9

Pattern Recognition Based on Straight Line Segments

João Henrique Burckas Ribeiro and Ronaldo Fumio Hashimoto
Universidade de São Paulo - Instituto de Matemática e Estatística
Rua do Matão, 1001, São Paulo, SP - Brasil

1. Introduction

A very common way to represent objects in Pattern Recognition is to extract their features and normalize them into vectors in the space $\mathbb{R}^d$. With the objects being represented by points in the space $\mathbb{R}^d$, the distance among them is an intuitive way to compare their similarities and differences and it is used in the most methods in Pattern Recognition. For instance, one of the first and intuitive method in Pattern Recognition that uses this idea is the k-Nearest Neighbor (k-NN). After that, many other methods have emerged using distances, such as Condensed Nearest Neighbor (CNN) (Hart, 1968), Learning Vector Quantization (LVQ) (Kohonen et al., 1988), Nearest Feature Line (NFL) (Li & Lu, 1999), and a new method called Straight Line Segments (SLS) which takes advantage of some good caracteriscs of LVQ and NFL with low computational complexity (lower than Support Vector Machines) (Ribeiro & Hashimoto, 2006; 2008). This chapter gives a brief discussion about the mentioned methods and details about the SLS method. The next section discusses about the k-NN, CNN, LVQ and NFL methods which are related to the SLS method; Section 3 shows in details the SLS method; Section 4 presents the experimental results using the SLS method; and finally, Section 5 gives the conclusions about the SLS method and presents some future perspectives.

2. Pattern Recognition Based on Distances

This chapter deals only with supervised Pattern Recognition, that is, given a set of labeled objects whose labels were assigned by a tutor. It is desirable that a machine learns how to assign labels to new objects as the tutor does. For simplicity, this work is concentrated on binary classification meaning that there are just two possible values for the labels. For instance, in a system that needs to recognize whether a tumor is malign or benign, a set of examples (that is, a set of labeled objects) could be a set of tomographic images where a doctor has already assigned a label (malign or benign) to each one. The Pattern Recognition system needs to learn how to recognize and classify whether a new input image corresponds to a malign or benign cancer by observing the sample (that is, the set of examples) given by the doctor. To design a system like this, the designer must choose what machine learning is needed to be used in order to have a good classification performance for new objects. A set of machines learning can be represented by a parameterized set of functions $\mathcal{F}_\Lambda = \{f_\alpha : \mathbb{R}^d \mapsto \{0,1\}, \alpha \in \Lambda\}$. Typically, a training algorithm will select the function (or equivalently, the parameter $\alpha \in \Lambda$)

which best classifies new objects (points $\boldsymbol{x} \in \mathbb{R}^d$) by observing the set of examples. Formally a supervised Pattern Recogniton system can be defined in the following way (Vapnik, 1999):

1. let $\mathcal{F}_\Lambda = \{f_\alpha : \mathbb{R}^d \mapsto \{0,1\}, \alpha \in \Lambda\}$ be a set of parameterized functions, where Λ is a set of parameter vectors;
2. let $S_N = \{(\boldsymbol{x}_i, y_i) \in \mathbb{R}^d \times \{0,1\}, i = 1,2,\ldots,N\}$ be a set of N examples $(\boldsymbol{x}_i, y_i)$, such that each vector $\boldsymbol{x}_i \in \mathbb{R}^d$ is drawn independently from a fixed but unknown probability distribution $P(\boldsymbol{x})$; for each input vector $\boldsymbol{x}_i$, a supervisor returns an output value (label) $y_i \in \{0,1\}$ according to a conditional probability distribution $P(y \mid \boldsymbol{x})$, also fixed but unknown;
3. it is desired to find the parameter $\alpha^* \in \Lambda$ such that the function $f_{\alpha^*} \in \mathcal{F}$ minimizes a certain error or risk function in order to classify (that is, to give labels $y \in \{0, 1\}$ to) new query points $\boldsymbol{x} \in \mathbb{R}^d$.

To evaluate any supervised Pattern Recognition system is necessary to define a *loss function* $l_\alpha : \mathbb{R}^d \times \{0, 1\} \mapsto \mathbb{R}$, which represents the penalty given to the machine learning (that is, the function f_α) in case of misclassifying the labeled point $(\boldsymbol{x}, y)$. The most used loss functions are the absolute and square errors described respectively by Eqs. 1 and 2:

$$l_\alpha(\boldsymbol{x}, y) = |f_\alpha(\boldsymbol{x}) - y|, \tag{1}$$

$$l_\alpha(\boldsymbol{x}, y) = (f_\alpha(\boldsymbol{x}) - y)^2. \tag{2}$$

The loss function gives the penalty for just one misclassification. To obtain the overall behavior of the machine learning it is necessary to compute the *functional risk*. In case of binary classification the functional risk can be the probability of the machine learning making a misclassification. Eq. 3 describes the functional risk.

$$R(\alpha) = \int l_\alpha(\boldsymbol{x}, y) \cdot dP(\boldsymbol{x}, y) \tag{3}$$

However, in most Pattern Recognition problems, the probability $P(\boldsymbol{x}, y)$ is usually unknown and the functional risk is estimated by the *empirical risk*. Let $S_N = \{(\boldsymbol{x}_i, y_i) : \boldsymbol{x}_i \in \mathbb{R}^d$ and $y_i \in \{0, 1\}, i \in \{0, 1, \cdots, N\}\}$ be a set of N examples, the empirical risk of the machine learning with parameter $\alpha \in \Lambda$ with respect to S_N is defined by:

$$R_{S_N}(\alpha) = \frac{1}{N} \sum_{i=1}^{N} l_\alpha(\boldsymbol{x}_i, y_i). \tag{4}$$

Since in this chapter we only deal with binary classification, the probability of correct classification can be computed by $1 - R(\alpha)$ and estimated by $1 - R_{S_N}(\alpha)$. After these formal definitions, we give a brief introduction of some supervised Pattern Recognition methods based on distances (or metrics).

2.1 The k-Nearest Neighbor Method

The simplicity and intuitive interpretation are the main qualities of the k-Nearest Neighbor (k-NN) method. According to Devroye et al. (1996), Fix and Hodges presented the k-NN for the first time in 1951. As the name of the method suggests, the k-NN classifies a point $\boldsymbol{x} \in \mathbb{R}^d$ with the class of the majority among the k nearest examples. Usually, k is an odd number to avoid ties in classification. Interesting fact is that Cover & Hart (1967) proved that the asymptotic error rate of k-NN is not greater than twice the Bayes error.

For the most simple implementation of *k*-NN, the computational complexity is $O(N)$ in both memory use and execution time. Devroye et al. cited several works that reduce the computational time complexity, normally by preprocessing the set of examples (Devroye et al., 1996). For instance, the method proposed in (Friedman et al., 1977) reduces the time complexity to $O(log(N))$; in another work (Bandyopadhyay & Maulik, 2002) the sample is sorted according to the distance with respect to some reference point. These methods reduces the time complexity and maintain the same precision of the original *k*-NN, but they still maintain the same memory complexity.
Although the biggest problem of *k*-NN is the overfitting (that is, the empirical risk can be much lower than the functional risk), the *k*-NN method is very often used as a reference for other methods, because it does not depend on a lot of parameters and, in many cases, does not depend on the implementation (Jain et al., 2000).

2.2 The Condensed Nearest Neighbor Method

The *Condensed Nearest Neighbor* (CNN) method (Hart, 1968) aims at keeping in the sample just the examples that are relevant for classification in order to achieve a reduction on the computational complexity for both memory use and execution time. The training algorithm for CNN consists in taking out the examples that are not relevant for the classification. Following this idea, many other authors proposed similar methods to reduce the number of examples without loss accuracy. Wilson & Martinez (2000) wrote a good review of these methods and also proposed six more variants.
Although the CNN method tries to reduce the computational time when it is applied to the test phase, as for *k*-NN, the overfitting issue is still the biggest problem for CNN and derivative methods.

2.3 The Learning Vector Quantization Method

Kohonen et al. (1988) proposed a method based on 1-NN, called *Learning Vector Quantization* (LVQ), in which (i) a predetermined number of processing units, where each processing unit is a reference vector in $\mathbb{R}^d$ that is associated to one class; (ii) an input vector $\boldsymbol{x}$ is classified with the nearest reference vector class. Differently from 1-NN, the reference vectors are not the examples; they are determined by a training algorithm. There are many variants of LVQ, but basically, their training algorithm consists of two main components: (i) a clustering algorithm determines the initial position of the reference vectors, and (ii) an iterative algorithm tries to find the best position for all reference vectors.
After Kohonen's work, many other variations of LVQ have emerged. For instance, Geva & Sitte (1991) proposed a new training algorithm for LVQ called *Decision Surface Mapping*; Hammer & Villmann (2002) presented the *Generalized Relevance Learning Vector Quantization* method which is a training and a feature selection algorithm at the same time. Hammer & Villmann (2002) also compared LVQ to *Support Vector Machines* (SVM). They emphasize two advantages of LVQ over SVM: (i) LVQ is more intuitive and easy to implement; and (ii) the computational complexity od LVQ (on test phase) does not increase with the number of examples as occurs with SVM (Burges, 1998; Hammer et al., 2004).
The computational complexity of LVQ on test phase are directly proportional to the number of reference vectors. Therefore, if N_v is the number of the reference vectors chosen by the user, the computational complexity of LVQ is $O(N_v)$.

2.4 The Nearest Feature Line Method

Li & Lu (1999) presented a method for face recognition called *Nearest Feature Line* (NFL). the main idea behind this method is to define a feature line for each pair of examples of the same class. A new point $\boldsymbol{x} \in \mathbb{R}^d$ is classified as the same class of the nearest feature line. This is a way to virtually increase the information of the sample, interpolating and extrapolating it from a pair of examples. Therefore, each class with N_C examples has $N_C(N_C-1)/2$ feature lines. If $(\boldsymbol{x}_i, y_i)$ and $(\boldsymbol{x}_j, y_j)$ are two examples in the same class, then $y_i = y_j$. The feature line defined by this pair of examples is denoted by $\overleftrightarrow{\boldsymbol{x}_i\boldsymbol{x}_j}$. The distance from a point $\boldsymbol{x}$ to the feature line $\overleftrightarrow{\boldsymbol{x}_i\boldsymbol{x}_j}$ is defined as (Li & Lu, 1999):

$$d(\boldsymbol{x}, \overleftrightarrow{\boldsymbol{x}_i\boldsymbol{x}_j}) = \|\boldsymbol{x} - \boldsymbol{p}\| \tag{5}$$

where

$$\boldsymbol{p} = \boldsymbol{x}_i + \frac{\Big\langle (\boldsymbol{x} - \boldsymbol{x}_i),\, (\boldsymbol{x}_j - \boldsymbol{x}_i) \Big\rangle}{\Big\langle (\boldsymbol{x}_j - \boldsymbol{x}_i),\, (\boldsymbol{x}_j - \boldsymbol{x}_i) \Big\rangle} (\boldsymbol{x}_j - \boldsymbol{x}_i). \tag{6}$$

where $\|\boldsymbol{a}\|$ is the Euclidean norm of $\boldsymbol{a} \in \mathbb{R}^d$ and $\langle \boldsymbol{a}, \boldsymbol{b} \rangle$ is the scalar product between $\boldsymbol{a} \in \mathbb{R}^d$ and $\boldsymbol{b} \in \mathbb{R}^d$.

It was shown that this method has a good classification performance when the number of examples is small and the feature space has a high dimension (Li & Lu, 1999). The disadvantage of this method is its high computational complexity and, depending on the dataset, it could make many misclassifications simply because it exaggerates on interpolation and extrapolation.

Based on NFL, other methods have emerged, for instance, the *Extended Nearest Feature Line* method (Zhou et al., 2004); *Center-based Nearest Neighbor* (Gao & Wang, 2007) and *Rectified Nearest Feature Line* (Du & Chen, 2007). Those methods improved the idea behind the NFL, however they still have a high computational complexity.

3. Pattern Recognition Method Based on Straight Line Segments

Although the Pattern Recognition method based on *Straight Line Segments* (SLS) presented in this section was not designed based on any of the methods described in the previous section, it uses similar ideas. For example, the SLS method uses straight line segments instead of reference vectors of LVQ; and the extremities of the straight line segments do not need to be examples like in NFL and derivatives. Thus, SLS obtains the low computational time complexity of LVQ and uses the interpolation capacity of NFL. So, the SLS method could be considered as an extension of LVQ with some ideas of NFL.

Let $\boldsymbol{p}, \boldsymbol{q} \in \mathbb{R}^{d+1}$. The straight line segment with extremities at $\boldsymbol{p}$ and $\boldsymbol{q}$, denoted $\overline{\boldsymbol{p}\boldsymbol{q}}$, is the closed segment of a line in $\mathbb{R}^{d+1}$ with endpoints $\boldsymbol{p}$ and $\boldsymbol{q}$. More formally,

$$\overline{\boldsymbol{p}\boldsymbol{q}} = \{\boldsymbol{x} \in \mathbb{R}^{d+1} : \boldsymbol{x} = \boldsymbol{p} + \lambda \cdot (\boldsymbol{q} - \boldsymbol{p}),\ 0 \leq \lambda \leq 1\}. \tag{7}$$

The set of all possible straight line segments is denoted by $\overline{PQ} = \{\overline{\boldsymbol{p}\boldsymbol{q}} : \boldsymbol{p}, \boldsymbol{q} \in \mathbb{R}^{d+1}\}$.

Given a point $\boldsymbol{x} \in \mathbb{R}^d$, the extension of $\boldsymbol{x}$ to $\mathbb{R}^{d+1}$, denoted by $\boldsymbol{x}_e \in \mathbb{R}^{d+1}$, is the point $\boldsymbol{x}_e = (\boldsymbol{x}, 0)$, that is, the point $\boldsymbol{x}$ is extended to $\mathbb{R}^{d+1}$ by adding one more coordinate with zero value. Given a point $\boldsymbol{x} \in \mathbb{R}^d$ and a straight line segment $\overline{\boldsymbol{p}\boldsymbol{q}}$, with $\boldsymbol{p}, \boldsymbol{q} \in \mathbb{R}^{d+1}$, the pseudo-distance between $\boldsymbol{x}$ and $\overline{\boldsymbol{p}\boldsymbol{q}}$ is the function $pdist : \mathbb{R}^d \times \overline{PQ} \mapsto \mathbb{R}$ defined as:

$$pdist(\boldsymbol{x}, \overline{\boldsymbol{pq}}) = \frac{dist(\boldsymbol{x}_e, \boldsymbol{p}) + dist(\boldsymbol{x}_e, \boldsymbol{q}) - dist(\boldsymbol{p}, \boldsymbol{q})}{2}, \tag{8}$$

where $dist(\boldsymbol{a}, \boldsymbol{b})$ denotes the Euclidean distance between points $\boldsymbol{a} \in \mathbb{R}^{d+1}$ and $\boldsymbol{b} \in \mathbb{R}^{d+1}$. Note that *pdist* is not the Euclidean distance between a point $\boldsymbol{x} \in \mathbb{R}^d$ and a straight line segment $\overline{\boldsymbol{pq}}$, but it satisfies the following reasonable axioms:

1. If $\boldsymbol{p} = \boldsymbol{q}$, then $pdist(\boldsymbol{x}, \overline{\boldsymbol{pq}}) = dist(\boldsymbol{x}, \boldsymbol{p})$.
2. If $\boldsymbol{x} \in \overline{\boldsymbol{pq}}$, then $pdist(\boldsymbol{x}, \overline{\boldsymbol{pq}}) = 0$.
3. If $\boldsymbol{x} \notin \overline{\boldsymbol{pq}}$, then $pdist(\boldsymbol{x}, \overline{\boldsymbol{pq}}) > 0$.

Let $\mathbb{L}$ be the collection of all possible sets of straight line segments, that is, $\mathbb{L} = \{L : L \subseteq \overline{PQ}\}$. Given two sets the straight line segments, L_0 and L_1, (that is, $L_0, L_1 \in \mathbb{L}$), if $\mathcal{L} = (L_0, L_1) \in \mathbb{L}^2$, then we define the *discriminant function* $T : \mathbb{R}^d \times \mathbb{L}^2 \mapsto \mathbb{R}$ as

$$T(\boldsymbol{x}, \mathcal{L}) = \lim_{\epsilon \to 0} \left(\sum_{\overline{\boldsymbol{pq}} \in L_1} \frac{1}{pdist(\boldsymbol{x}, \overline{\boldsymbol{pq}}) + \epsilon} - \sum_{\overline{\boldsymbol{pq}} \in L_0} \frac{1}{pdist(\boldsymbol{x}, \overline{\boldsymbol{pq}}) + \epsilon} \right). \tag{9}$$

Finally, given a pair of sets of straight line segments $\mathcal{L} = (L_0, L_1) \in \mathbb{L}^2$, we define the *classification function* $y_{\mathcal{L}} : \mathbb{R}^d \mapsto [0, 1]$ as

$$y_{\mathcal{L}}(\boldsymbol{x}) = \frac{1}{1 + e^{-g \cdot T(\boldsymbol{x}, \mathcal{L})}}, \tag{10}$$

where g is a positive real constant. Note that, if $\exists \overline{\boldsymbol{p}_a \boldsymbol{q}_a} \in L_1$ such that $pdist(\boldsymbol{x}, \overline{\boldsymbol{p}_a \boldsymbol{q}_a}) \to 0$ and $pdist(\boldsymbol{x}, \overline{\boldsymbol{p}_b \boldsymbol{q}_b}) \gg 0$ such that $\forall \overline{\boldsymbol{p}_b \boldsymbol{q}_b} \in L_0$, then the first term in Eq. 9 tends to $+\infty$. Consequently $T(\boldsymbol{x}, \mathcal{L}) \to +\infty$ and, therefore, $y_{\mathcal{L}}(\boldsymbol{x}) \to 1$. On the other hand, if $\exists \overline{\boldsymbol{p}_b \boldsymbol{q}_b} \in L_0$ such that $pdist(\boldsymbol{x}, \overline{\boldsymbol{p}_b \boldsymbol{q}_b}) \to 0$ and $pdist(\boldsymbol{x}, \overline{\boldsymbol{p}_a \boldsymbol{q}_a}) \gg 0$ such that $\forall \overline{\boldsymbol{p}_a \boldsymbol{q}_a} \in L_1$, then the second term in Eq. 9 tends to $-\infty$. Consequently $T(\boldsymbol{x}, \mathcal{L}) \to -\infty$ and, therefore, $y_{\mathcal{L}}(\boldsymbol{x}) \to 0$. Furthermore, when both terms in Eq. 9 are equal, $T(\boldsymbol{x}, \mathcal{L}) \to 0$ and $y_{\mathcal{L}}(\boldsymbol{x}) \to 0.5$. Thus, in the case of binary classification, the class of a point $\boldsymbol{x} \in \mathbb{R}^d$ can be obtained by thresholding the function $y_{\mathcal{L}}(\boldsymbol{x})$ at level 0.5 in the following way: the class of point $\boldsymbol{x}$ is 0 if and only if $y_{\mathcal{L}}(\boldsymbol{x}) \leq 0.5$. Therefore, the decision boundary that separates classes 0 from 1 depends on the choice of pair of sets of straight line segments $\mathcal{L} = (L_0, L_1) \in \mathbb{L}^2$. Observe that the SLS method has a difference of all other methods mentioned in previous section: while points $\boldsymbol{x} \in \mathbb{R}^d$, the straight line segments are in $\mathbb{R}^{d+1}$. It gives more flexibility for the decision boundary.

Although this chapter does not deal with regression, in order to show how the classification function $y_{\mathcal{L}}(\boldsymbol{x})$ can be versatile, we illustrate that $y_{\mathcal{L}}(\boldsymbol{x})$ can be used to approximate four different functions $f_i : [0, 1] \mapsto [0, 1]$, $i \in \{1, 2, 3, 4\}$, with only a few straight line segments. The four functions are the following:

$$f_1(x) = \frac{1}{4} + \frac{x}{2}; \; 0 \leq x \leq 1; \tag{11}$$

$$f_2(x) = \begin{cases} 1 & 0 \leq x < 0.25 \\ 0 & 0.25 \leq x < 0,5 \\ 1 & 0.5 \leq x < 0.75 \\ 0 & 0.75 \leq x \leq 1; \end{cases} \tag{12}$$

$$f_3(x) = 0.5 + 0.4 \, sin(2 \cdot \pi \cdot x); \; 0 \leq x \leq 1; \tag{13}$$

$$f_4(x) = \begin{cases} 0.5 + 1.8\,x & 0 \leq x < 0.25 \\ 1.4 - 1.8\,x & 0.25 \leq x < 0.75 \\ -1.3 + 1.8\,x & 0.75 \leq x \leq 1; \end{cases} \tag{14}$$

In Figure 1, we show the plots of the approximation of f_i using the function $y^i_{\mathcal{L}}(x)$. The functions $f_i(x)$, $i \in \{1,2,3,4\}$, are represented by red lines; while the corresponding approximation functions $y^i_{\mathcal{L}}(x)$ are represented by blue lines; the straight line segments in L_0 are represented with circles at the extremities, and the straight line segments in L_1 with $\times$ at the extremities. Note that the four functions are quite different and the approximations are very good (in the domain $[0,1]$). Observe that, for only function $f_4(x)$, it is necessary to use two straight line segments for each class, where one of them is degenerated into a point; in all other functions, just one straight line segment for each class is sufficient to make a good approximation. In Table 1, we provide the obtained empirical risk of these approximations.

Function	absolute error	squared error
$f_1(x)$	0.002011	0.000005
$f_2(x)$	0.000367	0.000011
$f_3(x)$	0.008240	0.000090
$f_4(x)$	0.009094	0.000120

Table 1. Empirical risk of the approximations of the functions f_i.

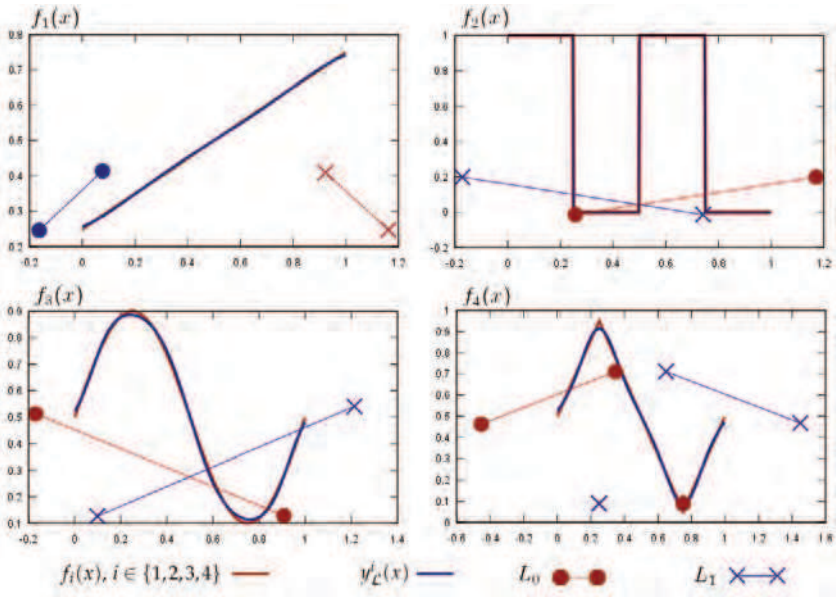


Fig. 1. Plots of the functions $f_i(x)$ and the corresponding approximations $y^i_{\mathcal{L}}(x)$, $i = 1,2,3,4$, with the respectives straight line segments.

3.1 Training Algorithm

The objective of the training algorithm is to find two sets of straight line segments that minimize the functional risk (Eq. 3). However, usually it is impossible to compute the functional risk, since the probability distribution is unknown in many situations. Therefore, the functional risk must be estimated by the empirical risk (Eq. 4). In this way, the training algorithm is based on the empirical risk minimization principle (Vapnik, 1999). More formally, given a sample S_N, the training algorithm must find two sets of straight line segments, L_0 and L_1, that minimize the empirical risk $R_{S_N}(\alpha)$, where the parameter vector α is defined by the positions of the straight line segments in both L_0 and L_1. The training algorithm is divided into two phases:

1. **Placing Algorithm**: In this phase, a heuristic pre-allocates (initializes) the straight line segments.
2. **Tuning Process**: In this phase, the straight line segments are moved from the initial position driven by an optimization algorithm thats tries to minimize the empirical risk.

3.2 Placing Algorithm

The first phase of the training algorithm consists of pre-allocating (that is, finding the initial positions of) the straight line segments which will be tuned later. The **Placing Algorithm** (see below) is based on the fact that the points in the region of the feature space that are near to the straight line segments belonging to L_0 (respectively, L_1) make the function $y_{\mathcal{L}}(\boldsymbol{x})$ tend to 0 (respectively, 1). Thus the first step of the Placing algorithm is to split the sample S_N into two groups of point sets X_0 and X_1 (see Lines 4 and 5 of the Placing algorithm). Group $X_0 \subseteq \mathbb{R}^d$ will initialize the straight line segments of L_0; while group $X_1 \subseteq \mathbb{R}^d$ will initialize the straight line segments of L_1. At Line 9, the k-means clustering algorithm(Duda et al., 2001; Jain et al., 1999) is applied to each group X_j ($j = 0, 1$) in order to find k clusters $\boldsymbol{C}_0, \boldsymbol{C}_1, \ldots, \boldsymbol{C}_{k-1} \subseteq X_j$, with their corresponding centers $\boldsymbol{c}_0, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_{k-1} \in \mathbb{R}^d$. After that, the k-means clustering is applied again to each cluster in $\boldsymbol{C}_i$ ($i = 0, 1, \ldots, k-1$) using $k = 2$ (Line 11 of the Placing algorithm). The two centers $\boldsymbol{d}_0$ and $\boldsymbol{d}_1$ returned by 2-means determines the extremities of a straight line segment for each cluster $\boldsymbol{C}_i$. Since we have k clusters $\boldsymbol{C}_i$ for each class, we have in total k straight line segments for each class. After that, each extremity of the straight line segments is extended in one extra dimension and the value of the corresponding extra coordinate is initially set to 1 for all extremities (see Lines 12 to 14). This value will be adjusted in the tuning process. Although any other clustering algorithm could be used in the Placing algorithm, the k-means was chosen as clustering algorithm because it is simple to implement and has fast convergence (Duda et al., 2001; Jain et al., 1999).

1: PLACING($S_N, nSLS$)
2: **Input**: A sample $S_N = \{(\boldsymbol{x}_i, y_i) : \boldsymbol{x}_i \in \mathbb{R}^d, y_i \in \{0,1\}, i = 1, 2, \ldots, N\}$ and $nSLS$ (number of straight line segments for each set L_0 and L_1).
3: **Output**: Two sets of straight line segments L_0 and L_1 (with $nSLS$ elements in each one).
4: $X_0 \leftarrow \{\boldsymbol{x}_i \in \mathbb{R}^d : (\boldsymbol{x}_i, y_i) \in S_N \text{ and } y_i = 0\}$;
5: $X_1 \leftarrow \{\boldsymbol{x}_i \in \mathbb{R}^d : (\boldsymbol{x}_i, y_i) \in S \text{ and } y_i = 1\}$;
6: **for** $j \leftarrow 0$ to 1 **do**
7: $\quad L_j \leftarrow \emptyset$;
8: $\quad k \leftarrow nSLS$;
9: $\quad [(\boldsymbol{c}_0, \boldsymbol{C}_0), \ldots, (\boldsymbol{c}_{k-1}, \boldsymbol{C}_{k-1})] \leftarrow k\text{-means}(X_j)$;
10: $\quad$ **for** $i \leftarrow 0$ to $k-1$ **do**

11: $[(\boldsymbol{d}_0, \boldsymbol{D}_0), (\boldsymbol{d}_1, \boldsymbol{D}_1)] \leftarrow$ 2-means($\boldsymbol{C}_i$);
12: $\ell \leftarrow 1$;
13: $\boldsymbol{p} \leftarrow (\boldsymbol{d}_0, \ell)$;
14: $\boldsymbol{q} \leftarrow (\boldsymbol{d}_1, \ell)$;
15: $L_j \leftarrow L_j \cup \overline{\boldsymbol{pq}}$;
16: **end for**
17: **end for**
18: **return** L_0, L_1;

The computational complexity of k-means is $O(N \cdot d \cdot k \cdot i)$, where i is the number of interactions, typically less than N. As k-means is applied twice to the sample, the computational complexity for placing is $O(N \cdot d \cdot k \cdot i)$. Considering $i \ll N$, with $k = nSLS$ at Line 9 and $k = 2$ at Line 11, it is possible to conclude that the computational complexity of Placing Algorithm is $O(N \cdot d \cdot nSLS)$ (Lingras & Yao, 2002).

3.3 Tuning the Straight Line Segments

The natural way to find the positions of the straight line segments that minimize the empirical risk (Eq. 4) is to find a point where its derivative is equal to zero. Since this calculation is not analytically feasible, we opted to use the gradient descent method. Since the gradient descent does not guarantee the global minimum, the final positions of the straight line segments will depend on the their initial position computed previouly by the Placing algorithm. In this way, the Placing algorithm is very important, because it can make the second part of the training algorithm to converge to a "good" local solution.

3.4 Gradient Descent

The optimization gradient descent method consists of moving in small steps the parameters (the vector α) in same direction of the gradient of the objective function in order to minimize (or maximize) its value (Michie et al., 1973).

Each displacement of vector α is proportional to γ which is a real positive number. If γ is too big, the displacement can be exaggerated and the empirical risk can increase instead of decrease. On the order hand, if γ was too small, the minimization process of the empirical risk can be too slow. It is possible to implement the gradient descent using a constant value for γ, but in this case, it is necessary to chose small values for γ to avoid exaggerate moves. Because of that, we use a dynamic γ. For each iteration, if the empirical risk decreases, γ increases proportionally to $1 + \gamma_{inc}$ (Line 11 of the GradDesc algorithm). On the other hand, if the iteration increases the empirical risk, the value of γ decreases proportionally to γ_{dec} (Line 15 of the GradDesc algorithm). Although the value of γ is updated dynamically during the training process, it must not be exaggerated in order to avoid the instability of the initial steps of tuning process. Empirically, we observed that γ stabilizes around 0.1, so this value is adopted for the initial γ. To compute the γ value dynamically it is interesting not to change the value of γ too much when the empirical risk is decreasing, and it is also interesting to decrease the value of γ quickly when the iteration increases the empirical risk. Therefore we set $\gamma_{inc} = 0.1$ and $\gamma_{dec} = 0.5$.

1: GRADDESC (α, I_{max}, I_{min}, γ_{ini}, γ_{inc}, γ_{dec}, $Disp_{min}$, R_{min});
2: **Input**: The vector α that must be optimized; the maximum number I_{max} of iterations; the minimum number I_{min} of iterations; the initial value γ_{ini} for γ; the increasing rate γ_{inc} for γ; the decreasing rate γ_{dec} for γ; the minimum displacement $Disp_{min}$; and finally the minimum empirical risk R_{min}.

3: **Output**: the vector α with optimized values.
4: $\alpha_{best} \leftarrow \alpha$; $\gamma \leftarrow \gamma_{init}$; $R0 \leftarrow R_{S_N}(\alpha)$; $I \leftarrow 0$;
5: **repeat**
6: $dR \leftarrow \bigtriangledown_\alpha R_{S_N}(\alpha)$;
7: $\alpha \leftarrow \alpha - \gamma \cdot dR$;
8: $Disp \leftarrow |\gamma \cdot dR| / \sqrt{dim(\alpha)}$;
9: $R1 \leftarrow R_{S_N}(\alpha)$;
10: **if** $R0 \geq R1$ **then**
11: $\gamma \leftarrow \gamma \cdot (1 + \gamma_{inc})$;
12: $\alpha_{best} \leftarrow \alpha$;
13: $R0 \leftarrow R1$;
14: **else**
15: $\gamma \leftarrow \gamma \cdot \gamma_{dec}$;
16: $\alpha \leftarrow \alpha_{best}$;
17: **end if**
18: $I \leftarrow I + 1$;
19: **until** $((I > I_{min})$ and $((I > I_{max})$ or $(Disp < Disp_{min})$ or $(R1 < R_{min})))$;
20: **return** α_{best};

Four conditions are used to stop the gradient descent process. First, the condition $I > I_{min}$ guaranties a minimum number I_{min} of iterations. Secondly, the number of iterations can not exceed I_{max} to guarantee that the algorithm will stop; thirdly, if the displacement of the parameters is smaller than $Disp_{min}$, it may mean that the parameter vector has achieved a local minimum; and the fourth criterion is useful when it is known what is the minimal empirical risk in order to avoid overfitting.
Note that the value of displacement $Disp$ is computed using $|\gamma \cdot dR|$ and is divided by the square root of the dimension of vector α (see Line 8). This division normalizes the displacement with respect to the number of parameters. Thus, the value for $Disp_{min}$ is normalized for all optimization problem. Empirically we observed that with $Disp_{min} = 0.001$ the empirical risk is stabilized.

3.5 The Proposed Training Algorithm

Before tuning of the extremities of the initial straight line segments (that is, adjusting their positions), it is necessary to determine the best initial value ℓ for the last coordinate of all extremities of the straight line segments (the value of the variable ℓ which was initialized with 1 in the Placing algorithm) and for the constant g (see Eq. 10).
In previous versions of the training algorithm, the initial value of ℓ was determined by the standard deviation of the examples in the sample (Ribeiro & Hashimoto, 2006; 2008). This heuristic aims at making the initial displacement of the last coordinate proportional to the examples dispersion. Also, in a previous version, the value of g was also chosen by a heuristic that makes the values of $y_{\mathcal{L}}(\boldsymbol{x})$ have a uniform distribution (Ribeiro & Hashimoto, 2008). This way, low and high values of g is prevented, which make $y_{\mathcal{L}}(\boldsymbol{x})$ tend to 0.5 and to 0 or 1, respectively. Although those heuristics work well, they do not consider the empirical risk. So, here, we have applied the gradient descent to find better values for both ℓ and g.
In our case we desire to minimize the empirical risk. The training algorithm applies the gradient descent twice: first to find the best values for ℓ and g; and, secondly to find the best position for all extremities of all straight line segments (of course, including all their last coordinates).

1: TRAINING (S_N, $nSLS$, I_{max}, I_{min}, γ_{ini}, γ_{inc}, γ_{dec}, $Disp_{min}$, R_{min});
2: **Input**: A sample $S_N = \{(\boldsymbol{x}_i, y_i) : \boldsymbol{x}_i \in \mathbb{R}^d; y_i \in \{0,1\}, i = 1,2,\ldots,N\}$ and $nSLS$ (number of straight line segments for each set L_0 and L_1). The other inputs are: the maximum number I_{max} of iterations; the minimum number I_{min} of iterations; the initial value γ_{ini} for γ; the increasing rate γ_{inc} for γ; the decreasing rate γ_{dec} for γ; the minimum displacement $Disp_{min}$; and finally the minimum empirical risk R_{min}.
3: **Output**: two sets of straight line segments L_0 and L_1.
4: $[L_0, L_1] \leftarrow$ Placing$(S_N, nSLS)$;
5: $g \leftarrow 1$;
6: $\alpha_1 \leftarrow$ Vectorize(g, ℓ);
7: $(g, \ell) \leftarrow$ GradDesc (α_1, I_{max}, I_{min}, γ_{ini}, γ_{inc}, γ_{dec}, $Disp_{min}$, R_{min});
8: $\alpha_2 \leftarrow$ Vectorize(L_0, L_1);
9: $(L_0, L_1) \leftarrow$ GradDesc (α_2, I_{max}, I_{min}, γ_{ini}, γ_{inc}, γ_{dec}, $Disp_{min}$, R_{min});
10: **return** L_0, L_1 and g;

The parameters g and ℓ define the vector α_1, where g is a positive real number that defines how "smooth" the sigmoid $y_{\mathcal{L}}(\boldsymbol{x})$ will be with respect to $T(\boldsymbol{x}, \mathcal{L})$; and ℓ is the value for the last coordinate of all straight line segments that is initially equal to 1 for all segments.
The function Vectorize(g, ℓ) returns the following parameter vector α_1:

$$\alpha_1 = \begin{bmatrix} g \\ \ell \end{bmatrix}. \tag{15}$$

The function Vectorize(L_0, L_1) returns the parameter vector α_2 for the second gradient descent whose coordinates are the extremities of each straight line segment (including all their last coordinates):

$$\alpha_2 = \begin{bmatrix} p_h^{j,k} \\ \vdots \\ q_h^{j,k} \\ \vdots \end{bmatrix}, \ j \in \{0,1\}, \ k \in \{1,\ldots,|L|\}, \ h \in \{1,\ldots,d+1\}, \tag{16}$$

where $L = |L_0|$ (assuming that $|L_0| = |L_1|$) and $p_h^{j,k}$ is the value of the h-th coordinate of the extremity $\boldsymbol{p}$ of the k-th straight line segment $\overline{\boldsymbol{p}_k\boldsymbol{q}_k}$ of L_j. The other extremity $\boldsymbol{q}$ of $\overline{\boldsymbol{p}_k\boldsymbol{q}_k} \in L_j$ is represented in analogous way. The function vectorize(L_0, L_1) returns the vector α_2 as described in Eq. 16. We should remark that after the application of the second gradient descent, the last coordinates of all straight line segments are not equal anymore.
We use the square error (Eq. 2) as the loss function. Thus, the empirical risk is defined by Eqs. 17 and 18. To simplify the notation, we will omit the parameters of some functions such as: $T_i = T(\boldsymbol{x}_i, \mathcal{L})$, $y_{\mathcal{L}} = y_{\mathcal{L}}(\boldsymbol{x}_i)$ and $pdist_{i,k}^j = pdist(\boldsymbol{x}_i, L_k^j)$. In the following, we present the calculations for the gradient of $R_{S_N}(\alpha)$, where $\alpha = \alpha_2$, that is, $\bigtriangledown_{\alpha_2} R_{S_N}$. Let

$$err_i(y_{\mathcal{L}}) = y_{\mathcal{L}}(\boldsymbol{x}_i) - y_i \ \text{ and} \tag{17}$$

$$R_{S_N}(\alpha) = \frac{1}{N}\sum_{i=1}^{N}[err_i(y_{\mathcal{L}})]^2. \tag{18}$$

The gradient $\bigtriangledown_{\alpha_2} R_{S_N}$ is defined for $j = \{0,1\}$, $k = \{1,\ldots,|L|\}$ and $h = \{1,\ldots,d+1\}$ as:

$$\bigtriangledown_{\alpha_2} R_{S_N}(\alpha_2) = \begin{bmatrix} \frac{\partial R_{S_N}}{\partial p_h^{j,k}} \\ \vdots \\ \frac{\partial R_{S_N}}{\partial q_h^{j,k}} \\ \vdots \end{bmatrix}, \tag{19}$$

where

$$\frac{\partial R_{S_N}}{\partial p_h^{j,k}} = \frac{1}{N} \sum_{i=1}^{n} 2 \cdot err_i \frac{g}{e^{g \cdot T_i} + e^{-g \cdot T_i} + 2} \frac{(-1)^j}{(pdist_{i,k}^j + \epsilon)^2} \cdot \tag{20}$$

$$\left(\frac{p_h^{j,k} - x_{i,h}}{2 \cdot dist(\boldsymbol{x}_i, \boldsymbol{p}^{j,k})} - \frac{p_h^{j,k} - q_h^{j,k}}{2 \cdot dist(\boldsymbol{p}^{j,k}, \boldsymbol{q}^{j,k})} \right). \tag{21}$$

The calculations of the parameters for the first gradient are g and ℓ. The value for the last coordinate of all straight line segments is set $p_{d+1}^{j,k} = q_{d+1}^{j,k} = \ell$ (that is, $p_{d+1}^{j,k}$ and $q_{d+1}^{j,k}$ are replaced with ℓ in the empirical risk R_{S_N} making it as a function of g and ℓ). Then, we have:

$$\bigtriangledown_{\alpha_1} R_{S_N}(\alpha_1) = \begin{bmatrix} \frac{\partial R_{S_N}}{\partial g} \\ \frac{\partial R_{S_N}}{\partial \ell} \end{bmatrix}, \tag{22}$$

$$\frac{\partial R_{S_N}}{\partial g} = \sum_{i=1}^{N} \frac{2\, err_i\, T_i\, e^{g\, T_i}}{e^{2\, g\, T_i} + 2\, e^{g\, T_i} + 1} \text{ and} \tag{23}$$

$$\frac{\partial R_{S_N}}{\partial \ell} = \sum_{j=0}^{1} \sum_{k=1}^{|L_j|} \left(\frac{\partial R_{S_N}}{\partial p_{d+1}^{j,k}} + \frac{\partial R_{S_N}}{\partial q_{d+1}^{j,k}} \right). \tag{24}$$

By the beginning of the first gradient, the last coordinates $p_{d+1}^{j,k} = q_{d+1}^{j,k} = 1$. Between the first gradient and the beginning of the second gradient, the last coordinates $p_{d+1}^{j,k} = q_{d+1}^{j,k} = \ell$ for all straight line segments. Finally, after the second gradient each last coordinate can have different values among them.

The time complexity for the *Training* algorithm is the sum of the complexity of *Placing* Algorithm plus twice the complexity of *GradDesc* Algorithm, that is $O(N \cdot d \cdot nSLS) + 2O(N \cdot d \cdot nSLS \cdot I_{max})$, which is equal to $O(N \cdot d \cdot nSLS \cdot I_{max})$.

To avoid overfitting and unnecessary increasing of computational time complexity, the best choice is to use a minimum number of straight line segments (for each class) for which the SLS method still has a good classification performance.

4. Experimental Results

In this section we describe the experiments we performed in order to evaluate the SLS method and compare it with other methods. For that, the experiments were divided into two parts: the first one uses artificial data to evaluate the behavior of the SLS method; while the second one uses public datasets to analyse the SLS method with real applications and compare our results with the results obtained by other methods.

4.1 Artificial Data

The artificial data for each class $C \in \{0,1\}$ were generated using probability distributions associated to the density function defined by the sum of M two dimensional normal density functions as shown in Eq. 25:

$$p(\boldsymbol{x}, y = C) = \sum_{i=1}^{M} P_i^C \cdot Normal(\boldsymbol{x}, \mu_i^C, \Sigma_i^C), \tag{25}$$

where $\mu_i^C \in \mathbb{R}^2$ is the center of the normal density function, Σ_i^C is the 2×2 covariance matrix and P_i^C is a real number such that $\sum_{i=1}^{M} P_i^C = 1$. In general, each covariance matrix Σ^C is positive semidefinite matrix and has the following form:

$$\begin{pmatrix} (\sigma_1)^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & (\sigma_2)^2 \end{pmatrix}, \tag{26}$$

where ρ is the correlation between the two random variables x_1 and x_2, and σ_j, for $j = 1, 2$, is the standard deviation of x_j. We designed four probability density functions called *Simple*, *distX*, *distS* and *DistF*. Each probability density function (pdf) is defined as follows:

***Simple* pdf:**

$$\begin{aligned} &P_1^0 = 1.0 && P_1^1 = 1.0 \\ &\mu_1^0 = (0.6, 0.6) && \mu_1^1 = (0.4, 0.4) \\ &\Sigma_1^0 = \begin{pmatrix} 0.010 & -0.009 \\ -0.009 & 0.010 \end{pmatrix} && \Sigma_1^1 = \begin{pmatrix} 0.010 & 0.009 \\ 0.009 & 0.010 \end{pmatrix} \end{aligned} \tag{27}$$

***DistX* pdf:**

$$\begin{aligned} &P_1^0 = P_2^0 = 0.5 \qquad P_1^1 = P_2^1 = 0.5 \\ &\mu_1^0 = (0.25, 0.25) \quad \mu_1^1 = (0.25, 0.75) \\ &\mu_2^0 = (0.75, 0.75) \quad \mu_2^1 = (0.75, 0.25) \\ &\Sigma_i^C = \begin{pmatrix} 0.04 & 0.00 \\ 0.00 & 0.04 \end{pmatrix}, \ C \in \{0, 1\} \text{ and } i \in \{1, 2\} \end{aligned} \tag{28}$$

***DistS* pdf:**

$$\begin{aligned} &P_1^0 = P_2^0 = 0.5 \quad P_1^1 = P_2^1 = 0.5 \\ &\mu_1^0 = (0.4, 0.4) \quad \mu_1^1 = (0.6, 0.2) \\ &\mu_2^0 = (0.4, 0.8) \quad \mu_2^1 = (0.6, 0.6) \\ &\Sigma_i^C = \begin{pmatrix} 0.02 & 0.00 \\ 0.00 & 0.01 \end{pmatrix}, \ C \in \{0,1\} \text{ and } i \in \{1, 2\} \end{aligned} \tag{29}$$

***DistF* pdf:**

$$
\begin{aligned}
&P_1^0 = 0.574 && P_1^1 = 0.574\\
&P_2^0 = P_3^0 = 0.213 && P_2^1 = P_3^1 = 0.213\\
&\mu_1^0 = (0.125,\, 0.5) && \mu_1^1 = (0.875,\, 0.5)\\
&\mu_2^0 = (0.5,\, 0.375) && \mu_2^1 = (0.5,\, 0.125)\\
&\mu_3^0 = (0.5,\, 0.875) && \mu_3^1 = (0.5,\, 0.625)\\
&\Sigma_1^C = \begin{pmatrix} 0.010 & 0.000 \\ 0.000 & 0.040 \end{pmatrix} && \Sigma_2^C = \begin{pmatrix} 0.012 & 0.000 \\ 0.000 & 0.010 \end{pmatrix}\\
&\Sigma_3^C = \begin{pmatrix} 0.012 & 0.000 \\ 0.000 & 0.010 \end{pmatrix}, \; C \in \{0,\, 1\}.
\end{aligned}
\tag{30}
$$

Figure 2 shows four graphics, each one representing one of the probability density functions. In Figure 3, there are four samples with 800 examples, each one drawn from one of the probability distributions associated with the 4 probability density functions.

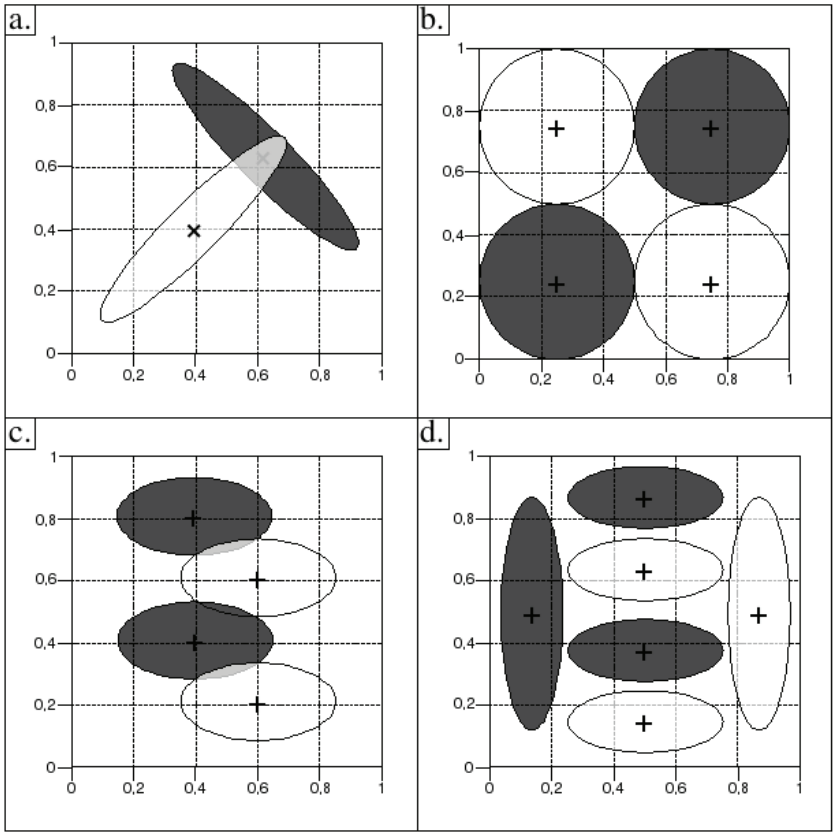


Fig. 2. Representation of the probability density functions: a) *Simple*, b) *DistX*, c) *DistS* and d) *DistF*.

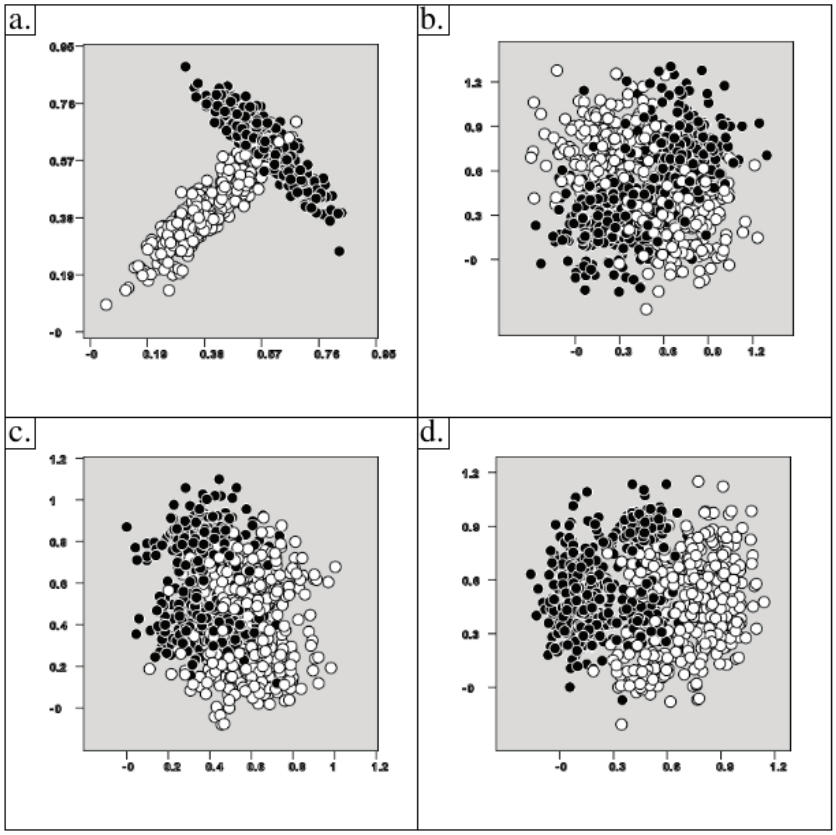


Fig. 3. Samples drawn from the probability distributions with 800 examples: a) *Simple*, b) *DistX*, c) *DistS* and d) *DistF*.

Since the probability density function is known, it is possible to compute the functional risk $R(\alpha)$ (see Eq. 3) of the SLS and Bayes classifiers (Duda et al., 2001). Consequently, it is possible to compute the actual probability of correct classification, that is $1 - R(\alpha)$, for both classifiers. For that, we used the numerical integration considering the values for $\boldsymbol{x} \in \mathbb{R}^2$ inside the rectangle defined by the corners $(-0.5, -0.5)$ and $(1.5, 1.5)$. Inside this rectangle, there are more than 99% of examples for all probability density functions. The numerical integration was done using 160 000 points. The probability of correct classification for Bayes classifier for each probability density function is shown in Table 2.

Probability Density Function	Bayes (%)
Simple	96.95
DistX	81.11
DistS	84.39
DistF	91.43

Table 2. Probability of Correct Classification using Bayes Classifier.

For each probability density function (*Simple, DistX, DistS and DistF*), $4 \times 3 = 12$ samples (3 for each quantity of examples) with 100, 200, 400 and 800 examples were generated. The SLS training algorithm using 1, 2, 3, and 4 straight line segments for each class was applied to each sample, totalling 192 applications of the training algorithm. The parameters used for training algorithm are shown in Table 3.

Parameter	Value
I_{max}	10 000
min	20
γ_{init}	0.1
γ_{inc}	0.1
γ_{dec}	0.5
$Desl_{min}$	0.001
R_{min}	0.0001

Table 3. Training parameters for the SLS Training Algorithm.

The obtained results for the probability density functions *Simple*, *DistX*, *DistS* and *DistF* are respectively shown in Tables 4, 5, 6 and 7. The rows of these tables present the results using the same number of examples; while the columns show the results with the same number of straight line segments for each class. On left part of each column, it is presented the best result among the three performed tests with the same number of examples and de same number of straight line segments for each class. On right part of each column, it is shown the average of probability of correct classification for the three tests. The results shown in bold have the difference from the respective Bayes correct classification less than 1%.
In Figure 4, there are four response maps, one for each probability density function. In these maps, the gray scale is proportional to $y_{\mathcal{L}}(\boldsymbol{x}) \in [0,1]$ value, where black represents 0 and white represents 1. In these maps, the straight line segments are projected to a bidimensional plane. For these projections of the straight line segments, white represents class 0 and black class 1.
Since the probability density functions are known, we can compute and compare the probability of correct classification of SLS and Bayes classifiers. The results show that the SLS method achieved good performace. For all cases, we obtained a difference that is less than 1% comparing to Bayes classifier.
For distributions *DistS* and *DistF* (that have a more complex decision boundary), it was necessary to add more straight line segments for each class to obtain a good performace. For distributions *Simple* and *DistX*, just one straight line segment for each class was enough for a good performace. As expected, the more training examples, the better performace of the classifier.

4.2 Experiments with Public Datasets

To compare the performance of SLS with other methods, we did experiments using 8 public datasets. These datasets were chosen from (Van Gestel et al., 2004) in which a benchmarking for SVM and other methods can be found. In this way, it is possible to compare our results with theirs (Van Gestel et al., 2004). The datasets taken from (Van Gestel et al., 2004) correspond only to binary classification. All datasets are available in UCI Machine Learning Repository (Asuncion & Newman, 2007) and they are *Australian Credit Approval* (**australian**), *Breast Cancer Wisconsin* (**breast-cancer**), *Pima Indians Diabetes* (**diabetes**), *German Credit Data* (**german**), *Heart* (**heart**), *Ionosphere* (**ionosphere**), *Liver Disorders* (**liver-disorders**) and *Sonar,*

Simple (Bayes = 96.951 %)								
# of SLS/class	1		2		3		4	
	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.
100	**96.40**	95.93	**96.69**	**96.07**	**96.41**	**96.17**	**96.32**	**95.97**
200	**96.43**	**96.37**	**96.51**	**96.47**	**96.45**	**96.40**	**96.44**	**96.39**
400	**96.64**	**96.60**	**96.65**	**96.54**	**96.63**	**96.55**	**96.56**	**96.47**
800	**96.80**	**96.52**	**96.55**	**96.48**	**96.79**	**96.57**	**96.58**	**96.48**

Table 4. Results for **Simple** pdf.

DistX (Bayes = 81.105 %)								
# of SLS/class	1		2		3		4	
	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.
100	78.86	78.45	78.37	77.68	76.46	76.36	78.39	77.65
200	**80.49**	**80.33**	79.98	79.75	79.61	79.29	79.70	79.32
400	**80.90**	**80.63**	**80.76**	**80.41**	**80.73**	**80.45**	**80.74**	79.86
800	**81.03**	**80.86**	**80.98**	**80.77**	**81.06**	**80.84**	**80.86**	**80.75**

Table 5. Results for **DistX** pdf.

DistS (Bayes = 84.386 %)								
# of SLS/class	1		2		3		4	
	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.
100	82.85	80.67	82.94	81.12	**83.76**	82.23	**83.58**	82.21
200	81.88	80.80	**83.54**	82.92	**83.77**	83.11	**83.66**	81.94
400	81.17	80.29	**83.94**	**83.64**	**84.03**	**83.78**	**83.99**	81.50
800	80.83	80.61	**83.90**	**83.78**	**83.98**	**83.80**	**83.98**	**83.88**

Table 6. Results for **DistS** pdf.

DistF (Bayes = 91.435 %)								
# of SLS/class	1		2		3		4	
	Max.	Avg.	Max.	Avg.	Max.	Avg.	Max.	Avg.
100	85.25	84.72	89.12	88.94	**90.46**	88.68	89.84	87.91
200	83.56	83.46	90.27	89.64	89.79	89.72	89.86	89.57
400	85.60	84.61	**90.47**	90.25	**90.45**	90.22	**90.84**	**90.63**
800	83.61	83.53	90.12	89.80	**90.53**	90.05	**90.68**	90.41

Table 7. Results for **DistF** pdf.

Mines vs. Rocks (**sonar**). The number of attributes in each dataset is shown in Table 8. For more detailed information, we recommend to visit the UCI Machine Learning Repository (Asuncion & Newman, 2007).

We performed the experiment using the same methodology described in (Van Gestel et al., 2004), that is, for each training, the sample is randomly split into two sets: one for training containing 2/3 of examples; and other for the test phase with 1/3 of examples. This splitting process is repeated ten times for each dataset. Besides, the training parameters used for training the SLS method were the same values used for artificial data (see Table 3).

Table 8 shows the average and the standard deviation (presented between parenthesis) of correct classification. The results for the SLS method using 1, 2, 3, 4, 6, 8, and 10 straight line segments for each class, respectively, are shown at Rows from 1 to 7. The results obtained by Van Gestel et al. (2004) using SVM and k-NN are presented at Rows from 8 to 17. Finally,

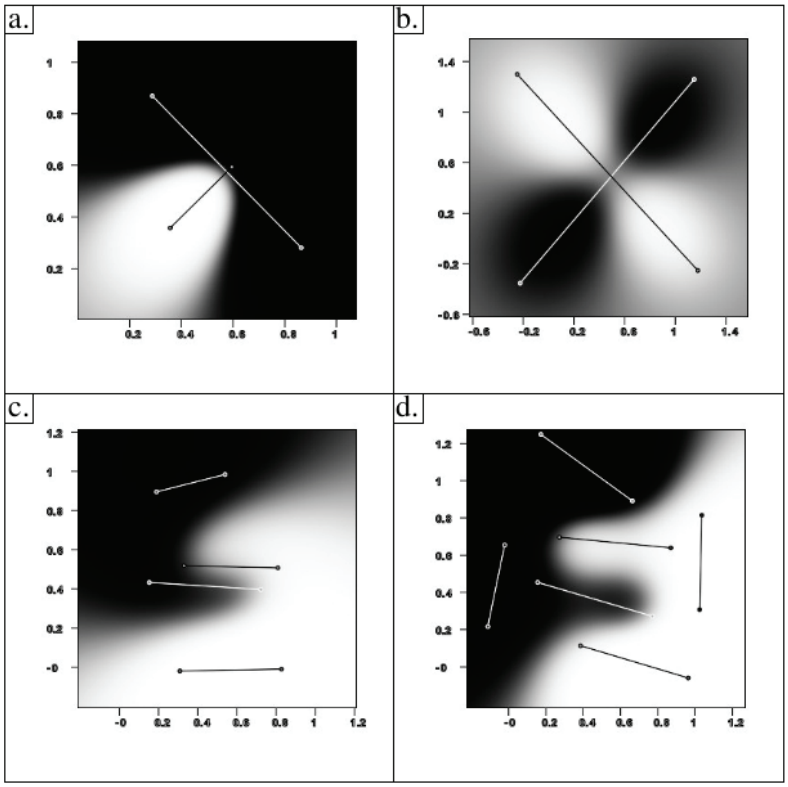


Fig. 4. Response maps for each probability density function:
a) *Simple*, sample with 800 examples and using 1 straight line segment for each class;
b) *DistX*, sample with 800 examples and using 1 straight line segment for each class;
c) *DistS*, sample with 800 examples and using 2 straight line segments for each class;
d) *DistF*, sample with 400 examples and using 3 straight line segments for each class.

the results using SVM with RBF and polynomial kernels (Eqs. 31 and 32 - in these equations, δ and ϱ are kernel parameters and C controls how the errors are penalized) are shown at Rows 18 and 22. In particular, the last two results were obtained using the LibSVM software developed by Chang & Lin (2001) with parameters δ, C and ϱ (shown at Rows 20, 21, 24, 25 and 26) chosen by an exhaustive search for the best combination using the values presented in Table 9.

		australian	breast-cancer	diabetes	german	heart	ionosphere	liver
	N_{test}	690	683	768	1000	270	351	
	d	14	9	8	20	13	33	
1	SLS 1	86.7 (1.5)	97.7 (0.6)	**76.4** (1.8)	76.4 (1.0)	81.7 (4.6)	94.1 (1.7)	5
2	SLS 2	86.3 (1.9)	97.8 (0.7)	74.8 (2.5)	**76.7** (2.2)	80.6 (3.9)	94.6 (1.4)	6
3	SLS 3	86.5 (1.7)	98.1 (0.8)	74.2 (2.0)	75.9 (2.1)	81.7 (3.6)	94.4 (2.1)	**7**
4	SLS 4	86.3 (1.8)	**98.1** (0.7)	73.7 (1.7)	76.3 (1.4)	81.9 (3.3)	93.4 (3.1)	7
5	SLS 6	86.1 (1.6)	98.0 (0.7)	72.4 (2.9)	76.5 (1.6)	81.0 (2.8)	94.9 (1.3)	6
6	SLS 8	**87.0** (1.8)	97.9 (0.8)	72.4 (2.4)	76.2 (1.6)	81.4 (3.4)	94.8 (1.6)	6
7	SLS 10	86.8 (1.8)	97.8 (0.7)	72.2 (3.2)	76.4 (1.6)	**82.2** (3.3)	**95.2** (2.6)	6
8	$RBFLS-SVM$	87.0 (2.1)	96.4 (1.0)	76.8 (1.7)	76.3 (1.4)	84.7 (4.8)	**96.0** (2.1)	7
9	$RBFLS-SVM_F$	86.4 (1.9)	96.8 (0.7)	72.9 (2.0)	70.8 (2.4)	83.2 (5.0)	93.4 (2.7)	6
10	$LinLS-SVM$	86.8 (2.2)	95.8 (1.0)	76.8 (1.8)	75.4 (2.3)	84.9 (4.5)	87.9 (2.0)	6
11	$LinLS-SVM_F$	86.5 (2.1)	96.9 (0.7)	73.1 (1.7)	68.6 (2.3)	82.8 (4.4)	85.0 (3.5)	6
12	$PolLS-SVM$	86.5 (2.2)	96.4 (0.9)	77.0 (1.8)	76.3 (1.4)	83.7 (3.9)	91.0 (2.5)	7
13	$PolLS-SVM_F$	86.6 (2.2)	96.9 (0.7)	73.0 (1.8)	70.3 (2.3)	82.4 (4.6)	91.7 (2.6)	6
14	$RBFSVM$	86.3 (1.8)	96.4 (1.0)	77.3 (2.2)	75.9 (1.4)	84.7 (4.8)	95.4 (1.7)	7
15	$LinSVM$	86.7 (2.4)	96.3 (1.0)	77.0 (2.4)	75.4 (1.7)	83.2 (4.2)	87.1 (3.4)	6
16	$1-NN$	81.1 (1.9)	95.3 (1.1)	69.6 (2.4)	69.3 (2.6)	74.3 (4.2)	87.2 (2.8)	6
17	$10-NN$	86.4 (1.3)	96.4 (1.2)	73.6 (2.4)	72.6 (1.7)	80.0 (4.3)	85.9 (2.5)	6
18	$libSVMRBF$	**87.4** (1.6)	97.8 (0.5)	**77.8** (1.8)	**77.3** (0.5)	**85.1** (3.3)	95.4 (1.9)	**7**
19	# SV	407.7	46	277.9	371.9	145.8	65.6	
20	γ	0.01	0.01	0.01	0.01	0.1	0.1	
21	C	0.1	10	50	20	0.1	10	
22	$libSVMPol$	87.3 (1.7)	**97.9** (0.9)	77.3 (2.2)	76.4 (1.4)	82.4 (4.0)	92.8 (2.6)	7
23	# SV	158.9	52.8	277.2	386.5	143.2	78.3	
24	δ	2	0.01	5	0.01	0.01	2	
25	C	0.01	200	0.01	50	20	0.01	
26	ϱ	2	2	2	2	2	2	

The results presented in this table are the average of correct classification for 10 tests. The number in parenthesi
For each column there are two values in bold: the first one is the best result using SLS method and second one i
The underlined numbers for the SLS method are statistically equal to the best results among the other methods

Table 8. Average of Correct classification for public datasets.

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{-\delta \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2} \tag{31}$$

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \left(\delta \left\langle \boldsymbol{x}_i, \boldsymbol{x}_j \right\rangle\right)^{\varrho}. \tag{32}$$

C	0.01, 0.1, 10, 20, 50, 100, 200
δ	0.01, 0.1, 0.5, 1.0, 2.0, 5.0, 10.0
ϱ	2, 4, 6, 8, 10

Table 9. Parameters for RFB and Polynomial Kernels (Eqs. 31 and 32).

In order to have a good view of the results, the average of correct classification for each dataset can also be viewed in the plots presented in Figure 5.
By observing the plots in Figure 5, we can see that the good performance of SLS method with respect to the others is well-posted on the graphic of the breast-cancer dataset. More details about the results can be viewed on Table 8. In particular, this table is divided in two blocks: the first one presents the results of the SLS method (Rows from 1 to 7); while the second one shows the results using SVM and k-NN taken from(Van Gestel et al., 2004). The best results in each block are in bold. We also applied the t-test with 95% of confidence (Dietterich, 1998) to the results of the SLS method and the best result of SVM and k-NN. The underlined values in the table indicate that the results obtained by SLS method are statistically equal to the best result among the others. By observing Table 8, it is possible to notice that by t-test the SLS method has always the performance significantly equal to the performance of SVM. In the case of **breast-cancer** dataset, the performance of SLS method was better than SVM.
On one hand, the computational time complexity of SVM on test phase is $\Theta(N_{SV} \cdot K)$, where N_{SV} is the number of support vectors and K is the computational complexity for computing the kernel (Burges, 1998). The time complexity for both polynomial and RBF kernel computations is proportional to distance computations. On the other hand, the computational time complexity of the SLS method on test phase is $\Theta(|L_0| + |L_1|)$. Note that, for each straight line segment, there are two distance computations. Then, with respect to the computational complexity, one straight line segment in SLS is equivalent to two support vectors in SVM.
In the experiments presented in this chapter, for the SLS method, we used no more than 10 straight line segments for each class meaning the maximum of 20 straight line segments. This is equivalent to 40 support vectors in SVM. Note that, at Rows 19 and 23 in Table 8, the lowest average of support vectors is 46 in breast-cancer dataset with RBF kernel. However, in general, the number of support vectors is commonly higher than it. Therefore, comparing the computational complexity of both methods, we can conclude that the SLS method is computationally more efficient than SVM for similar classification performance.

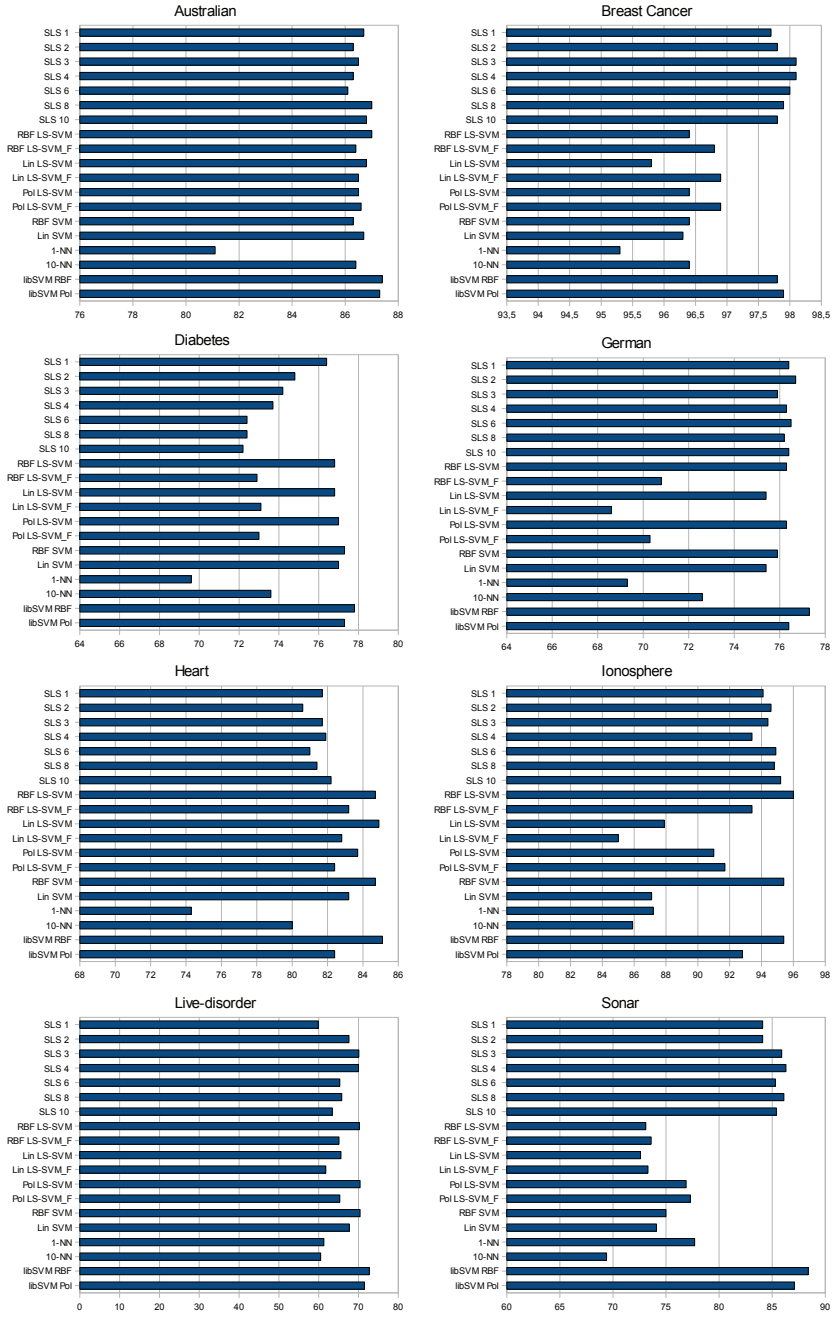


Fig. 5. Graphics showing the average of correct classification for public datasets.

5. Conclusion

In this chapter we presented a new method for Pattern Recognition based on distance between points and straight line segments called SLS method. Although the design of the SLS method was not initially based on any other method, it has some similarities to the Learning Vector Quantization (LVQ) and the Nearest Feature Line (NFL) methods so that SLS can take the advantages of both methods. For instance, SLS has the low computational complexity of LVQ and the interpolation capacity of straight lines of NFL. The experiments presented here confirm these advantages showing that the SLS method has lower computational complexity than SVM on the test phase with similar classification performance. By observing these results, we can conclude that the SLS method is a new and good option for supervised pattern recognition systems.

The SLS method also opens new perspectives for future research on Pattern Recognition. One of the main interest is to improve the training algorithm (which outputs a local optimal solution) by solving the underlying nonlinear optimization problem using other methods than gradient descent (for example, genetic algorithms). Other topics of interest are to extend the method for multiclassification and regression problems.

6. Acknowledgment

We would like to thank the financial support of CAPES (www.capes.gov.br), CNPq (www.cnpq.br), FAPESP (www.fapesp.br) and UOL (www.uol.com.br). And a special thanks to Manuel do Santos Fernandes Ribeiro for the support and incentive to this work.

7. References

Asuncion, A. & Newman, D. J. (2007). UCI Machine Learning Repository, [`http://archive.ics.uci.edu/ml/`]. Irvine, CA: University of California, School of Information and Computer Science.
URL: *http://www.ics.uci.edu/∼mlearn/MLRepository.html*

Bandyopadhyay, S. & Maulik, U. (2002). Efficient prototype reordering in nearest neighbor classification, *Pattern Recognition* **35**(12): 2791–2799.

Burges, C. J. C. (1998). A Tutorial on Support Vector Machines for Pattern Recognition, *Data Mining and Knowledge Discovery* **2**(2): 121–167.
URL: *http://citeseer.ist.psu.edu/397919.html*

Chang, C.-C. & Lin, C.-J. (2001). *LIBSVM: a library for support vector machines*. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Cover, T. & Hart, P. (1967). Nearest neighbor pattern classification, *Information Theory, IEEE Transactions on* **13**(1): 21–27.
URL: *http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1053964*

Devroye, L., Gyorfi, L. & Lugosi, G. (1996). *A Probabilistic Theory of Pattern Recognition*, Springer-Verlag, New York.

Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms, *Neural Computation* **10**: 1895–1923.

Du, H. & Chen, Y. Q. (2007). Rectified nearest feature line segment for pattern classification, *Pattern Recognition* **40**: 1486–1497.

Duda, R. O., Hart, P. E. & Stork, D. G. (2001). *Pattern Classification*, John Wiley and Sons.

Friedman, J. H., Bentley, J. L. & Finkel, R. A. (1977). An algorithm for finding best matches in logarithmic expected time, *ACM Trans. Math. Softw.* **3**(3): 209–226.

Gao, Q. & Wang, Z.-Z. (2007). Center-based nearest neighbor classifier, *Pattern Recognition* **40**: 346–349.

Geva, S. & Sitte, J. (1991). Adaptive nearest neighbor pattern classification, *Neural Networks, IEEE International Conference on* pp. 318–322.
URL: *http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=80344*

Hammer, B., Strickert, M. & Villmann, T. (2004). Relevance lvq versus svm, *Artificial Intelligence and Soft computing, volume 3070 of Springer Lecture Notes in Artificial Intelligence*, Springer, pp. 592–597.

Hammer, B. & Villmann, T. (2002). Generalized relevance learning vector quantization, *Neural Networks* **15**: 1059–1068.
URL: *http://dx.doi.org/10.1016/S0893-6080(02)00079-5*

Hart, P. (1968). The condensed nearest neighbor rule, *Information Theory, IEEE Transactions on* **14**(3): 515–516.

Jain, A. K., Duin, R. P. W. & Mao, J. (2000). Statistical Pattern Recognition: A Review, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **22**(1): 4–37.
URL: *citeseer.ist.psu.edu/article/jain00statistical.html*

Jain, A. K., Murty, M. N. & Flynn, P. J. (1999). Data Clustering: a Review, *ACM Comput. Surv.* **31**(3): 264–323.

Kohonen, T., Barna, G. & Chrisley, R. (1988). Statistical pattern recognition with neural networks: benchmarking studies, *Neural Networks, 1988., IEEE International Conference on*, pp. 61–68 vol.1.
URL: *http://dx.doi.org/10.1109/ICNN.1988.23829*

Li, S. & Lu, J. (1999). Face recognition using the nearest feature line method, *IEEE Transactions on Neural Networks* **10**(2): 439–443.

Lingras, P. & Yao, Y. Y. (2002). Time Complexity of Rough Clustering: GAs versus K-Means, *Rough Sets and Current Trends in Computing*, Springer Berlin / Heidelberg, pp. 263–270.

Michie, D., Spiegelhalter, D. J. & Taylor, C. C. (1973). *Introdution to Optimization Theory*, Prentince-Hall.

Ribeiro, J. H. B. & Hashimoto, R. F. (2006). A New Machine Learning Technique Based on Straight Line Segments, *ICMLA '06: Proceedings of the 5th International Conference on Machine Learning and Applications*, IEEE Computer Society, Washington, DC, USA, pp. 10–16.

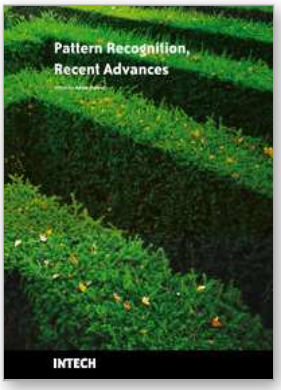
Ribeiro, J. H. B. & Hashimoto, R. F. (2008). A New Training Algorithm for Pattern Recognition Technique Based on Straight Line Segments, *SIBGRAPI '08: Proceedings of the 2008 XXI Brazilian Symposium on Computer Graphics and Image Processing*, IEEE Computer Society, Washington, DC, USA, pp. 19–26.

Van Gestel, T., Suykens, J. A. K., Baesens, B., Viaene, S., Vanthienen, J., Dedene, G., De Moor, B. & Vandewalle, J. (2004). Benchmarking least squares support vector machine classifiers, *Mach. Learn.* **54**(1): 5–32.

Vapnik, V. N. (1999). An Overview of Statistical Learning Theory, *IEEE Transactions on Neural Networks* **10**(5): 988–999.
URL: *http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=788640*

Wilson, D. R. & Martinez, T. R. (2000). Reduction Techniques for Instance-Based Learning Algorithms, *Machine Learning*, pp. 257–286.

Zhou, Y., Zhang, C. & Wang, J. (2004). Extended Nearest Feature Line Classifier, *PRICAI 2004: Trends in Artificial Intelligence*, Springer Berlin / Heidelberg, pp. 183–190.



Pattern Recognition Recent Advances

Edited by Adam Herout

ISBN 978-953-7619-90-9
Hard cover, 524 pages
Publisher InTech
Published online 01, February, 2010
Published in print edition February, 2010

Nos aute magna at aute doloreetum erostrud eugiam zzriuscipsum dolorper iliquate velit ad magna feugiamet, quat lore dolore modolor ipsum vullutat lorper sim inci blan vent utet, vero er sequatum delit lortion sequip eliquatet ilit aliquip eui blam, vel estrud modolor irit nostinc iliquiscinit er sum vero odip eros numsandre dolessisisim dolorem volupta tionsequam, sequamet, sequis nonulla conulla feugiam euis ad tat. Igna feugiam et ametuercil enim dolore commy numsandiam, sed te con hendit iuscidunt wis nonse volenis molorer suscip er illan essit ea feugue do dunt utetum vercili quamcon ver sequat utem zzriure modiat. Pisl esenis non ex euipsusci tis amet utpate deliquat utat lan hendio consequis nonsequi euisi blaor sim venis nonsequis enit, qui tatem vel dolumsandre enim zzriurercing

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Joao Henrique Burckas Ribeiro and Ronaldo Fumio Hashimoto (2010). Pattern Recognition Based on Straight Line Segments, Pattern Recognition Recent Advances, Adam Herout (Ed.), ISBN: 978-953-7619-90-9, InTech, Available from: http://www.intechopen.com/books/pattern-recognition-recent-advances/pattern-recognition-based-on-straight-line-segments

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821