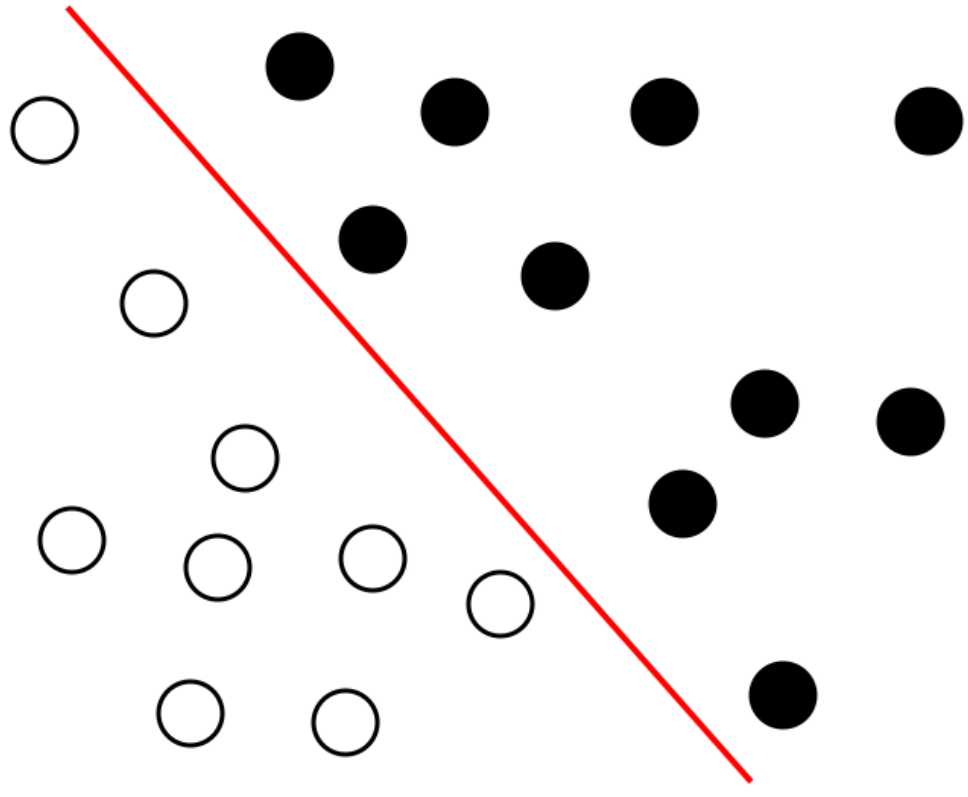


Image empruntée à Jean-Philippe Vert

EXEMPLE 1: CLASSIFICATION BINAIRE

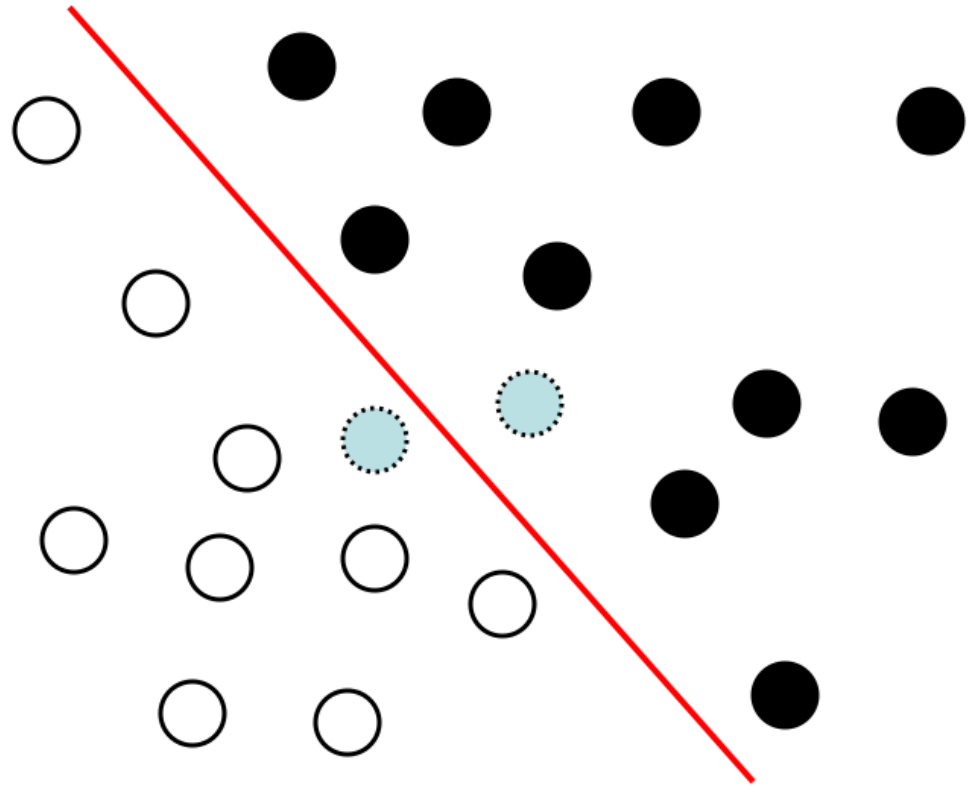


Image empruntée à Jean-Philippe Vert

EXEMPLE 1: CLASSIFICATION BINAIRE

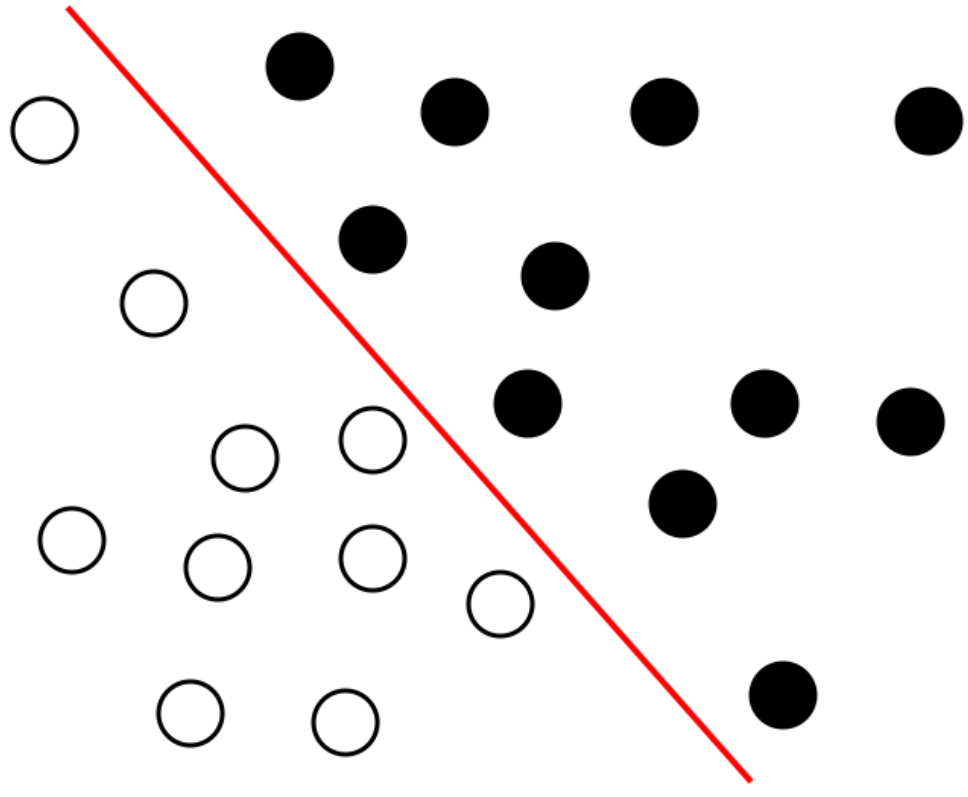
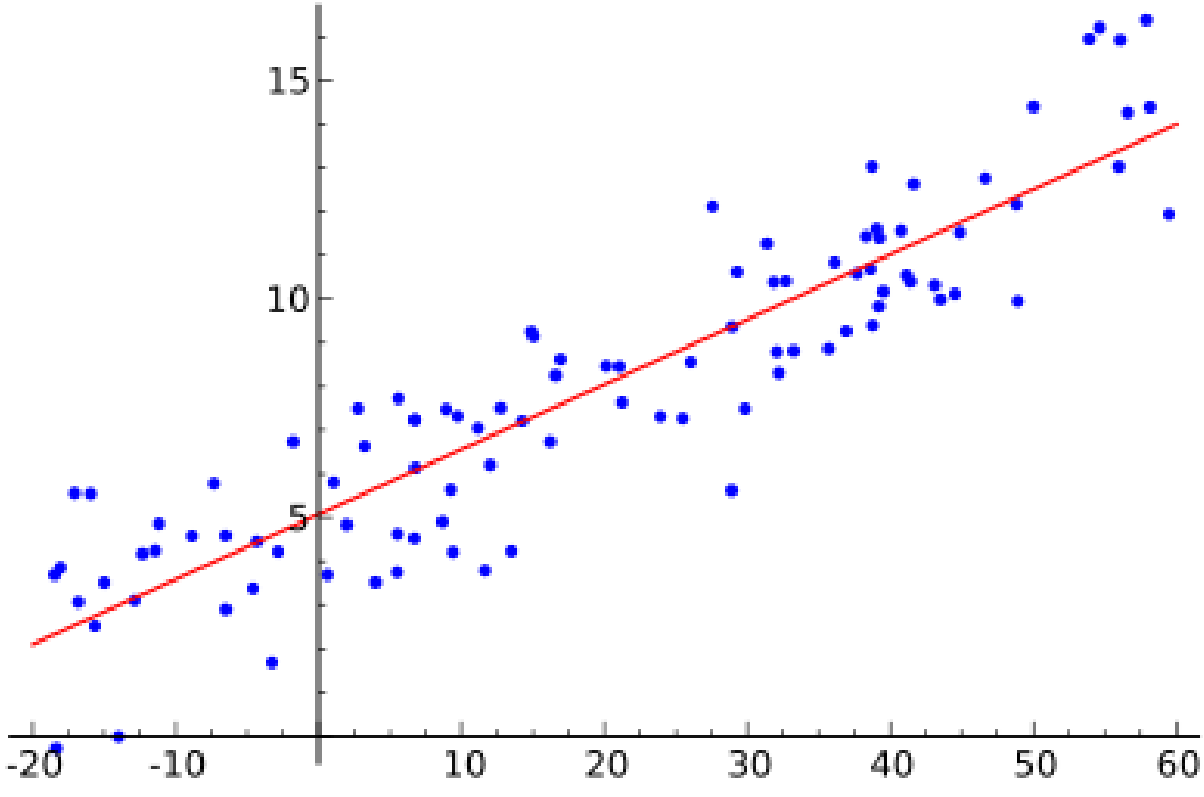


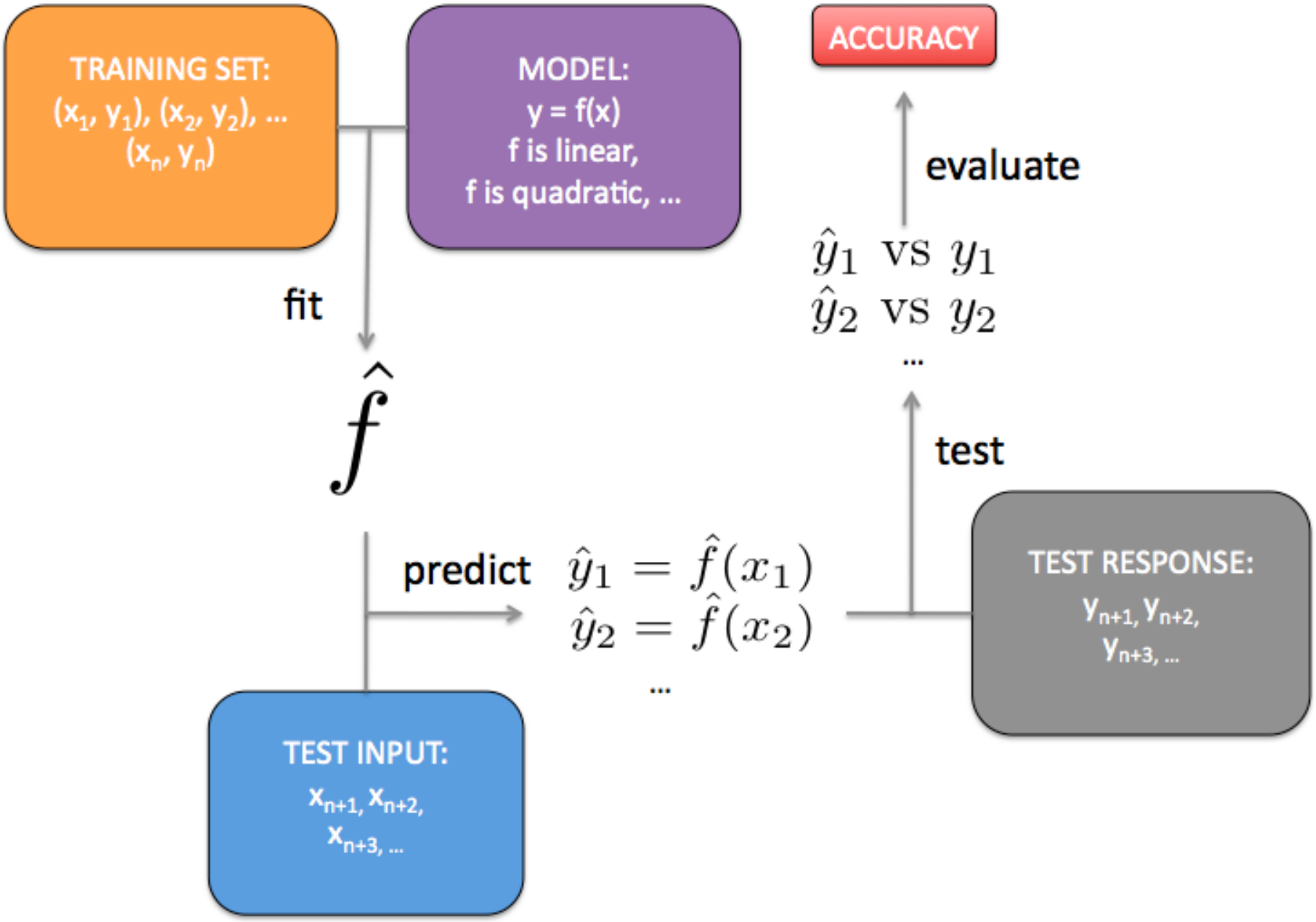
Image empruntée à Jean-Philippe Vert

EXEMPLE 2: RÉGRESSION LINÉAIRE

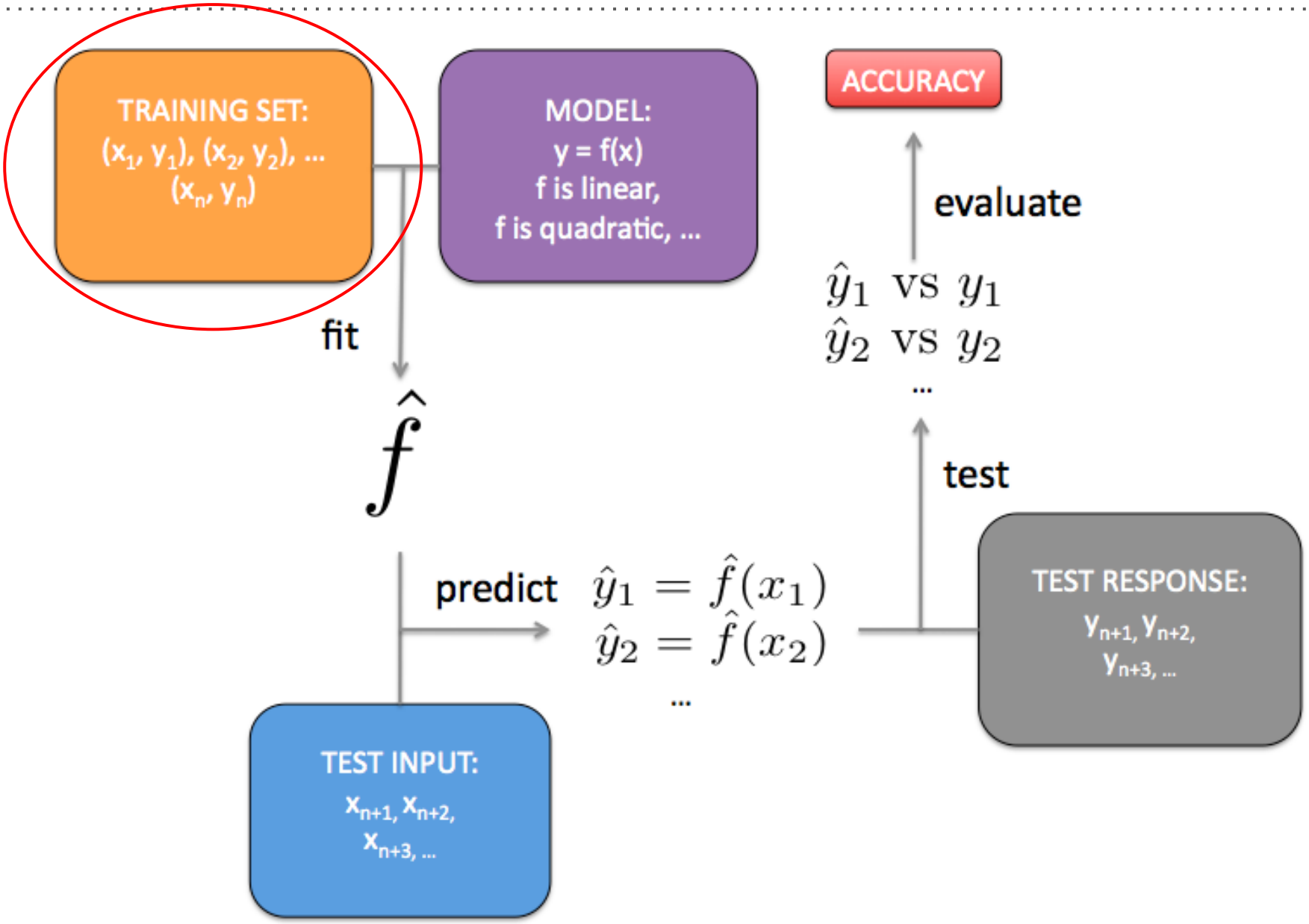


Source: wikipedia.com

PIPELINE


TRAINING SET:
$(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$
MODEL:
y = f(x)
f is linear,
f is quadratic, ...
ACCURACY
evaluate
$\hat{y}_1$ vs y_1
$\hat{y}_2$ vs y_2
...
fit
$\hat{f}$
test
predict
$\hat{y}_1 = \hat{f}(x_1)$
$\hat{y}_2 = \hat{f}(x_2)$
...
TEST RESPONSE:
$y_{n+1}, y_{n+2}, y_{n+3}, \ldots$
TEST INPUT:
$x_{n+1}, x_{n+2}, x_{n+3}, \ldots$

PIPELINE


TRAINING SET:
$(x_1, y_1), (x_2, y_2), \ldots$
(x_n, y_n)
MODEL:
y = f(x)
f is linear,
f is quadratic, ...
ACCURACY
evaluate
$\hat{y}_1$ vs y_1
$\hat{y}_2$ vs y_2
...
fit
$\hat{f}$
test
predict
$\hat{y}_1 = \hat{f}(x_1)$
$\hat{y}_2 = \hat{f}(x_2)$
...
TEST RESPONSE:
$y_{n+1}, y_{n+2},$
$y_{n+3}, \ldots$
TEST INPUT:
$x_{n+1}, x_{n+2},$
$x_{n+3}, \ldots$

ENSEMBLE D'APPRENTISSAGE

On entraîne le modèle sur **l'ensemble d'apprentissage** (training set).

Il est composé d'exemples de la forme (x_i, y_i) : pour chaque exemple i, on connaît donc la valeur d'entrée x_i et la réponse y_i.

Dans les cas que nous rencontrerons, x_i est souvent un **vecteur** et y_i est un **scalaire**.

On a n exemples: (x_1, y_1), (x_2, y_2), ..., (x_n, y_n).

	sexe	âge	diplôme		salaire
x_1	0	30	5	y_1	3000
x_2	1	25	2	y_2	1800
x_3	1	53	3	y_3	2900
x_n	0	20	0	y_n	1200

EXEMPLE

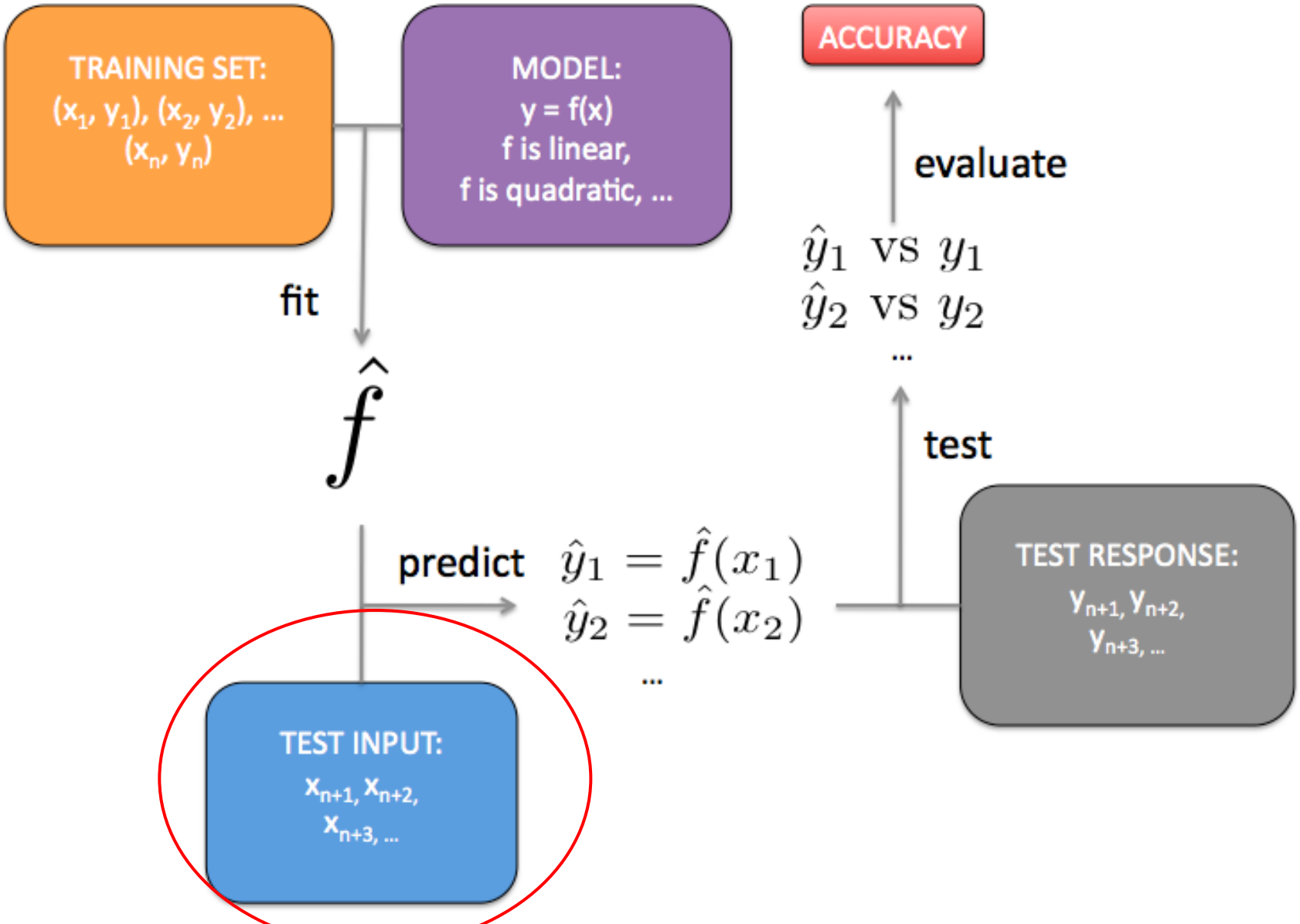
	voici	un	premier	texte	second	ce	document	contient
x_1	0.1	0.1	0.275	0	0	0	0	0
x_2	0.1	0.1	0	0	0.275	0	0	0
x_3	0	0	0	0	0	0.44	0.22	0.22

	spam
y_1	0
y_2	0
y_3	1

On veut apprendre à l'algorithme **ce qui fait que** les deux premiers messages ne sont pas des spams, le troisième oui, etc.

A la différence des premiers cours, on a ajouté l'information y**.**

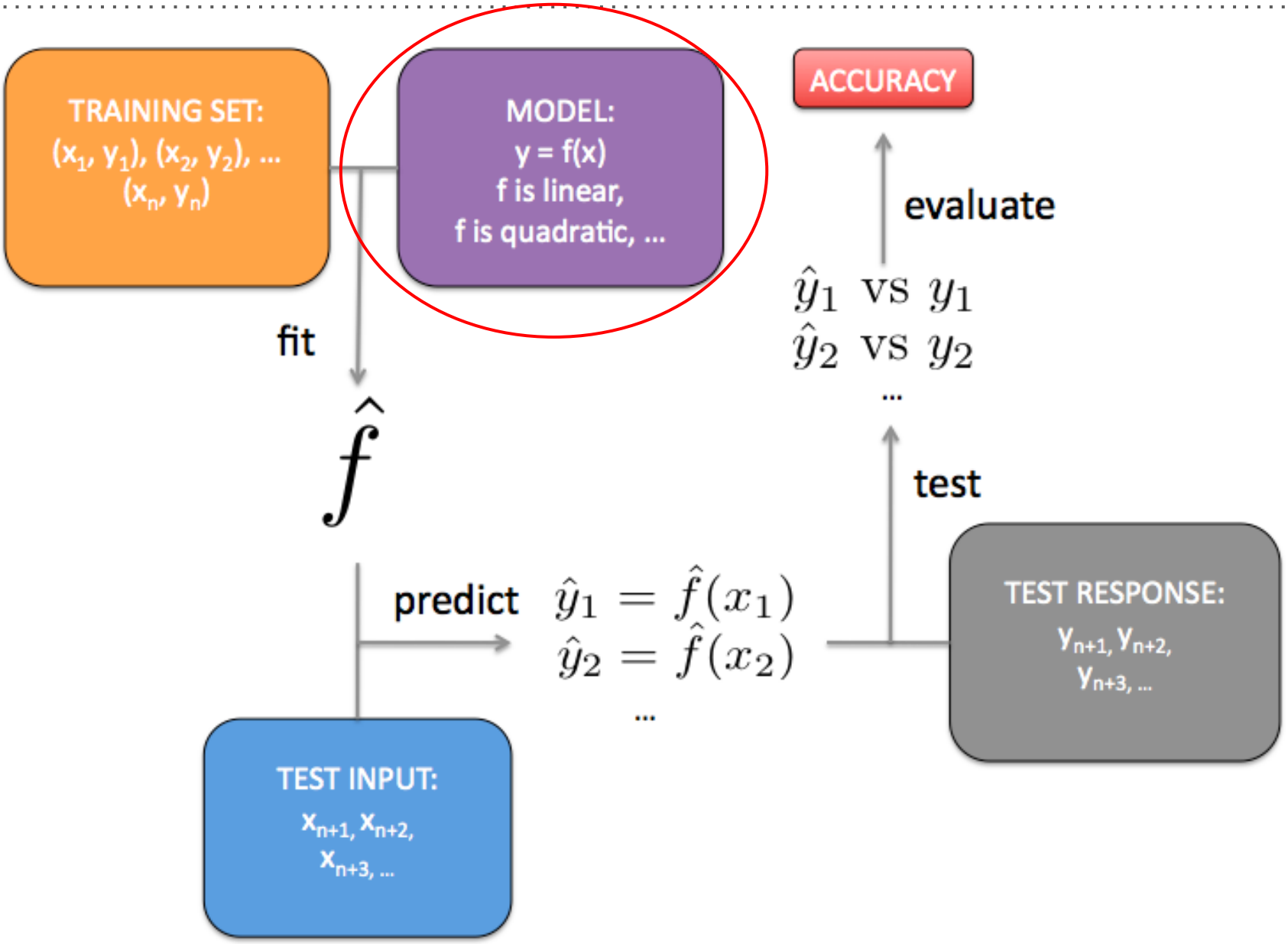
PIPELINE


TRAINING SET:
$(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$
MODEL:
y = f(x)
f is linear,
f is quadratic, ...
ACCURACY
evaluate
$\hat{y}_1$ vs y_1
$\hat{y}_2$ vs y_2
...
fit
$\hat{f}$
test
predict
$\hat{y}_1 = \hat{f}(x_1)$
$\hat{y}_2 = \hat{f}(x_2)$
...
TEST RESPONSE:
$y_{n+1}, y_{n+2}, y_{n+3}, \ldots$
TEST INPUT:
$x_{n+1}, x_{n+2}, x_{n+3}, \ldots$

ENSEMBLE DE TEST

Comme l'ensemble d'apprentissage, l'ensemble de test est composé d'exemples de la forme (x_i , y_i) : pour chaque exemple i, on connaît la valeur d'entrée x_i et la réponse y_i .
Ce sont des exemples que l'on a **mis de côté** au départ.

PIPELINE


TRAINING SET:
$(x_1, y_1), (x_2, y_2), \ldots$
(x_n, y_n)
MODEL:
y = f(x)
f is linear,
f is quadratic, ...
ACCURACY
evaluate
$\hat{y}_1$ vs y_1
$\hat{y}_2$ vs y_2
...
fit
$\hat{f}$
test
predict
$\hat{y}_1 = \hat{f}(x_1)$
$\hat{y}_2 = \hat{f}(x_2)$
...
TEST RESPONSE:
$y_{n+1}, y_{n+2},$
$y_{n+3}, \ldots$
TEST INPUT:
$x_{n+1}, x_{n+2},$
$x_{n+3}, \ldots$

LE MODÈLE

C’est ici que l’on fait des **hypothèses** sur la forme de f .

Par exemple:

f est **linéaire**:

$$y_i = \omega_1 x_{i,1} + \omega_2 x_{i,2} + \cdots + \omega_n x_{i,n}$$

Il s’agit de trouver les valeurs de ω_1, ω_2, ω_3...

LE MODÈLE

C’est ici que l’on fait des **hypothèses** sur la forme de f .

Par exemple:

- f est **linéaire**:

$$y_i = \omega_1 x_{i,1} + \omega_2 x_{i,2} + \cdots + \omega_n x_{i,n}$$

Il s’agit de trouver les valeurs de ω1, ω2, ω3...

- f est **quadratique**:

$$y_i = \omega_{1,1} x_{i,1}^2 + \omega_{2,2} x_{i,2}^2 + \omega_{3,3} x_{i,3}^2 + \omega_{1,2} x_{i,1} x_{i,2} + \omega_{1,3} x_{i,1} x_{i,3} + \omega_{2,3} x_{i,2} x_{i,3}$$

A nouveau, on cherche les valeurs des ω.

CONTRAINTES ?

De deux choses l'une :

Soit on a une **connaissance a priori** ou une **hypothèse pertinente** sur la forme de f : alors on peut **contraindre** f . C'est ce qu'on appelle un **data model**. Exemples : Régression Linéaire, Analyse Discriminante, Naive Bayes.

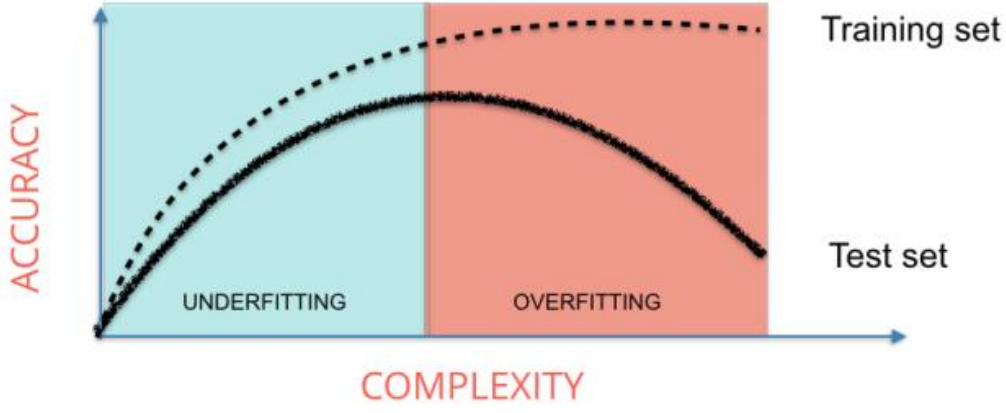
Soit on ne sait **rien** de f : on ne pose aucune contrainte. On parle alors de **modèle algorithmique**. Exemples : Plus Proches Voisins, Arbres de Décision, Support Vector Machines (SVM), Random Forests, Réseaux de Neurones.

DILEMME PERFORMANCE/COMPLEXITÉ

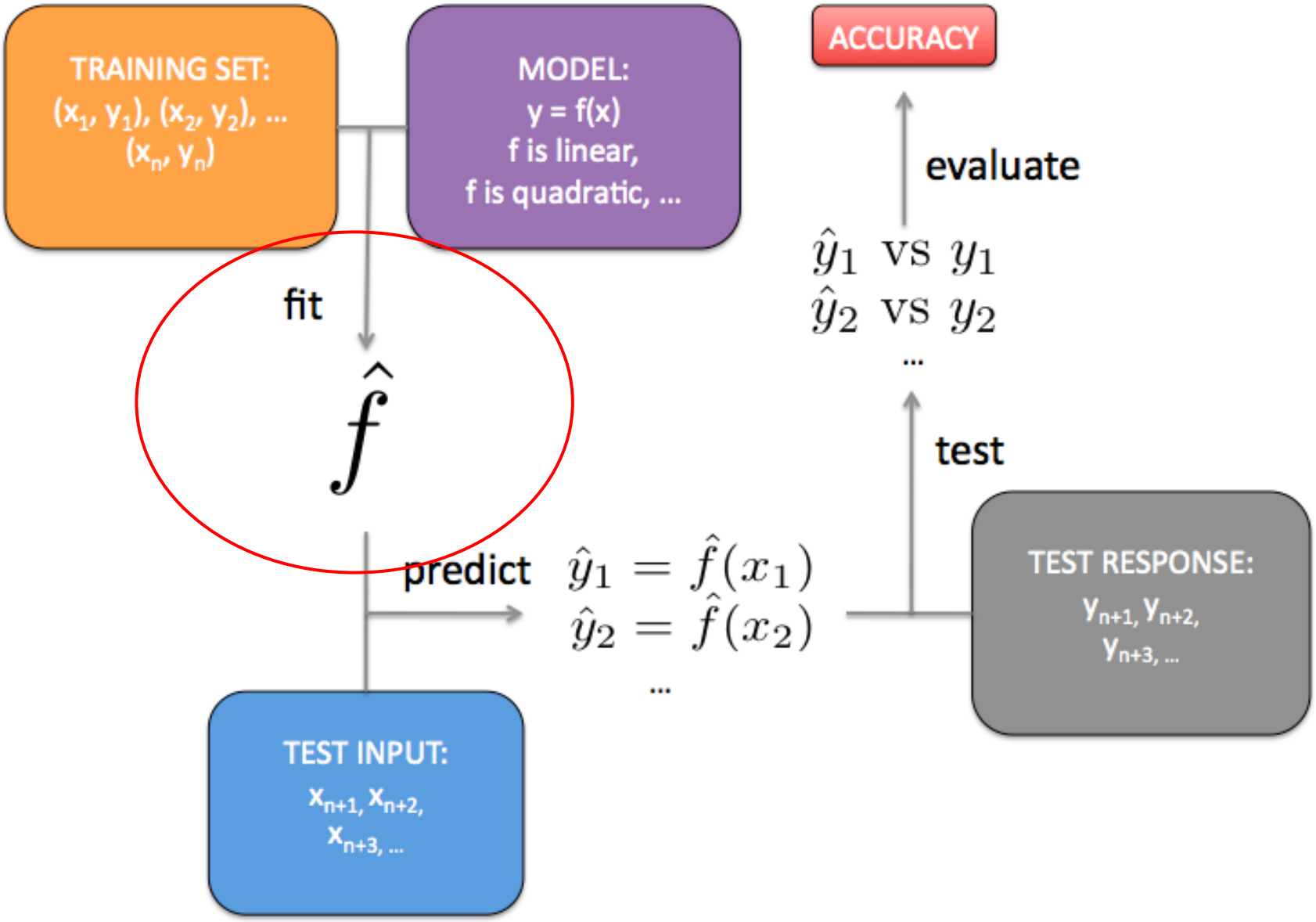
Plus le modèle est **simple**, plus il est facile de l'estimer mais moins il est proche de la réalité.

Plus le modèle est **complexe**, plus il s'approche de la réalité mais plus on risque de se tromper en l'estimant.

Dilemme complexité/performance : il faut trouver la complexité optimale.



PIPELINE

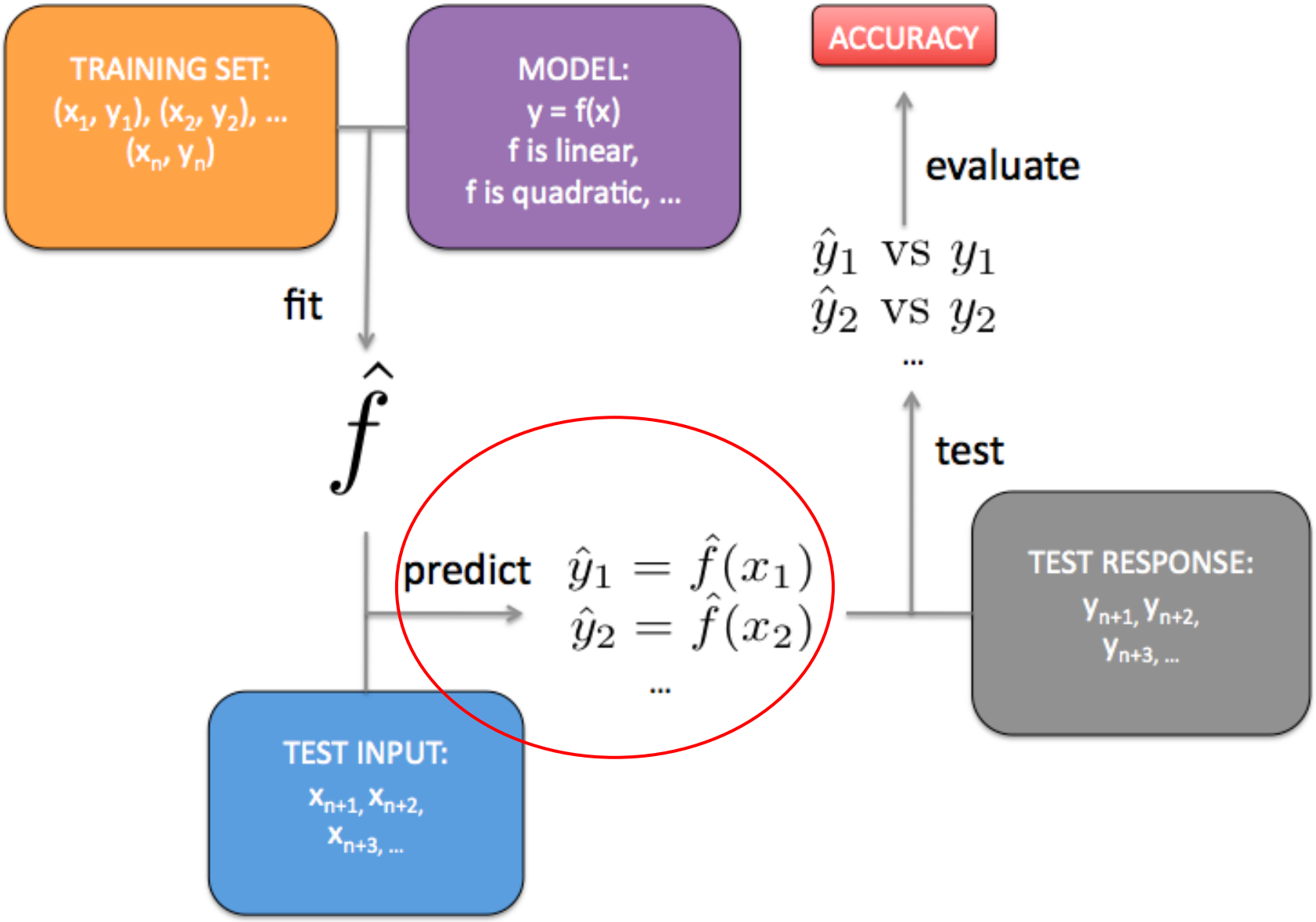

TRAINING SET:
$(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$
MODEL:
y = f(x)
f is linear,
f is quadratic, ...
ACCURACY
evaluate
$\hat{y}_1$ vs y_1
$\hat{y}_2$ vs y_2
...
fit
$\hat{f}$
test
predict
$\hat{y}_1 = \hat{f}(x_1)$
$\hat{y}_2 = \hat{f}(x_2)$
...
TEST RESPONSE:
$y_{n+1}, y_{n+2}, y_{n+3}, \ldots$
TEST INPUT:
$x_{n+1}, x_{n+2}, x_{n+3}, \ldots$

ESTIMATION

La phase **d'estimation** (fit) consiste, en fonction des hypothèses faites sur f, à **estimer la meilleure fonction** f dans le cadre des contraintes imposées.
La meilleure fonction est celle qui **généralise** le mieux et donne les meilleures prédictions. On appelle cette fonction : f^.

La manière de l'estimer dépend du modèle. Nous reviendrons là-dessus plus tard.

PIPELINE


TRAINING SET:
$(x_1, y_1), (x_2, y_2), \ldots$
(x_n, y_n)
MODEL:
y = f(x)
f is linear,
f is quadratic, ...
ACCURACY
evaluate
$\hat{y}_1$ vs y_1
$\hat{y}_2$ vs y_2
...
fit
$\hat{f}$
test
predict
$\hat{y}_1 = \hat{f}(x_1)$
$\hat{y}_2 = \hat{f}(x_2)$
...
TEST RESPONSE:
$y_{n+1}, y_{n+2},$
$y_{n+3}, \ldots$
TEST INPUT:
$x_{n+1}, x_{n+2},$
$x_{n+3}, \ldots$

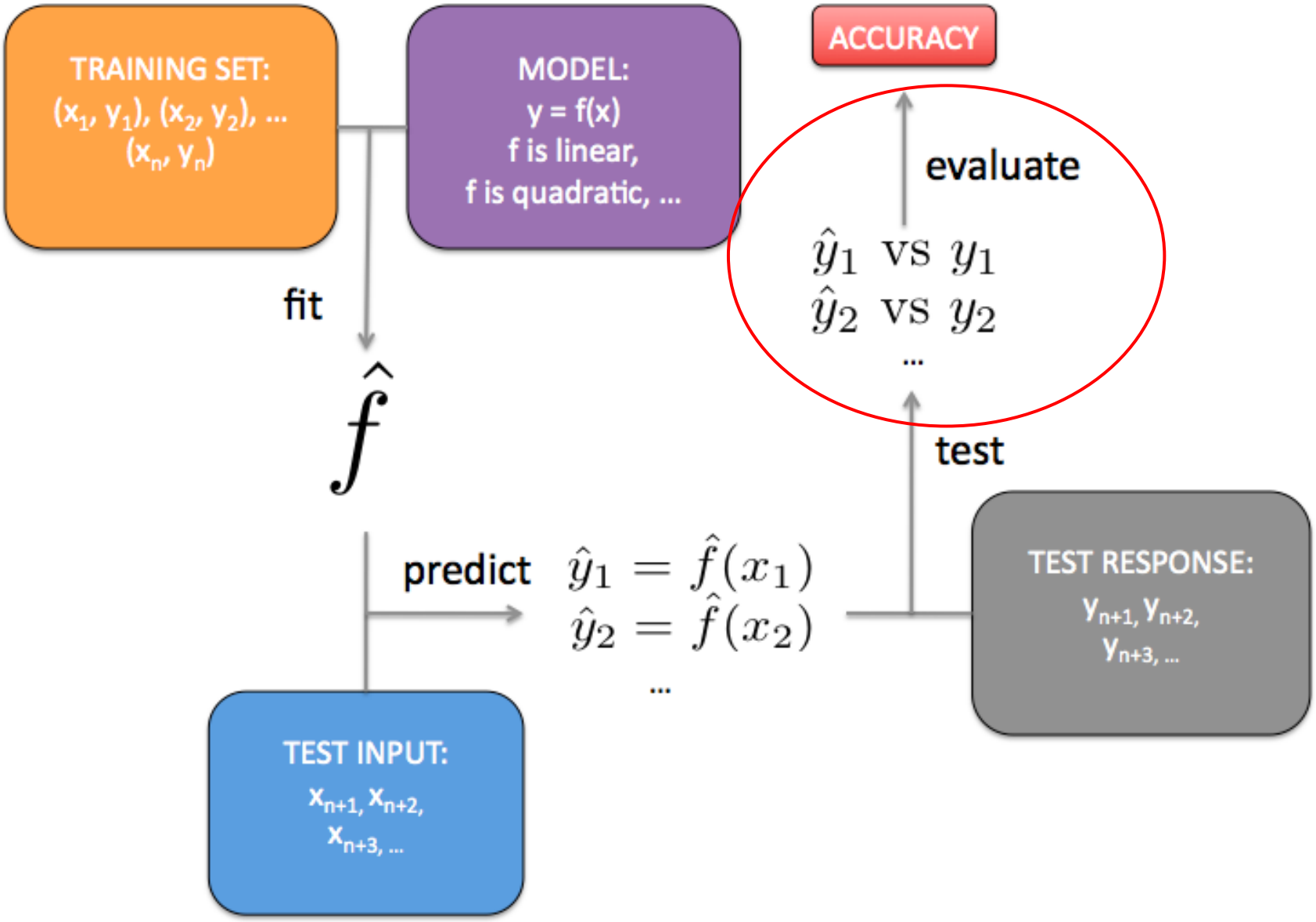
PRÉDICTION

Une fois le modèle estimé sur l'ensemble d'apprentissage, on l'utilise pour prédire les valeurs de l'ensemble de test. Pour chaque x_i du test, on prédit la réponse $\hat{y}_i$ avec $\hat{f}$:

$$\hat{y}_i = \hat{f}(x_i)$$

On peut alors **comparer** le $\hat{y}_i$ prédit avec le "vrai" y_i. Si les prédictions sont bonnes, le modèle est bon.

PIPELINE


TRAINING SET:
$(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$
MODEL:
y = f(x)
f is linear,
f is quadratic, ...
ACCURACY
evaluate
$\hat{y}_1$ vs y_1
$\hat{y}_2$ vs y_2
...
fit
$\hat{f}$
test
predict
$\hat{y}_1 = \hat{f}(x_1)$
$\hat{y}_2 = \hat{f}(x_2)$
...
TEST RESPONSE:
$y_{n+1}, y_{n+2}, y_{n+3}, \ldots$
TEST INPUT:
$x_{n+1}, x_{n+2}, x_{n+3}, \ldots$

EVALUATION

Régression : distance moyenne entre les prédictions et les vraies valeurs

$$erreur = \frac{1}{n_{test}} \sum_{i=1}^{n_{test}} (y_i - \hat{y}_i)^2$$

Classification : pourcentage des fois où le modèle a trouvé la bonne classe (TTC)

$$erreur = \frac{1}{n_{test}} \sum_{i=1}^{n_{test}} \delta(y_i = \hat{y}_i)$$

où δ(A) = 1 si A est vraie, 0 sinon.

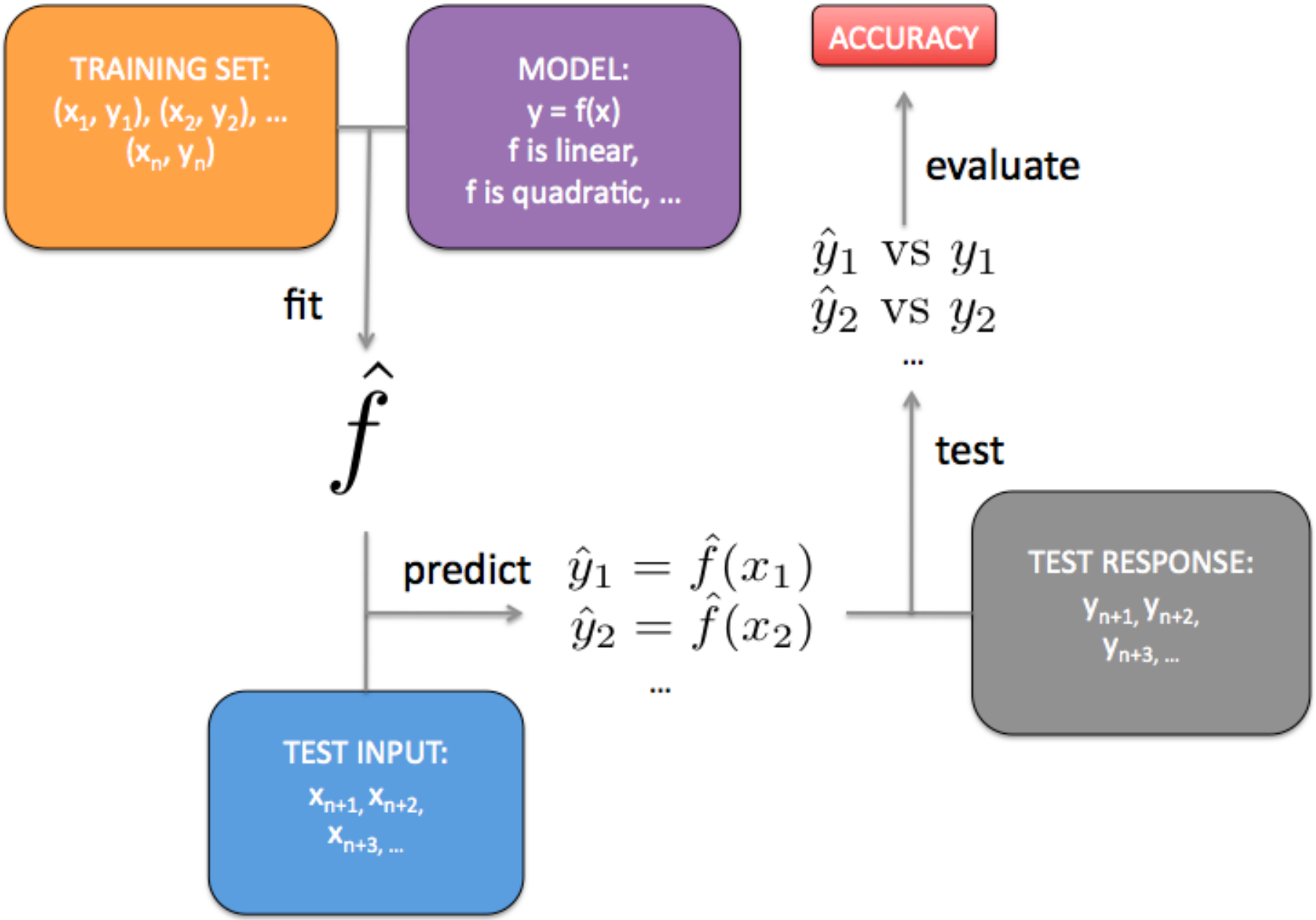
SUR-APPRENTISSAGE

On parle de **sur-apprentissage** (over-fitting) lorsque l'algorithme apprend **par cœur** l'ensemble d'apprentissage mais n'arrive pas à **généraliser** sur l'ensemble de test.

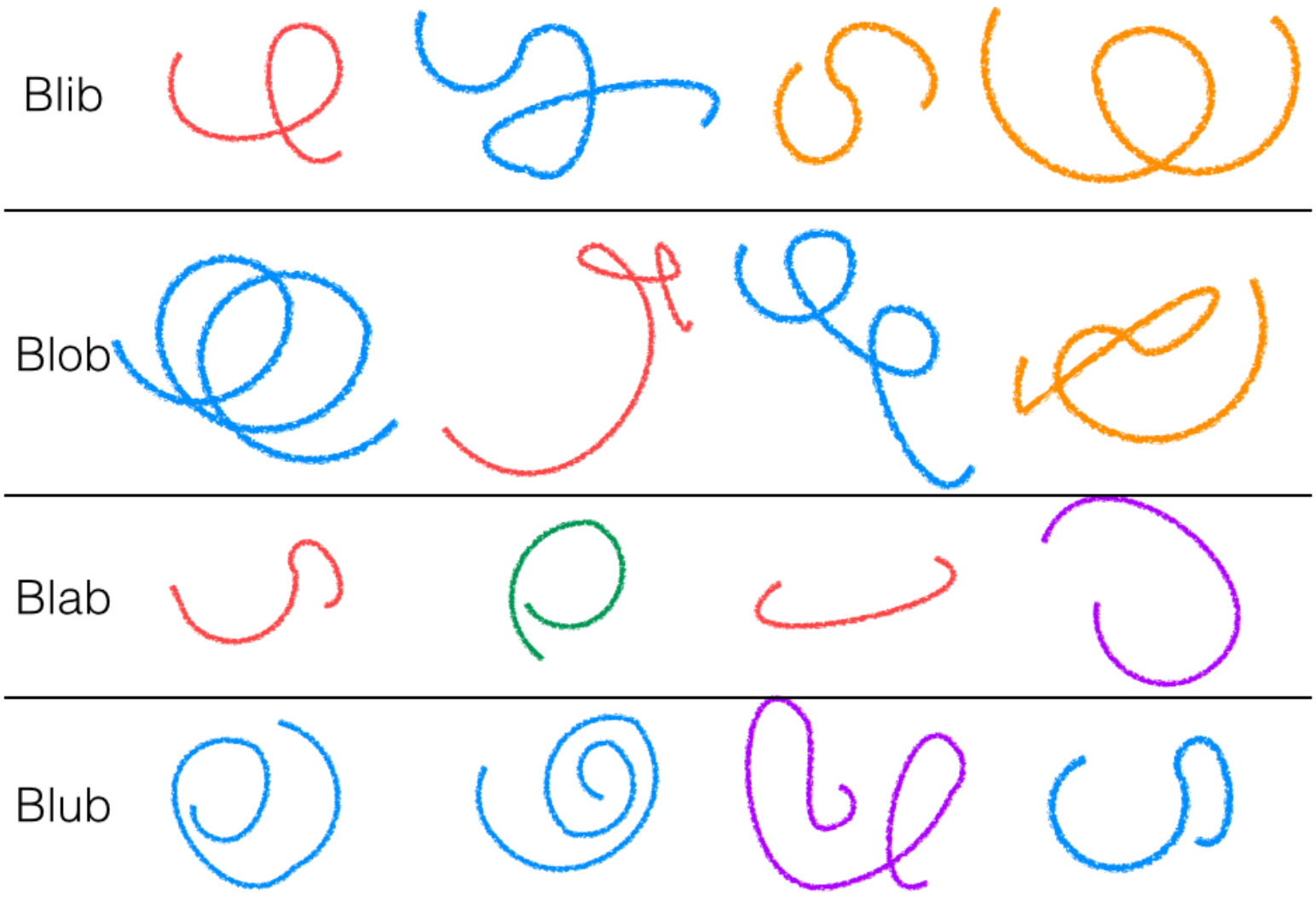
C'est pourquoi il est très important de **tester** le modèle.

Nous reviendrons sur ce point crucial en fin de chapitre.

PIPELINE

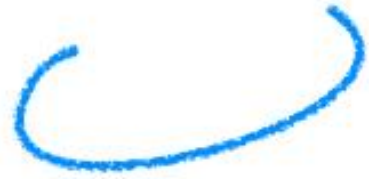

TRAINING SET:
$(x_1, y_1), (x_2, y_2), \ldots$
(x_n, y_n)
MODEL:
y = f(x)
f is linear,
f is quadratic, ...
ACCURACY
evaluate
$\hat{y}_1$ vs y_1
$\hat{y}_2$ vs y_2
...
fit
$\hat{f}$
test
predict
$\hat{y}_1 = \hat{f}(x_1)$
$\hat{y}_2 = \hat{f}(x_2)$
...
TEST RESPONSE:
$y_{n+1}, y_{n+2},$
$y_{n+3}, \ldots$
TEST INPUT:
$x_{n+1}, x_{n+2},$
$x_{n+3}, \ldots$

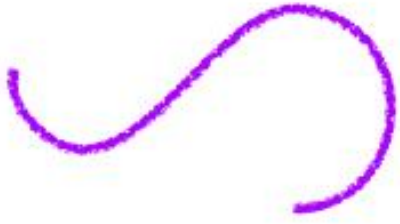
A LA PLACE DU PROGRAMME...



QUE PREDIRIEZ-VOUS?


?


?


?


?

COMMENT APPRENDRE ?

Les techniques d'apprentissage dépendent du problème.

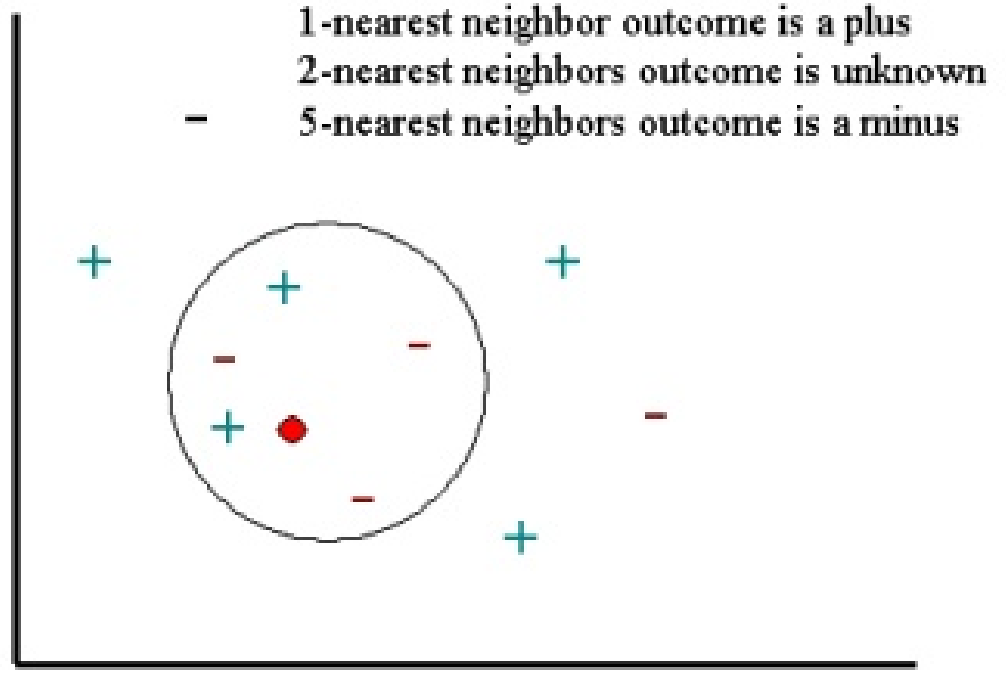
Plusieurs techniques peuvent fonctionner.
La performance d'un algorithme dépend beaucoup de l'encodage des données.

Nous allons voir différentes classes d'algorithmes qui ne ***"réfléchissent"*** **pas de la même manière**.

Les k plus proches voisins

LES K PLUS PROCHES VOISINS: PRINCIPE

K-plus proches voisins (K-nearest neighbors) : La valeur d'un point dépend de celles des points qui lui ressemblent.



Source : statsoft.com

MODÉLISATION/APPRENTISSAGE

Presque rien à faire, si ce n’est choisir un **nombre de voisins** et une **distance**.

CLASSIFICATION

Pour chaque élément de l'ensemble de test, on regarde la valeur des points voisins de l'ensemble d'apprentissage et on procède à un **vote majoritaire**.

Algorithm 1 Classification K-plus proches voisins

1: **INPUT :** : Ensemble d'apprentissage (X_{train}, Y_{train}), ensemble de test (X_{test} , Y_{test}), distance D, nombre de voisins K
2: **for** Chaque point du test x **do**
3: Calculer la distance avec chacun des points d'apprentissage ;
4: Choisir les K voisins les plus proches au sens de la distance D ;
5: Assigner à x la classe la plus fréquente chez ses K voisins ;
6: En cas d'indécision, choisir la classe du voisin le plus proche ;
7: **end for**

RÉGRESSION

Pour chaque élément de l'ensemble de test, on regarde la valeur des points voisins de l'ensemble d'apprentissage et on procède à une **moyenne**.

Algorithm 2 Régression K-plus proches voisins

1: **INPUT :** : Ensemble d'apprentissage (X_{train}, Y_{train}), ensemble de test (X_{test} , Y_{test}), distance D, nombre de voisins K
2: **for** Chaque point du test x **do**
3: Calculer la distance avec chacun des points d'apprentissage ;
4: Choisir les K voisins les plus proches au sens de la distance D ;
5: Assigner à x la moyenne des réponses de ses K voisins ;
6: **end for**

LA VALIDATION CROISEE

Afin de choisir la meilleure valeur pour K , on va procéder par **validation croisée** (cross-validation).

On **divise** l'ensemble d'apprentissage en N parties (ici, N = 5).

Tour à tour, on **entraîne** le modèle sur (N − 1) parties et on le **teste** sur la partie restante.

On calcule la **performance moyenne** sur les N tests.

On procède ainsi pour différentes valeurs de K .

On choisit la valeur de K qui a la **meilleure performance moyenne**.

VALIDATION CROISÉE, SUITE

Pourquoi utiliser la validation croisée ?

Lors du processus d'apprentissage, on ne peut pas utiliser l'ensemble de test, à aucun moment, ce serait "tricher".
En effet, si l'on utilise l'ensemble de test pour apprendre, le modèle sera forcément bon en test : **sur-apprentissage**.
Donc il faut choisir les paramètres sur l'ensemble d'apprentissage uniquement.
La validation croisée est un moyen de **tester durant l'apprentissage**.

Cette méthode est valable pour tous les algorithmes. C'est ce qu'on utilisera pour choisir le bon paramètre

CONCLUSION

Avantages :

Extrêmement simple

Intuition : faire comme ses voisins

Efficace dans certains cas et si la distance est bien choisie.

Inconvénients:

Il faut choisir K

Calculs potentiellement lourds si le nombre de points et le nombre de variables est grand.