Software development in AppStat

AppStat: Applied Statistics and Machine Learning

AppStat: Apprentissage Automatique et Statistique Appliquée

Balázs Kégl

Linear Accelerator Laboratory, CNRS/University of Paris Sud

Service Informatique

Nov 30, 2010

Overview

- Introduction
  - me
  - the team
  - collaborations
- Scientific projects → software
  - discriminative learning → boosting → `multiboost.org`
  - inference, Monte-Carlo integration → adaptive MCMC → integration into root (save it for next time)

Scientific path

Hungary	*1989 – 94*	M.Eng. Computer Science	BUTE
	1994 – 95	research assistant	BUTE
Canada	*1995 – 99*	Ph.D. Computer Science	Concordia U
	2000	postdoc	Queen's U
	2001 – 06	assistant professor	U of Montreal
France	*2006 –*	research scientist (CR1)	CNRS / U Paris Sud

- Research interests: machine learning, pattern recognition, signal processing, applied statistics
- Applications: image and music processing, bioinformatics, software engineering, grid control, experimental physics

The team

B. Kégl (team leader)
2006 -
- boosting
- MCMC
- Auger

R. Busa-Fekete (postdoc)
2008 -
- boosting
- optimization
- SysBio

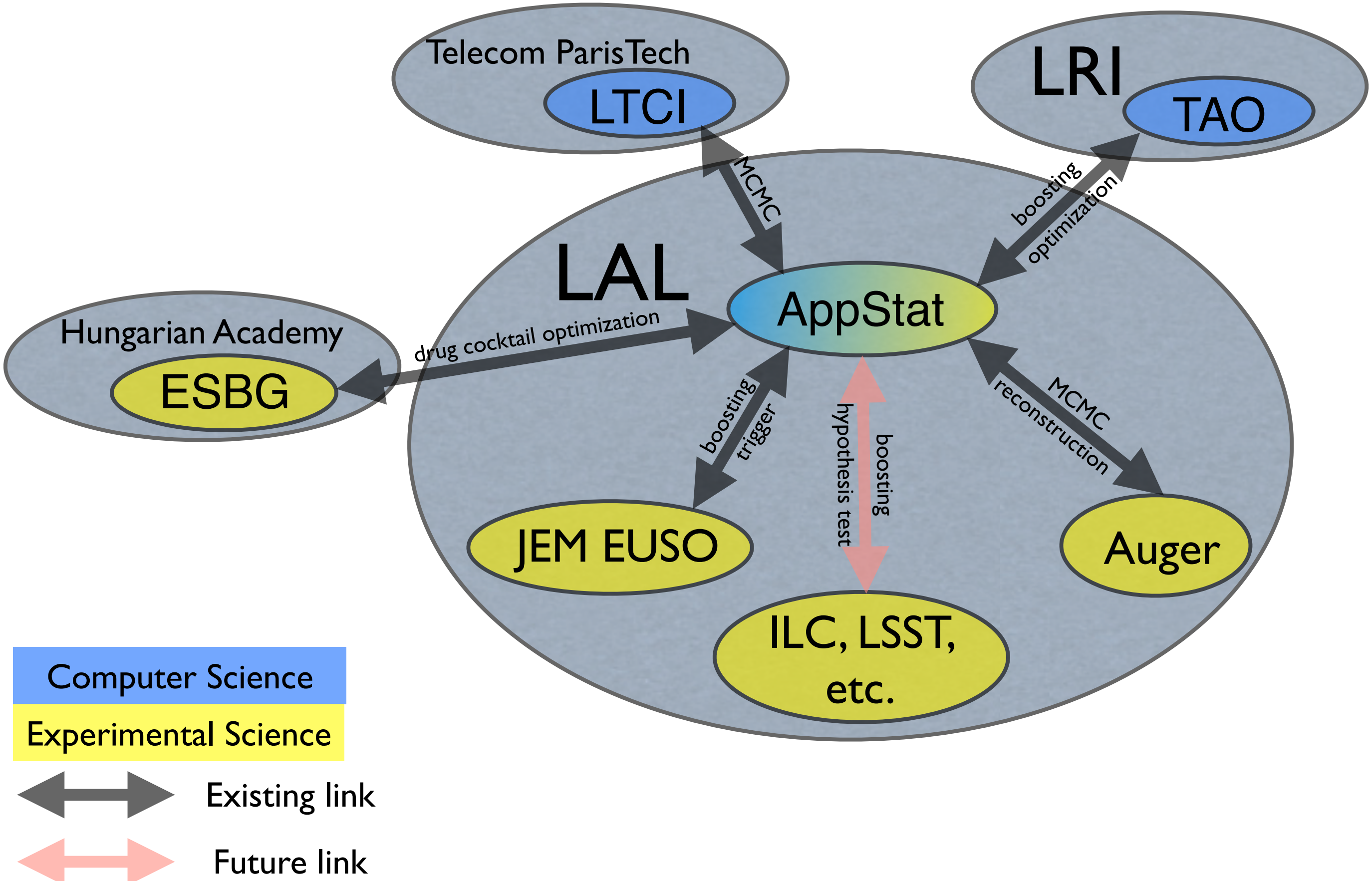
D. Benbouzid (Ph.D. student)
2010 -
- boosting
- JEM EUSO

R. Bardenet (Ph.D student)
2009 -
- MCMC
- optimization
- Auger

F-D. Collin (software engineer; 01/12/2010)
- multiboost.org
- MCMC in root
- system integration

D. Garcia (postdoc; 01/01/2011)
- generative models
- Auger / JEM EUSO
- tutoring

Collaborations



Funding

- ANR “jeune chercheur” MetaModel
  - 2007–2010, 150K€
- ANR “COSINUS” Siminole
  - 2010–2014, 1043K€ (658K€ at LAL)
- MRM Grille Paris Sud
  - 2010–2012, 60K€ (31K€ at LAL)

Siminole within ANR COSINUS

- COSINUS = Conception and Simulation
  - Theme 1: simulation and supercomputing
  - Theme 2: conception and optimization
  - Theme 3: large-scale data storage and processing
- Siminole
  - principal theme: Theme 2
  - secondary theme: Theme 1

Siminole within ANR COSINUS

- Simulation: third pillar of scientific discovery

- Improving simulation

  - algorithmic development inside the simulator

  - implementation on high-end computing devices

  - our approach: control the number of calls to the simulator

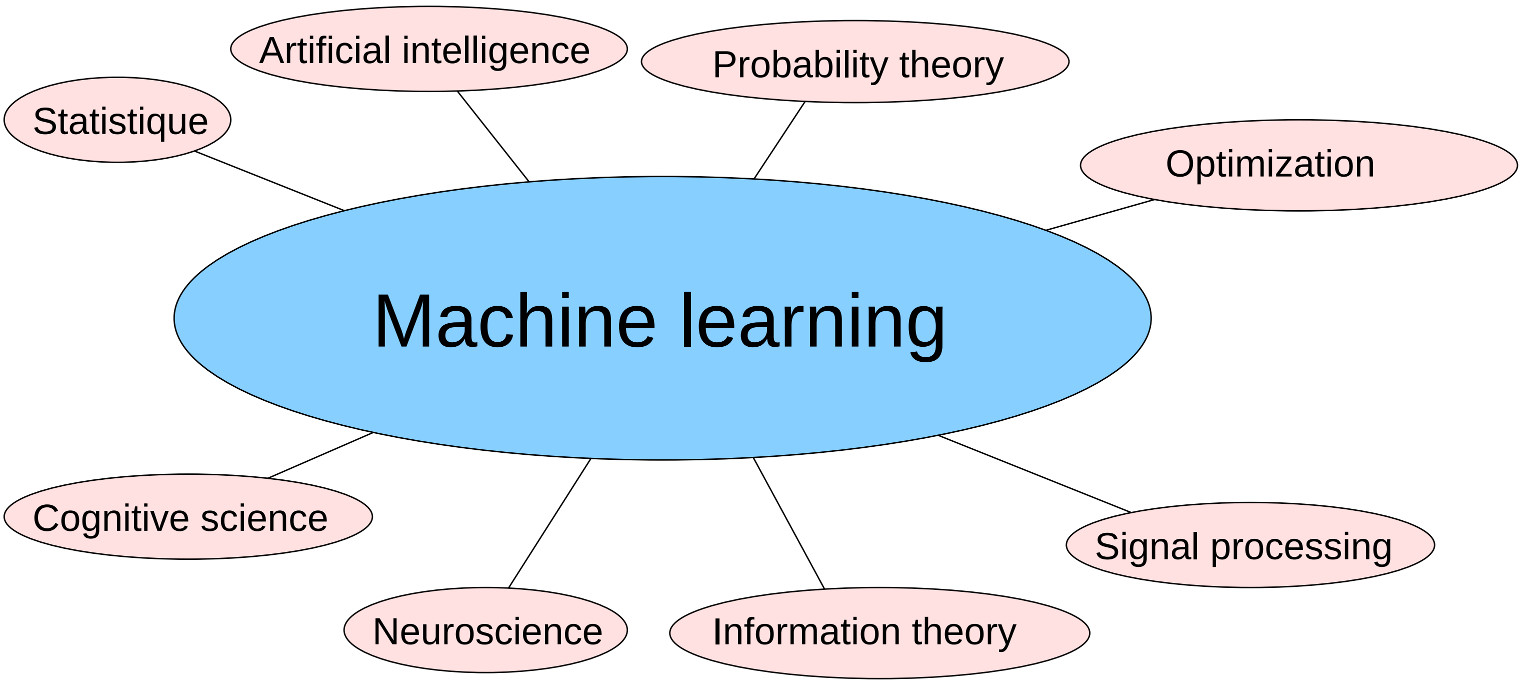
Siminole within ANR COSINUS

- Optimization
  - simulate from $f(x)$, find $\max_x f(x)$
- Inference
  - simulate from $p(x|\theta)$, find $p(\theta|x)$
- Discriminative learning
  - simulate from $p(x,\theta)$, find $\theta = f(x)$

Discriminative learning $\to$ boosting $\to$ `multiboost.org`

- Discriminative learning (classification)
  - Infer $f(\mathbf{x}) : \mathbb{R}^d \to 1, \ldots, K$ from a database $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$
- boosting, AdaBoost
  - one of the state-of-the-art classification algorithms
- `multiboost.org`
  - our implementation

Machine learning at the crossroads



Machine Learning

- From a statistical point of view
  - non-parametric fitting, capacity/complexity control
  - large dimensionality
  - large data sets, computational issues
  - mostly classification (categorization, discrimination)

Discriminative learning

- observation vector: $\mathbf{x} \in \mathbb{R}^d$
- class label: $y \in \{-1, 1\}$ – binary classification
- class label: $y \in \{1, \ldots, K\}$ – multi-class classification
- classifier: $g : \mathbb{R}^d \mapsto \{-1, 1\}$
- discriminant function: $f : \mathbb{R}^d \mapsto [-1, 1]$

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

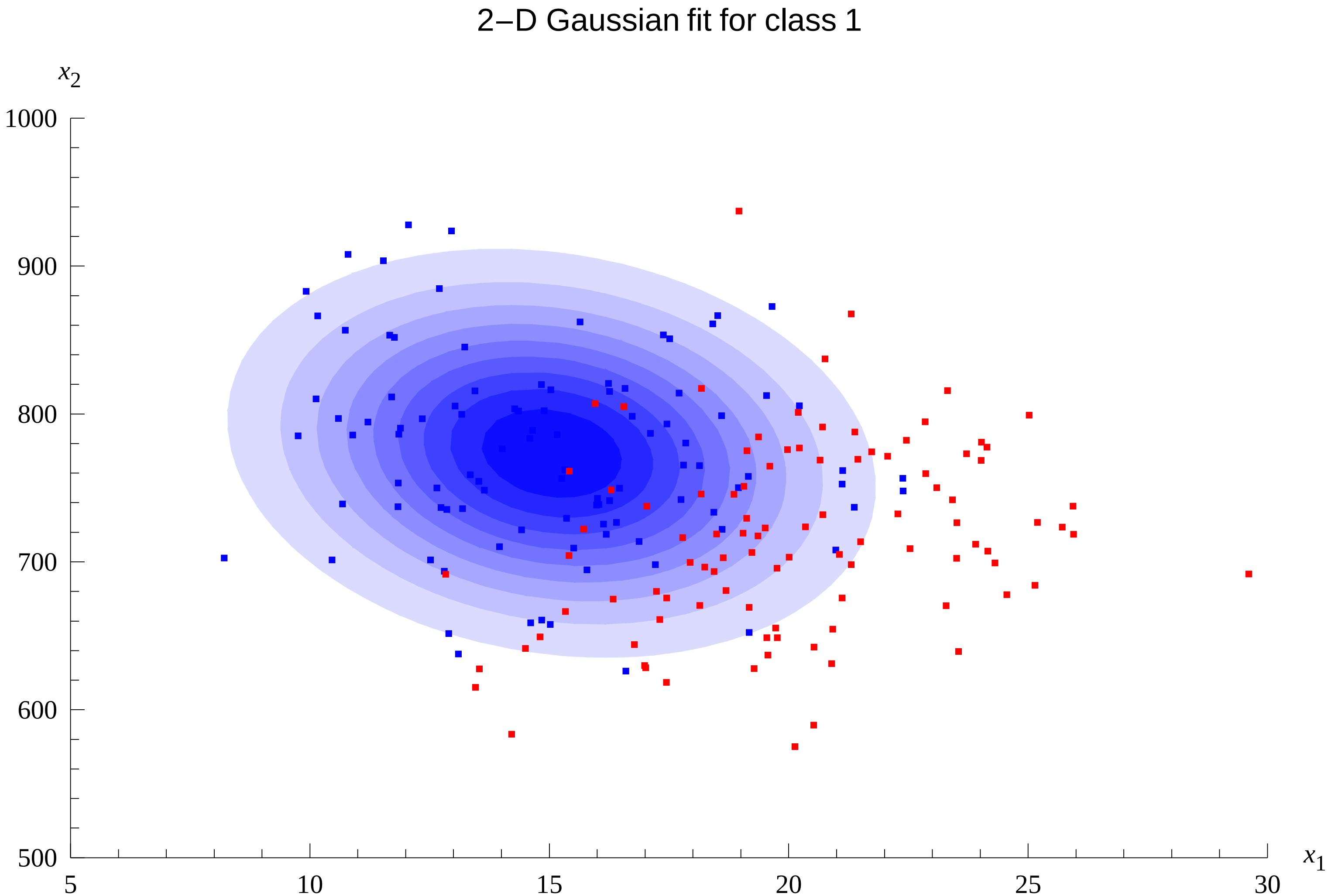
Discriminative learning

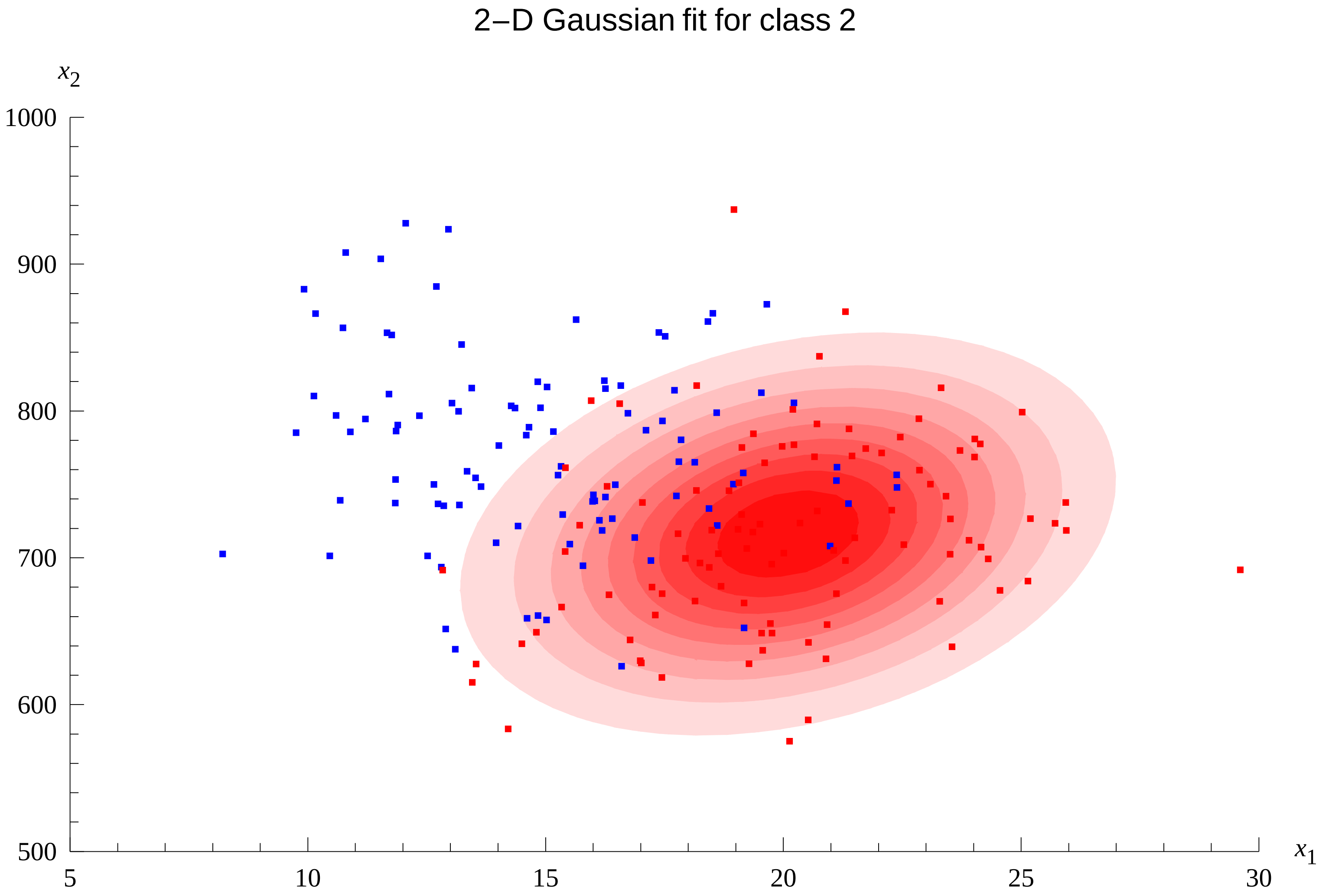
- Inductive learning
  - training sample: $D_n = \Big\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\Big\}$
  - function set: $\mathcal{F}$
  - learning algorithm: $\textsc{Algo} : \Big(\mathbb{R}^d \times \{-1, 1\}\Big)^n \mapsto \mathcal{F}$

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

  - goal: small generalization error $P\big[f(\mathbf{X}) \neq Y\big]$


Data for two−class classification problem
x_2
1000
900
800
700
600
500
5
10
15
20
25
30
x_1


2−D Gaussian fit for class 1
x2
1000
900
800
700
600
500
5
10
15
20
25
30
x1


2−D Gaussian fit for class 2
x2
1000
900
800
700
600
500
5
10
15
20
25
30
x1

Classification

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

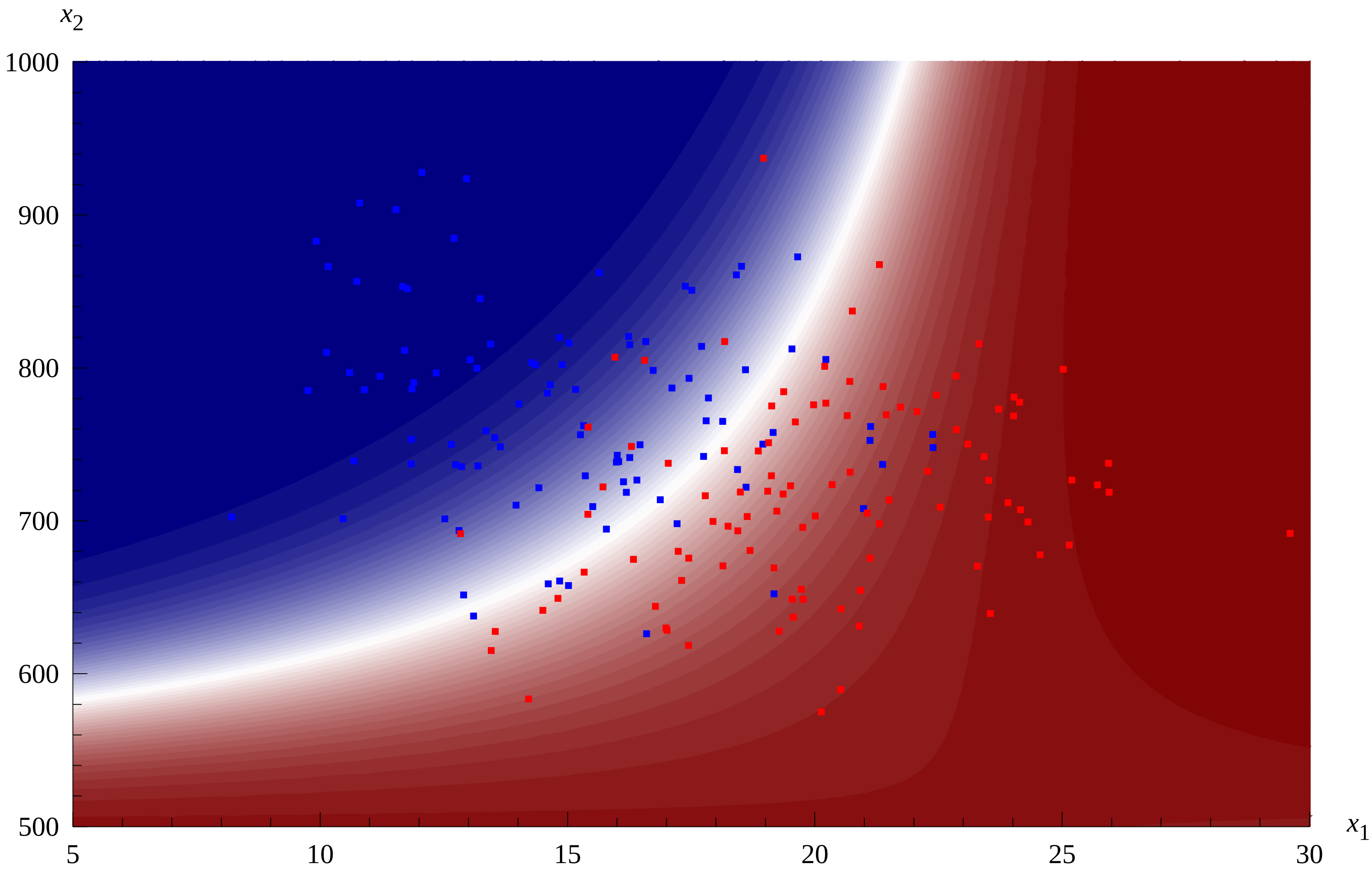
- Bayes theorem:

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

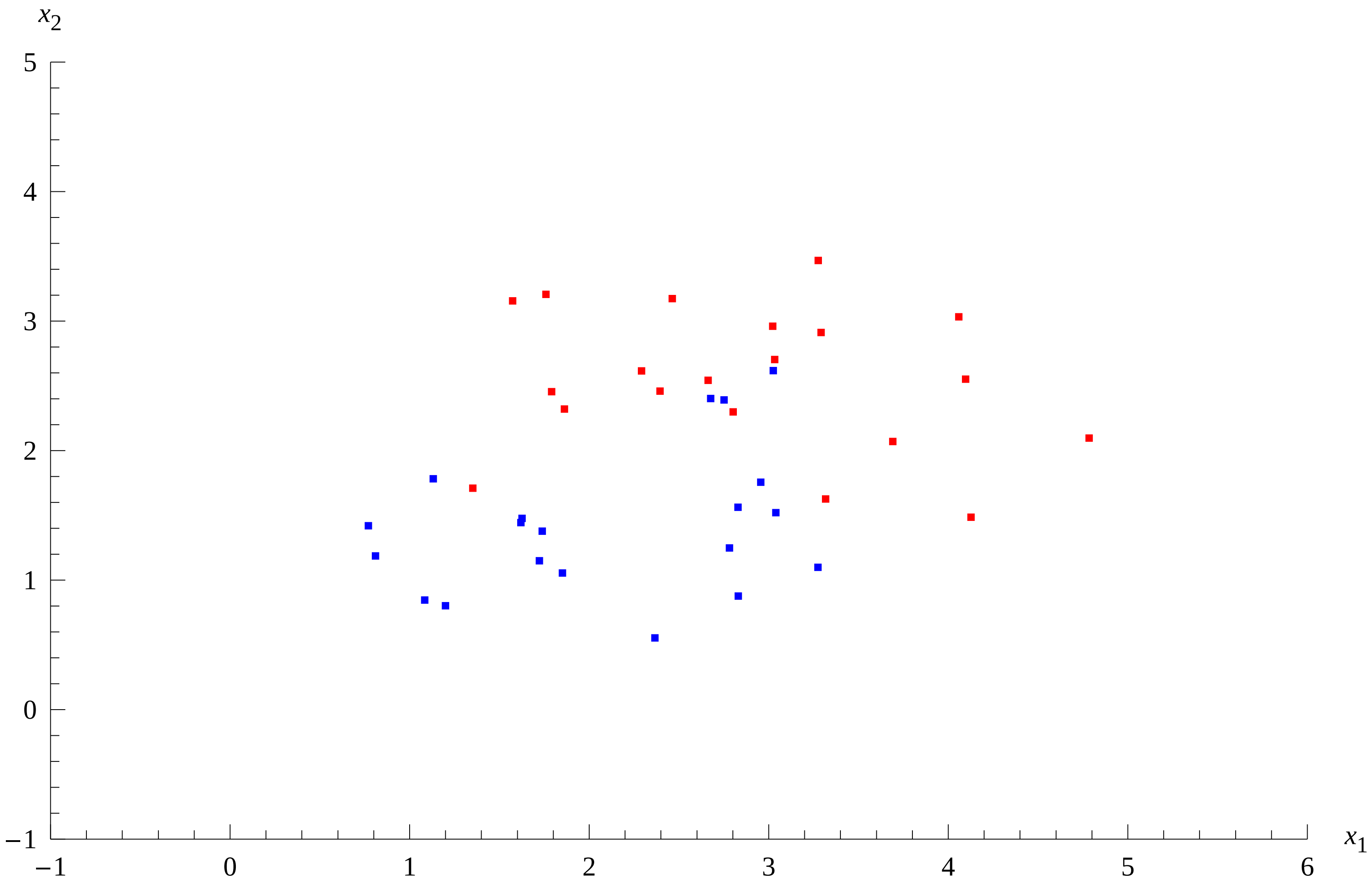
  - Decision:

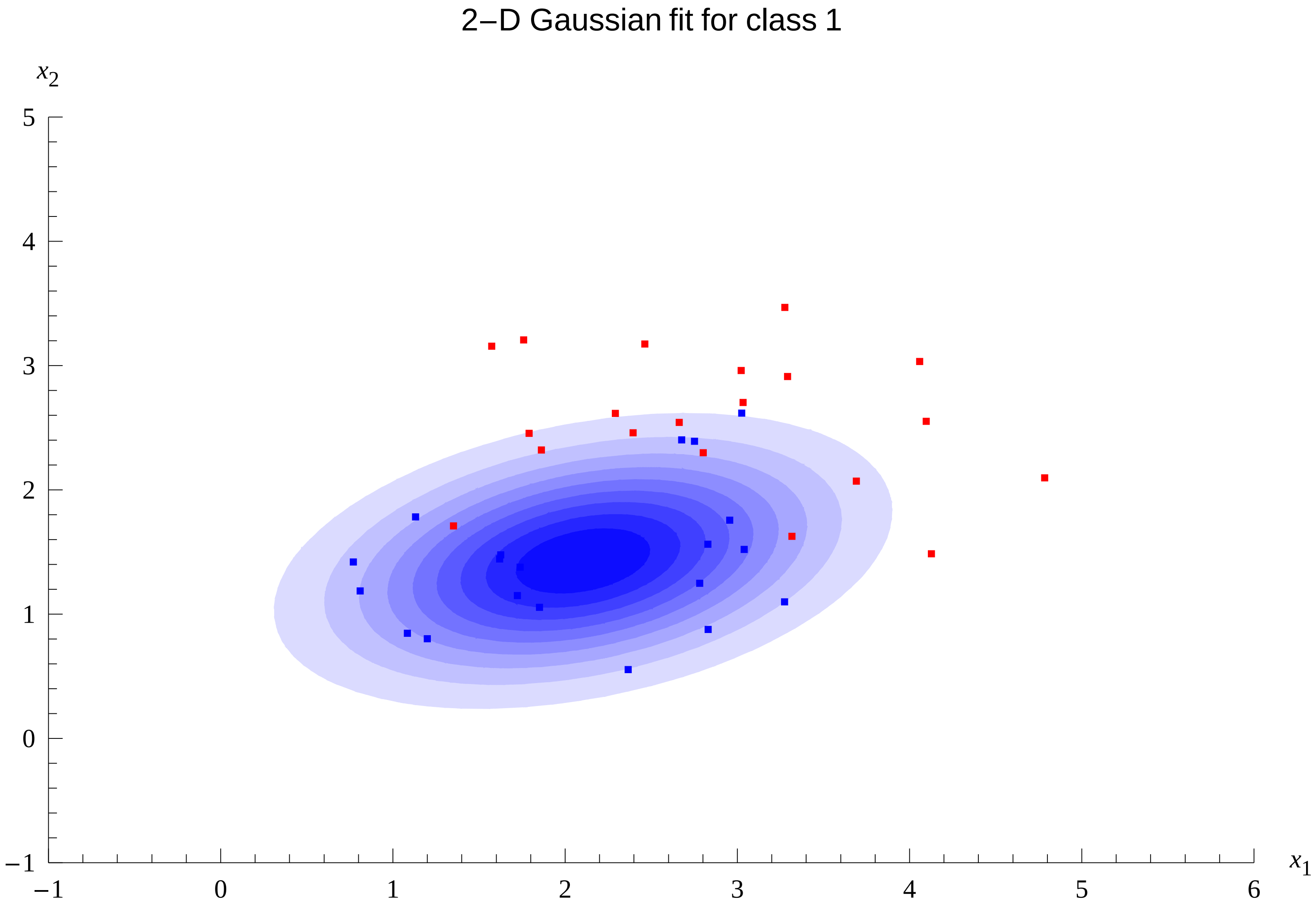
$$g(\mathbf{x}) = \begin{cases} 1 & \text{if } \frac{p(\mathbf{x}|Y=1)p(Y=1)}{p(\mathbf{x}|Y=-1)p(Y=-1)} > 1, \\ -1 & \text{otherwise.} \end{cases}$$

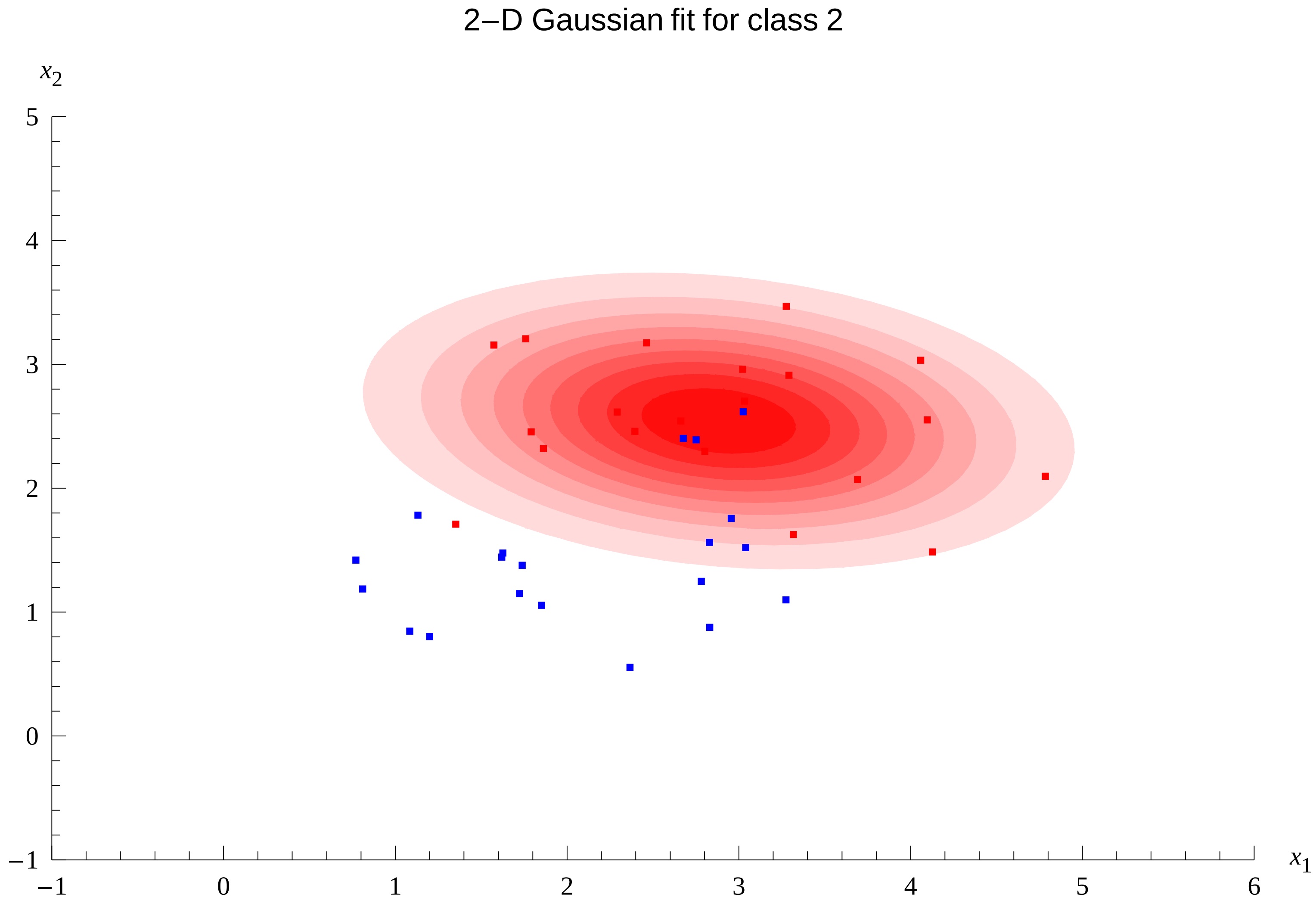
Discriminant function with Gaussian fits



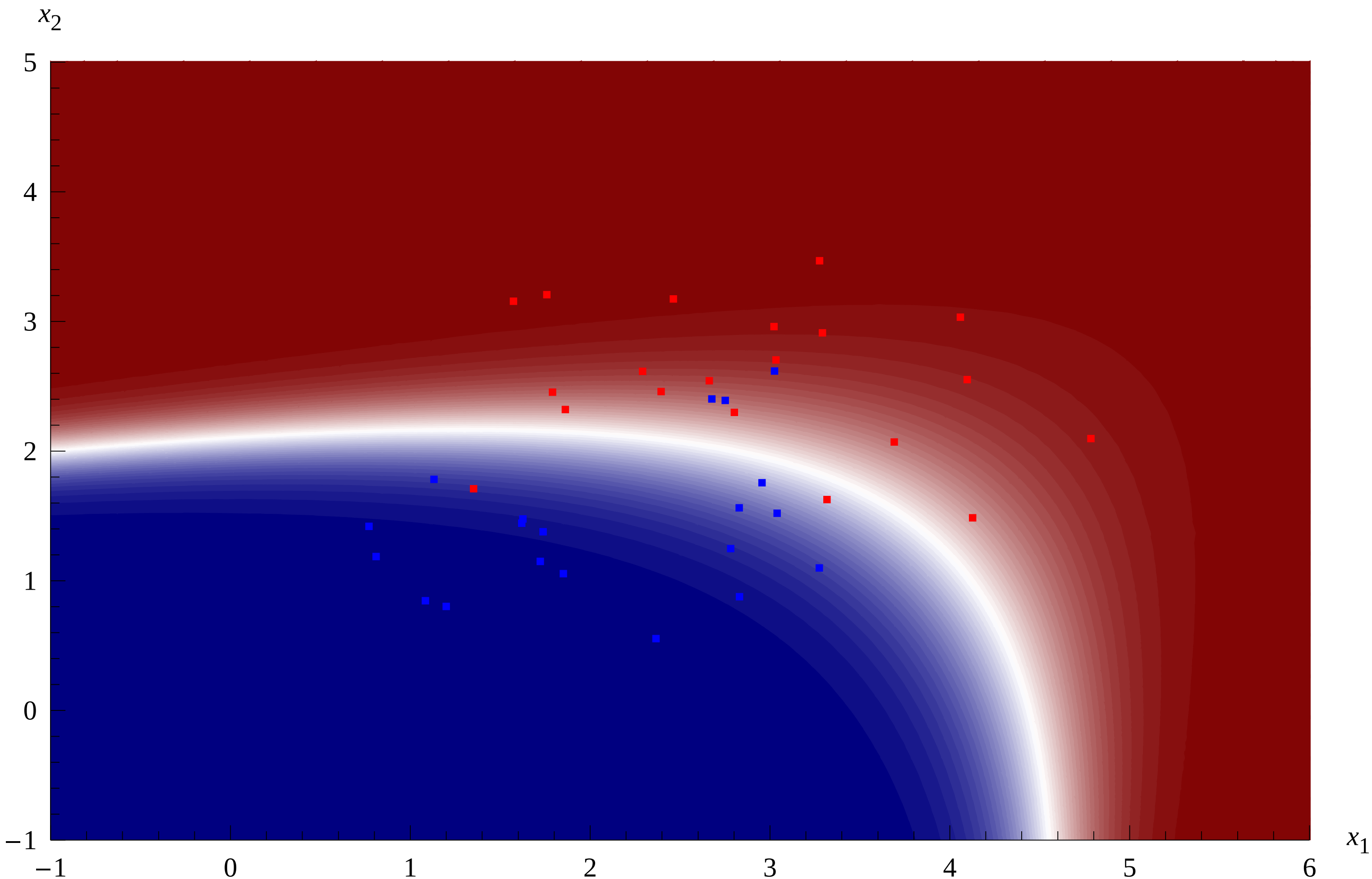
'Two Moons' data for two−class classification problem



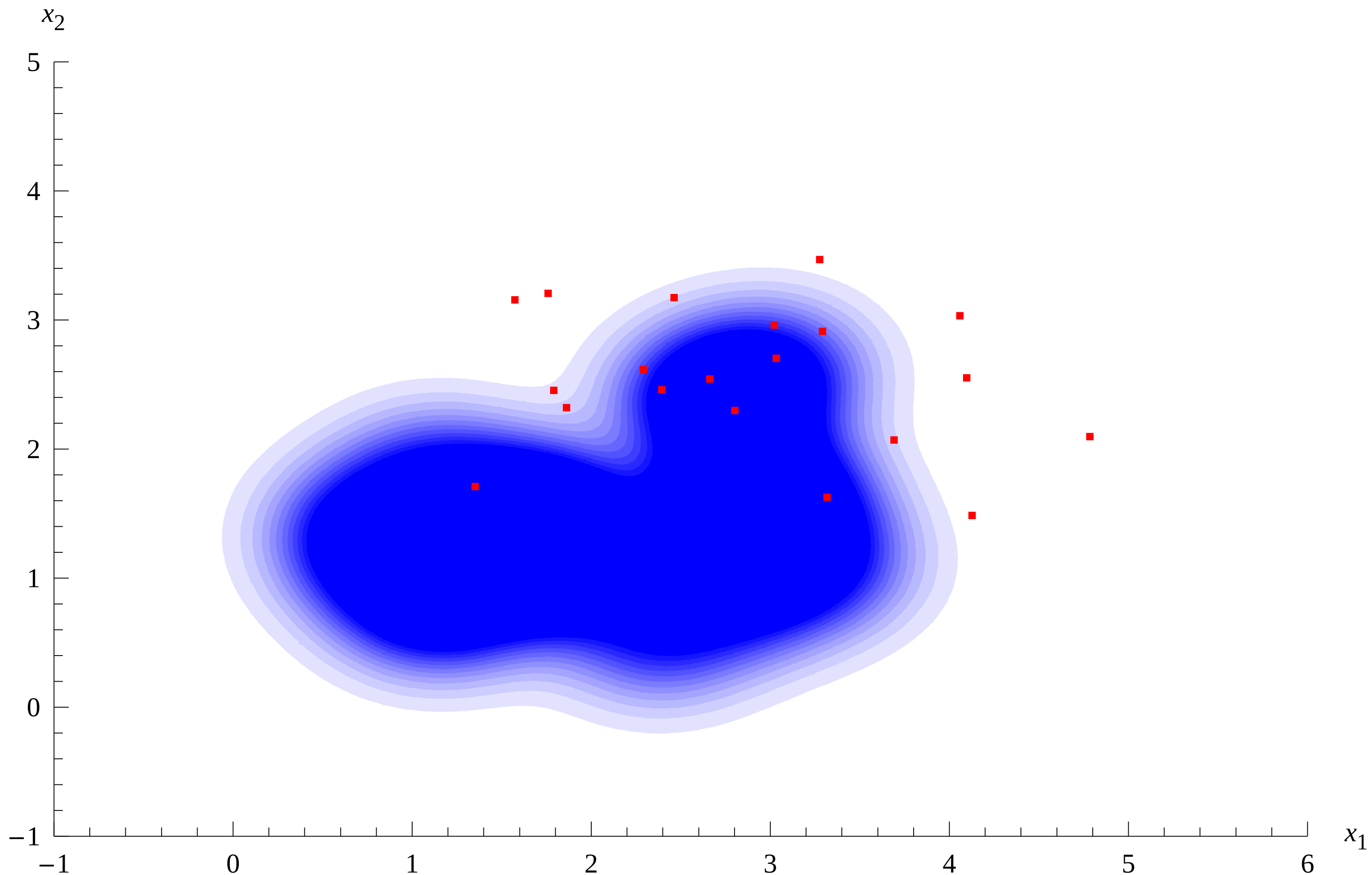

2−D Gaussian fit for class 1
x_2
5
4
3
2
1
0
−1
x_1
−1
0
1
2
3
4
5
6


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

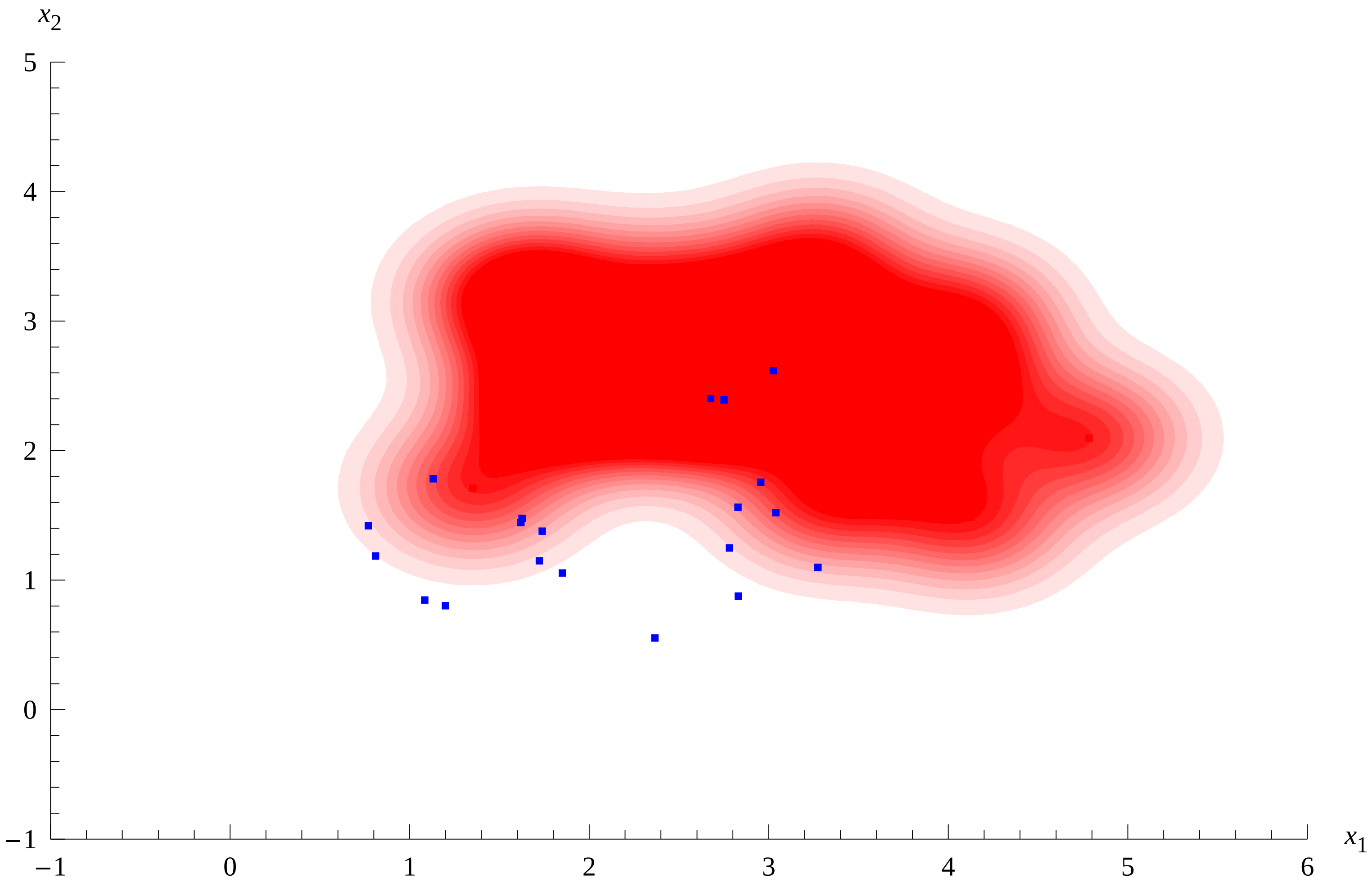
Discriminant function with Gaussian fits



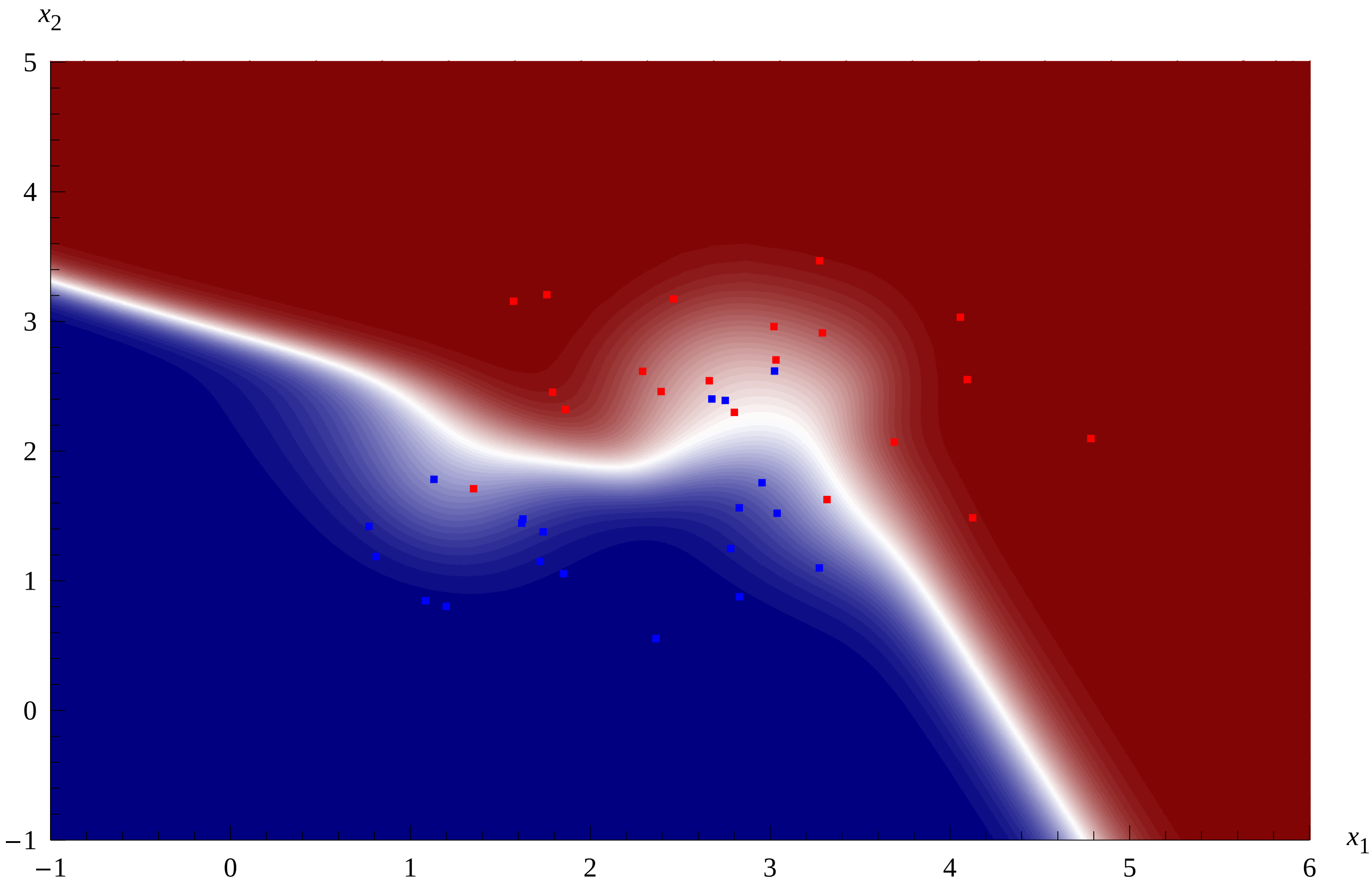
2−D Parzen fit for class 1, $h = 0.12$



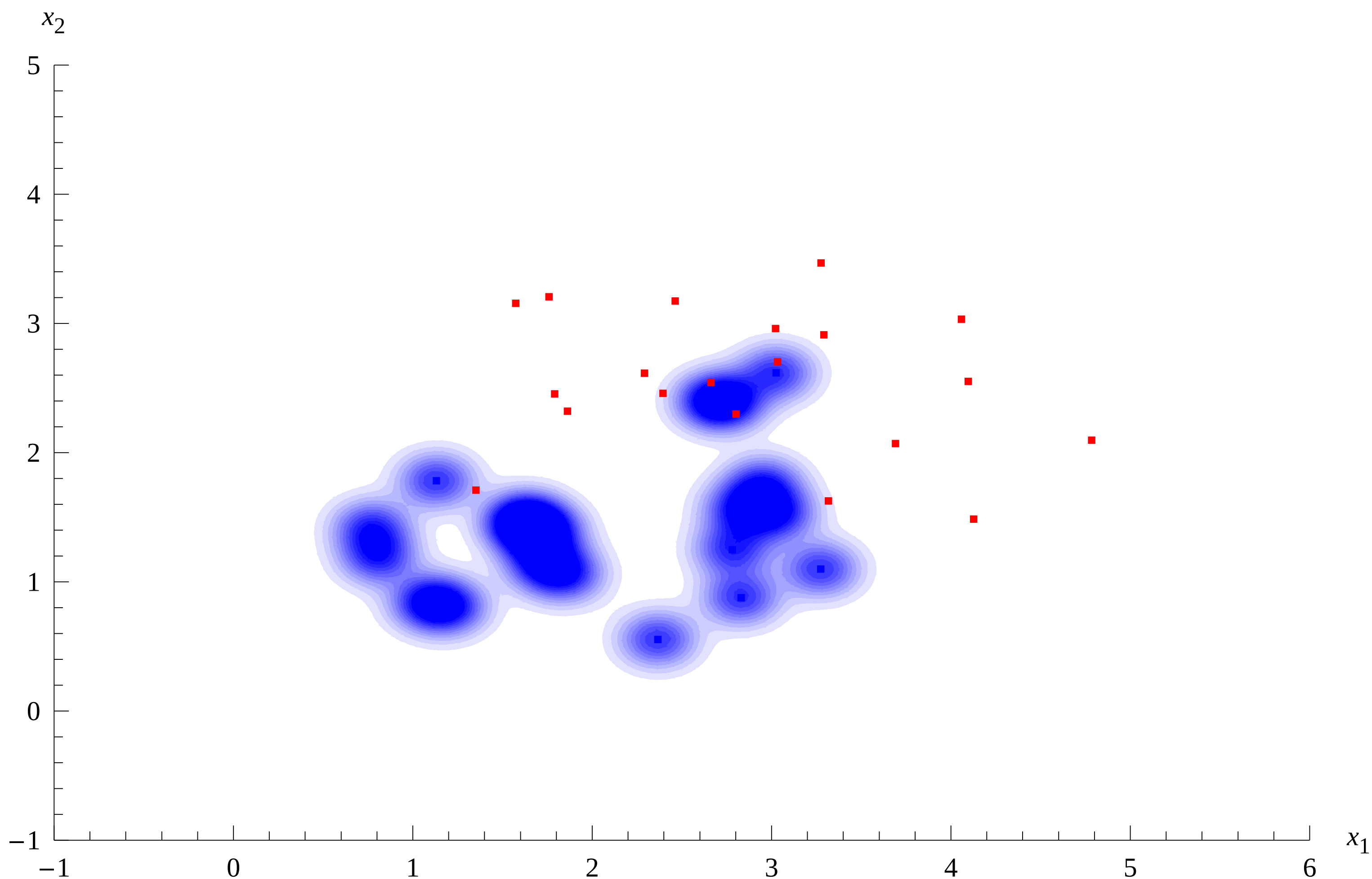
2–D Parzen fit for class 2, $h = 0.12$

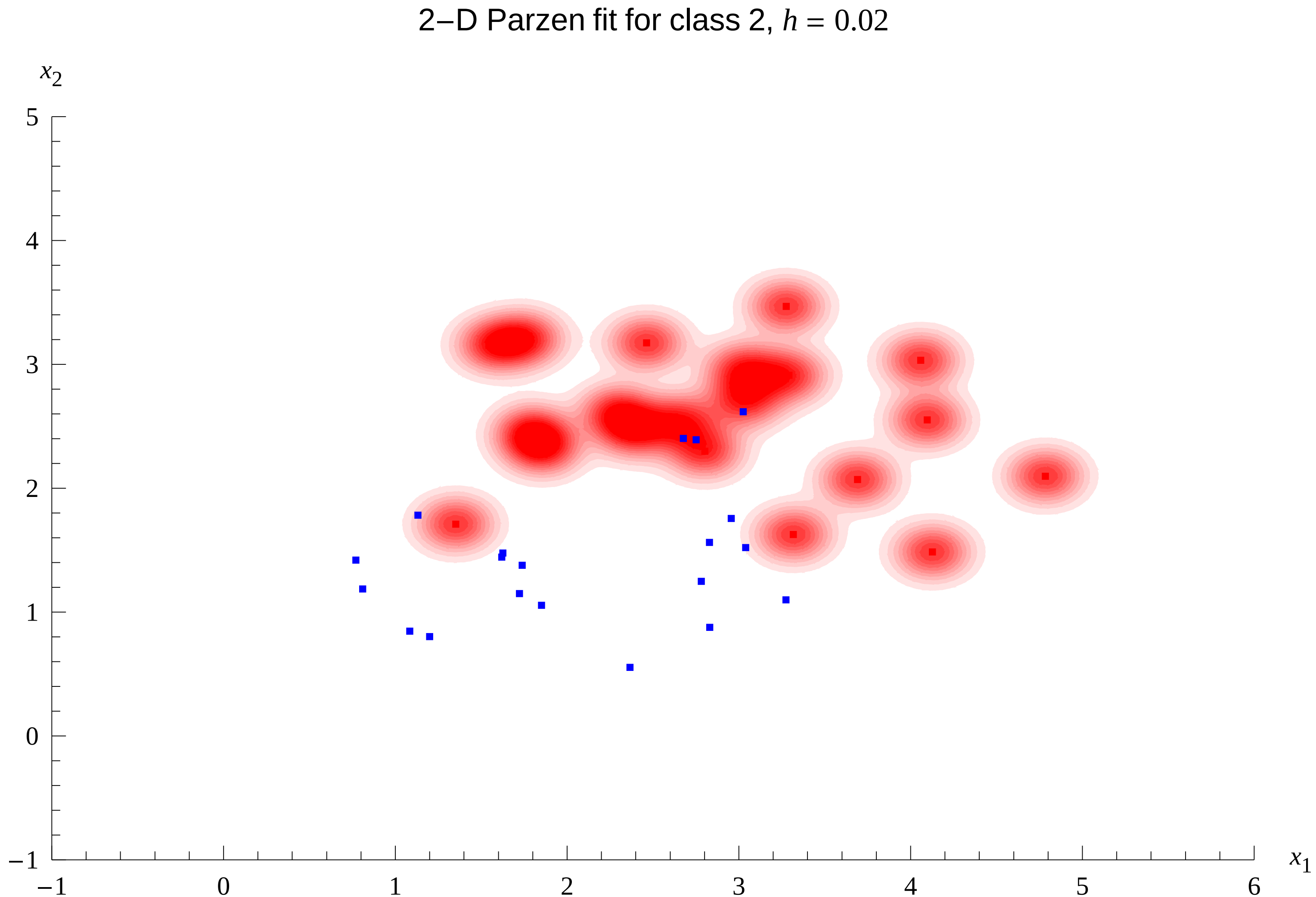


Discriminant function with Parzen fits, $h = 0.12$

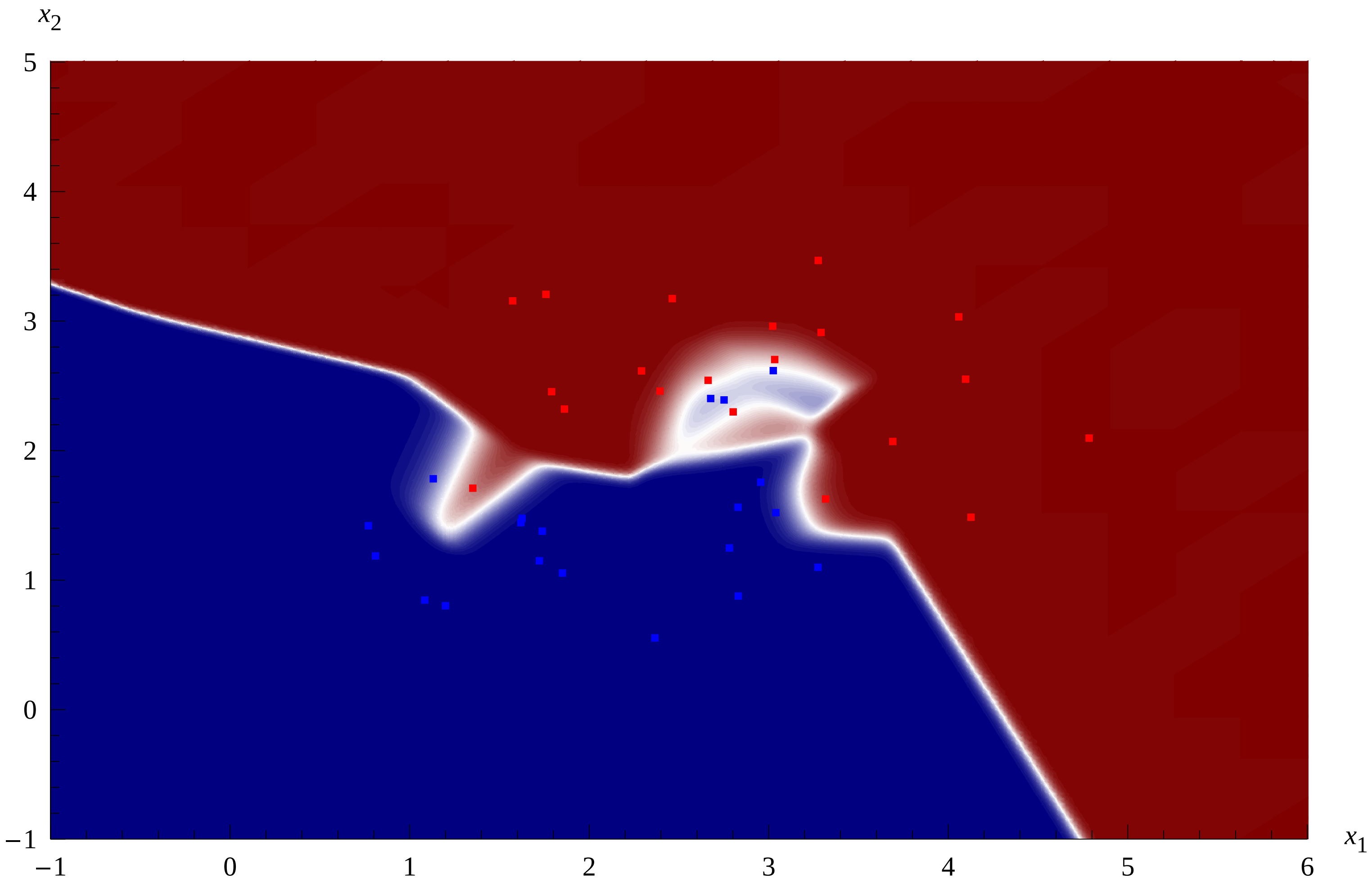


2−D Parzen fit for class 1, $h = 0.02$

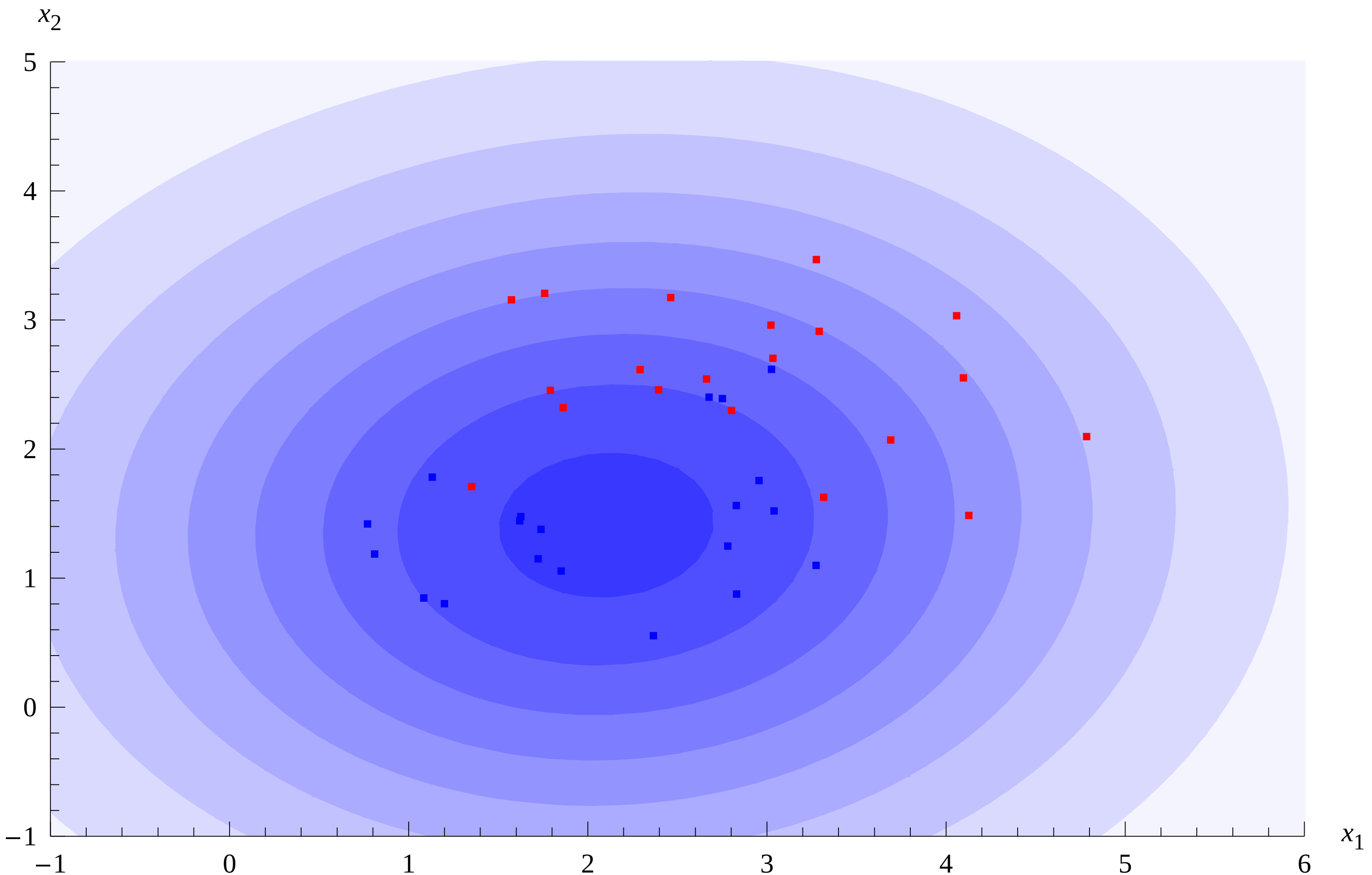



2−D Parzen fit for class 2, h = 0.02
x_2
5
4
3
2
1
0
−1
−1
0
1
2
3
4
5
6
x_1

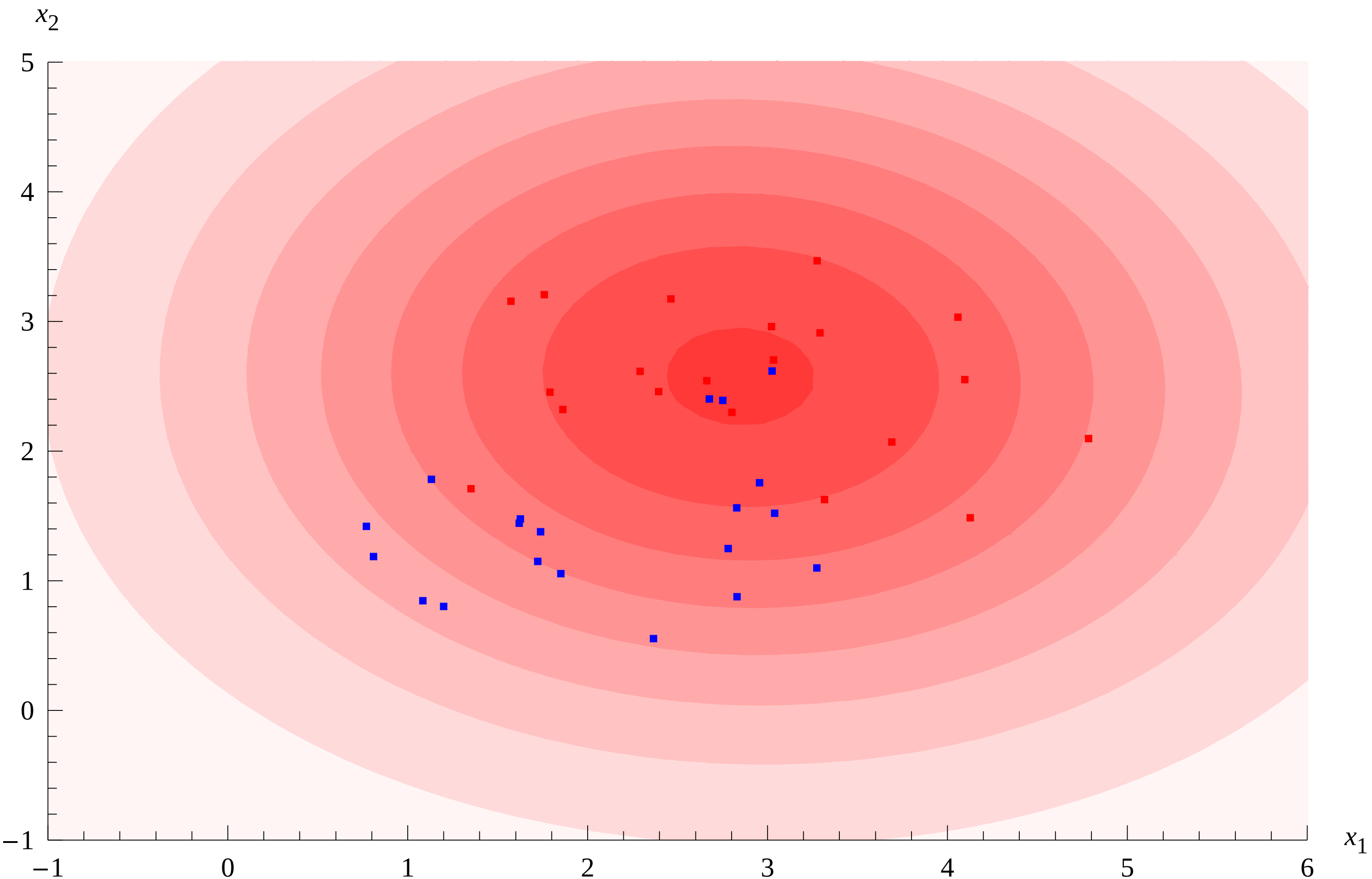
Discriminant function with Parzen fits, $h = 0.02$



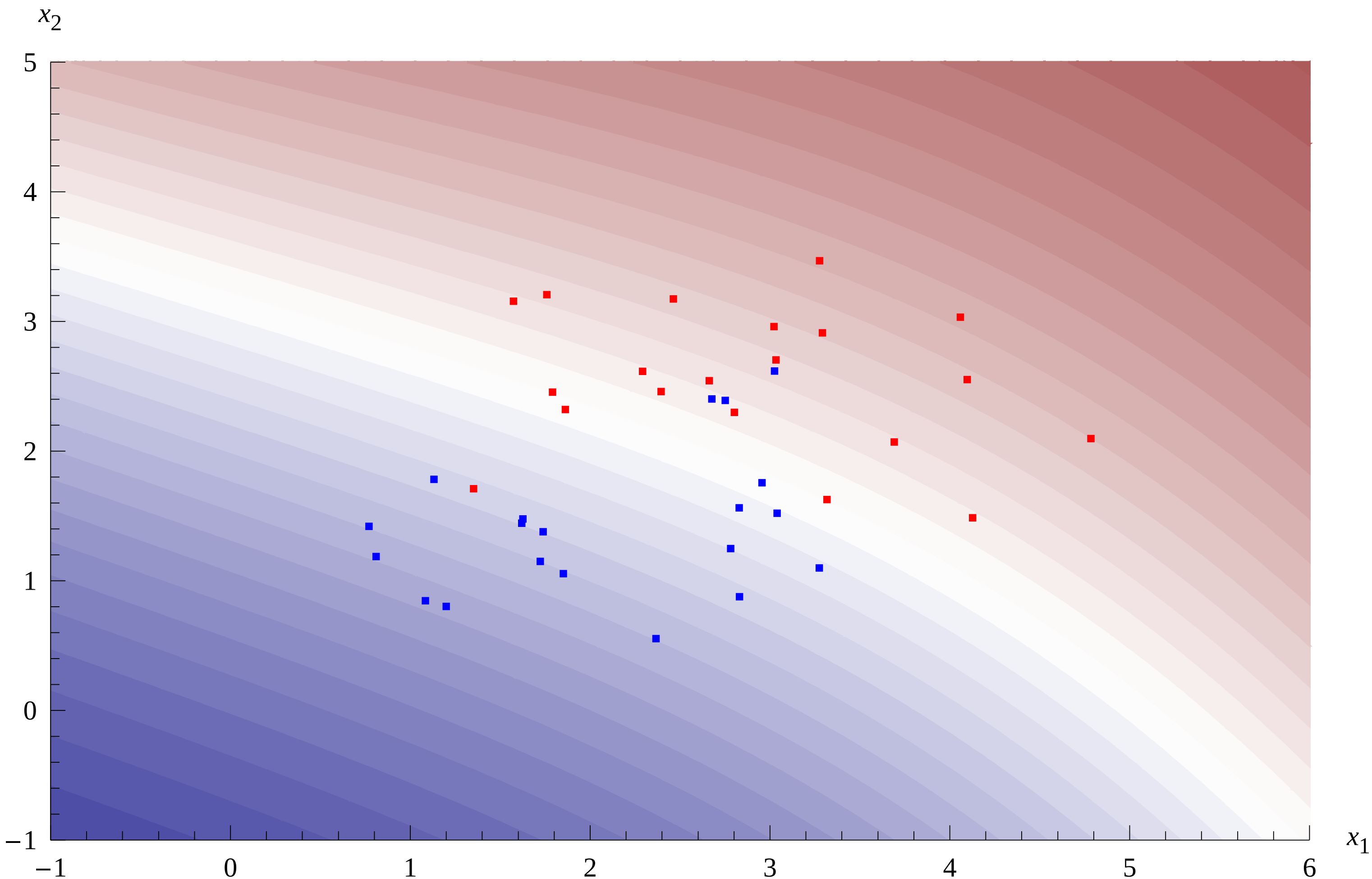
2–D Parzen fit for class 1, $h = 3$



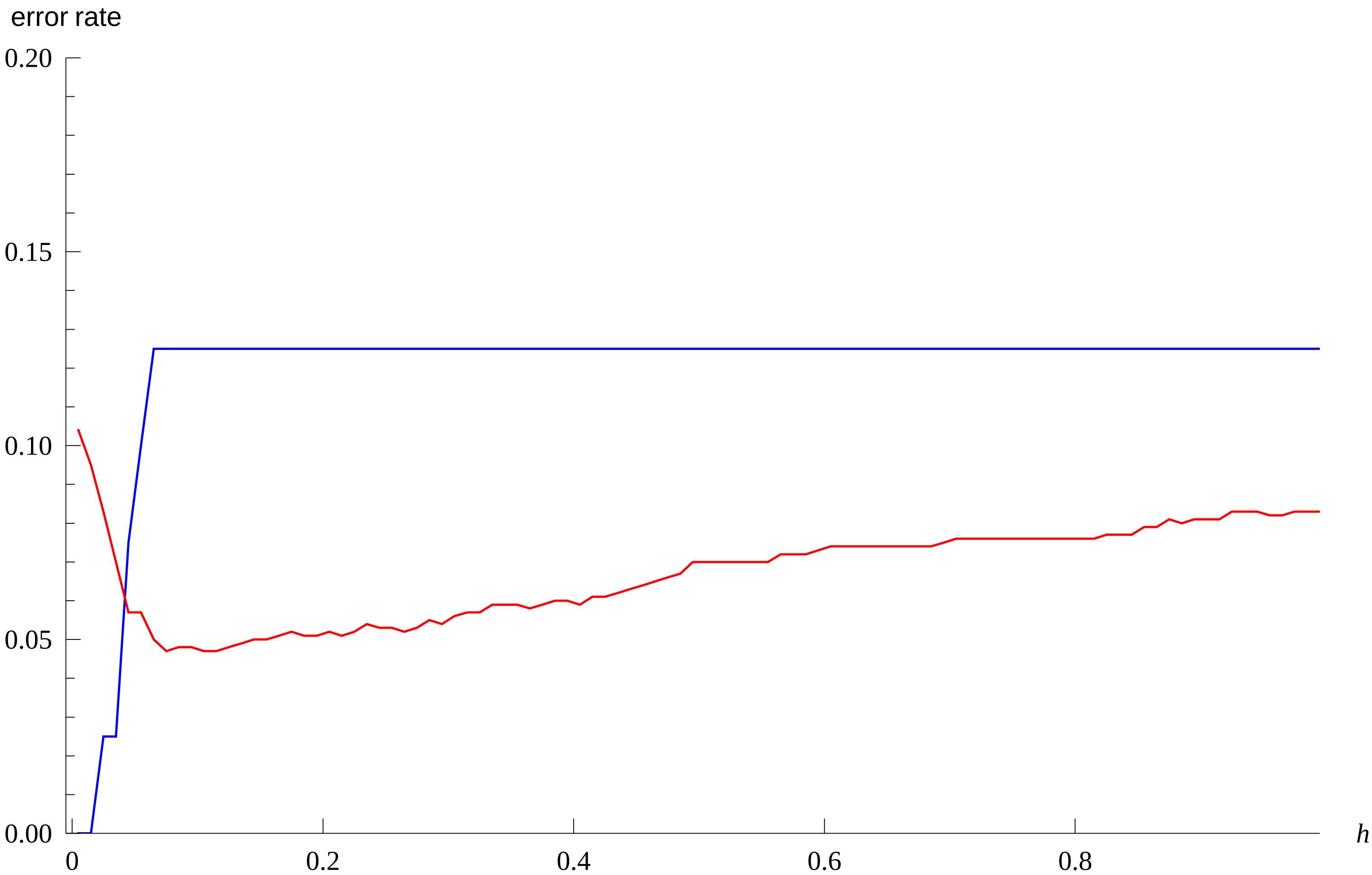
2–D Parzen fit for class 2, $h = 3$



Discriminant function with Parzen fits, $h = 3$



Training and test error rates for Parzen fits with different bandwidths



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

Curse of dimensionality

- Capacity/complexity control becomes a real issue in high-dimensional spaces
  - in a 10000-dimensional space a linear function has 10000 parameters!
- Examples
  - images
  - music
  - language, text
  - bioinfo (genetics, proteomics)

Machine learning problems

- Common goal: predict the future
  - make inferences on unknown future observations
- Non-supervised learning
  - density estimation $p(\mathrm{obs})$
  - clustering, dimensionality reduction, one-class learning
- Supervised learning
  - Classification : $f(\mathrm{obs}) \mapsto$ category
  - Regression : $f(\mathrm{obs}) \mapsto$ response

The supervised learning model

- observation vector: $\mathbf{x} \in \mathbb{R}^d$
- class label: $y \in \{-1, 1\}$ (or $y \in \{1, \ldots, K\}$)
- classifier: $g : \mathbb{R}^d \to \{-1, 1\}$
- Discriminant function: $f : \mathbb{R}^d \to [-1, 1]$
  - $\longrightarrow$ classifier

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

  - decision boundary: $\{\mathbf{x} : f(\mathbf{x}) = 0\}$

The supervised learning model

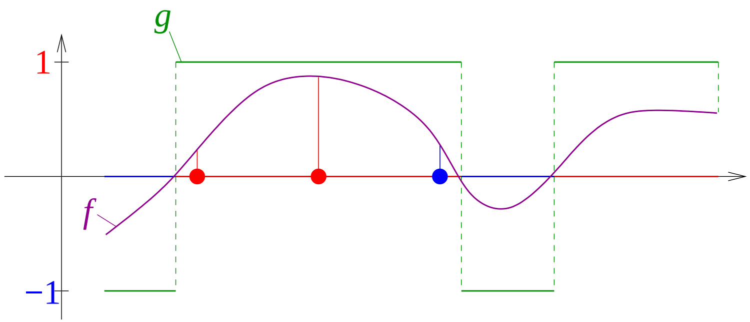
- Learning by experience, with a supervisor
  - training set : $D_n = \big\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\big\}$
  - function class : $\mathcal{F}$
  - learning algorithm : ALGO : $\big(\mathbb{R}^d \times \{-1, 1\}\big)^n \rightarrow \mathcal{F}$

$$\text{ALGO}(D_n) \mapsto f$$

  - goal: small generalization error $R(g) = P\big[g(\mathbf{X}) \neq Y\big] = P\big[f(\mathbf{X})Y \leq 0\big]$
  - learning principle: minimize the training error

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

The supervised learning model

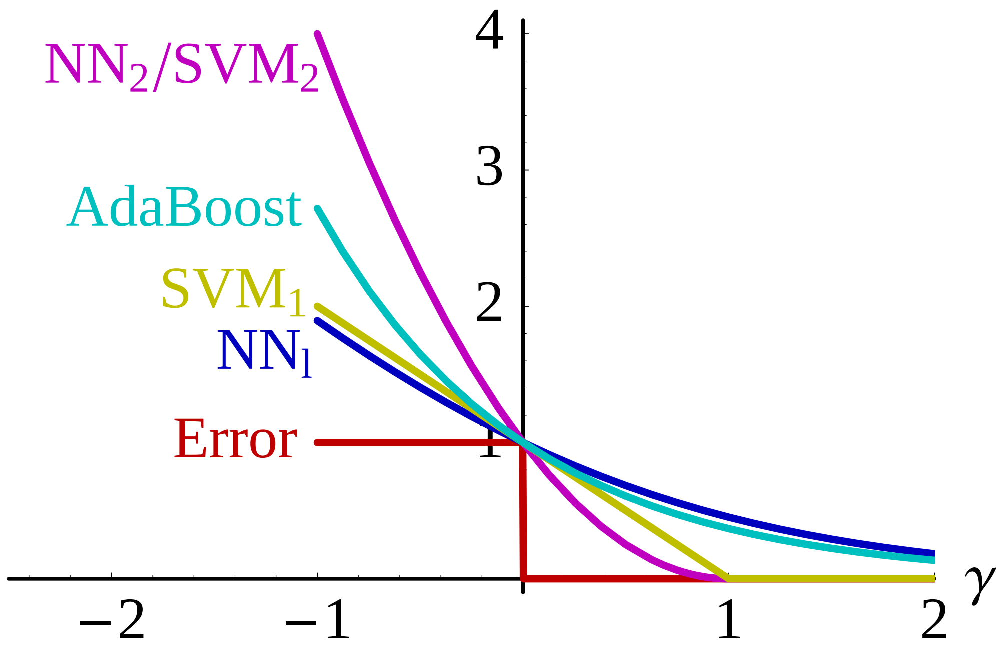


- Margin: $\gamma = y \cdot f(\mathbf{x})$
  - classification error $\equiv$ negative margin
  - the magnitude of a positive margin quantifies the confidence
  - learning principle: minimize a smooth loss function over the margin

$$\widehat{R}_\gamma(f) = \frac{1}{n}\sum_{i=1}^{n} L\big(f(\mathbf{x}_i)y_i\big)$$

The supervised learning model

- Margin loss functions

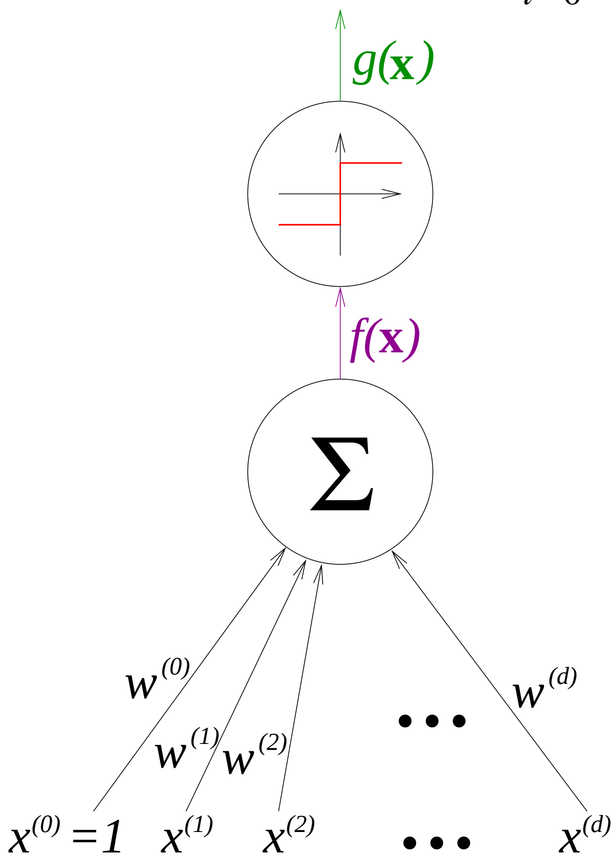


History

- Algorithms
  - 1958: Perceptron [Rosenblatt, '58] – [Minsky–Papert '69]
  - 1986: Multilayer perceptrons (neural networks) and the back-propagation algorithm [Rumelhart–Hinton–Williams, '86]
  - 1995: Support vector machines [Boser–Guyon–Vapnik, '92], [Cortes–Vapnik, '95]
  - 1997: boosting, AdaBoost [Freund, '95], [Freund–Schapire, '97]

The perceptron

- Linear discriminant functions: $f(\mathbf{x}) = \sum_{i=0}^{d} w^{(i)} \cdot x^{(i)} = \langle \mathbf{w}, \mathbf{x} \rangle$



The perceptron

- Linear discriminant functions: $f(\mathbf{x}) = \sum_{i=0}^{d} w^{(i)} \cdot x^{(i)} = \langle \mathbf{w}, \mathbf{x} \rangle$

- Algorithm
  - simple iterative error correction
  - convergence if the data is linearly separable
  - oscillation for linearly non-separable data

Generalized linear discriminant functions

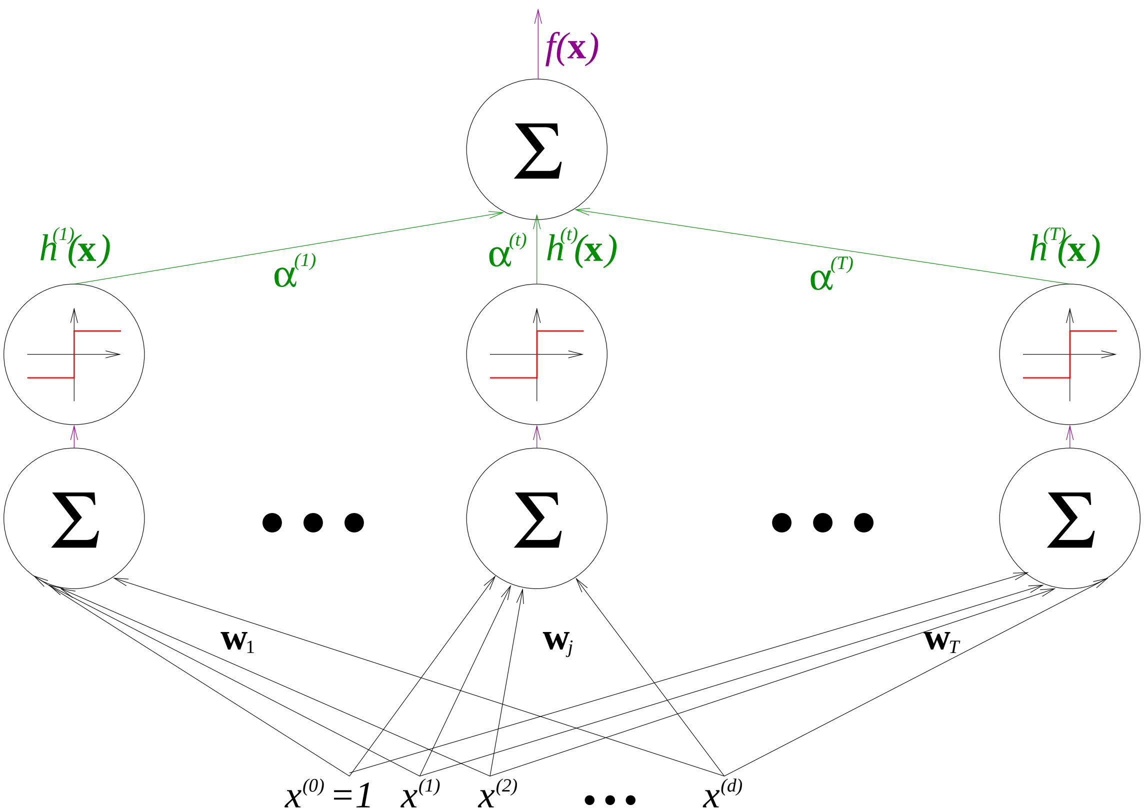
- Model:

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

- $h^{(j)} : \mathbb{R}^d \to [-1, 1]$
  - simple classifiers/discriminant functions, features, experts
- $\alpha^{(j)} \in \mathbb{R}^+$
  - weight of the expert $h^{(j)}$ in the final vote

Multilayer perceptron (neural net)

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



Multilayer perceptron (neural net)

- Model: $f(\mathbf{x}) = \sum_{j=1}^{N} \alpha^{(j)} \boldsymbol{\sigma}(\langle \mathbf{w}_j, \mathbf{x} \rangle)$
- Algorithm:
  - gradient descent optimization
  - differentiable error functions $\rightarrow$ margin loss
  - differentiable activation function σ: the sigmoid
  - local minima, “engineering”, parameters to tune

Support vector machine

- Model:

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

  - $I_{\text{SV}} \subset \{1, \ldots, n\}$ is the set of support vectors
  - $K(\cdot, \cdot)$ is a similarity function (kernel)
  - goal: classification boundary equidistant from classes
  - “sophisticated nearest neighbor”
  - slow and complex quadratic programming optimization
  - turn-key algorithm, very limited parameter tuning

Support vector machine

- Model:

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

- Kernel:
  - $K(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle \longrightarrow f(\mathbf{x})$ is linear
  - $K(\mathbf{x}, \mathbf{x}') = \big(1 + \langle \mathbf{x}, \mathbf{x}' \rangle\big)^d \longrightarrow f(\mathbf{x})$ is a polynomial of degree d
  - $K(\mathbf{x}, \mathbf{x}') = \exp\big(-1/h \|\mathbf{x} - \mathbf{x}'\|^2\big) \longrightarrow f(\mathbf{x})$ is a Gaussian mixture ($\rightarrow$ Parzen)

AdaBoost

- Model:

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

  - no restriction on the form of $h^{(j)}(\mathbf{x})$
  - often "decision stumps" :

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

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

AdaBoost

- Intuitive elementary algorithm
  - add one expert at a time
  - add the best expert on training points mis-classified by previous experts
  - weight of the expert chosen proportionally to its correctness

AdaBoost

- Weighting over the training points $w_1, \ldots, w_n$
  - normalized: $\sum_{i=1}^{n} w_i = 1$
  - initialized uniformly : $\mathbf{w} = (1/n, \ldots, 1/n)$
  - if $\mathbf{x}_i$ is mis-classified by $h^{(j)}$, increase w_i
  - otherwise, decrease w_i
  - "difficult" training points get larger weights gradually

AdaBoost [Freund – Schapire ’97]

$\textsc{AdaBoost}\big(D_n = \{(\mathbf{x}_i, y_i)\}_{i=1}^n, \textsc{Base}(\cdot,\cdot), T\big)$

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

for $t \leftarrow 1$ **to** T

  $h^{(t)} \leftarrow \textsc{Base}\big(D_n, \mathbf{w}^{(t)}\big)$ ▷ *calling the base learner*

  $\gamma^{(t)} \leftarrow \sum_{i=1}^{n} w_i^{(t)} h^{(t)}(\mathbf{x}_i) y_i$ ▷ $edge = 1 - 2 \times error$

  $\alpha^{(t)} \leftarrow \frac{1}{2} \ln \left(\frac{1+\gamma^{(t)}}{1-\gamma^{(t)}} \right)$ ▷ *coefficient of* $h^{(t)}$

  for $i \leftarrow 1$ **to** n ▷ *re-weighting the points*

    if $h^{(t)}(\mathbf{x}_i) \neq y_i$ **then**

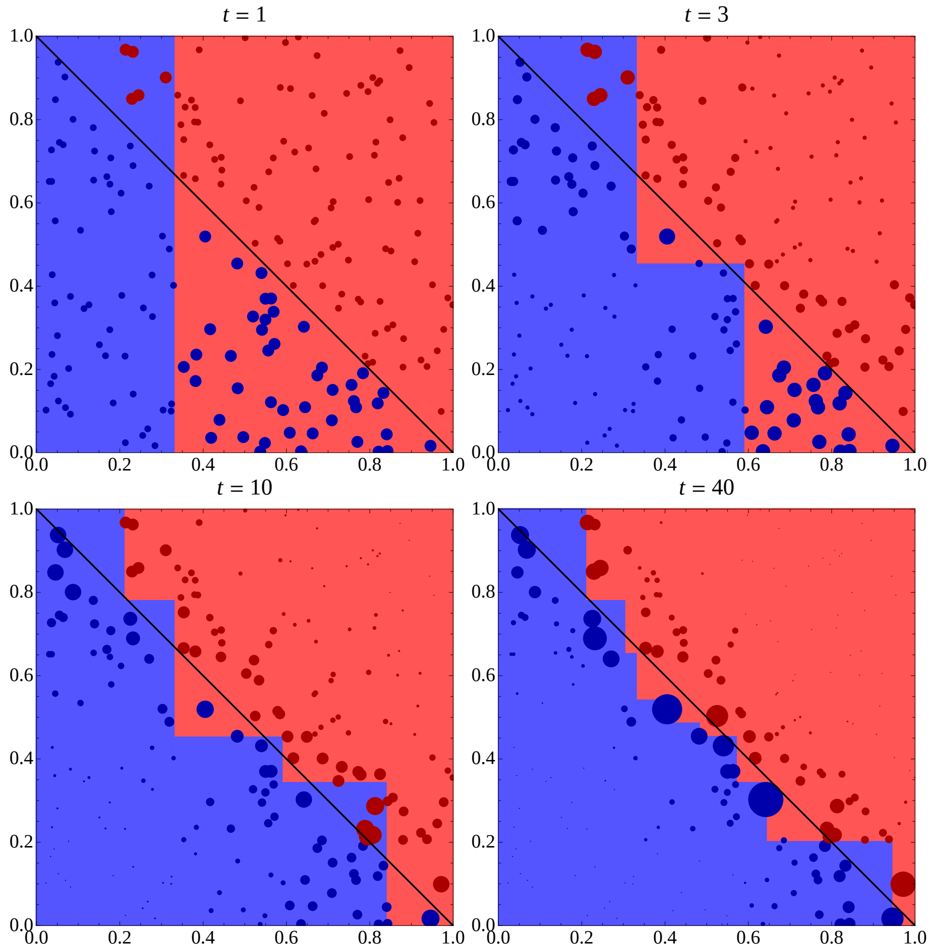
      $w_i^{(t+1)} \leftarrow w_i^{(t)} \frac{1}{1-\gamma^{(t)}}$

    else

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

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

AdaBoost



AdaBoost

- Algorithm
  - extremely simple learning, limited parameter tuning
  - fast
  - intuitive interpretation: weighted vote of experts
  - the choice of the pool of experts captures the a-priori knowledge
  - no restriction on the form of the experts
  - label noise can be a problem

multiboost.org

- Multi-class multi-label boosting software
  - based on AdaBoost.MH [Schapire-Singer '99]
  - started by Norman Casagrande (M.Sc. student in Montreal, now with last.fm)
  - multi-platform C++
  - command-line UI, easy-to-use for a non-expert
  - adapting to a new data type is easy for an advanced user
  - tons of features
  - scales nicely

multiboost.org

- Plan
  - going beyond classification: regression, ranking, collaborative filtering, reinforcement learning
  - technical improvements: multicore, GPU, grid, memory handling, etc.
  - redesign: orthogonal features → templates
  - implement a software development cycle (tests, etc.), can be tricky to balance between research and production

multiboost.org

- Plan for F-D.
  - get acquainted with Machine Learning through understanding the code (of course, we'll be there)
  - first concrete task: port it to multi-core (we have good understanding how) than to GPU (we have a vague understanding, more challenging to F-D.)
  - gradually implement a software development cycle
  - redesign