

Autres Algos : Juste une intuition

- K plus proches voisins (k-nearest neighbors)
- SVM : Machines à vecteurs de Support (Support Vector Machines)
- Arbre de décision : Decision tree

Autres Algos

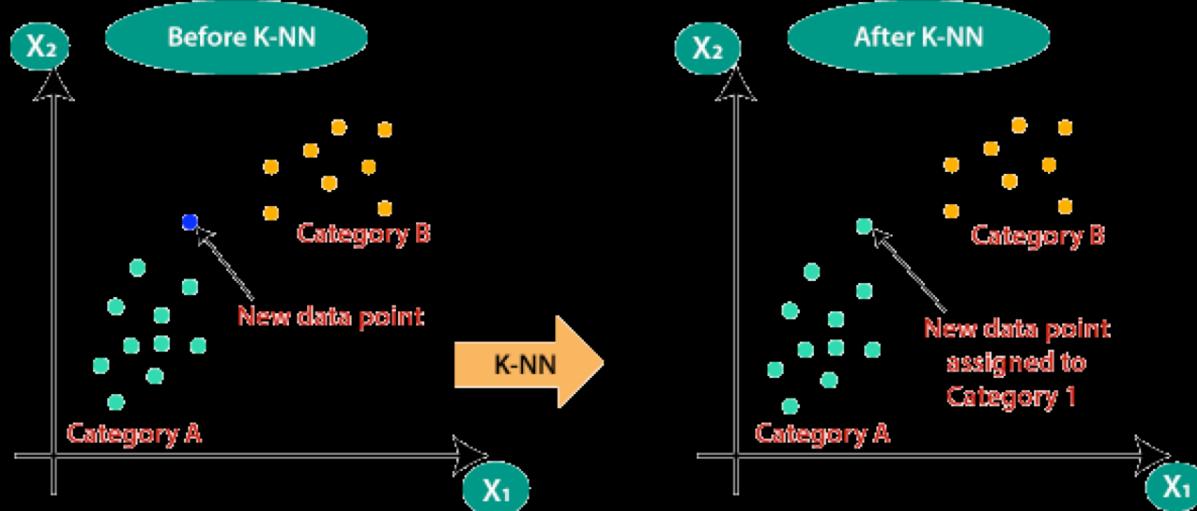
- K plus proches voisins (k-nearest neighbors)
- SVM : Machines à vecteurs de Support (Support Vector Machines)
- Arbre de décision : Decision tree

Définition

- L'algorithme de **K-nearest Neighbours** (le(s) plus proche(s) voisin(s)) permet de résoudre des problèmes régression et de classification à **plusieurs classes**
- L'idée générale :
 - Quand vous devez faire une nouvelle prédiction, trouvez dans votre Dataset les k exemples les plus proches
 - Régression : prendre la moyenne des k plus proches voisins
 - Classification : le label (l'étiquette) ayant le plus de vote

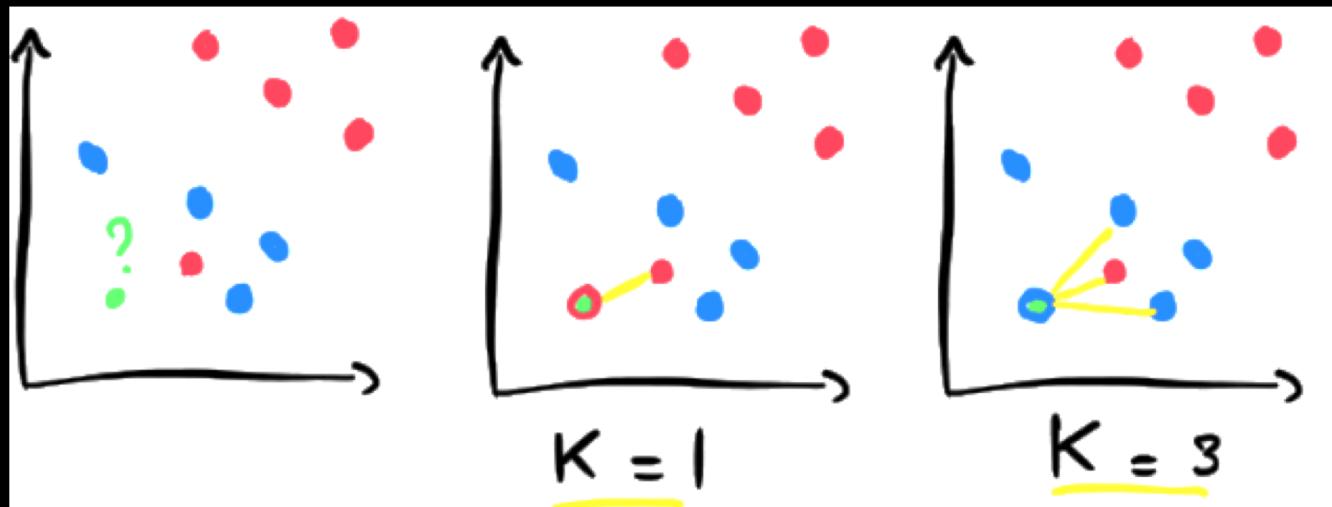
Intuition

- L'idée : choisir le plus proche voisin, en termes de distance, parmis les exemples du Dataset
- L'algorithme de Nearest Neighbour calcule la **distance** entre le point à traiter (bleu sur la figure) et les autres points du Dataset, puis associe le point à la classe dont l'exemple est le plus proche (ici la classe verte).



Le nombre de voisins K

- On peut demander à l'algorithme de trouver les **K** voisins les plus proches du point que l'on veut placer (prédir).
- Prendre plus de voisins permet d'améliorer l'algorithme.



Mise en œuvre sous cikit-learn

Classification

```
from sklearn.neighbors import KNeighborsClassifier  
  
knn = KNeighborsClassifier(n_neighbors=3)
```

Régression

```
from sklearn.neighbors import KNeighborsRegressor  
  
neigh = KNeighborsRegressor(n_neighbors=2)
```

Autres Algos

K plus proches voisins (k-nearest neighbors)

- SVM : Machines à vecteurs de Support (Support Vector Machines)
- Arbre de décision : Decision tree

SVM = Définition

- Une machine à vecteurs de support ou séparateur à vaste marge (SVM)
- Capable d'effectuer des classifications linéaires ou non linéaires, des régressions

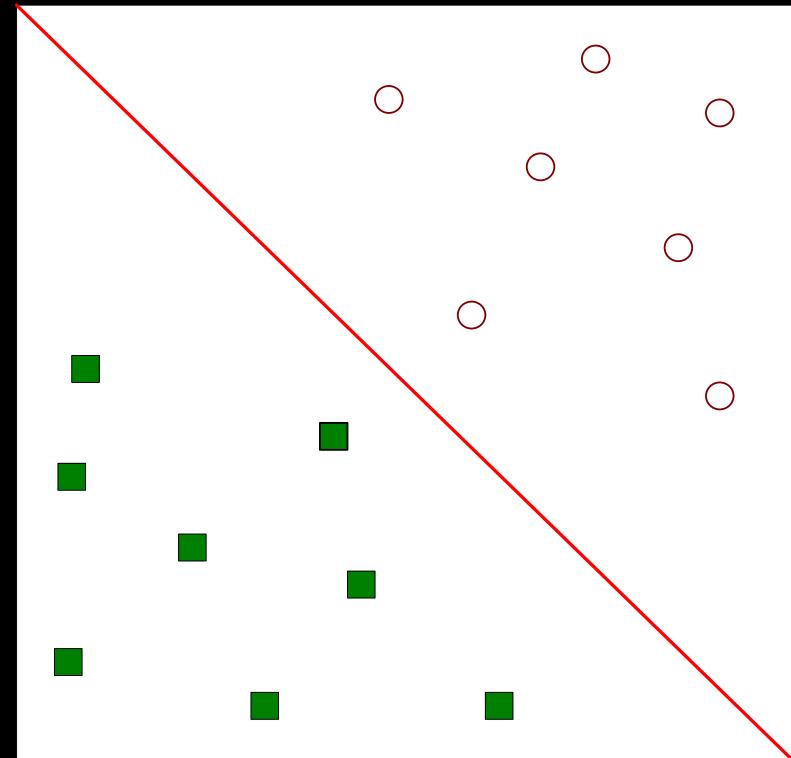
Classification par SVM

- Trouver une droite qui permet de séparer les données
- Plusieurs solutions possibles

	x1	x2	y
1	1	1	-1
2	3	1	-1
3	1	5	-1
4	4	4	-1
5	4	6	1
...
15	10	5	1

x2

x1

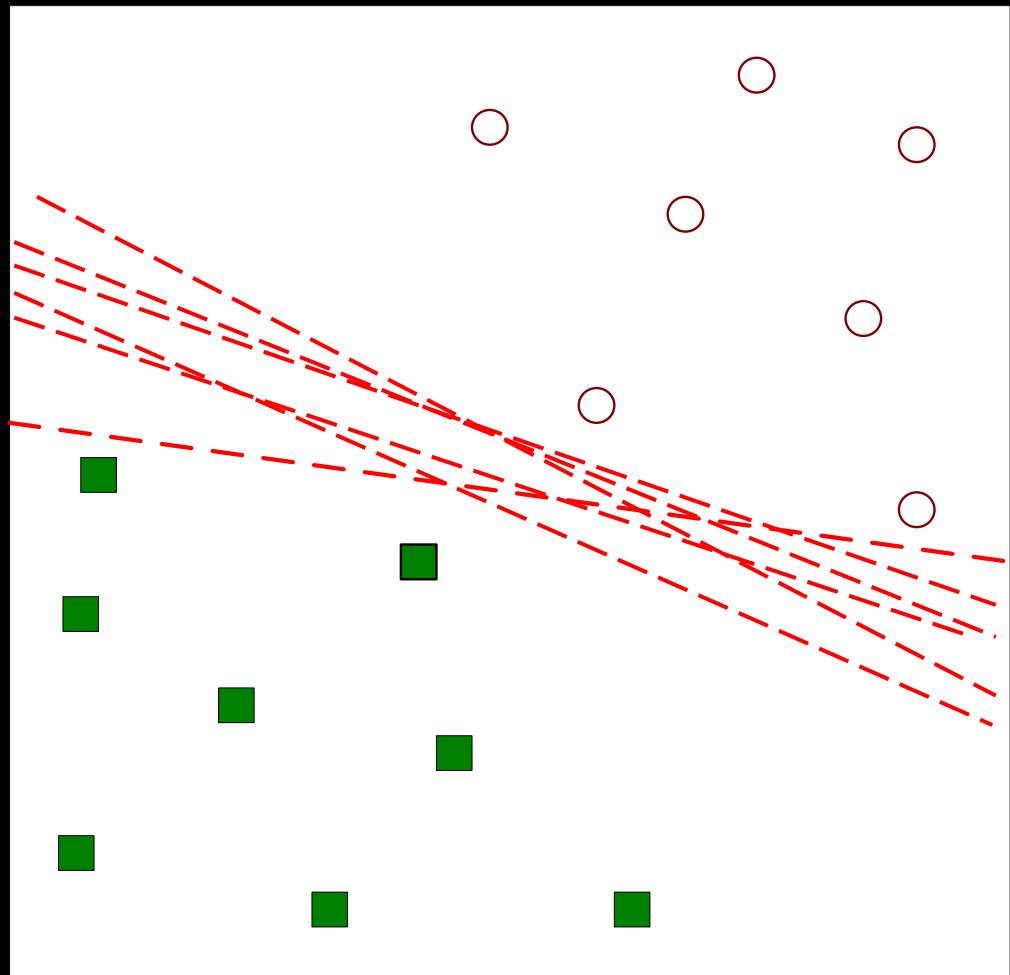


Trouver la droite (un hyperplan) de séparer les deux classes

$$f_{w,b}(x) = w_1x_1 + w_2x_2 + b$$

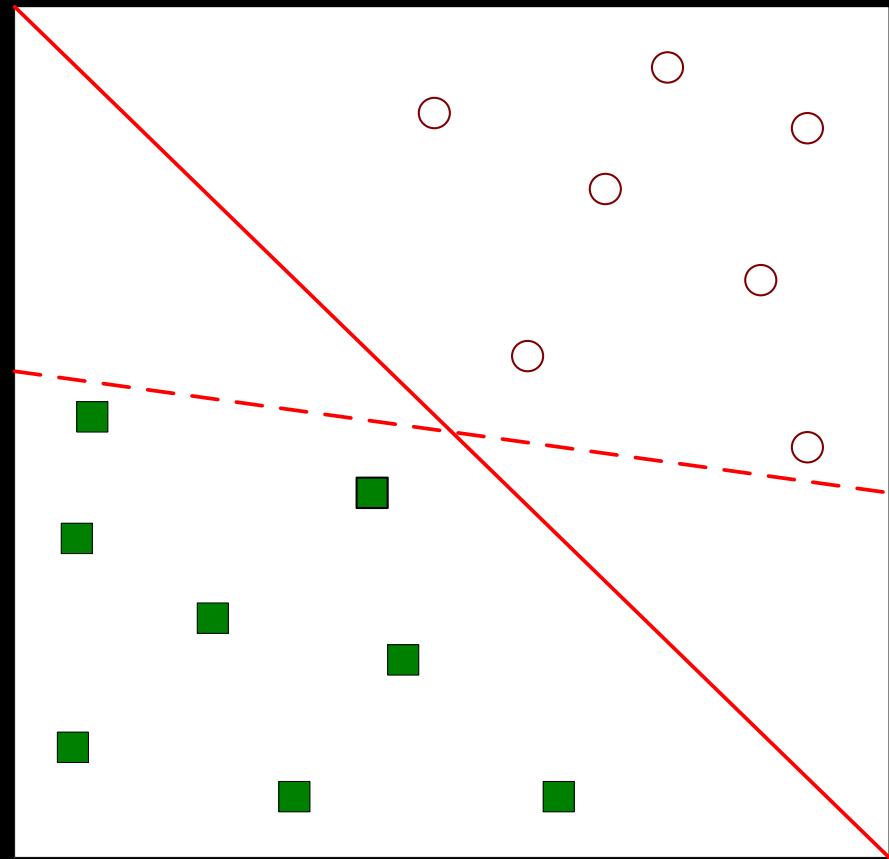
Classification par SVM

- Plusieurs solutions



Classification par SVM

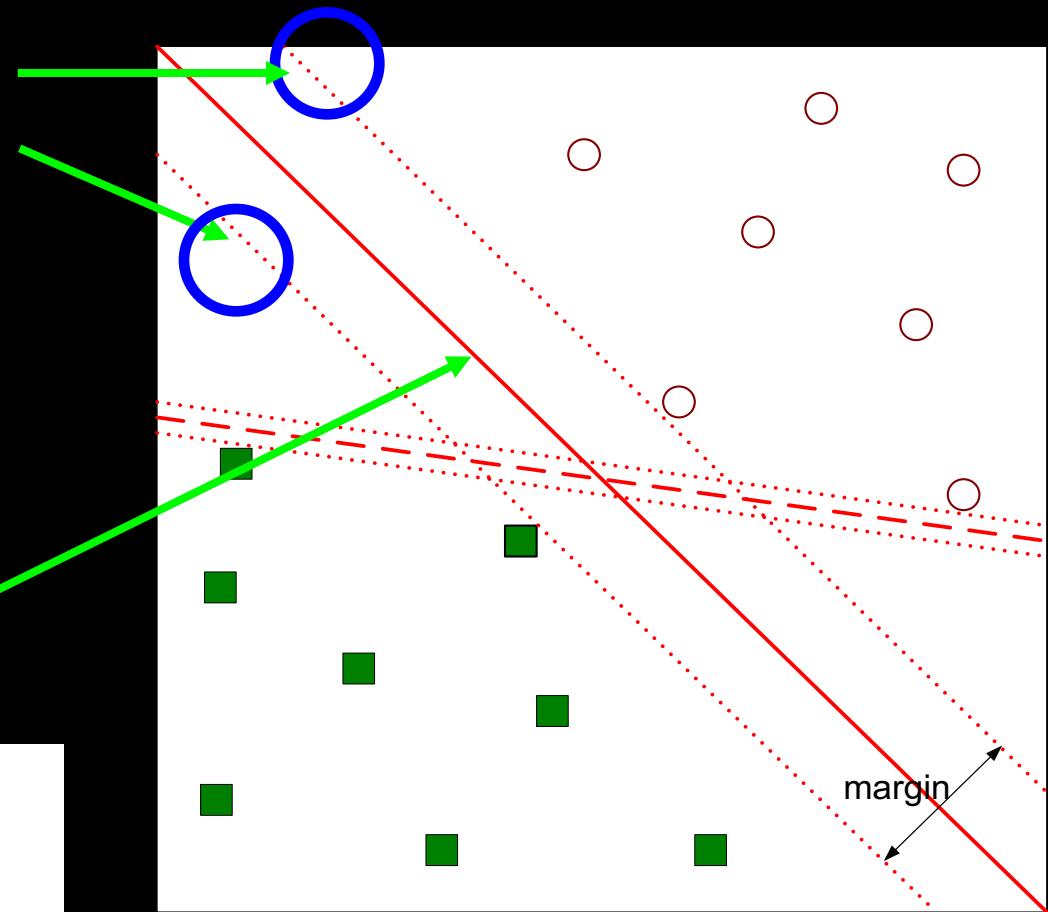
- Quel est le meilleur hyperplan?
B1 or B2?
- Comment identifier le
“meilleur”



Classification par SVM : principe

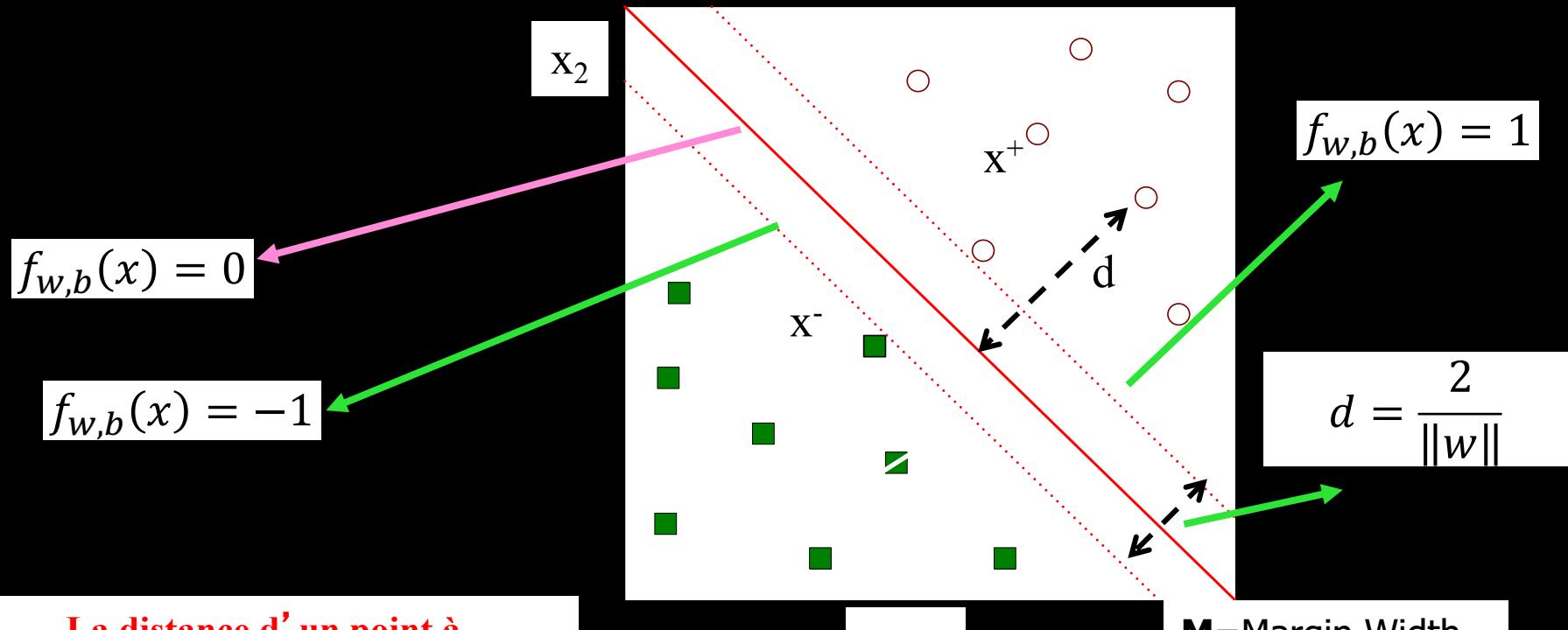
Vecteurs supports
(Support Vectors)

La frontière
 $f_{w,b}(x)=0$



- Trouver l'hyperplan qui maximise la marge (entre la frontière largeur entre les vecteurs supports de l'hyperplan)

Classification par SVM : principe



La distance d d'un point à l'hyperplan est:

$$d = \frac{|f_{w,b}(x)|}{\|w\|}$$

$$\|w\| = \sqrt{w_1^2 + w_2^2 + \cdots w_n^2}$$

On cherche un hyperplan de marge M maximale :

$$M = \frac{|(x^+ - x^-) \cdot w|}{\|w\|} = \frac{2}{\|w\|}$$

Classification par SVM : principe

- Construire un hyperplan qui maximise

$$\max \frac{2}{\|w\|} \quad \text{avec} \quad y_i (wx_i + b) \geq 1$$

$$\|w\| = \sqrt{w_1^2 + w_2^2 + \dots + w_n^2}$$

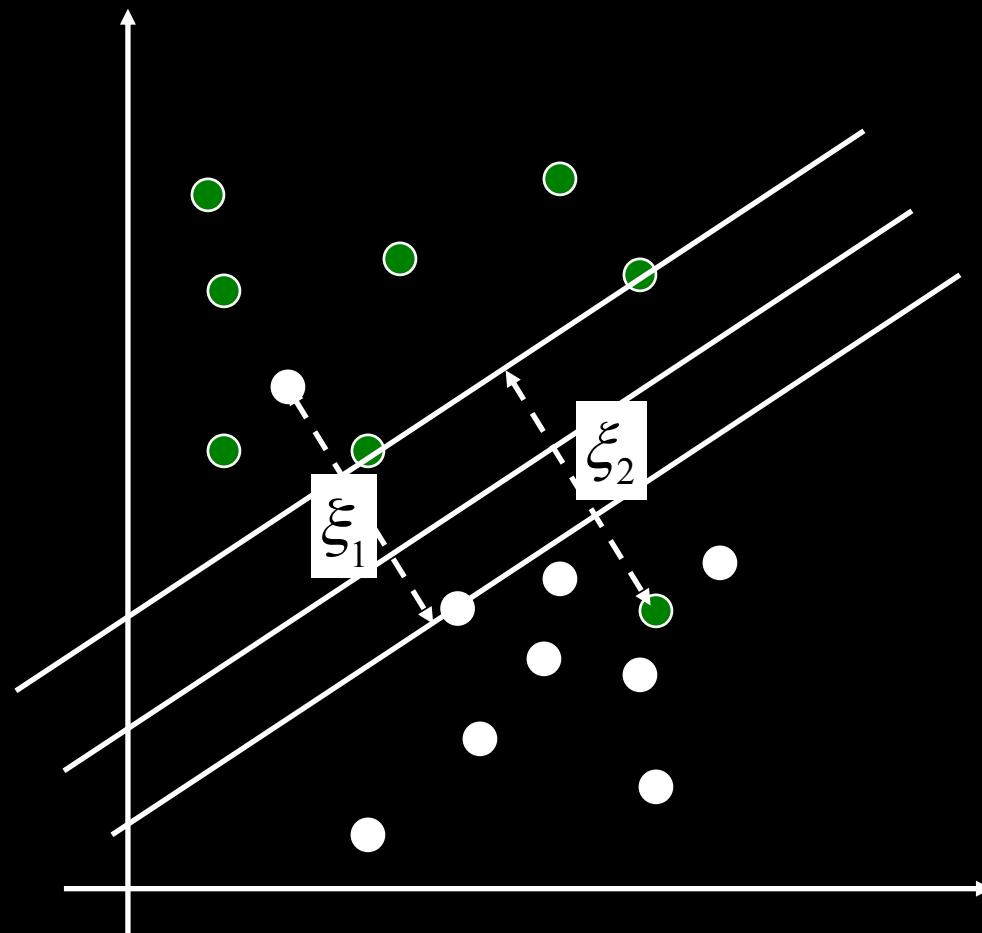
- Ceci revient à

Trouver w et b qui
minimisent $\frac{1}{2} w^T w$, pour tout $\{(x_i, y_i)\}$

Avec $y_i (wx_i + b) \geq 1$

Classification par SVM : Soft Margin

- Données bruitées
- Variables ressort (*slack variables* en anglais) ξ_i



$$f(x) = \begin{cases} 1 & \text{if } \langle w, x \rangle + b \geq 1 - \xi_i \\ -1 & \text{if } \langle w, x \rangle + b \leq -1 + \xi_i \end{cases}$$

SVM : Hard Margin v.s. Soft Margin

- Ancienne formulation:

Trouver w et b qui

Minimisent $\frac{1}{2} w^T w$ pour tout $\{(x_i, y_i)\}$

$$y_i (w^T x_i + b) \geq 1$$

- Nouvelle formulation

Trouver w et b qui

Minimisent $\frac{1}{2} w^T w + C \sum \xi_i$ Pour tout $\{(x_i, y_i)\}$

$$y_i (w^T x_i + b) \geq 1 - \xi_i \quad \text{et} \quad \xi_i \geq 0 \quad \text{quelque soit } i$$

Mise en œuvre sous Scikit-learn

- **sklearn.svm: Support Vector Machines**
- On y trouve des estimateurs pour la classification et la régression

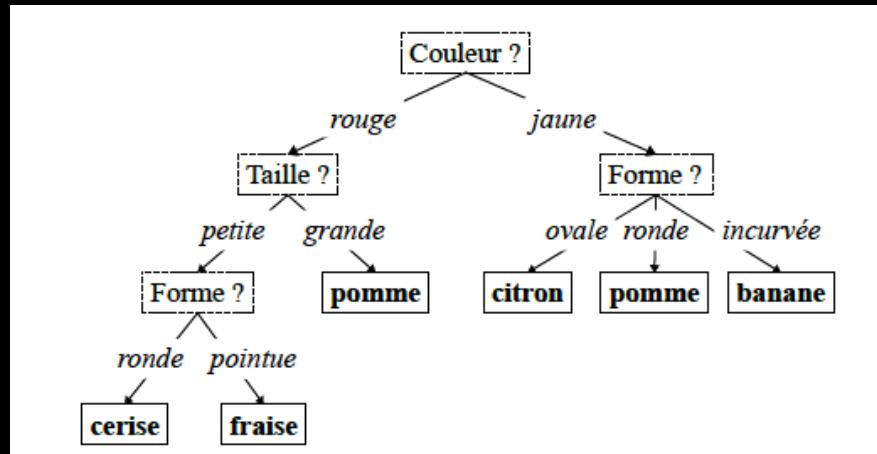
Estimators	
<code>svm.LinearSVC([penalty, loss, dual, tol, C, ...])</code>	Linear Support Vector Classification.
<code>svm.LinearSVR(*[, epsilon, tol, C, loss, ...])</code>	Linear Support Vector Regression.
<code>svm.NuSVC(*[, nu, kernel, degree, gamma, ...])</code>	Nu-Support Vector Classification.
<code>svm.NuSVR(*[, nu, C, kernel, degree, gamma, ...])</code>	Nu Support Vector Regression.
<code>svm.OneClassSVM(*[, kernel, degree, gamma, ...])</code>	Unsupervised Outlier Detection.
<code>svm.SVC(*[, C, kernel, degree, gamma, ...])</code>	C-Support Vector Classification.
<code>svm.SVR(*[, kernel, degree, gamma, coef0, ...])</code>	Epsilon-Support Vector Regression.
<code>svm.l1_min_c(X, y, *[, loss, fit_intercept, ...])</code>	Return the lowest bound for C.

Autres Algos

- K plus proches voisins (k-nearest neighbors)
- SVM : Machines à vecteurs de Support (Support Vector Machines)
- Arbres de décision (Decision tree)

Arbre de décision (Decision tree)

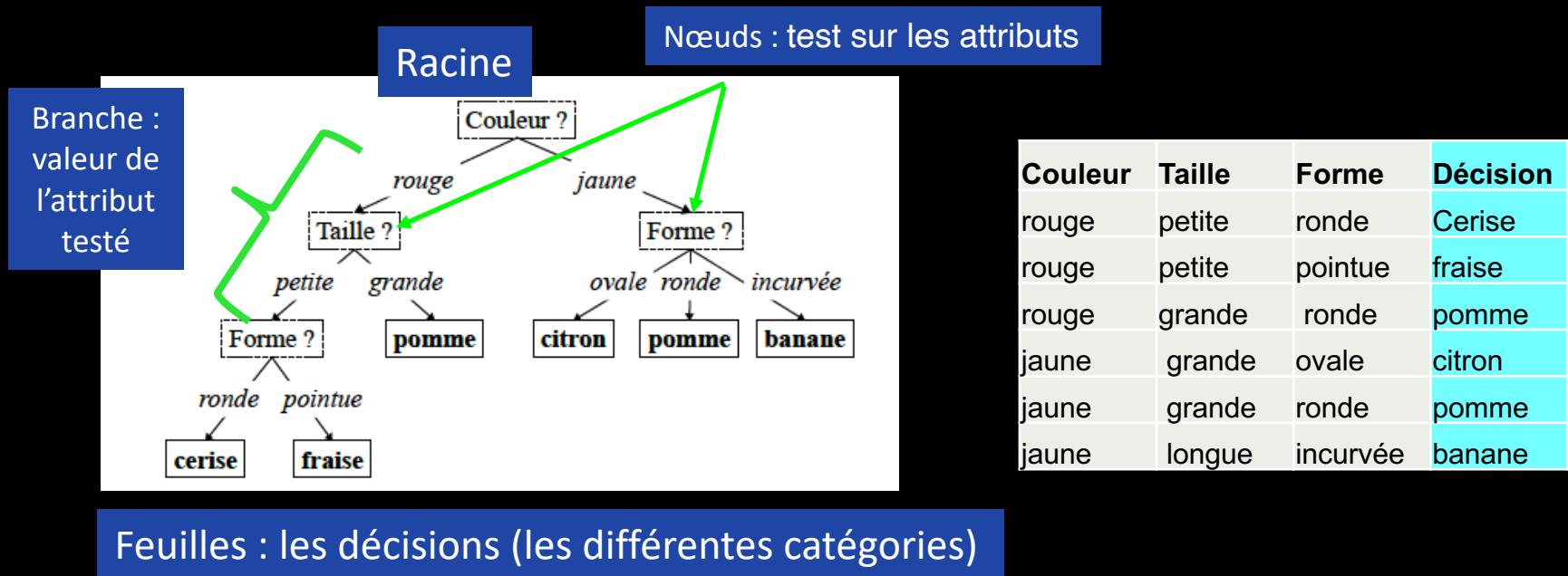
- Un arbre de décisions est un algorithme d'apprentissage supervisé non paramétrique utilisé à la fois pour les tâches de classification et de régression.
- Un arbre de decision est une suite de tests questions/réponses représentée sous forme d'une structure hiérarchique
- Chaque noeud de l'arbre teste une condition sur une variable et chacun de ses enfants correspond à une réponse possible à cette condition. Les feuilles de l'arbre correspondent à une etiquette (La decision).



Modèle d'apprentissage très utilisé dans le cas d'attributs (de variables) catégorielles

Arbre de décision (Decision tree)

- Illustration



L'apprentissage par arbre de décision consiste à partitionner les exemples d'apprentissage (les observations) en fonction des valeurs de leurs attributs

Arbre de décision (Decision tree) : Algorithme

- Phase d'entraînement
 - Soit un Dataset
 - Choisir un attribut (le meilleur) sur lequel les exemples seront partitionnés
 - Pour chaque valeur de cet attribut, créer une branche (descendant du noeud)
 - Refaire le processus pour chaque descendant
 - Affecter la décision (une valeur de sortie) quand on arrive à la feuille
 - Refaire le processus jusqu'à ce que tous les exemples soient traités (rangés)
- Phase de test
 - Soit un point (un exemple) à traiter
 - Suivre les branches de l'arbre depuis la racine jusqu'aux feuilles
 - Renvoyer la feuille (classe) la plus vraisemblable

Arbre de décision (Decision tree) : Algorithme

- Comment choisir le meilleur attribut
 - Choisir l'attribut qui permet une meilleure répartition des exemples → celui qui permet de réduire **l'impureté** des noeuds résultats(descendants)
 - Un noeud impure : noeud comportant des individus (des exemples) hétérogènes (ayant des valeurs d'attributs différentes)

Arbre de décision (Decision tree)

- Illustration : construction de l'arbre

Couleur	Taille	Forme	Décision
rouge	petite	ronde	Cerise
rouge	petite	pointue	fraise
rouge	grande	ronde	pomme
jaune	grande	ovale	citron
jaune	grande	ronde	pomme
jaune	longue	incurvée	banane

Données numériques

284 Chapter 7 Classification and Prediction

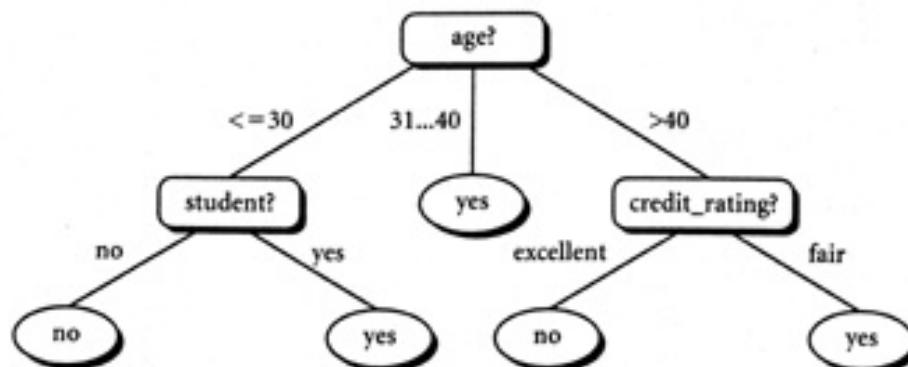


Figure 7.2 A decision tree for the concept *buys_computer*, indicating whether or not a customer at AllElectronics is likely to purchase a computer. Each internal (nonleaf) node represents a test on an attribute. Each leaf node represents a class (either *buys_computer* = yes or

Arbres de décision

- Intérêts
 - Classifieurs (souvent) interprétables contrairement aux autres algos
 - Fonctionnent bien sur des données qualitatives (données catégorielles)
 - Fonctionnent tant que le nombre de caractéristiques n'est pas trop grand

Arbres de décision : Scikit-learn

sklearn.tree: Decision Trees

The `sklearn.tree` module includes decision tree-based models for classification and regression.

User guide: See the [Decision Trees](#) section for further details.

`tree.DecisionTreeClassifier(*[, criterion, ...])` A decision tree classifier.

`tree.DecisionTreeRegressor(*[, criterion, ...])` A decision tree regressor.

`tree.ExtraTreeClassifier(*[, criterion, ...])` An extremely randomized tree classifier.

`tree.ExtraTreeRegressor(*[, criterion, ...])` An extremely randomized tree regressor.

`tree.export_graphviz(decision_tree[, ...])` Export a decision tree in DOT format.

`tree.export_text(decision_tree, *[, ...])` Build a text report showing the rules of a decision tree.

```
from sklearn.tree import DecisionTreeClassifier  
Clf = DecisionTreeClassifier()
```

```
from sklearn.tree import DecisionTreeRegressor  
reg = DecisionTreeRegressor()
```

Cartographie des algos de Machine learning



Chapitre 3 : Classification

4 notions fondamentales

- Le Dataset
- Le modèle
- La fonction coût
- L'algorithme d'apprentissage (minimisation du coût)

Classification : Dataset

- Objectif : prédire

- des sorties binaires ($y=1$ ou $y=0$) → classification binaire

X:			y:
Pclass	Sex	Age	Survived
3	male	22.0	non
1	female	38.0	oui
3	female	26.0	oui
1	female	35.0	oui
3	male	35.0	non

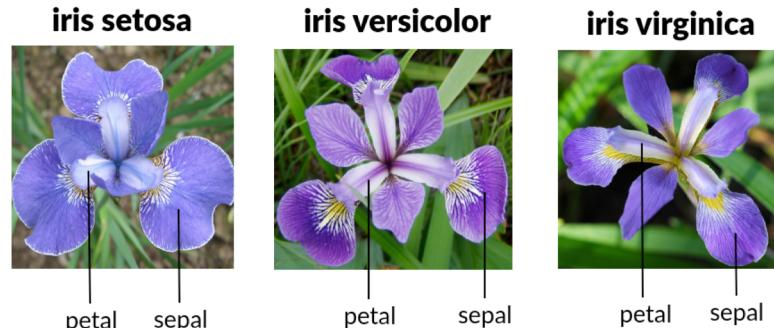
- ou un ensemble fini de valeurs discrètes ou de catégories → classification multi-classes



y: chien chat panda

Classification : Dataset

- Dataset IRIS : célèbre dataset
 - ensemble de catégories → classification multi-classes trois classes : Setosa, Versicolor et Virginia



Sepal length	Sepal width	Petal length	Petal width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
6.7	3.1	4.4	1.4	versicolor
5.8	2.7	5.1	1.9	virginica
7.1	3.0	5.9	2.1	virginica

- sepal_length / longueur du sépale
- sepal_width / largeur du sépale
- petal_length / longueur du pétales
- petal_width / largeur du pétales

```
from sklearn.model_selection import train_test_split  
# loading the iris dataset  
iris = datasets.load_iris()
```

Classification : Dataset

- Les techniques d'apprentissage automatiques ne manipulent QUE des variables Numérique
 - Dans le cas de la classification binaire Transformer les catégories en 0/1 (OUI = 1) et (NON = 0)

X:			y:
Pclass	Sex	Age	Survived
3	male	22.0	0
1	female	38.0	1
3	female	26.0	1
1	female	35.0	1
3	male	35.0	0

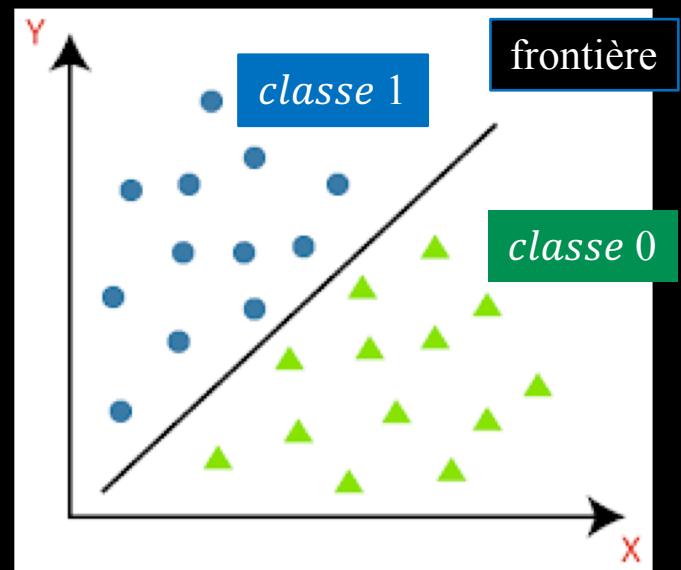
- Dans le cas de catégories → classification multi-classes, il faut donner une valeur numérique à chaque à chaque catégorie, par exemple
- Chien: 1, chat:2 et panda: 3



y: 1 2 3

Classification binaire : Le modèle

- Le modèle =Fonction de décision :
 - Dans le cas de la classification binaire : Le modèle que l'on apprend (fameuse fonction f , fonction de décision) prend directement ses valeurs dans $\{0,1\}$
 - on ajoute au modèle une frontière de décision (decision boundary) qui permet de classer un exemple (x) dans la *classe 0* ou la *classe 1*.
- Si $f_{w,b}(x) \geq \text{frontière} \rightarrow y_{pred} = 1$
- Si $f_{w,b}(x) < \text{frontière} \rightarrow y_{pred} = 0$



Algorithmes de classification

- Il existe une panoplie de méthodes (d'algos) de classification
 - Modèles linéaires
 - Régression Logistique (**adaptation de la régression linéaire**)
 - SVM (Support Vector Machine) (Machine à Vecteurs de supports),
 - Modèles Non linéaires
 - Arbres de décision (*Decision Tree Classification*)
 - Forêts aléatoires (Random Forest Classification)*
 - K-Nearest Neighbours*
 - Kernel SVM*
 - Naïve Bayes*

Régression logistique

- On est sur la même formulation que la régression linéaire
- Formulation
 - $((x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)}))$
 - $x^{(1)} \in \mathbb{R}^n$ et $y^{(i)}$ un label (0 ou 1, on se limite à ce cas)
 - $f_{w,b}$: une fonction de score linéaire
- Fonction de décision (classification binaire):
 - Les valeurs de sorties possibles (y) sont {0 ou 1}, donc il n'est pas pratique d'utiliser directement la fonction $f_{w,b}(x)$ car il est difficile de contrôler la frontière
 - il serait plus judicieux de ramener les valeurs de sorties entre 0 et 1, la frontière serait plus simple à déterminer (ça sera 0.5)

Régression logistique : Le modèle

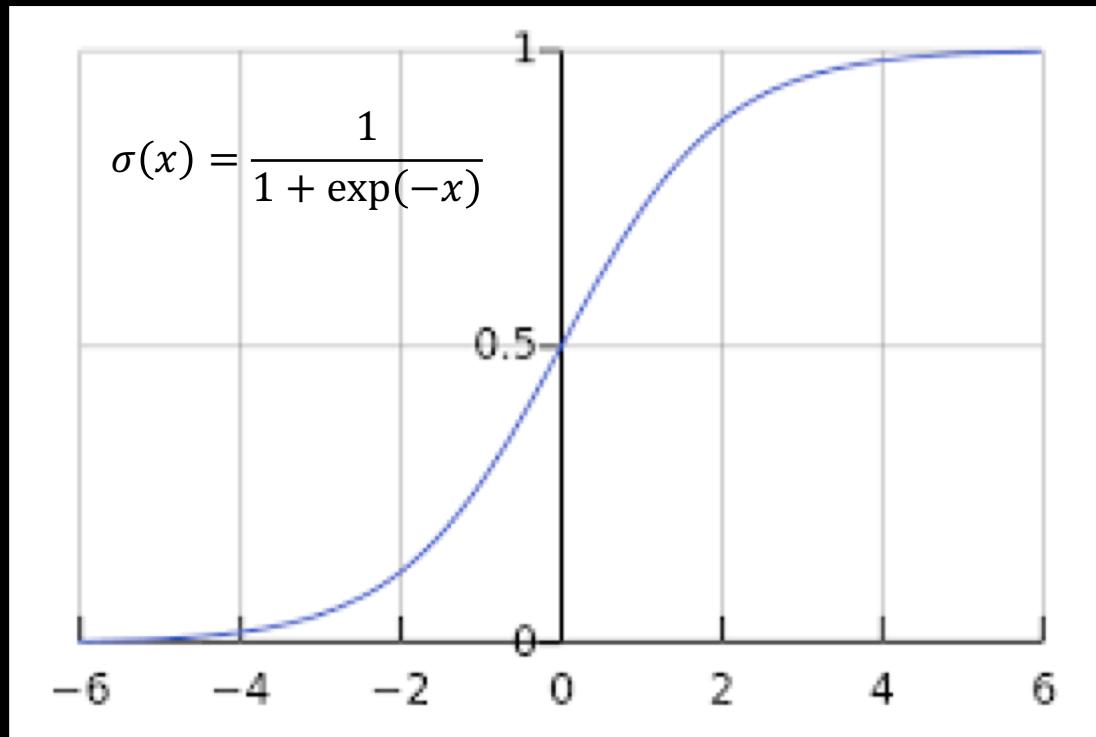
- **La fonction logistique** (aussi appelée fonction **sigmoïde** ou tout simplement sigma σ). Cette fonction a la particularité d'être toujours comprise en 0 et 1.
- Au lieu de considerer directement la valeur de sortie $z = f_{w,b}(x) \rightarrow$ on lui applique une sigmoïde

$$g_{w,b}(x) = \text{sigmoide}\left(f_{w,b}(x)\right) = \frac{1}{1 + \exp(-f_{w,b}(x))}$$

$$g_{w,b}(x) = \text{Relu}\left(f_{w,b}(x)\right) = \{0, si f_{w,b}(x) < 0; x, si f_{w,b}(x) \geq 0\}$$

Régression logistique : Le modèle

- La forme de $g_w(x) = \text{sigmoide} \left(f_{w,b}(x) \right) = \frac{1}{1+\exp(-f_{w,b}(x))}$



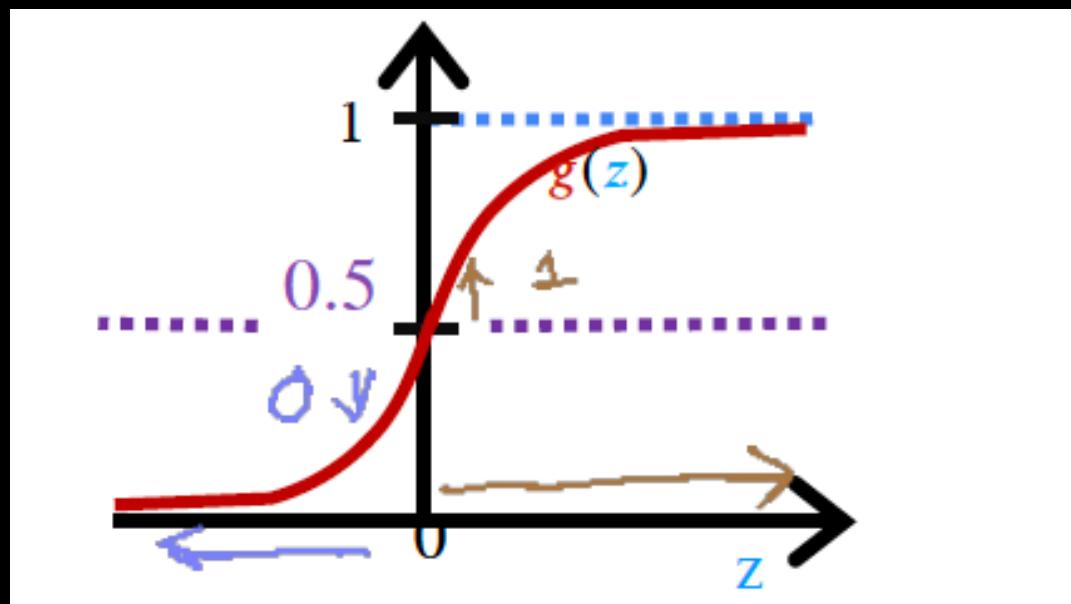
Régression logistique: frontière de décision

- A partir de cette fonction (g), il est possible de définir une **frontière de décision**. Typiquement, on définit un seuil à 0.5 comme ceci :

$$z = f_{w,b}(x) = w_1 x_1 + w_2 x_2 + b$$

$$g(x) = \frac{1}{1 + \exp(-f_{w,b}(x))}$$

$$\begin{cases} \text{si } g(x) \geq 0.5 \rightarrow y_{pred} = 1 \\ \text{si } g(x) < 0.5 \rightarrow y_{pred} = 0 \end{cases}$$



Interprétation de la régression logistique

- Interprétation de la sortie
- Considérons

- Fonction f linéaire :

$$f_{w,b}(x) = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

- Fonction de décision :

- $$g_{w,b}(x) = \frac{1}{1 + \exp(-f_{w,b}(x))}$$

- Les valeurs de g donnent la probabilité qu'un exemple (donné) appartienne à une classe (1 ou 0)

- Exemple :
 - On suppose que f est déjà apprise : w, b connus/appris
 - $X (x_1=1, x_2=0, x_3=1)$
 - $g(f_{w,b}(X))=0.7 \rightarrow$ il y a 70% de chance de survie=1 (exemple sur Excel)

$x_1:$ Pclass	$x_2:$ Sex	$x_3:$ Age	Survived
3	1	22.0	0
1	0	38.0	1
3	0	26.0	1
1	0	35.0	1
3	1	35.0	0

Régression logistique : Fonction coût

- Même principe que la régression linéaire
 - Choisir une fonction coût
 - Apprendre les valeurs w et b permettant de minimiser le coût → quelle fonction de coût ?

$x_1:$ Pclass	$x_2:$ Sex	$x_3:$ Age	Survived
3	1	22.0	0
1	0	38.0	1
3	0	26.0	1
1	0	35.0	1
3	1	35.0	0

Régression logistique : Fonction coût

- Moindres carrées (Moyenne de l'erreur quadratique)
 - On peut rester sur le même schéma que la régression linéaire

$$J(w, b) = \frac{1}{2m} * \sum_i^m \left(f_{w,b}(\vec{x}^{(i)}) - y^{(i)} \right)^2$$

Pour ne pas « traîner » plusieurs notations, on considère juste la fonction f (au lieu de g) même pour la régr. logistique

linear regression

$$f_{\vec{w}, b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$$

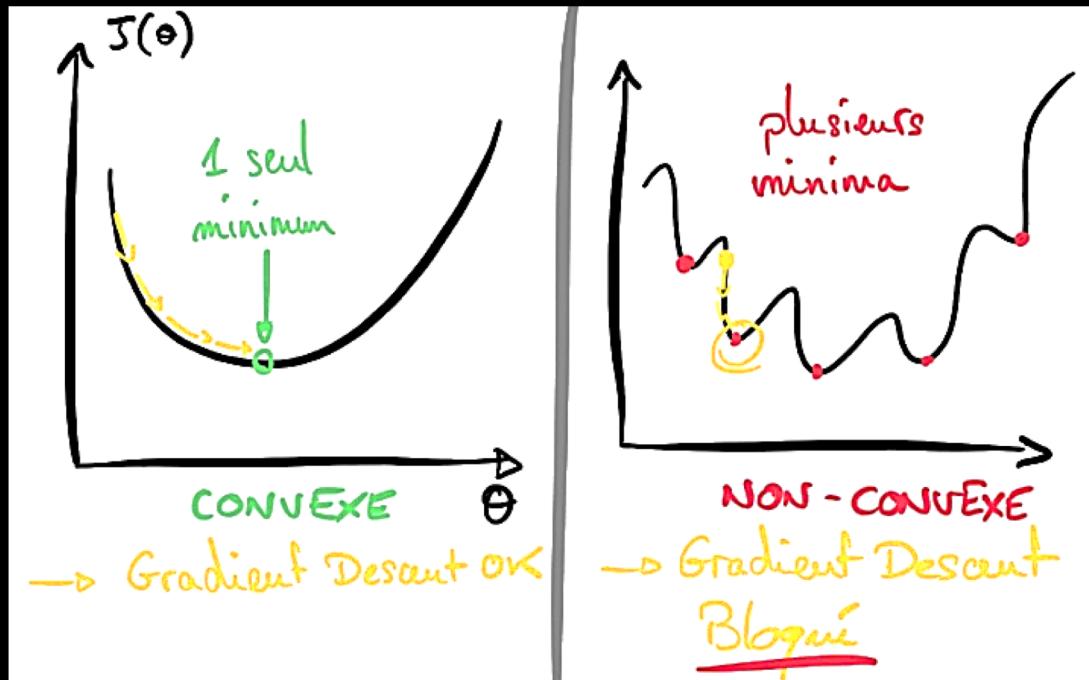
logistic regression

$$f_{\vec{w}, b}(\vec{x}) = \frac{1}{1 + e^{-(\vec{w} \cdot \vec{x} + b)}}$$

Problème avec la logistique régression → la fonction coût n'est pas convexe ?

Fonction coût : convexité

- Convexité



Utiliser les moindres carrés pour le modèle Logistique ne donnera pas de courbe convexe (dû à la non-linéarité) et l'algorithme de “Gradient Descent” bloquera au premier minima rencontré, sans trouver le minimum global.

Fonction coût logistique

- Fonction Coût spécifique pour la régression logistique. On utilise alors la fonction **logarithme** pour **transformer** la fonction f en fonction convexe en séparant les cas où $y = 1$ des cas où $y = 0$.

$$L(f_{\vec{w}, b}(\vec{x}^{(i)}), y^{(i)}) = \begin{cases} -\log(f_{\vec{w}, b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 1 \\ -\log(1 - f_{\vec{w}, b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 0 \end{cases}$$

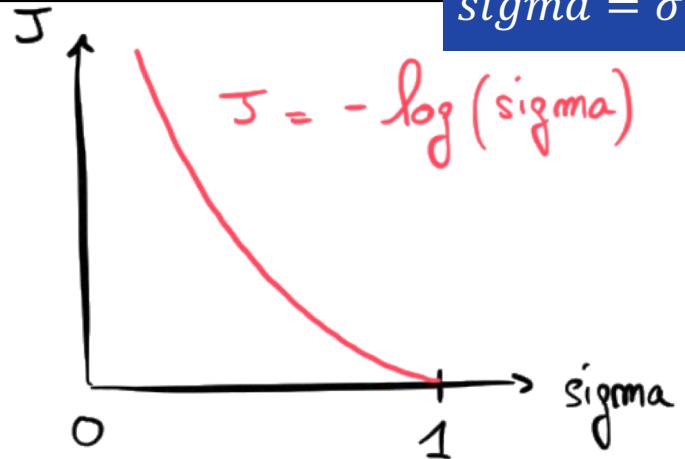
$$f_{w,b}(x) = \frac{1}{1 + \exp(-w \cdot x^{(i)} + b)}$$

Attention : w et x⁽ⁱ⁾ sont des vecteurs

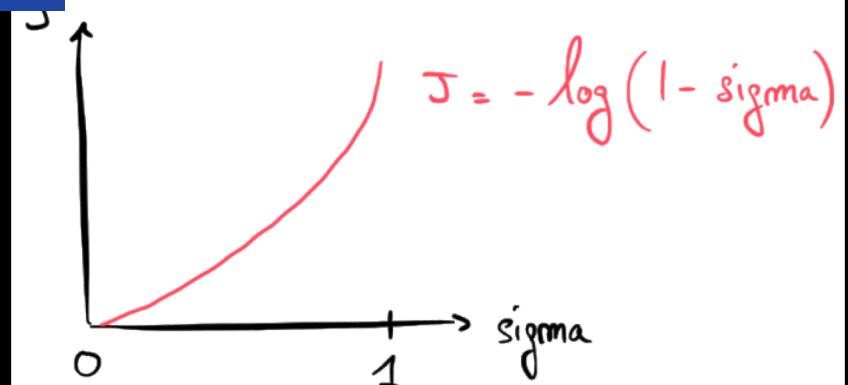
Fonction coût logistique : Explication

Cas où $y=1$

$$\text{sigma} = \sigma = f_{w,b}(x^{(i)})$$



Cas où $y=0$



Si le modèle prédit

- $f_{w,b}(x)=0$ alors que $y=1$
→ l'erreur, (Loss) très grande
→ Pénaliser
- $f(x)=1$ alors que $y=1$
→ l'erreur=0, il a tout bon

Si le modèle prédit

- $f(x)=1$ alors que $y=0$
→ l'erreur, (Loss) très grande → pénaliser
- $f(x)=0$ alors que $y=0$
→ l'erreur=0, il a tout bon

Fonction de coût : version simplifiée

- Fonction coût complète : une seule équation qui regroupe les deux cas

$$J(w, b) = \frac{-1}{m} \sum_{i=1}^m [y^{(i)} * \log(f_{w,b}(x^{(i)})) + (1 - y^{(i)}) \log(1 - (f_{w,b}(x^{(i)})))]$$

Cas où $y = 0$

$$J(w, b) = \frac{-1}{m} \sum_{i=1}^m [\cancel{y^{(i)} * \log(f_{w,b}(x^{(i)}))} + (1 - 0) \log(1 - f_{w,b}(x^{(i)}))]$$

Cas où $y = 1$

$$J(w, b) = \frac{-1}{m} \sum_{i=1}^m \cancel{1 * \log(f_{w,b}(x^{(i)}))} + (1 - \cancel{y^{(i)}}) \log(1 - \cancel{f_{w,b}(x^{(i)})})]$$

Descente du gradient

- L'algorithme du “Gradient Descent” s'applique exactement de la même manière que pour la régression linéaire.

$$J(w, b) = \frac{-1}{m} \sum_{i=1}^m [y^{(i)} * \log(f_{w,b}(x^{(i)})) + (1 - y^{(i)}) \log(1 - (f_{w,b}(x^{(i)})))]$$

Répéter {j=1..n}

$$- w_j = w_j - \alpha \frac{\partial J(w, b)}{\partial w_j}$$

$$- b = b - \alpha \frac{\partial J(w, b)}{\partial b}$$

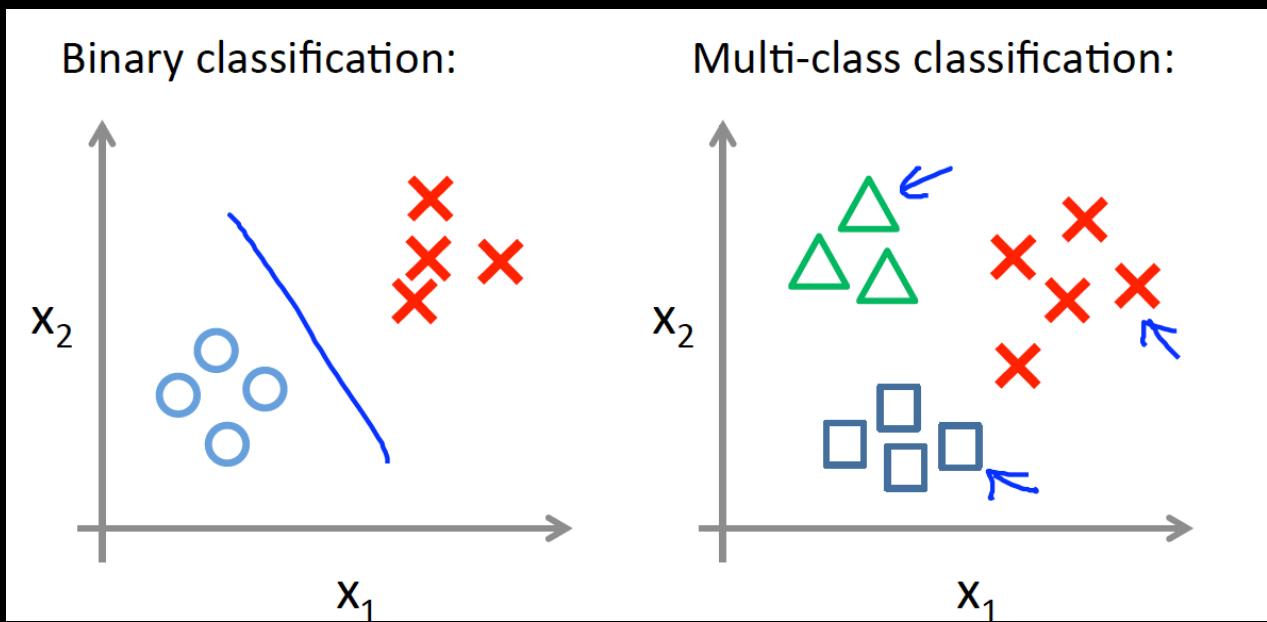
- *Modifier simultanément*

$$\frac{\partial J(w, b)}{\partial w_j} = \frac{1}{m} \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

$$\frac{\partial J(w, b)}{\partial b} = \frac{1}{m} \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)})$$

Classification multiclasses

- La régression logistique -> s'applique à des labels binaires.



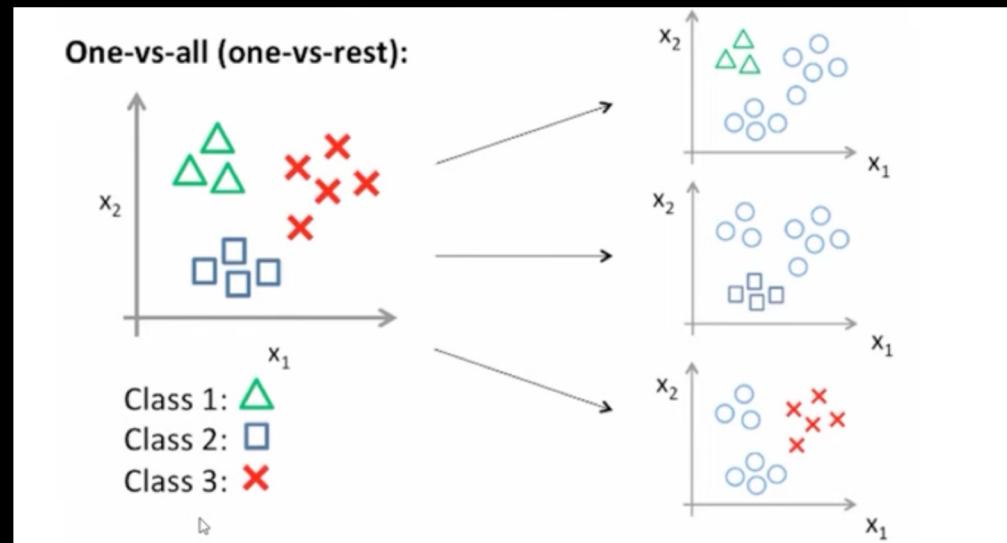
Classification multiclasses

- La variante multiclasses de la logistique régression est assez simple
 - Construire autant de modèles binaires que de classes
 - Prendre les exemples d'une classe donnée comme exemples positive (1), les exemples des autres classes comme exemples négatifs (0)
 - Appliquer l'algorithme d'apprentissage (classification binaire) pour cette classe
 - Refaire pour les autres classes.
- La technique est appelée : one-vs-all (one-vs-rest)

Classification multiclases

- La procédure
 - Prendre une classe donnée par exemple (le triangle) 
 - Les exemples du triangle seront considérés comme positifs les autres négatifs  
 - On entraîne un classifieur binaire Classique, $h^{(i)}(x)$, pour cette classe
 - Refaire la procedure, entraîner un classifieur pour chacune des autres classes.
 - A l'issue de la procedure, on obtient autant de classifieurs que de classes

Sur une nouvelle entrée (x), pour faire une prédiction, choisissez la classe qui maximise
 $\max_i h^{(i)}(x)$



Mise en œuvre sous scikit-learn

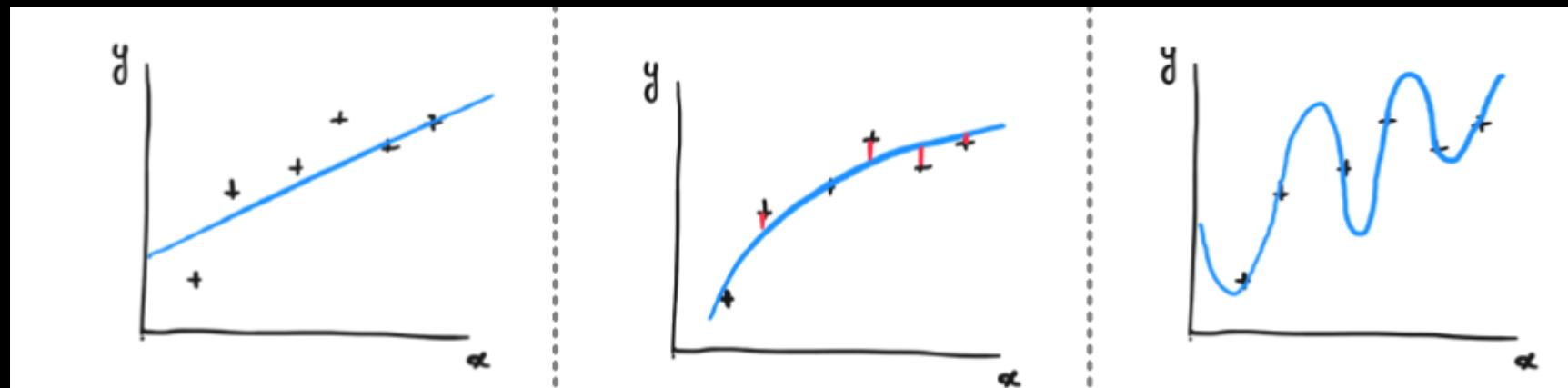
```
from sklearn.linear_model import LogisticRegression  
  
model_lg = LogisticRegression()
```

FIN

Chapitre : Sur apprentissage (overfitting) et sous apprentissage (underfitting)

Sur/sous apprentissage (over/under)fitting

- Cas de la régression



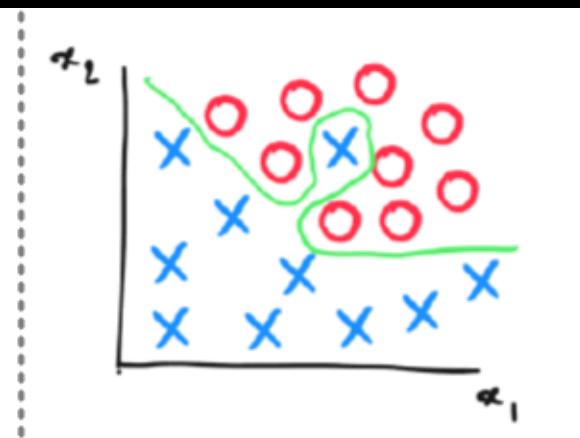
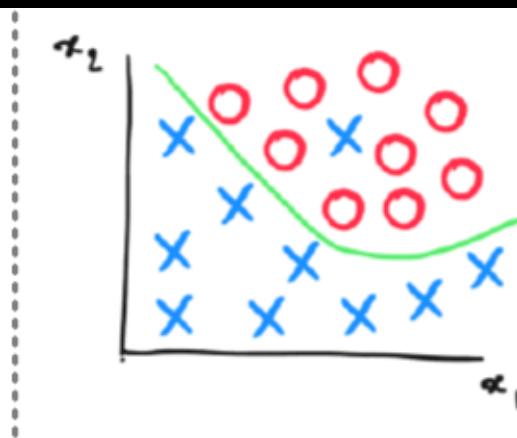
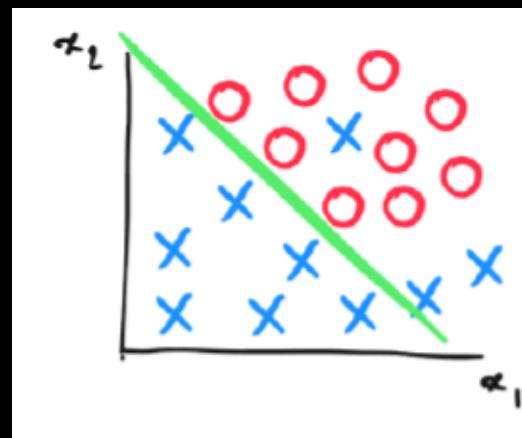
*Underfitting
Grand Biais
Le modèle appris
ne colle pas bien
aux données*

*Le bon modèle :
le juste milieu*

*Overfitting
Grand variance Le
modèle colle trop bien
aux données
d'apprentissage -> pb
de généralisation*

Sur/sous apprentissage (over/under)fitting

- Cas de la classification



*Underfitting
Grand Biais
Le modèle appris
ne colle pas bien
aux données*

*Le bon modèle :
le juste milieu*

*Overfitting
Grande variance Le
modèle colle trop bien
aux données
d'apprentissage -> pb
de généralisation*

Réduire le problème

- Prendre plus de données
- Réduire le nombre caractéristiques
 - sélectionner les caractéristiques importantes (feature selection)
- Réduire l'échelle des paramètres w et b
 - La régularisation

Régularisation

- Pénaliser la fonction Coût du modèle en y **ajoutant** un terme **de pénalité** sur ses paramètres

Dans le cas des modèles de regression aussi bien linéaire que logistique, le fait d'avoir les paramètres w et b avec des valeurs faibles permet de réduire sur apprentissage (overfitting)

Régression linéaire

$$J(w, b) = \frac{1}{2m} \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^n w_j^2$$

(Ridge ou L2
Régularisation)

Régression logistique

$$J(w, b) = \frac{-1}{m} \sum_{i=1}^m [y^{(i)} * \log(f_{w,b}(x^{(i)})) + (1 - y^{(i)}) \log(1 - (f_{w,b}(x^{(i)})))] + \frac{\lambda}{2m} \sum_{j=1}^n w_j^2$$

Régularisation

- Autres régularisations

- Lasso (L1)

$$J(w, b) = \dots \dots + \lambda \sum_{j=1}^n |w_j|$$

- ElasticNet (Lasso+Ridge)

$$J(w, b) = \dots \dots + \lambda_1 \sum_{j=1}^n |w_j| + \lambda_2 \sum_{j=1}^n |w_j|$$

Mise en œuvre sous scikit learn

Linear classifiers

`linear_model.LogisticRegression([penalty, ...])`

`linear_model.RidgeClassifier([alpha, ...])`

`penalty`{‘l1’, ‘l2’, ‘elasticnet’, None}, default=’l2’
Specify the norm of the penalty:

linear regressors

`linear_model.LinearRegression(*[, ...])`

`linear_model.Ridge([alpha, fit_intercept, ...])`

`linear_model.Lasso([alpha, fit_intercept, ...])`

`linear_model.ElasticNet([alpha, l1_ratio, ...])`

Logistic Regression (aka logit, MaxEnt) classifier.

Classifier using Ridge regression.

Ordinary least squares Linear Regression.

Linear least squares with L2 regularization.

Linear Model trained with L1 prior as regularizer (aka the Lasso)

Linear regression with combined L1 and L2 priors as regularizer.

FIN