Intelligent Systems

Escuela Técnica Superior de Informática

Universitat Politècnica de València

Block 2 Chapter 2

Learning discriminant functions: Perceptron

November, 2014

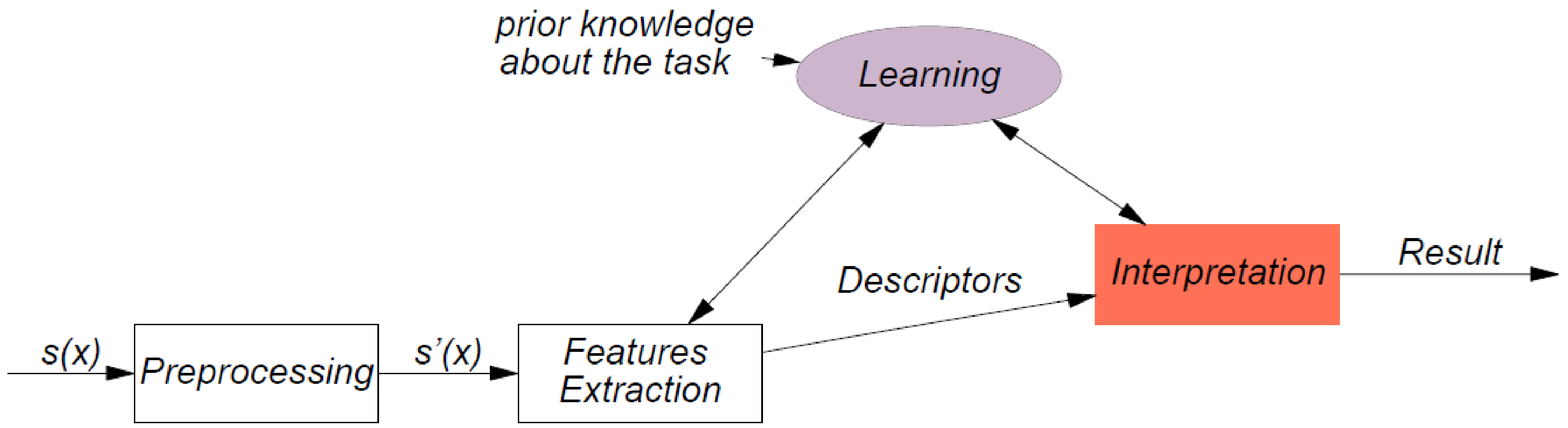
Syllabus

Syllabus

Pattern Recognition: goal and focus

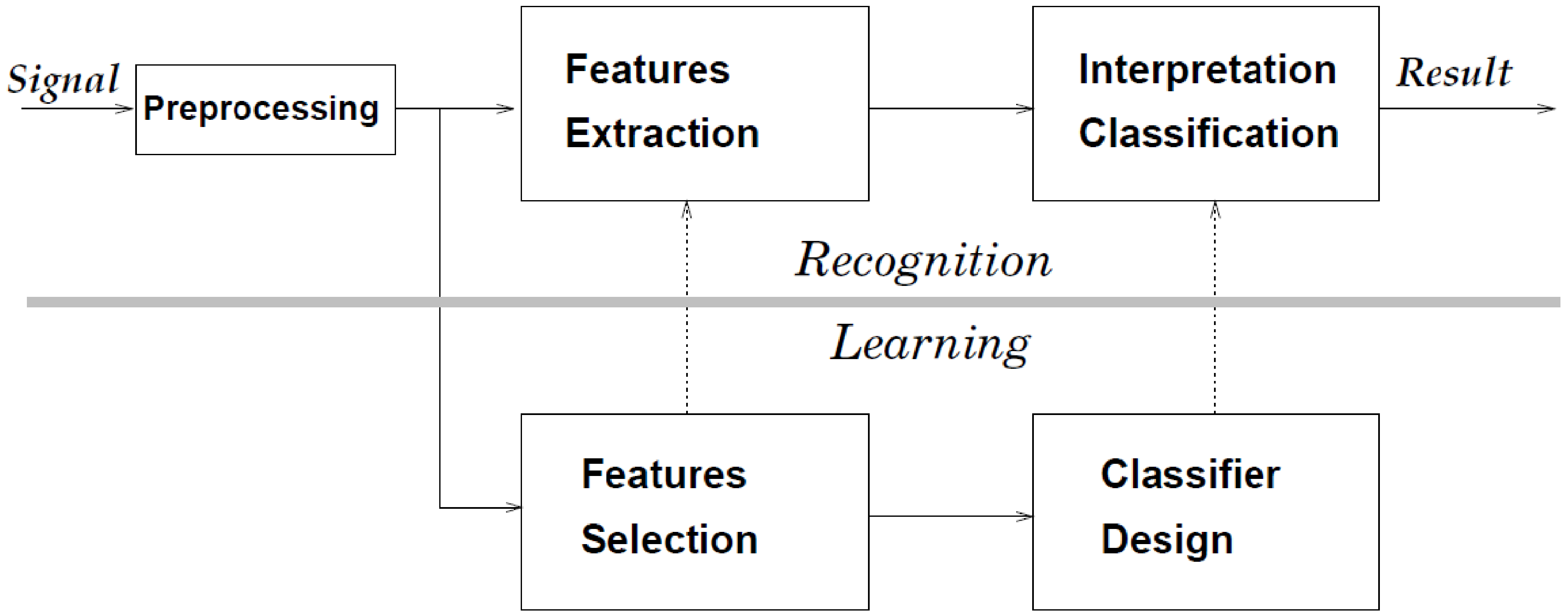
- Modelling the process of ***perception***
  - Difficult to formalize (an expert can seldom put his/her perceptual skills into words)
  - Extraordinary *plasticity*: ***learning*** (unconsciously) th-rough repeating exposition to the problems to solve (and their solutions)
- Modelling simple processes of *reasoning*
- *Intrinsic impossibility* to reach *exact* results
- Development of helpful systems to improve productivity and standard of living in general

Design of a Pattern Recognition (PR) System

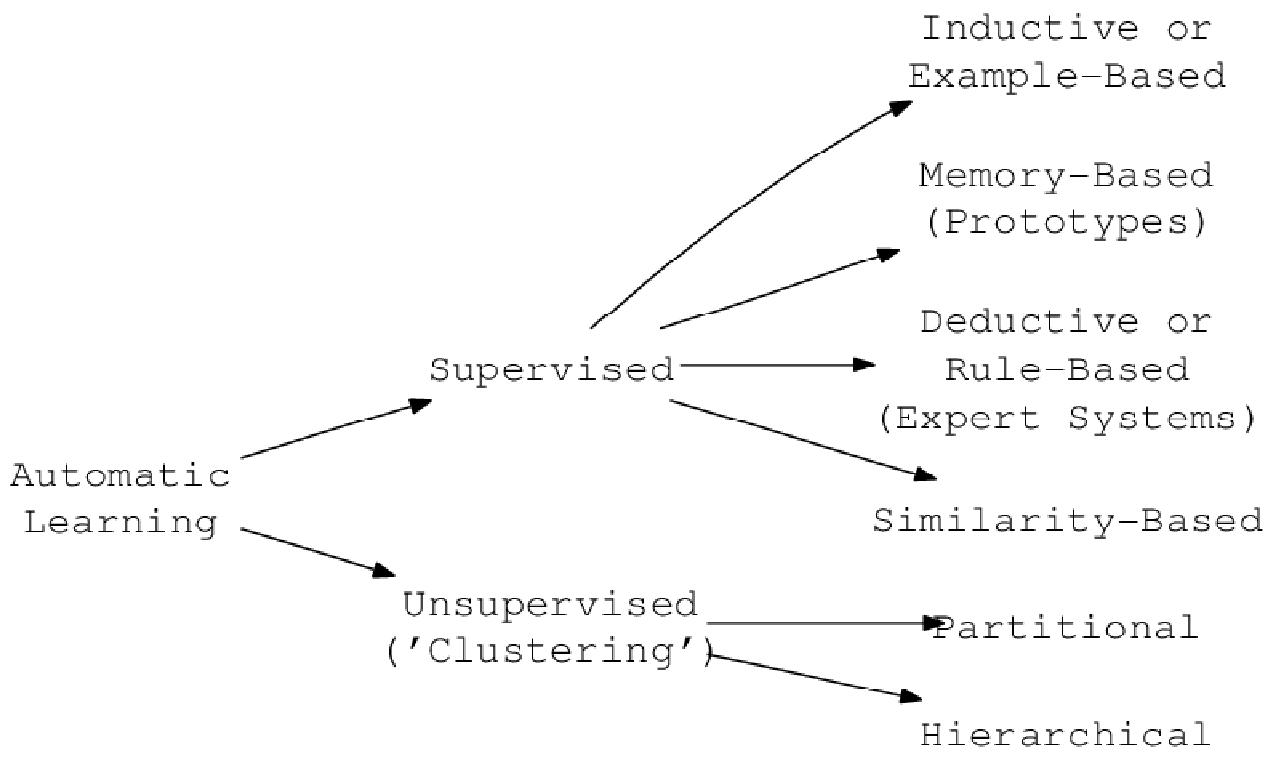


- $s(x)$: observable *sign* of an object x
- $y(x)$: *representation* of x as a set of *features* describing its properties
- *Response*: a *class label*, a complex structural information (word sequences in speech recognition, graphs of relationships among objects of an artificial vision image, . . .)
- *Learning*: on the basis of pairs *input–output* + domain *knowledge* of the task

Design of a PR System: a different diagram



Automatic learning techniques taxonomy


Automatic Learning
Supervised
Unsupervised ('Clustering')
Inductive or Example–Based
Memory–Based (Prototypes)
Deductive or Rule–Based (Expert Systems)
Similarity–Based
Partitional
Hierarchical

Syllabus

Features extraction and Representation space

Representation space:

- Space: usually a feature vector in a *n*-dimensional feature space
- An object representation is elicited via *pre-processing techniques* and *feature extraction*

Feature extraction: desirable properties

- *Continuity and discriminative capacity*: The similarity between the representations of two objects must be in direct correspondence with the similarity with which the objects are perceived:

  objects of the same class should have similar representations, while objects of different classes should have different representations.

- *Invariance under common transformations and distortions*: Different instances of the same object should have similar representations.

A classical example: Iris flowers classification

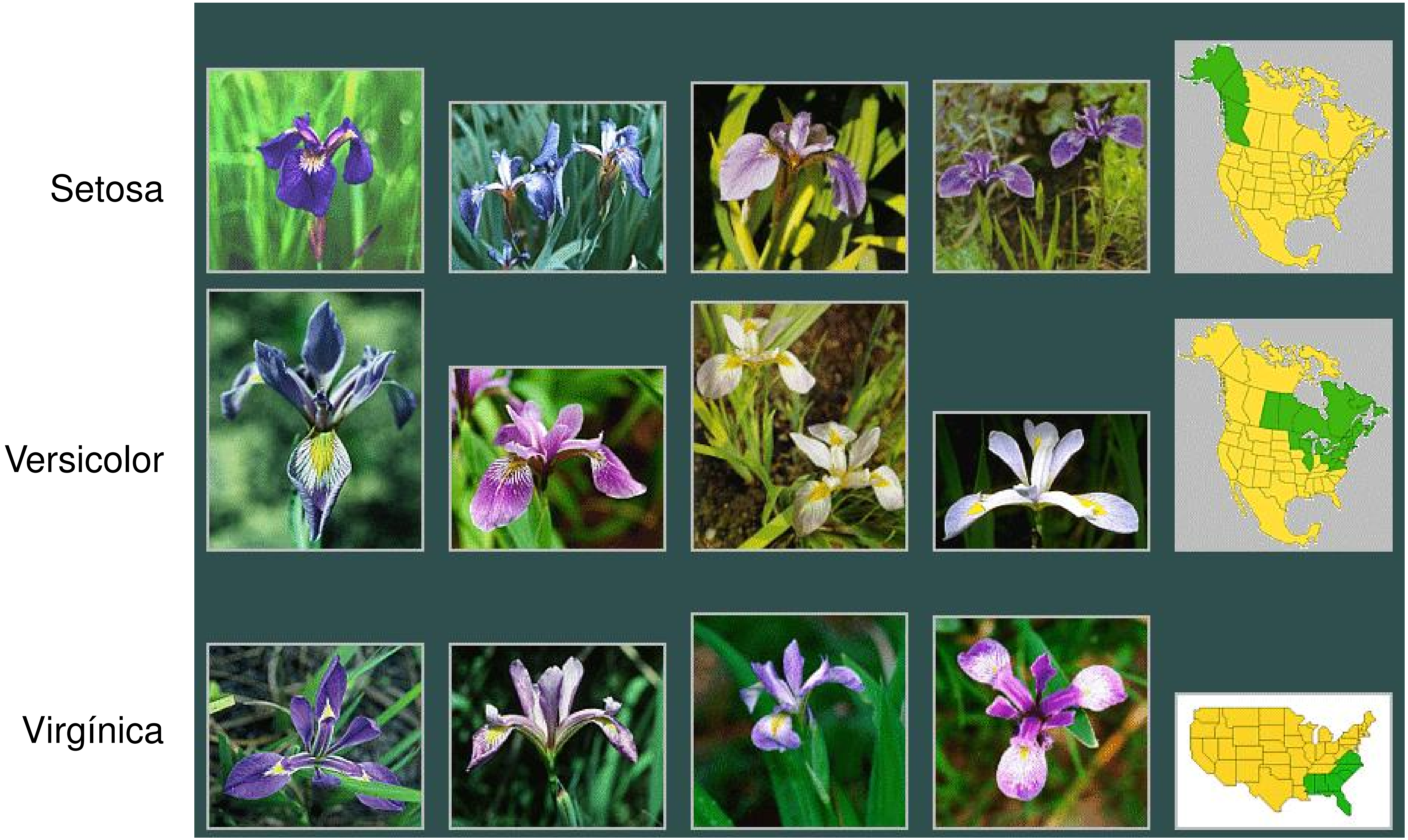
- A typical 'academic' problem, introduced by Fisher in 1936.
- The task is to classify correctly flower specimens of the *iris* family using the information of their *petal* and *sepal* sizes
- The data set contains the measures of 150 specimens of three subclasses: *Setosa*, *Versicolor* and *Virginica*
- It is frequently used as an example task for comparing the performance and possibilities of different methods of data analysis and pattern recognition

Setosa

Versicolor

Virgínica

Specimens of three varieties of iris flowers



Iris: representation space

Feature vectors in $\mathbb{R}^4$ (4 components) from three classes of iris flower. Components:

SEPAL LENGTH SEPAL WIDTH PETAL LENGTH PETAL WIDTH

Iris Setosa

5.1	3.5	1.4	0.2
4.9	3.0	1.4	0.2
4.7	3.2	1.3	0.2
4.6	3.1	1.5	0.2
5.0	3.6	1.4	0.2

. . .

Iris Versicolor

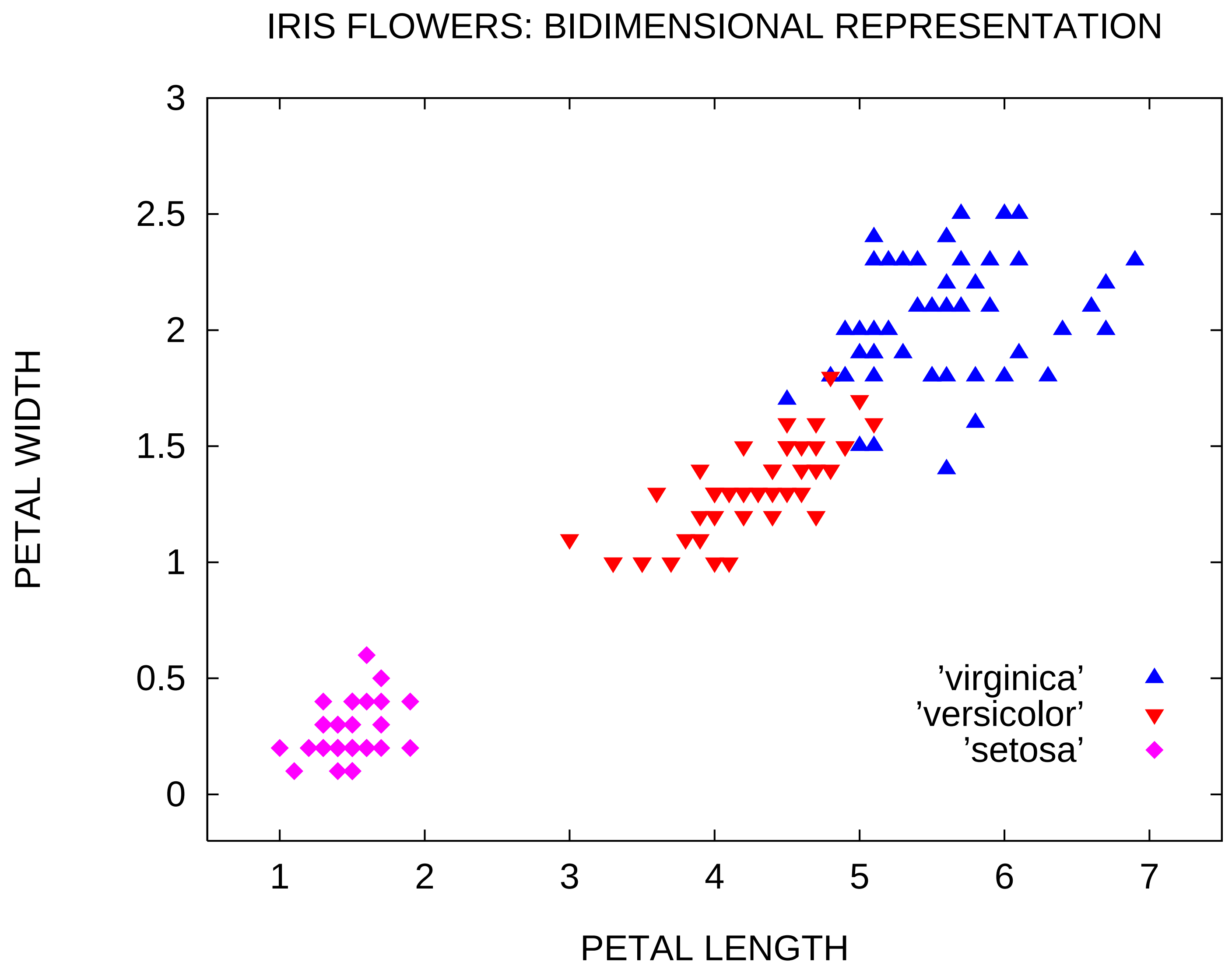
7.0	3.2	4.7	1.4
6.4	3.2	4.5	1.5
6.9	3.1	4.9	1.5
5.5	2.3	4.0	1.3
6.5	2.8	4.6	1.5

. . .

Iris Virgínica

6.3	3.3	6.0	2.5
5.8	2.7	5.1	1.9
7.1	3.0	5.9	2.1
6.3	2.9	5.6	1.8
6.5	3.0	5.8	2.2

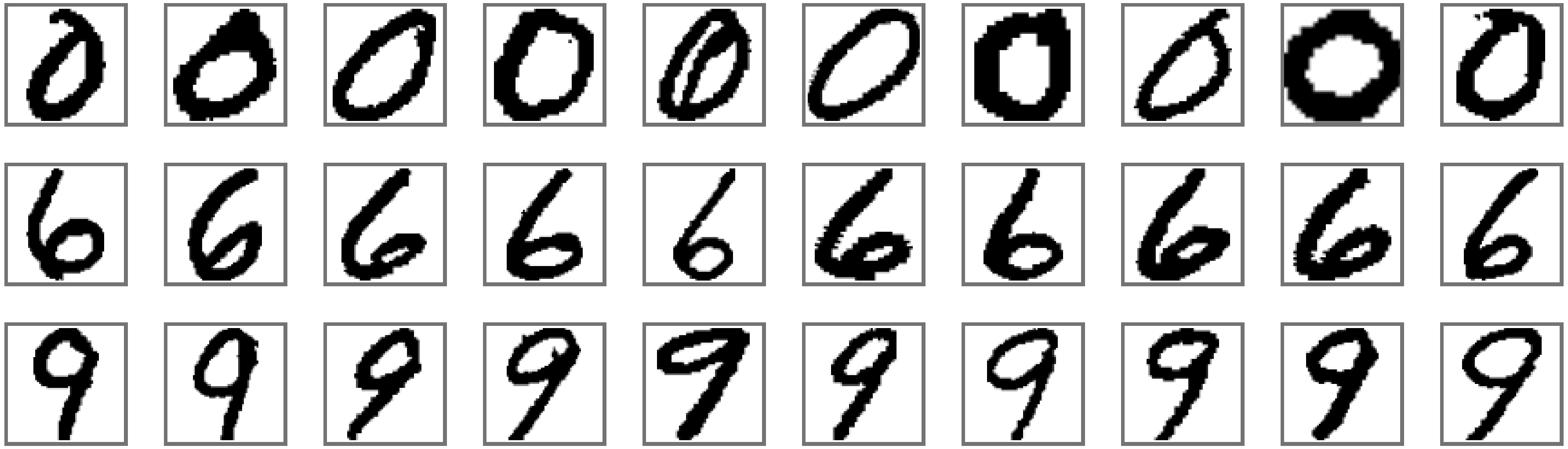
. . .



Another example: Handwritten characters (OCR task)

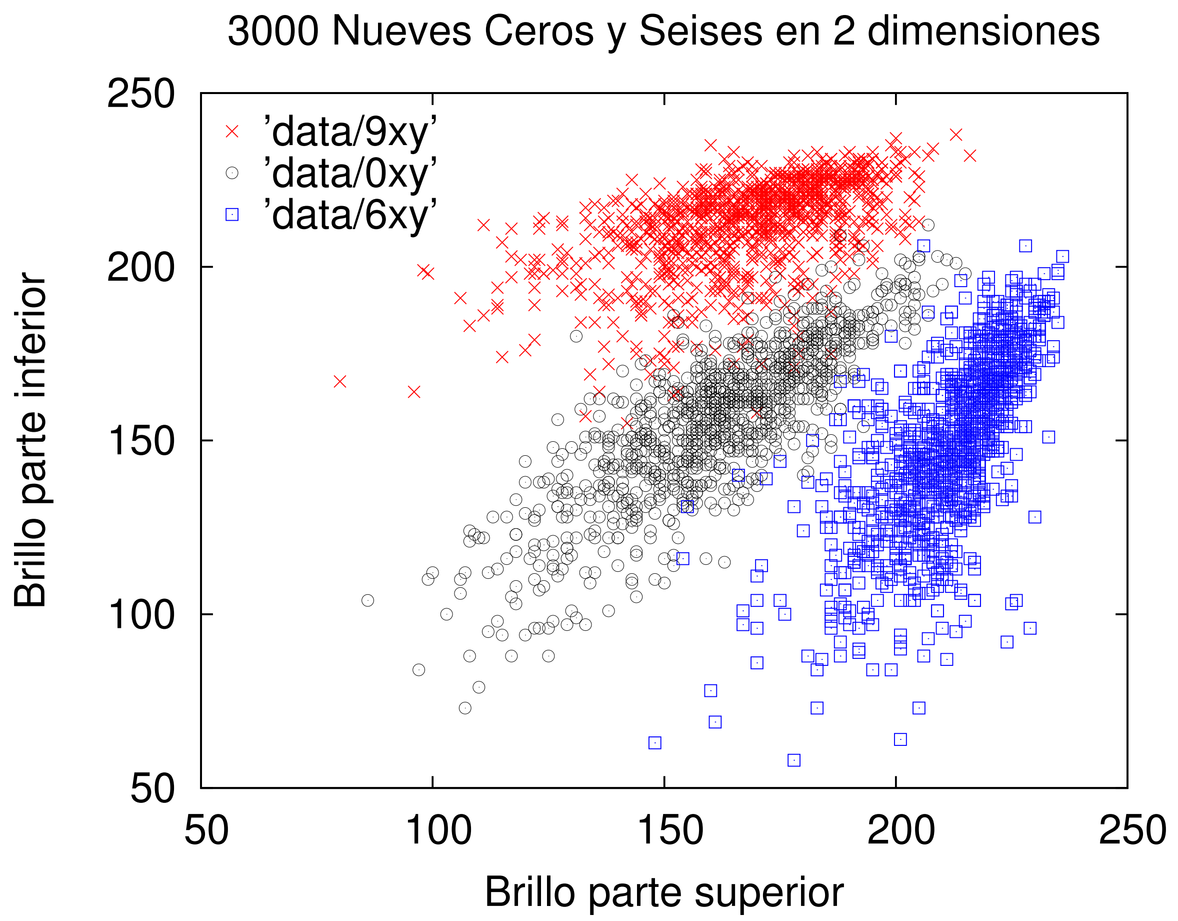
Examples of normalized images of the handwritten digits
zero, six and *nine*.

Images of *six* are *brighter* in the upper part,
and images of *nine* are brighter in the lower part.



Another example: Handwritten characters (OCR task)

3000 examples in a $\mathbb{R}^2$ space corresponding to two features: *bright in the middle upper part* of the image, and *bright in the middle lower part* of the image.



Classes, representation space and classifier: notation

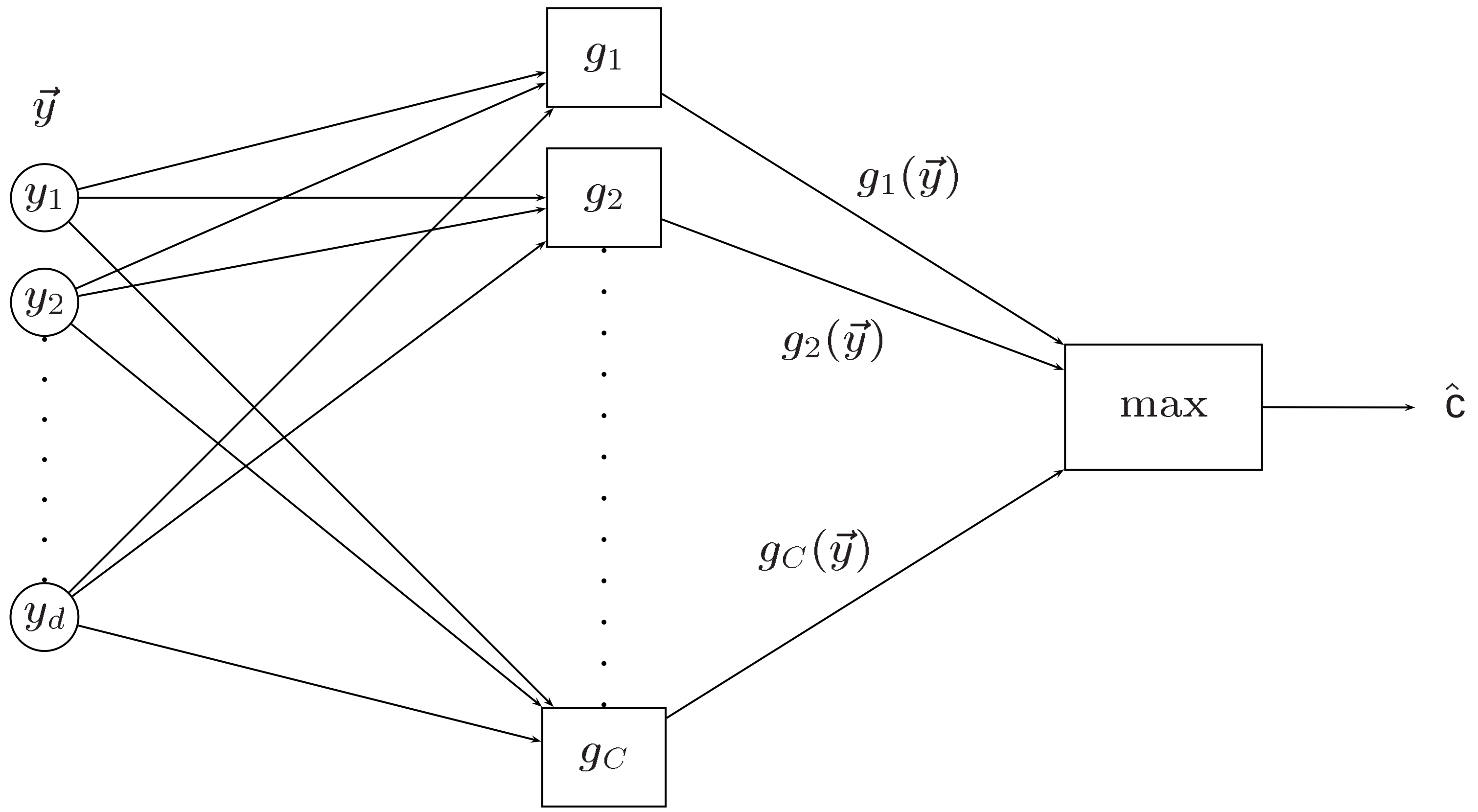
- ***Classes***: $\mathbb{C}$, $\mathbb{C} = \{1, 2, \dots, C\}$ unless otherwise specified
  - Each *object* (or its signal) is shown in a *Primary Space* or 'Universe', U
  - Let's assume that each object $x \in U$ belongs to a unique class $c(x) \in \mathbb{C}$
  - $\mathbb{C}$ is the set of all possible *identifiers* or *class labels*
- ***Representation Space***: E, generally $E = \mathbb{R}^D$
  - Let $\boldsymbol{y} = \boldsymbol{y}(x)$ be the result of the preprocessing and feature extraction process applied to an object $x \in U$
  - E encloses all the possible results: $\{\boldsymbol{y} : \boldsymbol{y} = \boldsymbol{y}(x),\ x \in U\} \subset E$
  - Given that two different objects from U can have the same representation in E, it is not guaranteed that each point in E belongs to a unique class.
- ***Classifier***: $G : E \to \mathbb{C}$
  - G is learnt with N *labelled samples* $(\boldsymbol{y}_1, c_1), \dots, (\boldsymbol{y}_N, c_N) \in E \times \mathbb{C}$
  - For a new object $x \in U$, its class is estimated as $\hat{c} = \hat{c}(x) = G(\boldsymbol{y}(x))$. The goal is to obtain the correct class; that is, $\hat{c} = c(x)$, the maximum number of times as possible

Syllabus

Classifiers and Discriminant Functions (DF)

Every classifier G into C classes can be stated in terms of C ***discriminant functions*** $g_c : E \to \mathbb{R}, \;\; 1 \leq c \leq C,$ and the corresponding *classifying rule*:

$$G = (g_1, g_2, \ldots, g_C), \qquad \hat{c} = G(\boldsymbol{y}) \equiv \underset{1 \leq c \leq C}{\text{argmax}} \;\; g_c(\boldsymbol{y})$$



Decision or classification boundaries

Any classifier partitions the representation space into C ***decision regions***, $R_1, \ldots, R_C$:

$$R_j = \{\mathbf{y} \in E : \;\; g_j(\mathbf{y}) > g_i(\mathbf{y}) \;\;\; i \neq j, \; 1 \leq i \leq C\}$$

- ***Decision boundary between two classes*** i, j:
  Geometric place of the points $\mathbf{y} \in E$ for which $g_i(\mathbf{y}) = g_j(\mathbf{y})$

  In general, they are *Hyperareas* defined by the equations:

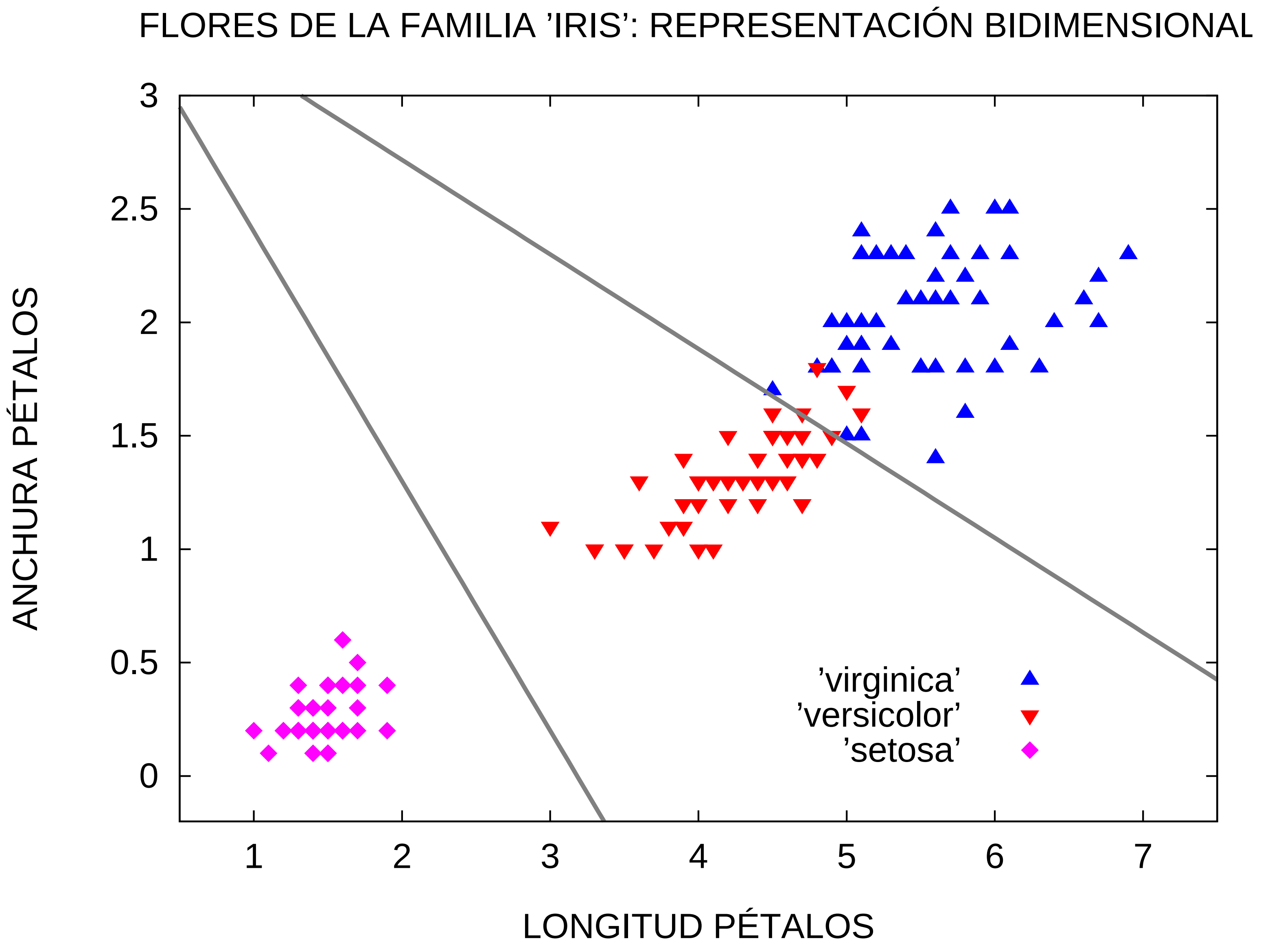
  $$g_i(\mathbf{y}) - g_j(\mathbf{y}) = 0 \qquad i \neq j, \;\; 1 \leq i, j \leq C$$

  - If $E \equiv \mathbb{R}^3$, the boundaries are surfaces (ex. *planes*)
  - If $E \equiv \mathbb{R}^2$, the boundaries are lines (ex. *straight lines*)
  - If $E \equiv \mathbb{R}$, the boundaries are points

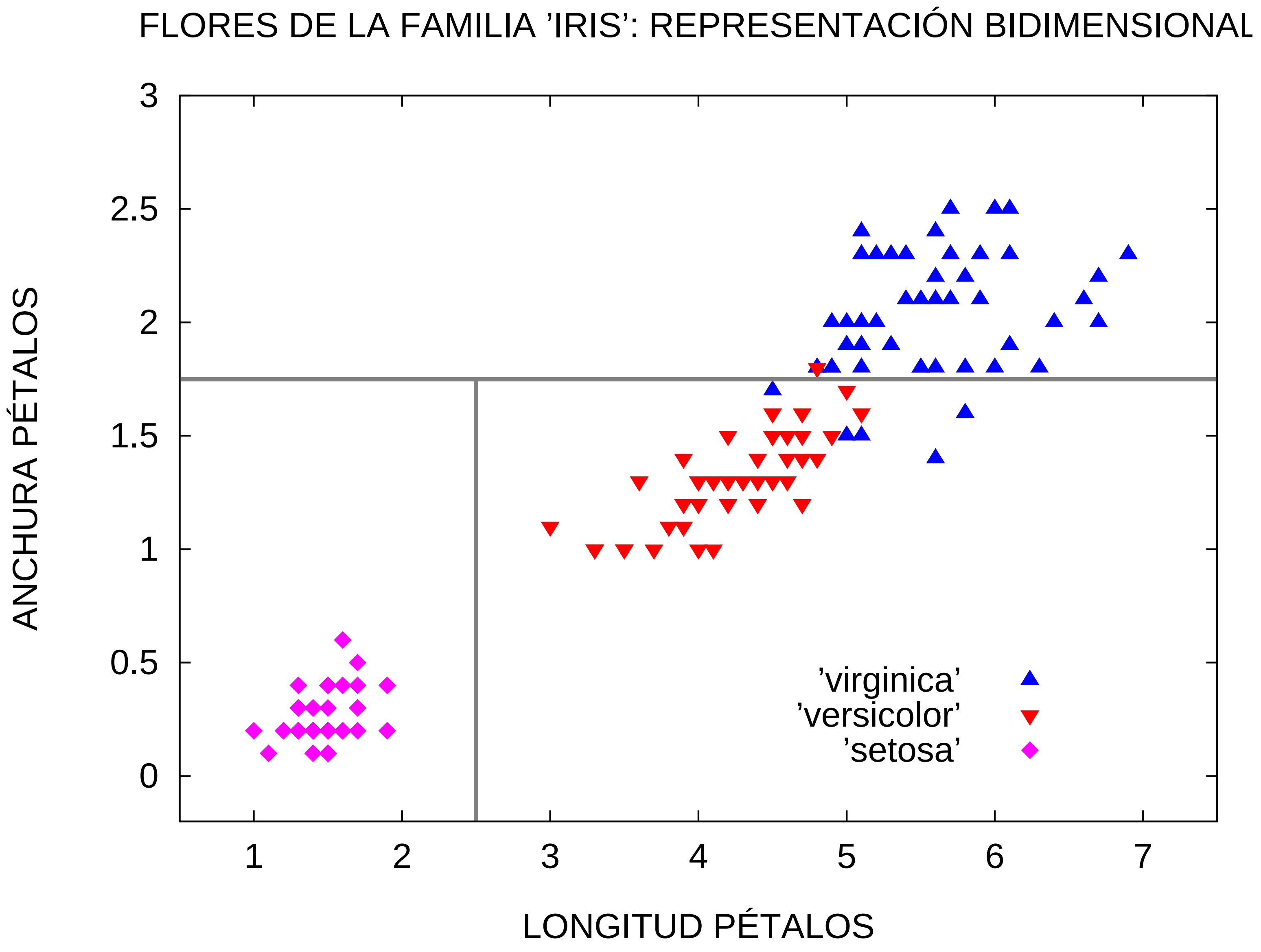
- ***Decision boundary of a single class*** i:
  Geometric place of the points $\mathbf{y} \in E$ for which:

  $$g_i(\mathbf{y}) \;\; = \;\; \max_{j \neq i} \; g_j(\mathbf{y})$$

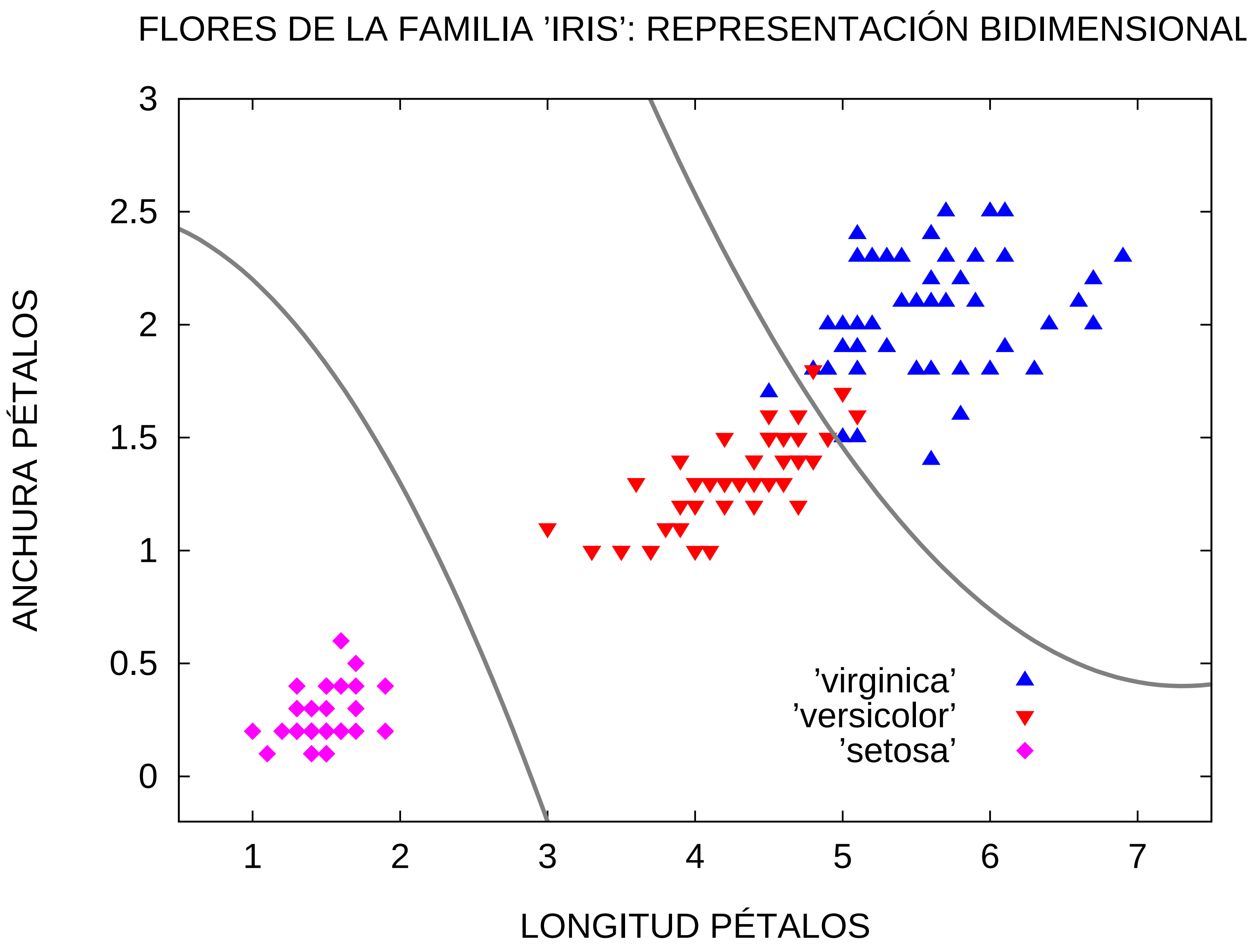
Decision boundary for the iris flowers task: linear boundaries



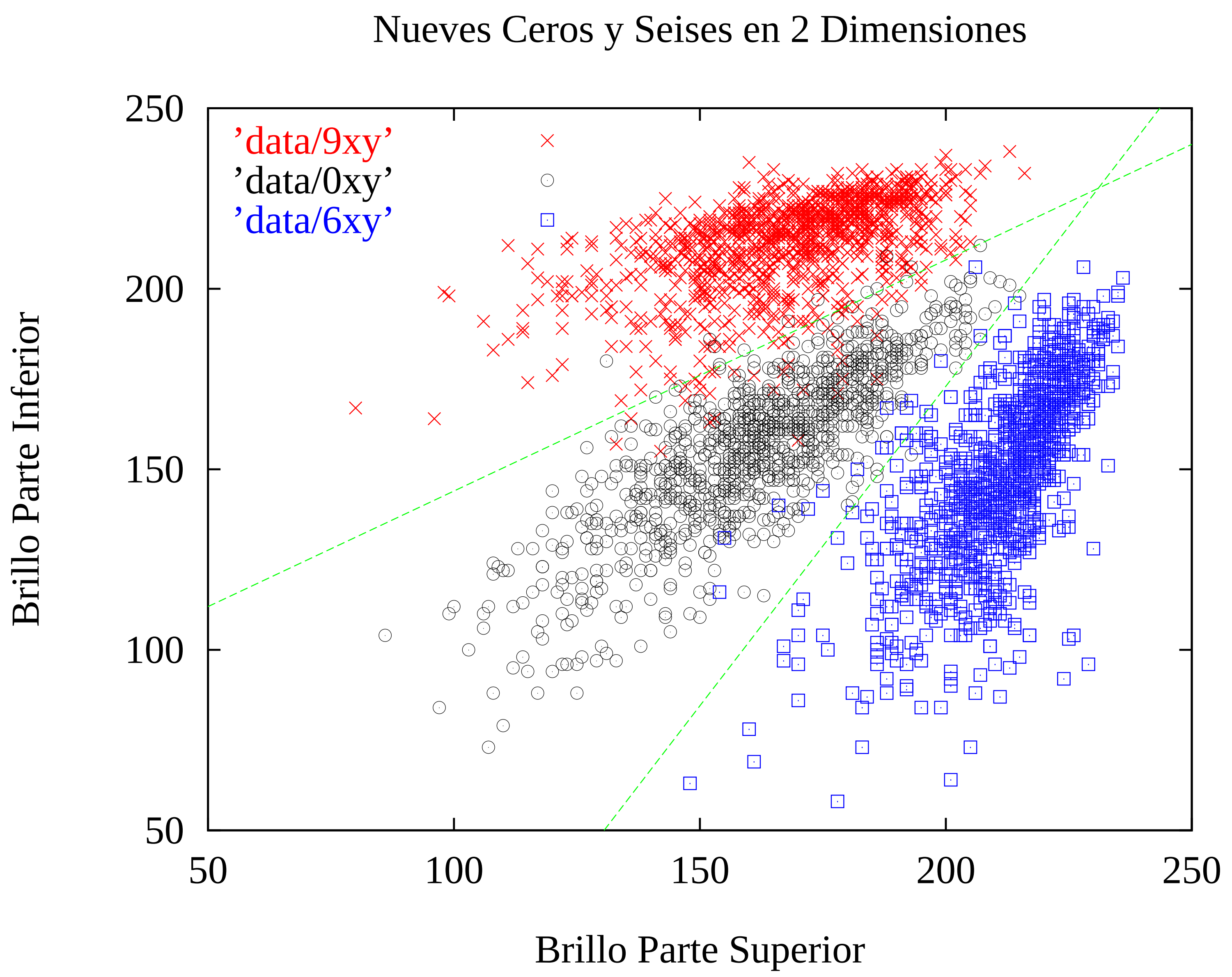
Decision boundary for the iris flowers task: linear boundaries that are parallel to the axes



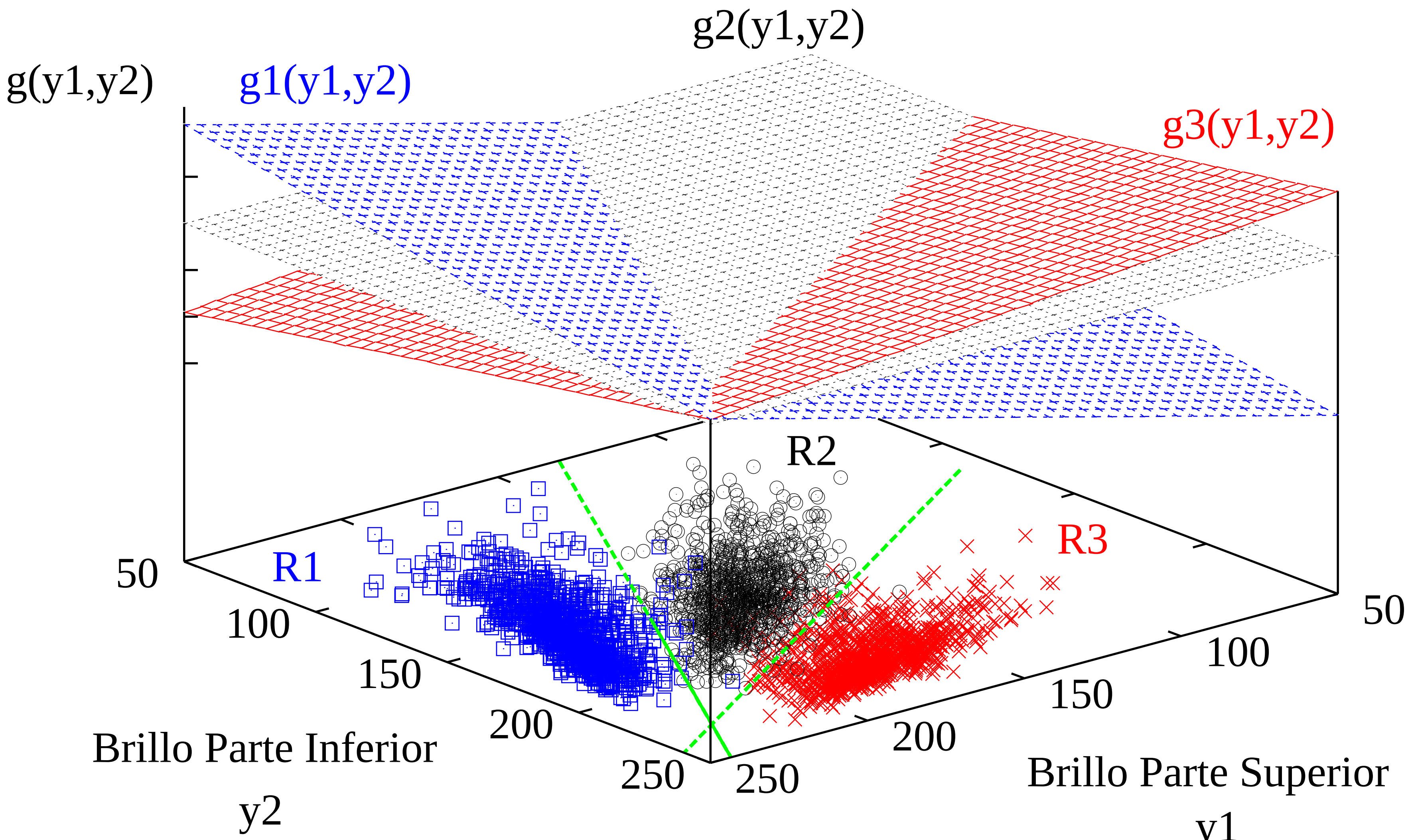
Decision boundary for the iris flowers task: quadatric boundaries



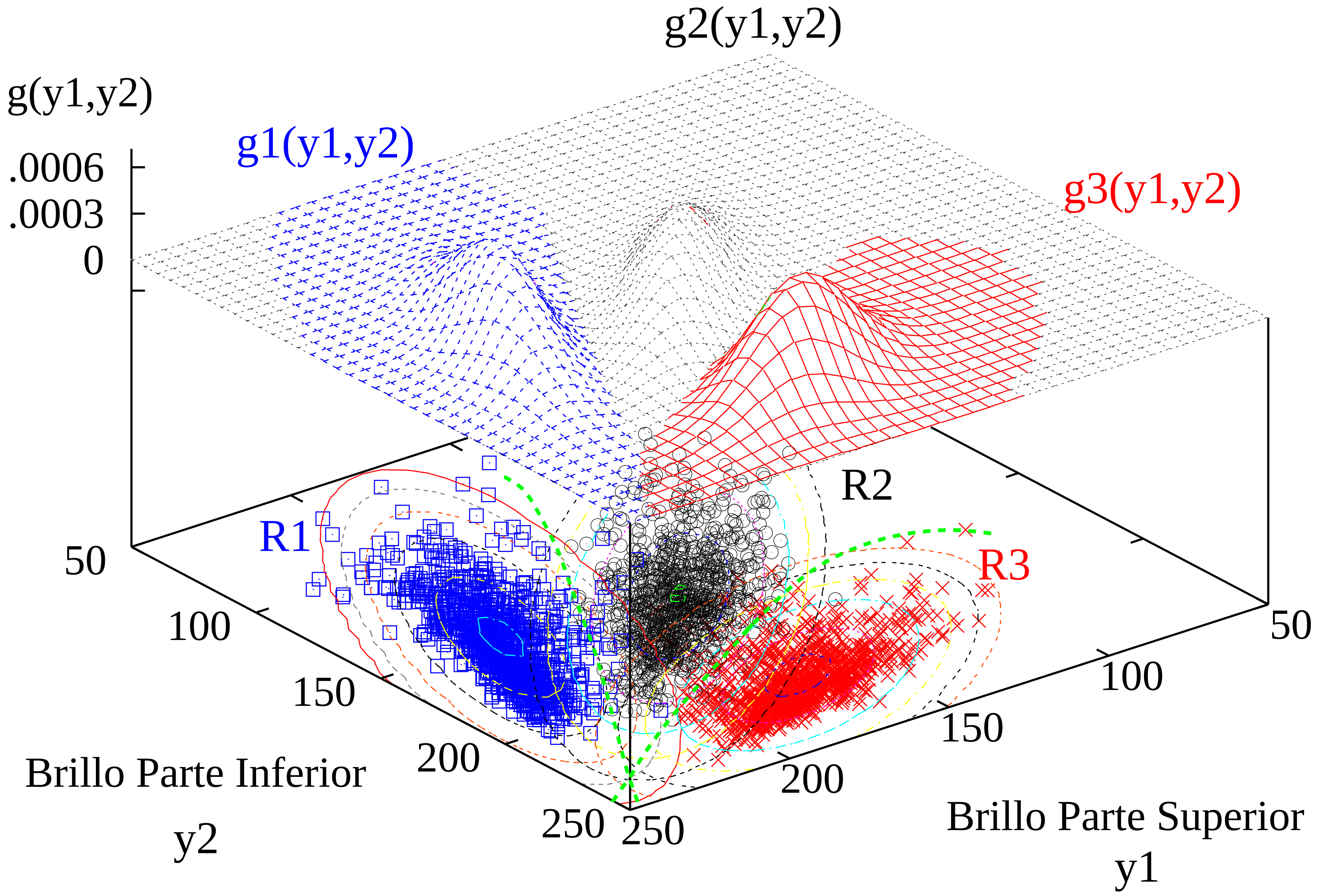
Decision boundaries for an image classifier of handwritten digits (bi-dimensional representation space)



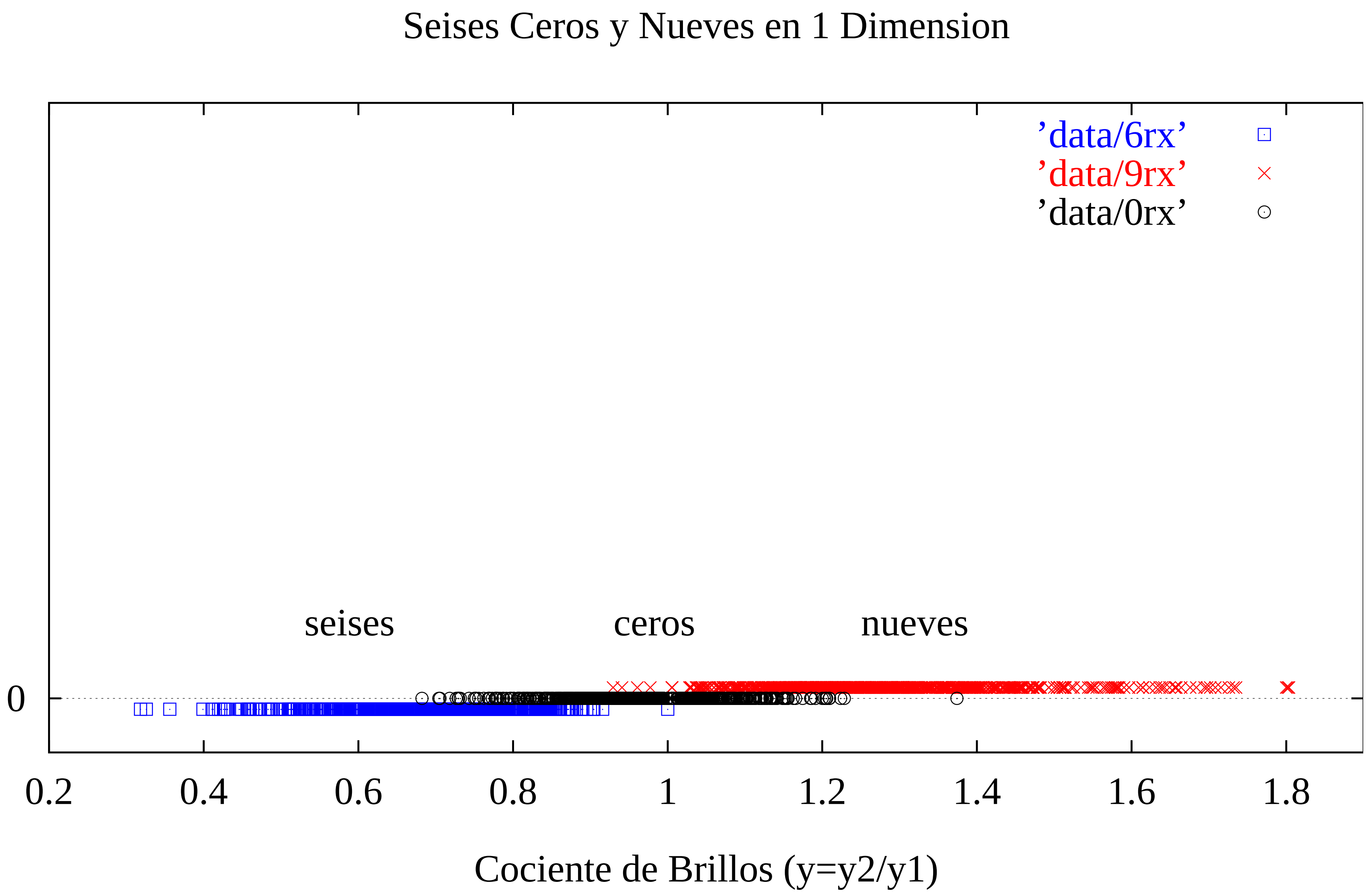
Fronteras de de Decisión y Funciones Discriminantes (lineales)



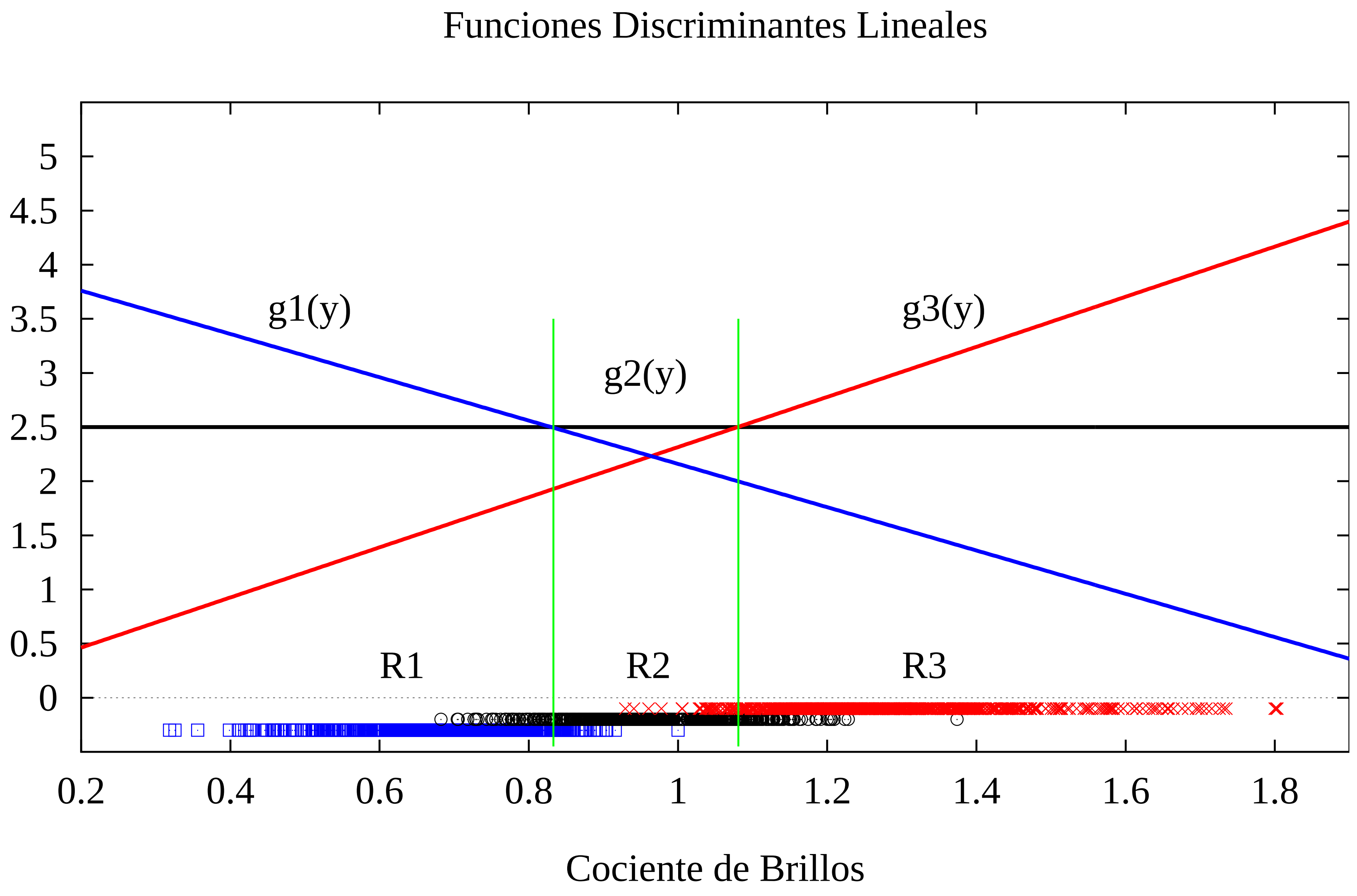
Fronteras de Decisión y Funciones Discriminantes (Gausianas)



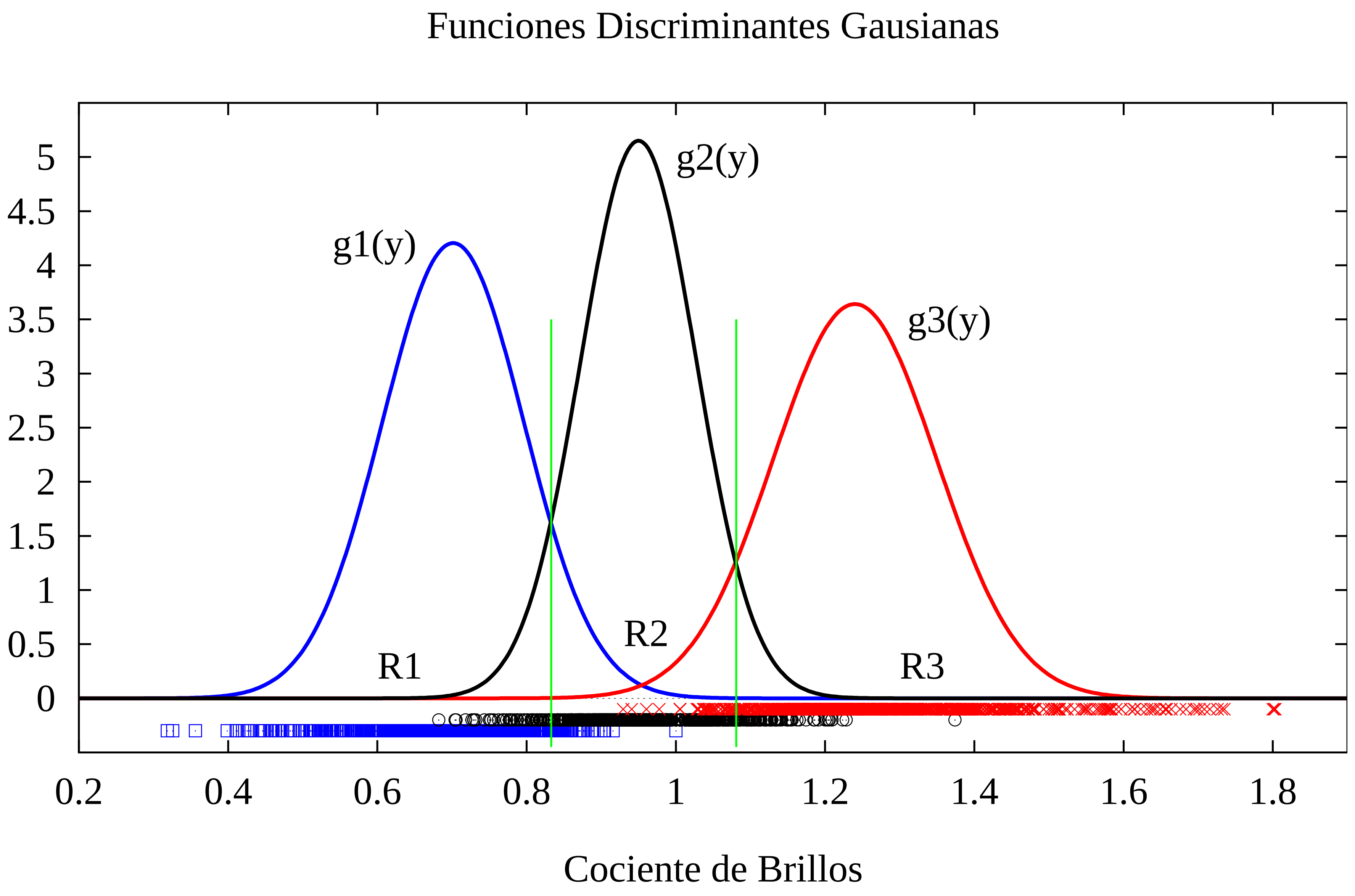
One-dimension examples



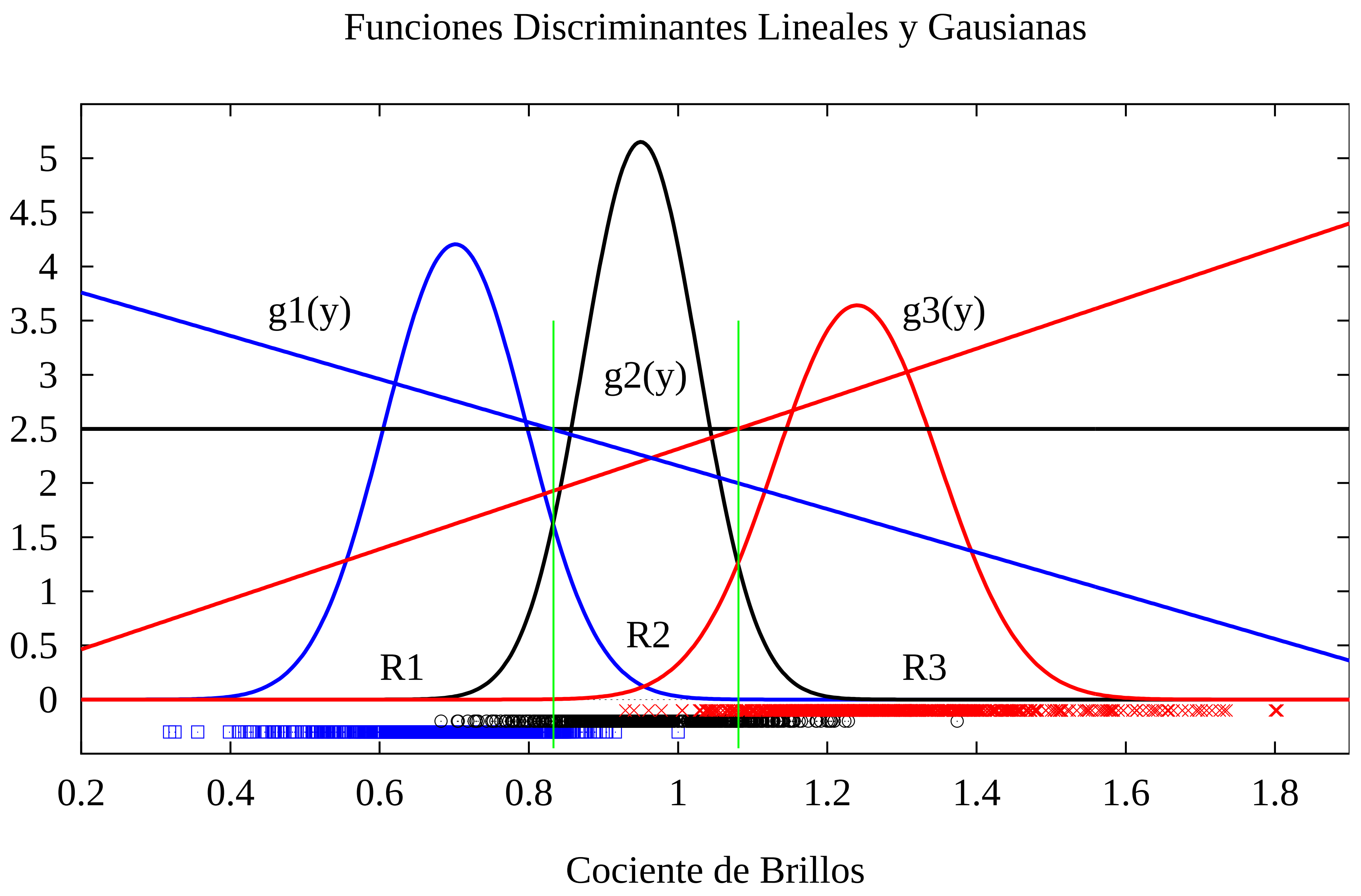
Linear discriminant functions and decision boundaries



Gaussian discriminant functions and decision boundaries



Equivalence between discriminant functions



Equivalent classifiers

Two classifiers $(g_1, \ldots, g_C)$ and $(g'_1, \ldots, g'_C)$ are equivalent if they infer the same decision boundaries; that is:

$$g_i(\mathbf{y}) > g_j(\mathbf{y}) \Leftrightarrow g'_i(\mathbf{y}) > g'_j(\mathbf{y}) \qquad \forall j \neq i, \;\; \forall \mathbf{y} \in E$$

Let $f : \mathbb{R} \to \mathbb{R}$ be any monotonically increasing function. Then, the following classifiers are equivalent:

$$(g_1, \ldots, g_C), \qquad (f(g_1), \ldots, f(g_C))$$

Examples:

$f(z) = az + b$ with $a > 0$

$f(z) = \log z$ with $g_i(\mathbf{y}) > 0$, $1 \leq i \leq C$

Linear and Quadratic discriminant functions

Linear discriminant functions:

$$g(\mathbf{y}) \;=\; \sum_{i=1}^{d} a_i y_i + a_0 \;=\; \mathbf{a}^t \mathbf{y} + a_0$$

$\mathbf{a}$ is named *weight vector* and a_0 *threshold weight* or *bias*.

- Number of parameters: $d+1$ (*linear* relation with d)
- In general, *the decision boundaries are hyperplanes*.

Quadratic discriminant functions:

$$g(\mathbf{y}) \;=\; \sum_{i=1}^{d} \sum_{j=1}^{d} a_{ij} y_i y_j + \sum_{i=1}^{d} a_i y_i + a_0 \;=\; \mathbf{y}^t A \mathbf{y} + \mathbf{a}^t \mathbf{y} + a_0$$

A is a matrix named *weight matrix* and a_0 *threshold weight*.

- Number of parameters: *quadratic* relation with d.
- In general, *the decision boundaries are hyperquadratic*.

Syllabus

Linear Discriminant Functions (LDF)

A classifier is *linear* if its Discriminant Functions are *linear functions* of the vectors in E. Let $\boldsymbol{y} \in E \equiv \mathbb{R}^D$ be the representation of any object:

$$g_c(\boldsymbol{y}) \;=\; \sum_{j=1}^{D} a_{cj} \cdot y_j + a_{c0} \;=\; \boldsymbol{a}_c{}^t \boldsymbol{y} + a_{c0}, \;\; 1 \leq c \leq C$$

Linear Discriminant Functions (LDF)

A classifier is *linear* if its Discriminant Functions are *linear functions* of the vectors in E. Let $\boldsymbol{y} \in E \equiv \mathbb{R}^D$ be the representation of any object:

$$g_c(\boldsymbol{y}) \;=\; \sum_{j=1}^{D} a_{cj} \cdot y_j + a_{c0} \;=\; \boldsymbol{a}_c^{\,t}\boldsymbol{y} + a_{c0}, \quad 1 \leq c \leq C$$

Compact notation (*note the changes in the letters* $\boldsymbol{a} \rightarrow \mathbf{a},\ \boldsymbol{y} \rightarrow \mathbf{y}$):

$$\mathbf{y} = (1, y_1, ..., y_D)^t, \quad \mathbf{a}_c = (a_{c0}, a_{c1}, \cdots, a_{cD})^t$$

$$g_c(\boldsymbol{y}) = \mathbf{a}_c^{\,t}\mathbf{y}$$

Linear Discriminant Functions (LDF)

A classifier is *linear* if its Discriminant Functions are *linear functions* of the vectors in E. Let $\boldsymbol{y} \in E \equiv \mathbb{R}^D$ be the representation of any object:

$$g_c(\boldsymbol{y}) \;=\; \sum_{j=1}^{D} a_{cj} \cdot y_j + a_{c0} \;=\; \boldsymbol{a}_c{}^t \boldsymbol{y} + a_{c0}, \;\; 1 \leq c \leq C$$

Compact notation (*note the changes in the letters* $\boldsymbol{a} \rightarrow \mathbf{a},\; \boldsymbol{y} \rightarrow \mathbf{y}$):

$$\mathbf{y} = (1, y_1, ..., y_D)^t, \;\; \mathbf{a}_c = (a_{c0}, a_{c1}, \cdots, a_{cD})^t$$

$$g_c(\boldsymbol{y}) = \mathbf{a}_c{}^t \mathbf{y}$$

Classification rule:

$$\hat{c} = G(\boldsymbol{y}) \equiv \operatorname*{argmax}_{1 \leq c \leq C} \; \mathbf{a}_c{}^t \mathbf{y}$$

Linear Discriminant Functions (LDF)

A classifier is *linear* if its Discriminant Functions are *linear functions* of the vectors in E. Let $\boldsymbol{y} \in E \equiv \mathbb{R}^D$ be the representation of any object:

$$g_c(\boldsymbol{y}) \;=\; \sum_{j=1}^{D} a_{cj} \cdot y_j + a_{c0} \;=\; \boldsymbol{a}_c^{\,t}\boldsymbol{y} + a_{c0}, \;\; 1 \le c \le C$$

Compact notation (*note the changes in the letters* $\boldsymbol{a} \to \mathbf{a},\; \boldsymbol{y} \to \mathbf{y}$):

$$\mathbf{y} = (1, y_1, ..., y_D)^t, \;\; \mathbf{a}_c = (a_{c0}, a_{c1}, \cdots, a_{cD})^t$$

$$g_c(\boldsymbol{y}) = \mathbf{a}_c^{\,t}\mathbf{y}$$

Classification rule:

$$\hat{c} = G(\boldsymbol{y}) \equiv \underset{1 \le c \le C}{\text{argmax}} \;\; \mathbf{a}_c^{\,t}\mathbf{y}$$

The *decision boundary* between any pair of classes i, j: $\mathbf{a}_i^{\,t}\mathbf{y} = \mathbf{a}_j^{\,t}\mathbf{y}$

Linear boundaries or *hyperplanes* of dimension D (lines if $D = 2$).

Syllabus

LDFs Learning [1]

Given N training samples $(\mathbf{y}_1, c_1), \dots, (\mathbf{y}_N, c_N)$, the goal is to find C weight vectors $\mathbf{a}_j,\ 1 \leq j \leq C,\ C$ that correctly classify (the most as possible) the given training samples; that is:

$$\mathbf{a}_{c_1}{}^t \mathbf{y}_1 > \mathbf{a}_j{}^t \mathbf{y}_1,\ \forall j \neq c_1$$

$$\mathbf{a}_{c_2}{}^t \mathbf{y}_2 > \mathbf{a}_j{}^t \mathbf{y}_2,\ \forall j \neq c_2$$
$$\dots$$
$$\mathbf{a}_{c_N}{}^t \mathbf{y}_N > \mathbf{a}_j{}^t \mathbf{y}_N,\ \forall j \neq c_N$$

Solution: Iteratively adjust some *initial weights* using the ***Perceptron Algorithm***, introduced in 1957 by Frank Rosenblatt.

The classifiers based on the Perceptron Algorithm can be considered as the simplest examples of *neural networks*.

There are convergence problems when the classes are not *linearly separable*.
Solutions: ’margin’ and/or ’Pocket-Perceptron’.

DLFs learning: Perceptron Algorithm

```
// Let: $\mathbf{a}_j,\ 1 \le j \le C,\quad C$ initial weight vectors;
//        $(\mathbf{y}_1, c_1), \dots, (\mathbf{y}_N, c_N),\quad N$ training samples;
//        $\alpha \in \mathbb{R}^{>0},\quad$ 'learning rate';
//        $b \in \mathbb{R},\quad$ 'margin' (to tune the convergence).

do {
   $m = 0$   // number of correctly classified samples
   for $(n = 1;\ n \le N;\ n$++) {
      $i = c_n$; $g = \mathbf{a}_i^t\,\mathbf{y}_n$ ; error=false
      for $(j = 1;\ j \le C;\ j$++) if $(j \ne i)$
         {if $(\mathbf{a}_j^t\,\mathbf{y}_n + b > g)$ {$\mathbf{a}_j = \mathbf{a}_j - \alpha\,\mathbf{y}_n$ ; error=true}}
      if (error) $\mathbf{a}_i = \mathbf{a}_i + \alpha\,\mathbf{y}_n$ ; else $m = m + 1$
   }
} while $(m < N)$
```

In case of error the algorithm updates the weight vector of all the classes that produce the error ($j:\ \mathbf{a}_j^t\,\mathbf{y}_n + b > \mathbf{a}_i^t\,\mathbf{y}_n$), and the weight vector of the correct class (i).

Perceptron Algorithm: Exercise (to do in class)

For a two-category (2 classes) problem with objects represented by a bi-dimensional feature vector, we have two training samples: $\boldsymbol{y}_1 = (0, 0)^t$, $\boldsymbol{y}_2 = (1, 1)^t$ de clases $c_1 = 1$, $c_2 = 2$, respectively.

Show a trace of the Perceptron algorithm execution with null *initial weight vectors*, *learning step* $\alpha = 1$ and *margin* $b =$0.1. The trace must show the successive updatings of the weight vectors of the classes.

Perceptron Algorithm: Exercise (to do in class)

For a two-category (2 classes) problem with objects represented by a bi-dimensional feature vector, we have two training samples: $\boldsymbol{y}_1 = (0,0)^t$, $\boldsymbol{y}_2 = (1,1)^t$ de clases $c_1 = 1$, $c_2 = 2$, respectively.

Show a trace of the Perceptron algorithm execution with null *initial weight vectors*, *learning step* $\alpha = 1$ and *margin* $b =$0.1. The trace must show the successive updatings of the weight vectors of the classes.

Summary of the solution (only the weight sequence):

The algorithm performs 3 iterations of the most external loop, thus producing the sequence of weight vectors (in compact notation):

$$\mathbf{a}_1 : \begin{pmatrix}0\\0\\0\end{pmatrix} \rightarrow \begin{pmatrix}1\\0\\0\end{pmatrix}, \begin{pmatrix}0\\-1\\-1\end{pmatrix} \rightarrow \begin{pmatrix}1\\-1\\-1\end{pmatrix}$$

$$\mathbf{a}_2 : \begin{pmatrix}0\\0\\0\end{pmatrix} \rightarrow \begin{pmatrix}-1\\0\\0\end{pmatrix}, \begin{pmatrix}0\\1\\1\end{pmatrix} \rightarrow \begin{pmatrix}-1\\1\\1\end{pmatrix}$$

Perceptron algorithm for *two* classes [1]

Classification rule for $C = 2$, with weight vectors $\mathbf{a}_1,\ \mathbf{a}_2$:

$$\hat{c} = G(\mathbf{y}) \ = \ \begin{cases} 1 \quad if \qquad \mathbf{a}_1{}^t\mathbf{y} > \mathbf{a}_2{}^t\mathbf{y} \\ 2 \quad else \ \ (\mathbf{a}_1{}^t\mathbf{y} \leq \mathbf{a}_2{}^t\mathbf{y}) \end{cases}$$

For this case, the decision boundary, the success criterion and the updating equations of the weight vectors are simplified by labelling the classes $\{1,2\}$ as $\{+1,-1\}$ and by using a single weight vector $\mathbf{a} = \mathbf{a}_1 - \mathbf{a}_2$:

Classifier:	$G(\mathbf{y}) \ = \ \text{sgn}(\mathbf{a}^t\mathbf{y})$
Decision boundary:	$\mathbf{a}^t\mathbf{y} \ = \ 0$
Success criterion:	$c_n\,\mathbf{a}^t\mathbf{y}_n \ > \ 0, \ \ n = 1, \ldots, N$
Leaning:	$\mathbf{a} \ = \ \mathbf{a} + \alpha\,(c_n - G(\mathbf{y}_n))\,\mathbf{y}_n, \ \ n = 1, \ldots, N$

Convergence and quality of results for the Perceptron algorithm

Three parameters control the convergence and quality of results:

α, b (margin), M (maximum number of iterations)

α determines the size of the corrections and therefore the *learning speed*. In general, $\alpha << \Rightarrow$ soft convergence, but with more iterations.

Convergence and quality of results for the Perceptron algorithm

Three parameters control the convergence and quality of results:

α, b (margin), M (maximum number of iterations)

α determines the size of the corrections and therefore the *learning speed*. In general, $\alpha << \Rightarrow$ soft convergence, but with more iterations.

- *With **linearly** separable training sets:*
  - Converges in a finite number of iterations $\forall \alpha > 0$
  - $b = 0$: the boundary decisions might present little 'separation', that is, they might be too close to some data
  - $b > 0$: if b is large enough, the boundary decisions are centered across the decision regions (typically much better than with $b = 0$)

Convergence and quality of results for the Perceptron algorithm

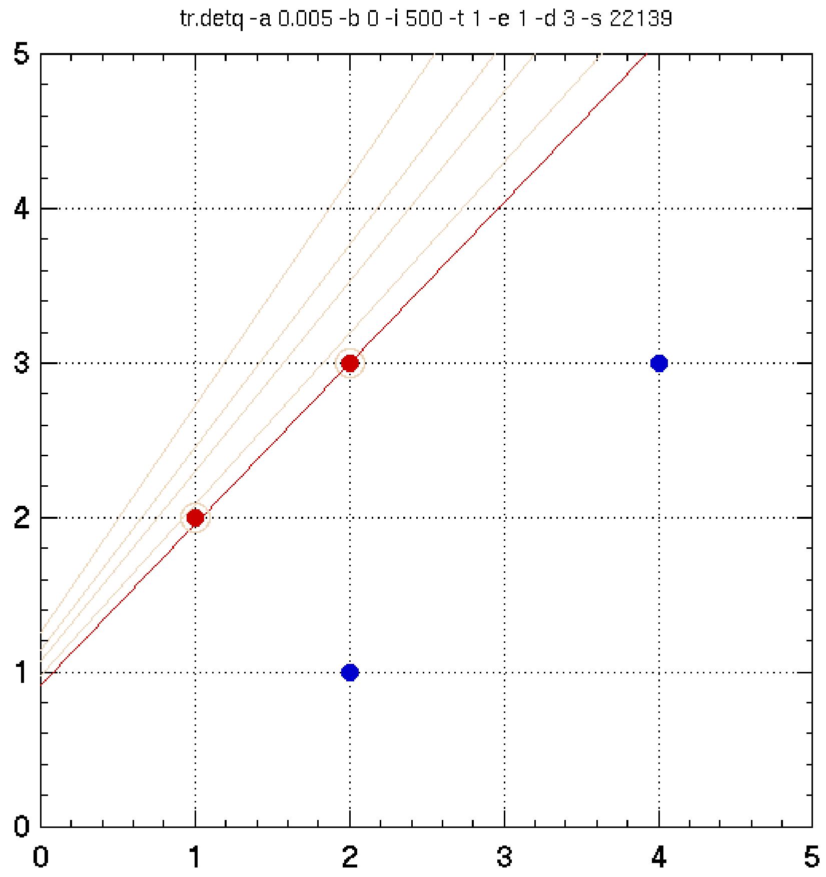
Three parameters control the convergence and quality of results:

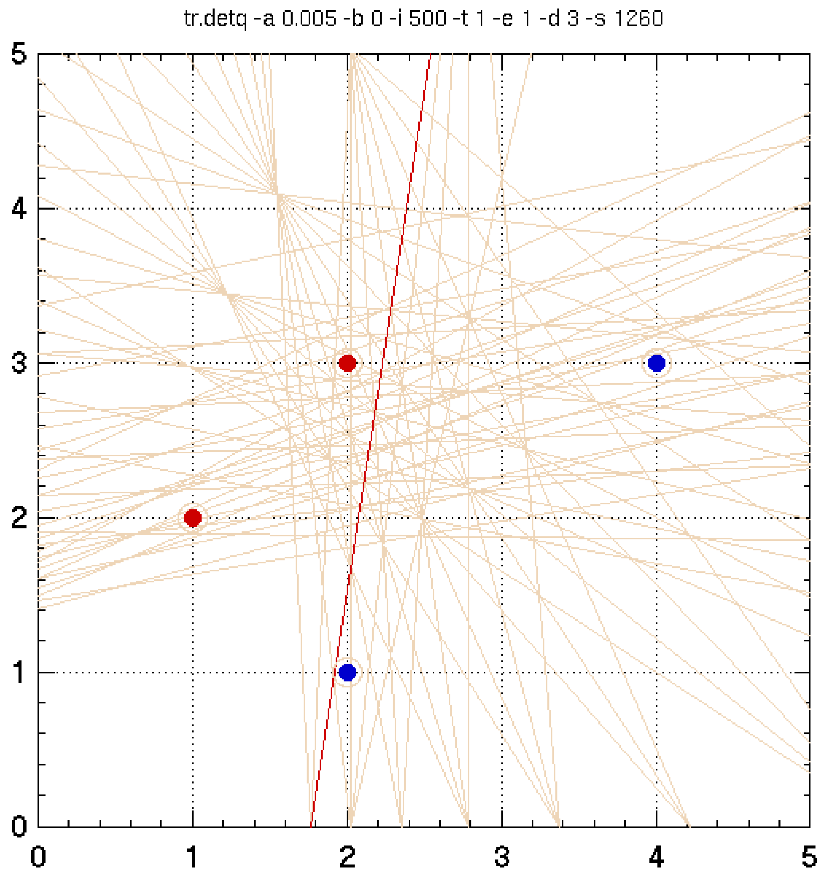
α, b (margin), M (maximum number of iterations)

α determines the size of the corrections and therefore the *learning speed*. In general, $\alpha << \Rightarrow$ soft convergence, but with more iterations.

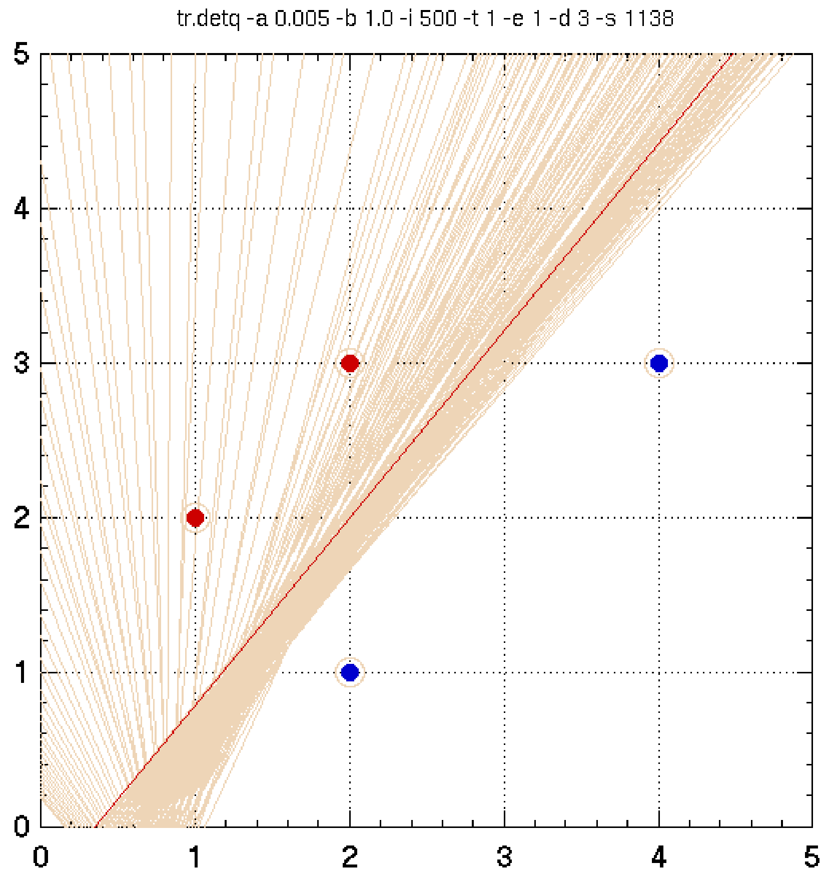
- *With **linearly** separable training sets:*
  - Converges in a finite number of iterations $\forall \alpha > 0$
  - $b = 0$: the boundary decisions might present little 'separation', that is, they might be too close to some data
  - $b > 0$: if b is large enough, the boundary decisions are centered across the decision regions (typically much better than with $b = 0$)
- *With **non-linearly** separable training sets:*
  - $b = 0$: no guarantee of convergence nor quality of the results;
  - $b > 0$: no convergence but with b and M large enough, good decision boundaries can be obtained; in general, (quasi-)*optimal*, with respect to the minimization of the classification error of the training set

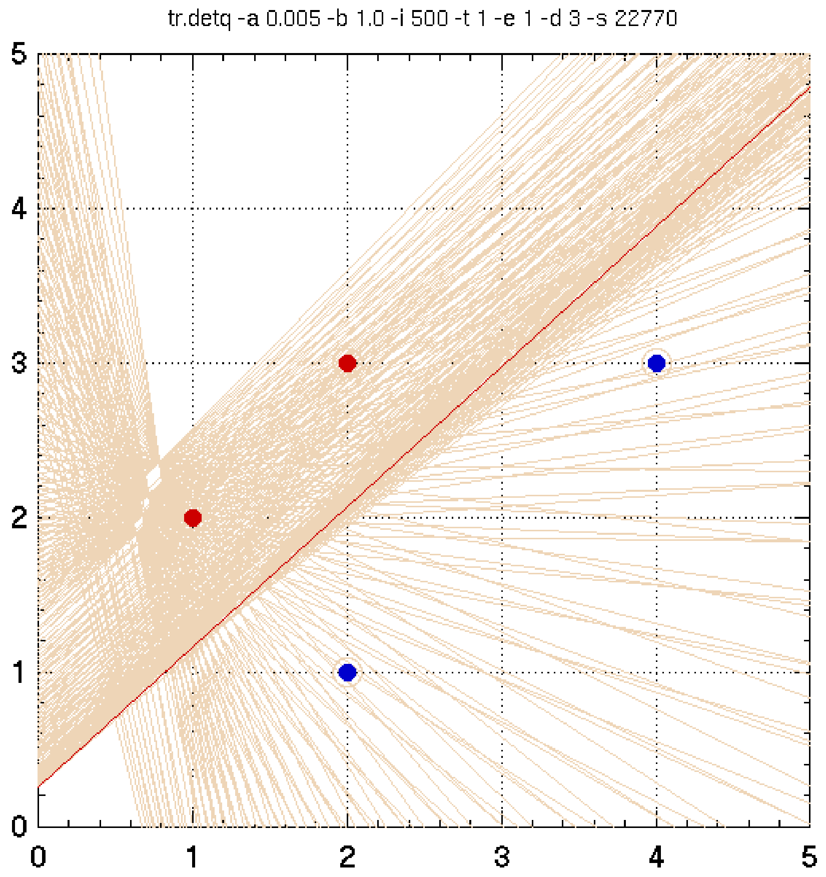
Perceptron algorithm: simple linearly separable example with *null margin*



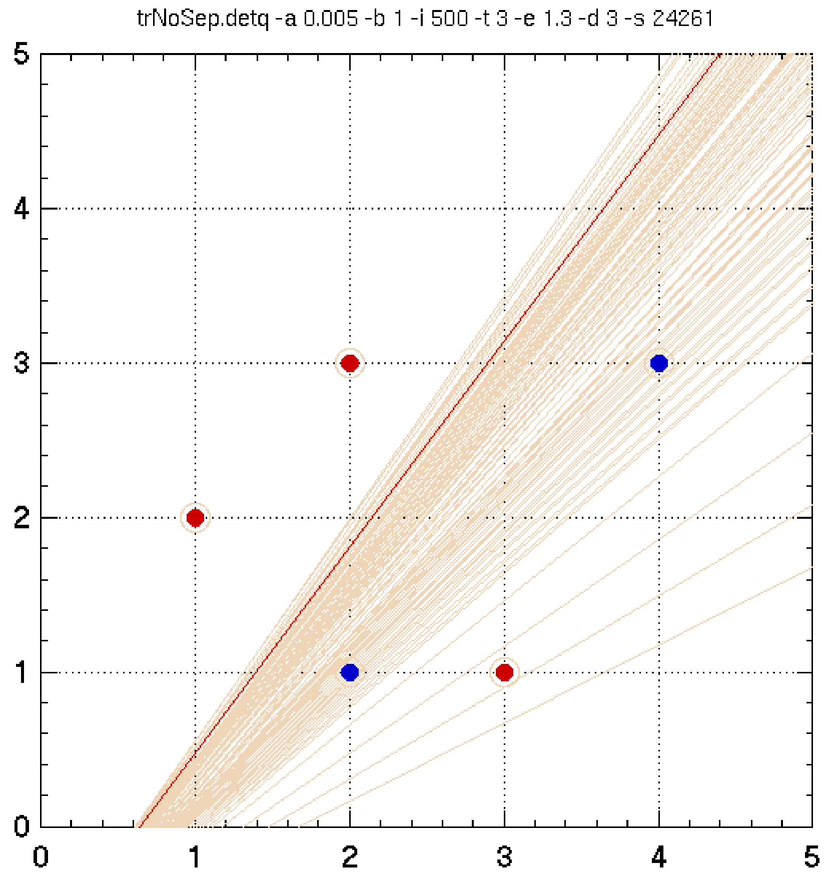


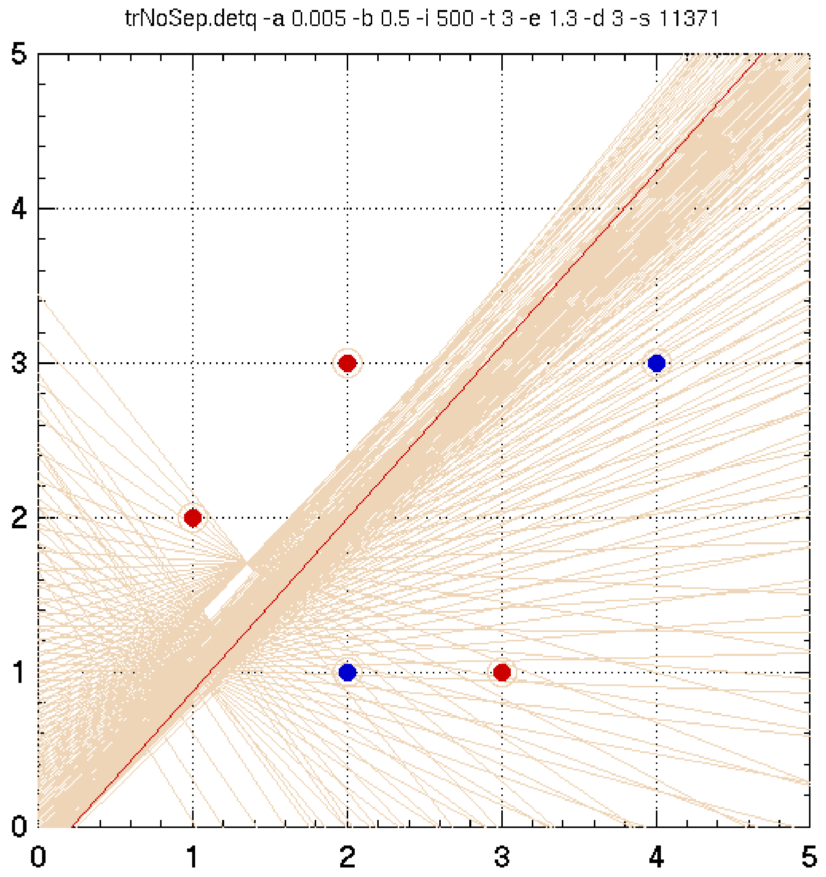
Perceptron algorithm: simple linearly separable example with *positive margin*





Perceptron algorithm: simple non-linearly separable example with *positive margin*

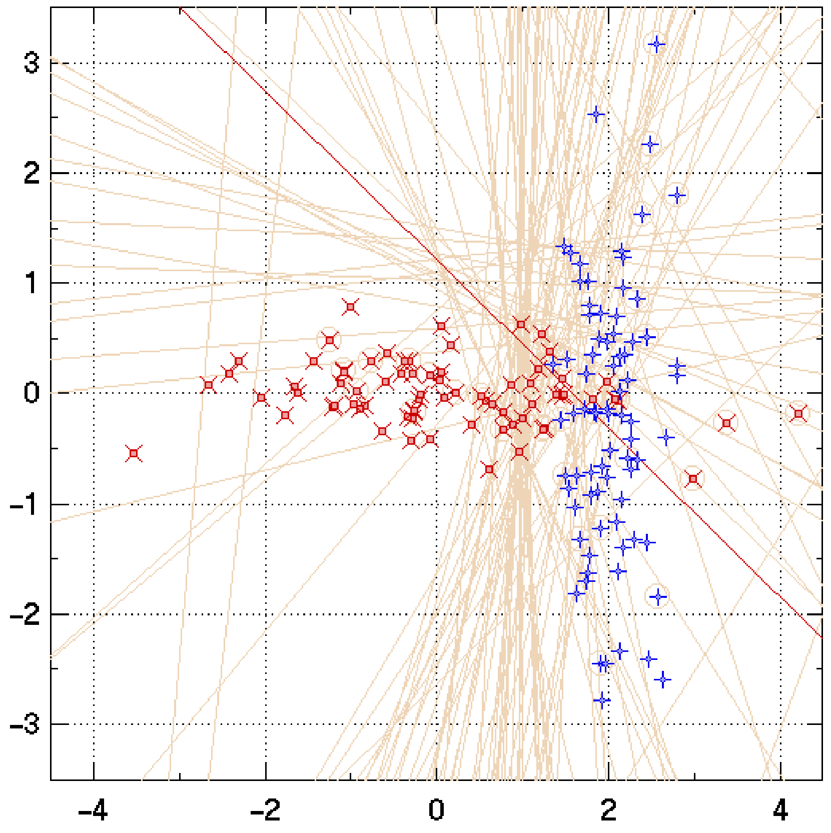




Perceptron algorithm: Gauss2D (non separable)

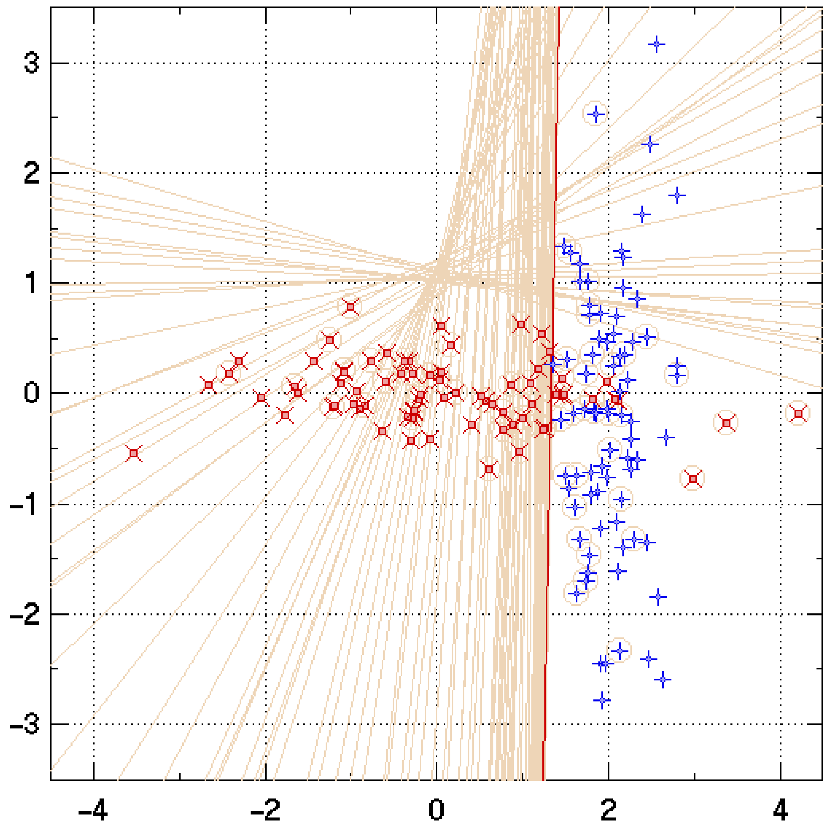
null margin

gauss2Dtr.detq -a 0.01 -b 0 -i 2000 -t 10 -e 1.1 -d 10 -s 17396



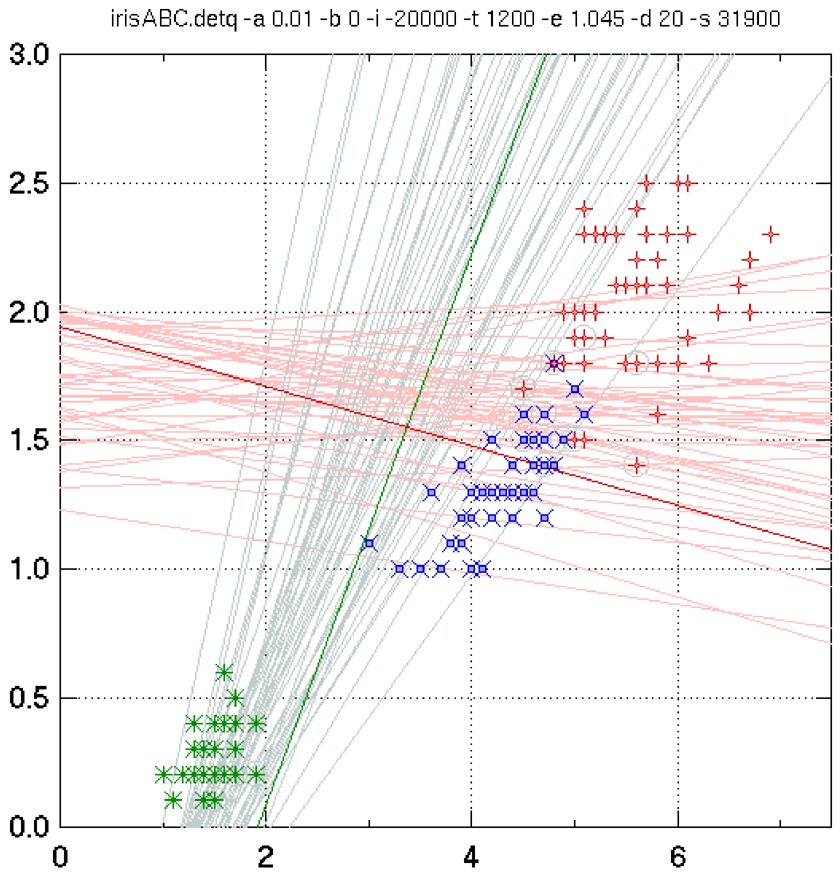
positive margin

gauss2Dtr.detq -a 0.01 -b 5 -i 2000 -t 10 -e 1.1 -d 10 -s 3

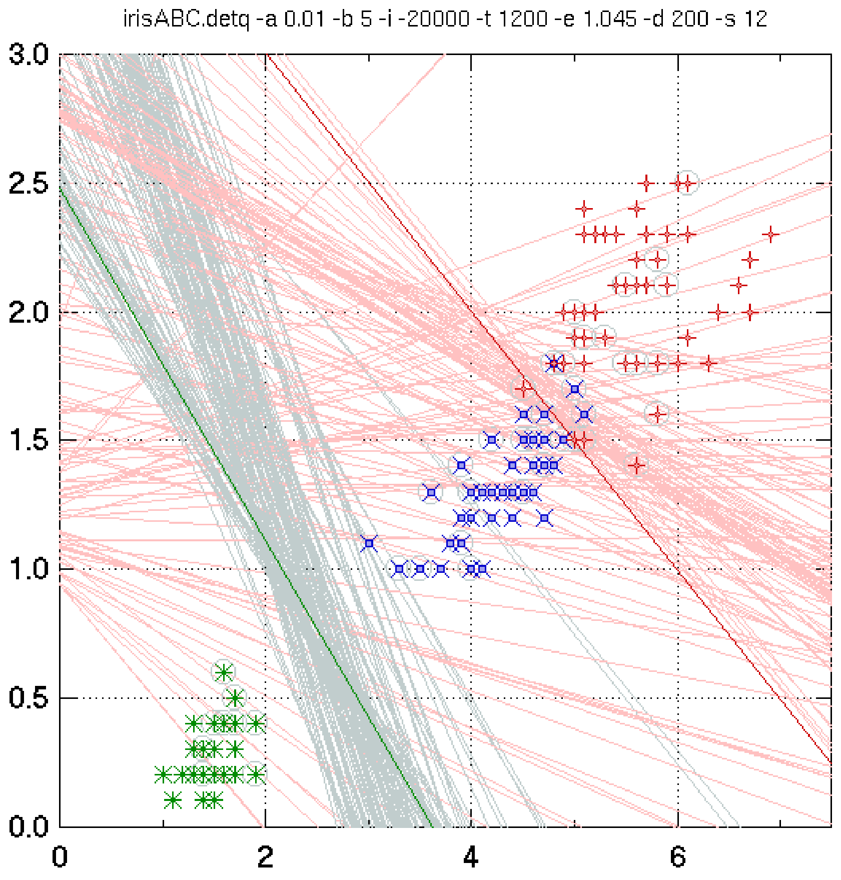


Perceptron algorithm: Iris2D (non separable)

null margin



positive margin



Syllabus

Empirical estimation of the decision error

Let p be the *true* probability of error of a decision system. An empirical estimate ($\hat{p}$) of p can be obtained by counting the number of decision errors, N_e, that occur in a test set of N data:

$$\hat{p} = \frac{N_e}{N}$$

If $N >>$, we can assume that $\hat{p}$ follows a normal distribution: $\hat{p} \sim \mathcal{N}\left(p,\ \frac{P(1-p)}{N}\right)$

$95\,\%$ confidence interval:

$$P(\hat{p} - \epsilon \leq p \leq \hat{p} + \epsilon) \ = \ 0.95; \qquad \epsilon \ = \ 1.96 \sqrt{\frac{\hat{p}(1-\hat{p})}{N}}$$

Example: We observe 50 wrong decisions of a 1000 data test. With a $95\,\%$ of confidence, we can ensure that the true error risk is:

$$p \ = \ 0.05 \pm 1{,}96 \sqrt{\frac{0{,}05 \cdot 0{,}95}{1000}} \ = \ 0{,}05 \pm 0{,}014 \ \ (5\,\% \pm 1{,}4\,\%)$$

Example: In case of 5 errors in 100 data set, the true error risk is:

$$p \ = \ \cdots \ = \ 0{,}05 \pm 0{,}043 \ \ (5\,\% \pm 4{,}3\,\%)$$

Methods for splitting data

An *automated learning* system not only needs data to estimate the error but also data to learn the decision model. Given a set of labeled data, there are different methods to split it into a *training* set and a *test* set.

- ***Re-substitution:*** all available data are used for training as well as for testing. Inconvenient: it is *(very) optimistic*.
- ***Hold out:*** data are split into a training set and a test set. Inconvenient: it fails to use all the available data.
- ***B-fold cross-validation***: data are randomly split into B equal subsets. Each subset is then used as test set for a system trained with the rest of sets. Inconvenient: the number of data in the training set can be small (specifically when the B is small) and the computation time increases in B.
- ***Leaving One Out:*** Each individual data (single observation) is used as test set for a system trained with the remaining $n-1$ observations. It is the extreme case of *B-fold cross-validation* when $B = n$. Inconvenient: high computational cost.

Syllabus

Bibliography

[1] R.O. Duda, D.G. Stork, P.E. Hart. Pattern Classification. Wiley, 2001.