STATISTICS, OPTIMIZATION AND INFORMATION COMPUTING
Stat., Optim. Inf. Comput., Vol. **6**, December 2018, pp 497–525.
Published online in International Academic Press (www.IAPress.org)

Weighted Machine Learning

Mahdi Hashemi*, Hassan A. Karimi

School of Computing and Information, University of Pittsburgh, USA

Abstract Sometimes not all training samples are equal in supervised machine learning. This might happen in different applications because some training samples are measured by more accurate devices, training samples come from different sources with different reliabilities, there is more confidence on some training samples than others, some training samples are more relevant than others, or for any other reason the user wants to put more emphasis on some training samples. In non-weighted machine learning techniques which are designed for equally important training samples: (a) the cost of misclassification is equal for training samples in parametric classification techniques, (b) residuals are equally important in parametric regression models, and (c) when voting in non-parametric classification and regression models, training samples either have equal weights or their weights are determined internally by kernels in the feature space, thus no external weights. The weighted least squares model is an example of a weighted machine learning technique which takes the training samples' weights into account. In this work, we develop the weighted versions of Bayesian predictor, perceptron, multilayer perceptron, SVM, and decision tree and show how their results would be different from their non-weighted versions.

Keywords Classification, Regression, Bayesian, Multilayer Perceptron, Support Vector Machines, Decision Tree

AMS 2010 subject classifications 62H30, 68T01, 62F15

DOI: 10.19139/soic.v6i4.479

1. Introduction

In this paper, machine learning algorithms are modified to take the training samples' weights into account. The weighted machine learning techniques developed in this work, not only provide developers with the opportunity to give different weights to training samples, but also can be embedded into other algorithms such as AdaBoost [1, 2], where there is a hierarchy of classifiers, each requiring to be trained using a different weighting over training samples. Figure 1 shows, schematically, how non-weighted linear predictors become biased when the training samples' weights are taken into account. Figure 2 shows the same for nonlinear classifiers. There are two classes, one indicated with circles and the other with squares. Darkness of training samples shows their weights and the classifier is represented with a dashed line. As shown in the figure, the weighted classifier decides in favor of more important samples by keeping more distance from them.

To bias the predictor in favor of more important training samples, we embed the training samples' weights into the cost function. This way we regulate the misclassification cost based on the weights during training. In other words, misclassifying more important training samples would be more costly and the predictor will attempt to avoid it. This approach is possible only for machine learning algorithms which are based on minimizing a cost function. For Bayesian predictors, we embed the training samples' weights into the probability distribution functions. This way we increase the likelihood of a class when the irresponsive sample (the sample with an unknown output) is close to training samples with large weights in that class. In short, training the weighted predictor is more concerned

*Correspondence to: Mahdi Hashemi (Email: m.hashemi1987@gmail.com). School of Computing and Information, University of Pittsburgh. 135 North Bellefield Avenue, Pittsburgh, PA 15213, USA.

ISSN 2310-5070 (online) ISSN 2311-004X (print)

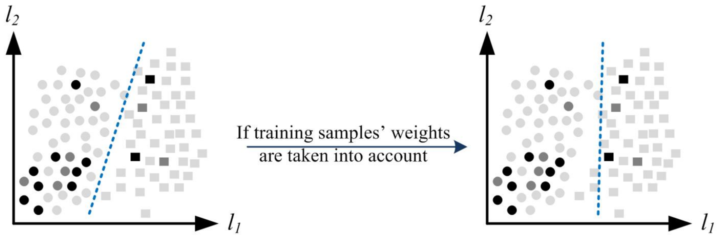

Figure 1. A non-weighted linear classifier (left) vs. a weighted linear classifier (right).

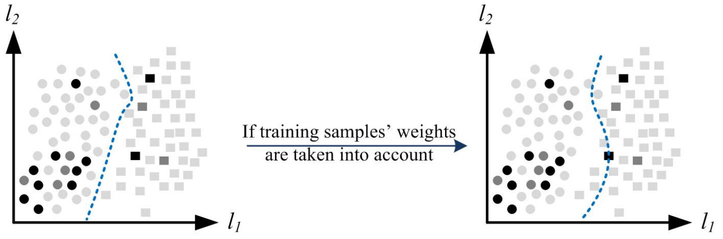

Figure 2. A non-weighted nonlinear classifier (left) vs. a weighted nonlinear classifier (right).

about correct prediction of training samples with larger weights than those with smaller weights. As a result, the trained model predicts in favor of training samples with larger weights. This makes the weighted predictor different than its non-weighted counterpart.

In this paper, we use the training dataset in Table 1 to show the difference between the weighted machine learning techniques developed here and their non-weighted counterparts. We consider two classes ω_1 and ω_2, each with 10 samples, and two features l_1 and l_2 to simplify the visualization. Training samples and their weights in this table are chosen carefully to emphasize the difference between weighted and non-weighted predictors. The training dataset is shown in Figure 3. Circles represent class ω_1 and squares represent class ω_2. Darkness of training samples shows their weight.

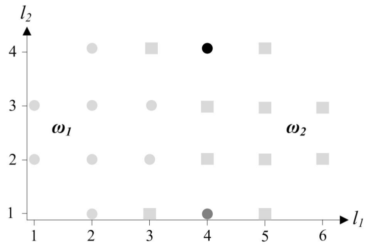

Figure 3. Training samples from two classes, circles and squares, shaded based on their weights.

2. Bayesian predictor

2.1. Classification

The Bayes classifier calculates the probability of different classes given the observed feature vector as $p(\omega_j|x) = p(\omega_j)p(x|\omega_j)/p(x)$ and then assigns x to the class with the highest probability [3, 4]; where $p(\omega_j|x)$ is the posterior probability, $p(\omega_j)$ is the prior probability, and $p(x|\omega_j)$ is the likelihood. The denominator, $p(x)$, is usually ignored in calculations as it is the same for all classes. A simple way to embed weights (g_i) for training samples into the

Table 1. Training samples and their weights.

l_1	l_2	Class	Weight
1	2	ω_1	1
1	3	ω_1	1
2	1	ω_1	1
2	2	ω_1	1
2	3	ω_1	1
2	4	ω_1	1
3	2	ω_1	1
3	3	ω_1	1
4	1	ω_1	2
4	4	ω_1	4
3	1	ω_2	1
3	4	ω_2	1
4	2	ω_2	1
4	3	ω_2	1
5	1	ω_2	1
5	2	ω_2	1
5	3	ω_2	1
5	4	ω_2	1
6	2	ω_2	1
6	3	ω_2	1

Bayes classifier is to define the prior probability ($p(\omega_j)$) as the sum of weights of training samples belonging to class ω_j divided by the sum of all weights (1).

$$p(\omega_j) = \sum_{\forall i | x_i \in \omega_j} g_i / \sum_{\forall i} g_i \tag{1}$$

Regardless of parametric or non-parametric definition of the likelihood ($p(x|\omega_j)$), an important drawback with this simple approach is that it does not consider where the irresponsive sample (x) is situated with respect to more important training samples in each class. For example, the irresponsive sample in Figure 4, shown with a cross, is closer to more important samples (darker ones in the figure) in ω_2 and one expects it to be classified in ω_2. However, based on the aforementioned approach, it will be classified in ω_1 because ω_1 has a larger prior ($p(\omega_1) > p(\omega_2)$) and the likelihoods for two classes are equal ($p(x|\omega_1) = p(x|\omega_2)$). Likelihoods, $p(x|\omega_1)$ and $p(x|\omega_2)$, are calculated without considering the weights. To solve this problem, weights need to be considered in likelihoods.

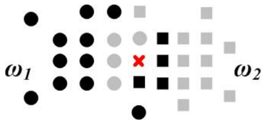

Figure 4. Two classes shown with circles and squares where the darkness of samples shows their weight.

To take into account the position of x with respect to more important training samples in each class, we define the likelihood ($p(x|\omega_j)$) based on non-parametric Parzen windows [5], shown in (2), instead of calculating the priors from (1).

$$p(x|\omega_j) = \frac{1}{N_j} \sum_{\forall i | x_i \in \omega_j} g_i K(x - x_i, \Sigma_j) \tag{2}$$

In this equation, N_j is the size of the class ω_j, x_i represents the i-th training sample's feature vector, x represents the irresponsive sample's feature vector, g_i is the i-th training sample's weight, K is the kernel function, and Σ_j is the covariance matrix of features obtained based only on samples from class ω_j. The step kernel in (3) or the Gaussian kernel in (4) can be used in (2), where l is the dimension of feature space and the subscript k in x_k and x_{i_k} refers to the k-th feature in the corresponding feature vector. More kernels are available in Hardle [6] and Fan and Gijbels [7]. Instead of choosing the kernel bandwidth to be a constant value, which is the common practice, we choose the covariance matrix for class ω_j (shown by Σ_j) divided by a constant value (shown by σ) as the kernel bandwidth for class j. The constant value (σ) can be tuned using cross-validation.

$$K(x - x_i, \Sigma_j) = \begin{cases} \dfrac{1}{|\Sigma_j/\sigma|^{1/2}} & |x_k - x_{i_k}| < \frac{1}{2}|\Sigma_j/\sigma|^{\frac{1}{2l}} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

$$K(x - x_i, \Sigma_j) = \frac{1}{(2\pi)^{l/2}|\Sigma_j/\sigma|^{1/2}} \exp(-\frac{1}{2}(x - x_i)^T(\Sigma_j/\sigma)^{-1}(x - x_i)) \tag{4}$$

Applying (2) to calculate the likelihoods in Figure 4 results in $p(x|\omega_1) < p(x|\omega_2)$ and consequently $p(\omega_1|x) < p(\omega_2|x)$ which classifies the irresponsive sample in ω_2.

2.2. *Regression*

In case of regression, (5) can be used to estimate the response at the irresponsive sample x. This equation estimates the response at x as the weighted average of other training samples' responses, where each training sample's weight is the multiplication of its original weight (g_i) by the output of the kernel for that training sample ($K(x - x_i, \Sigma)$). In other words, a training sample's weight in this equation ($g_i K(x - x_i, \Sigma)$) is the combination of its importance as well as its distance to the irresponsive sample in the feature space. The latter is what the kernel is concerned about.

$$y(x) = \frac{\sum_{i=1}^{N} y_i g_i K(x - x_i, \Sigma)}{\sum_{i=1}^{N} g_i K(x - x_i, \Sigma)} \tag{5}$$

Since there are no classes in regression, the covariance matrix of features (Σ) in (5) is defined over all training samples.

2.3. *Experiment*

Here we use the dataset in Table 1 to show the effect of embedding training samples' weights in likelihoods (2) on the irresponsive sample's classification. Priors are considered equal since the frequencies of the two classes are the same. The Gaussian kernel in (4) with a bandwidth of $\Sigma_j/3$ is used as Parzen window, where Σ_j shows the covariance matrix of class ω_j. Figure 5 shows the division of the feature space between the two classes with and without considering the training samples' weights in calculating the likelihoods. It is shown that when the weighted Bayesian classifier is applied, the classification of the irresponsive sample (shown with a cross) is switched from class ω_2 to class ω_1 because of its proximity to some important samples in class ω_1.

3. Linear predictors

3.1. *Least squares (LS)*

The output of the least squares (LS) predictor is $\boldsymbol{x}^T\boldsymbol{w}$ where $\boldsymbol{w}$ is the extended weight vector to include the threshold or intercept (w_0) and $\boldsymbol{x}$ is the extended feature vector to include a 1. The desired output is denoted with y_i. The weight vector will be computed so as to minimize the sum of squared errors between the desired and true outputs [8], that is:

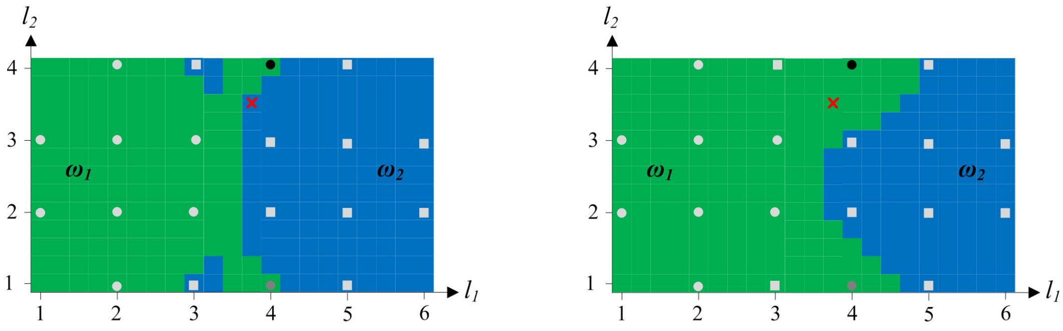

Figure 5. Division of the feature space between the two classes, circles and squares, without (left) and with (right) considering the training samples' weights (darkness of samples) in Bayesian classifier.

$$J(\boldsymbol{w}) = \sum_{i=1}^{N} (y_i - \boldsymbol{x}_i^T \boldsymbol{w})^2 \tag{6}$$

where N is the number of training samples. Minimizing the cost function in (6) with respect to $\boldsymbol{w}$ results in:

$$\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} = 0 \rightarrow \sum_{i=1}^{N} \boldsymbol{x}_i (y_i - \boldsymbol{x}_i^T \boldsymbol{w}) = 0 \rightarrow (\sum_{i=1}^{N} \boldsymbol{x}_i \boldsymbol{x}_i^T) w = \sum_{i=1}^{N} \boldsymbol{x}_i y_i \tag{7}$$

Let us define:

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ . \\ . \\ \boldsymbol{x}_N^T \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & ... & x_{1l} & 1 \\ x_{21} & x_{22} & ... & x_{2l} & 1 \\ . & . & ... & . & . \\ . & . & ... & . & . \\ x_{N1} & x_{N2} & ... & x_{Nl} & 1 \end{bmatrix} \quad \text{and} \quad y = \begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ y_N \end{bmatrix} \tag{8}$$

where $\boldsymbol{X}$ is an $N \times (l+1)$ matrix whose rows are the feature vectors with an additional 1, l is the number of features, and y is a vector consisting of the corresponding desired responses. Then:

$$\sum_{i=1}^{N} \boldsymbol{x}_i \boldsymbol{x}_i^T = \boldsymbol{X}^T \boldsymbol{X} \quad \text{and} \quad \sum_{i=1}^{N} \boldsymbol{x}_i y_i = \boldsymbol{X}^T y \tag{9}$$

By substituting (9) in (7) we have:

$$(\boldsymbol{X}^T \boldsymbol{X}) \boldsymbol{w} = \boldsymbol{X}^T y \rightarrow \boldsymbol{w} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T y \tag{10}$$

Matrix $\boldsymbol{X}^+ = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T$ is known as the pseudoinverse of $\boldsymbol{X}$ and is equal to $\boldsymbol{X}^{-1}$ if $\boldsymbol{X}$ is square. To develop the weighted version of LS predictor, we adjust the cost of error based on the weight of training samples (g_i),

$$J(w) = \sum_{i=1}^{N} g_i (y_i - \boldsymbol{x}_i^T \boldsymbol{w})^2 \tag{11}$$

Minimizing the cost function in (11) with respect to $\boldsymbol{w}$ results in:

$$\frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}} = 0 \rightarrow \sum_{i=1}^{N} g_i \boldsymbol{x}_i (y_i - \boldsymbol{x}_i^T \boldsymbol{w}) = 0 \rightarrow (\sum_{i=1}^{N} g_i \boldsymbol{x}_i \boldsymbol{x}_i^T) \boldsymbol{w} = \sum_{i=1}^{N} g_i \boldsymbol{x}_i y_i \tag{12}$$

Let us define:

$$G = \begin{bmatrix} g_1 & 0 & 0 & ... & 0 \\ 0 & g_2 & 0 & ... & 0 \\ . & . & . & ... & . \\ . & . & . & ... & . \\ 0 & 0 & 0 & ... & g_N \end{bmatrix} \tag{13}$$

Then:

$$\sum_{i=1}^{N} g_i \boldsymbol{x}_i \boldsymbol{x}_i^T = \boldsymbol{X}^T G \boldsymbol{X} \quad \text{and} \quad \sum_{i=1}^{N} g_i \boldsymbol{x}_i y_i = \boldsymbol{X}^T G y \tag{14}$$

Substituting (14) in (12) results in:

$$(\boldsymbol{X}^T G \boldsymbol{X})\boldsymbol{w} = \boldsymbol{X}^T G y \rightarrow \boldsymbol{w} = (\boldsymbol{X}^T G \boldsymbol{X})^{-1} \boldsymbol{X}^T G y \tag{15}$$

Equation (15) is known as weighted least squares [9]. Let us investigate what happens if the weight of all training samples is equal to a constant c. In this case, $G = c \times I_{N \times N}$ where $I_{N \times N}$ is the $N \times N$ identity matrix. Substituting this in (15) results in:

$$\boldsymbol{w} = (\boldsymbol{X}^T cI\boldsymbol{X})^{-1} \boldsymbol{X}^T cIy = (c\boldsymbol{X}^T \boldsymbol{X})^{-1} c\boldsymbol{X}^T y = \frac{1}{c} (\boldsymbol{X}^T \boldsymbol{X})^{-1} c\boldsymbol{X}^T y = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T y \tag{16}$$

In other words, the weighted LS is no different than the non-weighted LS if all weights are equal. This is the case with all weighted predictors developed in this work.

3.1.1. Experiment. Here we use the dataset in Table 1 to show the effect of embedding the training samples' weights in LS (15). Figure 6 shows the division of the feature space between the two classes with and without considering the training samples' weights in computing the linear classifier. Circles represent class ω_1 and squares represent class ω_2. Darkness of training samples shows their weight. In the weighted LS classifier, training samples with large weights from class ω_1 push the border toward class ω_2.

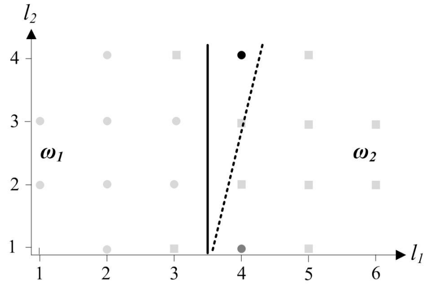

Figure 6. Division of the feature space between the two classes, circles and squares, without (solid line) and with (dashed line) considering the training samples' weights (darkness of samples) in LS classifier.

3.2. *Perceptron*

The perceptron cost function is defined as [10]:

$$J(\boldsymbol{w}) = \sum_{i=1}^{N} y_i \boldsymbol{w}^T \boldsymbol{x}_i \quad , \quad y_i = \begin{cases} +1 \text{if } \boldsymbol{w}\boldsymbol{x}_i > 0 \text{ but } \boldsymbol{x}_i \in \omega_2 \\ -1 \ \text{ if } \boldsymbol{w}\boldsymbol{x}_i < 0 \text{ but } \boldsymbol{x}_i \in \omega_1 \\ 0 \quad \text{if } \boldsymbol{w}\boldsymbol{x}_i > 0 \text{ and } \boldsymbol{x}_i \in \omega_1 \\ 0 \quad \text{if } \boldsymbol{w}\boldsymbol{x}_i < 0 \text{ and } \boldsymbol{x}_i \in \omega_2 \end{cases} \tag{17}$$

where N is the number of training samples, $\boldsymbol{x}_i$ is the i-th feature vector including an additional 1 as its last element, and $\boldsymbol{w}$ is the weight vector (the perpendicular vector to the hyperplane classifier toward class ω_1) including the threshold (w_0) as its last element. The cost function is minimized if the classifier produces a positive response for samples of class ω_1 and a negative response for samples of class ω_2. We can iteratively find the weight vector that minimizes the perceptron cost function using the gradient descent scheme [10, 11]:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \Delta\boldsymbol{w}_t = \boldsymbol{w}_t - \alpha\frac{\partial J(\boldsymbol{w})}{\partial\boldsymbol{w}}|_{\boldsymbol{w}=\boldsymbol{w}_t} = \boldsymbol{w}_t - \alpha\sum_{i=1}^{N} y_i\boldsymbol{x}_i \tag{18}$$

where $\boldsymbol{w}_t$ is the weight vector estimate at the t-th iteration and α is the training rate which is a small positive number.

We embed the training samples' weights (g_i) in the perceptron cost function (19) to punish the classifier more for misclassifying training samples with larger weights and less for training samples with smaller weights. In other words, the training samples' weights enter the cost function to adjust the perceptron cost based on the importance of training samples. The perceptron classifier is no longer equally fair to all training samples.

$$J(\boldsymbol{w}) = \sum_{i=1}^{N} g_i y_i \boldsymbol{w}^T\boldsymbol{x}_i \tag{19}$$

With the new cost function, the iterative steps for updating the weight vector through the gradient descent scheme will change to:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \Delta\boldsymbol{w}_t = \boldsymbol{w}_t - \alpha\frac{\partial J(\boldsymbol{w})}{\partial\boldsymbol{w}}|_{\boldsymbol{w}=\boldsymbol{w}_t} = \boldsymbol{w}_t - \alpha\sum_{i=1}^{N} g_i y_i\boldsymbol{x}_i \tag{20}$$

If we define $\tilde{\alpha}_i = \alpha g_i$ we obtain:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \sum_{i=1}^{N} \tilde{\alpha}_i y_i\boldsymbol{x}_i \tag{21}$$

Therefore, the weighted perceptron classifier can be obtained by including the weights in the cost and defining the training rate as $\tilde{\alpha}_i = \alpha g_i$ which means a different training rate for each training sample based on its weight. Adjusting the training rate based on the training samples' weights and including the weights in the cost function bias the trained perceptron in favor of training samples with larger weights.

3.2.1. Experiment. Here we use the dataset in Table 1 to show the effect of including training samples' weights in perceptron classifier. Figure 7 shows the division of the feature space between the two classes with and without considering the training samples' weights in computing the linear classifier. The high cost of misclassifying important samples from class ω_1 in weighted perceptron classifier pushes the border toward class ω_2.

3.3. SVM

3.3.1. Two linearly separable classes. Assume ω_1 and ω_2 are two linearly separable classes shown in Figure 8. SVM [12, 13, 14] maximizes the margin around the hyperplane separating the two classes. We know that the distance between a sample x_i and a hyperplane $f(x) = w^T x + w_0 = 0$ is obtained from $|f(x_i)|/||w||$. Assume x_1 is the nearest sample in class ω_1 to the hyperplane $f(x)$ and x_2 is the nearest sample in class ω_2 to the hyperplane $f(x)$. Then x_1 and x_2 are called support vectors. To maximize the margin, the hyperplane $f(x)$ must intersect the line connecting x_1 and x_2 at its midpoint, as shown in Figure 8. Therefore, we can scale w and w_0 so that $f(x_1) = 1$ and $f(x_2) = -1$. This leads to having a margin of:

$$\frac{|f(x_1)|}{||w||} + \frac{|f(x_2)|}{||w||} = \frac{1}{||w||} + \frac{1}{||w||} = \frac{2}{||w||} \tag{22}$$

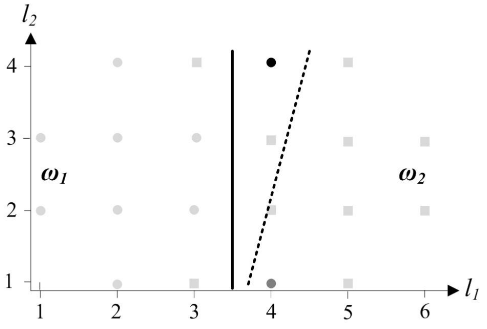

Figure 7. Division of the feature space between the two classes, circles and squares, without (solid line) and with (dashed line) considering the training samples' weights (darkness of samples) in perceptron classifier (logistic activation function and adaptive training rate with 1000 iterations).

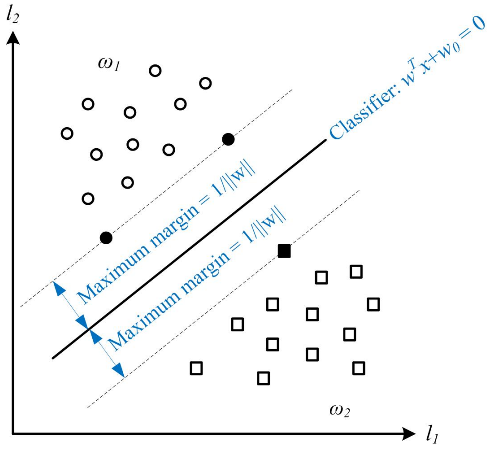

Figure 8. SVM classifier for two linearly separable classes; black points show support vectors.

Since x_1 and x_2 are the closest samples to the hyperplane $f(x)$, the distance of other samples from the hyperplane is greater than $1/||w||$, as shown in Figure 8. Therefore, we have:

$$\begin{cases} f(x_i) \geq 1 \quad \forall x_i \in \omega_1 \\ f(x_i) \leq -1 \; \forall x_i \in \omega_2 \end{cases} \tag{23}$$

We define:

$$y_i = \begin{cases} +1 \; \forall x_i \in \omega_1 \\ -1 \quad \forall x_i \in \omega_2 \end{cases} \tag{24}$$

Substituting (24) in (23) results in:

$$y_i f(x_i) \geq 1 \quad , \quad \forall x_i \tag{25}$$

We need to maximize the margin $(2/||w||)$ in (22) which is equivalent to minimizing the norm $||w||$. The mathematical formulation for finding w and w_0 of the hyperplane follows:

$$\begin{cases} minimize J(w, w_0) = \frac{1}{2}||w||^2 = \frac{1}{2}w^T w & (26) \\ \text{subject to } y_i(w^T x_i + w_0) \geq 1 \quad , \quad i = 1, 2, ..., N & (27) \end{cases}$$

where N is the number of training samples. The above cost function is convex and the constraints are linear and define a convex set of feasible solutions. The corresponding Lagrangian function $\mathcal{L}(w, w_0, \lambda)$ for the above convex programming problem is defined as follows [15, 16, 17, 18]:

$$\mathcal{L}(w, w_0, \lambda) = \frac{1}{2}w^T w - \sum_{i=1}^{N} \lambda_i [y_i(w^T x_i + w_0) - 1] \tag{28}$$

where λ_i, $i = 1, 2, , N$ are the Lagrangian multipliers associated with the constraint in (27). We need to find w, w_0, and λ by solving the Lagrangian duality: $max_{\lambda \geq 0} min_{w,w_0} \mathcal{L}(w, w_0, \lambda)$ [15, 16, 17, 18]. The Karush-Kuhn-Tucker conditions that $min_{w,w_0} \mathcal{L}(w, w_0, \lambda)$ has to satisfy are [15, 16, 17, 18]:

$$\begin{cases} \dfrac{\partial \mathcal{L}(w, w_0, \lambda)}{\partial w} = 0 \rightarrow w = \displaystyle\sum_{i=1}^{N} \lambda_i y_i x_i & (29) \\ \dfrac{\partial \mathcal{L}(w, w_0, \lambda)}{\partial w_0} = 0 \rightarrow \displaystyle\sum_{i=1}^{N} \lambda_i y_i = 0 & (30) \\ \lambda_i \geq 0 \quad , \quad i = 1, 2, ..., N & (31) \\ \lambda_i [y_i(w^T x_i + w_0) - 1] = 0 \quad , \quad i = 1, 2, ..., N \quad \text{(complementary slackness conditions)} & (32) \end{cases}$$

Equations (29) and (30) depend only on training samples whose $\lambda_i \neq 0$, referred to as support vectors. On the other hand, the conditions in (32) state that either λ_i or $y_i(w^T x + w_0) - 1$ must be zero. Therefore support vectors are training samples where $|w^T x + w_0| = 1$ (i.e. $y_i(w^T x + w_0) - 1 = 0$ and $\lambda_i \neq 0$) which means they are on the boundary of the margin. Therefore, (29) and (30) depend only on support vectors and consequently the hyperplane classifier is designed only based on support vectors and is independent of other training samples because their λ_i is zero. While, none of the training samples falls inside the margin (by construction), this is not necessarily the case for irresponsive samples. The intuition is that maximizing the margin on the training samples will lead to good separation on the irresponsive samples.

By expanding (28), we have:

$$\mathcal{L}(w, w_0, \lambda) = \frac{1}{2}w^T w - w^T \sum_{i=1}^{N} \lambda_i y_i x_i - w_0 \sum_{i=1}^{N} \lambda_i y_i + \sum_{i=1}^{N} \lambda_i$$

By replacing $\sum_{i=1}^{N} \lambda_i y_i = 0$ from (30) in the above equation, we get:

$$\mathcal{L}(w, w_0, \lambda) = \frac{1}{2}w^T w - w^T \sum_{i=1}^{N} \lambda_i y_i x_i + \sum_{i=1}^{N} \lambda_i$$

By substituting w from (29), we have:

$$\mathcal{L}(w, w_0, \lambda) = \frac{1}{2}\left[\sum_{i=1}^{N} \lambda_i y_i x_i\right]^T \left[\sum_{i=1}^{N} \lambda_i y_i x_i\right] - \left[\sum_{i=1}^{N} \lambda_i y_i x_i\right]^T \left[\sum_{i=1}^{N} \lambda_i y_i x_i\right] + \sum_{i=1}^{N} \lambda_i$$

$$\mathcal{L}(w, w_0, \lambda) = -\frac{1}{2}\left[\sum_{i=1}^{N} \lambda_i y_i x_i\right]^T \left[\sum_{i=1}^{N} \lambda_i y_i x_i\right] + \sum_{i=1}^{N} \lambda_i$$

$$\mathcal{L}(w, w_0, \lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j x_i^T x_j$$

Now we maximize the above Lagrangian function with respect to λ:

$$\begin{cases} \max\limits_{\lambda} \left(\sum\limits_{i=1}^{N} \lambda_i - \frac{1}{2} \sum\limits_{i=1}^{N} \sum\limits_{j=1}^{N} \lambda_i \lambda_j y_i y_j x_i^T x_j \right) & (33) \\ \text{subject to } \sum\limits_{i=1}^{N} \lambda_i y_i = 0 & (34) \\ \lambda_i \geq 0 \quad , \quad i = 1, 2, ..., N & (35) \end{cases}$$

Once the optimal Lagrangian multipliers (λ_i) have been computed by maximizing (33), w is obtained by replacing them in (29) and w_0 is computed as an average value obtained using complementary slackness conditions in (32) for support vectors ($\lambda_i \neq 0$).

The weighted version of SVM needs to be more sensitive to training samples with larger weights. In other words, the distance from training samples to the classifier hyperplane needs to be compromised based on their weights. From a geometric point of view, we develop the weighted SVM by moving training samples toward the classifier hyperplane by a factor proportional to their weight (g_i). We measure the distance of a training sample (x_i) from the classifier hyperplane ($f(x)$) through (36), where the actual distance is reduced by a factor of $1/(1+g_i)$. If a training sample's weight is zero, its distance to the classifier hyperplane, in (36), remains intact, and if its weight is very large, its distance will become close to zero.

$$\frac{|f(x_i)|}{||w|| \times (1+g_i)} = \frac{|w^T x_i + w_0|}{||w|| \times (1+g_i)} \tag{36}$$

Assume x_1 is the nearest sample in class ω_1 to the classifier hyperplane based on the distance calculated from (36) and x_2 is the nearest sample in class ω_2 to the classifier hyperplane. We can scale w and w_0 so that $f(x_1)/(1+g_1) = 1$ and $f(x_2)/(1+g_2) = -1$. This leads to having a margin of:

$$\frac{|f(x_1)|}{||w|| \times (1+g_1)} + \frac{|f(x_2)|}{||w|| \times (1+g_2)} = \frac{1}{||w||} + \frac{1}{||w||} = \frac{2}{||w||} \tag{37}$$

Since x_1 and x_2 are the closest samples to the hyperplane, the distance of other samples from the hyperplane (based on (36)) is larger than 1. Therefore, we have:

$$\begin{cases} f(x_i)/(1+g_i) \geq 1 \quad \forall x_i \in \omega_1 \\ f(x_i)/(1+g_i) \leq -1 \, \forall x_i \in \omega_2 \end{cases} \tag{38}$$

We define:

$$y_i = \begin{cases} +1 \, \forall x_i \in \omega_1 \\ -1 \quad \forall x_i \in \omega_2 \end{cases} \tag{39}$$

Substituting (39) in (38) results in:

$$y_i f(x_i)/(1+g_i) \geq 1 \quad , \quad \forall x_i \tag{40}$$

We need to maximize the margin ($2/||w||$) in (37) which is equivalent to minimizing the norm $||w||$. The mathematical formulation for finding w and w_0 of the hyperplane follows:

$$\begin{cases} minimize J(w, w_0) = \frac{1}{2}||w||^2 = \frac{1}{2} w^T w & (41) \\ \text{subject to } y_i \left(\frac{w^T x_i + w_0}{1+g_i} \right) \geq 1 \quad , \quad i = 1, 2, ..., N & (42) \end{cases}$$

where N is the number of training samples. The above cost function is convex and the constraints are linear and define a convex set of feasible solutions. The corresponding Lagrangian function $\mathcal{L}(w, w_0, \lambda)$ for the above convex programming problem is defined as follows:

$$\mathcal{L}(w, w_0, \lambda) = \frac{1}{2}w^T w - \sum_{i=1}^{N} \lambda_i \left[y_i \left(\frac{w^T x_i + w_0}{1 + g_i} \right) - 1 \right] \tag{43}$$

where λ_i, $i = 1, 2, , N$ are the Lagrangian multipliers associated with the constraint in (42). We need to find w, w_0, and λ by solving the Lagrangian duality: $max_{\lambda \geq 0} min_{w,w_0} \mathcal{L}(w, w_0, \lambda)$ [15, 16, 17, 18]. The Karush-Kuhn-Tucker conditions that $min_{w,w_0} \mathcal{L}(w, w_0, \lambda)$ has to satisfy are [15, 16, 17, 18]:

$$\begin{cases} \dfrac{\partial \mathcal{L}(w, w_0, \lambda)}{\partial w} = 0 \rightarrow w = \displaystyle\sum_{i=1}^{N} \frac{\lambda_i y_i x_i}{1 + g_i} & (44) \\ \dfrac{\partial \mathcal{L}(w, w_0, \lambda)}{\partial w_0} = 0 \rightarrow \displaystyle\sum_{i=1}^{N} \frac{\lambda_i y_i}{1 + g_i} = 0 & (45) \\ \lambda_i \geq 0 \quad , \quad i = 1, 2, ..., N & (46) \\ \lambda_i \left[y_i \left(\dfrac{w^T x_i + w_0}{1 + g_i} \right) - 1 \right] = 0 \quad , \quad i = 1, 2, ..., N \quad \text{(complementary slackness conditions)} & (47) \end{cases}$$

The conditions in (47) state that either λ_i or $y_i[(w^T x + w_0)/(1 + g_i)] - 1$ must be zero. Therefore support vectors are training samples where $|w^T x + w_0|/(1 + g_i) = 1$ (i.e. $y_i[(w^T x + w_0)/(1 + g_i)] - 1 = 0$ and $\lambda_i \neq 0$). It is now clear how our modified distance function in (36) affects the choice of support vectors. Before, support vectors were those geometrically closest to the hyperplane but now a trade-off between the weight (g_i) and the geometrical distance to the hyperplane determines whether a training sample is a support vector or not.

By expanding (43), we have:

$$\mathcal{L}(w, w_0, \lambda) = \frac{1}{2}w^T w - w^T \sum_{i=1}^{N} \frac{\lambda_i y_i x_i}{1 + g_i} - w_0 \sum_{i=1}^{N} \frac{\lambda_i y_i}{1 + g_i} + \sum_{i=1}^{N} \lambda_i$$

By replacing $\sum_{i=1}^{N} \frac{\lambda_i y_i}{1+g_i} = 0$ from (45) in the above equation, we get:

$$\mathcal{L}(w, w_0, \lambda) = \frac{1}{2}w^T w - w^T \sum_{i=1}^{N} \frac{\lambda_i y_i x_i}{1 + g_i} + \sum_{i=1}^{N} \lambda_i$$

By substituting w from (44), we have:

$$\mathcal{L}(w, w_0, \lambda) = \frac{1}{2} \left[\sum_{i=1}^{N} \frac{\lambda_i y_i x_i}{1 + g_i} \right]^T \left[\sum_{i=1}^{N} \frac{\lambda_i y_i x_i}{1 + g_i} \right] - \left[\sum_{i=1}^{N} \frac{\lambda_i y_i x_i}{1 + g_i} \right]^T \left[\sum_{i=1}^{N} \frac{\lambda_i y_i x_i}{1 + g_i} \right] + \sum_{i=1}^{N} \lambda_i$$

$$\mathcal{L}(w, w_0, \lambda) = -\frac{1}{2} \left[\sum_{i=1}^{N} \frac{\lambda_i y_i x_i}{1 + g_i} \right]^T \left[\sum_{i=1}^{N} \frac{\lambda_i y_i x_i}{1 + g_i} \right] + \sum_{i=1}^{N} \lambda_i$$

$$\mathcal{L}(w, w_0, \lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \frac{\lambda_i \lambda_j y_i y_j x_i^T x_j}{(1 + g_i)(1 + g_j)}$$

Now we maximize the above Lagrangian function with respect to λ:

$$\begin{cases} \max\limits_{\lambda} \left(\sum\limits_{i=1}^{N} \lambda_i - \frac{1}{2} \sum\limits_{i=1}^{N} \sum\limits_{j=1}^{N} \frac{\lambda_i \lambda_j y_i y_j x_i^T x_j}{(1+g_i)(1+g_j)} \right) & (48) \\ \text{subject to } \sum\limits_{i=1}^{N} \frac{\lambda_i y_i}{1+g_i} = 0 & (49) \\ \lambda_i \geq 0 \quad , \quad i = 1, 2, ..., N & (50) \end{cases}$$

Once the optimal Lagrangian multipliers (λ_i) have been computed, by maximizing (48), w is obtained by replacing them in (44) and w_0 is computed as an average value obtained using complementary slackness conditions in (47) for support vectors ($\lambda_i \neq 0$). Labeling a new sample is no different here; if $f(x) = w^T x + w_0 > 0$, x is classified in ω_1, and otherwise in ω_2.

An interesting observation is that the term $(1 + g_i)$ appears everywhere in the computations as a denominator of y_i. It means the weighted SVM can be obtained by replacing y_i with $y_i/(1 + g_i)$ in non-weighted SVM computations.

3.3.2. Two linearly nonseparable classes. If the two classes are not linearly separable which is usually the case in real-world problems, e.g. Figure 9, then it is not possible to find an empty band separating them. Each training sample will have one of the following constraints, as shown in Figure 9:

- it falls outside the band and is correctly classified, i.e. $y_i(w^T x_i + w_0) > 1$,
- it falls inside the band and is correctly classified, i.e. $0 \leq y_i(w^T x_i + w_0) \leq 1$, or
- it is misclassified, i.e. $y_i(w^T x_i + w_0) < 0$.

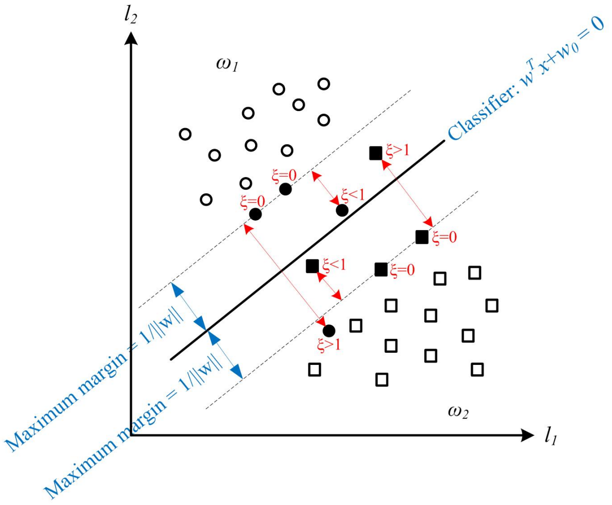

Figure 9. SVM classifier for two linearly nonseparable classes; black points show support vectors.

We can summarize the three above constraints in one by introducing the slack variable (ξ_i) [12]:

$$y_i(w^T x_i + w_0) \geq 1 - \xi_i \quad , \quad \begin{cases} \xi_i = 0 & \text{if } x_i \text{ is outside the band and correctly classified} \\ 0 < \xi_i \leq 1 & \text{if } x_i \text{ is inside the band and correctly classified} \\ \xi_i > 1 & \text{if } x_i \text{ is misclassified} \end{cases} \qquad (51)$$

The optimization task is now to maximize the margin (minimize the norm) while minimizing the slack variables [12]. The mathematical formulation for finding w and w_0 of the hyperplane follows:

$$\begin{cases} minimize J(w, w_0, \xi) = \frac{1}{2}||w||^2 + C\sum_{i=1}^{N}\xi_i = \frac{1}{2}w^T w + C\sum_{i=1}^{N}\xi_i & (52) \\ \text{subject to } y_i(w^T x_i + w_0) \geq 1 - \xi_i \quad , \quad i = 1, 2, ..., N & (53) \\ \xi_i \geq 0 \quad , \quad i = 1, 2, ..., N & (54) \end{cases}$$

The smoothing parameter C is a positive user-defined constant that controls the trade-off between the two competing terms in the cost function. The two terms are against each other because minimizing the norm (i.e. maximizing the margin) increases the slack variables by increasing the number of training samples inside the band. On the other hand, decreasing the number of samples inside the band is equivalent to decreasing the margin. Therefore, by choosing a very large $C \to \infty$, the width of the margin disappears, $2/||w|| \to 0$, because we allow the norm to grow much faster than slack variables (ξ_i). The corresponding Lagrangian function $\mathcal{L}(w, w_0, \xi, \lambda, \mu)$ for the above convex programming problem is defined as follows [15, 16, 17, 18]:

$$\mathcal{L}(w, w_0, \xi, \lambda, \mu) = \frac{1}{2}w^T w + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\mu_i\xi_i - \sum_{i=1}^{N}\lambda_i[y_i(w^T x_i + w_0) - 1 + \xi_i] \quad (55)$$

where λ_i, $i = 1, 2, , N$ are the Lagrangian multipliers associated with the constraint in (53) and μ_i, $i = 1, 2, , N$ are the Lagrangian multipliers associated with the constraint in (54). We need to find w, w_0, and λ by solving the Lagrangian duality: $max_{\lambda \geq 0} min_{w, w_0, \xi} \mathcal{L}(w, w_0, \xi, \lambda, \mu)$ [15, 16, 17, 18]. The KarushCKuhnCTucker conditions that $min_{w, w_0, \xi} \mathcal{L}(w, w_0, \xi, \lambda, \mu)$ has to satisfy are [15, 16, 17, 18]:

$$\begin{cases} \frac{\partial \mathcal{L}(w, w_0, \xi, \lambda, \mu)}{\partial w} = 0 \to w = \sum_{i=1}^{N}\lambda_i y_i x_i & (56) \\ \frac{\partial \mathcal{L}(w, w_0, \xi, \lambda, \mu)}{\partial w_0} = 0 \to \sum_{i=1}^{N}\lambda_i y_i = 0 & (57) \\ \frac{\partial \mathcal{L}(w, w_0, \xi, \lambda, \mu)}{\partial \xi_i} = 0 \to C - \mu_i - \lambda_i = 0 \quad , \quad i = 1, 2, ..., N & (58) \\ \mu_i \xi_i = 0 \quad , \quad i = 1, 2, ..., N & (59) \\ \mu_i \geq 0 \quad , \quad \lambda_i \geq 0 \quad , \quad i = 1, 2, ..., N & (60) \\ \lambda_i[y_i(w^T x_i + w_0) - 1 + \xi_i] = 0 \quad , \quad i = 1, 2, ..., N \quad \text{(complementary slackness conditions)} & (61) \end{cases}$$

Equations (56) and (57) depend only on training samples whose $\lambda_i \neq 0$, referred to as support vectors. On the other hand, the conditions in (61) state that either λ_i or $y_i(w^T x + w_0) - 1 + \xi_i$ must be zero. Therefore support vectors are training samples where $y_i(w^T x + w_0) = 1 - \xi_i$ (i.e. $y_i(w^T x + w_0) - 1 + \xi_i = 0$ and $\lambda_i \neq 0$). Therefore, correctly classified training samples outside the margin are not support vectors because we have $y_i(w^T x + w_0) > 1$ and $y_i(w^T x + w_0) - 1 + \xi_i$ cannot be zero considering $\xi_i \geq 0$. It means that support vectors are those on the edge of the margin ($\xi_i = 0$), correctly classified inside the margin ($0 < \xi_i < 1$), or misclassified ($\xi_i \geq 1$), as shown in Figure 9. From (58) and (59), we can see that $\lambda_i = C$ for support vectors falling inside the margin ($\xi_i > 0$) and $0 < \lambda_i < C$ for support vectors falling on the edge of the margin ($\xi_i = 0$). Therefore, (56) and (57) depend only on support vectors and consequently the hyperplane classifier is designed only based on support vectors and is independent of other training samples because their λ_i is zero.

By expanding (55), we have:

$$\mathcal{L}(w, w_0, \xi, \lambda, \mu) = \frac{1}{2}w^T w + \sum_{i=1}^{N}C\xi_i - \sum_{i=1}^{N}\mu_i\xi_i - w^T\sum_{i=1}^{N}\lambda_i y_i x_i - w_0\sum_{i=1}^{N}\lambda_i y_i + \sum_{i=1}^{N}\lambda_i - \sum_{i=1}^{N}\lambda_i\xi_i$$

$$\mathcal{L}(w, w_0, \xi, \lambda, \mu) = \frac{1}{2}w^T w + \sum_{i=1}^{N}(C - \mu_i - \lambda_i)\xi_i - w^T \sum_{i=1}^{N} \lambda_i y_i x_i - w_0 \sum_{i=1}^{N} \lambda_i y_i + \sum_{i=1}^{N} \lambda_i$$

By replacing $\sum_{i=1}^{N} \lambda_i y_i = 0$ from (57) and $C - \mu_i - \lambda_i = 0$ from (58), we get:

$$\mathcal{L}(w, w_0, \lambda) = \frac{1}{2}w^T w - w^T \sum_{i=1}^{N} \lambda_i y_i x_i + \sum_{i=1}^{N} \lambda_i$$

By substituting w from (56), we end up with:

$$\mathcal{L}(w, w_0, \lambda) = \frac{1}{2}\left[\sum_{i=1}^{N} \lambda_i y_i x_i\right]^T \left[\sum_{i=1}^{N} \lambda_i y_i x_i\right] - \left[\sum_{i=1}^{N} \lambda_i y_i x_i\right]^T \left[\sum_{i=1}^{N} \lambda_i y_i x_i\right] + \sum_{i=1}^{N} \lambda_i$$

$$\mathcal{L}(w, w_0, \lambda) = -\frac{1}{2}\left[\sum_{i=1}^{N} \lambda_i y_i x_i\right]^T \left[\sum_{i=1}^{N} \lambda_i y_i x_i\right] + \sum_{i=1}^{N} \lambda_i$$

$$\mathcal{L}(w, w_0, \lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j x_i^T x_j$$

Now we maximize the above Lagrangian function with respect to λ:

$$\left\{\begin{array}{lr} \max\limits_{\lambda} \left(\sum_{i=1}^{N} \lambda_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j x_i^T x_j\right) & (62) \\ \text{subject to } \sum_{i=1}^{N} \lambda_i y_i = 0 & (63) \\ 0 \le \lambda_i \le C \quad \text{due to Equation (58)}\ , \quad i = 1, 2, ..., N & (64) \end{array}\right.$$

Once the optimal Lagrangian multipliers (λ_i) have been computed, by maximizing (62), w is obtained by replacing them in (56) and w_0 is computed as an average value obtained using complementary slackness conditions in (61) for support vectors ($\lambda_i \neq 0$). However, ξ_i is also unknown in (61). We know from (58) and (59) that ξ_i is zero for training samples whose $\lambda_i < C$. Therefore, if we only use the training samples whose $0 < \lambda_i < C$ (support vectors falling on the edge of the margin) to find w_0 via (61), we can consider $\xi_i = 0$.

In the linearly nonseparable case the Lagrangian multipliers (λ_i) are bounded above by C, which is the only difference between the linearly separable and nonseparable cases. The slack variables, ξ_i, and their associated Lagrangian multipliers, μ_i, are not involved in finding the classifier hyperplane but their effect is indirectly felt through C [4].

The weighted version of SVM needs to be more sensitive to training samples with larger weights (g_i). In other words, the distance from training samples to the classifier hyperplane needs to be compromised based on their weights. However, the modified distance function in case of two nonseparable classes is different than separable classes. When the two classes are separable, we always move training samples toward the classifier hyperplane by a factor proportional to their weight because training samples are always on the correct side of the classifier hyperplane. On the other hand, in case of two nonseparable classes, a training sample might lie on the wrong side of the classifier hyperplane. Therefore, if a training sample lies on the correct side of the classifier hyperplane, we should move it toward the hyperplane and otherwise away from it by a factor proportional to its weight. This way we increase the sensitivity of the classifier to training samples with large weights. We introduce the modified distance function for weighted SVM, in case of two nonseparable classes, as:

$$\begin{cases} \frac{|f(x_i)|}{||w||} \times (\frac{1}{1+g_i}) = \frac{|f(x_i)|}{||w||} - \left(1 - \frac{1}{1+g_i}\right) \frac{|f(x_i)|}{||w||} & \text{if } x_i \text{ is correctly classified} \\ \frac{|f(x_i)|}{||w||} \times (1 + \frac{g_i}{1+g_i}) = \frac{|f(x_i)|}{||w||} + \left(1 - \frac{1}{1+g_i}\right) \frac{|f(x_i)|}{||w||} & \text{if } x_i \text{ is not correctly classified} \end{cases} \tag{65}$$

We define:

$$y_i = \begin{cases} +1 \; \forall x_i \in \omega_1 \\ -1 \quad \forall x_i \in \omega_2 \end{cases}$$

$$\rightarrow y_i \frac{f(x_i)}{|f(x_i)|} = \begin{cases} +1 \text{ if } x_i \text{ is correctly classified} \\ -1 \quad \text{if } x_i \text{ is not correctly classified} \end{cases} \tag{66}$$

Using (66), we can combine the two distance functions in (65) in one:

$$\frac{|f(x_i)|}{||w||} - \left(y_i \frac{f(x_i)}{|f(x_i)|}\right) \left(1 - \frac{1}{1+g_i}\right) \frac{|f(x_i)|}{||w||} = \frac{|f(x_i)| - y_i \left(1 - \frac{1}{1+g_i}\right) f(x_i)}{||w||} \quad , \quad \forall x_i \tag{67}$$

By scaling w and w_0, and introducing the slack variable (ξ_i) we can define the following constraint for training samples:

$$|f(x_i)| - y_i \left(1 - \frac{1}{1+g_i}\right) f(x_i) \geq 1 - \xi_i, \begin{cases} \xi_i = 0 & \text{if } x_i \text{ is outside the band and correctly classified} \\ 0 < \xi_i \leq 1 & \text{if } x_i \text{ is inside the band and correctly classified} \\ \xi_i > 1 & \text{if } x_i \text{ is misclassified} \end{cases} \tag{68}$$

The optimization task is now to maximize the margin (minimize the norm) while minimizing the slack variables (ξ_i). The mathematical formulation for finding w and w_0 of the hyperplane follows:

$$minimize J(w, w_0, \xi) = \frac{1}{2}||w||^2 + C \sum_{i=1}^{N} \xi_i = \frac{1}{2} w^T w + C \sum_{i=1}^{N} \xi_i \tag{69}$$

$$\text{subject to } |w^T x_i + w_0| - y_i \left(1 - \frac{1}{1+g_i}\right) (w^T x_i + w_0) \geq 1 - \xi_i \quad , \quad i = 1, 2, ..., N \tag{70}$$

$$\xi_i \geq 0 \quad , \quad i = 1, 2, ..., N \tag{71}$$

The corresponding Lagrangian function $\mathcal{L}(w, w_0, \xi, \lambda, \mu)$ for the above convex programming problem is defined as follows:

$$\begin{split} \mathcal{L}(w, w_0, \xi, \lambda, \mu) =& \frac{1}{2} w^T w + C \sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \mu_i \xi_i - \\ & \sum_{i=1}^{N} \lambda_i \left[|w^T x_i + w_0| - y_i \left(1 - \frac{1}{1+g_i}\right) (w^T x_i + w_0) - 1 + \xi_i \right] \end{split} \tag{72}$$

where $\lambda_i, i = 1, 2, , N$ are the Lagrangian multipliers associated with the constraint in (70) and $\mu_i, i = 1, 2, , N$ are the Lagrangian multipliers associated with the constraint in (71). We need to find w, w_0, and λ by solving the Lagrangian duality: $max_{\lambda \geq 0} min_{w, w_0, \xi} \mathcal{L}(w, w_0, \xi, \lambda, \mu)$ [15, 16, 17, 18]. The KarushCKuhnCTucker conditions that $min_{w, w_0, \xi} \mathcal{L}(w, w_0, \xi, \lambda, \mu)$ has to satisfy are [15, 16, 17, 18]:

$$\left\{\begin{array}{ll}
\dfrac{\partial\mathcal{L}(w,w_0,\xi,\lambda,\mu)}{\partial w}=0\rightarrow w=\displaystyle\sum_{i=1}^{N}\lambda_i x_i\left(\frac{|w^Tx_i+w_0|}{w^Tx_i+w_0}-y_i\left(1-\frac{1}{1+g_i}\right)\right) & (73)\\
\dfrac{\partial\mathcal{L}(w,w_0,\xi,\lambda,\mu)}{\partial w_0}=0\rightarrow \displaystyle\sum_{i=1}^{N}\lambda_i\left(\frac{|w^Tx_i+w_0|}{w^Tx_i+w_0}-y_i\left(1-\frac{1}{1+g_i}\right)\right)=0 & (74)\\
\dfrac{\partial\mathcal{L}(w,w_0,\xi,\lambda,\mu)}{\partial \xi_i}=0\rightarrow C-\mu_i-\lambda_i=0 \quad , \quad i=1,2,...,N & (75)\\
\mu_i\xi_i=0 \quad , \quad i=1,2,...,N & (76)\\
\mu_i\geq 0 \quad , \quad \lambda_i\geq 0 \quad , \quad i=1,2,...,N & (77)\\
\lambda_i\left[|w^Tx_i+w_0|-y_i\left(1-\dfrac{1}{1+g_i}\right)(w^Tx_i+w_0)-1+\xi_i\right]=0, \quad i=1,2,...,N \quad \text{(co. slack. con.)} & (78)
\end{array}\right.$$

By expanding (72), we have:

$$\mathcal{L}(w,w_0,\xi,\lambda,\mu)=\frac{1}{2}w^Tw+\sum_{i=1}^{N}C\xi_i-\sum_{i=1}^{N}\mu_i\xi_i- \\ \sum_{i=1}^{N}\lambda_i\left[|w^Tx_i+w_0|-y_i\left(1-\frac{1}{1+g_i}\right)(w^Tx_i+w_0)\right]+\sum_{i=1}^{N}\lambda_i-\sum_{i=1}^{N}\lambda_i\xi_i$$

$$\mathcal{L}(w,w_0,\xi,\lambda,\mu)=\frac{1}{2}w^Tw+\sum_{i=1}^{N}(C-\mu_i-\lambda_i)\xi_i- \\ \sum_{i=1}^{N}(w^Tx_i+w_0)\lambda_i\left[\frac{|w^Tx_i+w_0|}{w^Tx_i+w_0}-y_i\left(1-\frac{1}{1+g_i}\right)\right]+\sum_{i=1}^{N}\lambda_i$$

By replacing $C-\mu_i-\lambda_i=0$ from (75), we get:

$$\mathcal{L}(w,w_0,\lambda)=\frac{1}{2}w^Tw-\sum_{i=1}^{N}(w^Tx_i+w_0)\lambda_i\left[\frac{|w^Tx_i+w_0|}{w^Tx_i+w_0}-y_i\left(1-\frac{1}{1+g_i}\right)\right]+\sum_{i=1}^{N}\lambda_i$$

$$\mathcal{L}(w,w_0,\lambda)=\frac{1}{2}w^Tw-w^T\sum_{i=1}^{N}\lambda_i x_i\left[\frac{|w^Tx_i+w_0|}{w^Tx_i+w_0}-y_i\left(1-\frac{1}{1+g_i}\right)\right]+ \\ w_0\sum_{i=1}^{N}\lambda_i\left[\frac{|w^Tx_i+w_0|}{w^Tx_i+w_0}-y_i\left(1-\frac{1}{1+g_i}\right)\right]+\sum_{i=1}^{N}\lambda_i$$

By replacing $\sum_{i=1}^{N}\lambda_i\left(\frac{|w^Tx_i+w_0|}{w^Tx_i+w_0}-y_i\left(1-\frac{1}{1+g_i}\right)\right)=0$ from (74), we have:

$$\mathcal{L}(w,w_0,\lambda)=\frac{1}{2}w^Tw-w^T\sum_{i=1}^{N}\lambda_i x_i\left[\frac{|w^Tx_i+w_0|}{w^Tx_i+w_0}-y_i\left(1-\frac{1}{1+g_i}\right)\right]+\sum_{i=1}^{N}\lambda_i$$

$$\mathcal{L}(w,w_0,\lambda)=\sum_{i=1}^{N}\lambda_i+w^T\left(\frac{1}{2}w-\sum_{i=1}^{N}\lambda_i x_i\left[\frac{|w^Tx_i+w_0|}{w^Tx_i+w_0}-y_i\left(1-\frac{1}{1+g_i}\right)\right]\right)$$

By substituting w from (73), we end up with:

$$\mathcal{L}(w,w_0,\lambda)=\sum_{i=1}^{N}\lambda_i+\left[\sum_{i=1}^{N}\lambda_i x_i\left(\frac{|w^Tx_i+w_0|}{w^Tx_i+w_0}-y_i\left(1-\frac{1}{1+g_i}\right)\right)\right]^T$$
$$\left(\frac{1}{2}\left[\sum_{i=1}^{N}\lambda_i x_i\left(\frac{|w^Tx_i+w_0|}{w^Tx_i+w_0}-y_i\left(1-\frac{1}{1+g_i}\right)\right)\right]-\left[\sum_{i=1}^{N}\lambda_i x_i\left(\frac{|w^Tx_i+w_0|}{w^Tx_i+w_0}-y_i\left(1-\frac{1}{1+g_i}\right)\right)\right]\right)$$

$$\mathcal{L}(w,w_0,\lambda)=\sum_{i=1}^{N}\lambda_i-\frac{1}{2}\left[\sum_{i=1}^{N}\lambda_i x_i\left(\frac{|w^Tx_i+w_0|}{w^Tx_i+w_0}-y_i\left(1-\frac{1}{1+g_i}\right)\right)\right]^T$$
$$\left[\sum_{i=1}^{N}\lambda_i x_i\left(\frac{|w^Tx_i+w_0|}{w^Tx_i+w_0}-y_i\left(1-\frac{1}{1+g_i}\right)\right)\right]$$

Now we maximize the above Lagrangian function with respect to λ:

$$\left\{\begin{array}{ll}
\max\limits_{\lambda}\sum_{i=1}^{N}\lambda_i-\frac{1}{2}\left[\sum_{i=1}^{N}\lambda_i x_i\left(\frac{|w^Tx_i+w_0|}{w^Tx_i+w_0}-y_i\left(1-\frac{1}{1+g_i}\right)\right)\right]^T\times H(w,w_0,\lambda,x) & (79)\\
\text{subject to }\sum_{i=1}^{N}\lambda_i\left(\frac{|w^Tx_i+w_0|}{w^Tx_i+w_0}-y_i\left(1-\frac{1}{1+g_i}\right)\right)=0 & (80)\\
0\le\lambda_i\le C\quad\text{due to Equation (75)},\quad i=1,2,...,N & (81)
\end{array}\right.$$

where $H(w,w_0,\lambda,x)=\sum_{i=1}^{N}\lambda_i x_i\left(\frac{|w^Tx_i+w_0|}{w^Tx_i+w_0}-y_i\left(1-\frac{1}{1+g_i}\right)\right)$.

Once the optimal Lagrangian multipliers (λ_i) have been computed, by maximizing (79), w is obtained by replacing them in (73) and w_0 is computed as an average value obtained using complementary slackness conditions in (78) for support vectors whose $0<\lambda_i<C$ and considering $\xi_i=0$. Labeling a new sample is no different here; if $f(x)=w^Tx+w_0>0$, x is classified in ω_1, and otherwise in ω_2.

Maximizing the above Lagrangian function for weighted SVM with respect to λ is not as easy as (62) because this time w and w_0 are involved in the process of finding the Lagrangian multipliers (λ_i) in (79) while they are unknown. The appearance of w and w_0 in (79) originates from the dichotomy in the distance function in (65). Therefore, an iterative optimization technique must be adopted:

- w and w_0 are initialized for a non-weighted SVM,
- Loop: repeat until convergence
  - λ_i are calculated using (79)
  - w and w_0 are calculated using (73) and (78)

The time complexity of the above algorithm is k times more than the time complexity of finding the non-weighted SVM classifier hyperplane ($O(N^3)$ with a naive implementation of a quadratic programming solver [4]), where k is the number of iterations in the loop. Since w and w_0 are initialized using a non-weighted SVM, rather than randomly, the convergence is expected to happen in a few iterations.

From a geometric point of view, the loop in the above algorithm updates the classifier hyperplane by redefining the distance of training samples to the hyperplane in each iteration. This is equivalent to relocating the training samples after each iteration with respect to the hyperplane classifier based on their weights and updating the

Table 2. Training samples and their weights for SVM.

l_1	l_2	Class	Weight
1	1	ω_1	4
1	2	ω_1	2
1.4	1.5	ω_1	2
2	1	ω_2	1
2	2	ω_2	1

classifier hyperplane based on the relocated training samples. Therefore, the following algorithm offers an alternative but geometrically equivalent approach to the above algorithm with the same time complexity. The following algorithm can take advantage of existing software and libraries for non-weighted SVM to develop the weighted SVM.

- w and w_0 are initialized for a non-weighted SVM,
- Loop: repeat until convergence
  - $$\hat{X}_t = X - \left(1 - \frac{1}{1+g}\right) . \left(\frac{|Xw_{t-1} + w_{0_{t-1}}|}{||w_{t-1}||}\right) .y \frac{w_{t-1}^T}{||w_{t-1}||} \tag{82}$$
  - find w_t and w_{0_t} for the non-weighted SVM classifier hyperplane based on $\hat{X}_t$

where the subscript t stands for the iterator inside the loop. At the first step in the loop, X is the input feature matrix (each row representing one training sample), y is a column vector containing the responses, w is a column vector representing the norm of the classifier hyperplane, w_0 is the intercept of the classifier hyperplane, and g is a column vector containing the training samples' weights. The dot shows array (or element-wise) operations versus matrix operations shown with a cross. In (82), $\left(1 - \frac{1}{1+g}\right) . \left(\frac{|Xw+w_0|}{||w||}\right)$ is the magnitude by which we have to move the training samples, and $\left(-y \frac{w^T}{||w||}\right)$ is the movement direction. The movement magnitude is proportional to the training sample's weight. The movement is in the direction of the classifier's vector (w) for training samples in class ω_2 (y=-1) and in the opposite direction of w for training samples in class ω_1 (y=1). In other words, we have to update the position of a training sample by moving it $\left(1 - \frac{1}{1+g_i}\right) . \left(\frac{|x_i w+w_0|}{||w||}\right)$ toward the classifier hyperplane if it is correctly classified or the same amount away from the hyperplane if it is wrongly classified. Therefore, training samples with large weights which were not normally selected as support vectors, now have a higher chance of being selected as support vectors if the aforementioned shift could drop them inside the margin.

3.3.3. Experiment. Due to SVM's stability to changes in a small part of the training data, the dataset in Table 1 cannot differentiate between the non-weighted and weighted SVM. In other words, the two classifiers are the same for that dataset. Instead, we use the dataset in Table 2 to show the effect of embedding training samples' weights in SVM. Figure 10 shows the division of the feature space between the two classes with and without considering the training samples' weights in computing the linear classifier. In weighted SVM, the important samples in class ω_1 will move toward class ω_2, through (82), and repel the border toward class ω_2.

4. Nonlinear predictors

4.1. Decision trees

Ordinary binary decision trees (OBDTs) split the feature space into hyperrectangles with sides parallel to the axes [19]. Nodes in an OBDT, shown in Figure 11, are binary questions whose answers are either yes or no and the answer to these questions determines the path to a leaf which is equivalent to a response (nominal label in

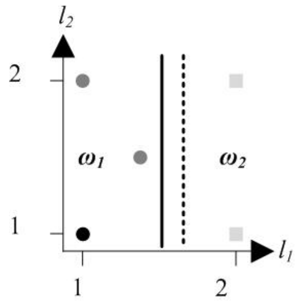

Figure 10. Division of the feature space between the two classes, circles and squares, without (solid line) and with (dashed line) considering the training samples' weights (darkness of samples) in SVM classifier (C=1).

classification or numerical estimate in regression). Questions at nodes are of the form is $x_k \leq \alpha$? where x_k is the k-th feature and α is a threshold. To predict the response of an irresponsive sample, one needs to answer the question at each node and traverse to the left or right node based on the answer until a leaf (response) is reached.

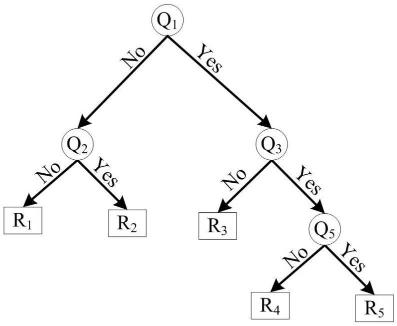

Figure 11. Ordinary binary decision trees; Q stands for question and R stands for response.

The training process involves designing the questions, structuring the tree, and associating each leaf with a response. Each node splits the training dataset into two disjoint groups, each corresponding to one of the answers: yes or no. Many questions can be asked at each node based on what feature (x_k) to choose and what threshold (α) to use. Different thresholds that can be considered for a specific feature at a node are determined based on the training samples at that node. For example, if there are N samples at a node, there could be N-1 different thresholds, each taken halfway between consecutive distinct values of x_k among the training samples at that node. Therefore, if there are l features and N training samples at a node, $(N-1) \times l$ different questions can be asked. The best question to ask at a node is the one which maximizes the impurity decrease (ΔI). The impurity decrease is calculated through (83) [19]:

$$\Delta I = I - \frac{N_Y}{N} I_Y - \frac{N_N}{N} I_N \tag{83}$$

where I is the impurity of the ancestor node, N is the number of training samples in the ancestor node, N_Y is the number of training samples in the descendant node corresponding with the answer yes to the question, N_N is the number of training samples in the descendant node corresponding with the answer no to the question, and I_Y and I_N are the impurities of the descendent nodes. Entropy of training samples at a node, in (84), is a common definition of node impurity in classification tasks ($I_{classification}$) [19], where N is the number of training samples at this node, M is the number of classes, and $N(\omega_i)$ is the number of training samples from class ω_i at this node. Therefore, in classification, impurity at a node is proportional to the heterogeneity of classes among training samples at that node. The largest impurity ($\log_2 M$) happens when training samples are equally distributed among classes and the least impurity (0) happens when all training samples belong to the same class.

The impurity of a node in regression tasks ($I_{regression}$) is commonly calculated as the variance, in (85), where y_i is the response of the i-th training sample at this node and $\bar{y}$ is the average of responses at this node.

$$I_{classification} = -\sum_{i=1}^{M} \frac{N(\omega_i)}{N} \log_2 \frac{N(\omega_i)}{N} \tag{84}$$

$$I_{regression} = \frac{\sum_{i=1}^{N} (y_i - \bar{y})^2}{N} \tag{85}$$

A node is considered a leaf if the maximum impurity decrease (ΔI_{max}) for that node is less than a user-defined threshold or it contains only a few training samples, although other alternative conditions have been used in the literature [19, 20]. The majority rule in case of classification or the average rule in case of regression is commonly used to determine the response at that leaf [19].

In the weighted version of OBDT, the impurity decrease (ΔI) and impurity (I) are calculated through the following equations:

$$\Delta I = I - \frac{\sum g_Y}{\sum g} I_Y - \frac{\sum g_N}{\sum g} I_N \tag{86}$$

$$I_{classification} = -\sum_{i=1}^{M} \frac{g(\omega_i)}{\sum g} \log_2 \frac{g(\omega_i)}{\sum g} \tag{87}$$

$$I_{regression} = \frac{\sum_{i=1}^{N} g_i (y_i - \bar{y})^2}{\sum_{i=1}^{N} g_i} \tag{88}$$

where $\sum g_Y$ and $\sum g_N$ are the sum of the weight of training samples corresponding to the answers yes and no, respectively, $\sum g$ is the sum of the weight of all training samples at the ancestor node, $g(\omega_i)$ is the sum of the weight of training samples belonging to class ω_i, and g_i is the i-th training sample's weight.

A node is considered a leaf if the maximum impurity decrease (ΔI_{max}) for that node is less than a user-defined threshold or the total weight of training samples inside it, is too small. In case of classification, the class with the largest total weight ($argmax_{\omega_j} \sum_{i \in \omega_j} g_i$) is associated with that leaf. In case of regression, the weighted average of the responses ($\sum_{i \in leaf} g_i y_i / \sum_{i \in leaf} g_i$) is associated with that leaf.

In the weighted decision tree, samples with larger weights play a more important role in deciding what question to ask at a node (by playing a more significant role in calculating impurity and impurity decrease), when to stop splitting the nodes, and what response to associate with a leaf.

4.1.1. Experiment. Here we use the dataset in Table 1 to show the effect of embedding training samples' weights in decision tree. Figure 12 shows the division of the feature space between the two classes with and without considering the training samples' weights in developing the decision tree. The important samples from class ω_1 change the way the weighted decision tree divides the feature space between the two classes in comparison with the non-weighted decision tree.

4.2. Multilayer perceptron (MLP)

In the backpropagation algorithm [21, 22, 23], the architecture of the network is fixed and its synaptic weights are computed so as to minimize a cost function defined as:

$$J(\boldsymbol{w}) = \sum_{i=1}^{N} \epsilon(i) \tag{89}$$

where N is the number of training samples and $\epsilon(i)$ is a function of the network's output ($\hat{y}(i)$) and the desired output ($y(i)$) for the i-th training sample. A common choice for $\epsilon(i)$ is the sum of squared errors in the output nodes [24, 25, 22, 23]:

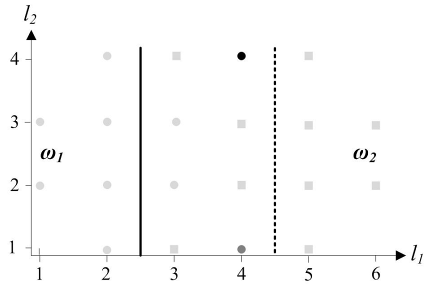

Figure 12. Division of the feature space between the two classes, circles and squares, without (solid line) and with (dashed line) considering the training samples' weights (darkness of samples) in decision tree classifier (minimum impurity decrease for splitting a node is considered 0.1).

$$\epsilon(i) = \frac{1}{2}\sum_{j=1}^{k_L}\left(\hat{y}_j^L(i) - y_j^L(i)\right)^2 \quad , \quad i = 1, 2, ..., N \tag{90}$$

where L refers to the output layer, k_L represents the number of nodes in the output layer, $\hat{y}_j^L(i)$ represents the output of the j-th node in the output layer, and $y_j^L(i)$ represents its corresponding desired value. We also have the following equation for calculating the output of the j-th node at the r-th layer for the i-th training sample ($\hat{y}_j^r(i)$):

$$\hat{y}_j^r(i) = f_j^r\left(\nu_j^r(i)\right) \tag{91}$$

$$\nu_j^r(i) = \sum_{k=1}^{k_{r-1}} w_{jk}^r \hat{y}_k^{r-1}(i) \tag{92}$$

where f_j^r is the activation function at the j-th node of the r-th layer, k_{r-1} is the number of nodes at the (r-1)-th layer, $\hat{y}_k^{r-1}(i)$ is the output of the k-th node in the (r-1)-th layer, and w_{jk}^r is the synaptic weight from the k-th node at the (r-1)-th layer to the j-th node at the r-th layer.

We can iteratively find the synaptic weight vectors that minimize the perceptron cost function using the gradient descent scheme [21, 22, 23]. In each iteration, the weight vector (including the threshold) of the j-th node in the r-th layer ($\boldsymbol{w}_j^r$) is modified through (93):

$$\boldsymbol{w}_j^r(new) = \boldsymbol{w}_j^r(old) + \Delta\boldsymbol{w}_j^r \tag{93}$$

The modification term in (93) ($\Delta\boldsymbol{w}_j^r$) is computed through (94) according to the gradient descent scheme:

$$\Delta\boldsymbol{w}_j^r = -\alpha\frac{\partial J(\boldsymbol{w})}{\partial\boldsymbol{w}_j^r} \tag{94}$$

By substituting the cost function from (89) in (94) and applying the chain rule in differentiation, we obtain:

$$\Delta\boldsymbol{w}_j^r = -\alpha\frac{\partial\sum_{i=1}^N \epsilon(i)}{\partial\boldsymbol{w}_j^r} = -\alpha\sum_{i=1}^N\frac{\partial\epsilon(i)}{\partial\boldsymbol{w}_j^r} = -\alpha\sum_{i=1}^N\frac{\partial\epsilon(i)}{\partial\nu_j^r(i)}\frac{\partial\nu_j^r(i)}{\partial\boldsymbol{w}_j^r} \tag{95}$$

By defining $\delta_j^r(i) = \frac{\partial\epsilon(i)}{\partial\nu_j^r(i)}$ in the above equation, we obtain:

$$\Delta\boldsymbol{w}_j^r = -\alpha\sum_{i=1}^N\delta_j^r(i)\frac{\partial\nu_j^r(i)}{\partial\boldsymbol{w}_j^r} \tag{96}$$

We can calculate $\frac{\partial \nu_j^r(i)}{\partial \boldsymbol{w}_j^r}$ using (92) as follows:

$$\frac{\partial \nu_j^r(i)}{\partial \boldsymbol{w}_j^r} = \begin{bmatrix} \frac{\partial \nu_j^r(i)}{\partial w_{j1}^r} \\ \cdot \\ \cdot \\ \frac{\partial \nu_j^r(i)}{\partial w_{jk_{r-1}}^r} \end{bmatrix} = \hat{y}^{r-1}(i) \tag{97}$$

where k_{r-1} is the number of nodes in the (r-1)-th layer and $\hat{y}^{r-1}(i)$ is the output vector of the (r-1)-th layer for the i-th training sample. By substituting (97) in (96) we obtain:

$$\Delta \boldsymbol{w}_j^r = -\alpha \sum_{i=1}^{N} \delta_j^r(i) \hat{y}^{r-1}(i) \tag{98}$$

The above equation obtains the correction term for batch mode [26]. In online or pattern mode, instead of summing up the corrections over all training samples and updating the weights at once, the weights are updated once for each individual training sample before moving on to the next [26]. In stochastic mode, the gradient at each iteration is calculated based on a random subset of training samples [27].

Now we have to compute $\delta_j^r(i)$ based on the definition of the cost function given in (90). First we calculate this term for the output layer ($r = L$):

$$\delta_j^L(i) = \frac{\partial \epsilon(i)}{\partial \nu_j^L(i)} \tag{99}$$

By substituting (90) and (91) in the above equation we get:

$$\delta_j^L(i) = \frac{\partial}{\partial \nu_j^L(i)} \left[\frac{1}{2} \sum_{m=1}^{k_L} \left(f_m^L(\nu_m^L(i)) - y_m^L(i) \right)^2 \right] \tag{100}$$

By keeping only the terms that are dependent on $\nu_j^L(i)$ we get:

$$\delta_j^L(i) = \frac{\partial}{\partial \nu_j^L(i)} \left[\frac{1}{2} \left(f_j^L(\nu_j^L(i)) - y_j^L(i) \right)^2 \right] = \left(\hat{y}_j^L(i) - y_j^L(i) \right) \frac{\partial f_j^L(\nu_j^L(i))}{\partial \nu_j^L(i)} \tag{101}$$

where $\hat{y}_j^L(i)$ is the output of the j-th node in the output layer for the i-th training sample, $y_j^L(i)$ is its corresponding desired value, and f_j^L is the activation function of the j-th node in the output layer which takes $\nu_j^L(i)$ as input.

Now we compute $\delta_j^r(i)$ for hidden layers ($r < L$):

$$\delta_j^r(i) = \frac{\partial \epsilon(i)}{\partial \nu_j^r(i)} = \sum_{k=1}^{k_{r+1}} \frac{\partial \epsilon(i)}{\partial \nu_k^{r+1}(i)} \frac{\partial \nu_k^{r+1}(i)}{\partial \nu_j^r(i)} = \sum_{k=1}^{k_{r+1}} \delta_k^{r+1}(i) \frac{\partial \nu_k^{r+1}(i)}{\partial \nu_j^r(i)} \tag{102}$$

We use (92) to calculate:

$$\frac{\partial \nu_k^{r+1}(i)}{\partial \nu_j^r(i)} = \frac{\partial}{\partial \nu_j^r(i)} \left[\sum_{m=1}^{k_r} w_{km}^{r+1} \hat{y}_m^r(i) \right] \tag{103}$$

Replacing $\hat{y}_m^r(i)$ with $f_m^r\left(\nu_m^r(i)\right)$ based on (91), we get:

$$\frac{\partial \nu_k^{r+1}(i)}{\partial \nu_j^r(i)} = \frac{\partial}{\partial \nu_j^r(i)} \left[\sum_{m=1}^{k_r} w_{km}^{r+1} f_m^r\left(\nu_m^r(i)\right) \right] \tag{104}$$

By keeping only the terms that are dependent on $\nu_j^r(i)$ we get:

$$\frac{\partial \nu_k^{r+1}(i)}{\partial \nu_j^r(i)} = \frac{\partial}{\partial \nu_j^r(i)} \left[w_{kj}^{r+1} f_j^r \left(\nu_j^r(i) \right) \right] = w_{kj}^{r+1} \frac{\partial f_j^r \left(\nu_j^r(i) \right)}{\partial \nu_j^r(i)} \tag{105}$$

Replacing the above equation in (102), we obtain:

$$\delta_j^r(i) = \sum_{k=1}^{k_{r+1}} \delta_k^{r+1}(i) w_{kj}^{r+1} \frac{\partial f_j^r \left(\nu_j^r(i) \right)}{\partial \nu_j^r(i)} \tag{106}$$

where k_{r+1} is the number of nodes in the (r+1)-th layer, w_{kj}^{r+1} is the synaptic weight from the j-th node in the r-th layer to the k-th node in the (r+1)-th layer, and f_j^r is the activation function of the j-th node in the r-th layer which takes $\nu_j^r(i)$ as input.

To develop the weighted version of MLP, we include each training sample's weight (g_i) in the cost function to adjust the MLP cost based on the importance of training samples. The MLP classifier will no longer be equally fair to all training samples. We modify the MLP cost function as in (107) to punish the classifier more for misclassifying training samples with larger weights and less for training samples with smaller weights.

$$J(\boldsymbol{w}) = \sum_{i=1}^{N} g_i \epsilon(i) \tag{107}$$

We can compute the modification term ($\Delta \boldsymbol{w}_j^r$) through gradient descent scheme as follows:

$$\Delta \boldsymbol{w}_j^r = -\alpha \frac{\partial J(\boldsymbol{w})}{\partial \boldsymbol{w}_j^r} = -\alpha \frac{\partial \sum_{i=1}^{N} g_i \epsilon(i)}{\partial \boldsymbol{w}_j^r} = -\alpha \sum_{i=1}^{N} g_i \frac{\partial \epsilon(i)}{\partial \boldsymbol{w}_j^r} \tag{108}$$

By applying the chain rule in differentiation:

$$\Delta \boldsymbol{w}_j^r = -\alpha \sum_{i=1}^{N} g_i \frac{\partial \epsilon(i)}{\partial \nu_j^r(i)} \frac{\partial \nu_j^r(i)}{\partial \boldsymbol{w}_j^r} \tag{109}$$

By replacing $\frac{\partial \epsilon(i)}{\partial \nu_j^r(i)}$ with $\delta_j^r(i)$, we obtain:

$$\Delta \boldsymbol{w}_j^r = -\alpha \sum_{i=1}^{N} g_i \delta_j^r(i) \frac{\partial \nu_j^r(i)}{\partial \boldsymbol{w}_j^r} \tag{110}$$

By applying (97), we obtain:

$$\Delta \boldsymbol{w}_j^r = -\alpha \sum_{i=1}^{N} g_i \delta_j^r(i) \hat{y}^{r-1}(i) \tag{111}$$

Therefore, the only difference between the above equation for computing the correction term and (98) is the presence of the training samples' weights (g_i) in the summand. If we define $\tilde{\alpha}(i) = \alpha g_i$, we obtain:

$$\Delta \boldsymbol{w}_j^r = -\sum_{i=1}^{N} \tilde{\alpha}(i) \delta_j^r(i) \hat{y}^{r-1}(i) \tag{112}$$

The above equation shows that the weighted version of the backpropagation algorithm for MLP is obtained by defining the training rate as $\tilde{\alpha}(i) = \alpha g_i$, which means a different training rate for each training sample based on its weight; remembering that the cost must also be calculated through (107) which includes different weights for training samples. Adjusting the training rate based on the weight of training samples and including the weights in the cost function bias the trained MLP in favor of training samples with larger weights.

4.2.1. Experiment. Here we use the dataset in Table 1 to show the effect of embedding training samples' weights in the cost function (107) and backpropagation algorithm (111). The MLP is designed with one hidden layer including two nodes. Including more hidden nodes will result in all training samples being correctly classified in both non-weighted and weighted MLP, a zero classification cost for both non-weighted (89) and weighted (107) MLP, and consequently similar classifiers. With two hidden nodes some training samples cannot be correctly classified, so we can see the difference between non-weighted and weighted MLP classifiers. Figure 13 shows the division of the feature space between the two classes with and without considering the training samples' weights in the cost function and the backpropagation algorithm. Despite both weighted and non-weighted MLP misclassify the same four training samples, the weighted MLP classifier provides a better fit (a lower error) for the two more important samples from class ω_1.

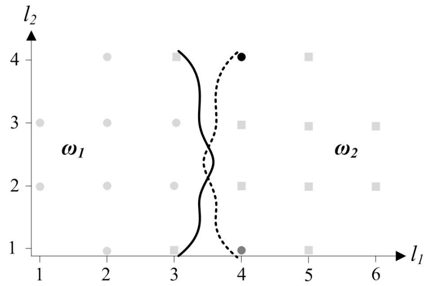

Figure 13. Division of the feature space between the two classes, circles and squares, without (solid line) and with (dashed line) considering the training samples' weights (darkness of samples) in MLP classifier (logistic activation function and adaptive training rate with 2000 iterations).

4.3. Nonlinear SVM

In nonlinear SVM [28], training samples are nonlinearly mapped from their original l-dimensional space (where they cannot be linearly separated) into a k-dimensional space ($k >> l$) where they are more likely to be linearly separable [29, 4]. However, there is no guarantee that training samples will be linearly separable in the new k-dimensional space. Therefore, linear SVM with slack variables is used to find the hyperplane separating the two classes in the k-dimensional space. Although the classifier is a hyperplane in the k-dimensional space, it is a hypersurface in the l-dimensional space due to the nonlinear mapping, hence the name nonlinear SVM. The next step is to find the dimensionality of the new space (k) and the mapping function. We use the following equations, obtained in Section 3.3.2, to find the SVM classifier hyperplane $f(\tilde{x}) = w^T\tilde{x} + w_0$ in the k-dimensional space, where $\tilde{x}_i$ is the i-th feature vector (x_i) mapped into the k-dimensional space.

$$\begin{cases} \max\limits_{\lambda} \left(\sum\limits_{i=1}^{N} \lambda_i - \frac{1}{2} \sum\limits_{i=1}^{N} \sum\limits_{j=1}^{N} \lambda_i \lambda_j y_i y_j \tilde{x}_i^T \tilde{x}_j \right) & (113) \\ \text{subject to } \sum\limits_{i=1}^{N} \lambda_i y_i = 0 & (114) \\ 0 \leq \lambda_i \leq C \quad \text{due to Equation (58)}, \quad i = 1, 2, ..., N & (115) \end{cases}$$

$$\begin{cases} w = \sum\limits_{i=1}^{N} \lambda_i y_i \tilde{x}_i & (116) \\ \lambda_i [y_i (w^T \tilde{x}_i + w_0) - 1 + \xi_i] = 0 \quad , \quad i = 1, 2, ..., N \quad \text{(complementary slackness conditions)} & (117) \end{cases}$$

By substituting w from (116) in (117) as well as in the hyperplane $f(\tilde{x}) = w^T\tilde{x} + w_0$ we end up with:

$$\begin{cases} f(\tilde{x}) = \left(\sum_{i=1}^{N} \lambda_i y_i \tilde{x}_i\right)^T \tilde{x} + w_0 = \sum_{i=1}^{N} \lambda_i y_i \left(\tilde{x}_i^T \tilde{x}\right) + w_0 & (118) \\ \lambda_i \left[y_i \left(\sum_{j=1}^{N} \lambda_j y_j (\tilde{x}_j^T \tilde{x}_i) + w_0 \right) - 1 + \xi_i \right] = 0, \quad i = 1, 2, ..., N (\text{complementary slackness cond.}) & (119) \end{cases}$$

An elegant property of the SVM helps to implicitly map the training samples into the k-dimensional space without knowing the mapping function and k. Notice that training samples enter into (113), (118), and (119) in pairs, in the form of inner products $(\tilde{x}_i^T \tilde{x}_j)$ in the k-dimensional space. Therefore, for finding w and w_0 of the hyperplane in the k-dimensional space and even for classifying a new sample using (118), only the inner product of pairs of feature vectors in the k-dimensional space is required. Knowing the mapping function and the dimensionality of the new space (k) is not necessary. We can use the kernel trick to find the inner product of two feature vectors in the k-dimensional space without actually mapping them from the l-dimensional space into the k-dimensional space. According to Mercer's theorem, for any kernel (K), there exists a space in which $K(x_i, x_j) = \tilde{x}_i^T \tilde{x}_j$ [30, 31, 32]. Equations (120), (121), and (122) are examples of kernel functions (31) called polynomial, radial basis function, and hyperbolic tangent, respectively, where σ is the kernel's bandwidth.

$$K(x_i, x_j) = (x_i^T x_j + 1)^q \quad , \quad q > 0 \tag{120}$$

$$K(x_i, x_j) = \exp\left(-\frac{||x_i - x_j||^2}{\sigma^2}\right) \tag{121}$$

$$K(x_i, x_j) = \tanh(\beta x_i^T x_j + \gamma) \quad \text{, for appropriate values of } \beta \text{ and } \gamma\text{, e.g. } \beta = 2 \text{ and } \gamma = 1 \tag{122}$$

Therefore, to convert the linear SVM to nonlinear SVM we just need to replace the inner product of the mapped feature vectors $(\tilde{x}_i^T \tilde{x}_j)$ by a kernel function of the original feature vectors $K(x_i, x_j)$:

$$\begin{cases} \max_{\lambda} \left(\sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j y_i y_j K(x_i, x_j) \right) & (123) \\ \text{subject to } \sum_{i=1}^{N} \lambda_i y_i = 0 & (124) \\ 0 \leq \lambda_i \leq C \quad \text{due to Equation (58)}\ , \quad i = 1, 2, ..., N & (125) \end{cases}$$

$$\begin{cases} f(x) = \sum_{i=1}^{N} \lambda_i y_i K(x_i, x) + w_0 & (126) \\ \lambda_i \left[y_i \left(\sum_{j=1}^{N} \lambda_j y_j K(x_j, x) + w_0 \right) - 1 + \xi_i \right] = 0 \quad , \quad i = 1, 2, ..., N \quad (\text{complem. slackness cond.}) & (127) \end{cases}$$

Although $f(x)$ is linear in the k-dimensional space, it is nonlinear in the l-dimensional space due to the nonlinearity of the kernel function.

Here we explain why we cannot develop the weighted version of nonlinear SVM. We use the following equations, obtained in Section 3.3.2, to find the weighted SVM classifier hyperplane $f(\tilde{x}) = w^T \tilde{x} + w_0$ in the k-dimensional space, where $\tilde{x}_i$ is the i-th feature vector (x_i) mapped into the k-dimensional space.

$$\begin{cases} \max\limits_{\lambda} \sum\limits_{i=1}^{N} \lambda_i - \frac{1}{2} \left[\sum\limits_{i=1}^{N} \lambda_i \tilde{x}_i \left(\frac{|w^T \tilde{x}_i + w_0|}{w^T \tilde{x}_i + w_0} - y_i \left(1 - \frac{1}{1+g_i} \right) \right) \right]^T \times T(w, w_0, \lambda, x) & (128) \\ \text{subject to } \sum\limits_{i=1}^{N} \lambda_i \left(\frac{|w^T \tilde{x}_i + w_0|}{w^T \tilde{x}_i + w_0} - y_i \left(1 - \frac{1}{1+g_i} \right) \right) = 0 & (129) \\ 0 \leq \lambda_i \leq C \quad \text{due to Equation (75)}, \quad i = 1, 2, ..., N & (130) \end{cases}$$

where $T(w, w_0, \lambda, x) = \sum\limits_{i=1}^{N} \lambda_i \tilde{x}_i \left(\frac{|w^T \tilde{x}_i + w_0|}{w^T \tilde{x}_i + w_0} - y_i \left(1 - \frac{1}{1+g_i} \right) \right)$.

$$\begin{cases} w = \sum\limits_{i=1}^{N} \lambda_i \tilde{x}_i \left(\frac{|w^T \tilde{x}_i + w_0|}{w^T \tilde{x}_i + w_0} - y_i \left(1 - \frac{1}{1+g_i} \right) \right) & (131) \\ \lambda_i \left[|w^T \tilde{x}_i + w_0| - y_i \left(1 - \frac{1}{1+g_i} \right) (w^T \tilde{x}_i + w_0) - 1 + \xi_i \right] = 0, \quad i = 1, 2, ..., N \quad \text{(co. slack. con.)} & (132) \end{cases}$$

Training samples do not enter into (128), (131), and (132) in the form of their inner products ($\tilde{x}_i^T \tilde{x}_j$) in the k-dimensional space, thus the kernel trick cannot be used here.

A plausible approach for developing the weighted nonlinear SVM is to develop the weighted linear SVM in the k-dimensional space using the iterative algorithm at the end of Section 3.3.2:

- w and w_0 are initialized for a non-weighted SVM in the k-dimensional space,
- Loop: repeat until convergence
  - $$\hat{\tilde{X}}_t = \tilde{X} - \left(1 - \frac{1}{1+g} \right) \cdot \left(\frac{|\tilde{X} w_{t-1} + w_{0_{t-1}}|}{||w_{t-1}||} \right) .y \frac{w_{t-1}^T}{||w_{t-1}||} \tag{133}$$
  - find w_t and w_{0_t} for the non-weighted SVM classifier hyperplane based on $\hat{\tilde{X}}_t$

The above algorithm attempts to develop the non-weighted linear SVM classifier, $f(\tilde{x})$, in the k-dimensional space, iteratively relocate the mapped feature vectors ($\hat{\tilde{x}}_i$) with respect to this hyperplane based on their weights (133), and find the new hyperplane $f(\hat{\tilde{x}}_i)$ until convergence. However, the Mercer's theorem provides neither the dimensionality of the new space (k) nor the mapping function [33]. Therefore, it is not possible to map the feature vectors into the k-dimensional space and we do not know $\tilde{X}$ in (133).

To bypass the lack of knowledge about $\tilde{X}$ in the k-dimensional space, one might consider updating the position of feature vectors in the original l-dimensional space with respect to the nonlinear classifier hypersurface based on their weights iteratively until convergence. However, the hypersurface in the l-dimensional space is not known. It is worth noting that even if the hypersurface was known, moving the training samples perpendicularly toward/away from a hypersurface is a challenging mathematical problem.

5. Experiment with breast cancer data

The University of Wisconsin hospital breast cancer dataset was obtained from UCI machine learning repository [34]. This dataset has 683 samples, after samples with missing data are removed. This is a classification problem with 9 input features and 2 classes. The features include: clump thickness, uniformity of cell size, uniformity of cell shape, marginal adhesion, single epithelial cell size, bare nuclei, bland chromatin, normal nucleoli, mitoses. All input features are numerical values between 1 and 10. The two classes, that need to be predicted, are benign (444 samples) and malignant (239 samples). For weighted machine learning, we need a weight associated with each training sample, reflecting its reliability or accuracy. Due to the lack of such weights in this dataset (and to our knowledge in other machine learning datasets), we apply the following procedure to produce artificial weights for training samples. A function runs through each training sample, switches its output class from what it is to the other one with a random probability of $0 < k < 1$, and assigns a weight of $1 - k$ to that training sample. This way, a training sample's weight shows how reliable that training sample is and there are no changes imposed on test samples. Weighted machine learning techniques take advantage of these weights to take the training samples' reliability into account but non-weighted machine learning techniques ignore these weights. Leave-one-out or one-fold cross validation is used to estimate the accuracy of weighted and non-weighted machine learning techniques, reported in Table 3, where hyperparameters are optimized using cross-validation.

Table 3. Accuracy of different classification techniques for breast cancer prediction.

Technique	Overall accuracy (%)		Settings
	Non-weighted version	Weighted version	
Bayesian	46.71	76.72	Non-parametric Parzen windows with Gaussian kernel (10) where σ=1 Priors are based on class frequencies
LS	45.39	92.39	
SVM	51.24	64.86	Smoothing parameter (C)=18
Decision tree	48.17	89.90	A node is considered a leaf if the maximum impurity decrease (ΔI_{max}) for that node is less than 0.04
Perceptron	51.10	93.12	Logistic activation function Cost function: Sum of squared errors Maximum number of iterations: 1000 Not updating the weights after those iterations resulting in an increase in the total cost Multiply all learning rates by 1.1 or 0.8 after each step based on whether the total cost decreases or increases Adaptive learning rate: multiply the learning rate for a parameter by 1.2 if the partial derivative of the loss, with respect to that parameter, remains the same sign in successive steps and multiply it by 0.7 otherwise [35]
MLP	55.64	89.02	In addition to the settings for the perceptron: Maximum number of iterations: 2000 Number of hidden nodes for MLP: 2

The higher accuracy of weighted machine learning techniques (see Table 3) comes as no surprise since they take advantage of weights while non-weighted machine learning techniques do not. Nevertheless, it proves the proposed weighted machine learning techniques' efficiency in appropriately taking the training samples' weights into account, whenever such weights are available, in order to improve the prediction accuracy. Clearly, as mentioned earlier, the weighted SVM classifier shows the least difference with its non-weighted counterpart, underlying its relative reluctance to react to weights in comparison with other weighted predictors. The weighted least squares and weighted perceptron achieve the highest accuracy, shedding light on the linear separability of the two classes in this specific application. On the other hand, the weighted Bayesian classifier makes more dramatic changes in the border, resulting in a lower accuracy than other weighted non-linear classifiers (decision tree and MLP), for the same reason, i.e. the linear separability of the two classes in this specific application.

6. Conclusions

The weighted machine learning techniques developed in this paper provide developers with the opportunity to give different weights to training samples. These weights are used to adjust the classifier/regressor in favor of more importance samples, and thereby giving a higher significance to more important samples. It is worth noting that the weighted linear and nonlinear classifiers change the division of the feature space only around the border (the most uncertain area) and areas far from the border are less likely to change their label. The weighted SVM classifier showed the least difference with its non-weighted counterpart when the training samples' weights are not much variant. The reason for this is that SVM classifier is designed only based on support vectors not all training samples. If the weights of training samples are not much different, their relocation based on their weights might not be large enough to change the selection of support vectors. In other words, the weighted SVM classifier would be different than its non-weighted version only if relocating training samples based on their weights would result in a rearrangement of training samples significant enough to change the selection of support vectors. In other words, the weighted SVM has the highest stability with respect to weight changes in a small subset of training samples. The weighted MLP also takes the training samples' weights into account through smallest adjustments in its nonlinear border. However, the MLP's behavior is highly dependent on the network's size. In other words, a larger number of hidden nodes will result in a more significant difference between the weighted and non-weighted MLP. On the other hand, the weighted decision tree and weighted Bayesian classifiers showed the most dramatic changes in how the feature space is divided between the two classes in comparison with their non-weighted counterparts. The reason is that these two models are highly local, especially the non-parametric Bayesian classifier. Therefore, even small changes in the training samples' weights would result in a different classification of the feature space. The weighted LS and perceptron showed slight and similar changes in how they divide the feature space between the two classes in comparison with their non-weighted counterparts. The similar behavior is because both models minimize the sum of squared errors, although in different ways. The difference between weighted and non-weighted versions being slight in these two cases originates from their linear nature and consequently their rather restricted flexibility in modifying their shape. How the weighted machine learning techniques developed in this work will contribute in improving the prediction accuracy in different real-world applications is yet to be seen. Our next step, toward underscoring the significance of weighted machine learning models, is to apply them to spatial-temporal or environmental data, where the training samples' weights reflect the spatial-temporal autocorrelation between each training sample and the irresponsive (or test) sample.

REFERENCES

1. J. R. Quinlan, *Bagging, boosting, and C4.5*, Proceedings of the Thirteenth National Conference on Artificial Intelligence. Portland, Oregon: AAAI Press, pp. 725–730, 1996.
2. J. Friedman, T. Hastie, and R. Tibshirani, *Additive logistic regression: a statistical view of boosting*, The Annals of Statistics, vol. 28, no. 2, pp. 337–407, 2000.
3. L. Devroye, L. Gy?rfi, and G. Lugosi, *A probabilistic theory of pattern recognition*, Springer, 1996.
4. S. Theodoridis, and K. Koutroumbas, *Pattern Recognition*, (4th ed.). Elsevier, 2009.
5. E. Parzen, *On estimation of a probability density function and mode*, The Annals of Mathematical Statistics, vol. 33, no. 3, pp. 1065–1076, 1962.
6. W. Hardle, *Applied nonparametric regression*, Cambridge, UK: Cambridge University Press, 1990.
7. J. Fan, and I. Gijbels, *Local polynomial modelling and its applications: monographs on statistics and applied probability*, London: CRC Press, 1996.
8. S. J. Leon, *Linear algebra with applications*, (8th ed.). Upper Saddle River, NJ: Prentice Hall, 2010.
9. L. Wasserman, *All of nonparametric statistics*, Berlin: Springer, 2006.
10. F. Rosenblatt, *The perceptron: a probabilistic model for information storage and organization in the brain*, Psychological Review, vol. 65, no. 6, pp. 386–408, 1958.
11. M. L. Minsky, and S. Papert, *Perceptrons*, expanded edition. MA: MIT Press, 1988.
12. C. Cortes, and V. Vapnik, *Support-vector networks*, Machine Learning, vol. 20, no. 3, pp. 273–297, 1995.
13. V. N. Vapnik, *The nature of statistical learning theory*, New York: Springer, 1995.
14. V. N. Vapnik, *Statistical learning theory*, New York: Wiley, 1998.
15. M. S. Bazaraa, H. D. Sherali, and C. M. Shetty, *Nonlinear programming: theory and algorithms*, (3rd ed.). New Jersey: Wiley, 2006.
16. D. P. Bertsekas, *Nonlinear programming*, (2nd ed.). Belmont, Massachusetts: Athena Scientific, 1999.
17. R. Fletcher, *Practical methods of optimization*, (2nd ed.). Chichester, West Sussex, England: Wiley, 1987.

18. S. G. Nash, and A. Sofer, *Linear and nonlinear programming*, New York: McGraw-Hill, 1996.
19. L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and regression trees*, CRC press, 1984.
20. B. D. Ripley, *Neural networks and related methods for classification*, Journal of the Royal Statistical Society, Series B, vol. 56, no. 3, pp. 409–456, 1994.
21. P. J. Werbos, *Beyond regression: New tools for prediction and analysis in the behavioral sciences*, Cambridge, MA: Ph.D. Thesis, Harvard University, 1974.
22. C. M. Bishop, *Neural networks for pattern recognition*, Oxford University Press, 1995.
23. S. S. Haykin, *Neural networks: a comprehensive foundation*, (2nd ed.). Prentice Hall, 1999.
24. I. Witten, E. Frank, M. Hall, and C. Pal, *Data Mining: Practical machine learning tools and techniques*, (4th ed.). Morgan Kaufmann, 2016.
25. M. D. Richard, and R. P. Lippmann, *Neural network classifiers estimate Bayesian a posteriori probabilities*, Neural Computation, vol. 3, no. 4, pp. 461–483, 1991.
26. J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the theory of neural computation*, Addison-Wesley, 1991.
27. I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, MIT Press, 2016.
28. B. E. Boser, I. M. Guyon, and V. N. Vapnik, *A training algorithm for optimal margin classifiers*, Proceedings of the Fifth Annual Workshop on Computational Learning Theory. ACM, pp. 144–152, 1992.
29. T. M. Cover, *Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition*, IEEE Transactions on Electronic Computers, vol. 14, no. 3, pp. 326–334, 1965.
30. J. Mercer, *Functions of positive and negative type, and their connection with the theory of integral equations*, Philosophical Transactions of the Royal Society A, vol. 209, no. 1, pp. 415–446, 1909.
31. J. Shawe-Taylor, and N. Cristianini, *Kernel methods for pattern analysis*, Cambridge University Press, 2004.
32. B. Scholkopf, and A. J. Smola, *Learning with kernels: support vector machines, regularization, optimization, and beyond*, MIT Press, 2002.
33. R. Courant, and D. Hilbert, *Methods of Mathematical Physics (Vol. I)*, New York: Interscience, 1953.
34. W. H. Wolberg, *Breast Cancer Wisconsin Data Set, 1992*, Retrieved 2017, from http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Original%29
35. R. A. Jacobs, *Increased rates of convergence through learning rate adaptation*, Neural Networks, vol. 1, no. 4, pp. 295–307, 1988.