Techniques d'apprentissage

IFT 603 / IFT 712

Combinaison de modèles

Par
Pierre-Marc Jodoin
/
Hugo Larochelle

1

1

Pourquoi utiliser un seul modèle?

- Pourquoi utiliser un seul modèle?
  - un système combinant une multitude de modèles différents ne serait-il pas meilleur?
- En pratique, la réponse est presque toujours oui !
  - le résultat de la combinaison de plusieurs modèles est appelé **ensemble** ou **comité**

2

2

Pourquoi utiliser un seul modèle?

- La façon la plus simple d'obtenir M modèles est d'utiliser M algorithmes d'apprentissage différents :
  - POUR i = 1 à m
    - Entraîner un modèle $y_{W,i}(\vec{x})$ à l'aide du i-ième algo d'entraînement
- Retourner le **modèle ensemble** (ou comité)
  - Pour la **régression**
    - $y_{COM}(\vec{x}) = \frac{1}{m}\sum_{i=1}^{m} y_{W,i}(\vec{x})$
  - Pour la **classification**
    - **vote majoritaire**

3

3

Pourquoi utiliser un seul modèle?

- **Les *M* algorithmes pourraient être le même algorithme avec avec *M* sélections d'hyperparamètres différents.**
  - POUR i = 1 à m
    - Entraîner un modèle $y_{W,i}(\vec{x})$ à l'aide du i-ième algo d'entraînement
- Retourner le **modèle ensemble** (ou comité)
  - Pour la **régression**
    - $y_{COM}(\vec{x}) = \frac{1}{m}\sum_{i=1}^{m} y_{W,i}(\vec{x})$
  - Pour la **classification**
    - **vote majoritaire**

4

4

Pourquoi utiliser un seul modèle?

- Même avec **un seul algorithme et les mêmes hyperparamètres**, on peut améliorer sa performance à l'aide d'un ensemble.
  - ***Bagging*** : approprié pour combiner des modèles avec une **forte capacité**
  - ***Boosting*** : approprié pour combiner des modèles avec une **faible capacité**

5

5

Pourquoi utiliser un seul modèle?

- Même avec **un seul algorithme et les mêmes hyperparamètres**, on peut améliorer sa performance à l'aide d'un ensemble.
  - ***Bagging*** : approprié pour combiner des modèles avec une **forte capacité**
  - ***Boosting*** : approprié pour combiner des modèles avec une **faible capacité**

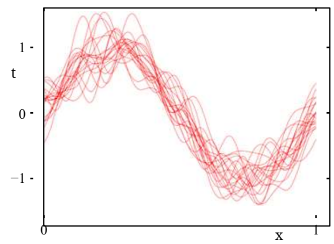
6

6

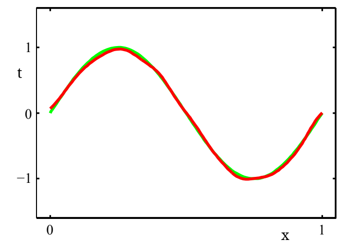
Boostrap : réduction de la variance

• Régression polynomiale de degré 25

100 modèles entraînés sur
100 ensembles d'entraînement différents



Ensemble des 100 modèles vs. vrai modèle

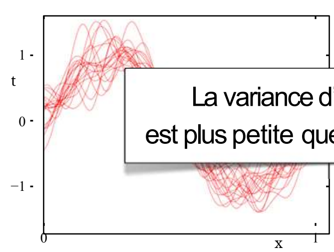


7

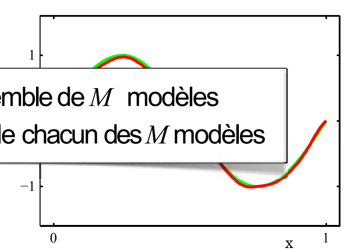
Boostrap : réduction de la variance

• Régression polynomiale de degré 25

100 modèles entraînés sur
100 ensembles d'entraînement différents



Ensemble des 100 modèles vs. vrai modèle



La variance d'un ensemble de M modèles
est plus petite que celle de chacun des M modèles

8

Boostrap

À part pour des données synthétiques, on ne peut pas produire des données sur demande.

Solution : ***Boostraping***.

$D_{bootstrap} = \{\ \}$

Pour N itérations

- Choisir aléatoirement un entier parmi $\{1,\ldots,N\}$
- $D_{bootstrap} = D_{bootstrap} \cup \{(\vec{x}_n, t_n)\}$

retourner $D_{bootstrap}$

9

9

Boostrap

À part pour des données synthétiques, on ne peut pas générer des données sur demande.

Solution : ***Boostraping***.

$D_{bootstrap} = \{\ \}$

Pour N itérations

- Choisir aléatoirement un entier parmi $\{1,\ldots,N\}$
- $D_{bootstrap} = D_{bootstrap} \cup \{(\vec{x}_n, t_n)\}$

échantillonne N exemples avec remplacement

retourner $D_{bootstrap}$

10

10

Bagging (**B**oostrap **AGG**regat**ING**)

1. Générer m ensembles d'entraînement avec ***Boostrap*** $\{D_1, D_2, ..., D_m\}$

2. **Entraîner** m modèles $y_{W,i}(\vec{x})$ (un pour chaque ensemble)

3. **Combiner** les m modèles

Régression: $y_{com}(\vec{x}) = \frac{1}{m}\sum_{i=1}^{m} y_{W,i}(\vec{x})$

Classification: $y_{com}(\vec{x}) = sign\left(\sum_{i=1}^{m} y_{W,i}(\vec{x})\right)$ (2 classes)

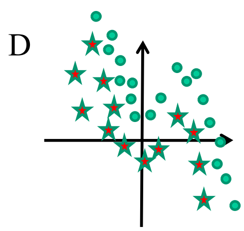
$y_{com}(\vec{x}) = \arg\max_c \left(\sum_{i=1}^{m} y_{W,i}(\vec{x})\right)$ (K classes)

11

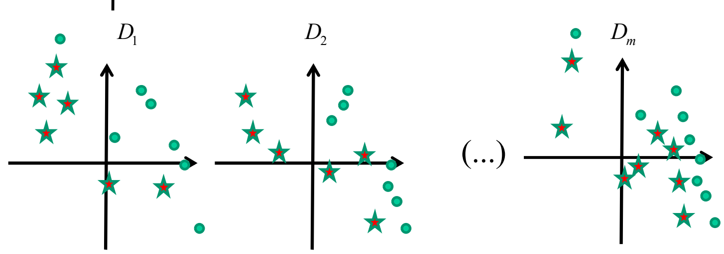
11

Illustration: classification 2Classes

1- **échantillonnage** avec replacement (*Boostrap*)



Chaque ensemble D_i contient N échantillons



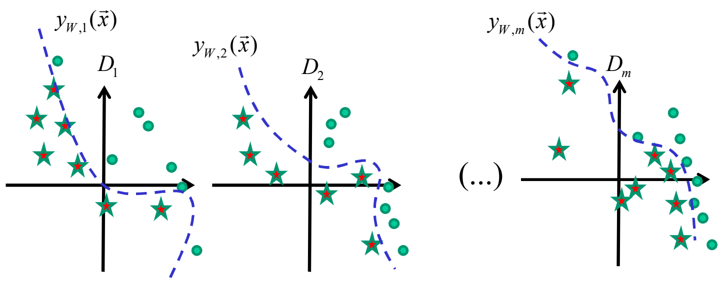
Note: un échantillon $\vec{x}_j$ peut apparaître plusieurs fois dans un même ensemble d'entraînement D_i

12

Illustration: classification 2Classes

2- **Entraînement**

Entraîner un modèle $y_{W,i}(\vec{x})$ (peut être perceptron, SVM, noyaux, etc.) sur chaque ensemble d'entraînement



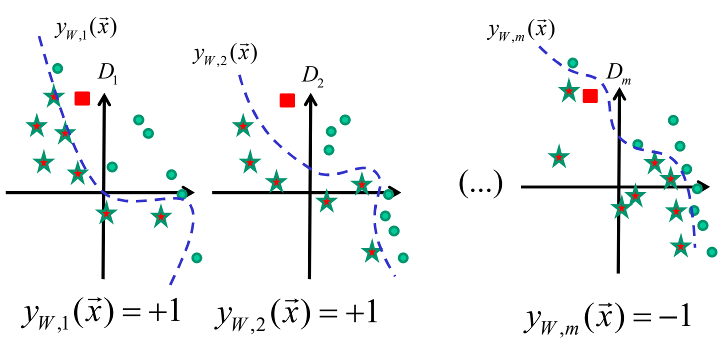
13

13

Illustration: classification 2Classes

3- **Vote majoritaire**

$$y_{com}(\vec{x}) = sign\left(\sum_{i=1}^{m} y_{W,i}(\vec{x})\right)$$



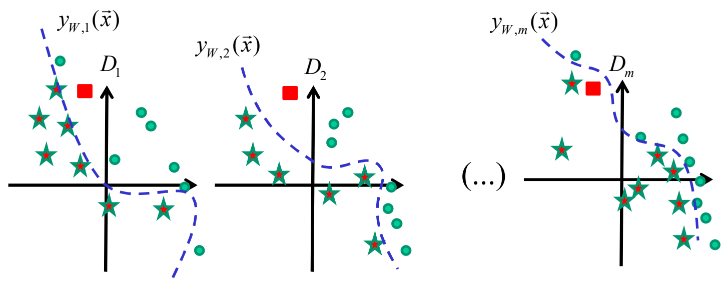
14

14

Illustration: classification 2Classes

3- **Vote majoritaire**

$$y_{com}(\vec{x}) = sign\left(1+1-1+1-1-1+...+1\right) = +1$$



$y_{W,1}(\vec{x}) = +1$ $y_{W,2}(\vec{x}) = +1$ $y_{W,m}(\vec{x}) = -1$

Bagging

Analyse théorique de l'erreur : au tableau

17

Pourquoi utiliser un seul modèle?

Même avec un seul algorithme sans hyper-paramètres, on peut améliorer sa performance à l'aide d'un ensemble.

- ***Bagging*** : approprié pour combiner des modèles avec une **forte capacité**
- ***Boosting*** **:** approprié pour combiner des modèles avec une **faible capacité**

18

AdaBoost

La méthode du boosting a pour objectif de **combiner plusieurs modèles faibles** (*week learners*) afin d'obtenir un classifieur avec une plus grande capacité.

Trois (3) différences majeures avec le Bagging.

1. Implémente une **combinaison pondérée** de modèles. Ex. 2 classes

$$y_{com}(\vec{x}) = sign\left(\sum_{i=1}^{m} \alpha_i y_{W,i}(\vec{x})\right)$$

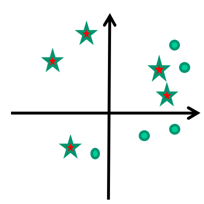
2. **Pas de bootstrap**: chaque donnée $\vec{x}_n$ est utilisée pour entraîner les modèles

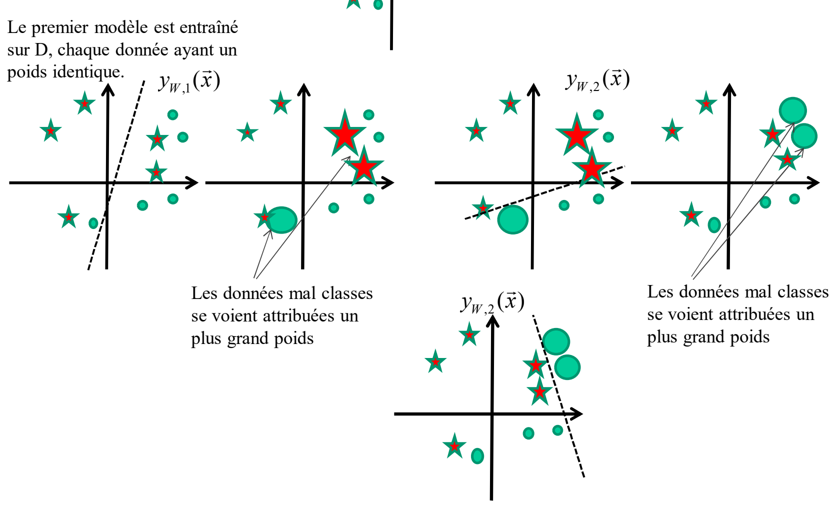
3. Les **données mal classées** par un modèle $y_{W,i}(\vec{x})$ auront **plus de poids** lors de l'entraînement du prochain modèle $y_{W,i+1}(\vec{x})$

R. Schapire and Y. Freund **A decision theoretic generalization of on-line learning and an application to Boosting** Journal of Computer and System Sciences, 1997, 55: 119-139.

AdaBoost

(illustration 2 classes)

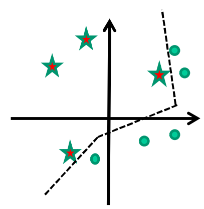




AdaBoost
(illustration 2 classes)

Le modèle combiné implémente une **combination pondérée** des 3 classifieurs faibles

$$y_{com}(\vec{x}) = sign\left(\alpha_1 y_{w,1}(\vec{x}) + \alpha_2 y_{w,2}(\vec{x}) + \alpha_3 y_{w,3}(\vec{x})\right)$$



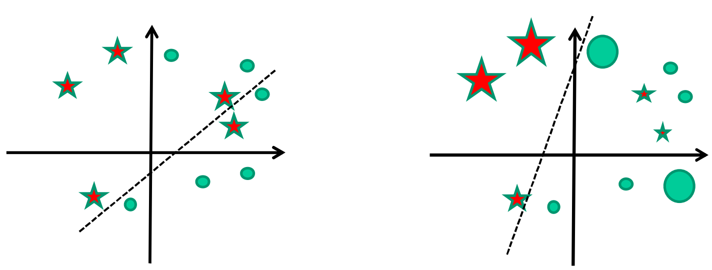
NOTE: plus un classifieur a une **exactitude élevée**, plus son α_i sera élevé. 21

21

AdaBoost

Idée fondamentale: chaque donnée $\vec{x}_i$ a un **poids** β_i

Lorsque les données ont toutes un **poids égale**, alors le **modèle faible** devient un classifieur linéaire comme un **perceptron** ou une regression logistique.



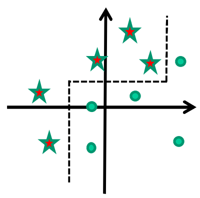
Poids égaux Vs poids non égaux

22

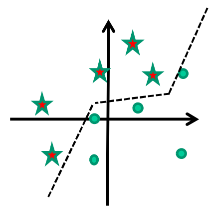
22

AdaBoost

Les modèles de type ***stump*** sont des classifieurs perpendiculaires à un axe
La combinaison de modèles ***stump*** mène à des frontières de décision "crénelées"



Au lieu de



2 advantages: très **rapide** et permet d'identifier les **caractéristiques utiles**:

$$y_{com}(\vec{x}) = sign\left(\sum_{i=1}^{m} \alpha_i y_{W,i}(\vec{x})\right)$$

Les caractéristiques utiles sont celles pour lesquelles α_i est élevé

23

Adaboost

1- initialiser le poids des N données d'apprentissage: $\beta_i = \frac{1}{N} \quad \forall i$

POUR *i=1 à m*

2- Entraîner le modèle $y_{W,i}(\vec{x})$ avec les données D et les poids $\{\beta_1,...,\beta_N\}$
3- Calculer l'erreur d'entraînement: ε_i

$$\varepsilon_i = \frac{\sum_{\vec{x}_k \in \Psi} \beta_k}{\sum_{i=1}^{N} \beta_i} \quad \text{où } \Psi \text{ l'ensemble des données mal classées}$$

4- calculer $\alpha_i = \ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$

5- mise à jour du poids des données mal classées par $y_{W,i}(\vec{x})$

$$\beta_n = \beta_n \exp\{\alpha_i\}$$

6- Normaliser les poids afin qu'ils somment à 1

$$\beta_k = \frac{\beta_k}{\sum_j \beta_j}$$

Le classifieur combiné: $y_{com}(\vec{x}) = sign\left(\sum_{i=1}^{m} \alpha_i y_{W,i}(\vec{x})\right)$ 24

24

Adaboost

1- initialiser le poids des N données d'apprentissage: $\beta_i = \frac{1}{N} \quad \forall i$

POUR *i=1 à m*

2- Entraîner le modèle $y_{W,i}(\vec{x})$ avec les données D et les poids $\{\beta_1,...,\beta_N\}$

3- Calculer l'erreur d'entraînement: ε_i

$$\varepsilon_i = \frac{\sum_{\vec{x}_k \in \Psi} \beta_k}{\sum_{i=1}^{N} \beta_i}$$

où Ψ l'ensemble d

Plus un classifieur a une erreur faible, plus son poids α_i sera élevé

4- calculer $\alpha_i = \ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$

5- mise à jour du poids des données mal classées par $y_{W,i}(\vec{x})$

$$\beta_n = \beta_n \exp\{\alpha_i\}$$

6- Normaliser les poids afin qu'ils somment à 1

$$\beta_k = \frac{\beta_k}{\sum_j \beta_j}$$

Le classifieur combiné: $y_{com}(\vec{x}) = sign\left(\sum_{i=1}^{m} \alpha_i y_{W,i}(\vec{x})\right)$

25

25

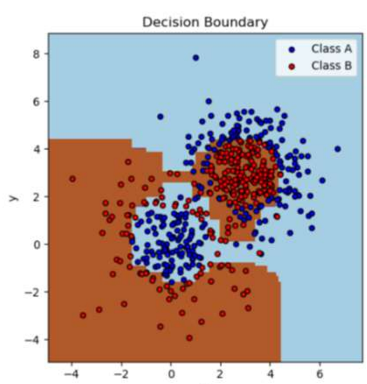
Sklearn

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
(…)

bdt = AdaBoostClassifier(DecisionTreeClassifier(max_depth=1),
                         algorithm="SAMME",
                         n_estimators=200)

bdt.fit(X, y)
```

Classifieur « Stump »
Algo de la page précédente
200 modèles



26

26

27

27

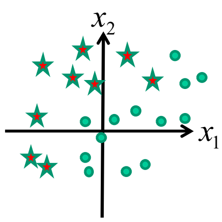
Arbres de décision

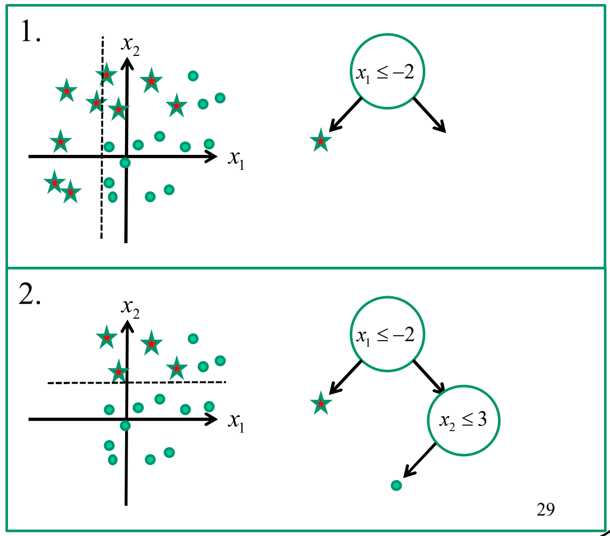
28

28

Arbres de décision

Exemple simple avec des classifieurs *stump*

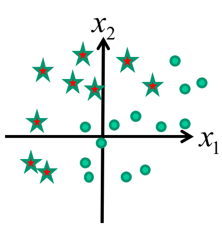


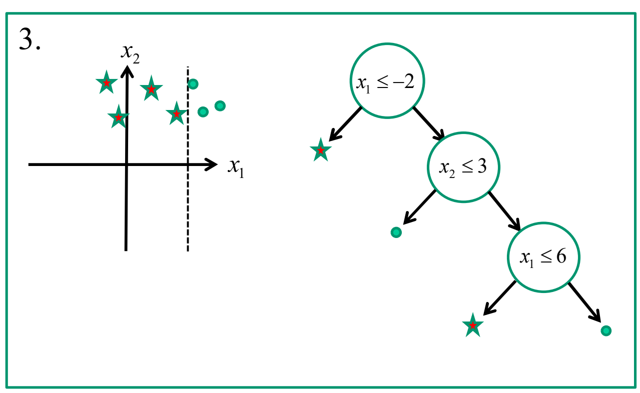


29

Arbres de décision

Exemple simple avec des classifieurs *stump*



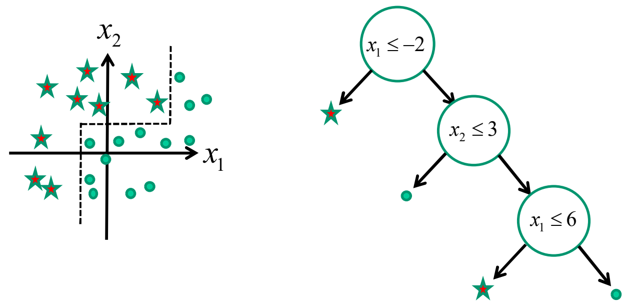


30

Arbres de décision

Exemple simple avec des classifieurs *stump*

Au final



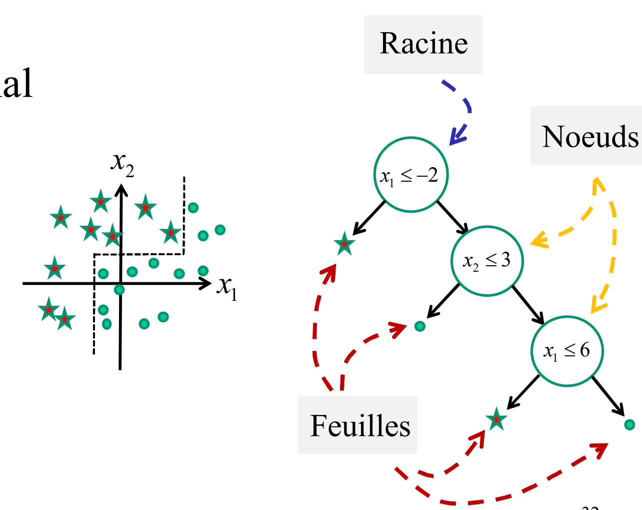
31

31

Arbres de décision

Exemple simple avec des classifieurs *stump*

Au final



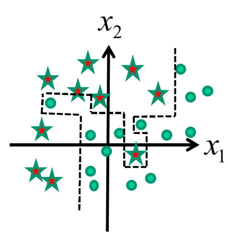
32

32

Arbres de décision

Le plus gros problème des arbres de decision est qu'ils ont tendance à **sur-apprendre**

Example avec deux données aberrantes:



Grosse question : soit le noeud d'un arbre dont l'erreur d'entraînement n'est pas nulle **devons-nous le subdiviser ou non**?

33

33

Arbres de décision

La décision de subdiviser ou non un noeud depend d'une notion d' "impureté" d'un noeud

Si l'impureté d'un nœud est **élevée** ➔ alors **on subdivise**
Si l'impureté d'un nœud est **faiblee** ➔alors **on ne subdivise pas**

Deux mesures d'impureté fréquemment utilisées

1. **L'entropie**

$$i(node) = -\sum_{j=1,2} P(c_j)\log_2(P(c_j))$$

où $P(c_j)$est la proportion de données dans la classe c_j

2. **L'indice de Gini**

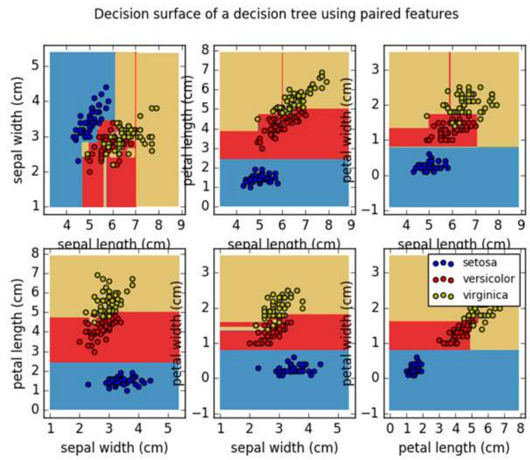
$$i(node) = 1 - \sum_{j=1,2} P^2(c_j)$$

34

34

scikit-learn

```
from sklearn.tree import DecisionTreeClassifier
(…)
cfl = DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=10,
                  min_impurity_split=0.0001)
cfl.fit(X,Y)
```



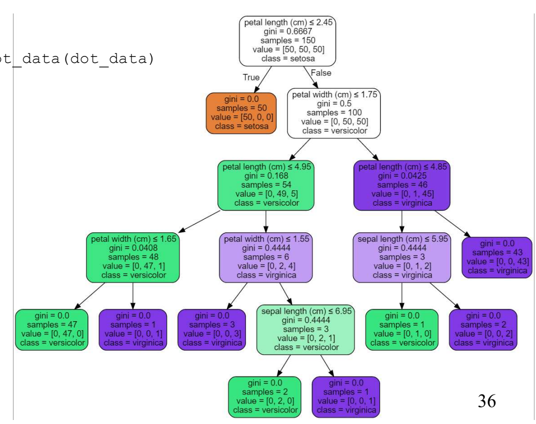
35

35

scikit-learn

```
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from IPython.display import Image
import pydotplus
(…)
cfl = DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=10,
                  min_impurity_split=0.0001)
cfl.fit(X,Y)

dot_data = export_graphviz(clf)
graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())
```

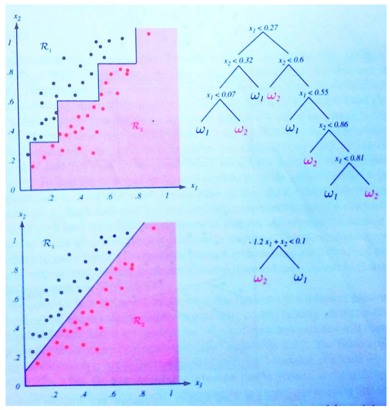


36

36

Arbres de décision

Les abres de décision peuvent également inclure des classifieurs linéaires plus généraux



Credit, Duda Hart.

J Shotton, A Fitzgibbon, M Cook, T Sharp, M Finocchio, R Moore, A Kipman, and A Blake **Real-Time Human Pose Recognition in Parts from a Single Depth Image**

37

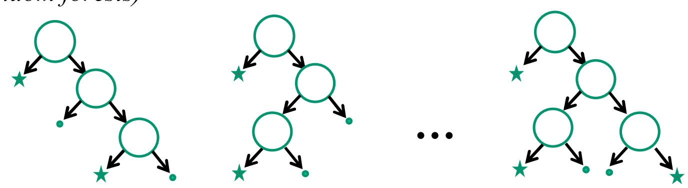
37

38

38

Forêt d'arbres décisionnels (FAD)

(Random forests)



$y_{AD,1}(\vec{x})$ $y_{AD,2}(\vec{x})$ ... $y_{AD,m}(\vec{x})$

En gros, les **FAD** ont pour but de combiner *m* **arbres de decision** à l'aide d'un **vote majoritaire**

$$y_{FAD}(\vec{x}) = sign\left(\sum_{i=1}^{m} y_{AD,i}(\vec{x})\right) \quad \text{(segmentation 2 classes)}$$

Très bonne reference!

https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm

39

39

Entraînement d'une FAD

FOR *i=1 to m*

Sélectionner au hazard N données $D_i = \{(\vec{x}_1, t_1), (\vec{x}_2, t_2) ..., (\vec{x}_N, t_N)\}, \quad \vec{x}_i \in R^d$

*Sélectionner au hazard P dimensions=> $\hat{D}_i = \{(\vec{x}_1, t_1), (\vec{x}_2, t_2) ..., (\vec{x}_N, t_N)\}, \quad \vec{x}_i \in R^p, p < d$

Entraîner un arbre de décisions $y_{AD,i}(\vec{x})$ avec $\hat{D}_i$

Bootstrapping

Généralisation

Soit une nouvelle donnée $\vec{x} \in R^d$

FOR *i=1 to m*

$C_i = y_{AD,i}(\vec{x})$

$class = VoteMajoritaire(C_1, C_2, ..., C_m)$

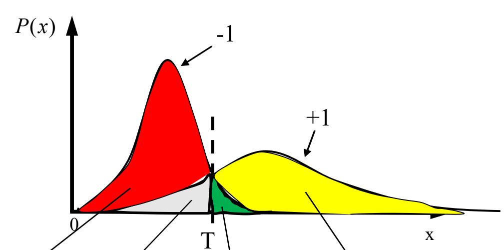
* Afin que les **erreurs** produites par les AD soient les **moins corrélées** possible.

40

Métriques d'évaluation

41

Évaluation



Taux de vrais négatifs
$\int_{-\infty}^{T} P(x\,|-1)dx$

Taux de faux négatifs
$\int_{-\infty}^{T} P(x\,|+1)dx$

Taux de faux positifs
$\int_{T}^{\infty} P(x\,|-1)dx$

Taux de vrais positifs
$\int_{T}^{\infty} P(x\,|+1)dx$

Matrice de confusion

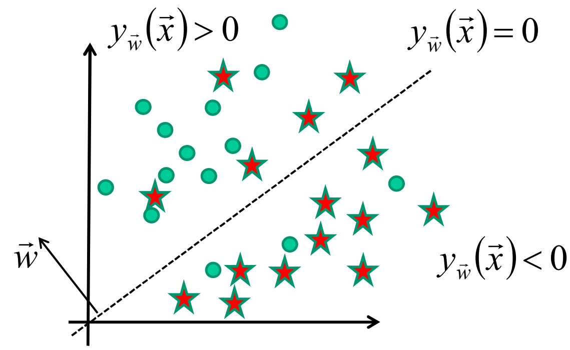
Sortie du modèle	Vérité terrain: Positifs	Vérité terrain: Négatifs
Positifs	TVP	TFP
Négatifs	TFN	TVN

TVN + TFP = 1
TVP + TFN = 1

TVN + TFP = prop des données dans la classe -1
TVP + TFN = prop des données dans la classe 1

Évaluation

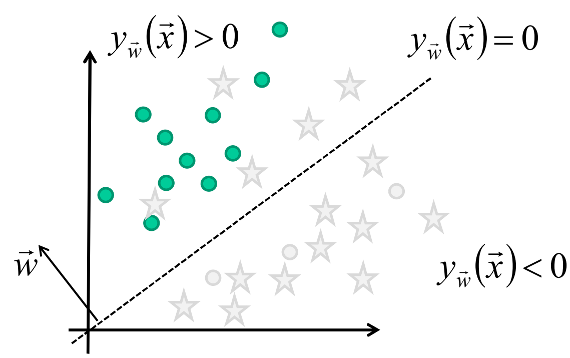
Validation empirique



43

Évaluation

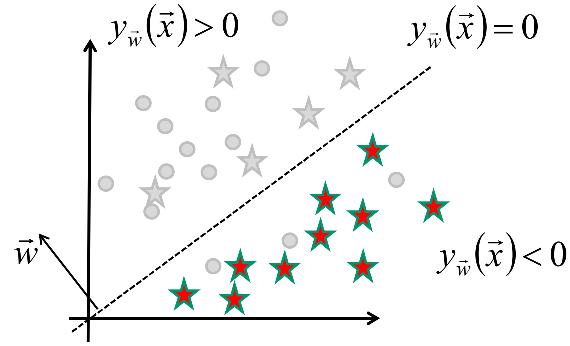
Vrais positifs



44

Évaluation

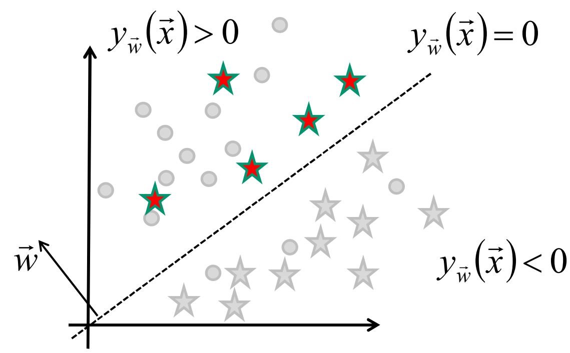
Vrais négatifs



45

Évaluation

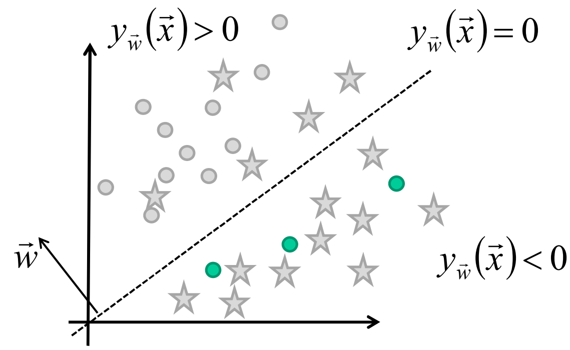
Faux positifs



46

Évaluation

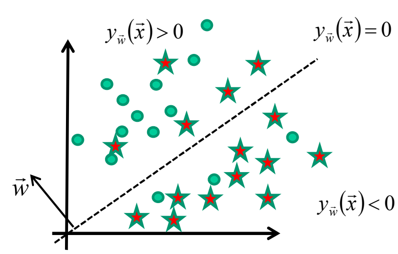
Faux négatifs



47

Évaluation

Validation empirique



On obtient une matrice de confusion **en contant les données bien et mal classés**

Sortie du modèle \ Vérité terrain	Positifs	Négatifs
Positifs	VP = 11	FP=5
Négatifs	FN=3	VN=10

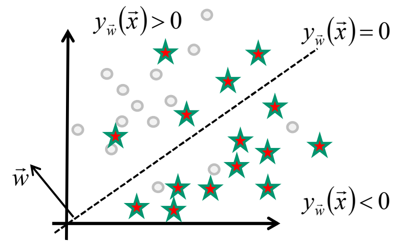
VN + FP = 15 = nombre TOTAL de négatifs
VP + FN = 14 = nombre TOTAL de positifs

VP + FP = 16 = nombre d'éléments classés +1
FN + VN = 13= nombre d'éléments classés -1

48

Évaluation

Validation empirique



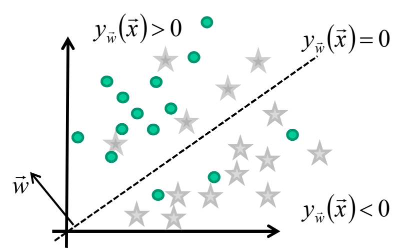
Taux de faux positifs (**TFP)** = FP/(FP+VN) = 5/15

49

49

Évaluation

Validation empirique



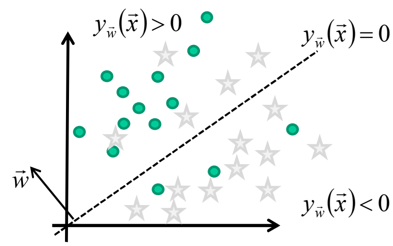
Taux de faux négatifs (**FNR**) = FN/(FN+VP) = 3/14

50

50

Évaluation

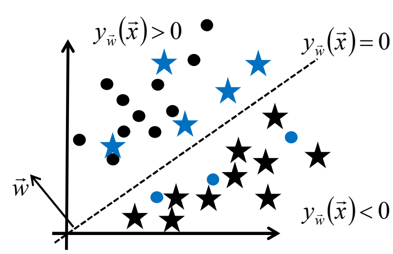
Validation empirique



Rappel (*Recall* - **Re**) = VP/(FN+VP)=1-TFN=11/14

Évaluation

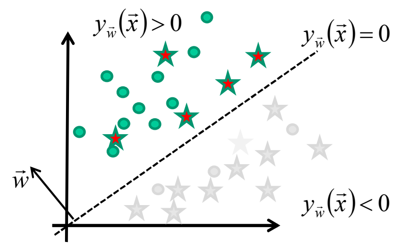
Validation empirique



Justesse (« accuracy », **prop. de biens classés**) = (TP+TN)/(FP+FN+TP+TN) = 21/29

Évaluation

Validation empirique



Précision (**Pr**) = VP/(VP+FP) = 11/16

53

53

Évaluation

Sortie du modèle	Vérité terrain: Positifs	Vérité terrain: Négatifs
Positifs	TP = 11	FP=5
Négatifs	FN=3	TN=10

VN + FP = 15 = nombre de VRAIS négatifs
VP + FN = 14 = nombre de VRAIS positifs

VP + FP = 16 = nombre d'éléments classés +1
VP + FP = 13= nombre d'éléments classés -1

Taux de faux positifs (**TFP**) = FP/(FP+VN) = 5/15
Taux de faux négatifs (**TFN**) = FN/(FN+VP) = 3/14

Spécificité (**Sp**) = VN/(FP+VN)=1-FPR=10/15
Rappel (*recall*-**Re**) = VP/(VP+VN)=11/14
Précision (**Pr**) = VP/(VP+FP) = 11/16

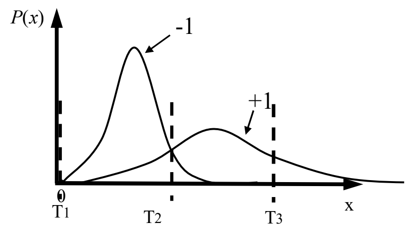
Justesse (*Accuracy*, **prop. de biens classés**) = (VP+VN)/(FP+FN+VP+VN) = 21/29
F-measure = 2* (Re*Pr)/(Pr+Re)=0.73

54

54

Évaluation

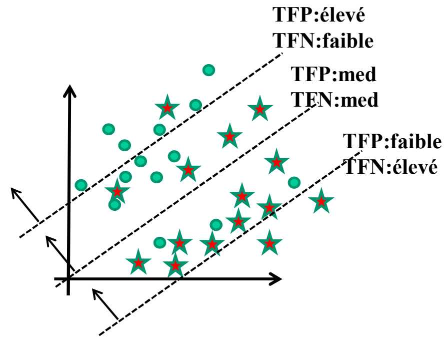
Différents seuils = différents résultats



TFP:élevé
TFN:faible

TFP:med
TFN:med

TFP:faible
TFN:élevé

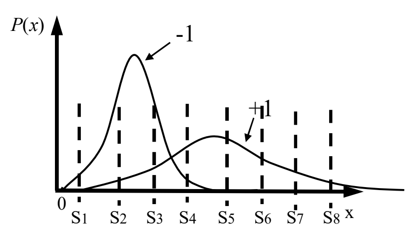


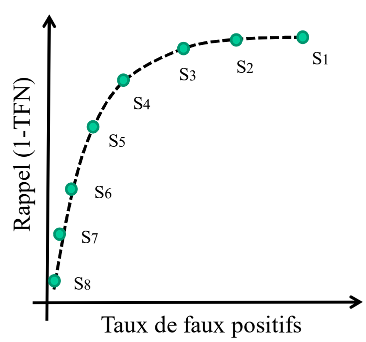
55

55

Évaluation

Courbe ROC : enregistrer le TFP et le TVP pour **différents seuils**





56

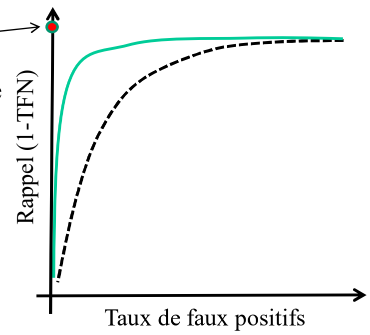
56

Validation

Courbes ROC: très bonne façon de comparer des méthodes

Quelle méthode est la meilleure?

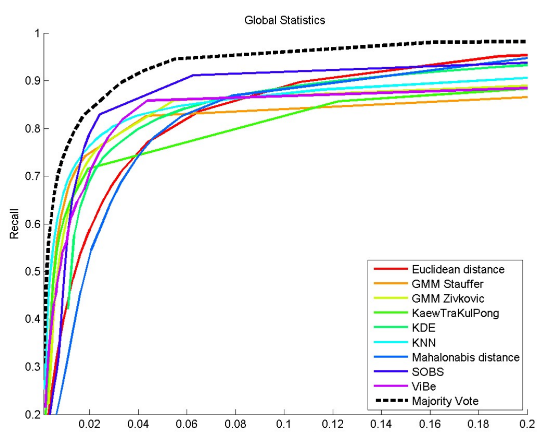
Objectif ultime
La meilleure méthode
est celle avec
TFP=TFN=0



57

Évaluation

Exemple : 10 méthodes de détection de mouvement

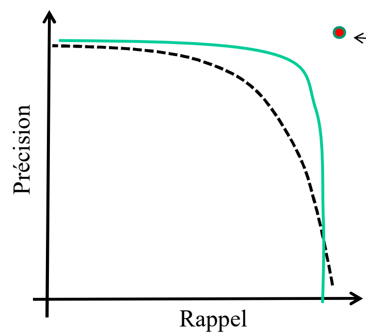


58

Évaluation

Courbe de précision-rappel : similaire à la courbe ROC

Qui est le meilleur?



Bull's eye
Pr=Re=1

59