Préliminaires

- Balázs Kégl (prononcé “Bolage”)
- balazs.kegl@gmail.com
- Arrêtez-moi si qqchose n’est pas clair!
- Posez des questions!

Résumé

- Introduction
- Exemple : OCR – MNIST
- Une solution : AdaBoost
- Extraction des traits : filtres de Haar

Algorithmes “classiques” vs. d’apprentissage

- Approche classique
  - description formelle des contraintes de l’entrée et de la sortie souhaitée
  - compréhension du problème computationnel
  - design d’une solution algorithmique basée sur ces connaissances
- Problèmes
  - Connaissances incomplètes
  - Algorithme trop coûteux

Algorithmes “classiques” vs. d’apprentissage

- Approche d’apprentissage
  - données (exemples) de forme (entrée,sortie)
  - compréhension partielle du problème : connaissances a-priori
  - apprendre : chercher dans un ensemble de fonctions

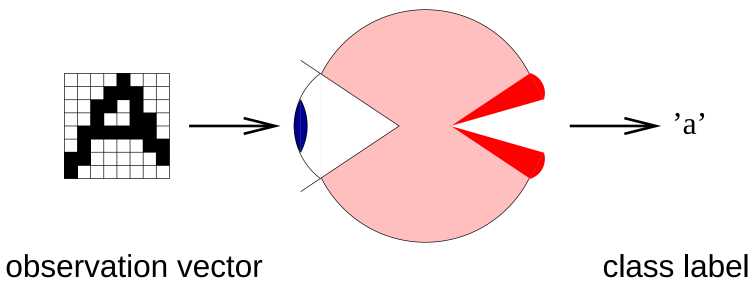
Catégories d'apprentissage automatique

- Classification
  - classifier le nouvel exemple
- Régression
  - faire une prédiction à partir du nouvel exemple
- Estimation de densité
  - dire si le nouvel exemple ressemble aux exemples déjà vus
- (Modèles graphiques)

Exemple de classification

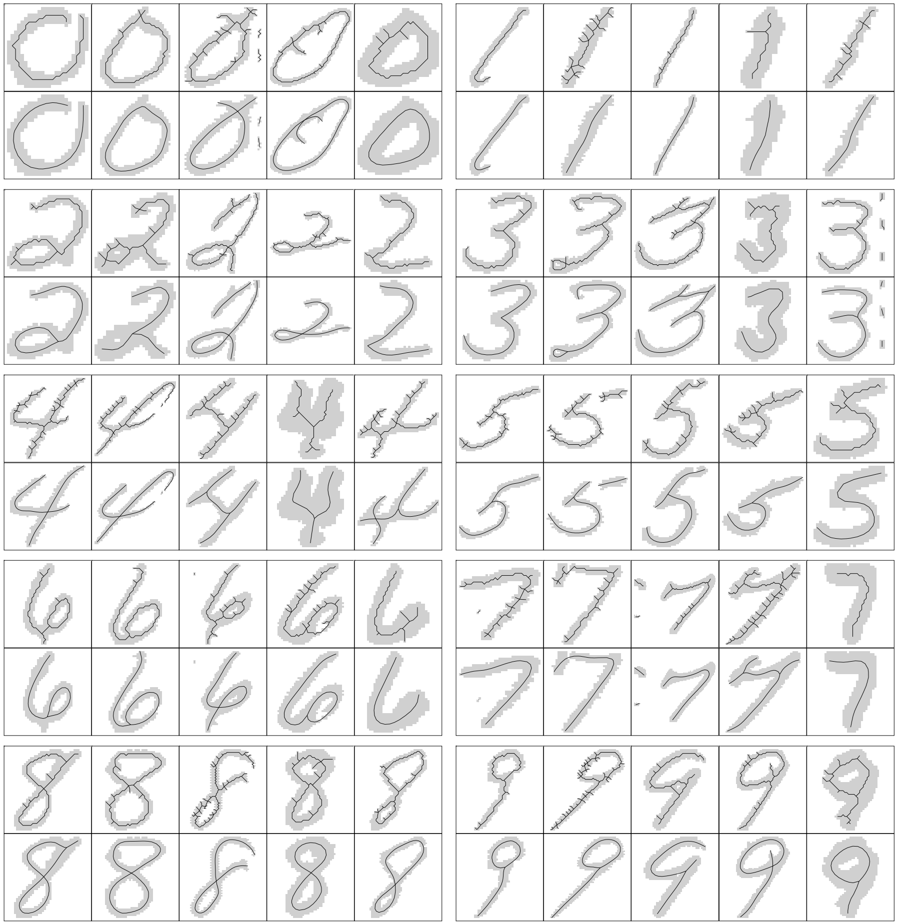
- OCR $=$ Optical character recognition
  - entrée : images de caractères (chiffres) manuscrites
  - sortie : l'étiquette ou classe
  - but : design d'une fonction de classification :

$$f\ : \{\text{image}\} \rightarrow \{0,1,2,3,4,5,6,7,8,9\}$$



Exemple de classification

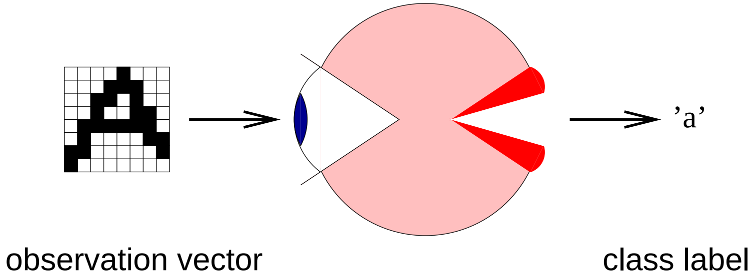
- MNIST (`yann.lecun.com/exdb/mnist/`)



Exemple de classification

- MNIST (`yann.lecun.com/exdb/mnist/`)
  - extrait de la base de données NIST
  - images grayscale, 28×28
  - 60000 entraînement, 10000 test

Le modèle d'apprentissage supervisé

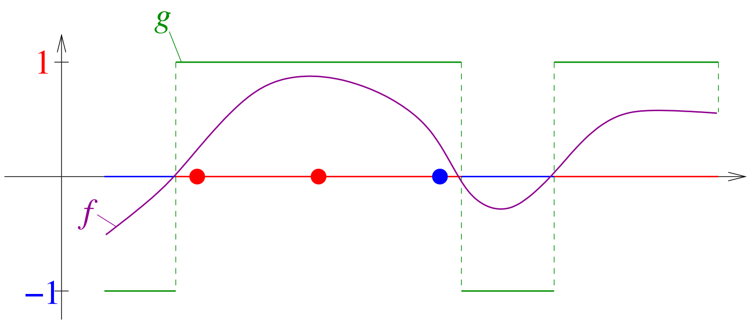


- vecteur d'observation: $\mathbf{x} \in \mathbb{R}^d$

- étiquette de classe: $y \in \{-1, 1\}$ (ou $y \in \{1, \ldots, K\}$)

- classifieur: $g : \mathbb{R}^d \to \{-1, 1\}$

Le modèle d'apprentissage supervisé

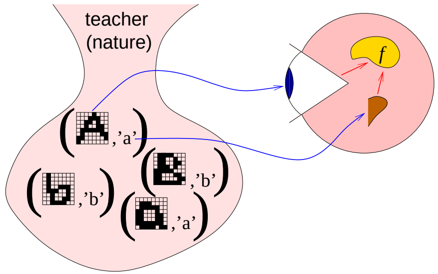


- Fonction discriminante: $f : \mathbb{R}^d \rightarrow [-1, 1]$
  - $\longrightarrow$ fonction de classification

$$g(\mathbf{x}) = \begin{cases} 1, & \text{if } f(\mathbf{x}) \geq 0, \\ -1, & \text{if } f(\mathbf{x}) < 0 \end{cases}$$

  - frontière de décision: $\{\mathbf{x} : f(\mathbf{x}) = 0\}$

Le modèle d'apprentissage supervisé



- Apprentissage par l'expérience, avec un superviseur

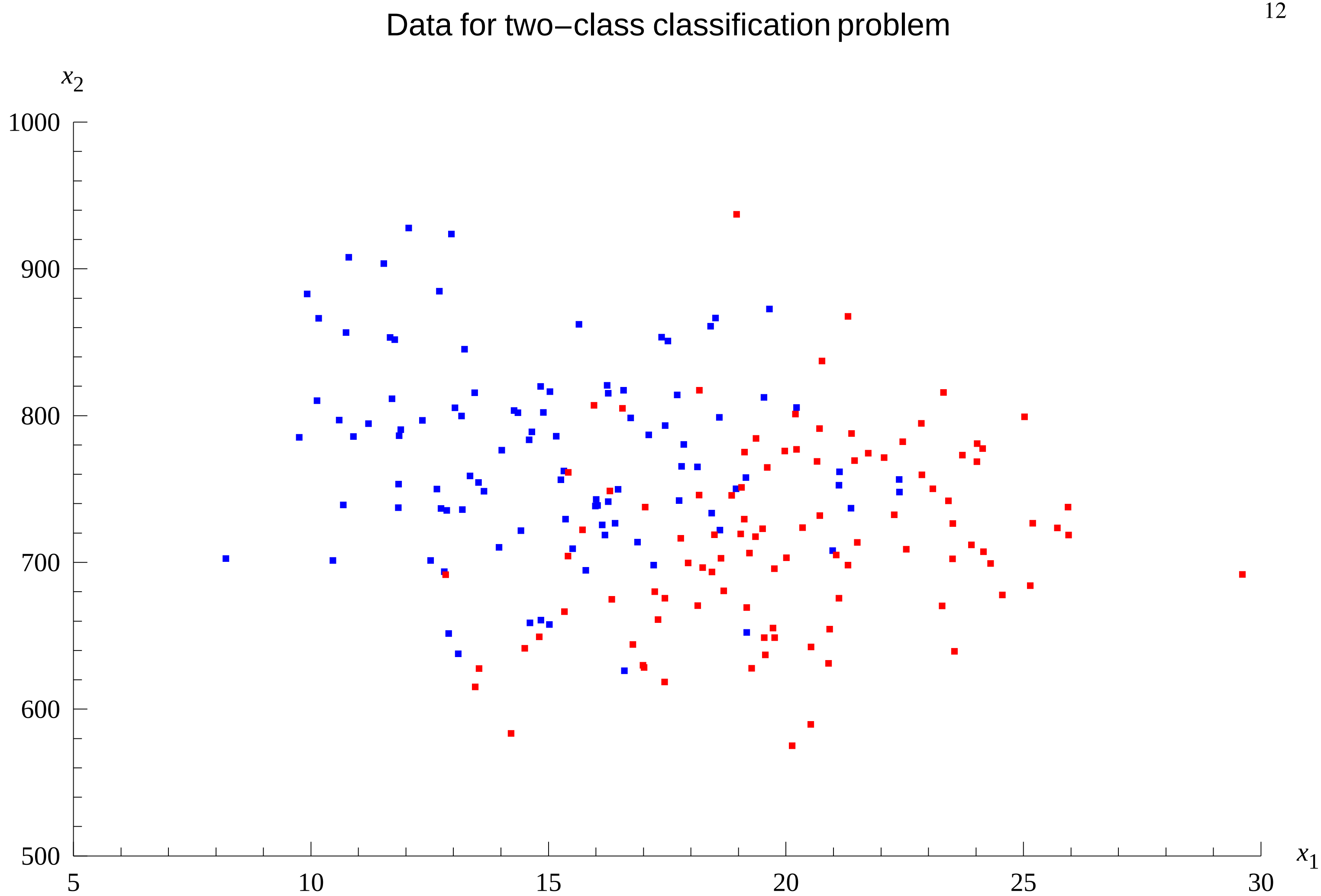
  - échantillon d'entraînement: $D_n = \left\{ (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n) \right\}$

  - ensemble des fonctions: $\mathcal{F}$

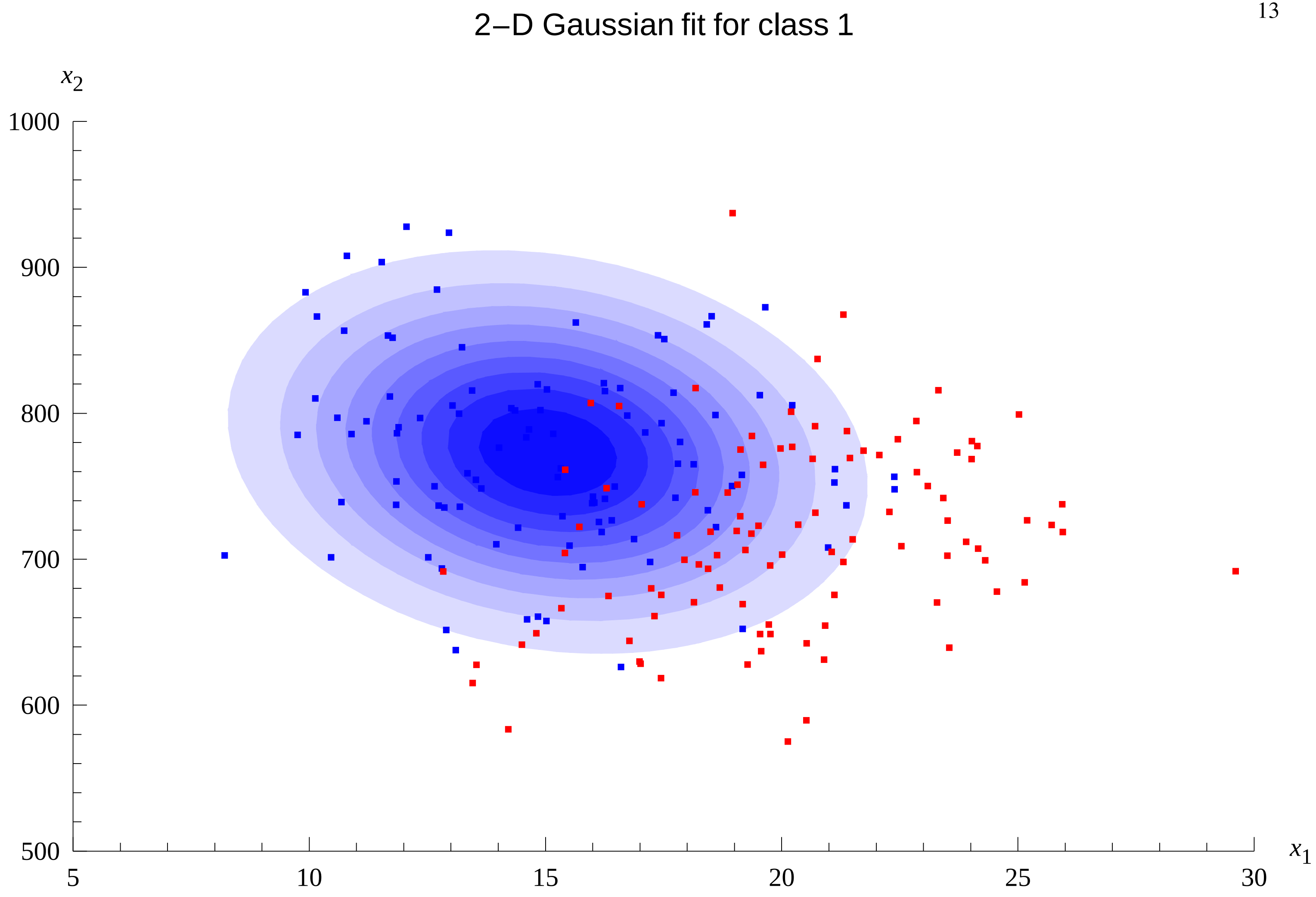
  - algorithme d'apprentissage: $\textsc{Algo} : \left(\mathbb{R}^d \times \{-1, 1\} \right)^n \to \mathcal{F}$

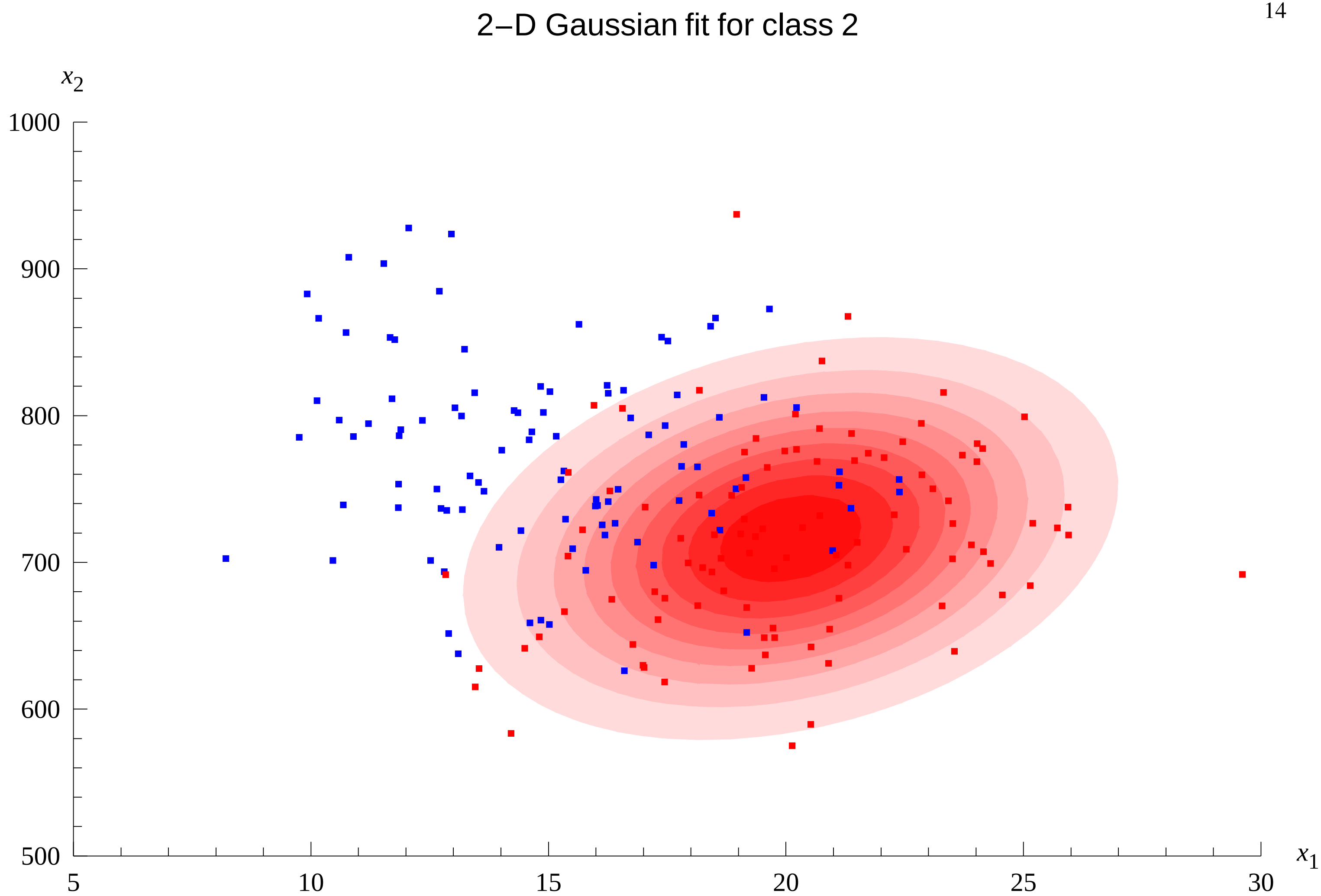
$$\textsc{Algo}(D_n) \mapsto f$$

  - but: petite erreur de généralisation $P\big[g(\mathbf{X}) \neq Y \big] = P\big[f(\mathbf{X})Y \leq 0 \big]$



Data for two−class classification problem
x2
1000
900
800
700
600
500
5
10
15
20
25
30
x1


2−D Gaussian fit for class 1
x_2
1000
900
800
700
600
500
5
10
15
20
25
30
x_1


2–D Gaussian fit for class 2
x_2
1000
900
800
700
600
500
5
10
15
20
25
30
x_1

Classification

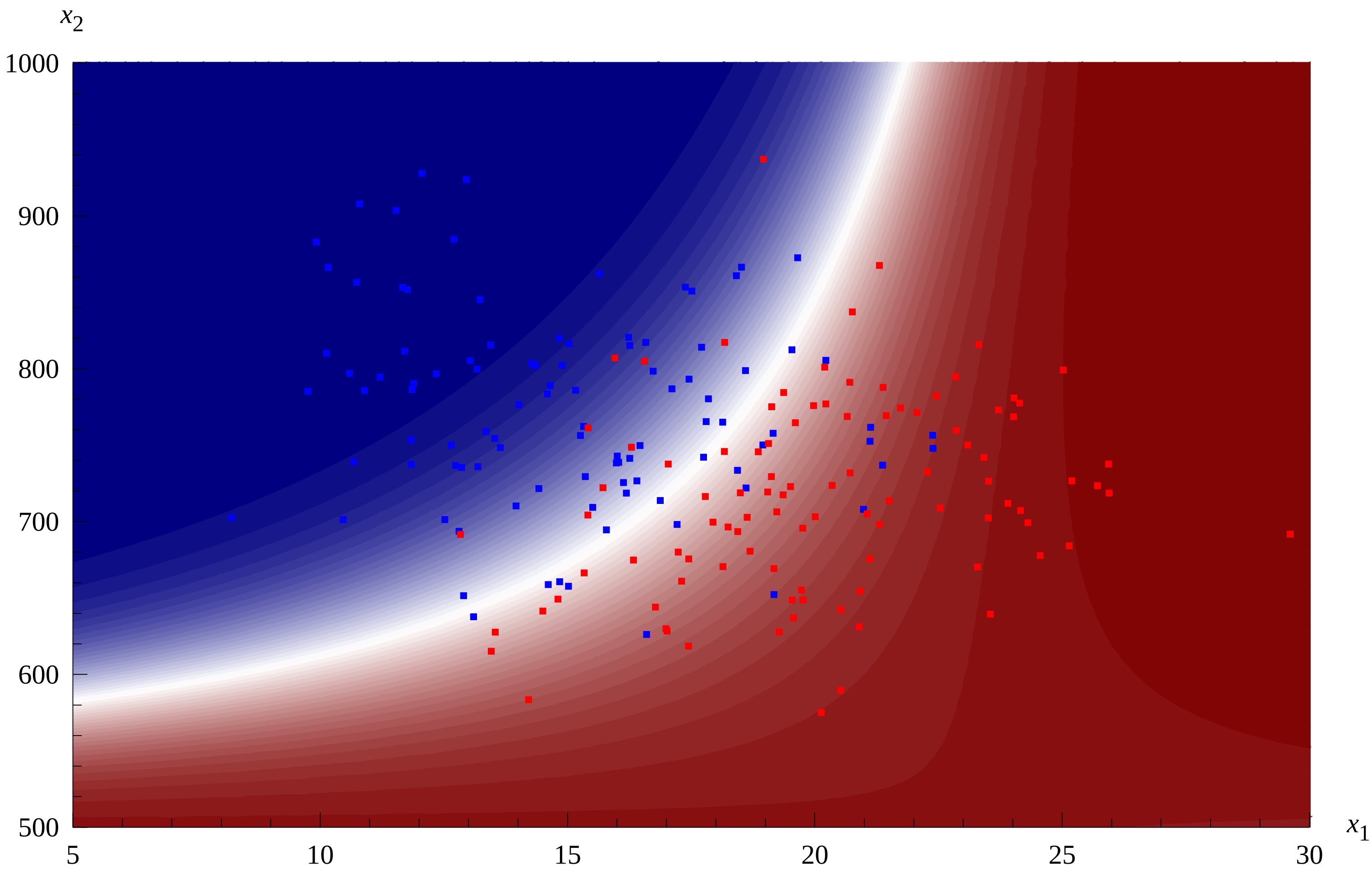
- Terminology
  - Conditional densities: $p(\mathbf{x}|Y=1)$, $p(\mathbf{x}|Y=-1)$
  - Prior probabilities: $p(Y=1)$, $p(Y=-1)$
  - Posterior probabilities: $p(Y=1|\mathbf{x})$, $p(Y=-1|\mathbf{x})$
- Bayes theorem:

$$p(Y=1|\mathbf{x}) = \frac{p(\mathbf{x}|Y=1)p(Y=1)}{p(\mathbf{x})} \sim p(\mathbf{x}|Y=1)p(Y=1)$$

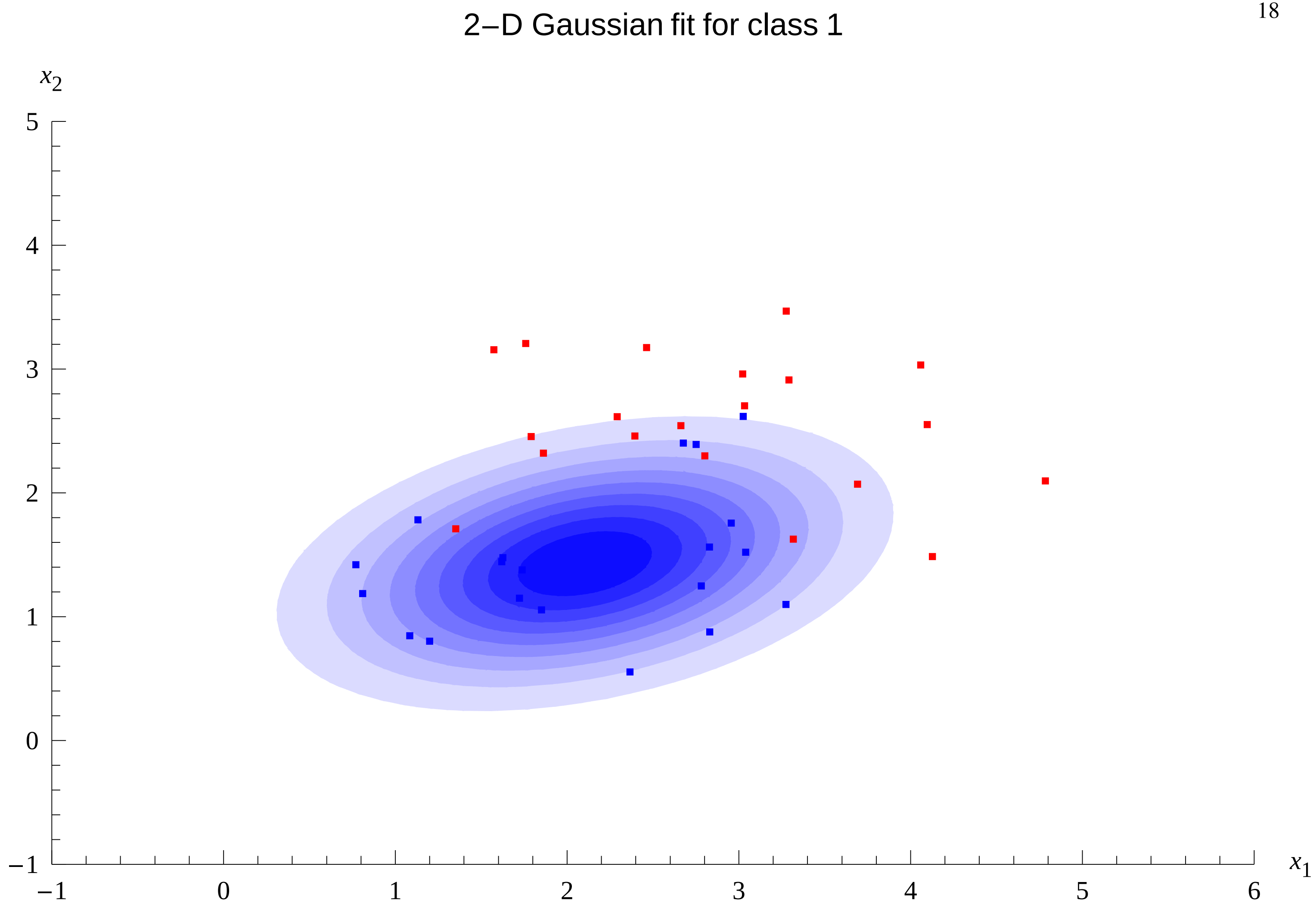
  - Decision:

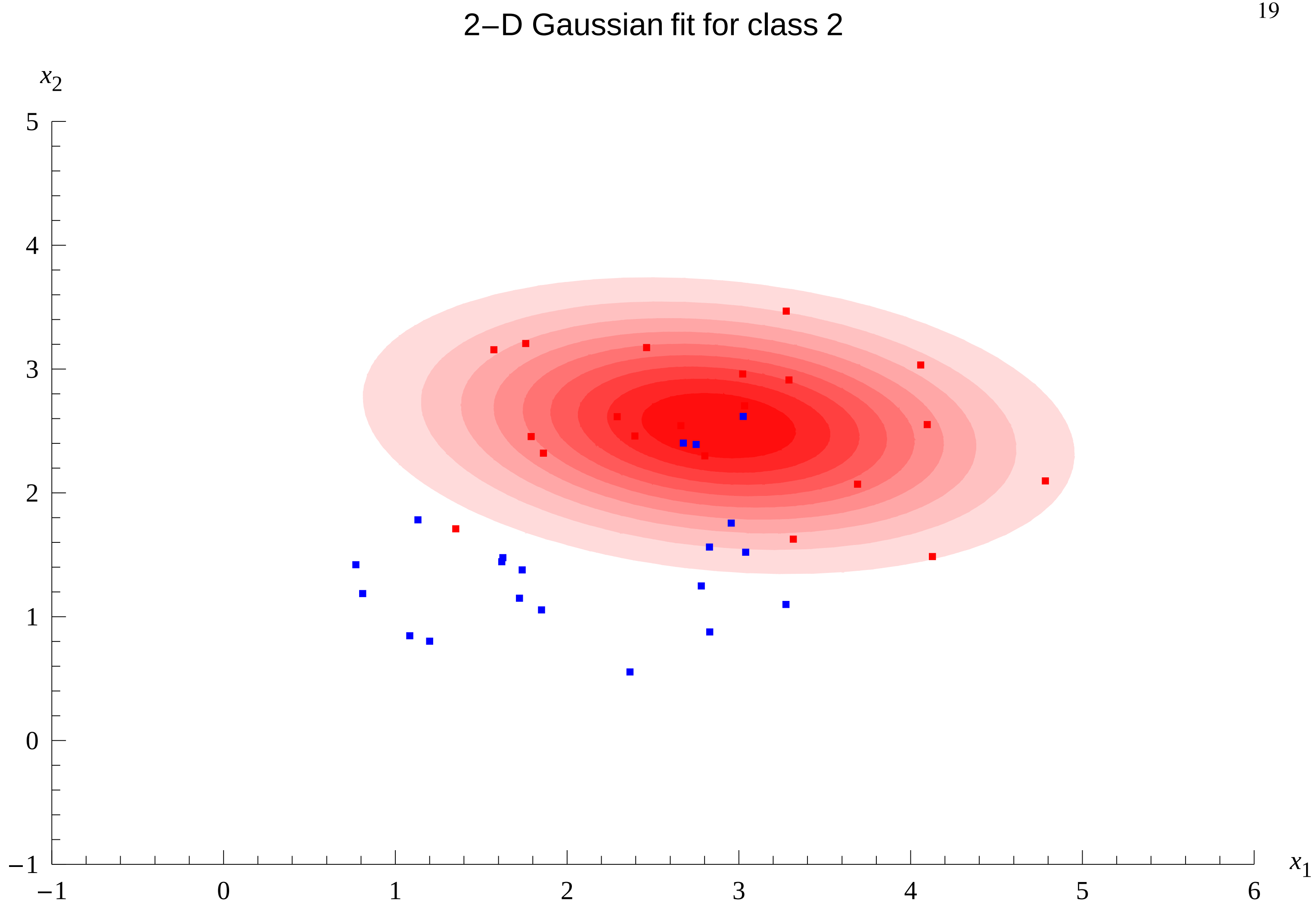
$$g(\mathbf{x}) = \begin{cases} 1 & \text{if } \frac{p(\mathbf{x}|Y=1)p(Y=1)}{p(\mathbf{x}|Y=-1)p(Y=-1)} > 1, \\ -1 & \text{otherwise.} \end{cases}$$

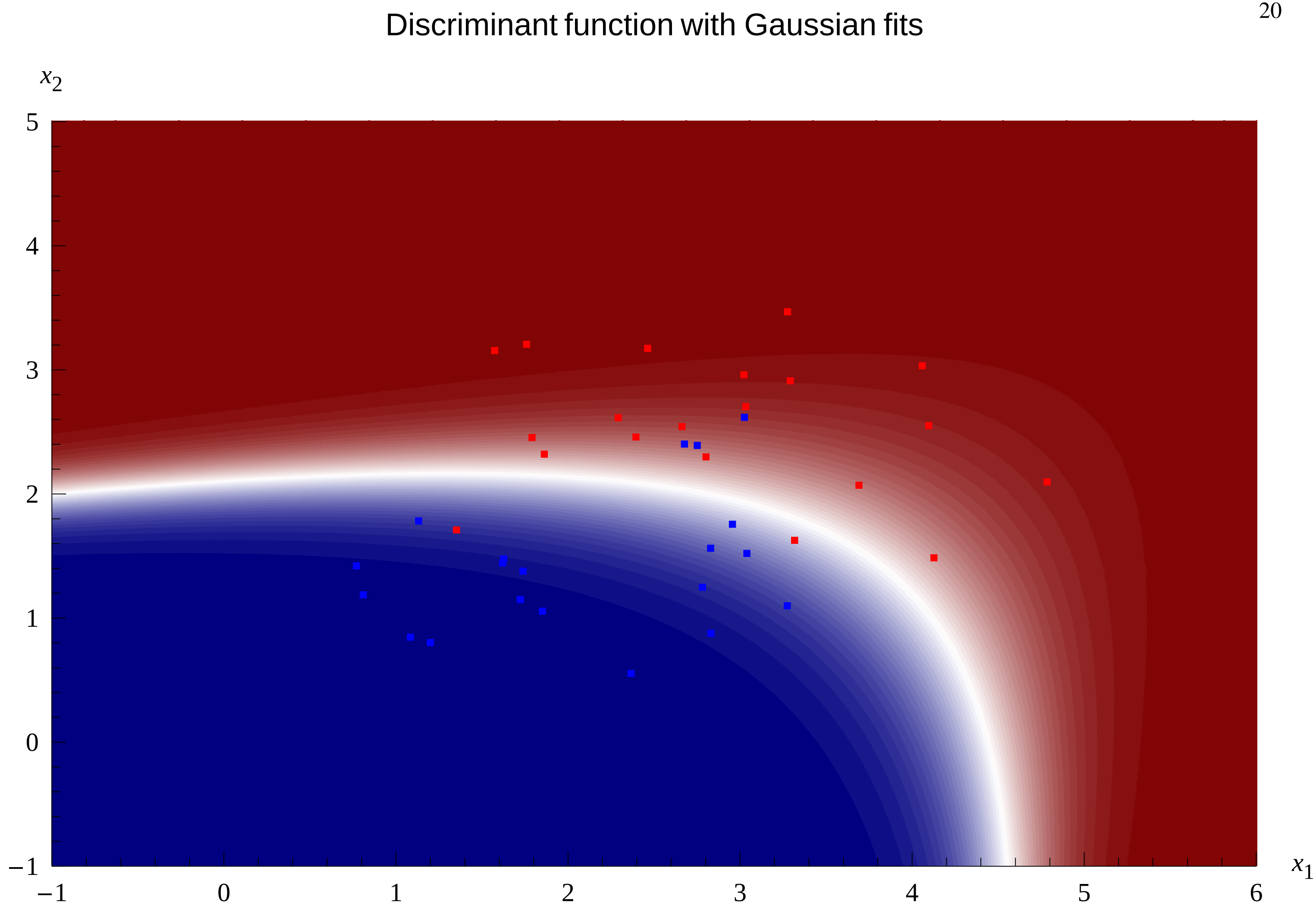
Discriminant function with Gaussian fits

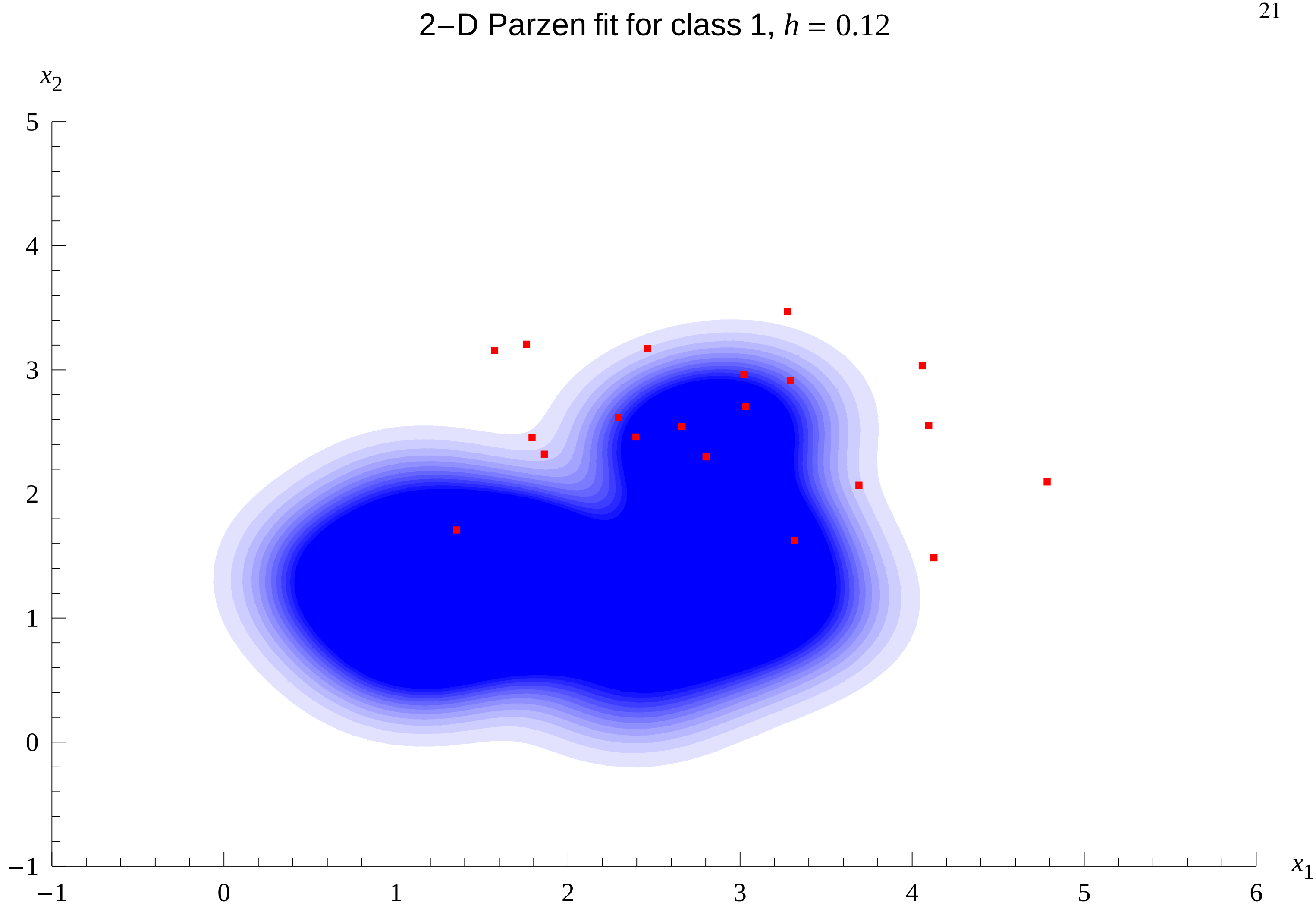


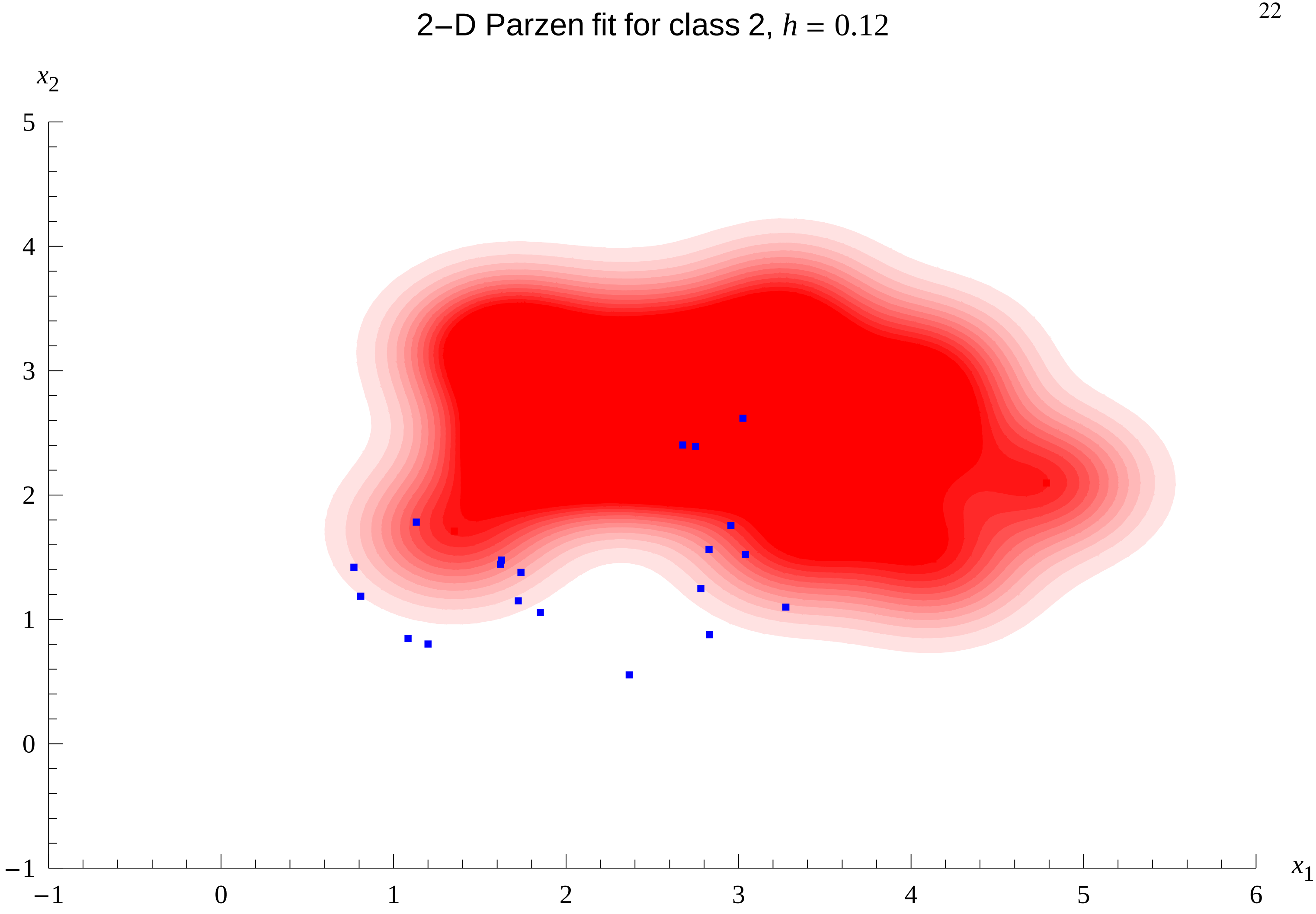

'Two Moons' data for two−class classification problem
x_2
5
4
3
2
1
0
−1
−1
0
1
2
3
4
5
6
x_1


2−D Gaussian fit for class 1
x_2
5
4
3
2
1
0
−1
−1
0
1
2
3
4
5
6
x_1


2−D Gaussian fit for class 2
x_2
5
4
3
2
1
0
−1
−1
0
1
2
3
4
5
6
x_1

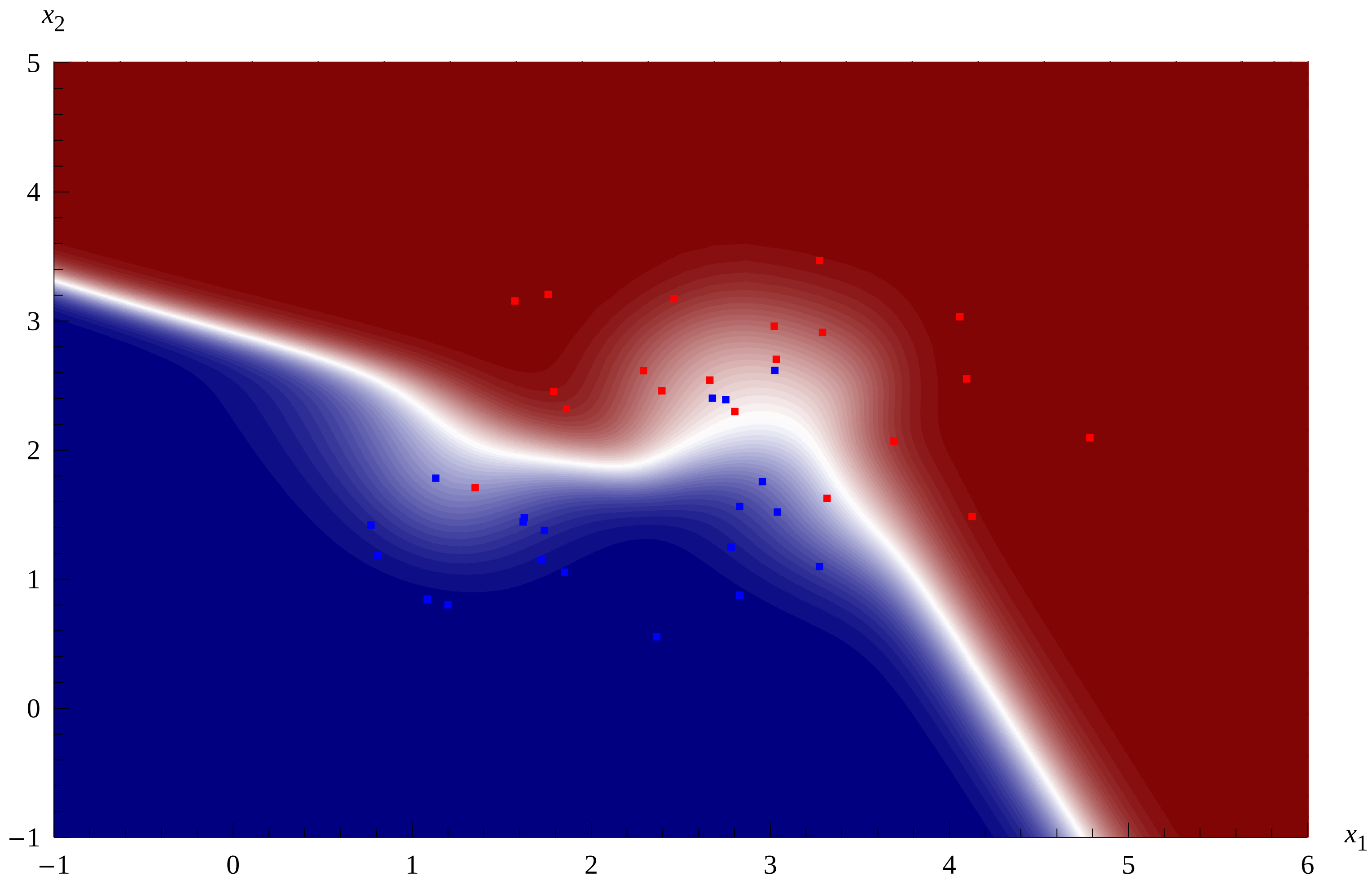

Discriminant function with Gaussian fits
x_2
5
4
3
2
1
0
−1
−1
0
1
2
3
4
5
6
x_1

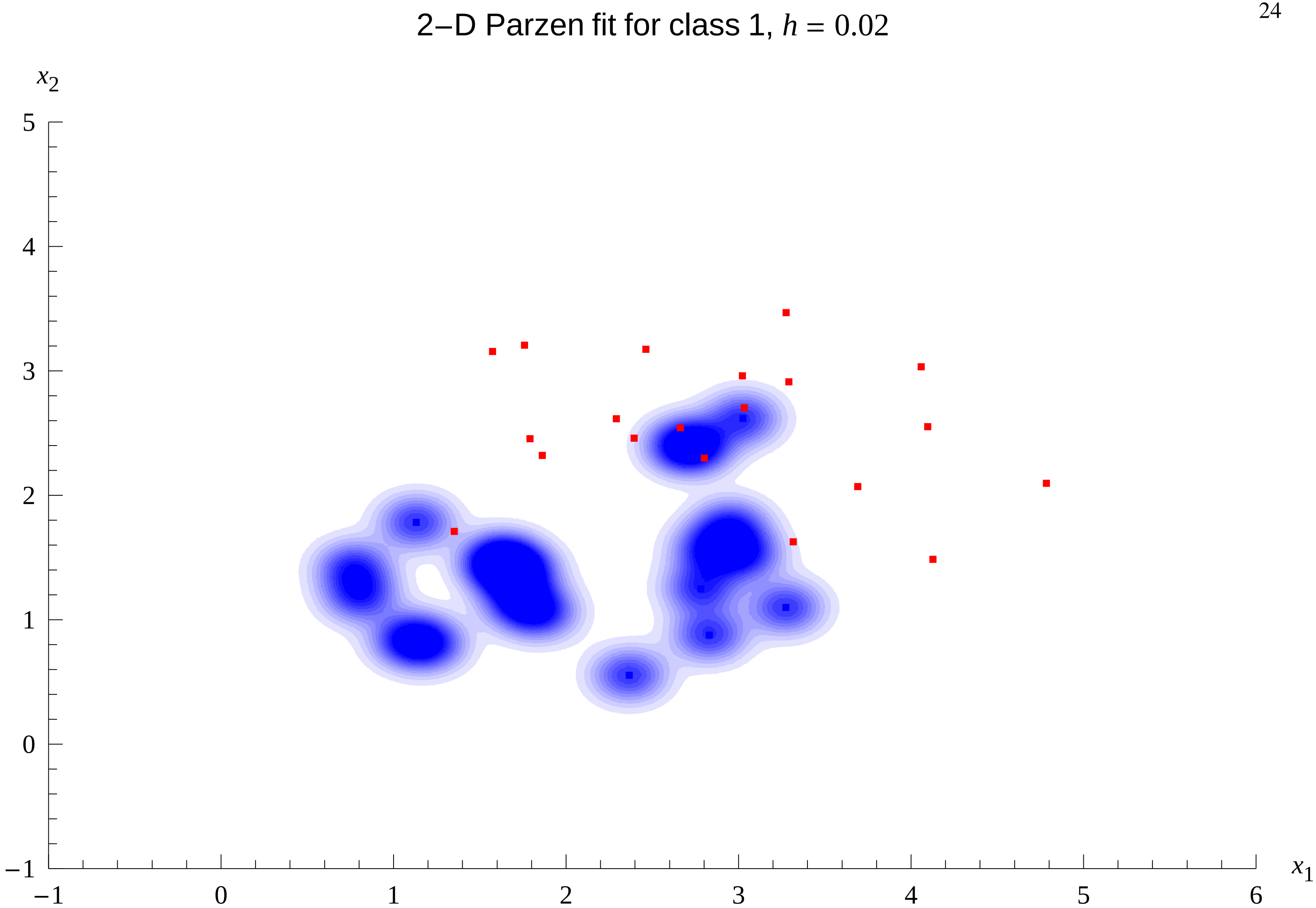

2−D Parzen fit for class 1, $h = 0.12$
x_2
5
4
3
2
1
0
−1
−1
0
1
2
3
4
5
6
x_1

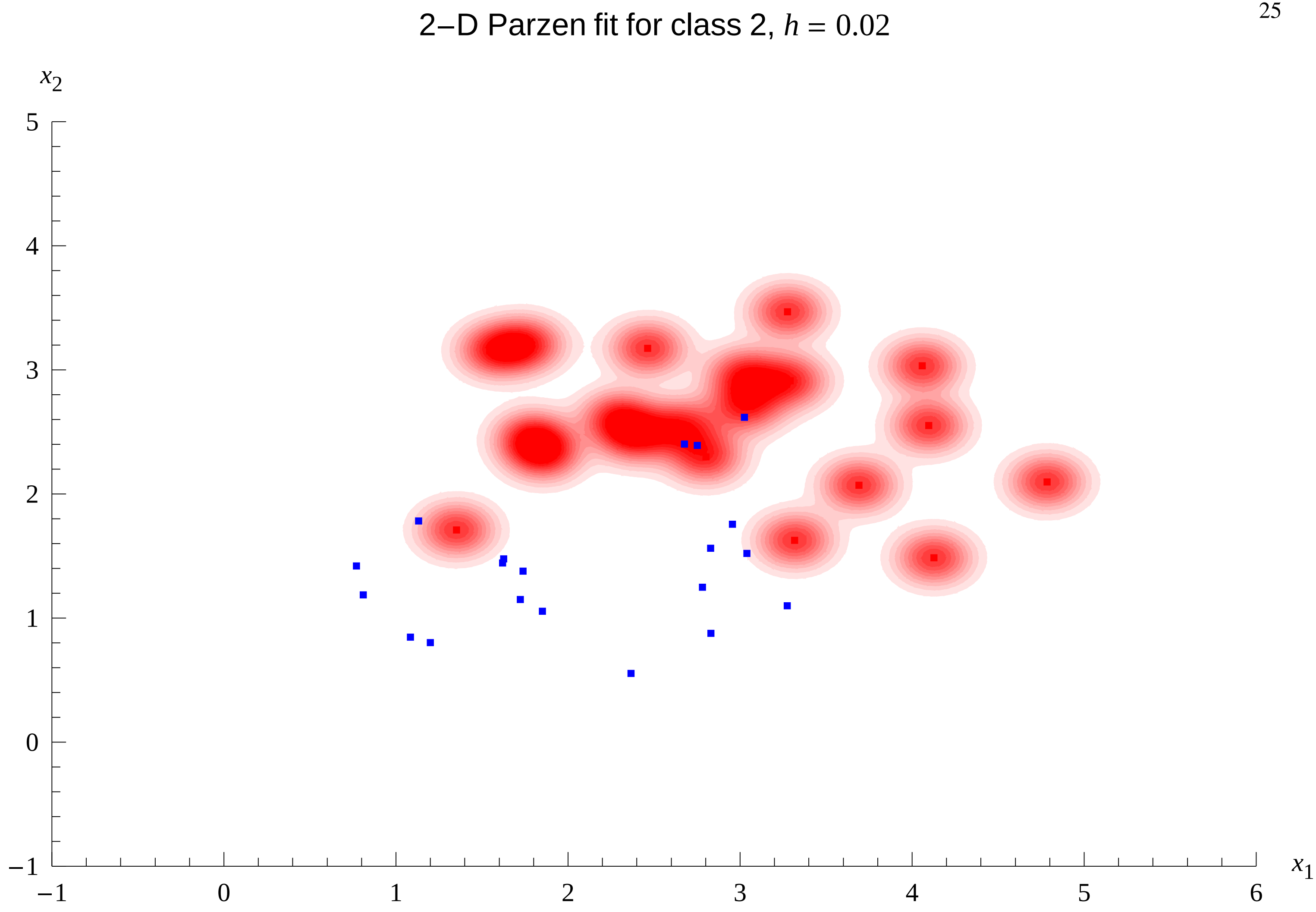


2−D Parzen fit for class 2, $h = 0.12$
x_2
5
4
3
2
1
0
−1
−1
0
1
2
3
4
5
6
x_1

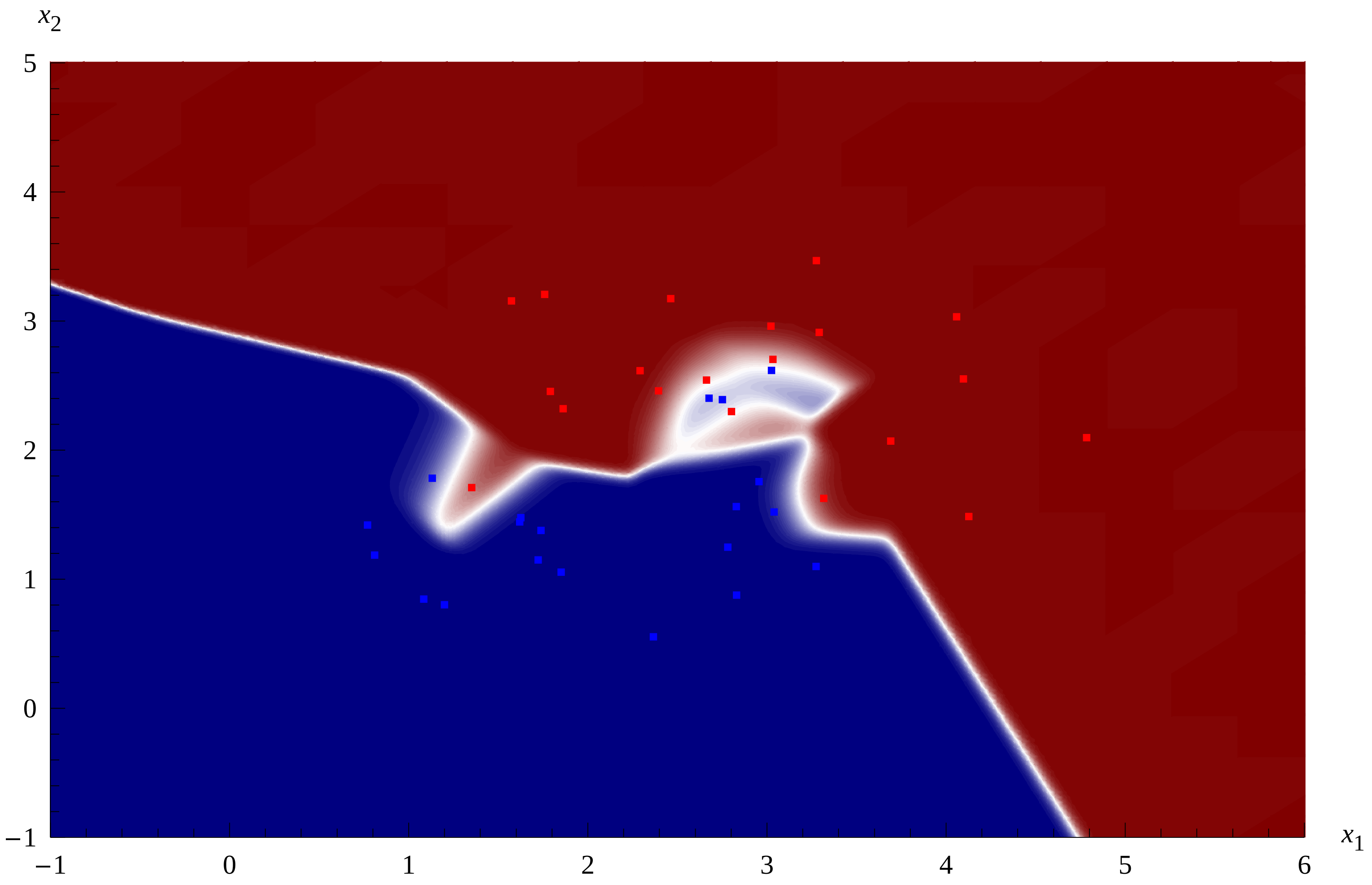
Discriminant function with Parzen fits, $h = 0.12$

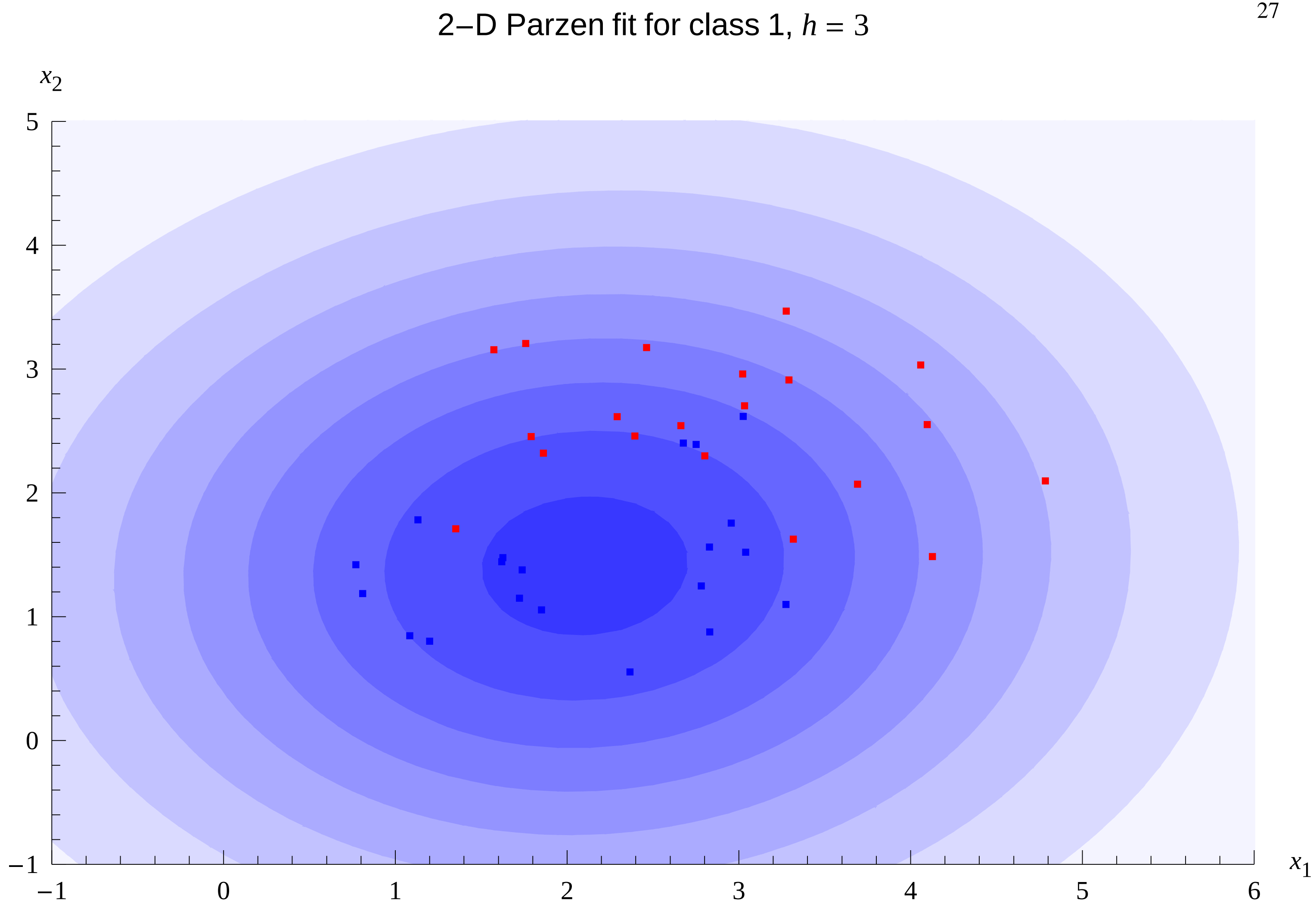



2−D Parzen fit for class 1, h = 0.02
x_2
5
4
3
2
1
0
−1
−1
0
1
2
3
4
5
6
x_1

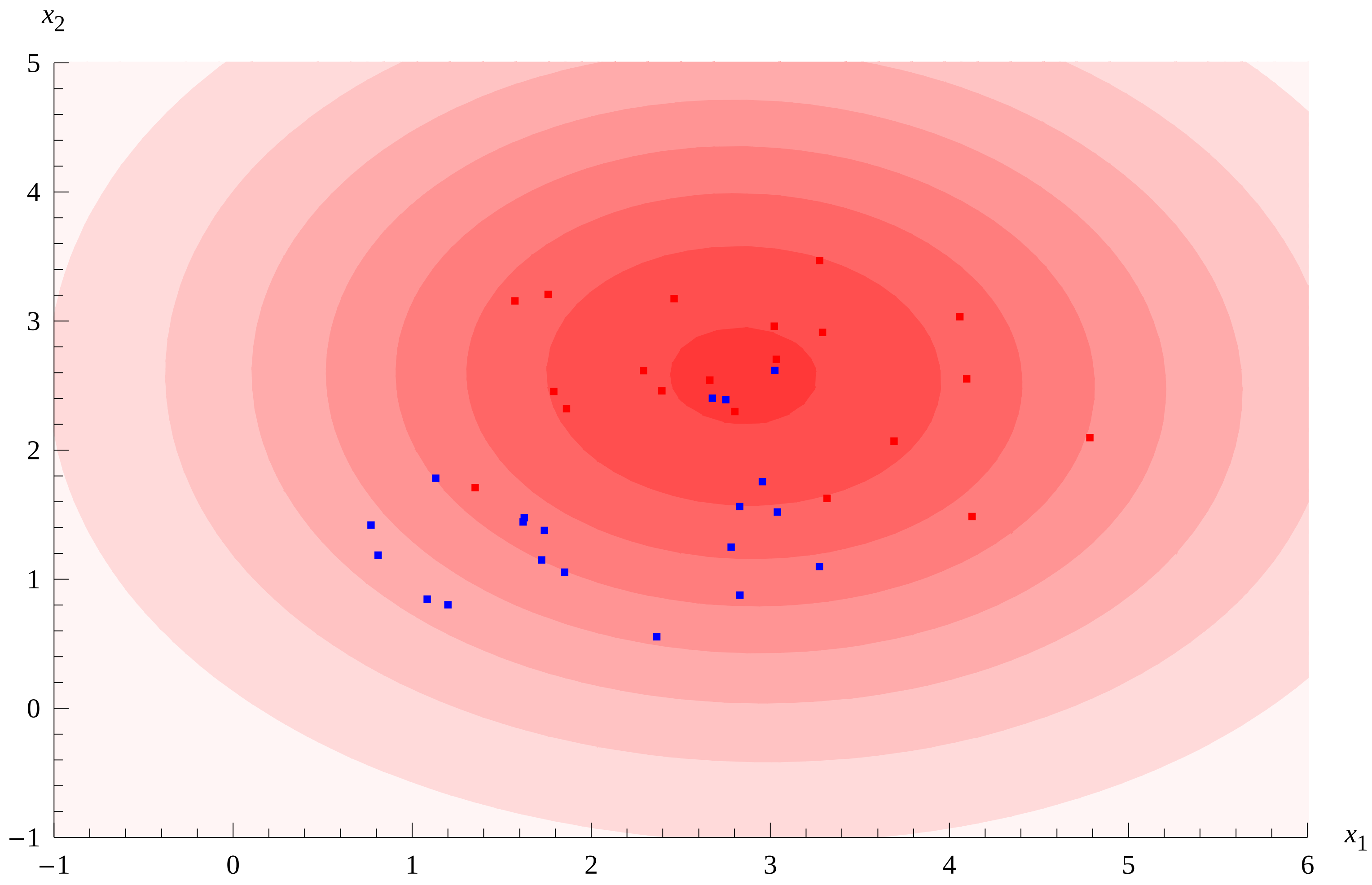

2−D Parzen fit for class 2, h = 0.02
x_2
5
4
3
2
1
0
−1
−1
0
1
2
3
4
5
6
x_1

Discriminant function with Parzen fits, $h = 0.02$

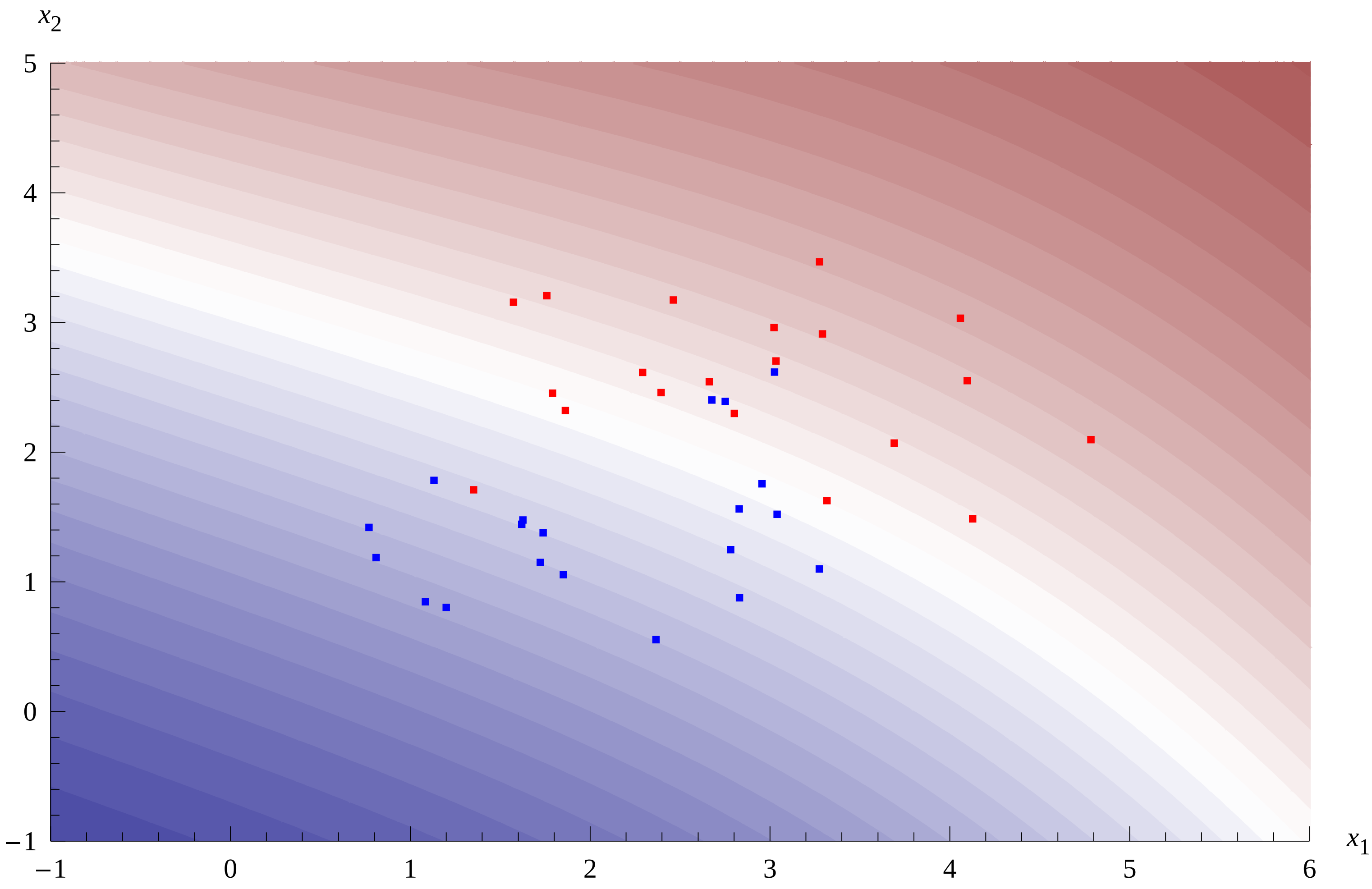


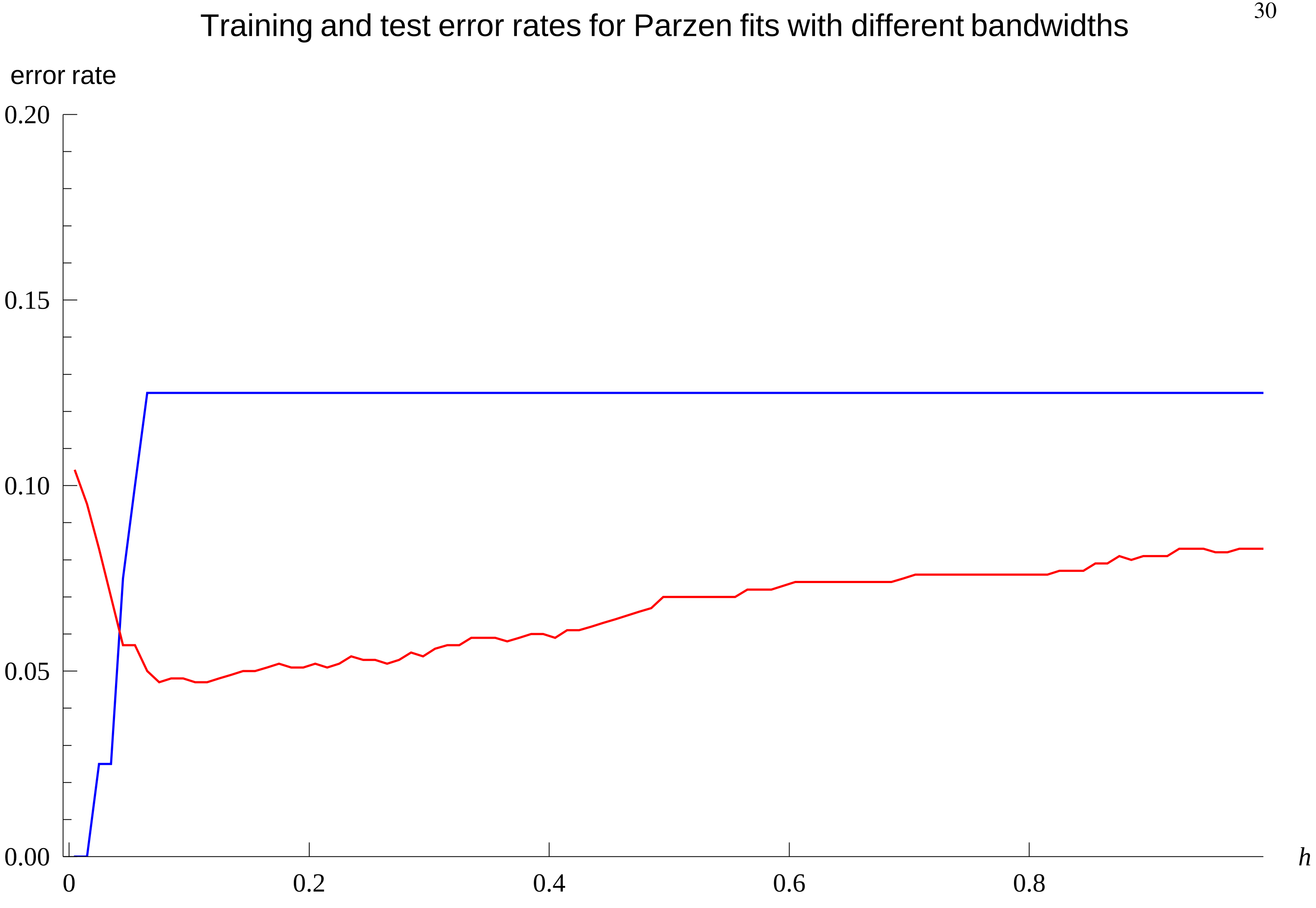

2–D Parzen fit for class 1, h = 3
x_2
5
4
3
2
1
0
−1
−1
0
1
2
3
4
5
6
x_1

2–D Parzen fit for class 2, $h = 3$



Discriminant function with Parzen fits, $h = 3$





Training and test error rates for Parzen fits with different bandwidths
error rate
0.20
0.15
0.10
0.05
0.00
0
0.2
0.4
0.6
0.8
h

Non-parametric fitting

- Capacity control, regularization
  - trade-off between approximation error and estimation error
  - complexity grows with data size
  - no need to correctly guess the function class

Histoire

- Algorithmes
  - 1958 : Perceptron [Rosenblatt, ’58] – [Minsky–Papert ’69]
  - 1986 : Réseaux de neurones et l’algorithme de rétro-propagation [Rumelhart–Hinton–Williams, ’86]
  - 1995: Machines à vecteurs de support [Boser–Guyon–Vapnik, ’92], [Cortes–Vapnik, ’95]
  - 1997: boosting, AdaBoost [Freund, ’95], [Freund–Schapire, ’97]

Méthodes

- Classification linéaire
  - Perceptron
  - Machine à vecteurs de support (SVM) linéaire
- Classification non-linéaire
  - Perceptron multi-couche (réseaux de neurones)
  - Plus proche voisin
  - Arbres de décision
  - SVM
  - Boosting

Le plus proche voisin

- Le plus proche voisin:

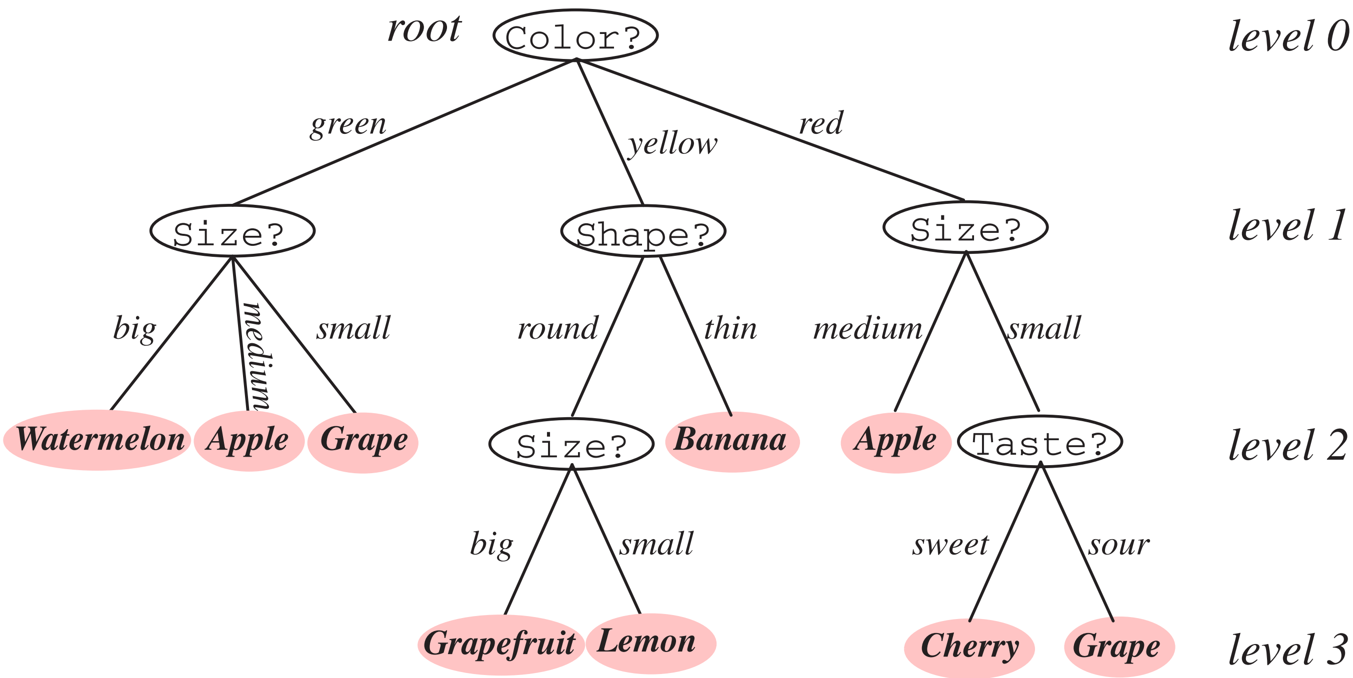
$$f(\mathbf{x}) = y_{i_\mathbf{x}}$$

ou

$$i_\mathbf{x} = \arg\min_{i:1,\ldots,n} \mathrm{d}(\mathbf{x}, \mathbf{x}_\mathrm{i})$$

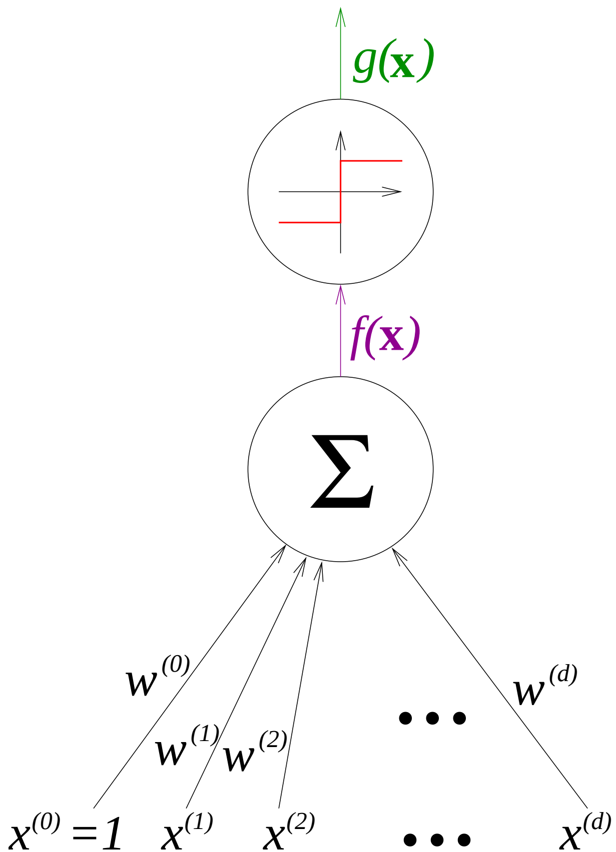
  - très simple
  - problèmes algorithmiques et statistiques

Arbres de décision



Le perceptron

- Fonctions discriminantes linéaires : $f(\mathbf{x}) = \sum_{i=0}^{d} w^{(i)} \cdot x^{(i)} = \langle \mathbf{w}, \mathbf{x} \rangle$



Fonctions discriminantes linéaires généralisées

- Modèle :

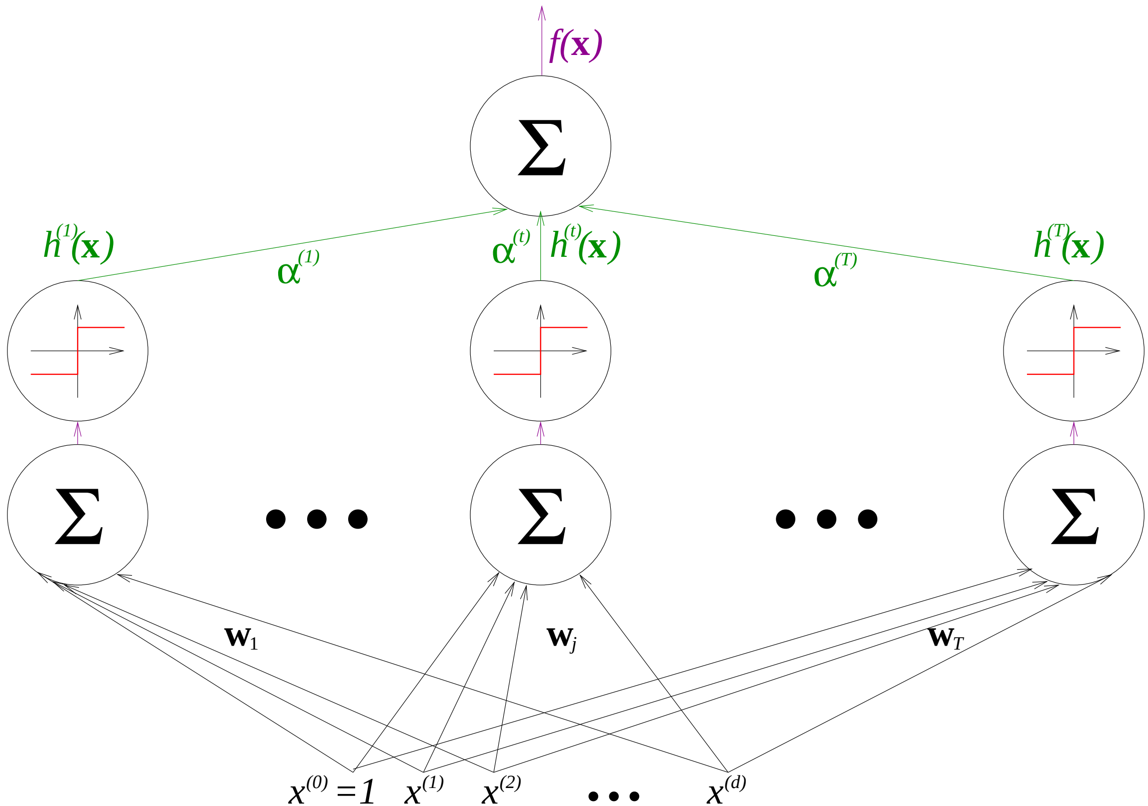
$$f(\mathbf{x}) = \sum_{j=1}^{N} \alpha^{(j)} h^{(j)}(\mathbf{x})$$

- $h^{(j)} : \mathbb{R}^d \rightarrow [-1, 1]$
  - classifieurs/fonctions discriminantes simples, traits, experts

- $\alpha^{(j)} \in \mathbb{R}^+$
  - poids de l'expert $h^{(j)}$ dans le vote final

Le perceptron multi-couche

- Modèle : $f(\mathbf{x}) = \sum_{j=1}^{N} \alpha^{(j)} \sigma(\langle \mathbf{w}_j, \mathbf{x} \rangle)$



La machine à vecteur de support

- Modèle :

$$f(\mathbf{x}) = \sum_{j \in I_{\text{SV}}} \alpha^{(j)} y_j K(\mathbf{x}_j, \mathbf{x})$$

  - $I_{\text{SV}} \subset \{1, \ldots, n\}$ est l'ensemble de vecteurs de support
  - $K(\cdot, \cdot)$ est une fonction de similarité (noyau)
  - But : une frontière équidistante des classes
  - "plus proche voisin sophistiqué"

La machine à vecteur de support

- Modèle :

$$f(\mathbf{x}) = \sum_{j \in I_{\mathrm{SV}}} \alpha^{(j)} y_j K(\mathbf{x}_j, \mathbf{x})$$

- Noyaux :
  - $K(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle \longrightarrow f(\mathbf{x})$ est linéaire
  - $K(\mathbf{x}, \mathbf{x}') = \left(1 + \langle \mathbf{x}, \mathbf{x}' \rangle\right)^d \longrightarrow f(\mathbf{x})$ est un polynôme de degré d
  - $K(\mathbf{x}, \mathbf{x}') = \exp\left(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2\right) \longrightarrow f(\mathbf{x})$ est un mélange de gaussiens

AdaBoost

- Modèle :

$$f(\mathbf{x}) = \sum_{j=1}^{N} \alpha^{(j)} h^{(j)}(\mathbf{x})$$

  - aucune restriction sur la forme de $h^{(j)}(\mathbf{x})$
  - souvent des “decision stumps” :

$$h_{\ell,\theta}(\mathbf{x}) = \begin{cases} +1 & \text{si } x^{(\ell)} \geq \theta, \\ -1 & \text{sinon} \end{cases}$$

  où $\mathbf{x} = \left(x^{(1)}, \ldots, x^{(d)}\right)$

AdaBoost

- L'intuition de l'algorithme
  - ajouter un expert (stump) à la fois
  - ajouter le meilleur sur les points d'entraînement mal-classifiés par des experts précédents
  - mettre le poids de l'expert choisi proportionnel à son erreur
  - à la fin, retourner le vote pondéré

AdaBoost

- Pondération sur les points d'entraînement $w_1, \ldots, w_n$
  - pondération normalisée : $\sum_{i=1}^{n} w_i = 1$
  - initialisée uniformément : $\mathbf{w} = (1/n, \ldots, 1/n)$
  - si $\mathbf{x}_i$ est mal-classifié par $h^{(j)}$, augmenter w_i
  - sinon, diminuer w_i
  - au fur et à mesure les points "difficiles" obtiennent des poids élevés

AdaBoost

- Choix d'expert
  - experts binaires : $h^{(j)} : \mathcal{X} \rightarrow \{-1, 1\}$
  - erreur pondérée

$$\varepsilon = \varepsilon(h) = \sum_{i=1}^{n} w_i \mathbb{I}\{h(\mathbf{x}_i) \neq y_i\}$$

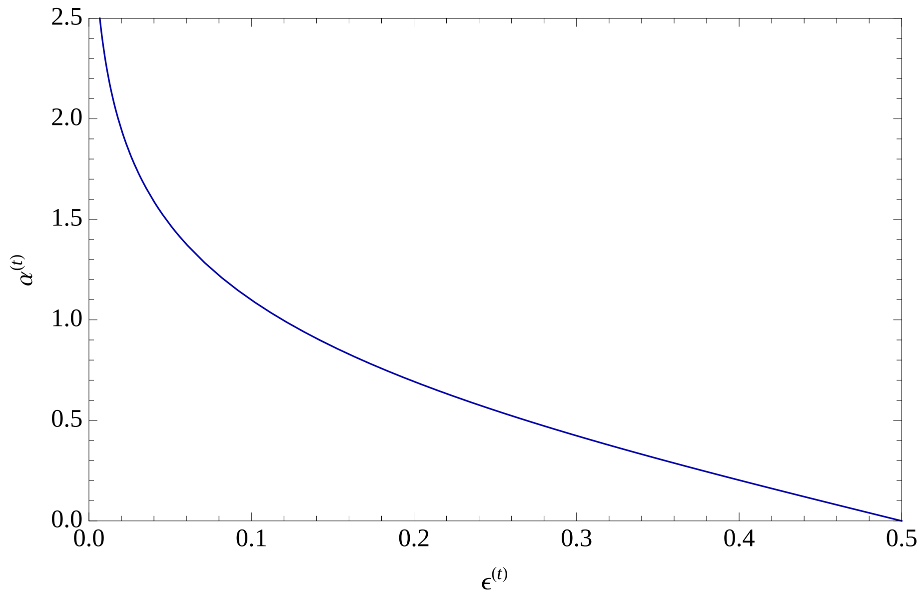
  - choisir l'expert qui minimise l'erreur pondérée

$$h^{(t)} = \arg\min_h \varepsilon(h)$$

AdaBoost

- Choix d'expert
  - mettre son poids à

$$\alpha^{(t)} = \frac{1}{2}\ln\frac{1-\varepsilon}{\varepsilon}$$



AdaBoost

- Re-pondération des points
  - si $h(\mathbf{x}_i) \neq y_i$ alors $w_i \leftarrow w_i \frac{1}{2\varepsilon}$
  - si $h(\mathbf{x}_i) = y_i$ alors $w_i \leftarrow w_i \frac{1}{2(1-\varepsilon)}$

AdaBoost

$\textsc{AdaBoost}(D_n, \textsc{Base}(D_n, \mathbf{w}), T)$

$\mathbf{w}^{(1)} \leftarrow (1/n, \ldots, 1/n)$ ▷ *poids initiaux*

pour $t \leftarrow 1$ **à** T

  $h^{(t)} \leftarrow \textsc{Base}(D_n, \mathbf{w})$ ▷ *choix d'expert*

  $\varepsilon^{(t)} \leftarrow \sum_{h^{(t)}(\mathbf{x}_i) \neq y_i} w_i$ ▷ *erreur pondérée*

  $\alpha^{(t)} \leftarrow \frac{1}{2} \ln \left(\frac{1-\varepsilon^{(t)}}{\varepsilon^{(t)}} \right)$ ▷ *poids de* $h^{(t)}$

  pour $i \leftarrow 1$ **à** n ▷ *re-pondération des points*

    si $h^{(t)}(\mathbf{x}_i) \neq y_i$ **alors**

      $w_i^{(t+1)} \leftarrow \frac{w_i^{(t)}}{2\varepsilon^{(t)}}$

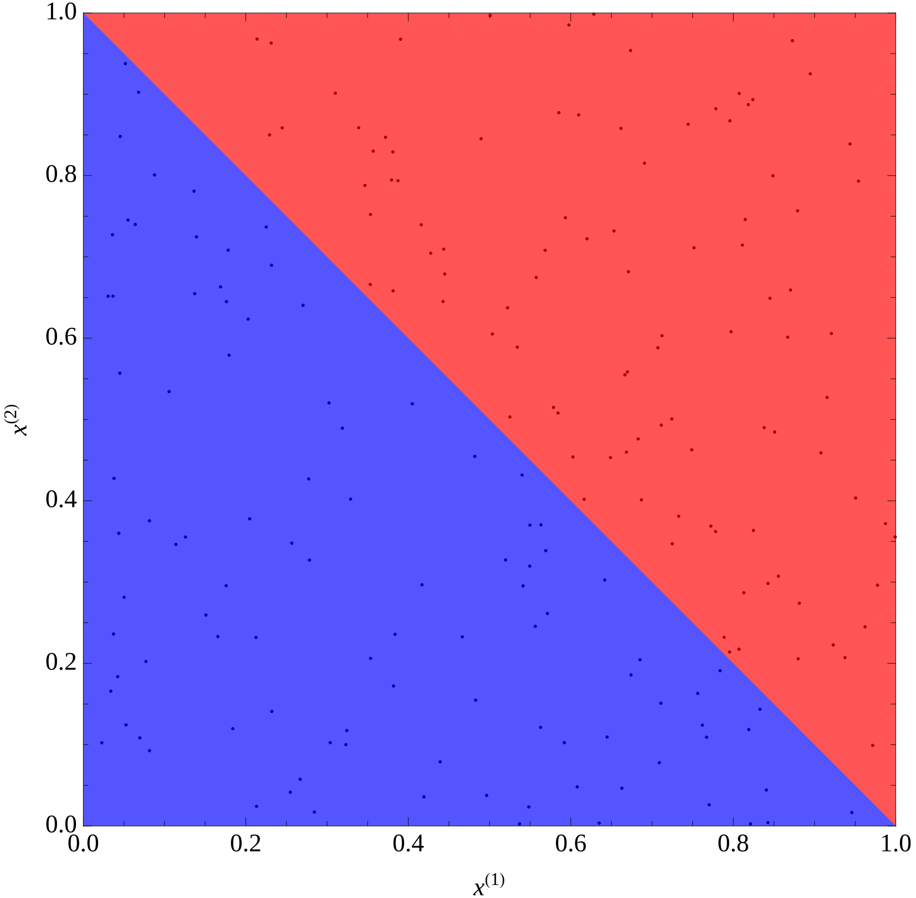
    sinon

      $w_i^{(t+1)} \leftarrow \frac{w_i^{(t)}}{2(1-\varepsilon^{(t)})}$

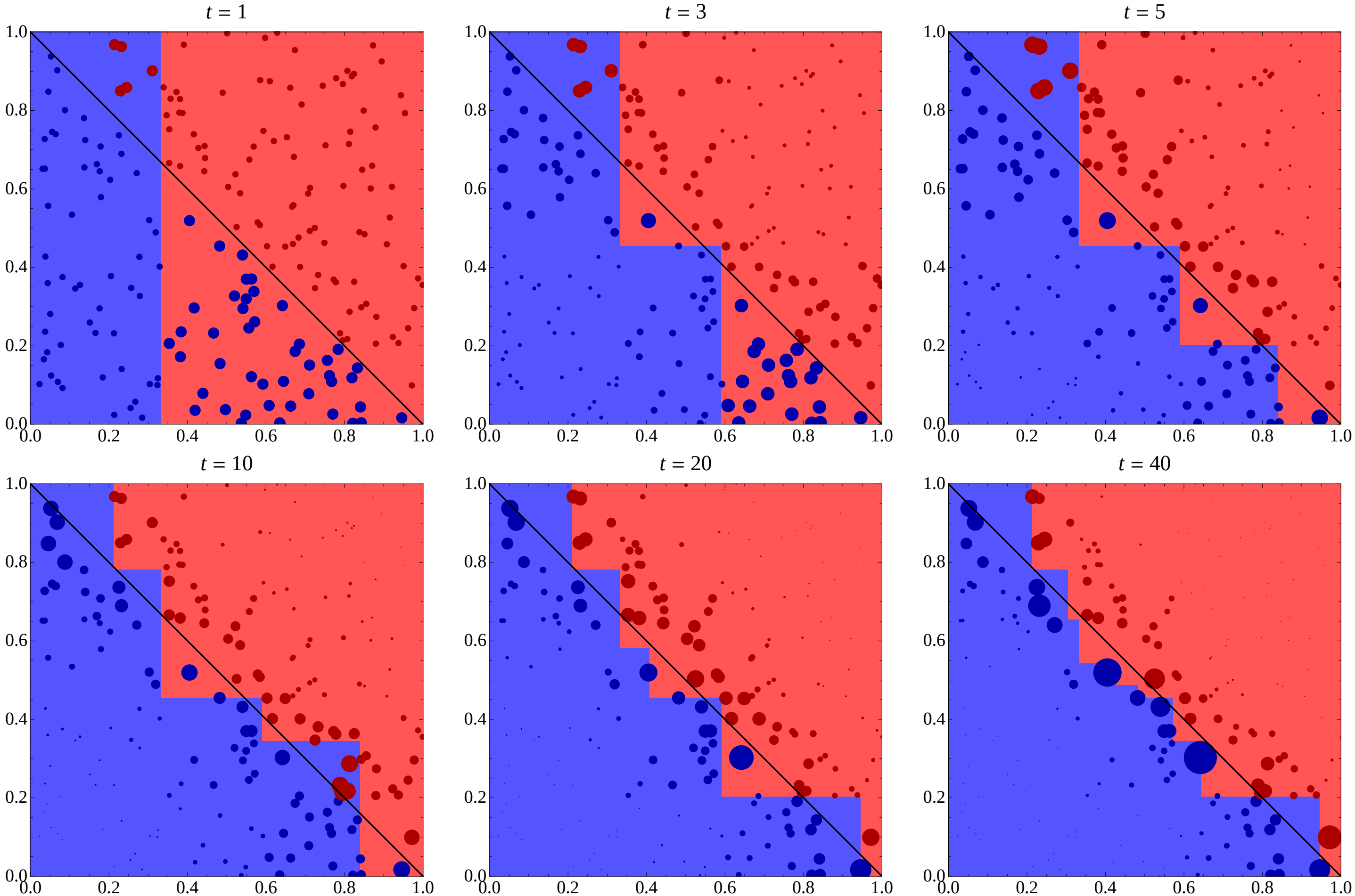
retourner $f^{(T)}(\cdot) = \sum_{t=1}^{T} \alpha^{(t)} h^{(t)}(\cdot)$

AdaBoost

- Données:

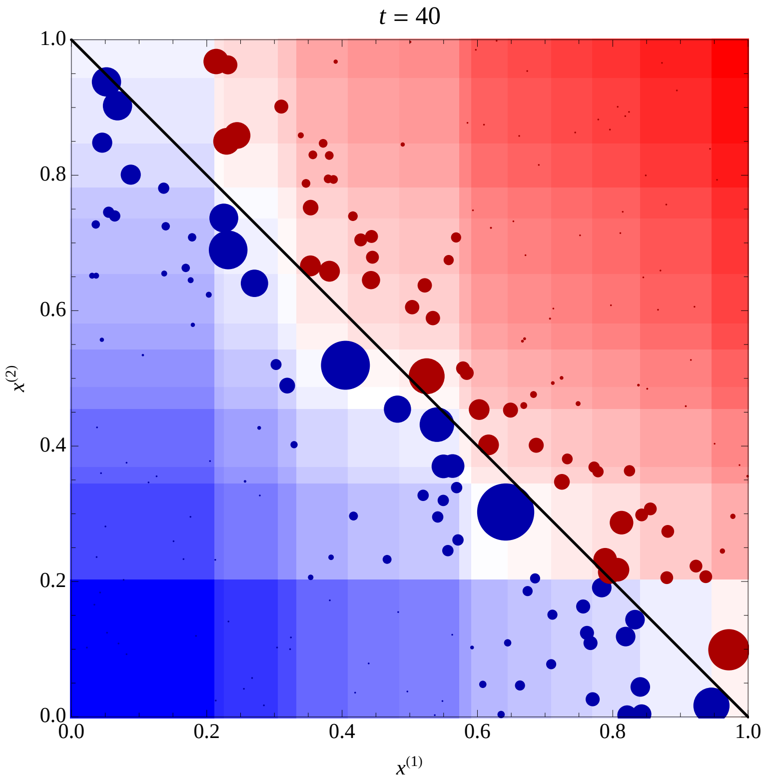


AdaBoost



AdaBoost

- Marges:



AdaBoost

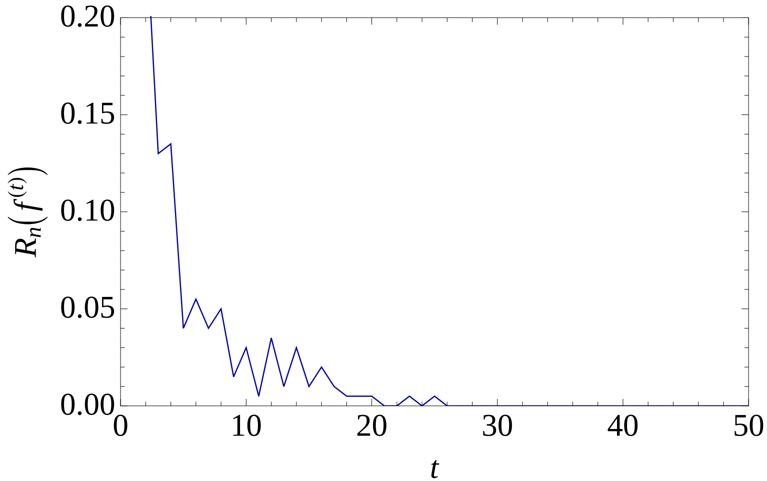
- Convergence
  - soit $\varepsilon \leq \frac{1}{2} - \delta$ dans toutes les itérations : h est un peu meilleur qu'une décision aléatoire
  - l'erreur d'entraînement

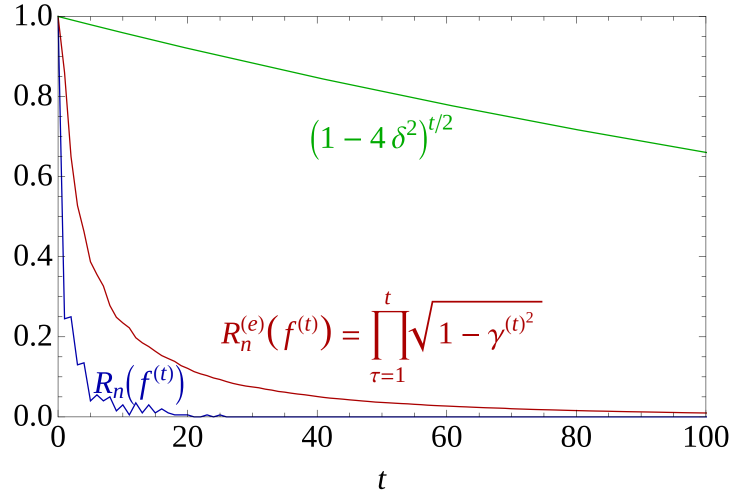
$$\widehat{R}(f) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}\{f(\mathbf{x}_i)y_i < 0\}$$

  est 0 après $\left\lceil \frac{\ln n}{2\delta^2} \right\rceil + 1$ itérations

AdaBoost

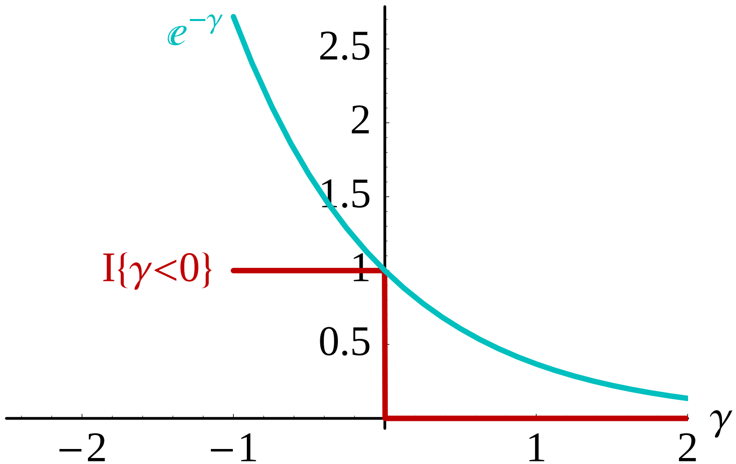
- Convergence





AdaBoost

- Justification des formules :
  - minimiser un coût exponentiel sur la marge $\gamma = f(\mathbf{x})y$
  - descente de gradient dans l'espace fonctionnel des experts



AdaBoost

- Extension 1 : experts ternaires : $h^{(j)} : \mathcal{X} \rightarrow \{-1, 0, 1\}$
  - experts peuvent s'abstenir
  - experts locaux
  - experts spécialistes
- Extension 2 : experts avec confiances : $h^{(j)} : \mathcal{X} \rightarrow [-1, 1]$
  - $\mathrm{signe}(h^{(j)}(\mathbf{x}))$ est la classification de $\mathbf{x}$ par l'expert
  - $|h^{(j)}(\mathbf{x})|$ est la confiance de l'expert

Multi-classe

- Approches générales
  - one-against-all
  - one-against-one
- Fonction discriminante de multi-classe: $\mathbf{f} : \mathbf{x} \to \mathbb{R}^K$
  - notation : $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_K(\mathbf{x}))$
- Classifieur de multi-classe:

$$g(\mathbf{x}) = \arg\max_{\ell \in \{1,\ldots,K\}} f_\ell(\mathbf{x})$$

Multi-classe

- Experts binaires de multi-classe : $\mathbf{h}\ : \mathbf{x} \rightarrow \{-1, 1\}^K$
  - Convertir un expert binaire ϕ à multi-classe:

$$\mathbf{h}(\mathbf{x}) = \mathbf{v} \cdot \phi(\mathbf{x})$$

  - $\mathbf{v} \in \{-1, 1\}^K$ est le vecteur de votes

- Étiquettes de multi-classe : $\mathbf{y} = \{-1, \ldots, -1, 1, -1, \ldots, -1\}$

$$y_\ell = \begin{cases} 1 & \text{si } \ell \text{ est la vraie classe } \ell(\mathbf{x}), \\ -1 & \text{sinon} \end{cases}$$

Multi-classe

- Erreurs de multi-classe :

$$\widehat{R}(\mathbf{f}) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}\left\{\ell(\mathbf{x}_i) \neq \widehat{\ell}(\mathbf{x}_i)\right\}$$

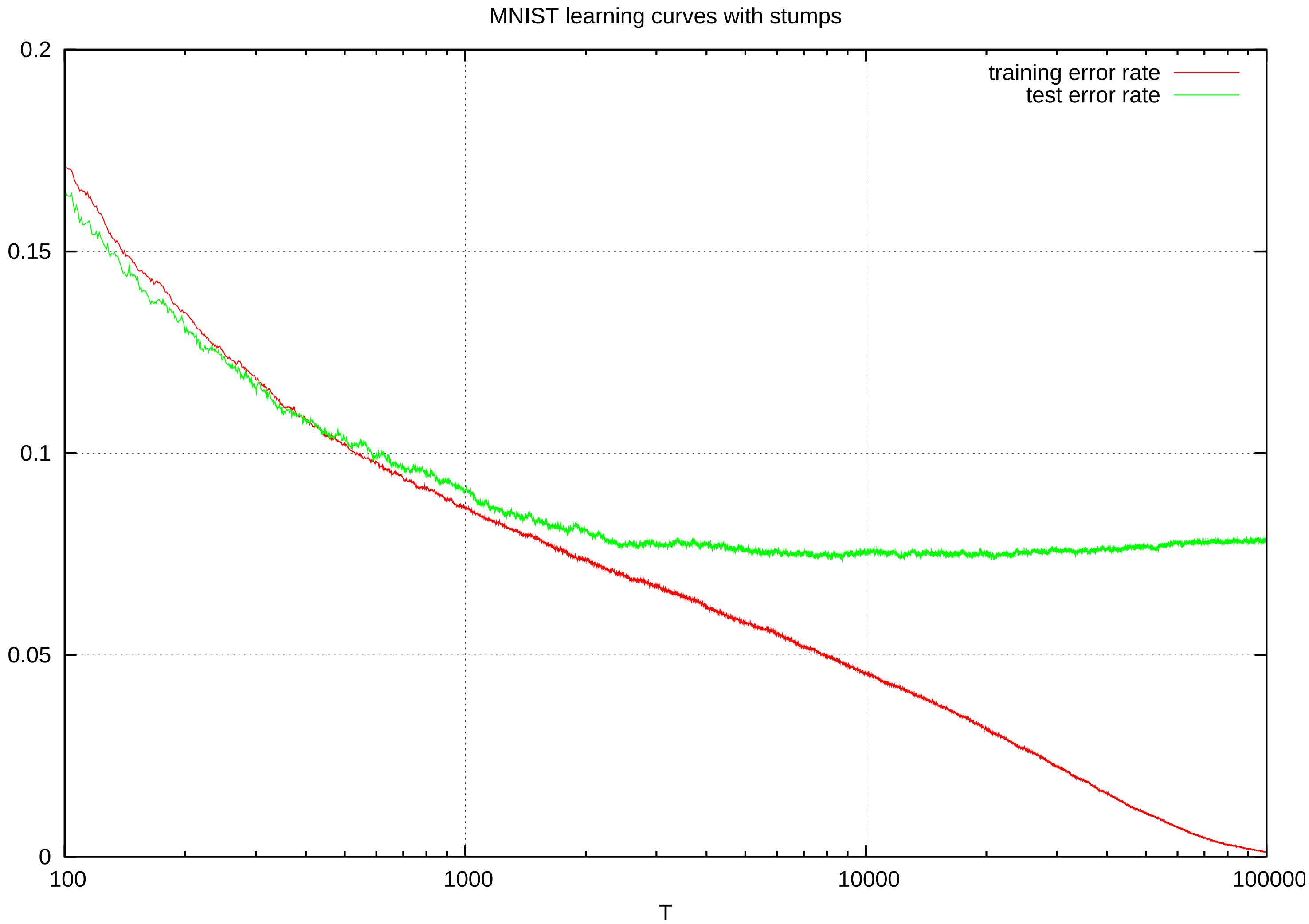
$$\widehat{R}\left(\mathbf{f}, \mathbf{w}^{(1)}\right) = \sum_{i=1}^{n}\sum_{\ell=1}^{K} w_{i,\ell}^{(1)} \mathbb{I}\left\{f_\ell(\mathbf{x}_i)y_\ell < 0\right\}$$

  - par exemple

$$w_{i,\ell}^{(1)} = \begin{cases} \frac{1}{2n} & \text{si } \ell \text{ est la vraie classe (si } y_{i,\ell} = 1\text{),} \\ \frac{1}{2n(K-1)} & \text{sinon (si } y_{i,\ell} = -1\text{).} \end{cases}$$

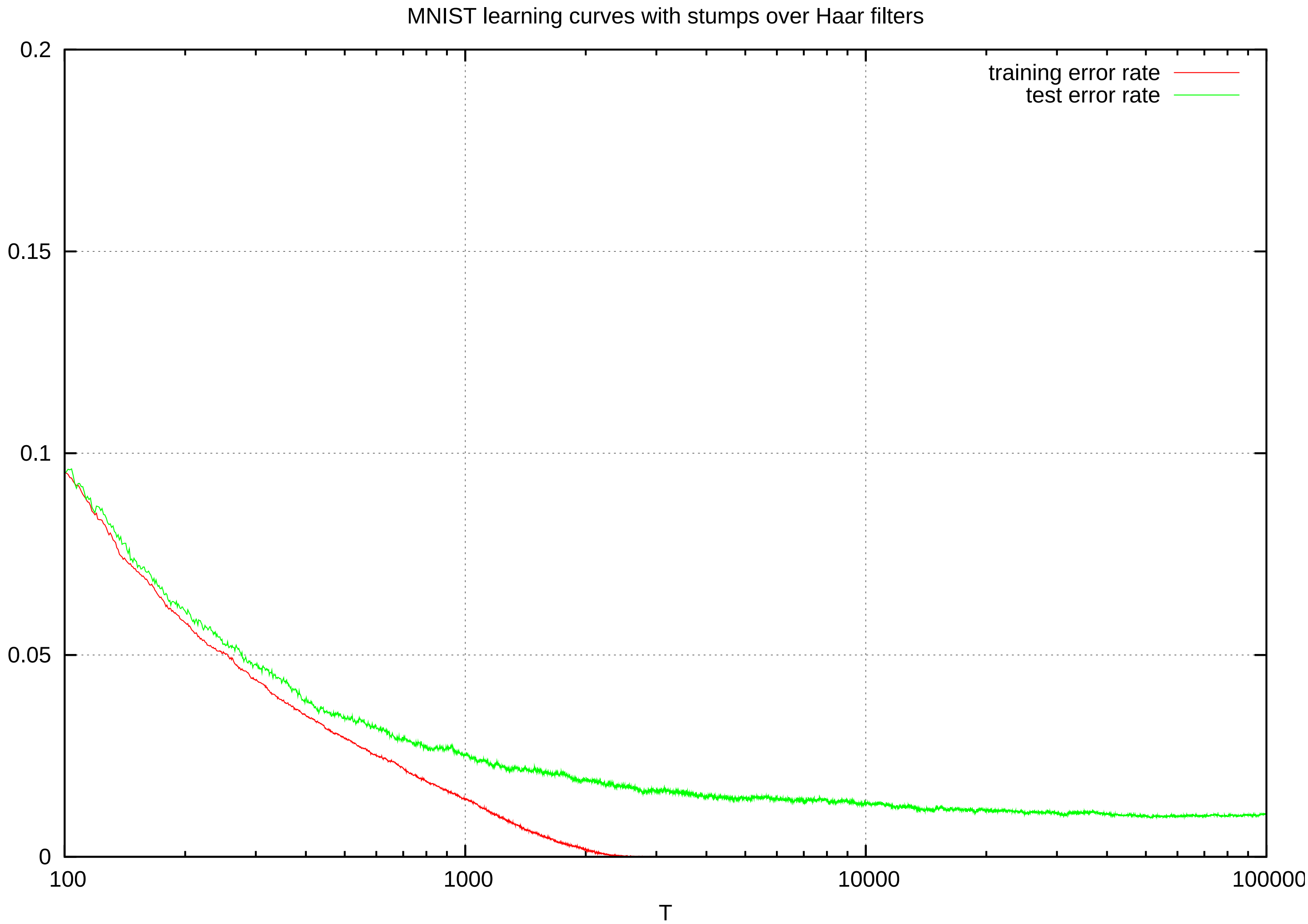
Multi-classe

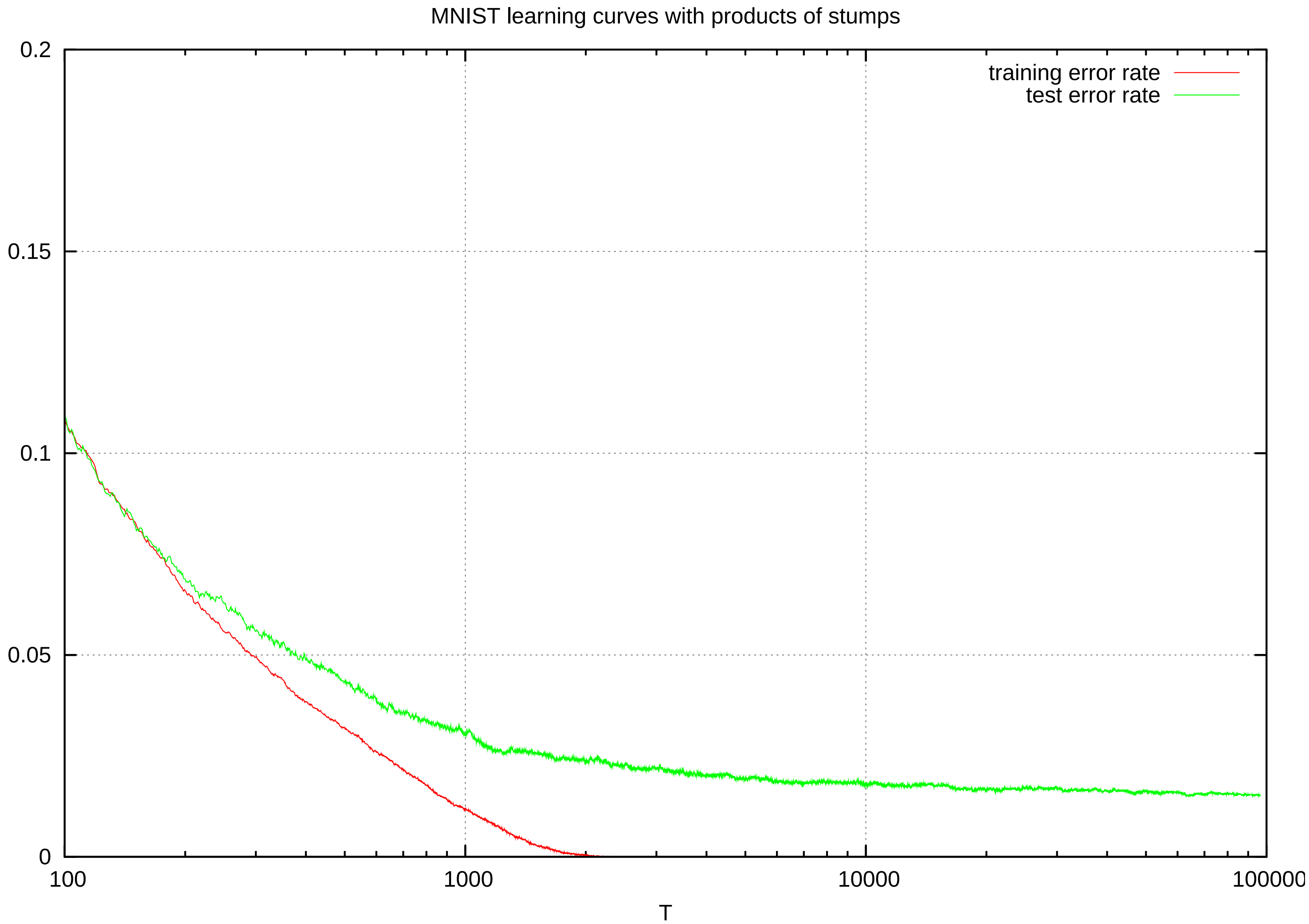
- AdaBoost
  - pondération sur les points d'entraînement et sur les classes :
$$\left\{w_{i,\ell}\right\}_{i=1,\ldots n}^{\ell=1,\ldots,K}$$
  - $w_{i,\ell}$ est initialisé à $w_{i,\ell}^{(1)}$
  - trouver le meilleur stump $\phi(\mathbf{x})$ et le meilleur vecteur de votes $\mathbf{v}$ reste efficace ($O(nK)$)

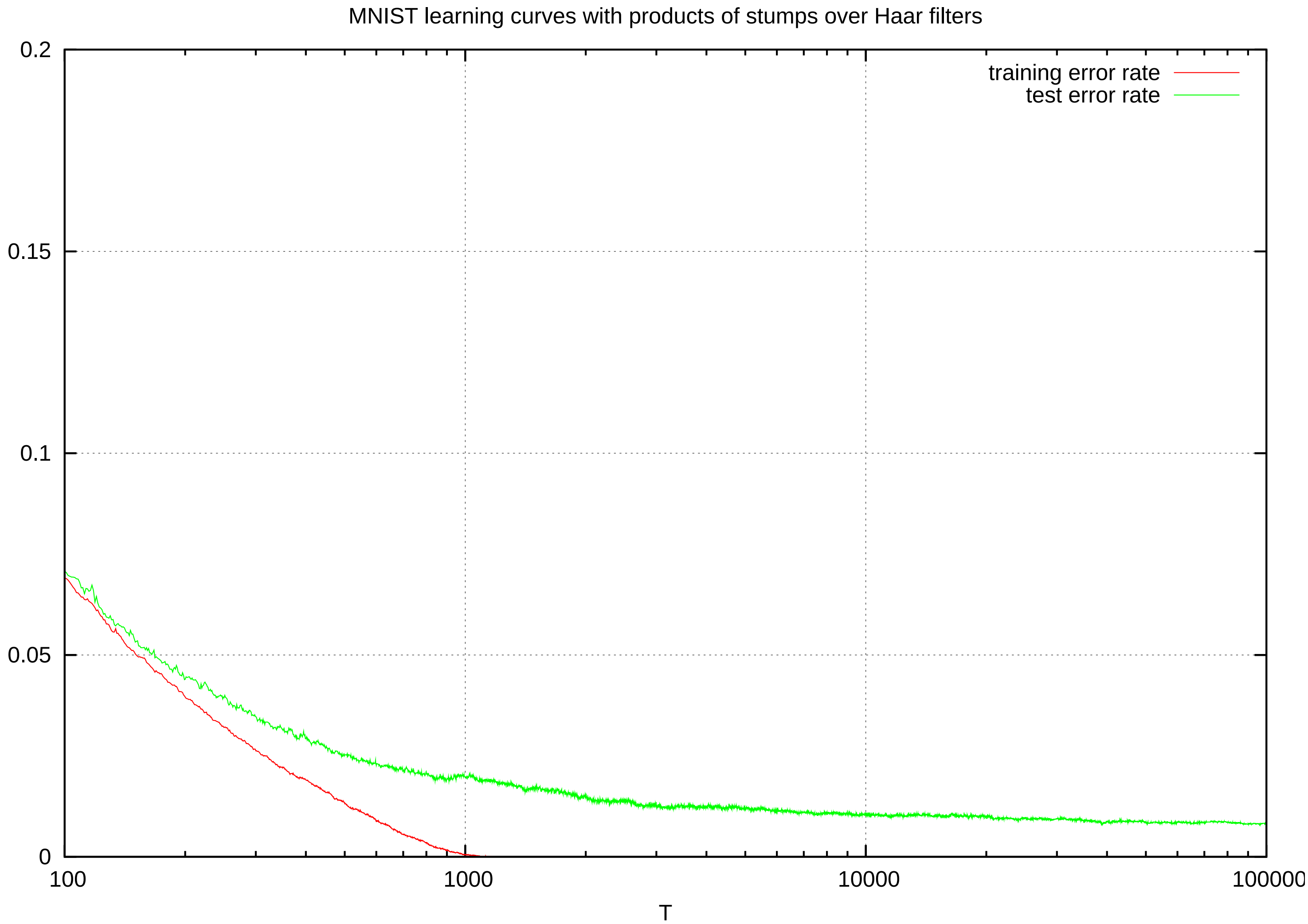

MNIST learning curves with stumps
training error rate
test error rate
0.2
0.15
0.1
0.05
0
100
1000
10000
100000
T

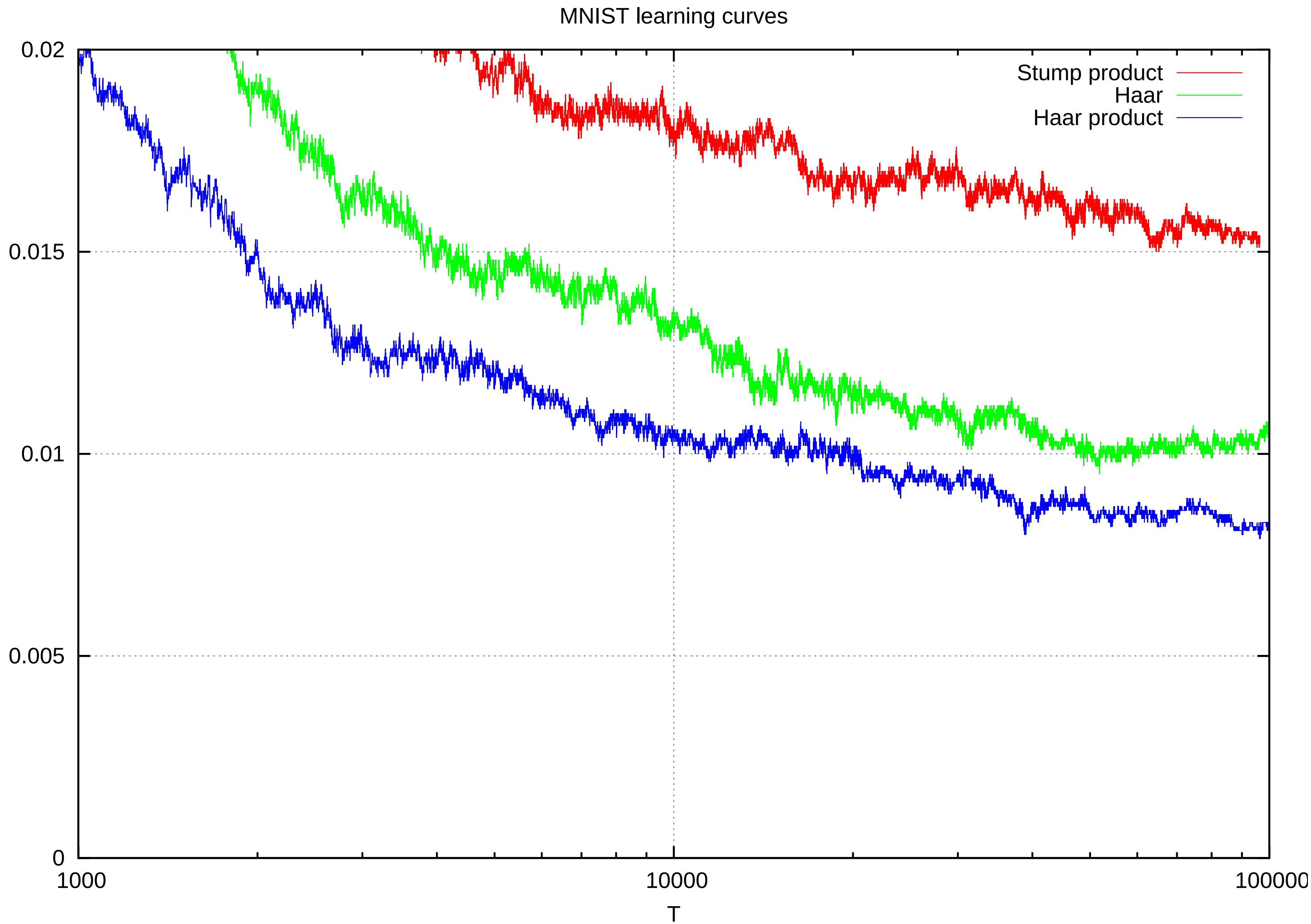
Meilleur experts

- Experts génériques plus puissants
  - arbres
  - produits
- Experts spécifiques aux applications
  - filtres, extraction des traits


MNIST learning curves with stumps over Haar filters
training error rate
test error rate
0.2
0.15
0.1
0.05
0
100
1000
10000
100000
T


MNIST learning curves with products of stumps
training error rate
test error rate
0.2
0.15
0.1
0.05
0
100
1000
10000
100000
T


MNIST learning curves with products of stumps over Haar filters
training error rate
test error rate
0.2
0.15
0.1
0.05
0
100
1000
10000
100000
T


MNIST learning curves
Stump product
Haar
Haar product
0.02
0.015
0.01
0.005
0
1000
10000
100000
T

AdaBoost

- Avantages
  - apprentissage extrêmement simple, pas de descente de gradient, pas d'optimisation numérique compliquée
  - rapide
  - interprétation intuitive : vote pondéré des experts
  - le choix du pool d'experts “capsule” les connaissances à priori
  - aucune restriction sur la forme des experts
  - extension naturelle à la classification multi-classe
- Désavantages
  - pas très bon sur des étiquettes bruitées
  - l'extension à la régression est un peu forcée