

Techniques d'apprentissage  
IFT 603-712

Classification linéaire  
Par  
Pierre-Marc Jodoin  
/  
Hugo Larochelle

1

---

---

---

---

---

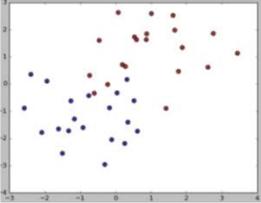
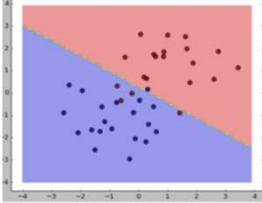
---

---

---

### Classification supervisée (illustrée)

Entraînement

Soient des données de 2 classes ● et ● (ici dans un espace 2D)

Le but est de trouver une fonction  $y_w(\vec{x})$  telle que

$y_w(\bullet) = \text{classe 1}$   
 $y_w(\bullet) = \text{classe 2}$

2

2

---

---

---

---

---

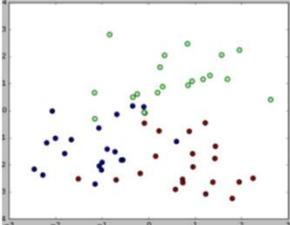
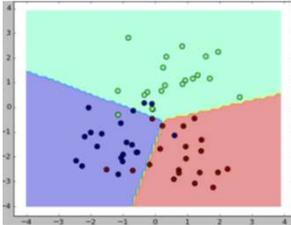
---

---

---

### Classification supervisée (illustrée)

Entraînement avec plus de 2 classes

Soient des données de 3 classes ●, ● et ● (ici dans un espace 2D)

Le but est de trouver une fonction  $y_w(\vec{x})$  telle que

$y_w(\bullet) = \text{classe 1}$   
 $y_w(\bullet) = \text{classe 2}$   
 $y_w(\bullet) = \text{classe 3}$

3

3

---

---

---

---

---

---

---

---

## Notation

**Ensemble d'entraînement:**  $D = \{(\bar{x}_1, t_1), (\bar{x}_2, t_2), \dots, (\bar{x}_N, t_N)\}$

$\bar{x}_i \in \mathfrak{R}^d$  vecteur de données du n-ème élément

$t_i \in \{c_1, c_2, \dots, c_k\}$  étiquette de classe du i-ème élément

**Fonctions:** avec  $D$ , on doit *apprendre* une **fonction de classification**

$$y_{\hat{w}}(\bar{x}): \mathfrak{R}^d \rightarrow \{c_1, c_2, \dots, c_k\}$$

qui nous informe à quelle classe appartient le vecteur  $\bar{x}$ .

5

5

---

---

---

---

---

---

---

---

## Au menu : 5 méthodes



Régression  
Modèles génératifs  
Discriminant de Fisher

} Émettent l'**hypothèse** que les données sont **gaussiennes**  
Solution de type « **closed form** » (inversion de matrice)

Perceptron  
Régression logistique

} **Aucune hypothèse** quant à la distribution des données  
Solution obtenue grâce à une **descente de gradient**.

6

6

---

---

---

---

---

---

---

---

## Introduction à la classification linéaire

**Au tableau !!!**

7

7

---

---

---

---

---

---

---

---

### Séparation linéaire

(2D et 2 classes)

$$y_w(\vec{x}) = w_0 + w_1x_1 + w_2x_2$$

biais      poids  
 $\downarrow$                        $\swarrow$

$$= w_0 + \vec{w}^T \vec{x}$$

$$= \vec{w}'^T \vec{x}'$$

$y_w(\vec{x}) = \vec{w}'^T \vec{x}'$

Par simplicité

2 grands **avantages**. Une fois l'entraînement terminé,

1. Plus besoin de données d'entraînement
2. Classification est très rapide (**produit scalaire** entre 2 vecteurs)

8

---

---

---

---

---

---

---

---

---

---

8

### Séparation linéaire

Exemple jouet

$$\vec{w}'^T = (2.2, -5.5, 4.4)$$

$\vec{x}_a$  est en FACE du plan

$$y_w(\vec{x}_a) = \vec{w}'^T \vec{x}_a = (2.2, -5.5, 4.4) \begin{bmatrix} 1 \\ 3 \\ 7 \end{bmatrix} = 16.5$$

$$y_w(\vec{x}_b) = \vec{w}'^T \vec{x}_b = (2.2, -5.5, 4.4) \begin{bmatrix} 1 \\ 6 \\ 1 \end{bmatrix} = -26.4$$

$\vec{x}_b$  est DERRIÈRE le plan

9

---

---

---

---

---

---

---

---

---

---

9

10

---

---

---

---

---

---

---

---

---

---

10

Régression	}	Émettent l'hypothèse que les données sont <b>gaussiennes</b> Solution de type « <i>closed form</i> » (inversion de matrice)
Modèles génératifs		
Discriminant de Fisher		
Perceptron	}	Émettent <b>aucune hypothèse</b> quant à la distribution des données Solution obtenue grâce à une <b>descente de gradient</b> .
Régression logistique		

11

11

---

---

---

---

---

---

---

---

## Régression par les moindres carrés

(section 4.1.3, Bishop)

12

12

---

---

---

---

---

---

---

---

## Régression par les moindres carrés

Cas 2 classes

On peut **classifier des données** en utilisant une approche de **régression** comme celle vue au chapitre précédent.

- On pourrait **prédire directement** la valeur de la cible ( $t=1.0$  vs  $t=-1.0$ )
- Si  $y_{\hat{w}}(\bar{x}) \geq 0$  on classe dans *Classe1* sinon dans *Classe2*

13

13

---

---

---

---

---

---

---

---

### Régression par moindres carrés **RAPPEL**

On a vu qu'on peut utiliser une fonction d'erreur **par moindres carrés**

**Maximum de vraisemblance**  $\tilde{w} = \arg \min_w \underbrace{\sum_{n=1}^N (t_n - y_w(\tilde{x}_n))^2}_{E_D(\tilde{w})}$

**Maximum a posteriori**  $\tilde{w} = \arg \min_w \underbrace{\sum_{n=1}^N (t_n - y_w(\tilde{x}_n))^2 + \lambda \tilde{w}^T \tilde{w}}_{E_D(\tilde{w})}$

**On peut prendre la même approche pour la classification**

14

---

---

---

---

---

---

---

---

14

### Régression par les moindres carrés **RAPPEL**

Cas 2 classes

**Maximum de vraisemblance**  $\tilde{w} = \arg \min_w \sum_{n=1}^N (t_n - y_w(\tilde{x}_n))^2$

$$\tilde{w}_{MV} = (X^T X)^{-1} X^T T$$

**Maximum a posteriori**  $\tilde{w} = \arg \min_w \sum_{n=1}^N (t_n - y_w(\tilde{x}_n))^2 + \lambda \tilde{w}^T \tilde{w}$

$$\tilde{w}_{MAP} = (X^T X + \lambda I)^{-1} X^T T$$

**Ces fonctions de coût s'appuient sur l'hypothèse de données gaussiennes**

15

---

---

---

---

---

---

---

---

15

### Régression par moindres carrés

- Régression logistique (à voir plus loin)
- Moindres carrés (maximum de vraisemblance)

Données gaussiennes

Données Non-gaussiennes

16

---

---

---

---

---

---

---

---

16

## Régression par les moindres carrés

Cas  $K > 2$  classes

On va traiter le cas  $K$  classes comme une **régression multiple**

- **Cible** : vecteur à  $K$  dim. indiquant a quelle classe appartient l'entrée
- **Exemple** : Pour  $K=5$  classes et une entrée associée à la classe 2

$$t_n = (-1 \quad 1 \quad -1 \quad -1 \quad -1)^T$$

- **Classification**: On classifie dans la classe  $k$  une donnée dont la valeur de  $y_{\tilde{w}_k}(\vec{x})$  est la plus élevée.

17

17

---

---

---

---

---

---

---

---

---

---

## Régression par les moindres carrés

Cas  $K > 2$  classes

Le modèle doit maintenant **prédire un vecteur**

$$y_W(\vec{x}) = W^T \vec{x}$$

où  $W$  est une matrice  $K \times d$

Chaque ligne de  $W$  peut être vue comme un vecteur  $\tilde{w}_k$  du modèle  $y_{\tilde{w}_k}(\vec{x}) = \tilde{w}_k^T \vec{x}$  pour la  $k^e$  cible

18

18

---

---

---

---

---

---

---

---

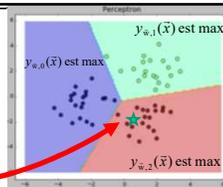
---

---

## Cas $K=3$ classes

Exemple

★ (1.1, -2.0)



$$y_W(\vec{x}) = W\vec{x} = \begin{bmatrix} -.6 & -.49 & .4 \\ -.4 & .41 & .24 \\ -.2 & .05 & -.36 \end{bmatrix} \begin{bmatrix} 1 \\ 1.1 \\ -2.0 \end{bmatrix} = \begin{bmatrix} -1.9 \\ -.43 \\ .58 \end{bmatrix} \begin{matrix} \text{Classe 0} \\ \text{Classe 1} \\ \text{Classe 2} \end{matrix}$$

19

19

---

---

---

---

---

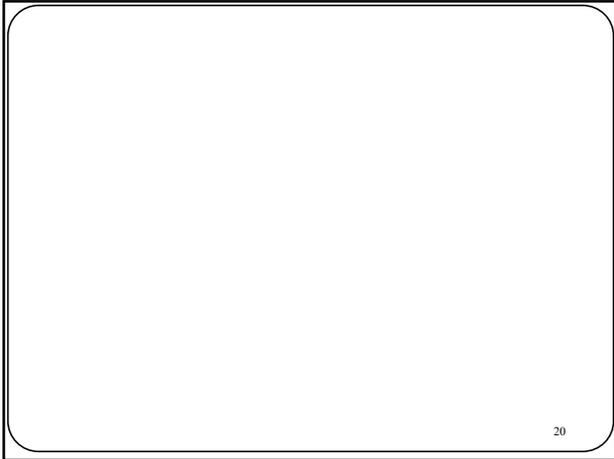
---

---

---

---

---



20

---

---

---

---

---

---

---

---

Régression	}	Émettent l'hypothèse que les données sont <b>gaussiennes</b> Solution de type « <i>closed form</i> » (inversion de matrice)
<b>Modèles génératifs</b>		
Discriminant de Fisher		
Perceptron	}	Émettent aucune hypothèse quant à la distribution des données Solution obtenue grâce à une <b>descente de gradient</b> .
Régression logistique		

21

---

---

---

---

---

---

---

---

**Modèles probabilistes génératifs**  
(section 4.2, Bishop)

22

---

---

---

---

---

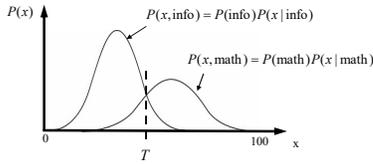
---

---

---

## Prenons le cas 1D, 2 Classes

Ex: examen de statistiques avec des étudiants en math et en informatique



T est le seuil qui **minimise l'erreur de classification**

$$P(\text{info})P(x = T | \text{info}) = P(\text{math})P(x = T | \text{math})$$

$$P(\text{info})P(x | \text{info}) \underset{\text{math}}{\overset{\text{info}}{\gtrless}} P(\text{math})P(x | \text{math})$$

23

23

---

---

---

---

---

---

---

---

---

---

**Take Note**

$$P(\text{info})P(x | \text{info}) \underset{\text{math}}{\overset{\text{info}}{\gtrless}} P(\text{math})P(x | \text{math})$$

est équivalent à un **maximum a posteriori**

$$t = \arg \max_t P(t | x) \quad \text{où } t \in \{\text{math}, \text{info}\}$$

Inconnue      Connue

$$= (\dots)$$

$$= \arg \max_t P(t)P(x | t)$$

24

24

---

---

---

---

---

---

---

---

---

---

## Prenons le cas 1D, 2 Classes

Ex: examen de math avec étudiant en math et en informatique

$$P(\text{info})P(x | \text{info}) \underset{\text{math}}{\overset{\text{info}}{\gtrless}} P(\text{math})P(x | \text{math})$$

Si on suppose que la **vraisemblance** de chaque classe est **gaussienne**:

$$P(x | \text{info}) = \frac{1}{\sqrt{2\pi}\sigma_{\text{info}}} \exp\left(-\frac{(x - \mu_{\text{info}})^2}{2\sigma_{\text{info}}^2}\right)$$

$$P(x | \text{math}) = \frac{1}{\sqrt{2\pi}\sigma_{\text{math}}} \exp\left(-\frac{(x - \mu_{\text{math}})^2}{2\sigma_{\text{math}}^2}\right)$$

où

$\mu_{\text{math}}$  : moyenne des étudiants de math .

$\sigma_{\text{math}}$  : écart - type des étudiants de math.

25

25

---

---

---

---

---

---

---

---

---

---

## Prenons le cas 1D, 2 Classes

Ex: examen de math avec étudiant en math et en informatique

$$P(\text{info})P(x | \text{info}) \underset{\text{math}}{\gtrsim} P(\text{math})P(x | \text{math})$$

Si on suppose que la **vraisemblance** de chaque classe est **gaussienne**:

$$P(x | \text{info}) = \frac{1}{\sqrt{2\pi}\sigma_{\text{info}}} \exp\left(-\frac{(x - \mu_{\text{info}})^2}{2\sigma_{\text{info}}^2}\right)$$

$$P(x | \text{math}) = \frac{1}{\sqrt{2\pi}\sigma_{\text{math}}} \exp\left(-\frac{(x - \mu_{\text{math}})^2}{2\sigma_{\text{math}}^2}\right)$$

et que

$$P(\text{info}) = \frac{\text{nb étudiants info}}{\text{nb tot étudiants}} \quad (\text{Proportion des étudiants en info})$$

$$P(\text{math}) = \frac{\text{nb étudiants math}}{\text{nb tot étudiants}} \quad (\text{Proportion des étudiants en math})$$

26

26

---

---

---

---

---

---

---

---

---

---

## Modèle probabiliste génératif

Algorithme du seuil « optimal »

$$\mu_{\text{info}} = \frac{1}{N_{\text{info}}} \sum_{i \in \text{info}} x_i, \quad \mu_{\text{math}} = \frac{1}{N_{\text{math}}} \sum_{i \in \text{math}} x_i$$

$$\sigma_{\text{info}}^2 = \frac{1}{N_{\text{info}}} \sum_{i \in \text{info}} (x_i - \mu_{\text{info}})^2, \quad \sigma_{\text{math}}^2 = \frac{1}{N_{\text{math}}} \sum_{i \in \text{math}} (x_i - \mu_{\text{math}})^2$$

$$P(\text{math}) = \frac{N_{\text{math}}}{N_{\text{info}} + N_{\text{math}}}, \quad P(\text{info}) = \frac{N_{\text{info}}}{N_{\text{info}} + N_{\text{math}}}$$

POUR CHAQUE note  $x$  FAIRE

$$P_i = \frac{P(\text{info})}{\sqrt{2\pi}\sigma_{\text{info}}} \exp\left(-\frac{(x - \mu_{\text{info}})^2}{2\sigma_{\text{info}}^2}\right)$$

$$P_m = \frac{P(\text{math})}{\sqrt{2\pi}\sigma_{\text{math}}} \exp\left(-\frac{(x - \mu_{\text{math}})^2}{2\sigma_{\text{math}}^2}\right)$$

SI  $P_i > P_m$  ALORS

$t = 1$  /\* étudiant « info » \*/

SINON

$t = 0$  /\* étudiant « math » \*/

27

27

---

---

---

---

---

---

---

---

---

---

L'algorithme de la page précédente  
revient à un **classificateur quadratique**

$$y_{\vec{w}}(x) = w_2 x^2 + w_1 x + w_0 = 0$$

28

28

---

---

---

---

---

---

---

---

---

---





Tel que mentionné au chapitre 4.2.2, lorsque les 2 classes n'ont pas la même variance-covariance, on peut utiliser le modèle linéaire mais avec la matrice

$$\Sigma = P(C_1)\Sigma_1 + P(C_2)\Sigma_2$$

32

32

---

---

---

---

---

---

---

---

## Modèle probabiliste génératif

Classificateur linéaire, cas 1D, 2 Classes

Si on suppose que  $P(\text{info}) = P(\text{math})$

$$\frac{P(\text{info})}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu_{\text{info}})^2}{2\sigma^2}\right) = \frac{P(\text{math})}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu_{\text{math}})^2}{2\sigma^2}\right)$$

$$y_{\bar{w}}(x) = w_1 x + w_0 = 0$$

$$w_1 = \frac{(\mu_{\text{math}} - \mu_{\text{info}})}{\sigma^2}$$

$$w_0 = \frac{\mu_{\text{info}}^2}{2\sigma^2} - \frac{\mu_{\text{math}}^2}{2\sigma^2} - \ln\left(\frac{P(\text{info})}{P(\text{math})}\right)$$

33

33

---

---

---

---

---

---

---

---

## Modèle probabiliste génératif

Classificateur linéaire, cas d-D, K Classes

On peut généraliser au cas à **plusieurs classes**

➤ Voir fin des sections 4.2 et 4.2.1

34

34

---

---

---

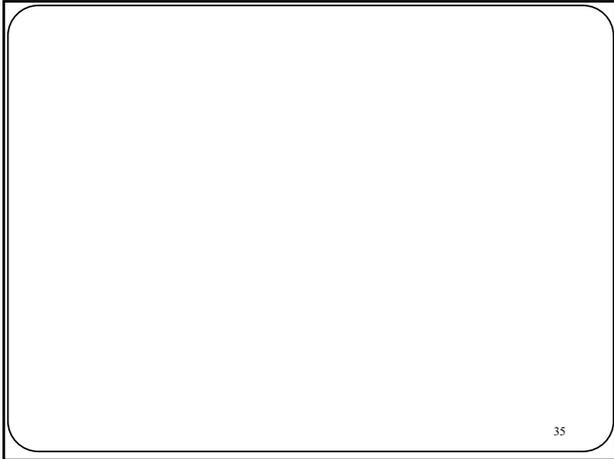
---

---

---

---

---



35

---

---

---

---

---

---

---

---

Régression	}	Émettent l'hypothèse que les données sont <b>gaussiennes</b> Solution de type « <i>closed form</i> » (inversion de matrice)
Modèles génératifs		
Discriminant de Fisher		
Perceptron	}	Émettent <b>aucune hypothèse</b> quant à la distribution des données Solution obtenue grâce à une <b>descente de gradient</b> .
Régression logistique		

36

---

---

---

---

---

---

---

---

**Discriminant linéaire de Fisher**  
(section 4.1.4, Bishop)

37

---

---

---

---

---

---

---

---

### Classification linéaire = Projection 1D

(2D et 2 classes)

$$y_w(\vec{x}) = w_0 + \vec{w}^T \vec{x}$$

biais poids

Tel que vu en classe, une classification linéaire revient à une projection vers un espace 1D.

38

38

---

---

---

---

---

---

---

---

---

---

### Classification linéaire = Projection 1D

(2D et 2 classes)

$$y_w(\vec{x}) = w_0 + \vec{w}^T \vec{x}$$

Hypothèse gaussienne...

$$\hat{\mu}_1 = \vec{w}^T \mu_1 + w_0$$

$$\hat{\mu}_2 = \vec{w}^T \mu_2 + w_0$$

39

39

---

---

---

---

---

---

---

---

---

---

### Classification linéaire = Projection 1D

(2D et 2 classes)

$$y_w(\vec{x}) = w_0 + \vec{w}^T \vec{x}$$

Intuitivement, une bonne solution  $y_w(\vec{x})$  en est une pour laquelle la distance entre les moyennes projetées est grande.

$$\vec{w} = \arg \max_w |\hat{\mu}_1 - \hat{\mu}_2|$$

$$= \arg \max_w |\vec{w}^T \mu_1 + w_0 - \vec{w}^T \mu_2 - w_0|$$

$$= \arg \max_w |\vec{w}^T (\mu_1 - \mu_2)|$$

40

40

---

---

---

---

---

---

---

---

---

---

### Classification linéaire = Projection 1D

(2D et 2 classes)

$$y_{\vec{w}}(\vec{x}) = w_0 + \vec{w}^T \vec{x}$$


$$\vec{w} = \arg \max_{\vec{w}} |\vec{w}^T (\vec{\mu}_1 - \vec{\mu}_2)|$$

Ce **problème est mal posé** car il suffit d'augmenter **W** infiniment pour maximiser cette fonction.

41

41

---

---

---

---

---

---

---

---

### Classification linéaire = Projection 1D

(2D et 2 classes)

$$y_{\vec{w}}(\vec{x}) = w_0 + \vec{w}^T \vec{x}$$


$$\vec{w} = \arg \max_{\vec{w}} |\vec{w}^T (\vec{\mu}_1 - \vec{\mu}_2)|$$

Par contre si on impose que la **norme de  $w = 1$**  on obtient que

$$\vec{w} \propto (\vec{\mu}_1 - \vec{\mu}_2)$$

(preuve au tableau)

42

42

---

---

---

---

---

---

---

---

### Discriminant linéaire

Une fois  $w$  calculé, il faut trouver le biais  $w_0$

- > Un choix fréquent lorsque les classes sont équilibrées

$$w_0 = -\frac{\vec{w}^T \vec{\mu}_1 + \vec{w}^T \vec{\mu}_2}{2}$$

- > Sinon

$$w_0 = -\vec{w}^T \left( \frac{N_1}{N_1 + N_2} \vec{\mu}_1 + \frac{N_2}{N_1 + N_2} \vec{\mu}_2 \right)$$

où  $N_1$  et  $N_2$  sont le nombre d'éléments dans chaque classe.

43

43

---

---

---

---

---

---

---

---

### Discriminant linéaire

(2D et 2 classes)

Les 2 gaussiennes projetées:

$$\hat{\mu}_1 = \bar{w}^T \mu_1 + w_0 \quad \hat{\mu}_2 = \bar{w}^T \mu_2 + w_0$$

$$\hat{\sigma}_1^2 = \bar{w}^T \Sigma_1 \bar{w} \quad \hat{\sigma}_2^2 = \bar{w}^T \Sigma_2 \bar{w}$$

44

44

---

---

---

---

---

---

---

---

### Discriminant linéaire de Fisher

(2D et 2 classes)

**Critère de Fisher**

$$\bar{w} = \arg \max_w \frac{(\hat{\mu}_1 - \hat{\mu}_2)^2}{\hat{\sigma}_1^2 + \hat{\sigma}_2^2}$$

On obtient le meilleur  $\bar{w}$  en forçant le gradient à 0

$$\nabla_{\bar{w}} \frac{(\hat{\mu}_1 - \hat{\mu}_2)^2}{\hat{\sigma}_1^2 + \hat{\sigma}_2^2} = 0$$

45

45

---

---

---

---

---

---

---

---

### Discriminant linéaire de Fisher

(2D et 2 classes)

**Critère de Fisher**

$$\bar{w} = \arg \max_w \frac{(\hat{\mu}_1 - \hat{\mu}_2)^2}{\hat{\sigma}_1^2 + \hat{\sigma}_2^2}$$

On obtient le meilleur  $\bar{w}$  en forçant le gradient à 0

$$\nabla_{\bar{w}} \frac{(\hat{\mu}_1 - \hat{\mu}_2)^2}{\hat{\sigma}_1^2 + \hat{\sigma}_2^2} = 0$$

46

46

---

---

---

---

---

---

---

---

## Discriminant linéaire de Fisher

(2D et 2 classes)

**Critère de Fisher**

$$\bar{w} = \arg \max_{\bar{w}} \frac{(\bar{\mu}_1 - \bar{\mu}_2)^2}{\bar{\sigma}_1^2 + \bar{\sigma}_2^2}$$

Preuve en classe ou en devoir

$$\Rightarrow \bar{w} \propto \Sigma_{\bar{w}}^{-1} (\bar{\mu}_1 - \bar{\mu}_2)$$

$$\Rightarrow \Sigma_{\bar{w}} = \sum_{i_n=C_1} (\bar{x}_n - \bar{\mu}_1)(\bar{x}_n - \bar{\mu}_1)^T + \sum_{i_n=C_2} (\bar{x}_n - \bar{\mu}_2)(\bar{x}_n - \bar{\mu}_2)^T$$

47

---

---

---

---

---

---

---

---

---

---

47

## Discriminant linéaire de Fisher

(2D et 2 classes)

Sans minimisation  
intra-classe

$$\bar{w} \propto (\bar{\mu}_1 - \bar{\mu}_2)$$

Avec minimisation  
intra-classe

$$\bar{w} \propto \Sigma_{\bar{w}}^{-1} (\bar{\mu}_1 - \bar{\mu}_2)$$

48

---

---

---

---

---

---

---

---

---

---

48

## Discriminant linéaire de Fisher

**Algorithme 2-Classes, entraînement**

Calculer  $\bar{\mu}_1, \bar{\mu}_2$

$$\Sigma_{\bar{w}} = \sum_{i_n=C_1} (\bar{x}_n - \bar{\mu}_1)(\bar{x}_n - \bar{\mu}_1)^T + \sum_{i_n=C_2} (\bar{x}_n - \bar{\mu}_2)(\bar{x}_n - \bar{\mu}_2)^T$$

$$\bar{w} = \Sigma_{\bar{w}}^{-1} (\bar{\mu}_1 - \bar{\mu}_2)$$

$$w_0 = -\frac{\bar{w}^T \bar{\mu}_1 + \bar{w}^T \bar{\mu}_2}{2} \quad \left( \text{ou } w_0 = -\bar{w}^T \left( \frac{N_1}{N_1 + N_2} \bar{\mu}_1 + \frac{N_2}{N_1 + N_2} \bar{\mu}_2 \right) \right)$$

**Algorithme 2-Classes, généralisation**

POUR CHAQUE donnée test  $\bar{x}$  FAIRE

$t = y_{\bar{w}}(\bar{x}) = \bar{w}^T \bar{x} + w_0$

SI  $t < 0$  ALORS

$t = 1$

SINON

$t = 2$

49

---

---

---

---

---

---

---

---

---

---

49

## Discriminant linéaire de Fisher

- On peut voir l'analyse discriminante linéaire comme un cas particulier des **moindres carrés**
  - voir section 4.1.5
- Il est possible de généraliser au cas à **plus de 2 classes**
  - voir section 4.1.6

50

---

---

---

---

---

---

---

---

50

51

---

---

---

---

---

---

---

---

51

<p>Régression Modèles génératifs Discriminant de Fisher</p>	}	<p>Émettent l'hypothèse que les données sont <b>gaussiennes</b> Solution de type « <i>closed form</i> » (inversion de matrice)</p>
<p>Perceptron Régression logistique</p>	}	<p>Émettent <b>aucune hypothèse</b> quant à la distribution des données Solution obtenue grâce à une <b>descente de gradient</b>.</p>

52

---

---

---

---

---

---

---

---

52

# Perceptron

(section 4.1.7, Bishop)

53

---

---

---

---

---

---

---

---

53

## Perceptron (2 classes)

Contrairement aux approches précédentes, le perceptron **n'émet pas** l'hypothèse que les données sont **gaussiennes**

Le perceptron part de la définition brute de la classification binaire par **hyperplan**

$$y_w(\vec{x}) = \text{sign}(\vec{w}^T \vec{x})$$

$$= \text{sign}(w_0 + w_1 x_1 + w_2 x_2 + \dots + w_d x_d)$$

↑ **biais**      ↑ **poids**

54

---

---

---

---

---

---

---

---

54

## Perceptron

(2D et 2 classes)

$$y_w(\vec{x}) = w_0 + w_1 x_1 + w_2 x_2$$

$$= \vec{w}^T \vec{x}$$

$$= \vec{w}'^T \vec{x}'$$

$$\Rightarrow \vec{w}^T \vec{x}$$

55

---

---

---

---

---

---

---

---

55

### Perceptron (2D et 2 classes)

$y_w(\vec{x}) = \text{sign}(\vec{w}^T \vec{x})$

56

---

---

---

---

---

---

---

---

### Perceptron (2D et 2 classes)

**Neurone**  
Produit scalaire + fonction d'activation

57

---

---

---

---

---

---

---

---

### Perceptron (N-D and 2-class case)

**Example 3D**

$y_w(\vec{x}) = +1$

$y_w(\vec{x}) = -1$

58

---

---

---

---

---

---

---

---

## Nouvelle fonction de coût pour **apprendre $W$**

**Le but:** avec des données d'entraînement  $D = \{(\vec{x}_1, t_1), (\vec{x}_2, t_2), \dots, (\vec{x}_N, t_N)\}$ , estimer  $\vec{w}$  afin que:

$$y_{\vec{w}}(\vec{x}_n) = t_n \quad \forall n$$

En d'autres mots, minimiser l'**erreur d'entraînement**

$$E_D(\vec{w}) = \frac{1}{N} \sum_{n=1}^N l(y_{\vec{w}}(\vec{x}_n), t_n)$$

où  $l(\dots)$  est une **fonction de perte** (*loss function* en anglais).

Trouver la bonne fonction de perte et le bon algorithme d'**optimisation** est un sujet central en **apprentissage machine**.

59

59

---

---

---

---

---

---

---

---

---

---

## Régression et classification

**RAPPEL**

Vous vous souvenez de la **régression**?

Maximum de vraisemblance

$$\vec{w} = \arg \min_{\vec{w}} \underbrace{\sum_{n=1}^N \frac{(t_n - y_{\vec{w}}(\vec{x}_n))^2}{2}}_{E_D(\vec{w})}$$

Maximum *a posteriori*

$$\vec{w} = \arg \min_{\vec{w}} \underbrace{\sum_{n=1}^N (t_n - y_{\vec{w}}(\vec{x}_n))^2}_{E_D(\vec{w})} + \lambda \vec{w}^T \vec{w}$$



C'est un peu la même idée pour le Perceptron mais avec une **nouvelle fonction de coût**.

60

60

---

---

---

---

---

---

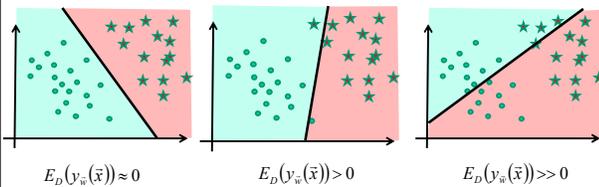
---

---

---

---

## Fonction de perte



61

---

---

---

---

---

---

---

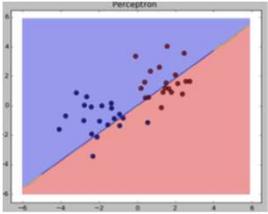
---

---

---

### Nouvelle fonction de coût pour **apprendre $W$**

Une fonction simple et indépendante de la distribution des données serait de compter 1 pour chaque donnée mal classée et 0 sinon

$$E_D(\vec{w}) = \sum_{z \in M} 1 \quad \text{où } M \text{ est l'ensemble des données mal classées}$$


Exemple:  
 $E_D(\vec{w}) = 15$

Ainsi, la meilleure solution serait celle pour laquelle on **aurait aucune donnée mal classée**.  
 Malheureusement, cette fonction n'est **pas dérivable** partout et  $\nabla_{\vec{w}} E_D(\vec{w}) = 0$  pour des **solutions non-optimales**

62

---

---

---

---

---

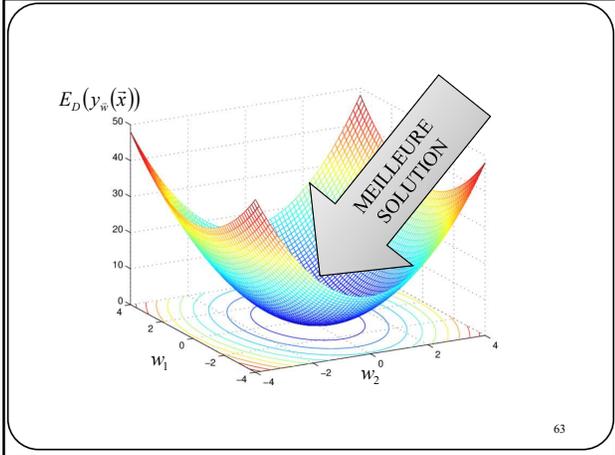
---

---

---

---

---



63

---

---

---

---

---

---

---

---

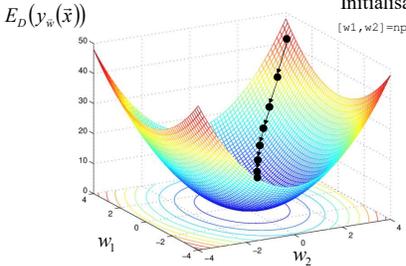
---

---

### Perceptron

Question: comment trouver la meilleure solution?  $\nabla_{\vec{w}} E_D(y_{\vec{w}}(\vec{x})) = 0$

Initialisation aléatoire  
`[w1, w2] = np.random.randn(2)`



64

---

---

---

---

---

---

---

---

---

---

## Gradient descent

Question: how to find the best solution?  $\nabla E_D(y_{\vec{w}}(\vec{x})) = 0$

$$\vec{w}^{[k+1]} = \vec{w}^{[k]} - \eta \nabla E_D(y_{\vec{w}^{[k]}}(\vec{x}))$$

↙ Taux d'apprentissage  
(Learning rate)  
↘ Gradient de la perte

65

65

---

---

---

---

---

---

---

---

## Critère du perceptron (perte)

### Observation

Une donnée mal classée survient lorsque

$$\vec{w}^T \vec{x}_n > 0 \text{ et } t_n = -1$$

ou

$$\vec{w}^T \vec{x}_n < 0 \text{ et } t_n = +1.$$

DONC  $-\vec{w}^T \vec{x}_n t_n$  est TOUJOURS positif pour des données mal classées

66

66

---

---

---

---

---

---

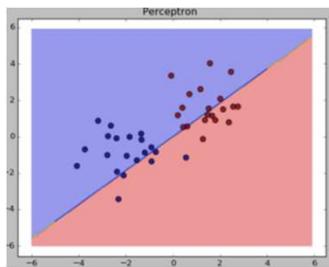
---

---

## Critère du perceptron

Le critère du perceptron est une fonction qui pénalise les données mal classées

$$E_D(\vec{w}) = \sum_{\vec{x}_n \in M} -\vec{w}^T \vec{x}_n t_n \quad \text{où } M \text{ est l'ensemble des données mal classées}$$



$$E_D(\vec{w}) = 464.15$$

67

67

---

---

---

---

---

---

---

---

# Perceptron

**Question:** comment trouver la meilleure solution  $\vec{w}$  avec cette fonction de perte?

**Réponse:** une solution frequente est la **descente de gradient**.

$$\vec{w}^{(k+1)} = \vec{w}^{(k)} - \eta \nabla E_D(\vec{w}^{(k)})$$

$\swarrow$  Gradient de la fonction de coût  
 $\searrow$  Taux d'apprentissage (*learning rate*).

## Descente de gradient de base

Initialiser  $\vec{w}$   
 $k=0$   
 FAIRE  $k=k+1$   
 $\vec{w} = \vec{w} - \eta \nabla E_D(\vec{w})$   
 JUSQU'À ce que toutes les données soient bien classées

68

68

---

---

---

---

---

---

---

---

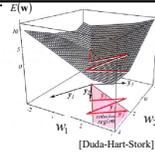
---

---

# Perceptron

Pour le critère du Perceptron

$$\nabla E_D(\vec{w}) = \sum_{\vec{x}_n \in M} -t_n \vec{x}_n$$



## Batch optimization

Initialiser  $\vec{w}$   
 $k=0$   
 DO  $k=k+1$   
 $\vec{w} = \vec{w} - \eta \left( \sum_{\vec{x}_n \in M} -t_n \vec{x}_n \right)$   
 UNTIL toutes les données sont bien classées

NOTE importante sur le **taux d'apprentissage  $\eta$**

- **Trop faible** => convergence lente
- **Trop grand** => peut ne pas converger (et même diverger)
- Peut **décroître** à chaque itération (e.g.  $\eta^{(k)} = cst/k$ )

69

69

---

---

---

---

---

---

---

---

---

---

# Perceptron

Une autre version de l'algorithme consiste à analyser **une donnée par itération**.

## Descente de gradient stochastique

Initialiser  $\vec{w}$   
 $k=0$   
 DO  $k=k+1$   
 FOR  $n = 1$  to  $N$   
 IF  $\vec{w}^T \vec{x}_n t_n < 0$  THEN /\* donnée mal classée \*/  
 $\vec{w} = \vec{w} + \eta t_n \vec{x}_n$   
 UNTIL toutes les données sont bien classées.

70

70

---

---

---

---

---

---

---

---

---

---

# Critère du perceptron

Fonctions d'énergie similaires au critère du Perceptron dont le gradient est le même

$$E_D(\vec{w}) = \sum_{\vec{x}_n \in M} -\vec{w}^T \vec{x}_n I_n \quad \text{où } M \text{ est l'ensemble des données mal classées}$$

$$E_D(\vec{w}) = \sum_{n=1}^N \max(0, -t_n \vec{w}^T \vec{x}_n)$$

$$E_D(\vec{w}) = \sum_{n=1}^N \max(0, 1 - t_n \vec{w}^T \vec{x}_n) \quad \text{"Hinge Loss" or "SVM" Loss}$$

Chapitre 6

71

---

---

---

---

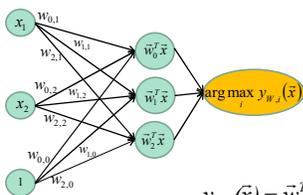
---

---

---

---

# Perceptron Multiclasse (2D et 3 classes)



$$y_{\vec{w},0}(\vec{x}) = \vec{w}_0^T \vec{x} = w_{0,0} + w_{0,1}x_1 + w_{0,2}x_2$$

$$y_{\vec{w},1}(\vec{x}) = \vec{w}_1^T \vec{x} = w_{1,0} + w_{1,1}x_1 + w_{1,2}x_2$$

$$y_{\vec{w},2}(\vec{x}) = \vec{w}_2^T \vec{x} = w_{2,0} + w_{2,1}x_1 + w_{2,2}x_2$$

72

---

---

---

---

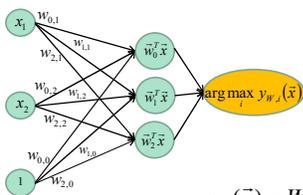
---

---

---

---

# Perceptron Multiclasse (2D et 3 classes)



$$y_W(\vec{x}) = W^T \vec{x}$$

$$y_W(\vec{x}) = \begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} \\ w_{1,0} & w_{1,1} & w_{1,2} \\ w_{2,0} & w_{2,1} & w_{2,2} \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$

73

---

---

---

---

---

---

---

---

# Perceptron Multiclasse

Exemple

$(1.1, -2.0)$

$$y_w(\vec{x}) = \begin{bmatrix} -2 & -3.6 & 0.5 \\ -4 & 2.4 & 4.1 \\ -6 & 4 & -4.9 \end{bmatrix} \begin{bmatrix} 1 \\ 1.1 \\ -2 \end{bmatrix} = \begin{bmatrix} -6.9 \\ -9.6 \\ 8.2 \end{bmatrix}$$

Classe 0  
Classe 1  
Classe 2

74

---

---

---

---

---

---

---

---

74

$$y_w(\vec{x}) = \begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} \\ w_{1,0} & w_{1,1} & w_{1,2} \\ w_{2,0} & w_{2,1} & w_{2,2} \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$

Tel qu'illustré ici, chaque **ligne de la matrice W** contient les paramètres (**normale + biais**) du **plan de séparation** linéaire de chaque classe.

<http://vision.stanford.edu/teaching/cs231n-demos/linear-classify/>

75

---

---

---

---

---

---

---

---

75

$$y_w(\vec{x}) = \begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} \\ w_{1,0} & w_{1,1} & w_{1,2} \\ w_{2,0} & w_{2,1} & w_{2,2} \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$

La **zone bleue** est la zone pour laquelle le score de la **classe 0** est le plus élevé.

76

---

---

---

---

---

---

---

---

76

$$y_W(\vec{x}) = \begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} \\ w_{1,0} & w_{1,1} & w_{1,2} \\ w_{2,0} & w_{2,1} & w_{2,2} \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$

La **zone verte** est la zone pour laquelle le score de la **classe 1** est le plus élevé.

77

77

---

---

---

---

---

---

---

---

---

---


$$y_W(\vec{x}) = \begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} \\ w_{1,0} & w_{1,1} & w_{1,2} \\ w_{2,0} & w_{2,1} & w_{2,2} \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$

La **zone rouge** est la zone pour laquelle le score de la **classe 2** est le plus élevé.

78

78

---

---

---

---

---

---

---

---

---

---

### Perceptron Multiclasse

Fonction de coût

$$E_D(W) = \sum_{\vec{x}_n \in M} (\tilde{w}_j^T \vec{x}_n - \tilde{w}_r^T \vec{x}_n)$$

Somme sur l'ensemble des données mal classées (indicated by a black arrow pointing to the summation symbol)  
 Score de la mauvaise classe (indicated by a red arrow pointing to the first term)  
 Score de la bonne classe (indicated by a green arrow pointing to the second term)

$$\nabla E_D(W) = \sum_{\vec{x}_n \in M} \vec{x}_n$$

79

79

---

---

---

---

---

---

---

---

---

---

### Perceptron Multiclasse

Descente de gradient stochastique

```

Initialiser W
k=0, i=0
DO k=k+1
  FOR n= 1 to N
    j = argmax Wj x̄n
    IF j ≠ tn THEN /* donnée mal classée*/
      w̄j = w̄j - η x̄n
      w̄i = w̄i + η x̄n
  UNTIL toutes les données sont bien classées.

```

80

---

---

---

---

---

---

---

---

80

### Perceptron Multiclasse

Exemple d'entraînement (η=1)

$\vec{x}_n = (0.4, -1), t_n = 0$

$$y_W(\vec{x}) = \begin{bmatrix} -2 & 3.6 & 0.5 \\ -4 & 2.4 & 4.1 \\ -6 & 4 & -4.9 \end{bmatrix} \begin{bmatrix} 1 \\ 0.4 \\ -1 \end{bmatrix} = \begin{bmatrix} -1.6 \\ -7.1 \\ 0.5 \end{bmatrix}$$

Classe 0  
Classe 1  
Classe 2

**FAUX!**

81

---

---

---

---

---

---

---

---

81

### Perceptron Multiclasse

Exemple d'entraînement (η=1)

$\vec{x}_n = (0.4, -1.0), t_n = 0$

$$\vec{w}_0 \leftarrow \vec{w}_0 + \vec{x}_n \quad \begin{bmatrix} -2.0 \\ 3.6 \\ 0.5 \end{bmatrix} + \begin{bmatrix} 1 \\ 0.4 \\ -1 \end{bmatrix} = \begin{bmatrix} -1.0 \\ 4.0 \\ -0.5 \end{bmatrix}$$

$$\vec{w}_2 \leftarrow \vec{w}_2 - \vec{x}_n \quad \begin{bmatrix} -6.0 \\ 4.0 \\ -4.9 \end{bmatrix} - \begin{bmatrix} 1 \\ 0.4 \\ -1 \end{bmatrix} = \begin{bmatrix} -7.0 \\ 3.6 \\ -3.9 \end{bmatrix}$$

82

---

---

---

---

---

---

---

---

82

## En résumé

2 classes

$$E_D(\tilde{w}) = \sum_{\tilde{x}_n \in M} -t_n \tilde{w}^T \tilde{x}_n \quad \text{où } M \text{ est l'ensemble des données mal classées}$$

$$E_D(\tilde{w}) = \sum_{n=1}^N \max(0, -t_n \tilde{w}^T \tilde{x}_n)$$

$$E_D(\tilde{w}) = \sum_{n=1}^N \max(0, 1 - t_n \tilde{w}^T \tilde{x}_n) \quad \text{"Hinge Loss" or "SVM" Loss}$$

K classes

$$E_D(W) = \sum_{\tilde{x}_n \in M} (\tilde{w}_j^T \tilde{x}_n - \tilde{w}_i^T \tilde{x}_n) \quad \text{où } M \text{ est l'ensemble des données mal classées}$$

$$E_D(W) = \sum_{n=1}^N \sum_j \max(0, \tilde{w}_j^T \tilde{x}_n - \tilde{w}_i^T \tilde{x}_n)$$

$$E_D(W) = \sum_{n=1}^N \sum_j \max(0, 1 + \tilde{w}_j^T \tilde{x}_n - \tilde{w}_i^T \tilde{x}_n) \quad \text{"Hinge Loss" or "SVM" Loss} \quad 83$$

83

---

---

---

---

---

---

---

---

---

---

---

---



84

84

---

---

---

---

---

---

---

---

---

---

---

---

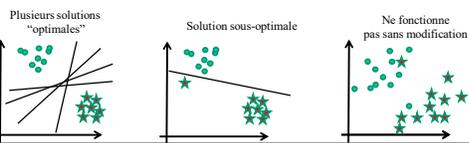
## Perceptron

### Avantages:

- Très simple
- Ne suppose pas que les données sont **gaussiennes**.
- Si les données sont linéairement séparables, le Perceptron est **garanti(!)** de **converger** en un nombre fini d'itérations (voir Duda-Hart-Stork pour la preuve)

### Limitations:

- Gradient nul pour plusieurs solutions => plusieurs solutions "optimales"
- Les données doivent être **linéairement séparables**



85

---

---

---

---

---

---

---

---

---

---

---

---

# Perceptron

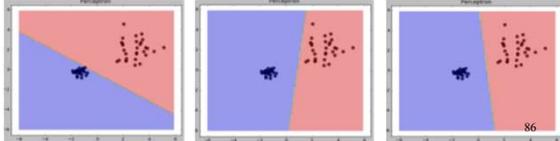
## Avantages:

- Très simple
- Ne suppose pas que les données sont **gaussiennes**.
- Si les données sont linéairement séparables, le Perceptron est **garanti (!) de converger** en un nombre fini d'itérations (see Duda-Hart-Stork for proof)

## Limitations:

- Gradient nul pour plusieurs solutions => plusieurs solutions "optimales"
- Les données doivent être **linéairement séparables**

Plusieurs solutions "optimales"



86

---

---

---

---

---

---

---

---

# Comment améliorer le Perceptron?

Trois façons d'améliorer le Perceptron

1. Nouvelle **fonction d'activation** + nouvelle **fonction de coût**
2. Utiliser des **fonctions de base** → **Régression logistique**
3. **Nouveau réseau** → **Méthodes à noyau (chap. 5-6)**  
→ **Réseaux de neurones multicouches (chap. 7)**

87

87

---

---

---

---

---

---

---

---

# Régression logistique

(Sections 4.2.0, 4.3.2, 5.2.0 –Bishop)

88

88

---

---

---

---

---

---

---

---

### Amélioration du Perceptron

(2D, 2 classes)  
Nouvelle fonction d'activation : **sigmoïde logistique**

$$\sigma(v) = \frac{1}{1 + e^{-v}}$$

Fonction d'activation sigmoïdale

$$y_{\vec{w}}(\vec{x}) = \sigma(\vec{w}^T \vec{x}) = \frac{1}{1 + e^{-\vec{w}^T \vec{x}}}$$

89

---

---

---

---

---

---

---

---

---

---

89

### Amélioration du Perceptron

(2D, 2 classes)  
Nouvelle fonction d'activation : **sigmoïde logistique**

$$\sigma(t) = \frac{1}{1 + e^{-t}}$$

Neurone

$$y_{\vec{w}}(\vec{x}) = \sigma(\vec{w}^T \vec{x}) = \frac{1}{1 + e^{-\vec{w}^T \vec{x}}}$$

90

---

---

---

---

---

---

---

---

---

---

90

### Amélioration du Perceptron

(2D, 2 classes)  
Nouvelle fonction d'activation : **sigmoïde logistique**

Neurone

$$y_{\vec{w}}(\vec{x}) = \sigma(\vec{w}^T \vec{x})$$

91

---

---

---

---

---

---

---

---

---

---

91

## Amélioration du Perceptron

(N-D, 2 classes)

Nouvelle fonction d'activation : **sigmoïde logistique**

Exemple 3D

Neurone

92

---

---

---

---

---

---

---

---

---

---

## Amélioration du Perceptron

(N-D, 2 classes)

Exemple

$\vec{x}_n = (0.4, -1.0), \vec{w} = [2.0, -3.6, 0.5]$

Puisque 0.125 est inférieur à 0.5,  $\vec{x}_n$  est **derrière** le plan.

93

---

---

---

---

---

---

---

---

---

---

## Amélioration du Perceptron

(N-D, 2 classes)

Avec une sigmoïde, on peut **simuler une probabilité conditionnelle** sur  $c_1$  étant donné  $\vec{x}$

$$y_w(\vec{x}) = \sigma(\vec{w}^T \vec{x}) \Rightarrow P(c_1 | \vec{x})$$

**Preuve:**

$$P(c_1 | \vec{x}) = \frac{P(\vec{x} | c_1)P(c_1)}{P(\vec{x} | c_0)P(c_0) + P(\vec{x} | c_1)P(c_1)} \quad (\text{Bayes})$$

$$= \frac{1}{1 + \frac{P(\vec{x} | c_0)P(c_0)}{P(\vec{x} | c_1)P(c_1)}}$$

$$= \frac{1}{1 + e^{-a}} \quad \text{où } a = \ln \left[ \frac{P(\vec{x} | c_0)P(c_0)}{P(\vec{x} | c_1)P(c_1)} \right]$$

$$= \sigma(a)$$

94

---

---

---

---

---

---

---

---

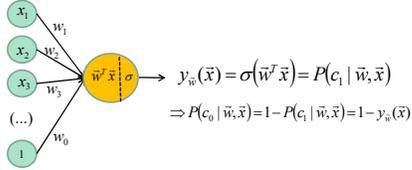
---

---

## Amélioration du Perceptron

(N-D, 2 classes)

En d'autres mots, si on entraîne correctement un réseau logistique, on finit par apprendre la **probabilité conditionnelle de la classe c**.



Quelle est la fonction de coût d'un réseau logistique?

95

---

---

---

---

---

---

---

---

---

---

95

## Fonction de coût d'un réseau logistique?

(2 classes)

Dans le cas d'un réseau logistique nous avons

Ensemble d'entraînement :  $D = \{(\bar{x}_1, t_1), (\bar{x}_2, t_2), \dots, (\bar{x}_n, t_n)\}$

Sortie du réseau :  $y_w(\bar{x}) = \sigma(\bar{w}^T \bar{x}) = P(c_1 | \bar{w}, \bar{x})$

$$P(D | \bar{w}) = \prod_{n=1}^N P(c_1 | \bar{w}, \bar{x}_n)^{t_n} (1 - P(c_1 | \bar{w}, \bar{x}_n))^{1-t_n}$$

$$= \prod_{n=1}^N y_w(\bar{x}_n)^{t_n} (1 - y_w(\bar{x}_n))^{1-t_n}$$

101

101

---

---

---

---

---

---

---

---

---

---

## Fonction de coût d'un réseau logistique?

(2 classes)

$$P(D | \bar{w}) = \prod_{n=1}^N y_w(\bar{x}_n)^{t_n} (1 - y_w(\bar{x}_n))^{1-t_n}$$

**Solution : Maximum de vraisemblance**

$$W = \arg \max_W P(D | W)$$

$$= \arg \max_W \prod_{n=1}^N y_w(\bar{x}_n)^{t_n} (1 - y_w(\bar{x}_n))^{1-t_n}$$

$$= \arg \min_W \sum_{n=1}^N -\ln \left[ y_w(\bar{x}_n)^{t_n} (1 - y_w(\bar{x}_n))^{1-t_n} \right]$$

$$= \arg \min_W \underbrace{\sum_{n=1}^N t_n \ln(y_w(\bar{x}_n)) + (1 - t_n) \ln(1 - y_w(\bar{x}_n))}_{E_D(\bar{w})}$$

102

102

---

---

---

---

---

---

---

---

---

---

### Fonction de coût d'un réseau logistique?

(2 classes)

$$P(D|\bar{w}) = -\prod_{n=1}^N y_n(\bar{x}_n)^{y_n} (1 - y_n(\bar{x}_n))^{1-y_n}$$

La fonction de coût est **-ln de la vraisemblance**

$$E_D(\bar{w}) = -\sum_{n=1}^N [t_n \ln(y_n(\bar{x}_n)) + (1-t_n) \ln(1-y_n(\bar{x}_n))]$$

On peut également démontrer que

$$\nabla_{\bar{w}} E_D(\bar{w}) = \sum_{n=1}^N (y_n(\bar{x}_n) - t_n) \bar{x}_n$$

Preuve en classe ou en devoir

Entropie croisée (Cross entropy)

Contrairement au Perceptron le gradient ne dépend pas seulement des données mal classées

103

---

---

---

---

---

---

---

---

---

---

103

### Optimisation d'un réseau logistique

Optimisation par batch

Initialiser  $\bar{w}$   
 $k=0, i=0$   
 DO  $k=k+1$

$$\frac{dE_D(\bar{w})}{d\bar{w}} = \sum_{n=1}^N (y_n(\bar{x}_n) - t_n) \bar{x}_n$$

$$\bar{w} = \bar{w} - \eta \frac{dE_D(\bar{w})}{d\bar{w}}$$

UNTIL  $K=K\_MAX$ .

Descente de gradient stochastique

Initialiser  $\bar{w}$   
 $k=0, i=0$   
 DO  $k=k+1$

FOR  $n = 1$  to  $N$

$$\bar{w} = \bar{w} - \eta (y_n(\bar{x}_n) - t_n) \bar{x}_n$$

UNTIL  $K=K\_MAX$ .

104

---

---

---

---

---

---

---

---

---

---

104

### Réseau logistique

Avantages:

- Plus stable que le Perceptron
- Fonctionne mieux avec des données non séparables

Perceptron

Réseau logistique

105

---

---

---

---

---

---

---

---

---

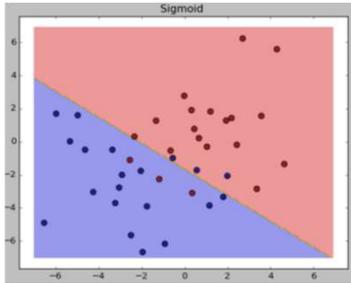
---

105

# Réseau logistique

Avantages:

- Plus stable que le Perceptron
- Fonctionne mieux avec des données non séparables



106

106

---

---

---

---

---

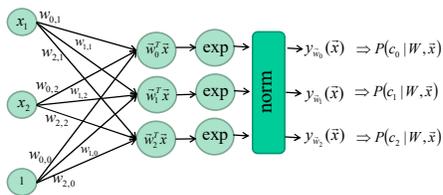
---

---

---

# Et pour K>2 classes?

Nouvelle fonction d'activation : **Softmax**



$$y_{w_i}(\vec{x}) = \frac{e^{w_i^T \vec{x}}}{\sum_c e^{w_c^T \vec{x}}}$$

107

107

---

---

---

---

---

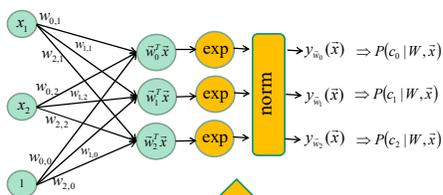
---

---

---

# Et pour K>2 classes?

Nouvelle fonction d'activation : **Softmax**



Softmax

108

108

---

---

---

---

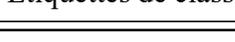
---

---

---

---

## Et pour K>2 classes?

airplane		'airplane' => t = [1000000000]
automobile		'automobile' => t = [0100000000]
bird		'bird' => t = [0010000000]
cat		'cat' => t = [0001000000]
deer		'deer' => t = [0000100000]
dog		'dog' => t = [0000010000]
frog		'frog' => t = [0000001000]
horse		'horse' => t = [0000000100]
ship		'ship' => t = [0000000010]
truck		'truck' => t = [0000000001]

Étiquettes de classe : **one-hot vector**<sub>109</sub>

109

---

---

---

---

---

---

---

---

---

---

---

---

## Et pour K>2 classes?

$$P(D|W) = \prod_{n=1}^N \prod_{k=1}^K (P(t_n | W, \bar{x}_n))^{t_{nk}}$$

Entropie croisée (*cross entropy*)

$$\rightarrow E_D(W) = -\ln(P(D|W)) = -\sum_{n=1}^N \sum_{k=1}^K t_{kn} \ln P(t_n | W, \bar{x}_n)$$

Puisqu'on veut que la sortie du réseau  $y_W(\bar{x}_n)$  soit égale à  $P(t_n | W, \bar{x}_n)$

$$\rightarrow E_D(W) = -\sum_{n=1}^N \sum_{k=1}^K t_{kn} \ln y_{W_k}(\bar{x}_n)$$

110

110

---

---

---

---

---

---

---

---

---

---

---

---

## Et pour K>2 classes?

En general, on ajoute 1/N pour normaliser le calcul de la loss

$$E_D(W) = -\frac{1}{N} \sum_{n=1}^N \sum_{k=1}^K t_{kn} \ln y_{W_k}(\bar{x}_n)$$

On peut montrer que

$$\nabla_W E_D(W) = \frac{1}{N} \sum_{n=1}^N \bar{x}_n (y_W(\bar{x}_n) - t_{kn})$$

111

111

---

---

---

---

---

---

---

---

---

---

---

---

### Optimisation d'un réseau logistique multiclass

Optimisation par batch

Initialiser  $W$   
 $k=0, i=0$   
 DO  $k=k+1$

$$\nabla_w E(W) = \frac{1}{N} \sum_{i=1}^N (y_i(\vec{x}_i) - t_i) \vec{x}_i$$

$$W = W - \eta \nabla_w E(W)$$

UNTIL  $K=K\_MAX$ .

Descente de gradient stochastique

Initialiser  $W$   
 $k=0, i=0$   
 DO  $k=k+1$   
 FOR  $n = 1$  to  $N$   
 $W = W - \eta (y_n(\vec{x}_n) - t_n) \vec{x}_n$

UNTIL  $K=K\_MAX$ .

112

---

---

---

---

---

---

---

---

---

---

112

### Wow! Beaucoup d'information...

### Résumons...

113

---

---

---

---

---

---

---

---

---

---

113

### Réseaux de neurones

2 classes

Sign activation

sigmoid activation

114

---

---

---

---

---

---

---

---

---

---

114

### Réseaux de neurones

K classes

Softmax activation

115

115

---

---

---

---

---

---

---

---

### Fonctions de coûts

2 classes

$$E_D(\tilde{w}) = \sum_{\tilde{x}_n \in M} -t_n \tilde{w}^T \tilde{x}_n \quad \text{où } M \text{ est l'ensemble des données mal classées}$$

$$E_D(\tilde{w}) = \sum_{n=1}^N \max(0, -t_n \tilde{w}^T \tilde{x}_n)$$

$$E_D(\tilde{w}) = \sum_{n=1}^N \max(0, 1 - t_n \tilde{w}^T \tilde{x}_n) \quad \text{“Hinge Loss” ou “SVM” Loss}$$

$$E_D(\tilde{w}) = -\sum_{n=1}^N t_n \ln(y_{\tilde{w}}(\tilde{x}_n)) + (1 - t_n) \ln(1 - y_{\tilde{w}}(\tilde{x}_n)) \quad \text{Entropie croisée (ou cross entropy)}$$

116

116

---

---

---

---

---

---

---

---

### Fonctions de coûts

K classes

$$E_D(W) = \sum_{\tilde{x}_n \in M} (W_j^T \tilde{x}_n - W_{i'}^T \tilde{x}_n) \quad \text{où } M \text{ est l'ensemble des données mal classées}$$

$$E_D(W) = \sum_{n=1}^M \sum_j \max(0, W_j^T \tilde{x}_n - W_{i'}^T \tilde{x}_n)$$

$$E_D(W) = \sum_{n=1}^M \sum_j \max(0, 1 + W_j^T \tilde{x}_n - W_{i'}^T \tilde{x}_n) \quad \text{“Hinge Loss” ou “SVM” Loss}$$

$$E_D(W) = -\sum_{n=1}^N \sum_{k=1}^K t_{kn} \ln y_{W,k}(\tilde{x}_n) \quad \text{Entropie croisée avec « one hot vector » (ou cross entropy)}$$

117

117

---

---

---

---

---

---

---

---

## Optimisation

**Descente de gradient**

$$\vec{w}^{[k+1]} = \vec{w}^{[k]} - \eta^{[k]} \nabla E$$

↗ Gradient de la fonction de coût  
 ↘ Taux d'apprentissage ou "learning rate".

**Optimisation par Batch**

Initialiser  $\vec{w}$   
 $k=0$   
 FAIRE  $k=k+1$

$$\vec{w} = \vec{w} - \eta^{[k]} \sum_T \nabla E(\vec{x}_i)$$

JUSQU'À ce que toutes les données  
 sont bien classées ou  $k=MAX\_ITER$

**Descente de gradient stochastique**

Initialiser  $\vec{w}$   
 $k=0$   
 FAIRE  $k=k+1$   
 FOR  $n = 1$  to  $N$   
 $\vec{w} = \vec{w} - \eta^{[k]} \nabla E(\vec{x}_n)$

JUSQU'À ce que toutes les données  
 sont bien classées ou  $k=MAX\_ITER$

Parfois  $\eta^{[k]} = cst / k$

118

---

---

---

---

---

---

---

---

---

---

118

## Régularisation

119

---

---

---

---

---

---

---

---

---

---

119

## Régularisation

Différents poids peuvent donner le même score

$\vec{x} = (1.0, 1.0, 1.0)$   
 $\vec{w}_1 = [1, 0, 0]$   
 $\vec{w}_2 = [1/3, 1/3, 1/3]$

$\vec{w}_1 \vec{x} = \vec{w}_2 \vec{x} = 1$

Quels poids sont les meilleurs?

Solution:  
 Maximum a posteriori

120

---

---

---

---

---

---

---

---

---

---

120

**Maximum a posteriori**

Régularisation

**Maximum de vraisemblance**

$$W = \operatorname{argmax}_W P(D|W)$$

...

$$= \operatorname{argmin}_W \underbrace{\sum_{n=1}^N -\ln P(t_n | \tilde{x}_n, W)}_{E_D(W)}$$

**Maximum a posteriori**

$$W = \operatorname{argmax}_W P(W|D)$$

...

$$= \operatorname{argmin}_W \underbrace{\sum_{n=1}^N -\ln P(t_n | \tilde{x}_n, W)}_{E_D(W)} + \lambda R(W)$$

121

121

---

---

---

---

---

---

---

---

**Note :**

il est fréquent de combiner différentes **fonctions de coût** avec différentes **fonctions de régularisation**

123

123

---

---

---

---

---

---

---

---

**Maximum a posteriori**

$$E(W) = \sum_{n=1}^N l(y_W(\tilde{x}_n), t_n) + \lambda R(W)$$

↑ **Fonction de perte**      ↑ **Constante**  
↑ **Regularisation**

$$R(\theta) = \|W\|_1 \text{ ou } \|W\|_2$$

124

124

---

---

---

---

---

---

---

---

## Maximum a posteriori

Exemple : Hinge loss + régularisation L2

$$E(w) = \sum_{n=1}^N \max(0, 1 - t_n w^T \tilde{x}_n) + \lambda \|w\|^2$$

$$\nabla_w E(w) = \sum_{\tilde{x}_n \in M} -t_n \tilde{x}_n + 2\lambda \sum_{d=0}^D w_d$$

125

125

---

---

---

---

---

---

---

---

## Maximum a posteriori

Exemple : entropie croisée + régularisation L2

$$\arg \min_w -\ln(P(D|W)) + \lambda \|W\|^2$$

$$\arg \min_w -\sum_{n=0}^N t_n \ln(y_w(\tilde{x}_n)) + (1 - t_n) \ln(1 - y_w(\tilde{x}_n)) + \lambda \sum_{i=1}^d (w_i)^2$$

$$\nabla_w E(w) = \sum_{n=1}^N (y_w(\tilde{x}_n) - t_n) \tilde{x}_n + 2\lambda \sum_{d=0}^D w_d$$

126

126

---

---

---

---

---

---

---

---

## Exemples

```
from sklearn.linear_model import SGDClassifier
C = SGDClassifier(loss='perceptron', learning_rate='constant',
                  eta0=1, n_iter=1000)

C = SGDClassifier(loss='log', learning_rate='constant',
                  eta0=1, n_iter=1000)

C = SGDClassifier(loss='hinge', penalty='l2', alpha=0.01,
                  learning_rate='invscaling', eta0=1, n_iter=1000)
```

127

127

---

---

---

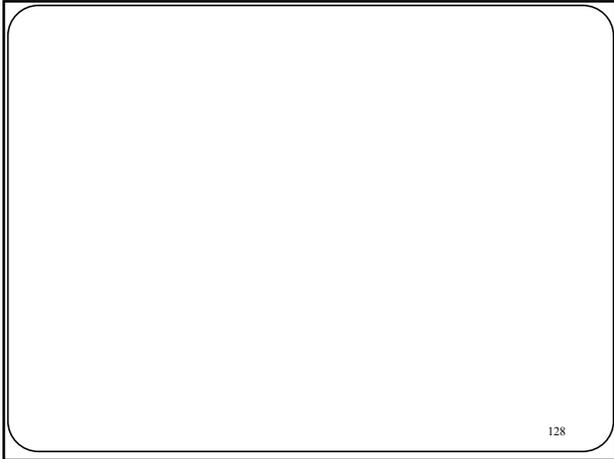
---

---

---

---

---



128

128

---

---

---

---

---

---

---

---



129

129

---

---

---

---

---

---

---

---

**Entropie croisée vs *Hinge Loss***

Dépendamment de la *loss* utilisée, la **sortie du réseau** sera différente

- *Hinge loss* : sortie multiplication matrice-vecteur
- Entropie croisée : sortie softmax

$y_{W_i}(\vec{x}_n) = \vec{W}_i^T \vec{x}$

$y_{W_i}(\vec{x}_n) = \frac{e^{\vec{W}_i^T \vec{x}}}{\sum_j e^{\vec{W}_j^T \vec{x}}}$

130

130

---

---

---

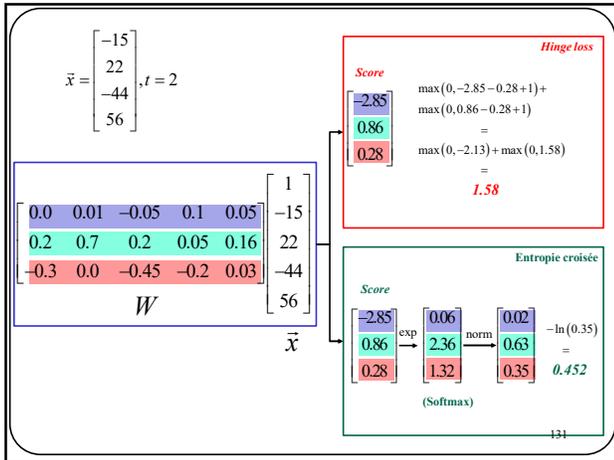
---

---

---

---

---




---

---

---

---

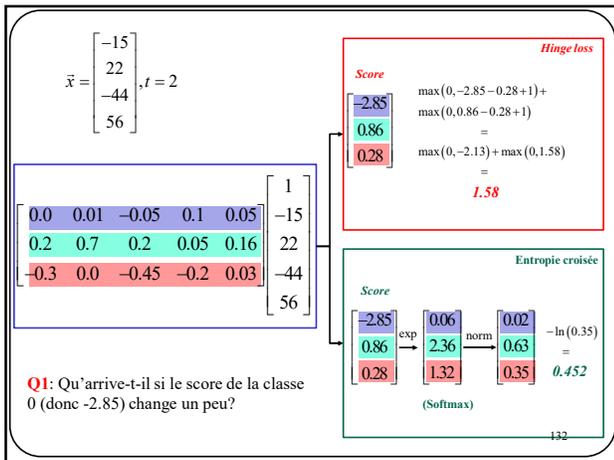
---

---

---

---

131




---

---

---

---

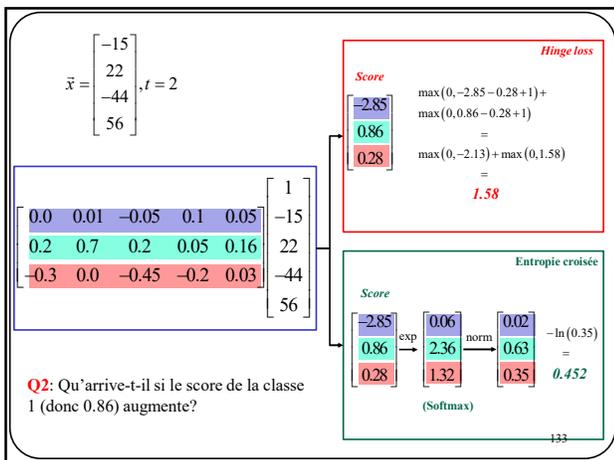
---

---

---

---

132




---

---

---

---

---

---

---

---

133

