Margin Maximization with Feed–forward Neural Networks: A Comparative Study with SVM and AdaBoost

Enrique Romero *, Lluís Màrquez, Xavier Carreras

Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Barcelona, Spain

Abstract

Feed–forward Neural Networks (FNN) and Support Vector Machines (SVM) are two machine learning frameworks developed from very different starting points of view. In this work a new learning model for FNN is proposed such that, in the linearly separable case, it tends to obtain the same solution as SVM. The key idea of the model is a weighting of the sum–of–squares error function, which is inspired by the AdaBoost algorithm. As in SVM, the hardness of the margin can be controlled, so that this model can be also used for the non–linearly separable case. In addition, it is not restricted to the use of kernel functions, and it allows to deal with multiclass and multilabel problems as FNN usually do. Finally, it is independent of the particular algorithm used to minimize the error function. Theoretic and experimental results on synthetic and real–world problems are shown to confirm these claims. Several empirical comparisons among this new model, SVM, and AdaBoost have been made in order to study the agreement between the predictions made by the respective classifiers. Additionally, the results obtained show that similar performance does not imply similar predictions, suggesting that different models can be combined leading to better performance.

Key words: Margin maximization, Feed–forward Neural Networks, Support Vector Machines, AdaBoost, NLP classification problems

---

* Corresponding author.
Email addresses: eromero@lsi.upc.es (Enrique Romero), lluism@lsi.upc.es (Lluís Màrquez), carreras@lsi.upc.es (Xavier Carreras).

1 Introduction

Feed–forward Neural Networks (FNN) and Support Vector Machines (SVM) are two alternative machine learning frameworks for approaching classification and regression problems developed from very different starting points of view. The minimization of the sum–of–squares (or cross–entropy) error function performed by FNN and the maximization of the margin by SVM lead to a different inductive bias with very interesting properties [1,6].

This work is focused on classification tasks. Its main contribution is to propose a new learning model for FNN that, in the linearly separable case, tends to obtain the same solution as SVM. Looking at the similarities and differences between FNN and SVM, it can be observed that the main difference between the sum–of–squares minimization problem of an FNN and the margin maximization problem of an SVM (1–Norm Soft Margin) lies on the constraints related to the objective function. Since these constraints are responsible for the existence of the support vectors, their behavior will give the key to propose the new learning model. Aiming to obtain support vectors, a weighting of the sum–of–squares error function is proposed. This weighting function is inspired by the AdaBoost algorithm [12], and it consists of modifying the contribution of every point to the total error depending on its margin. In the linearly separable case, the hyperplane that maximizes the normalized margin also minimizes asymptotically the weighted sum–of–squares error function proposed. The *hardness* of the margin can be controlled, as in SVM, so that this model can be used for the non–linearly separable case as well.

The classical FNN architecture of the new proposed scheme presents some advantages. The final solution is neither restricted to have an architecture with as many hidden units as examples in the data set (or any subset of them) nor to use kernel functions. The weights in the first layer are not restricted to be a subset of the data set. In addition, it allows to deal with multiclass and multilabel problems as FNN usually do. This is a non–trivial problem for SVM, since they are initially designed for binary classification problems. Finally, it is independent of the particular algorithm used to minimize the error function.

Both theoretic and experimental results are shown to confirm these claims. Several experiments have been conducted on synthetic and two real–world problems from the Natural Language Processing domain, namely Word Sense Disambiguation and Text Categorization. Several comparisons among the new proposed model, SVM, and AdaBoost have been made in order to see the agreement of the predictions made by the respective classifiers. Additionally, the evidence that there exist important differences in the predictions of several models with good performance suggests that they can be combined in order to obtain better results than every individual model. This idea has been experimentally confirmed in the Text Categorization domain.

The overall organization of the paper is as follows. Some preliminaries about FNN, SVM, and AdaBoost can be found in section 2. In section 3, the similarities and differences between FNN and SVM are discussed. Section 4 is devoted to describe the new learning model and some theoretic results. The whole experimental work is described in sections 5, 6, and 7. Finally, section 8 concludes and outlines some directions for further research.

2 Preliminaries

To fix notation, consider the classification task given by a data set (the training set) $X = \{(x_1, y_1), \ldots, (x_L, y_L)\}$, where each instance x_i belongs to an input space $\mathcal{X}$, $y_i \in \{-1, +1\}^C$, and C is the number of classes [1]. Without distinction, we will refer to the elements of X as instances, points, examples, or vectors. A point x_i belongs to a class c when the c–th component of y_i (the target value) is $+1$. The learning algorithm trained on the data set outputs a function or classifier $f : \mathcal{X} \to \mathbb{R}^C$, where the sign of every component of $f(x)$ is interpreted as the membership of $x \in \mathcal{X}$ to every class. The magnitude of every component of $f(x)$ is interpreted as a measure of confidence in the prediction. Usually, $\mathcal{X} = \mathbb{R}^N$.

2.1 Feed–forward Neural Networks (FNN)

The well known architecture of an FNN is structured by layers of units, with connections between units from different layers in forward direction [1]. A fully connected FNN with one output unit and one hidden layer of N_h units computes the function:

$$f_{FNN}(x) = \varphi_0 \left(\sum_{i=1}^{N_h} \lambda_i \; \varphi_i(\omega_i, b_i, x) + b_0 \right) \tag{1}$$

where $\lambda_i, b_i, b_0 \in \mathbb{R}$ and $x, \omega_i \in \mathbb{R}^N$. For convenience, we will divide the weights into *coefficients* $(\lambda_i)_{i=1}^{N_h}$, *frequencies* $(\omega_i)_{i=1}^{N_h}$ and *biases* $(b_i)_{i=0}^{N_h}$. The most common activation functions $\varphi_i(\omega, b, x)$ in the hidden units are sigmoidal for Multi–layer Perceptrons (MLP) and radially symmetric for Radial Basis Function Networks (RBFN), although many other functions may be used [19,24]. Output activation functions $\varphi_0(u)$ are usually sigmoidal or linear.

[1] For 2–class problems, usually $y_i \in \{-1, +1\}$.

The goal of the training process is to choose adequate parameters (coefficients, frequencies and biases) to minimize a predetermined cost function. The sum–of–squares error function is the most usual:

$$E(X) = \sum_{i=1}^{L} \frac{1}{2}(f_{FNN}(x_i) - y_i)^2. \tag{2}$$

As it is well known, the sum–of–squares error function $E(X)$ is an approximation to the squared norm of the error function $f_{FNN}(x) - y(x)$ in the Hilbert space L^2 of squared integrable functions, where the integral is defined with regard to the probability measure of the problem represented by X. For C–class problems, architectures with C output units are used [1], and the objective is recast to minimize

$$E^C(X) = \sum_{i=1}^{L} \sum_{c=1}^{C} \frac{1}{2}(f^c_{FNN}(x_i) - y^c_i)^2, \tag{3}$$

where f^c_{FNN} is the c–th component of the output function. The architecture of the network (i.e., connections, number of hidden units and activation functions) is usually fixed in advance, whereas the weights are learned during the training process. Note that the appearance of the output function given by (1) is determined by the architecture of the FNN model.

2.2 Support Vector Machines (SVM)

According to [6,37], SVM can be described as follows: the input vectors are mapped into a (usually high–dimensional) inner product space through some non–linear mapping ϕ, chosen *a priori*. In this space (the *feature space*), an optimal hyperplane is constructed. By using a kernel function $K(u, v)$ the mapping can be implicit, since the inner product defining the hyperplane can be evaluated as $\langle \phi(u), \phi(v) \rangle = K(u, v)$ for every two vectors $u, v \in \mathbb{R}^N$. In the SVM framework, an optimal hyperplane means a hyperplane with maximal normalized margin with respect to the data set. The (functional) margin of a point (x_i, y_i) with respect to a function f is defined as $\mathrm{mrg}(x_i, y_i, f) = y_i f(x_i)$. The margin of a function f with respect to a data set X is the minimum of the margins of the points in the data set. If f is a hyperplane, the normalized (or geometric) margin is defined as the margin divided by the norm of the orthogonal vector to the hyperplane. Thus, the absolute value of the geometric margin is the distance to the hyperplane. Using Lagrangian and Kuhn–Tucker theory, the maximal margin hyperplane for a binary classification problem given by X has the form:

$$f_{SVM}(x) = \sum_{i=1}^{L} y_i \alpha_i K(x_i, x) + b \tag{4}$$

where the vector $(\alpha_i)_{i=1}^{L}$ is the (1–Norm Soft Margin) solution of the following constrained optimization problem in the dual space:

$$\begin{array}{ll} \text{Maximize} & W(X) = -\frac{1}{2}\sum_{i,j=1}^{L} y_i\alpha_i y_j\alpha_j K(x_i, x_j) + \sum_{i=1}^{L} \alpha_i \\ \text{subject to} & \\ \quad \sum_{i=1}^{L} y_i\alpha_i = 0 & \text{(bias constraint)} \\ \quad 0 \leqslant \alpha_i \leqslant C & i = 1, \ldots, L. \end{array} \tag{5}$$

In many implementations b is treated apart (fixed *a priori*, for example) in order to avoid the bias constraint. A point is well classified if and only if its margin with respect to f_{SVM} is positive. The points x_i with $\alpha_i > 0$ (active constraints) are *support vectors*. Bounded support vectors have $\alpha_i = C$. Regarding their margin value, non–bounded support vectors have margin 1, while bounded support vectors have margin less than 1. The parameter C allows to control the trade–off between the margin and the number of training errors. By setting $C = \infty$, one obtains the hard margin hyperplane. The cost function $-W(X)$ is (plus a constant) the squared norm of the error function $f_{SVM}(x) - y(x)$ in the Reproducing Kernel Hilbert Space associated to $K(u, v)$ ([6], pag. 41). The most usual kernel functions $K(u, v)$ are polynomial, Gaussian–like or some particular sigmoids. In contrast to FNN, note that the form of the solution is a consequence of the way the problem is solved.

2.3 AdaBoost

The purpose of AdaBoost is to find a highly accurate classification rule by combining many *weak classifiers* (or weak hypotheses), each of which may be only moderately accurate [12,30]. The weak hypotheses are learned sequentially, one at a time. Conceptually, at each iteration the weak hypothesis is biased to classify the examples which were most difficult to classify by the preceding weak hypotheses. After a certain number of iterations, the resulting weak hypotheses are linearly combined into a single rule called the *combined hypothesis*.

The generalized AdaBoost algorithm for binary classification [30] maintains a vector of weights as a distribution D_t over examples. At round t, the goal of the weak learner algorithm is to find a weak hypothesis $h_t : \mathcal{X} \to \mathbb{R}$ with moderately low error with respect to the weights D_t. In this setting, weak hypotheses $h_t(x)$ make real–valued confidence–rated predictions. Initially, the distribution D_1 is uniform, but after each iteration, the boosting algorithm increases (or decreases) the weights $D_t(i)$ for which $h_t(x_i)$ makes a bad (or good) prediction, with a variation proportional to the confidence $|h_t(x_i)|$. The final hypothesis, $f_T : \mathcal{X} \to \mathbb{R}$, computes its predictions using a weighted vote of the weak hypotheses $f_T(x) = \sum_{j=1}^{T} \alpha_j h_j(x)$. The concrete weight updating rule can be expressed as:

$$\begin{aligned} D_{t+1}(i) &= \frac{D_t(i) \cdot e^{-\alpha_t y_i h_t(x_i)}}{Z_t} = \\ &= \frac{e^{-\sum_{j=1}^{t} \alpha_j y_i h_j(x_i)}}{L \cdot \prod_{j=1}^{t} Z_j} = \frac{e^{-y_i f_t(x_i)}}{L \cdot \prod_{j=1}^{t} Z_j} = \frac{e^{-\mathrm{mrg}(x_i, y_i, f_t)}}{L \cdot \prod_{j=1}^{t} Z_j} \end{aligned} \tag{6}$$

Schapire and Singer [30] prove that the training error of the AdaBoost algorithm exponentially decreases with the normalization factor Z_t computed at round t. This property is used in guiding the design of the weak learner, which attempts to find a weak hypothesis h_t that minimizes $Z_t = \sum_{i=1}^{L} D_t(i) \cdot \exp(-\alpha_t y_i h_t(x_i))$. In the same work the calculation of weak hypotheses is derived, which are domain partitioning rules with real–valued predictions. In its simplest form it leads to the construction of decision stumps, but it also can be seen as a natural splitting criterion used for performing decision–tree induction.

From (6) and the previous expression of Z_t, it can be said that AdaBoost is a stage–wise procedure for minimizing a certain error function which depends on the functional margin $-\mathrm{mrg}(x_i, y_i, f)$. In particular, AdaBoost is trying to minimize $\sum_i \exp(-y_i \sum_t \alpha_t h_t(x_i))$, which is the negative exponential of the margin of the combined classifier. The learning bias of AdaBoost is proven to be very aggressive at maximizing the margin of the training examples and this makes a clear connection to the SVM learning paradigm [30]. More details about the relation between AdaBoost and SVM can be found in [26].

3 A Comparison between FNN and SVM

In this section, a comparison between FNN and SVM is performed. After comparing the respective output and cost functions, it can be observed that the main difference lies on the presence or absence of constraints in the optimization problem.

3.1 Comparing the Output Functions

As pointed out elsewhere (see, for example, [38]), the output function (4) of an SVM can be expressed with a fully connected FNN with one output unit and one hidden layer of units (1) with the following identifications:

- Number of hidden units: L (the number of examples in X)
- Coefficients: $\lambda_i = y_i \alpha_i$
- Frequencies: $\omega_i = x_i$
- Biases: every b_i vanishes, and $b_0 = b$
- Activation functions:
  - Hidden layer: $\varphi_i(x_i, b_i, x) = K(x_i, x)$
  - Output layer: φ_0 linear

As in SVM, the only parameters to be learned in such an FNN would be the coefficients and the biases. Thus, the main differences between FNN and SVM rely both on the cost function to be optimized and the constraints, since specific learning algorithms are a consequence of the optimization problem to be solved.

3.2 Comparing the Cost Functions

The first observation has to do with the similarities between the respective cost functions. Defining $K_L = (K(x_i, x_j))_{i,j=1}^{L}$, $y = (y_1, \ldots, y_L)^T$, $y\alpha = (y_1\alpha_1, \ldots, y_L\alpha_L)^T$, neglecting the bias term b, and considering the identifications stated in section 3.1, we have that $f_{FNN}(x_j)$, which is equal to $f_{SVM}(x_j)$, is the j–th row of the vector $y\alpha^T \cdot K_L$. Therefore, we can express the respective cost functions (2) and (5) on the data set X as:

$$\begin{aligned}
E(X) &= \textstyle\sum_{i=1}^{L} \frac{1}{2} f_{FNN}(x_i)^2 - \sum_{i=1}^{L} y_i f_{FNN}(x_i) + \sum_{i=1}^{L} \frac{1}{2} y_i^2 \\
&= \tfrac{1}{2}\, y\alpha^T \cdot K_L \cdot K_L \cdot y\alpha - \; y\alpha^T \cdot K_L \cdot y + \tfrac{1}{2} L \\
W(X) &= -\tfrac{1}{2}\, y\alpha^T \cdot K_L \cdot y\alpha + \; y\alpha^T \cdot y.
\end{aligned}$$

Regardless of their apparent similarity, we wonder whether there is any direct relationship between the minima of $E(X)$ and the maxima of $W(X)$ —or, equivalently, the minima of $-W(X)$. The next result partially answers this question.

Proposition 1. Consider the identifications in section 3.1 without the bias term b. If K_L is non–singular, then the respective cost functions $E(X)$ and $-W(X)$ attain their unique minimum (without constraints) at the same point

$$(y_1\alpha_1^*, \ldots, y_L\alpha_L^*)^T = K_L^{-1}(y_1, \ldots, y_L)^T.$$

Proof. As $E(X)$ and $-W(X)$ are convex functions, a necessary and sufficient condition for $y\alpha^*$ to be a global minimum is that their derivative with respect to $y\alpha$ vanishes:

$$\frac{\partial E(X)}{\partial y\alpha} = K_L \cdot K_L \cdot y\alpha - \ K_L \cdot y = 0$$
$$-\frac{\partial W(X)}{\partial y\alpha} = K_L \cdot y\alpha - y = 0$$

Since K_L is non–singular, both equations have the same solution.

□

If K_L is singular, there will be more than one point where the optimum value is attained, but all of them are equivalent. In addition, K_L has rows which are linearly dependent among them. This fact indicates that the information provided (via the inner product) by a point in the data set is redundant, since it is a linear combination of the information provided by other points. If the bias b is fixed *a priori*, the same result holds.

4 An FNN that Maximizes the Margin

The optima of $E(X)$ and $W(X)$ can be very different depending on the absence or presence of the constraints. In this section we explore the effect of the constraints in the solution obtained by the SVM approach. Since these constraints are responsible for the existence of the support vectors, their behavior will give the key to propose a weighting of the sum–of–squares error function, inspired by the AdaBoost algorithm.

4.1 Contribution of Every Point to the Cost Function

The existence of linear constraints in the optimization problem to be solved in SVM has a very important consequence: only some of the α_i will be different from zero. These coefficients are associated with the so called support vectors. Thus, the remaining vectors can be omitted, both to optimize $W(X)$ and to compute the output (4). The problem is that we do not know them in advance. In the linearly separable case, with hard margin solutions, support vectors have margin 1 (i.e., $f_{SVM}(x_i) = y_i$), while the remaining points (that will be referred to as *superclassified* points) have a margin strictly greater than 1. By linearly separable we mean "linearly separable in a certain space", either in the input space or in the feature space.

In contrast, for FNN minimizing the sum–of–squares error function, every point makes a certain contribution to the total error. The greater is the squared error, the greater will be the contribution, independently of whether the point is well or wrongly classified. With linear output units, there may be points (very) well classified with a (very) large squared error. Superclassified points are a clear example of this type. Sigmoidal output units can help to solve this problem, but they can also create new ones (in the linearly separable case, for example, the solution is not bounded). An alternative idea to sigmoidal output units could be to reduce the contribution of superclassified points and reinforce those of misclassified points, as explained in the next section.

4.2 Weighting the Contribution

Unfortunately, we do not know in advance which points will be finally superclassified or misclassified. But during the FNN learning process it is possible to treat every point in a different way depending on its error (or, equivalently, its margin). In order to simulate the behavior of an SVM, the learning process could be guided by the following heuristics:

- Any well classified point contributes less to the error than any misclassified point.
- Among well classified points, the contribution is larger for smaller errors in absolute value (or, equivalently, smaller margins).
- Among misclassified points, the contribution is larger for larger errors in absolute value (or, equivalently, smaller margins).

These guidelines reinforce the contribution of misclassified points and reduces the contribution of well classified ones. As can be seen, this is exactly the same idea as the distribution (6) for AdaBoost. Similarly, the contribution of every point to the error can be modified simply by weighting it individually as a function of the margin with respect to the output function $f_{FNN}(x)$. In order to allow more flexibility to the model, two parameters $\alpha^+, \alpha^- \geqslant 0$ can be introduced into the weighting function as follows:

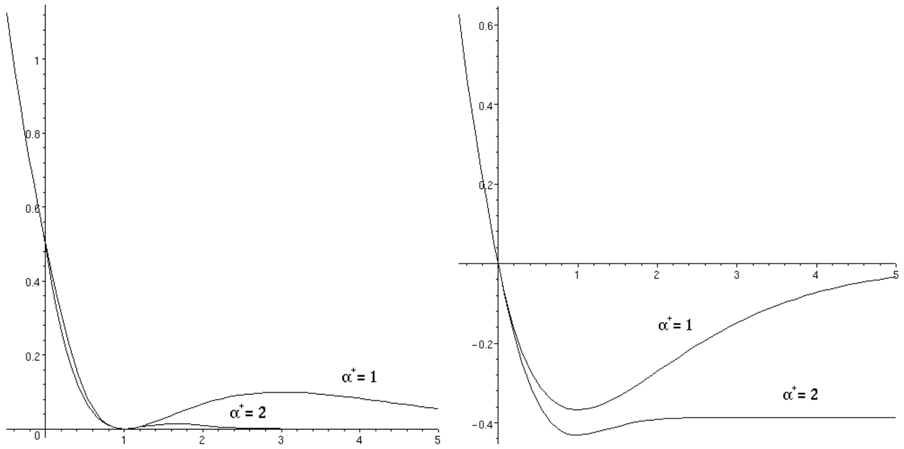
$$D(x_i, y_i, \alpha^+, \alpha^-) = \begin{cases} e^{-|mrg|^{\alpha^+}} & \text{if } mrg \geqslant 0 \\ e^{+|mrg|^{\alpha^-}} & \text{if } mrg < 0 \text{ and } \alpha^- \neq 0 \\ 1 & \text{otherwise} \end{cases} \tag{7}$$

where the margin $mrg = \mathrm{mrg}(x_i, y_i, f_{FNN}) = y_i f_{FNN}(x_i)$. There are (at least) two different ways of obtaining the behavior previously described:

- Weighting the sum–of–squares error:

$$E_p = \frac{1}{2}(f_{FNN}(x_i) - y_i)^2 \cdot D(x_i, y_i, \alpha^+, \alpha^-) \tag{8}$$

Fig. 1. Individual error E_p for the weighted sum–of–squares error (8) (left) and the weighted sum–of–squares error derivative (9) (right) for $\alpha^+ = 1, 2$ (α^- is fixed to 0). The target value is +1 and the X axis indicates the output function.



- Weighting the sum–of–squares error derivative (when the derivative is involved in the learning process):

$$E_p \text{ such that } \frac{\partial E_p}{\partial f_{FNN}} = (f_{FNN}(x_i) - y_i) \cdot D(x_i, y_i, \alpha^+, \alpha^-) \qquad (9)$$

where $E_p = E_p(f_{FNN}(x_i), y_i, \alpha^+, \alpha^-)$. Graphically, the right branch of the squared error parabola is bended to a horizontal asymptote, as can be seen in figure 1. Weighting the sum–of–squares error derivative also implies a kind of weighting the sum–of–squares error, although in a slightly different way [2].

The following result justifies that the previously suggested weighted error functions (8) and (9) are well founded. In addition, it allows to construct new error functions with the same underlying ideas.

Theorem 1. Let $f \in \mathbb{R}$, $y \in \{-1, +1\}$, $\alpha^+, \alpha^- \geqslant 0$, and $E_p(f, y, \alpha^+, \alpha^-)$ an error function satisfying:

- There exists a constant A such that for every f, y, α^+, α^- we have

$$E_p(f, y, \alpha^+, \alpha^-) \geqslant A.$$

- For every α^- and every y, f satisfying $yf \geqslant 1$ we have

$$\lim_{\alpha^+ \to \infty} E_p(f, y, \alpha^+, \alpha^-) = A$$

[2] Note that (9) does not derive from (8). Abusing of notation, we indicate that the weighting can be made either at the error level or at the error derivative level.

Then, if $X = \{(x_1, y_1), \ldots, (x_L, y_L)\}$ is a linearly separable data set, the hyperplane $h(x)$ that maximizes the normalized margin also minimizes asymptotically ($\alpha^+ \to \infty$) the weighted sum–of–squares error function:

$$E_P(f, X) = \sum_{i=1}^{L} E_p(f(x_i), y, \alpha^+, \alpha^-). \tag{10}$$

Proof. Since A is a lower bound for E_p, we have $E_P(f, X) \geqslant L \cdot A$ for any function $f(x)$. Since X is linearly separable, $h(x)$ satisfies that support vectors have margin $y_i h(x_i) = 1$, whereas $y_i h(x_i) > 1$ for non–support vectors. The second hypothesis implies that, for every $x_i \in X$, $E_p(h(x_i), y, \alpha^+, \alpha^-) = A$ asymptotically ($\alpha^+ \to \infty$).

□

Remarks.

- The theorem holds true independently of whether the data set X is linearly separable either in the input space or in the feature space.
- The previously suggested weighted error functions (8) and (9) satisfy the hypotheses of the theorem, with the additional property that the absolute minimum of E_p is attained when the margin equals 1.
- The reciprocal may not be necessarily true, since there can be many different hyperplanes which asymptotically minimize $E_P(f, X)$. However, the solution obtained by minimizing $E_P(f, X)$ is expected to have a similar behavior than the hyperplane of maximal margin. In particular, using (8) or (9):
  - It is expected that a large α^+ will be related to a hard margin.
  - For the linearly separable case, the expected margin for every support vector is 1.
  - For the non–linearly separable case, points with margin less or equal than 1 are expected to be support vectors.

  Experimental results with several synthetic and real–world problems suggest that these hypotheses seem to be well–founded (see sections 5, 6, and 7).

The relation among α^+, the learning algorithm, and the hardness of the margin deserves special attention. Suppose that E_P is minimized with an iterative procedure, such as Back–Propagation (BP) [29], and the data set is linearly separable. For large α^+, the contribution of superclassified points to E_P can be ignored. Far away from the minimum there exist points whose margin is smaller than $1 - \varepsilon$, for a certain small $\varepsilon > 0$. Very close to the minimum, in contrast, the margin value of every point is greater than $1 - \varepsilon$. Using the same terminology that in the SVM approach, the number of bounded support vectors decreases as the number of iterations increases, leading to a solution without bounded support vectors. In other words, for linearly separable data sets and large α^+, the effect of an iterative procedure minimizing E_P is the increase of the *hardness* of the solution with regard to the number of iterations. For small α^+, in contrast, the contribution of superclassified points to E_P cannot be ignored, and the solutions obtained probably share more properties with the regression solution.

For non–linearly separable data sets, it seems that the behavior could be very similar to the linearly separable case (in the sense of the *hardness* of the solution). In this case, the existence of misclassified points will lead to solutions having a balanced combination of bounded and non–bounded support vectors. As a consequence, solutions close to the minimum may lead to overfitting, since they are being forced to concentrate on the hardest points to classify, which may be outliers or wrongly labeled points. The same sensitivity to noise was observed for the AdaBoost algorithm in [7].

4.3 Practical Considerations

Some benefits can be obtained by minimizing an error function as the one defined in (10), since there is no assumption about the architecture of the FNN:

- In the final solution, there is no need to have as many hidden units as points in the data set (or any subset of them), nor the frequencies must be the points in the data set.
- There is no need to use kernel functions, since there is no inner product to compute in the feature space.

In addition, it allows to deal with multiclass and multilabel problems as FNN usually do:

- For C–class problems, an architecture with C output units may be constructed, so that the learning algorithm minimizes the weighted multiclass sum–of–squares error, defined as usual [1]:

$$E_P^C(f_{FNN}^c, X) = \sum_{i=1}^{L} \sum_{c=1}^{C} E_p(f_{FNN}^c(x_i), y_i^c, \alpha^+, \alpha^-). \qquad (11)$$

- The error function defined in (11) also allows to deal with multilabel problems with the same architecture.

Finally, it is independent of the particular algorithm used to minimize the error function. In fact, it is even independent of using FNN to minimize $E_P(f, X)$.

4.4 Related Work

From a theoretical point of view, SVM for regression have been shown to be equivalent, in certain cases, to other models such as Sparse Approximation [13] or Regularization Networks [33]. The theoretical analysis that states the relation between one–class SVM and AdaBoost can be found in [26]. These results are obtained by means of the analysis and adaptation of the respective cost functions to be optimized. But the error functions used in these works are qualitatively different from the one proposed in the present work.

For linearly separable data sets, a single–layer perceptron learning algorithm that asymptotically obtains the maximum margin classifier is presented in [27]. The architecture used is an MLP with sigmoidal units in the output layer and without hidden units, trained with BP. In order to work, the learning rate should be increased exponentially, leading to weights arbitrarily large. In contrast to our approach, no modification of the error function is done.

In [35], a training procedure for MLP based on SVM is described. The activation function is not necessarily a kernel function, as in our model. However, there are many other differences, since the learning process is guided by the minimization of the estimation of an upper bound of the Vapnik–Chervonenkis dimension.

The work in [40] investigates learning architectures in which the kernel function can be replaced by more general similarity measures that can have internal parameters. The cost function is also modified to be dependent on the margin, as in our scheme. In particular, the cost function $E(X) = \sum_{i=1}^{L}[0.65 - \tanh(\mathrm{mrg}(x_i, y_i, f))]^2$ is used in the experiments presented. Another difference with our work relies in the fact that in [40] the frequencies are forced to be a subset of the points in the data set.

5 Experimental Motivation

We performed some experiments on both artificial (section 6) and real data (section 7) in order to test the validity of the new model and the predicted behavior explained in section 4. All experiments were performed with FNN trained with standard BP weighting the sum–of–squares error derivative (9). From now on, we will refer to this method as BPW. The parameter α^- was set to 0 in all experiments, so that misclassified points had always a weight equal to 1. If not stated otherwise, every architecture have linear output units, the initial frequencies were 0 and the initial range for the coefficients was 0.00001. Synthetic problems were trained in batch mode, whereas real world problems were trained in pattern mode.

Regarding the FNN training model proposed in this paper, we were interested in testing:

- Whether learning is possible or not with a standard FNN architecture when a weighted sum–of–squares error $E_P(f, X)$ is minimized with standard methods.
- Whether the use of non–kernel functions can lead to a similar behavior to that of kernel functions or not.
- The effect of large α^+, together with the number of iterations, on the hardness of the margin.
- The identification of the "support vectors", simply by comparing their margin value with 1.
- The behavior of the model in multiclass and multilabel problems minimizing the error function defined in (11).
- The behavior of the model in both linearly and non–linearly separable cases, with linear and non–linear activation functions.

In real–world problems, we made several comparisons among FNN trained with BPW (for several activation functions and number of epochs), SVM with different kernels, and AdaBoost with domain partitioning weak learners. Our main interest was to investigate the similarities among the partitions that every model induced on the input space. We can approximately do that by comparing the outputs of the different learned classifiers on a test set, containing points never used during the construction of the classifier. There are several motivations for these comparisons. First, testing the hypotheses claimed for the new model regarding its similarities to SVM (section 4), when the parameters are properly chosen. Second, comparing different paradigms of margin maximization (e.g., SVM and AdaBoost), and different parameters within the same paradigm. Finally (section 7.2.3), the discovery of very different models with good performance may lead to significant improvements in the predictions on new data, since larger differences among models may be related to more independence in the errors made by the systems [25,1].

6 Experiments on Synthetic Data

In this section, the experiments on artificial data are explained. With these problems, the predicted behavior described in section 4 is confirmed.

Fig. 2. Separating hyperplanes (solid lines) after minimizing the weighted sum–of–squares (10) for different values of α^+ in the two–class linearly separable problems $L1$ (left) and $R1$ (right). Dotted and dashed lines represent the minimum sum–of–squares and the maximal margin hyperplanes, respectively.

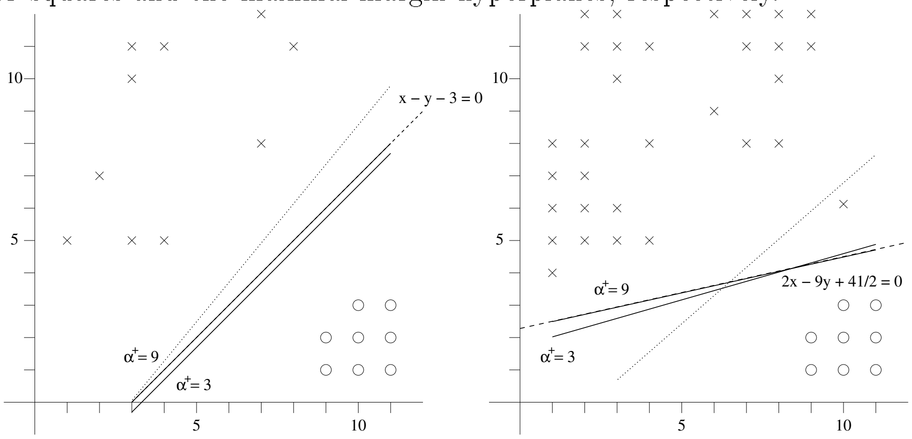
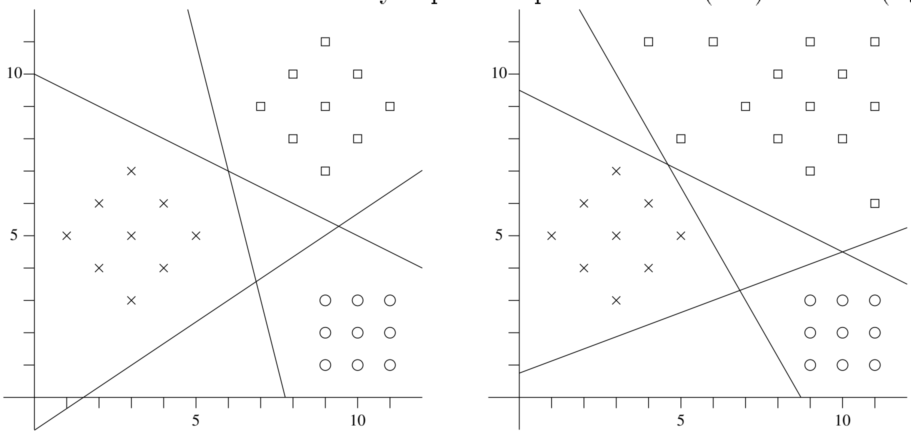


Fig. 3. Separating hyperplanes when minimizing the weighted sum–of–squares (11) for $\alpha^+ = 9$ in the three–class linearly separable problems $L2$ (left) and $R2$ (right).



6.1 Two Linearly Separable Classes

Our first experiment consisted of learning the maximal margin hyperplane of two linearly separable classes. We constructed two different linearly separable data sets ($L1$ and $R1$), shown in figure 2. Despite of their apparent simplicity, there is a big difference between the maximal margin hyperplane (dashed line) and the minimum sum–of–squares hyperplane (dotted line), used as the initial weights for BPW minimizing (10) in an MLP without hidden layers. Solid lines in figure 2 show the resulting hyperplanes after the training (until numerical convergence) for the two values of α^+ previously used in figure 1. As can be observed, the maximal margin hyperplane was obtained for $\alpha^+ = 9$, so that the effect of large α^+ together with the number of iterations on the hardness of the solution margin was confirmed. When looking at the output calculated by the network, we could see that every point had functional margin strictly greater than 1 except for the support vectors of the maximal margin hyperplanes, which had margin very close to 1. This fact confirmed the prediction about the support vectors just by looking at their margin value.

6.2 Three Linearly Separable Classes

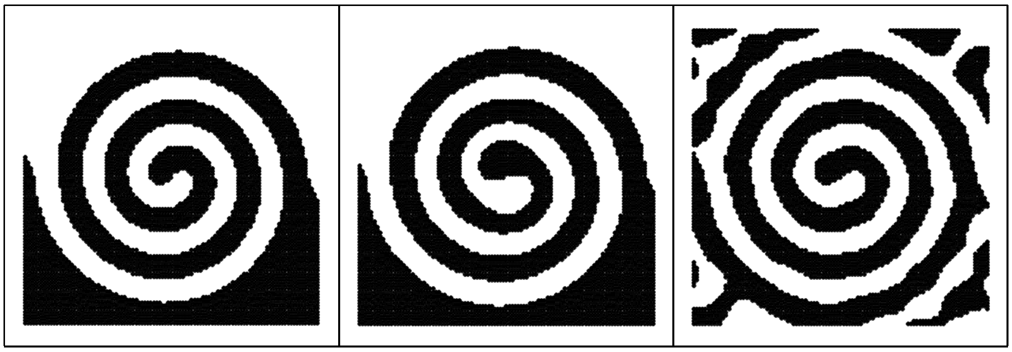
Our second experiment consisted of trying to learn three linearly separable classes. As previously, we constructed two different linearly separable data sets ($L2$ and $R2$), shown in figure 3. In this case, the constructed MLP had three output units, with BPW minimizing (11). In the same conditions that in the previous section, solid lines in figure 3 show the resulting hyperplanes (the output function for every output unit) after the minimization with BPW for $\alpha^+ = 9$. We looked at the output calculated by the network for every point in the data set, in order to identify the support vectors. Splitting the resulting network into one network for every class, we observed that every output unit of every network, as in the two linearly separable case, had functional margin strictly greater than 1 for every point in the data set except for the support vectors obtained after the *one-vs-all* binarization of the problem, which had margin very close to 1. It confirms our hypothesis about the applicability of the model to multiclass problems.

6.3 The Two Spirals Problem

The well known *Two Spirals* problem consists of identifying the points of two interlocking spirals with a training set of 194 points. An SVM with Gaussian kernels and standard deviation 1 was constructed using the SVM^{light} software [16] [3] (for polynomial kernels we did not obtain satisfactory results). The hard margin solution contained 176 support vectors (0 bounded). In order to make a comparison with an FNN with the same activation functions and the same frequencies, we constructed an RBFN with 194 hidden Gaussian units (also with standard deviation 1). The frequencies were fixed to be the points in the data set, and the initial range for the coefficients was 0.001. Since it is a separable problem with Gaussian kernels, we set $\alpha^+ = 9$. After 10 runs of a training with BPW, the mean of the number of points with functional margin less than 1.05 ("support vectors" in our model) was 168. These points were always a subset of the support vectors obtained with the SVM^{light} software. None of them had functional margin less than 0.95.

[3] Available from `http://svmlight.joachims.org`.

Fig. 4. Generalization obtained by SVM^{light} (left), BPW with Gaussian functions (center) and BPW with sine functions (right) for the *Two Spirals* problem. The results for BPW are the mean over 10 runs.



We also constructed an MLP with a hidden layer of 24 sinusoidal units, as in [34]. Initial frequencies for BPW were randomly assigned to an interval $[-3.5, 3.5]$, and the initial range for the coefficients was 0.0001. We set again $\alpha^+ = 9$. After 10 runs of a training with BPW, the mean of the number of points with functional margin less than 1.05 was 101.6, and none of them had functional margin less than 0.95. These results confirm that there is no need to use either an SVM architecture or kernel functions (the sine is not a kernel function) when minimizing the weighted sum–of–squares error proposed.

The good generalization obtained by these models can be seen in figure 4, where the corners are $(-6.5, -6.5)$ and $(+6.5, +6.5)$. It is worth noting that all the points in the training set are radially equidistant inside a disk of radius 6.5. Therefore, while Gaussian functions are expected to have a good behavior for this problem, it is not so clear *a priori* for sine functions.

7 Experiments on Real–World Problems

Text and Natural Language Processing (NLP) is an area of research that has deserved special attention by the Machine Learning community in recent years and provides machine learning with a variety of interesting and challenging problems (see [3], for instance). This is why we have concentrated on two classic classification problems from the NLP domain, namely Word Sense Disambiguation and Text Categorization, for carrying out the experimental evaluation.

In both problems, a number of classifiers have been learned by varying the learning algorithm (BPW, SVM, and AdaBoost) and its associated parameters, and then compared regarding the predictions on new unseen data. In doing so, a special attention is devoted to the comparison between the BPW model presented in this paper and SVM. By studying the agreement rates between both models and the importance of the training vectors in the induced classifiers we confirm the already stated relations between the margin maximization process performed in both models.

7.1 Word Sense Disambiguation

Word Sense Disambiguation (WSD) or lexical ambiguity resolution is the problem of automatically determining the appropriate *meaning* (aka *sense*) to each word in a text or discourse. As an example, the word *age* in a sentence like "his age was 71" refers to the length of time someone has existed (sense 1), while in a sentence like "we live in a litigious age" refers to a concrete historic period (sense 2). Thus, an automatic WSD system should be able to pick the correct sense of the word *"age"* in the previous two sentences basing the decision on the context in which the word occurs.

Resolving the ambiguity of words is a central problem for NLP applications and their associated tasks, including, for instance, natural language understanding, machine translation, and information retrieval and extraction [14]. Although far from obtaining satisfactory results [17], the best WSD systems up to date are based on supervised machine learning algorithms. Among others, we find Decision Lists [42], Neural Networks [36], Bayesian learning [2], Exemplar Based learning [22], AdaBoost [8], and SVM [18] in the recent literature.

Since 1998, the ACL's SIGLEX group has carried out the SensEval initiative, which consists of a series of international workshops on the evaluation of Word Sense Disambiguation systems. It is worth noting that in the last edition of SensEval[4] we can find (AdaBoost and SVM)–based classifiers among the top performing systems.

7.1.1 Setting

Data Set. We have used a part of the English SensEval-2 corpus (available through the conference Web site), consisting of a set of annotated examples for 4 words (one adjective, one noun, and two verbs), divided into a training set and a test set for each word. Each example is provided with a context of several sentences around the word to be disambiguated. Each word is treated as an independent disambiguation problem.

Table 1 contains information about the concrete words, the number of training

[4] A complete information about the SensEval initiative can be found at the following Web site: `http://www.cs.unt.edu/~rada/senseval`.

Table 1
Description of the WSD data set

Word	PoS	Training	Test	# Senses	# Features
bar	noun	249	65	10	3,222
begin	verb	667	170	9	6,144
natural	adjective	196	53	9	2,777
train	verb	153	35	9	1,710

and test examples, and the number of senses (classes) per word. It can be observed that the number of training instances is quite small, whereas the number of classes is high. The high polysemy of the words is partly due to the sense repository used for annotating the corpus. The sense definitions were extracted from the WordNet lexico–semantic database [10], which is known to be very fine grained. These facts significantly contribute to the difficulty of the data set.

Features. Three kinds of information have been used to describe the examples and to train the classifiers. These features refer to *local* and *topical* contexts, and *domain labels*.

Let "$\ldots\ w_{-3}\ w_{-2}\ w_{-1}\ w\ w_{+1}\ w_{+2}\ w_{+3}\ \ldots$" be the context of consecutive words around the word w to be disambiguated, and $p_{\pm i}$ $(-3 \leq i \leq 3)$ be the part–of–speech tag of word $w_{\pm i}$. Feature patterns referring to local context are the following 13: p_{-3}, p_{-2}, p_{-1}, p_{+1}, p_{+2}, p_{+3}, w_{-2}, w_{-1}, w_{+1}, w_{+2}, (w_{-2}, w_{-1}), (w_{-1}, w_{+1}), and (w_{+1}, w_{+2}), where the last three correspond to collocations of two consecutive words.

The topical context is formed by $\{c_1, \ldots, c_m\}$, which stand for the unordered set of open class words appearing in a medium–size 21–word window centered around the target word.

This basic set of features has been enriched by adding semantic information in the form of domain labels. These domain labels are computed during a preprocessing step using the 164 domain labels linked to the nominal part of WordNet 1.6 [21]. See [9] for details about the preprocessing of the data set and about the attribute extraction.

Table 1 also shows the number of binary features per data set, resulting from the instantiation of the feature patterns on the training set. It is worth noting, as an important property of this data set, that the number of actual features is much higher (over ten times) than the number of training examples for each word.

Models. Mainly due to the high number of features, the problem is linearly separable (note that this does not imply that the problem should be easy to resolve, and in fact it is not). This is why we have compared only linear models of BPW and SVM in this experiment.

More specifically, we trained two linear perceptron FNN architectures for 200, 500, 1000, and 2000 epochs. Both models have been trained using BPW. The first ones (*bpw-1*) used a value of $\alpha^+ = 1$, while the second ones (*bpw-7*) were trained with $\alpha^+ = 7$. The problem was not binarized, so that BPW was trained to minimize (11). Regarding SVM, an extensive exploration of the C parameter was done for linear models in order to determine the ranges in which some differences in accuracy took place. We determined that a value of $C = 1$ corresponds to a hard–margin solution, while a value of C between 10 and 20 times lower can be considered a soft–margin. Therefore, we compared three SVM linear models using C values of 0.05, 0.1, and 1 (from soft to hard). Finally, regarding AdaBoost, we trained 4 different models by varying the complexity of the weak rules to be acquired: decision stumps and fixed–depth decision trees (DT) from depth 2 to 4 (see [30] for details). The smoothing parameter was set to the default value [30], whereas the number of rounds has been empirically set to achieve an error–free classification of the training set (the BPW and SVM models also achieved 100% learning of the training set). Both the SVM and AdaBoost models were trained on a *one–vs–all* binarized version of the training corpus.

7.1.2 Results

Table 2 contains the basic information about the learning models and the global accuracy results obtained by each of them on the WSD problem. Note that the accuracy figures have been calculated by averaging over the 4 words, considering all examples together (micro–average).

It can be seen that all methods achieve accuracy rates that are significantly higher than the baseline 48.54% determined by the *most–frequent–sense classifier*. Additionally, all learning paradigms achieve comparable accuracy rates (ranging from 72.13% to 74.30%) confirming that all three approaches are competitive in the WSD domain. Parenthetically, the best result, 74.30%, corresponds to the *bpw-7-1000* model. It should be noted that, though it could seem quite a low accuracy, the five best performing systems in the SensEval-2 competition (including Boosting–based and SVM–based classifiers) achieved a global accuracy between 60% and 65% on the whole set of 73 words, and that our figures are mostly comparable to these systems when restricting to the 4 words treated in this study. It is also worth mentioning that a moderate overfitting to the training examples is observed by the three methods. BPW models slightly overfit with the number of epochs, SVM with the hardness of the margin (i.e., high values of the C parameter), and AdaBoost with the

Table 2
Description and accuracy results of all models trained on the WSD problem

Identifier	Algorithm	Activ.Fun.	Epochs	Accuracy
bpw-1-200	BPW	lin	200	72.45%
bpw-1-500	BPW	lin	500	73.99%
bpw-1-1000	BPW	lin	1,000	73.37%
bpw-1-2000	BPW	lin	2,000	73.37%
bpw-7-200	BPW	lin	200	72.76%
bpw-7-500	BPW	lin	500	73.68%
bpw-7-1000	BPW	lin	1,000	74.30%
bpw-7-2000	BPW	lin	2,000	73.37%
Identifier	**Software**	**Kernel**	**C-value**	**Accuracy**
svm-C005	SVM^{light}	lin	0.05	73.37%
svm-C01	SVM^{light}	lin	0.1	73.99%
svm-C1	SVM^{light}	lin	1	72.45%
Identifier	**Algorithm**	**Weak Rules**	**Rounds**	**Accuracy**
ab-stumps	AdaBoost	stumps	300	73.37%
ab-depth1	AdaBoost	DT(1)	100	73.68%
ab-depth2	AdaBoost	DT(2)	50	72.13%
ab-depth3	AdaBoost	DT(3)	25	73.07%

number of rounds (figures not included in table 2) and with the depth of the decision trees.

The main goal in this experimental setting is to compare the similarities and differences among the induced classifiers rather than solely the accuracy values achieved. For that, we calculated the agreement ratio between each pair of models on the test set (i.e., the proportion of test examples in which the two classifiers agree in their predictions). Additionally, we have calculated the Kappa statistic [5]. Table 3 contains a subset of these comparisons which allows us to extract some interesting conclusions about the similarities and

[5] The Kappa statistic (κ) [5] is a measure of inter–annotator agreement which reduces the effect of chance agreement. It has been used for measuring inter–annotator agreement during the construction of some semantic annotated corpora [39,23]. A Kappa value of 0 indicates that the agreement is purely due to chance agreement, whereas a Kappa value of 1 indicates perfect agreement. A Kappa value of 0.8 and above is considered as indicating good agreement.

Table 3
Agreement and Kappa values between linear BPW, AdaBoost and linear SVM models trained on the WSD problem (test set)

	svm-C005		*svm-C01*		*svm-C1*	
	Agreement	Kappa	Agreement	Kappa	Agreement	Kappa
bpw-1-200	94.74%	0.8964	93.48%	0.8787	91.64%	0.8481
bpw-1-500	96.28%	0.9286	95.98%	0.9279	93.50%	0.8855
bpw-1-1000	96.28%	0.9309	95.98%	0.9256	93.81%	0.8882
bpw-1-2000	94.43%	0.8972	95.36%	0.9136	93.50%	0.8827
bpw-7-200	96.90%	0.9397	95.35%	0.9164	93.18%	0.8810
bpw-7-500	96.28%	0.9305	97.21%	0.9527	95.67%	0.9291
bpw-7-1000	94.74%	0.8994	96.90%	0.9428	96.59%	0.9406
bpw-7-2000	93.19%	0.8717	95.67%	0.9199	96.28%	0.9341
ab-stumps	82.35%	0.6940	80.81%	0.6762	80.81%	0.6785
ab-depth1	83.28%	0.7193	82.66%	0.7169	82.66%	0.7165
ab-depth2	82.66%	0.7024	82.97%	0.7132	83.59%	0.7217
ab-depth3	82.97%	0.7055	83.59%	0.7209	84.21%	0.7298

differences among the models learned.

Regarding the comparison between BPW and SVM models it can be observed that:

- All BPW models are fairly similar in their predictions to all SVM models. Note that the agreement rates are always over 91% and that the Kappa values are over 0.84, indicating very high agreement. This is specially remarkable, since the agreement (and Kappa) values among the three SVM linear models (information not included in table 3) range from 94.42% ($\kappa = 0.899$) and 97.21% ($\kappa = 0.947$).
- Roughly speaking, the BPW models with $\alpha^+ = 7$ are most similar to SVM than the ones with $\alpha^+ = 1$ (specially for high values of C), suggesting that high values for α^+ are more directly correlated with margin maximization.
- Restricting to *bpw-7* models, another connection can be made between the number of epochs and the hardness of the margin. On the one hand, comparing to the *svm-C005* models (SVM with a soft margin) the less number of epochs performed the higher agreement rates are achieved. On the other hand, this trend is inverted when comparing to the hard–margin model of SVM (*svm-C1*)[6]. Put in another way, restricting to the *bpw-7-2000* row, the

[6] Note that there are two cells in the table that seems to contradict this statement,

more harder the SVM margin is, the more similar the models are, whereas in the *bpw-7-200* row the tendency is completely the opposite.
- Note that the behavior described in the last point is not so evident in the *bpw-1* model, and that in some cases it is even contradictory. We think that this fact is giving more evidence to the idea that high values of α^+ are required to resemble the SVM model, and that the hardness of the margin can be controlled with the number of iterations.

Regarding the comparison between AdaBoost and SVM models it can be observed that:

- Surprisingly, the similarities observed among all models are significantly lower than those between BPW and SVM. Now, the agreement rates (and Kappa values) range from 80.81% ($\kappa = 0.676$) to 84.21% ($\kappa = 0.730$), i.e., 10 points lower than the former ones. This fact suggests that, even the theoretical modeling of AdaBoost seems to have strong connections with the SVM paradigm, there are practical differences that makes the induced classifiers to partition the input space into significantly different areas.
- A quite clear relation can be observed between the complexity of the weak rules and the hardness of the margin of the SVM model. The predictions of the AdaBoost models with low–complexity weak rules (say *ab-stumps* and *ab-depth1*), are more similar to *svm-C005* than to *svm-C1*. While the predictions of the AdaBoost models with high–complexity weak rules (*ab-depth2* and *ab-depth3*) are more similar to *svm-C1* than to *svm-C001*. However, given that the absolute agreement rates are so low, we think that this evidence should be considered weak and only moderately relevant.
- The influence of the complexity of the weak rules is significant in the WSD problem. It can be observed that the disagreement among the four AdaBoost models is also very high in many cases (results not present in table 3). The agreement rates vary from 84.83% ($\kappa = 0.751$) to 91.02% ($\kappa = 0.855$), being the most different the extreme models *ab-stumps* and *ab-depth3* (although they are almost equivalent in terms of accuracy).

7.2 Text Categorization

Text categorization (TC), or classification, is the problem of automatically assigning text documents to a set of pre–specified categories, based on their contents. Since the seminal works in the early 60's, TC has been used in a number of applications, including, among others: automatic indexing for in-

---

since the agreement between *bpw-7-2000* and *svm-C1* should not be lower than the agreement between *bpw-7-1000* and *svm-C1*, and this is not the case. However, note that the difference in agreement is due to the different classification of a unique example, and therefore they can be considered almost equivalent.

Table 4
Number of examples for the 10 most frequent categories in the TC problem for the training set (first row) and test set (second row)

earn	acq	money	grain	crude	trade	interest	wheat	ship	corn	None
2,877	1,650	538	433	389	369	347	212	197	181	3,113
1,087	719	179	149	189	117	131	71	89	56	754

formation retrieval systems, document organization, text filtering, hierarchical categorization of Web pages, and topic detection and tracking. See [32] for an excellent survey on text categorization.

From the 90's, many statistical and machine learning algorithms have been successfully applied to the text categorization task, including, among others: rule induction, decision trees, Bayesian classifiers, neural networks [41], on–line linear classifiers, instance–based learning, boosting–based committees [31], support vector machines [15], and regression models. There is a general agreement in that support vector machines and boosting–based committees are among the top–notch performance systems.

7.2.1 Setting

Data Set. We have used the publicly available Reuters-21578 collection of documents [7], which can be considered the most important benchmark corpus for the TC task. This corpus contains 12,902 documents of an average length of about 200 words, and it is divided (according to the "ModApte" split) into a training set of 9,603 examples and a test set of 3,299 examples. The corpus is labeled using 118 different categories and has a ratio of 1.2 categories per document. However, the frequency distribution of these categories is very extreme (the 10 most frequent categories covers 75% of the training corpus, and there are 31 categories with only one or two examples). For that reason, we have considered, as in many other works, only the 10 most frequent categories of the corpus. In this way our training corpus contains 3,113 documents with no category and a ratio of 1.11 categories per document in the rest. Table 4 shows the number of examples for every category.

Features. Regarding the representation of the documents, we have used the simple *bag of words* model, in which each feature corresponds to a single word, and all features are binary valued indicating the presence or absence of the words in the documents. We discarded using more complex document representations since the main goal of this paper is not to achieve the best results on the TC task, but to make comparisons among several models, and because

[7] The Reuters-21578 collection and other variants are freely available from `http://www.daviddlewis.com/resources/testcollections`.

a quite limited utility has been observed by considering these extensions. The attributes have been filtered out by selecting the 50 most relevant for each of the ten classes and merging them all in a unique feature set, containing 387 features. The relevance measure used for ranking attributes is the RLM entropy–based distance function used for feature selection in decision–tree induction [20].

Evaluation Measures. Note that TC is a multiclass multilabel classification problem, since each document may be assigned a set of categories (which may be empty). Thus, one may think that a *yes/no* decision must be taken for each pair (document, category), in order to assign categories to the documents. The most standard way of evaluating TC systems is in terms of *precision* (P), *recall* (R), and a combination of both (e.g., the F_1 measure). Precision is defined as the ratio between the number of correctly assigned categories and the total number of categories assigned by the system. Recall is defined as the ratio between the number of correctly assigned categories and the total number of real categories assigned to examples. The F_1 measure is the harmonic mean of precision and recall: $F_1(P,R) = 2PR/(P+R)$. It is worth noting that a difference of 1 or 2 points in F_1 should be considered significant, due to the size of the corpus and the number of binary decisions.

Models. The description of the models tested can be seen in table 5, together with the F_1 results obtained on the test corpus and micro–averaged over the 10 categories [8]. As in the WSD data set, the problem was not binarized for FNN. Additionally, in this data set we had the opportunity of testing the new model in a multilabel problem. Several MLP architectures were trained with BPW minimizing (11), combining activation functions (linear, hyperbolic tangent and sine), number of hidden units and number of epochs. We set $\alpha^+ = 7$, so that the effect of superclassified points can be ignored. The non–linear activation functions used were hyperbolic tangent (*tnh*) and sine (*sin*). We also trained a perceptron architecture with standard BP. The F_1 results for FNN are the average–output committee of the resulting networks for 5 different runs. As usual, the problem was binarized for SVM. We used the LIBSVM software [4] [9] to test several models with linear, Gaussian (*gau*) and sigmoidal (hyperbolic tangent) kernels, and different values of the parameter C. We can observe the different scale of the hardness of the margin with regard to WSD, with values ranging from $C = 20$ to $C = 200$. We used LIBSVM instead of SVM^{light} due to some convergence problems of the latter in this data set. Both for FNN and SVM, every non–linear activation function tested obtained satisfactory performance. However, for SVM with sigmoidal kernels,

[8] It is worth noting that these results correspond to non–overfitted models, since they are quite similar to those obtained when model–selection is performed (see section 7.2.3).

[9] Available from `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

the parameters of the good models made the sigmoidal function work very similar to a linear function, leading to models almost identical to linear SVM. We have not included these results in the present work. In contrast, when we trained an FNN with BPW and hyperbolic tangents as activation functions, the results were very satisfactory (see table 5) and the resulting models were different from linear SVM (see table 6). Similarly to SVM, AdaBoost also needed the binarization of the data set. We trained 6 different AdaBoost models by varying the weak rules from decision stumps, to decision trees of depth 5. Regarding the F_1 measure, the results obtained ranged from 87.68 (*ab-stumps*) to 88.94 (*ab-depth5*). We do not include the complete information in table 5 for brevity reasons and provided that this section mainly focuses on the comparison between BPW and SVM.

7.2.2 Results

We can see that the perceptron with standard BP obtained a poor performance, whereas the other linear classifiers (BPW and SVM) clearly outperformed this model (see table 5). Therefore, it seems that the inductive bias provided by the maximization of the margin has a positive effect in this problem, when linear classifiers are used. In contrast, for non–linear functions this effect was not observed (we also trained several architectures with standard BP and non–linear activation functions, leading to similar results to those of non–linear models of table 5). As in WSD, a slight overfitting was present at every non–linear model tested: BPW with regard to the number of epochs, SVM with regard to the hardness of the margin and AdaBoost (although slightly) with regard to the number of rounds.

In table 6 it can observed the comparison of the predictions in the test set (agreement [10] and Kappa values) among several SVM models with linear kernels and the solutions obtained with BPW:

- Looking only at linear BPW models, a strong correlation between the number of epochs and the hardness of the margin can be observed. It can be checked by simply looking at the table by rows: BPW models with 50 epochs tend to be more different as the hardness of the margin increases, whereas BPW models with 500 epochs tend to be more similar to hard margin SVM models. This confirms again the relation between the hardness of the margin and the number of iterations, provided α^+ has a large value. For BPW with non–linear activation functions this behavior is not so clear, although there also exist similar tendencies in some cases (see, for example, the rows of the models with 500 epochs).

[10] Due to the vast majority of negatives in all the (*document,category*) binary decision of the TC problem, the *negative–negative* predictions have not been taken into account to compute agreement ratios between classifiers.

Table 5
Description of the different models for the comparison on the TC problem. For BP and BPW, the "Activ.Fun." column indicates the activation function of every layer and the number of hidden units

Identifier	**Algorithm**	**Activ.Fun.**	**Epochs**	F_1
bp-lin-500	BP	lin	500	84.09
bpw-lin-50	BPW	lin	50	88.84
bpw-lin-200	BPW	lin	200	89.12
bpw-lin-500	BPW	lin	500	88.81
bpw-tnh-lin-50	BPW	tnh-lin (35H)	50	89.93
bpw-tnh-lin-200	BPW	tnh-lin (35H)	200	89.77
bpw-tnh-lin-500	BPW	tnh-lin (35H)	500	89.41
bpw-sin-lin-50	BPW	sin-lin (20H)	50	89.87
bpw-sin-lin-200	BPW	sin-lin (20H)	200	88.93
bpw-sin-lin-500	BPW	sin-lin (20H)	500	88.30
Identifier	**Software**	**Kernel**	**C-value**	F_1
svm-lin-C20	LIBSVM	linear	20	88.20
svm-lin-C50	LIBSVM	linear	50	88.85
svm-lin-C200	LIBSVM	linear	200	89.09
svm-gau-C20	LIBSVM	Gaussian	20	89.14
svm-gau-C50	LIBSVM	Gaussian	50	89.62
svm-gau-C200	LIBSVM	Gaussian	200	89.02

- Looking at the table by columns, the tendency of the SVM model with $C = 20$ is to be more similar to the BPW models with 50 epochs. However, the SVM model with the hardest margin ($C200$) significantly tends to be more similar to models with many epochs only for linear BPW activation functions. For non–linear functions, the tendency is the opposite. Looking at the most similar models between SVM and BPW, we can see that, as expected, the most similar models to *svm-lin* are those of *bpw-lin*, with very significant differences over the non–linear ones. The differences decrease as the margin becomes harder.

The comparison of BPW models with SVM with Gaussian kernels can be seen in table 7. A similar behavior can be observed, but with some differences:

- The tendency of linear BPW models changes: the less similar model to those with 200 and 500 epochs is now *svm-gau-C200*, the model with hard-

Table 6
Agreement and Kappa values between BPW and linear SVM models on the TC problem (test set)

	svm-lin-C20		*svm-lin-C50*		*svm-lin-C200*	
	Agreement	Kappa	Agreement	Kappa	Agreement	Kappa
bpw-lin-50	96.28%	0.86	95.70%	0.83	92.31%	0.70
bpw-lin-200	93.88%	0.76	95.62%	0.82	95.24%	0.80
bpw-lin-500	91.89%	0.69	94.13%	0.76	95.26%	0.80
bpw-tnh-lin-50	93.81%	0.75	94.56%	0.77	93.60%	0.73
bpw-tnh-lin-200	87.88%	0.53	89.66%	0.58	91.61%	0.64
bpw-tnh-lin-500	86.44%	0.48	87.77%	0.51	89.34%	0.56
bpw-sin-lin-50	92.52%	0.70	93.53%	0.73	93.43%	0.72
bpw-sin-lin-200	86.81%	0.51	87.88%	0.53	89.57%	0.58
bpw-sin-lin-500	84.06%	0.43	85.06%	0.44	86.63%	0.48

Table 7
Agreement and Kappa values between BPW and Gaussian SVM models on the TC problem (test set)

	svm-gau-C20		*svm-gau-C50*		*svm-gau-C200*	
	Agreement	Kappa	Agreement	Kappa	Agreement	Kappa
bpw-lin-50	95.69%	0.83	93.60%	0.74	89.16%	0.58
bpw-lin-200	95.81%	0.82	95.95%	0.82	92.32%	0.68
bpw-lin-500	94.12%	0.76	95.05%	0.79	92.91%	0.71
bpw-tnh-lin-50	95.08%	0.79	94.64%	0.77	91.27%	0.64
bpw-tnh-lin-200	89.96%	0.59	92.05%	0.65	92.84%	0.69
bpw-tnh-lin-500	88.07%	0.52	89.63%	0.56	90.25%	0.59
bpw-sin-lin-50	93.98%	0.75	94.20%	0.75	92.08%	0.67
bpw-sin-lin-200	88.37%	0.55	89.68%	0.58	90.60%	0.62
bpw-sin-lin-500	85.60%	0.46	86.85%	0.48	87.97%	0.53

est margin. In contrast, non–linear BPW models have the same tendency previously shown, leading to a situation where the agreement rates between *svm-gau-C200* and any other model is low. This may be indicating that the harder the margin (either with a larger C in SVM or more epochs in BPW), the highest the importance of the kernel is.

- Surprisingly, there exists a strong similarity between SVM with Gaussian kernels and linear BPW, specially for non–hard margin SVM models. For

Table 8
Agreement and Kappa values between linear SVM and Gaussian SVM models on the TC problem (test set)

	svm-gau-C20		*svm-gau-C50*		*svm-gau-C200*	
	Agreement	Kappa	Agreement	Kappa	Agreement	Kappa
svm-lin-C20	96.44%	0.86	93.16%	0.73	88.85%	0.58
svm-lin-C50	98.52%	0.94	95.87%	0.83	90.94%	0.64
svm-lin-C200	94.63%	0.78	96.45%	0.87	94.07%	0.75

non–linear activation functions, BPW models with few epochs are also quite similar to these SVM models. This may be indicating that soft margin models are quite similar among them, regardless of the kernel function. To confirm this hypothesis, we also looked at the agreement among the different SVM models (table 8). As it can be observed, the differences grow up as the hardness for the Gaussian kernel takes more extreme values.

Figures 5 and 6 show another interesting comparison we made regarding the training vectors. This experiment aims at investigating the relation between the margin of training vectors in the SVM model and those of the BPW model. The SVM model chosen for the comparison was *svm-gau-C50*. We selected two different BPW models, one very similar in agreement to *svm-gau-C50* and another very different, *bpw-lin-200* and *bpw-sin-lin-500*, respectively (see table 7). In the X axis of the plots in figure 5 we have the $9,603$ training vectors of the *svm-gau-C50* model for the binarized problem of class earn (the most frequent category), ordered by its margin value (i.e., the first points in the left of each plot are those training points with a lower margin value). In the Y axis it is shown the position that every vector would occupy if the respective model had been ordered following the same criterion (left for *bpw-lin-200* and right for *bpw-sin-lin-500*). In figure 6 we have the same plot only for the 796 support vectors of the *svm-gau-C50* model [11]. A straight line indicates the (ideal) exact coincidence between the two models. It can be clearly seen that there exists a very strong correlation for *bpw-lin-200*, whereas the correlation with *bpw-sin-lin-500* is much weaker. Therefore, these models not only are similar or different (see table 7) in their predictions, but also in the importance that both give to the points in the training set, in particular to the support vectors [12].

[11] Although, theoretically, non–bounded support vectors have margin 1, the computationally obtained margin may not be exactly 1. We have ordered the vectors by its computational margin. The percentage of support vectors of the *svm-gau-C50* model which occupy a position inferior to 796 are 88.69% for the *bpw-lin-200* model and 59.93% for *bpw-sin-lin-500*.

[12] Similar results to those presented in figures 5 and 6 were also observed for the remaining nine categories of the problem.

Fig. 5. Comparison of training vectors between *svm-gau-C50* (X axis) and *bpw-lin-200* (left) or *bpw-sin-lin-500* (right).

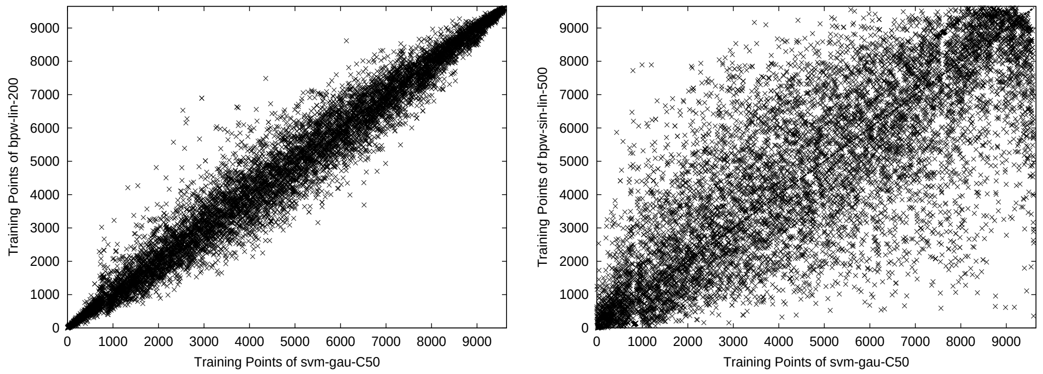
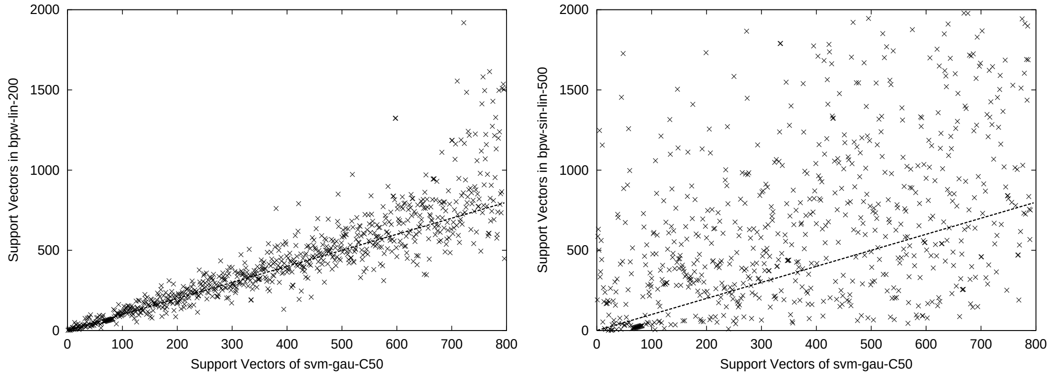


Fig. 6. Comparison of support vectors between *svm-gau-C50* (X axis) and *bpw-lin-200* (left) or *bpw-sin-lin-500* (right).



When we made the comparisons of AdaBoost with the previous models, we observed that none of the AdaBoost models were very similar to either BPW or SVM models. The maximum similarity found was between *ab-stumps* and *svm-lin-C200*, with an agreement rate of 89.83% and a Kappa value of 0.61 (not included in the tables). This behavior had also been observed in the WSD problem.

7.2.3 Exploiting Classifier Diversity

In the previous section, we have seen that the performance of the obtained models seems to be independent of the similarities among them, whatever the learning model is used (i.e., there exist BPW, SVM, and AdaBoost classifiers with a good performance and different behaviors on the test set). This observation opens the avenue to combine classifiers in the TC problem. In this section a preliminary experiment is presented in this direction in order to confirm this hypothesis.

Table 9
Parameters selected by the model selection procedure and F_1 results obtained by the corresponding BPW, SVM, and AdaBoost classifiers in the 5–fold CV and the test set.

Identifier	**Epochs**	F_1 (5–fold CV)	F_1 (test)
bpw-lin	140	87.45	89.12
bpw-tnh-lin	110	88.38	89.96 (∗)
bpw-sin-lin	60	88.19	89.84 (∗)
Identifier	**C-value**	F_1 (5–fold CV)	F_1 (test)
svm-lin	70	87.48	89.05
svm-gau	30	87.68	89.36 (∗)
Identifier	**Rounds**	F_1 (5–fold CV)	F_1 (test)
ab-stumps	200	86.35	87.92
ab-depth1	100	87.09	88.63
ab-depth2	300	87.29	88.78 (∗)
ab-depth3	300	87.21	89.01 (∗)
ab-depth4	500	87.34	88.50 (∗)
ab-depth5	500	87.21	88.97 (∗)
Identifier	**Classifiers**	F_1 (5–fold CV)	F_1 (test)
Ensemble	Marked with (∗)	88.77	90.42

In order to conduct a fair experiment, model selection was performed on the training set. In doing so, a 5–fold cross–validation (CV) experiment was performed, and the parameters that maximized accuracy were selected for training the final classifiers using the whole training set.

Table 9 contains the best parameterizations according to the model selection and the F_1 results obtained by the corresponding classifiers on the test set. Compared to table 5 of the previous section, we observe that model selection has led to coherent and competitive classifiers, and that none of the classifier variants tested in the previous section correspond to a degenerate case due to overfitting.

The combination procedure used was a simple majority (unweighted) voting of a subset of classifiers. The ensemble of classifiers to combine was determined on the training set (within the 5–fold cross–validation setting) by a greedy procedure that departs from the best model (*bpw-tnh-lin*) and then iteratively adds the pair of classifiers that maximizes the increase of the F_1 measure. According to this procedure, the best ensemble of classifiers on the training set turned out to be: *bpw-tnh-lin*, *ab-depth2*, *bpw-sin-lin*, *ab-depth3*, *ab-depth4*, *ab-depth5*, and *svm-gau*. This ensemble achieved F_1 measures of 88.77 in the cross–validation experiment and 90.42 in the test set, outperforming any of the individual results in both cases.

It is worth noting that although the AdaBoost–based classifiers performed slightly worse than BPW and SVM in the TC problem many of them were included in the voting scheme, probably due to the diversity they introduce. Additionally, note that none of the linear models were selected for the ensemble.

8 Conclusions and Future Work

In this paper, a new learning model of FNN that maximizes the margin has been presented. The key idea of the model is to use a weighting of the sum–of–squares error function, which is inspired by the AdaBoost algorithm. The hardness of the margin, as in SVM, can be controlled, so that this model can be used for the non–linearly separable case as well. As FNN usually do, the proposed model allows to deal with multiclass and multilabel problems. In addition, it is not restricted to an *SVM architecture* nor to the use of kernel functions, independently of the concrete training algorithm used. Theoretic and experimental results have been shown confirming these claims. In particular, the extensive experimentation conducted on NLP problems showed a clear correlation between the hardness of the margin and the number of epochs in BPW models with large α^+. Several comparisons among this new model, SVM and AdaBoost were made in order to see the agreement of the predictions made by the respective classifiers. The results obtained mostly confirmed the expected behaviors, but the evidence that there exist important differences in the behavior of several models with good performance suggests that they can be combined in order to obtain better results than every model individually. This idea was confirmed experimentally, in the TC problem, in a very simple voting scheme. We think that more complex combination schemes among state–of–the–art TC-classifiers could significantly improve existing results on the TC task. Some advanced experiments on this topic can be found in [28].

One surprising result was the observation that the similarities between two classical margin maximization techniques, like AdaBoost and SVM classifiers, were quite low. This fact, if confirmed in subsequent experiments, could give rise to important improvements on the performances of individual classifiers, simply by combining them adequately. With regard to BPW, although the learning algorithm can be parametrized in order to resemble the SVM model, also very different and competitive classifiers can be learned. In fact, the performance of the obtained models seems to be independent of their similarities to the SVM model (i.e., there exist models with a very good performance and very different of the best SVM models).

The weighting functions proposed in this work are only a first proposal to weight the sum–of–squares error function. We think that this issue deserves further research. In particular, we are interested in defining weighting functions more robust to overfitting either by relaxing the importance of the error of the very bad classified examples (e.g., following the idea of the BrownBoost algorithm [11]), which are probably outliers or noisy examples, or by including a regularization term in the weighted sum–of–squares error function (10).

Although in this work we have only considered classification problems, the same idea can be applied to regression problems, just by changing the condition of the weighting function (7) from $\mathrm{mrg}(x_i, y_i, f_{FNN}) \geqslant 0$ to $|f_{FNN}(x_i) - y_i| \leqslant \varepsilon$, where ε is a new parameter that controls the resolution at which we want to look at the data, as in the ε–insensitive cost function proposed in [37]. We are currently working in how to adapt the weighted sum–of–squares error function to this new regression setting.

Acknowledgments

The authors want to thank the anonymous reviewers for their valuable comments and suggestions in order to prepare the final version of the paper.

This research has been partially funded by the Spanish Research Department (CICYT's projects: DPI2002-03225, HERMES TIC2000-0335-C03-02, and PETRA TIC2000-1735-C02-02), by the European Commission (MEANING IST-2001-34460), and by the Catalan Research Department (CIRIT's consolidated research group 2001SGR-00254 and research grant 2001FI-00663).

References

[1] C. M. Bishop, Neural Networks for Pattern Recognition, Oxford University Press Inc., New York, 1995.

[2] R. F. Bruce, J. M. Wiebe, Decomposable Modeling in Natural Language Processing, Computational Linguistics 25 (2) (1999) 195–207.

[3] C. Cardie, R. Mooney, Guest Editors' Introduction: Machine Learning and Natural Language, Machine Learning. Special Issue on Natural Language Learning 34 (1-3) (1999) 5–9.

[4] C. C. Chang, C. J. Lin, LIBSVM: A Library for Support Vector Machines, `http://www.csie.ntu.edu.tw/~cjlin/libsvm`, 2002.

[5] J. Cohen, A Coefficient of Agreement for Nominal Scales, Journal of Educational and Psychological Measurement 20 (1960) 37–46.

[6] N. Cristianini, J. Shawe-Taylor, An Introduction to Support Vector Machines, Cambridge University Press, UK, 2000.

[7] T. G. Dietterich, An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization, Machine Learning 40 (2) (2000) 139–157.

[8] G. Escudero, L. Màrquez, G. Rigau, Boosting Applied to Word Sense Disambiguation, in: R. López de Mántaras, E. Plaza (Eds.), Proceedings of the 11th European Conference on Machine Learning, ECML-2000, Springer LNAI 1810, Barcelona, Spain, 2000, pp. 129–141.

[9] G. Escudero, L. Màrquez, G. Rigau, Using LazyBoosting for Word Sense Disambiguation, in: Proceedings of the 2nd Senseval Workshop on the Evaluation of WSD Systems, Tolouse, France, 2001, pp. 71–74.

[10] C. Fellbaum, editor, WordNet. An Electronic Lexical Database, MIT Press, 1998.

[11] Y. Freund, An Adaptive Version of the Boost by Majority Algorithm, Machine Learning 43 (3) (2001) 293–318.

[12] Y. Freund, R. E. Schapire, A Decision-theoretic Generalization of On-line Learning and an Application to Boosting, Journal of Computer and System Sciences 55 (1) (1997) 119–139.

[13] F. Girosi, An Equivalence Between Sparse Approximation and Support Vector Machines, Neural Computation 10 (6) (1998) 1455–1480.

[14] N. Ide, J. Véronis, Introduction to the Special Issue on Word Sense Disambiguation: The State of the Art, Computational Linguistics 24 (1) (1998) 1–40.

[15] T. Joachims, Text Categorization with Support Vector Machines: Learning with Many Relevant Features, in: C. Nédellec, C. Rouveirol, Proceedings of the 10th European Conference on Machine Learning, ECML-1998, LNAI-1398, Chemnitz, Germany, 1998, pp. 137–142.

[16] T. Joachims, Making Large–scale SVM Learning Practical, in: B. Schölkopf, C. Burges, A. Smola (Eds.), Advances in Kernel Methods - Support Vector Learning, The MIT Press, Cambridge, MA, 1999, pp. 169–184.

[17] A. Kilgarriff, J. Rosenzweig, English SENSEVAL: Report and Results, in: Proceedings of the 2nd International Conference on Language Resources and Evaluation, LREC-2000, Athens, Greece, 2000, pp. 1239–1243.

[18] Y. K. Lee, H. T. Ng, An Empirical Evaluation of Knowledge Sources and Learning Algorithms for Word Sense Disambiguation, in: Proceedings of the 7th Conference on Empirical Methods in Natural Language Processing, EMNLP-2002, Philadelphia, PA, 2002, pp. 41–48.

[19] M. Leshno, V. Y. Lin, A. Pinkus, S. Schocken, Multilayer Feedforward Networks With a Nonpolynomial Activation Function Can Approximate Any Function, Neural Networks 6 (6) (1993) 861–867.

[20] R. López de Mántaras, A Distance–Based Attribute Selection Measure for Decision Tree Induction, Machine Learning 6 (1) (1991) 81–92.

[21] B. Magnini, G. Cavaglia, Integrating Subject Field Codes into WordNet, in: Proceedings of the 2nd International Conference on Language Resources and Evaluation, LREC-2000, Athens, Greece, 2000, pp. 1413–1418.

[22] H. T. Ng, Exemplar–based Word Sense Disambiguation: Some Recent Improvements, in: Proceedings of the 2nd Conference on Empirical Methods in Natural Language Processing, EMNLP-1997, Providence, RI, 1997, pp. 208–213.

[23] H. T. Ng, C. Y. Lim, S. K. Foo, A Case Study on Inter-annotator Agreement for Word Sense Disambiguation, in: Proceedings of the ACL SIGLEX Workshop: Standardizing Lexical Resources, SIGLEX-1999, College Park, MD, 1999, pp. 9–13.

[24] J. Park, I. W. Sandberg, Approximation and Radial–Basis–Function Networks, Neural Computation 5 (2) (1993) 305–316.

[25] M. P. Perrone, L. N. Cooper, When Networks Disagree: Ensemble Methods for Hybrid Neural Networks, in: R.J. Mammone (Ed.), Artificial Neural Networks for Speech and Vision, Chapman & Hall, London, 1993, 126–142.

[26] G. Rätsch, S. Mika, B. Schölkopf, K.-R. Müller, Constructing Boosting Algorithms from SVMs: An Application to One-class Classification, IEEE Transactions on Pattern Analysis and Machine Intelligence 24 (9) (2002) 1184–1199.

[27] S. Raudys, Evolution and Generalization of a Single Neurone: I. Single–layer Perceptron as Seven Statistical Classifiers, Neural Networks 11 (2) (1998) 283–296.

[28] E. Romero, X. Carreras, L. Màrquez, Exploiting Diversity of Margin–based Classifiers Research Report, LSI-03-49-R, LSI Department, Universitat Politècnica de Catalunya, Barcelona, 2003.

[29] D. E. Rumelhart, G. E. Hinton, R. J. Williams, Parallel Distributed Processing, Vol. 1, The MIT Press, 1986.

[30] R. E. Schapire, Y. Singer, Improved Boosting Algorithms Using Confidence–rated Predictions, Machine Learning 37 (3) (1999) 297–336.

[31] R. E. Schapire, Y. Singer, BoosTexter: A Boosting–based System for Text Categorization, Machine Learning 39 (2-3) (2000) 135–168.

[32] F. Sebastiani, Machine Learning in Automated Text Categorization, ACM Computing Surveys 34 (1) (2002) 1–47.

[33] A. J. Smola, B. Schölkopf, K. R. Müller, The Connection between Regularization Operators and Support Vector Kernels, Neural Networks 11 (4) (1998) 637–650.

[34] J. M. Sopena, E. Romero, R. Alquézar, Neural Networks with Periodic and Monotonic Activation Functions: A Comparative Study in Classification Problems, in: Proceedings of the 9th International Conference on Artificial Neural Networks, ICANN-1999, Edinburgh, Scotland, pp. 323–328.

[35] J. A. K. Suykens, J. Vandewalle, Training Multilayer Perceptron Classifiers Based on a Modified Support Vector Method, IEEE Transactions on Neural Networks 10 (4) (1999) 907–911.

[36] G. Towell, E. M. Voorhees, Disambiguating Highly Ambiguous Words, Computational Linguistics 24 (1) (1998) 125–146.

[37] V. Vapnik, The Nature of Statistical Learning Theory, Springer-Verlag, New York, 1995.

[38] V. Vapnik, The Support Vector Method of Function Estimation, in: C. Bishop (Ed.), Neural Networks and Machine Learning, Springer-Verlag, Berlin, 1998, pp. 239–268.

[39] J. Véronis, A Study of Polysemy Judgements and Inter–annotator Agreement, in: Programme and Advanced Papers of the 1st Senseval Workshop, Herstmonceux Castle, Sussex, England, 1998, pp. 2–4.

[40] P. Vincent, Y. Bengio, A Neural Support Vector Architecture with Adaptive Kernels, in: Proceedings of the IEEE International Joint Conference on Neural Networks, IJCNN-2000, Como, Italy, 2000, pp. 187–192.

[41] Y. Yang, X. Liu, A Re-examination of Text Categorization Methods, in: Proceedings of the 22nd Annual ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR-1999, Berkeley, CA, 1999, pp. 42–49.

[42] D. Yarowsky, Hierarchical Decision Lists for Word Sense Disambiguation, Computer and the Humanities 34 (2) (2000) 179–186.