Multiplicative Update Rules for Incremental Training of Multiclass Support Vector Machines

Symeon Nikitidis[†*], Nikos Nikolaidis[†*] and Ioannis Pitas[†*]

[*]Informatics and Telematics Institute, Center for Research and Technology Hellas, Greece
[†]Department of Informatics, Aristotle University of Thessaloniki, Greece
email:{nikitidis,nikolaid,pitas}@aiia.csd.auth.gr

Abstract

We present a new method for the incremental training of multiclass Support Vector Machines that can simultaneously modify each class separating hyperplane and provide computational efficiency for training tasks where the training data collection is sequentially enriched and dynamic adaptation of the classifier is required over time. An auxiliary function has been designed, that incorporates some desired characteristics in order to provide an upper bound for the objective function, which summarizes the multiclass classification task. A novel set of multiplicative update rules is proposed, which is independent from any kind of learning rate parameter, provides computational efficiency compared to the conventional batch training approach and is easy to implement. Convergence to the global minimum is guaranteed, since the optimization problem is convex and the global minimizer for the enriched dataset is found using a warm-start algorithm. Experimental evidence on various data collections verified that our method is faster than retraining the classifier from scratch, while the achieved classification accuracy rate is maintained at the same level.

Keywords: Support vector machines, Online training, Incremental learning, Quadratic programming, Warm-start algorithm.

1. Introduction

Support Vector Machines (SVMs) [1]-[3] have become popular in pattern recognition problems due to their excellent generalization performance on unseen data and their wide range of applicability in various classification tasks. For the simple binary pattern recognition problem, given a set of training examples, each belonging to one of two pattern classes, SVMs usually map each example to a point in a higher or infinite dimensional feature space. Consequently, a SVM training algorithm, based on the statistical learning theory, constructs an optimal decision hyperplane in this feature space, which separates the two classes, while at the same time, maximizes the margin between itself and the nearest training examples (called the support vectors) of each class. Classification of unseen examples is then performed by firstly mapping them on the same high dimensional feature space and assigning them to a class based on which side of the separating hyperplane they fall in. The optimization process during SVM training is driven by the Structural Risk Minimization (SRM) [1] principle, attempting to find the optimal balance between the minimization of the mean error rate in the training set (called the empirical risk) and the prevention of overfitting.

The major drawback of SVMs is their complicated training procedure which restricts their applicability especially for large scale industrial applications. Indeed, SVM training involves the solution of a large Quadratic Programming (QP) problem [4], which demands significant computational effort and requires considerable memory resources. These requirements are further increased when training is performed on large scale datasets containing high dimensional data, thus making SVM training almost impractical.

To this end, significant effort has been spent in order to provide simplified, fast and less resource demanding SVM batch training algorithm variants. Among the most notable and widespread approaches for fast SVM training is the one proposed by Frieß *et al* [5], where the kernel theory and the ability to form linear decision surfaces in feature space have been adapted to the Adatron algorithm. Another well known variant is the Sequential Minimal Optimization (SMO) [6] algorithm by Platt, which works with reduced problems by splitting the QP problem in the smallest possible subproblem including at each step the optimization of a single pair of Lagrange multipliers. In order to choose the two Lagrange multipliers for joint optimization, SMO computes the violation on Karush-Kuhn-Tucker (KKT) conditions constraints [4] and selects for optimization the pair that causes the highest violation. Next, the optimal values for these multipliers are computed and SMO updates the SVM to reflect the new optimal solution. Another popular variant is the MinOver algorithm by Krauth and Mezard [7], which defines the maximum margin hyperplane in linearly separable problems by determining synaptic matrices of optimal stability, thus achieving faster convergence compared with the conventional SVM batch training approach.

The main limitation of all the aforementioned methods is that they employ a batch training approach which requires all training data to be available at once and training is performed in one batch. This implies that if more training data are available subsequently, the SVM classifier should be retrained from scratch. Let us investigate the scenario, where we have obtained the optimal Lagrange multipliers defining the normal vector of the decision surface for n initial training samples that form the so called *base* training set, and we seek the new optimal values for the *augmented* dataset formed when m new training pairs are added to the base training set. In such a case, since the classifier was initially well trained, resolving the optimization problem from scratch over the augmented training dataset, is computationally inefficient. An alternative approach could be to use the initial solution and the optimal Lagrange multipliers obtained from the base training dataset, as an advanced starting point to warm-start the new optimization process. The computational advantage of such an approach is extremely large in cases where a small amount of new training samples is added in a large base training dataset ($m \ll n$), over which the SVM classifier has already been well trained.

To this end, numerous approaches for online training of two class classifiers have been proposed in the literature. In online training when a single data point is added and/or removed, these algorithms can efficiently update the trained model without re-training it from scratch. Huller algorithm [8] is such a method, which tries to solve the two class SVM quadratic problem in an online manner based on geometrical arguments. The ideas introduced in Huller were further extended in LASVM [9] algorithm, which uses stochastic gradient decent to approximate the optimum solution. However, optimization algorithms that rely on the full gradient computation are less attractive since gradient is usually very large and not sparse. To overcome this issue, the LaRank algorithm [10] which exploits only the related to the support vectors partial gradient information was introduced. Another approach for incremental SVM training has been proposed in [11] by Cauwenberghs *et al*, that ensures fulfilment of the KKT conditions on all previously seen training samples, while "adiabatically" adding a single data sample to the solution. An extension of this algorithm that incorporates an efficient storage design able to cope with limited memory resources and a different organization of the computation procedure that significantly speeds up training was presented in [12]. An implementation of the two previously mentioned methods on one-class support vector classifiers has been presented in [13].

It is common knowledge that support vectors comprise a small fraction of the total training data samples set. Moreover, considering that the optimal decision hyperplane resulting from SVMs training is determined explicitly by those training samples that correspond to support vectors, in order to reduce the computational cost it is essential to train the classifier using the smallest possible number of non support vector samples. An attractive approach towards this direction is active set optimization, since these methods are able to reduce problem dimensionality by reducing the number of training data, ideally considering only the support vectors (or those training samples appearing as support vector candidates). Solving a linearly constrained QP problem, as the one appearing in SVMs training, an active set method divides imposed constraints into two sets: the set of active constraints (the active set) and the set of inactive ones. Consequently, in the context of SVMs training an active set method starts with an initial set of constraints and iteratively adjusts it by adding and removing constraints, while testing if the solution remains feasible, until the optimal active set and, hence, the optimal solution is reached.

Towards this direction, active set methods have been effectively combined with warm-start algorithms. The main motivation for applying a warm-start strategy is the expectation that two closely related optimization problems, such as the batch training over a base training set and the incremental training over an augmented training set, should, in general, share similar characteristics. More precisely, the new decision surface is expected to have minimal disturbances with respect to its previous form, when a small number of new training samples is added to a base training dataset. An active set approach which involves a warm-start algorithm to incrementally train SVMs was proposed in [14], while in [15] incremental training achieved by solving the primal minimization problem instead. Another active set approach that bounds the training set size by seeking to identify possible support vectors among the available training samples has been presented in [16]. In order to do so, only those training data samples belonging to different classes whose distance in the projected feature space is less than a defined threshold are included in the working set during the learning procedure. Similarly, Katagiri *et al* proposed in [17] an incremental training algorithm for one-class SVMs that considers a hypersphere in order to identify support vector candidate samples utilized for incremental training, while in [18] the training data working set is determined based on geometrical arguments.

Although various papers have been published proposing more efficient variants of the SVM batch training and also many have considered incremental training of two class SVM classifiers, the problem to handle incremental training of multiclass classifiers remains unsettled. The dominant approach for solving multiclass problems using SVMs has been based on reducing a single multiclass problem into multiple binary ones. For instance, a common method is to build a set of binary classifiers where each classifier distinguishes members of one class to the rest. The main disadvantage of such an approach is that such a classifier cannot capture correlations between the different classes. Direct Acyclic Graph (DAG) SVMs [19] are among the most popular such methods, consisting of multiple binary classifiers organized so that during testing, a path is traversed starting from a root node and reaching a leaf node indicating the predicted class. Although DAGSVMs achieve fast testing time, which makes them suitable for practical use, since they are composed by multiple binary classifiers, correlations between different classes are neglected, thus making online training of DAGSVM classifiers not a trivial task. In [20] incremental and decremental training of DAGSVMs has been attempted based on the idea presented in [11]. However, the computational gain attained by the applied incremental training strategy is counterbalanced, since DAGSVM training requires to adapt $k(k-1)/2$ binary classifiers, where k is the number of classes. Consequently, the application of such an approach becomes impractical for problems involving a large number of classes.

Few attempts have been made to generalize SVMs to multiclass problems that handle all available data together [21, 22]. In these attempts, extensions of the binary case into a multiclass one are achieved by adding appropriate constraints for every class. Consequently, the size of the quadratic optimization problem is proportional to the number of classes in the classification task at hand. Moreover, the result of these approaches is often a homogeneous quadratic problem which is hard to solve and difficult to store [23]. Crammer and Singer [24] proposed an approach for multiclass problems by solving a single optimization schema. This approach considers all variables together, which is achieved using a direct multiclass formulation and the output space handles all classes simultaneously. As a result, correlations between different classes are retained and thus simultaneous modifications of the various separating hyperplanes, as the training samples of each class adjust, can be studied. This property could be exploited by a proper incremental learning algorithm which is required to simultaneously modify the decision hyperplanes of all classes when new samples are added in the training dataset. We explore this property and propose an incremental training method suitable for dynamic multiclass training tasks.

In this paper, we propose a set of multiplicative update rules for the evaluation of the optimal Lagrange multipliers that determine the normal vectors of a multiclass SVM classifier, in an incremental manner. The proposed incremental learning algorithm can simultaneously modify the form of each class separating hyperplane, when new samples are inserted in the training data set. The update rules are derived using a proper auxiliary function and are totaly independent from any learning rate parameter, provide computational efficiency compared to the conventional batch training approach and are easy to implement. Moreover, convergence to the global minimum is guaranteed, since the optimization problem is convex and, as a result, any reached minimum is global. In order to further speed up the optimization procedure, we have selected to warm-start the solution process by using the previous solution and the optimal Lagrange multipliers computed over the base training set. With this approach, faster convergence is expected, since, in general, the new training samples modify the decision hyperplane in a relatively smooth manner. As a result, only a small portion of the SVM parameters should be evaluated and only few of the old Lagrange multipliers would require an update. Additionally, the proposed update rules can be applied to sign insensitive kernels, thus enabling us to perform the kernel trick [25] and to separate linearly classes projected in higher dimensional feature space using arbitrary Mercer's kernels [26].

In summary, the novel contributions of this paper are the following:

- An extension of the conventional multiclass SVMs formulation that enables the simultaneous incremental update of the SVM parameters.

- The design of a convex auxiliary function bounding from above the objective function summarizing the multiclass classification problem.

- A novel set of multiplicative update rules for the evaluation of the optimal Lagrange multipliers associated with the multiclass SVM classifier. The proposed updates monotonically converge to the global minimum.

- The proposed update rules are combined with a warm-start algorithm to further speed up the optimization procedure.

The rest of the paper is organized as follows. Section 2 briefly reviews the general configuration of multiclass SVMs and provides an extension of this formulation in order to facilitate the proposed incremental training method. In Section 3, the design of an auxiliary function that provides the upper bound of the objective is described and the proposed multiplicative update rules are derived. Verification regarding convergence of the proposed multiplicative update rules is also provided and the concept of the warm-start framework is introduced. Section 4 describes the conducted experiments and presents experimental evidence regarding the convergence and the computational efficiency of the proposed incremental training method, while concluding remarks are drawn in Section 5. This paper substantially extends our preliminary work in [27]. Compared to the previous work, here we extend the proposed method such as to handle arbitrary Mercer's kernels, relax the non-negativity constraint applied in the Lagrange multipliers, demonstrate in detail the design of the used auxiliary function and prove the desired optimization properties.

2. Problem Formulation

2.1. Multiclass SVMs

Crammer and Singer [24] proposed an approach for multiclass classification problems by solving a single optimization schema. Given a set of l training data $\mathcal{X} = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_l, y_l)\}$ where $\mathbf{x}_i \in R^d, i = 1, ..., l$ are the input feature vectors, $y_i \in \{1, ..., k\}$ is the class label associated with sample $\mathbf{x}_i$, d is the dimensionality of the input feature vectors and k is the number of classes, the idea is to consider all available training data at once and construct k two-class classification rules. Solving this single optimization problem leads to the construction of k decision functions, where the p-th decision surface $\mathbf{w}_p^T \phi(\mathbf{x})$ determined by its normal vector $\mathbf{w}_p \in R^d$, separates the training vectors of the p-th class from all the others, by solving the following primal minimization problem:

$$\min_{\mathbf{w}_p, \xi_i} \frac{1}{2} \sum_{p=1}^{k} \mathbf{w}_p^T \mathbf{w}_p + C \sum_{i=1}^{l} \xi_i \tag{1}$$

subject to the constraints:

$$\mathbf{w}_{y_i}^T \phi(\mathbf{x}_i) - \mathbf{w}_p^T \phi(\mathbf{x}_i) \geq b_i^p - \xi_i, \quad i = 1, \ldots, l. \tag{2}$$

Here, $\phi(.)$ is a function that maps the input feature vector $\mathbf{x}_i$ to an arbitrary-dimensional space $\mathcal{F}$, which usually has the structure of a Hilbert space [28][29], where the data are supposed to be linearly or near linearly separable. C is the term that penalizes the training errors, $\boldsymbol{\xi} = [\xi_1, ..., \xi_l]^T$

is the slack variables vector and $\mathbf{b}$ is a bias vector defined for $p = 1, ..., k$ as:

$$b_i^p = 1 - \delta_{y_i}^p = \begin{cases} 1 & \text{, if } y_i \neq p \\ \\ 0 & \text{, if } y_i = p \end{cases} \tag{3}$$

where $\delta_{y_i}^p$ is the Kronecker delta function which is 1 for $y_i = p$ and 0 otherwise. The decision function is:

$$\arg \max_{p=1,\ldots,k} \left(\mathbf{w}_p^T \phi(\mathbf{x}) \right). \tag{4}$$

Switching to the dual formulation, the solution of the constrained optimization problem summarized in (1) and (2) can be found from the saddle point of the Lagrangian function:

$$\begin{aligned} L\left(\mathbf{w}, \boldsymbol{\xi}, \mathbf{n}\right) \quad &= \frac{1}{2}\sum_{p=1}^{k} \mathbf{w}_p^T \mathbf{w}_p + C \sum_{i=1}^{l} \xi_i - \sum_{i=1}^{l}\sum_{p=1}^{k} n_i^p \left[\left(\mathbf{w}_{y_i}^T - \mathbf{w}_p^T \right) \phi\left(\mathbf{x}_i\right) + \xi_i - b_i^p \right] \\ \text{subject to:} \qquad &\forall i, p \quad n_i^p \geq 0 \quad , i = 1, \ldots, l \quad , p = 1, \ldots, k \end{aligned} \tag{5}$$

where $\mathbf{n} = \left[n_1^1, ..., n_1^k, ..., n_l^1, ..., n_l^k \right]^T$ are the Lagrange multipliers associated with the constraints in (2). The Lagrangian function in (5) has to be maximized with respect to the dual variables $\mathbf{n}$ and minimized with respect to the primal ones $\mathbf{w}$ and $\boldsymbol{\xi}$. Since the minimum over the primal variables satisfies the KKT conditions for $p = 1, \ldots, k$, the following equations hold:

$$\begin{aligned} \frac{\partial L\left(\mathbf{w}, \boldsymbol{\xi}, \mathbf{n}\right)}{\partial \xi_i} \quad &= 0 \Rightarrow \sum_{p=1}^{k} n_i^p = C \\ \frac{\partial L\left(\mathbf{w}, \boldsymbol{\xi}, \mathbf{n}\right)}{\partial \mathbf{w}_p} \quad &= 0 \Rightarrow \mathbf{w}_p = \sum_{i=1}^{l} \phi\left(\mathbf{x}_i\right) \left(n_i^p - C\delta_{y_i}^p \right). \end{aligned} \tag{6}$$

Substituting terms from (6) into (5), the saddle point of the Lagrangian is reformulated to the maximization of the Wolfe dual problem:

$$\max_{\mathbf{n}} \qquad W(\mathbf{n}) = -\frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l} \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j) \sum_{p=1}^{k} \left(C\delta_{y_i}^p - n_i^p \right) \left(C\delta_{y_j}^p - n_j^p \right) + \sum_{i=1}^{l}\sum_{p=1}^{k} n_i^p b_i^p. \tag{7}$$

In order to produce a more compact equation form, let us introduce the following notations:

$$[\mathbf{K}]_{i,j} = \mathbf{K}(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j), \quad \overline{\mathbf{n}}_i = \left[n_i^1, ..., n_i^k \right]^T \quad \text{and} \quad \overline{\mathbf{b}}_i = \left[b_i^1, ..., b_i^k \right]^T, \tag{8}$$

where $\mathbf{K}$ is the kernel matrix, whose (i,j)-th element is $[\mathbf{K}]_{i,j} = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$. Additionally, we define the k-dimensional vector $\overline{\mathbf{1}}_{y_i}$ with all its components equal to zero, except the y_i-th, which is equal to one. Then we perform the substitution: $\overline{\boldsymbol{\alpha}}_i = C\overline{\mathbf{1}}_{y_i} - \overline{\mathbf{n}}_i$. Now the maximization of the Wolfe dual problem in (7) is equivalent to the following minimization problem:

$$\min_{\boldsymbol{\alpha}} \quad W(\boldsymbol{\alpha}) = \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}[\mathbf{K}]_{i,j}\,\overline{\boldsymbol{\alpha}}_i^T\overline{\boldsymbol{\alpha}}_j - C\sum_{i=1}^{l}\overline{\mathbf{1}}_{y_i}\overline{\mathbf{b}}_i + \sum_{i=1}^{l}\overline{\boldsymbol{\alpha}}_i^T\,\overline{\mathbf{b}}_i \tag{9}$$

which is a quadratic function in terms of $\boldsymbol{\alpha} = \left[\alpha_1^1, \ldots, \alpha_1^k, \ldots, \alpha_l^1, \ldots, \alpha_l^k\right]^T$ since $\sum_{i=1}^{l}\overline{\mathbf{1}}_{y_i}\overline{\mathbf{b}}_i = 0$ and the bias vector is defined as $\mathbf{b} = \left[b_1^1, \ldots, b_1^k, \ldots, b_l^1, \ldots, b_l^k\right]^T$. Consequently, the multiclass classification task can be summarized to the following single optimization problem:

$$\min_{\boldsymbol{\alpha}} \quad W(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^T\mathbf{H}\boldsymbol{\alpha} + \mathbf{b}^T\boldsymbol{\alpha}, \tag{10}$$

under the following linear constraints:

$$\forall i, p \quad \alpha_i^p \leq \begin{cases} 0 & \text{, if } y_i \neq p \\ C & \text{, if } y_i = p. \end{cases} \tag{11}$$

Here $\mathbf{H}$ is the Hessian matrix defined as $\mathbf{H} = \mathbf{K}\otimes\mathbf{I}$, $\mathbf{I}$ is a k by k identity matrix and $\otimes$ denotes the Kronecker product. The Hessian matrix $\mathbf{H}$, is symmetric and positive semidefinite, since it has been derived by a direct product operation on the kernel matrix $\mathbf{K}$, which is also symmetric and positive semidefinite. This property reveals a so called quadratic program, since the objective function $W(\boldsymbol{\alpha})$ is a convex quadratic function with linear constraints and, consequently, its optimization problem has a global minimizer. Finally, the decision rule for a test sample $\mathbf{x}$ is given by:

$$\arg\max_{p=1\ldots k} \quad \sum_{i=1}^{l}\alpha_i^p\mathbf{K}(\mathbf{x}_i, \mathbf{x}). \tag{12}$$

2.2. Extending the Conventional Multiclass SVM Formulation

We investigate the incremental training task by examining the scenario where the SVM classifier has been trained over a base dataset $\mathcal{X}_n$ of n training pairs $\mathcal{X}_n = \{(\mathbf{x}_i, y_i), i = 1, \ldots, n\}$ and the optimal Lagrange multipliers $\boldsymbol{\alpha}_{n,o}$ that minimize the objective function in (10) have been evaluated. When m new training samples $\mathcal{X}_m = \{(\mathbf{x}_s, y_s), s = n+1, \ldots, n+m\}$ are added to the base training dataset, creating the augmented training dataset $\mathcal{X}_{n+m} = \mathcal{X}_n \bigcup \mathcal{X}_m$, we want to update the current SVM configuration and obtain a new SVM classifier that also incorporates information from the new training data contained in set $\mathcal{X}_m$.

In order to facilitate dynamic adaptation of the SVM classifier to the augmented training dataset $\mathcal{X}_{n+m}$, we express the new training task with respect to its initial form, along with an update term corresponding to the new training samples set $\mathcal{X}_m$. However, since the base and the augmented classification problems do not have the same number of constraints and variables, it is required to expand vectors $\boldsymbol{\alpha}_{n,o}$, $\mathbf{b}_n$ and the Hessian matrix $\mathbf{H}_n$ accordingly:

$$\begin{aligned} \boldsymbol{\alpha}_{n+m} &= \begin{bmatrix} \boldsymbol{\alpha}_{n,o} \\ \boldsymbol{\alpha}_s \end{bmatrix} \quad , \quad \mathbf{H}_{n+m} = \begin{bmatrix} \mathbf{H}_n & \mathbf{K}(\mathbf{x}_i, \mathbf{x}_s) \otimes \mathbf{I} \\ \mathbf{K}(\mathbf{x}_i, \mathbf{x}_s)^T \otimes \mathbf{I} & \mathbf{K}(\mathbf{x}_s, \mathbf{x}_s) \otimes \mathbf{I} \end{bmatrix} \\ \mathbf{b}_{n+m} &= \begin{bmatrix} \mathbf{b}_n \\ \mathbf{b}_s \end{bmatrix} \quad , \quad i = 1, ..., n \quad , \quad s = n+1, ..., n+m, \end{aligned} \tag{13}$$

where $\boldsymbol{\alpha}_{n,o} = \left[\alpha_1^1, ..., \alpha_1^k, ..., \alpha_n^1, ..., \alpha_n^k\right]^T$, $\boldsymbol{\alpha}_s = \left[\alpha_{n+1}^1, ..., \alpha_{n+1}^k, ..., \alpha_{n+m}^1, ..., \alpha_{n+m}^k\right]^T$ and bias vectors $\mathbf{b}_n = \left[b_1^1, ..., b_1^k, ..., b_n^1, ..., b_n^k\right]^T$ and $\mathbf{b}_s = \left[b_{n+1}^1, ..., b_{n+1}^k, ..., b_{n+m}^1, ..., b_{n+m}^k\right]^T$. $\mathbf{H}_n$ is the $nk \times nk$ Hessian matrix computed over the n base training samples of set $\mathcal{X}_n$, while $\mathbf{K}(\mathbf{x}_i, \mathbf{x}_s)$ and $\mathbf{K}(\mathbf{x}_s, \mathbf{x}_s)$ are $n \times m$ and $m \times m$ kernel matrices used in order to expand the initial Hessian matrix $\mathbf{H}_n$ to $\mathbf{H}_{n+m}$ and are evaluated the first using only the training pairs of the base dataset, while the latter over the augmented training dataset.

The Hessian matrix $\mathbf{H}_{n+m}$ can be expressed as the sum $\mathbf{H}_{n+m} = \bar{\mathbf{H}}_n + \mathbf{H}_m$, of the Hessian $\bar{\mathbf{H}}_n$ computed over the base training dataset and an update term, matrix $\mathbf{H}_m$, evaluated using the training samples of the augmented set $\mathcal{X}_m$. Both matrices are defined as:

$$\bar{\mathbf{H}}_n = \begin{bmatrix} \mathbf{H}_n & \mathbf{0}_{nk\times mk} \\ \mathbf{0}_{mk\times nk} & \mathbf{0}_{mk\times mk} \end{bmatrix} , \quad \mathbf{H}_m = \begin{bmatrix} \mathbf{0}_{nk\times nk} & \mathbf{K}(\mathbf{x}_i, \mathbf{x}_s) \otimes \mathbf{I} \\ \mathbf{K}(\mathbf{x}_i, \mathbf{x}_s)^T \otimes \mathbf{I} & \mathbf{K}(\mathbf{x}_s, \mathbf{x}_s) \otimes \mathbf{I} \end{bmatrix} \quad \begin{matrix} i = 1, ..., n \\ s = n+1, ..., n+m, \end{matrix} \tag{14}$$

where $\mathbf{0}$ is an all-zero matrix of appropriate dimensions. In order to handle sign-insensitive kernels, whose elements can be positive, negative or zero, we decompose $\bar{\mathbf{H}}_n$ such as $\bar{\mathbf{H}}_n = \bar{\mathbf{H}}_n^+ - \bar{\mathbf{H}}_n^-$, where the non-negative matrices $\bar{\mathbf{H}}_n^+$ and $\bar{\mathbf{H}}_n^-$ are expressed in terms of the positive and negative elements of matrix $\bar{\mathbf{H}}_n$ as:

$$[\bar{\mathbf{H}}_n^+]_{ij} = \begin{cases} [\bar{\mathbf{H}}_n]_{ij} & \text{, if } [\bar{\mathbf{H}}_n]_{ij} > 0 \\ 0 & \text{, otherwise} \end{cases} \qquad [\bar{\mathbf{H}}_n^-]_{ij} = \begin{cases} |[\bar{\mathbf{H}}_n]_{ij}| & \text{, if } [\bar{\mathbf{H}}_n]_{ij} < 0 \\ 0 & \text{, otherwise.} \end{cases} \tag{15}$$

Similarly $\mathbf{H}_m$ is decomposed into $\mathbf{H}_m^+$ and $\mathbf{H}_m^-$. Subsequently, the objective function $W(\boldsymbol{\alpha}_{n+m})$ minimized over the augmented training set is formulated as:

$$\min_{\boldsymbol{\alpha}_{n+m}} \quad W(\boldsymbol{\alpha}_{n+m}) = \frac{1}{2}\boldsymbol{\alpha}_{n+m}^T \left(\bar{\mathbf{H}}_n^+ - \bar{\mathbf{H}}_n^- + \mathbf{H}_m^+ - \mathbf{H}_m^-\right) \boldsymbol{\alpha}_{n+m} + \mathbf{b}_{n+m}^T \boldsymbol{\alpha}_{n+m} \tag{16}$$

under the linear constraints summarized in (11), where, in this case, $i = 1, \ldots, n+m$.

3. Incremental Training Algorithm

In this section, we demonstrate the design of an auxiliary function that provides an upper bound of the objective function $W(\boldsymbol{\alpha}_{n+m})$ formulated in (16), which summarizes the multiclass classification task as a single optimization problem. We also derive the proposed multiplicative update rules that guarantee convergence to the objective function global minimum and finally, we introduce the warm-start optimization framework.

3.1. Auxiliary Function

We define an auxiliary function F in order to identify the global minimizer of the objective function $W(\boldsymbol{\alpha}_{n+m})$. The derivation of the auxiliary function we have followed is similar with the one presented in [30], [31]. Similar techniques have been also used in order to establish the convergence of many statistical learning algorithms, e.g., the Expectation-Maximization algorithm [32] for maximum likelihood estimation and Non-negative Matrix Factorization [33]. Since the minimization problem is a QP problem and has a global and no local minima, we seek to define an appropriate convex auxiliary function F, which will provide an upper bound on the objective. This auxiliary function should satisfy the following properties:

1. It should bound the objective function from above:

$$W(\mathbf{u}) \leq F(\mathbf{u}, \boldsymbol{\alpha}_{n+m}). \tag{17}$$

2. The following equality should hold:

$$W(\boldsymbol{\alpha}_{n+m}) = F(\boldsymbol{\alpha}_{n+m}, \boldsymbol{\alpha}_{n+m}). \tag{18}$$

Such an auxiliary function provides an upper bound on the objective function $W(\boldsymbol{\alpha}_{n+m})$. Our goal is to use this auxiliary function F in order to derive a series of minimizers $\boldsymbol{\alpha}'$, using the update rule $\boldsymbol{\alpha}' = \arg\min_{\mathbf{u}} F(\mathbf{u}, \boldsymbol{\alpha})$, which will never increase the objective function, since the following inequality is valid:

$$W\left(\boldsymbol{\alpha}'\right) \leq F\left(\boldsymbol{\alpha}', \boldsymbol{\alpha}\right) \leq F(\boldsymbol{\alpha}, \boldsymbol{\alpha}) = W(\boldsymbol{\alpha}). \tag{19}$$

The minimizer $\boldsymbol{\alpha}'$ can be found by computing the derivative of the auxiliary function with respect to $\mathbf{u}$ and setting it equal to zero. By iterating this update, a series of minimizers $\boldsymbol{\alpha}'$ are generated that improve the objective function and will eventually lead to the global minimum, since the convexity property of $W(\boldsymbol{\alpha}_{n+m})$ implies that any reached local minimum is also a global one.

3.2. Multiplicative Update Rules Derivation

As has been shown in [30], the following inequalities hold:

$$\begin{aligned} \frac{1}{2}\mathbf{u}^T\bar{\mathbf{H}}_n^+\mathbf{u} &\leq \frac{1}{2}\sum_i \frac{\left[\bar{\mathbf{H}}_n^+\boldsymbol{\alpha}_{n+m}\right]_i}{\left[\boldsymbol{\alpha}_{n+m}\right]_i}u_i^2 && (20)\\ -\frac{1}{2}\mathbf{u}^T\bar{\mathbf{H}}_n^-\mathbf{u} &\leq -\frac{1}{2}\sum_{ij}\left[\bar{\mathbf{H}}_n^-\right]_{ij}\left[\boldsymbol{\alpha}_{n+m}\right]_i\left[\boldsymbol{\alpha}_{n+m}\right]_j\left(1+\log\frac{u_iu_j}{\left[\boldsymbol{\alpha}_{n+m}\right]_i\left[\boldsymbol{\alpha}_{n+m}\right]_j}\right) && (21)\end{aligned}$$

where $\left[\bar{\mathbf{H}}_n^-\right]_{ij}$ and $\left[\boldsymbol{\alpha}_{n+m}\right]_i$ denote the (i,j)-th and i-th components of matrix $\bar{\mathbf{H}}_n^-$ and vector $\boldsymbol{\alpha}_{n+m}$, respectively. Similar inequalities also hold for the non-negative matrices $\mathbf{H}_m^+$ and $\mathbf{H}_m^-$. In terms of the non-negative matrices $\bar{\mathbf{H}}_n^+$, $\bar{\mathbf{H}}_n^-$, $\mathbf{H}_m^+$ and $\mathbf{H}_m^-$, we can decompose the quadratic part of the objective function and formulate it as follows:

$$W(\boldsymbol{\alpha}_{n+m}) = W_c(\boldsymbol{\alpha}_{n+m}) - W_d(\boldsymbol{\alpha}_{n+m}) + W_e(\boldsymbol{\alpha}_{n+m}) - W_f(\boldsymbol{\alpha}_{n+m}) + W_b(\boldsymbol{\alpha}_{n+m}), \qquad (22)$$

where each part is defined as:

$$\begin{aligned} W_c(\boldsymbol{\alpha}_{n+m}) &= \frac{1}{2}\boldsymbol{\alpha}_{n+m}^T\bar{\mathbf{H}}_n^+\boldsymbol{\alpha}_{n+m} \quad , \quad W_d(\boldsymbol{\alpha}_{n+m}) = \frac{1}{2}\boldsymbol{\alpha}_{n+m}^T\bar{\mathbf{H}}_n^-\boldsymbol{\alpha}_{n+m},\\ W_e(\boldsymbol{\alpha}_{n+m}) &= \frac{1}{2}\boldsymbol{\alpha}_{n+m}^T\mathbf{H}_m^+\boldsymbol{\alpha}_{n+m} \quad , \quad W_f(\boldsymbol{\alpha}_{n+m}) = \frac{1}{2}\boldsymbol{\alpha}_{n+m}^T\mathbf{H}_m^-\boldsymbol{\alpha}_{n+m}, \qquad (23)\\ W_b(\boldsymbol{\alpha}_{n+m}) &= \mathbf{b}_{n+m}^T\boldsymbol{\alpha}_{n+m}.\end{aligned}$$

The upper bound of the objective function can be determined by applying inequalities (20) and (21) separately in each one of the five terms of (22). Hence, the auxiliary function $F(\mathbf{u},\boldsymbol{\alpha}_{n+m})$ can be derived by adding the upper bounds of each term in the objective:

$$\begin{aligned} F(\mathbf{u},\boldsymbol{\alpha}_{n+m}) &= \frac{1}{2}\sum_i\frac{\left[\bar{\mathbf{H}}_n^+\boldsymbol{\alpha}_{n+m}\right]_i}{\left[\boldsymbol{\alpha}_{n+m}\right]_i}u_i^2 - \frac{1}{2}\sum_{ij}\left[\bar{\mathbf{H}}_n^-\right]_{ij}\left[\boldsymbol{\alpha}_{n+m}\right]_i\left[\boldsymbol{\alpha}_{n+m}\right]_j\left(1+\log\frac{u_iu_j}{\left[\boldsymbol{\alpha}_{n+m}\right]_i\left[\boldsymbol{\alpha}_{n+m}\right]_j}\right)\\ &+\frac{1}{2}\sum_i\frac{\left[\mathbf{H}_m^+\boldsymbol{\alpha}_{n+m}\right]_i}{\left[\boldsymbol{\alpha}_{n+m}\right]_i}u_i^2 - \frac{1}{2}\sum_{ij}\left[\mathbf{H}_m^-\right]_{ij}\left[\boldsymbol{\alpha}_{n+m}\right]_i\left[\boldsymbol{\alpha}_{n+m}\right]_j\left(1+\log\frac{u_iu_j}{\left[\boldsymbol{\alpha}_{n+m}\right]_i\left[\boldsymbol{\alpha}_{n+m}\right]_j}\right)\\ &+\sum_i\left[\mathbf{b}_{n+m}\right]_i u_i \end{aligned} \qquad (24)$$

which is an auxiliary function for $W\left(\boldsymbol{\alpha}_{n+m}\right)$ satisfying $F(\mathbf{u},\boldsymbol{\alpha}_{n+m}) \geq W(\boldsymbol{\alpha}_{n+m})$ by construction via inequalities (20) and (21). The following relation also holds:

$$\begin{aligned}
F(\boldsymbol{\alpha}_{n+m},\boldsymbol{\alpha}_{n+m}) \;&= \frac{1}{2}\sum_{i}\left[\bar{\mathbf{H}}_n^{+}\boldsymbol{\alpha}_{n+m}\right]_i\left[\boldsymbol{\alpha}_{n+m}\right]_i - \frac{1}{2}\sum_{ij}\left[\bar{\mathbf{H}}_n^{-}\right]_{ij}\left[\boldsymbol{\alpha}_{n+m}\right]_i\left[\boldsymbol{\alpha}_{n+m}\right]_j \\
&+\frac{1}{2}\sum_{i}\left[\mathbf{H}_m^{+}\boldsymbol{\alpha}_{n+m}\right]_i\left[\boldsymbol{\alpha}_{n+m}\right]_i - \frac{1}{2}\sum_{ij}\left[\mathbf{H}_m^{-}\right]_{ij}\left[\boldsymbol{\alpha}_{n+m}\right]_i\left[\boldsymbol{\alpha}_{n+m}\right]_j + \sum_{i}\left[\mathbf{b}_{n+m}\right]_i\left[\boldsymbol{\alpha}_{n+m}\right]_i \\
&= \frac{1}{2}\sum_{i}\Big(\left[\bar{\mathbf{H}}_n^{+}\boldsymbol{\alpha}_{n+m}\right]_i\left[\boldsymbol{\alpha}_{n+m}\right]_i - \left[\bar{\mathbf{H}}_n^{-}\boldsymbol{\alpha}_{n+m}\right]_i\left[\boldsymbol{\alpha}_{n+m}\right]_i\Big)+ \\
&\frac{1}{2}\sum_{i}\Big(\left[\mathbf{H}_m^{+}\boldsymbol{\alpha}_{n+m}\right]_i\left[\boldsymbol{\alpha}_{n+m}\right]_i - \left[\mathbf{H}_m^{-}\boldsymbol{\alpha}_{n+m}\right]_i\left[\boldsymbol{\alpha}_{n+m}\right]_i\Big) + \sum_{i}\left[\mathbf{b}_{n+m}\right]_i\left[\boldsymbol{\alpha}_{n+m}\right]_i \\
&= \frac{1}{2}\boldsymbol{\alpha}_{n+m}^{T}\left(\bar{\mathbf{H}}_n^{+} - \bar{\mathbf{H}}_n^{-}\right)\boldsymbol{\alpha}_{n+m} + \frac{1}{2}\boldsymbol{\alpha}_{n+m}^{T}\left(\mathbf{H}_m^{+} - \mathbf{H}_m^{-}\right)\boldsymbol{\alpha}_{n+m} + \mathbf{b}_{n+m}^{T}\boldsymbol{\alpha}_{n+m} \\
&= W(\boldsymbol{\alpha}_{n+m}).
\end{aligned} \tag{25}$$

We can rewrite the auxiliary function in (24) as the sum of the second order convex functions $f_i(u_i)$ as:

$$F\left(\mathbf{u},\boldsymbol{\alpha}_{n+m}\right) = \sum_{i} f_i\left(u_i\right) - \frac{1}{2}\boldsymbol{\alpha}_{n+m}^{T}\bar{\mathbf{H}}_n^{-}\boldsymbol{\alpha}_{n+m} - \frac{1}{2}\boldsymbol{\alpha}_{n+m}^{T}\mathbf{H}_m^{-}\boldsymbol{\alpha}_{n+m}, \tag{26}$$

where:

$$\begin{aligned}
f_i\left(u_i\right) \;&= \frac{1}{2}\frac{[\bar{\mathbf{H}}_n^{+}\boldsymbol{\alpha}_{n+m}]_i}{[\boldsymbol{\alpha}_{n+m}]_i}u_i^2 + \frac{1}{2}\frac{[\mathbf{H}_m^{+}\boldsymbol{\alpha}_{n+m}]_i}{[\boldsymbol{\alpha}_{n+m}]_i}u_i^2 - [\bar{\mathbf{H}}_n^{-}\boldsymbol{\alpha}_{n+m}]_i\left[\boldsymbol{\alpha}_{n+m}\right]_i \log\frac{u_i}{\left[\boldsymbol{\alpha}_{n+m}\right]_i} \\
&-[\mathbf{H}_m^{-}\boldsymbol{\alpha}_{n+m}]_i\left[\boldsymbol{\alpha}_{n+m}\right]_i \log\frac{u_i}{\left[\boldsymbol{\alpha}_{n+m}\right]_i} + [\mathbf{b}_{n+m}]_i u_i.
\end{aligned} \tag{27}$$

By examining the second order partial derivative of f_i with respect to u_i, it is found that:

$$\frac{\partial^2 f_i}{\partial^2 u_i} = \frac{[\bar{\mathbf{H}}_n^{+}\boldsymbol{\alpha}_{n+m}]_i}{[\boldsymbol{\alpha}_{n+m}]_i} + \frac{[\bar{\mathbf{H}}_n^{-}\boldsymbol{\alpha}_{n+m}]_i[\boldsymbol{\alpha}_{n+m}]_i}{u_i^2} + \frac{[\mathbf{H}_m^{+}\boldsymbol{\alpha}_{n+m}]_i}{[\boldsymbol{\alpha}_{n+m}]_i} + \frac{[\mathbf{H}_m^{-}\boldsymbol{\alpha}_{n+m}]_i[\boldsymbol{\alpha}_{n+m}]_i}{u_i^2} > 0 \tag{28}$$

since for the negative vector $\boldsymbol{\alpha}_{n+m}$, vectors elements $[\bar{\mathbf{H}}_n^{-}\boldsymbol{\alpha}_{n+m}]_i$, $[\bar{\mathbf{H}}_n^{+}\boldsymbol{\alpha}_{n+m}]_i$, $[\mathbf{H}_m^{-}\boldsymbol{\alpha}_{n+m}]_i$ and $[\mathbf{H}_m^{+}\boldsymbol{\alpha}_{n+m}]_i$ are also negative and cannot all be simultaneously equal to zero. This implies that $f_i(u_i)$ is strictly positive in u_i and, consequently, the auxiliary function $F(\mathbf{u},\boldsymbol{\alpha}_{n+m})$ is the sum of strictly convex functions $f_i(u_i)$.

The minimizer $\boldsymbol{\alpha}'$ of $F(\mathbf{u},\boldsymbol{\alpha}_{n+m})$, can be determined by finding the minimum of each individual function $f_i(u_i)$ separately. To do so, we compute the first order partial derivative of each $f_i(u_i)$

with respect to u_i, set it equal to zero and solve for u_i:

$$\frac{\partial f_i}{\partial u_i} = \frac{[\bar{\mathbf{H}}_n^{+}\boldsymbol{\alpha}_{n+m}]_i}{[\boldsymbol{\alpha}_{n+m}]_i} u_i - \frac{[\bar{\mathbf{H}}_n^{-}\boldsymbol{\alpha}_{n+m}]_i[\boldsymbol{\alpha}_{n+m}]_i}{u_i} + \frac{[\mathbf{H}_m^{+}\boldsymbol{\alpha}_{n+m}]_i}{[\boldsymbol{\alpha}_{n+m}]_i} u_i - \frac{[\mathbf{H}_m^{-}\boldsymbol{\alpha}_{n+m}]_i[\boldsymbol{\alpha}_{n+m}]_i}{u_i} + [\mathbf{b}_{n+m}]_i = 0 \tag{29}$$

leading to the following multiplicative update rules:

$$u_i = \left(\frac{-[\mathbf{b}_{n+m}]_i - \sqrt{[\mathbf{b}_{n+m}]_i^2 + 4\left([\bar{\mathbf{H}}_n^{+}\boldsymbol{\alpha}_{n+m}]_i + [\mathbf{H}_m^{+}\boldsymbol{\alpha}_{n+m}]_i\right)\left([\bar{\mathbf{H}}_n^{-}\boldsymbol{\alpha}_{n+m}]_i + [\mathbf{H}_m^{-}\boldsymbol{\alpha}_{n+m}]_i\right)}}{2\left([\bar{\mathbf{H}}_n^{+}\boldsymbol{\alpha}_{n+m}]_i + [\mathbf{H}_m^{+}\boldsymbol{\alpha}_{n+m}]_i\right)} \right) [\boldsymbol{\alpha}_{n+m}]_i . \tag{30}$$

As can be observed from the applied SVM formulation and the resulting multiplicative updates, this iterative optimization rule enforces non-positivity constraints across all optimized Lagrange multipliers. However, the normal vector $\mathbf{w}_p$ is obtained subject to the constraints in (11), which involves the derivation of mixed sign Lagrange multipliers. As a result, we need to derive a multiplicative update that operates on α_i^p, when $y_i \neq p$ and on $C - \alpha_i^p$, when $y_i = p$. More precisely, we define the new variable:

$$\hat{\alpha}_i^p \triangleq \begin{cases} \alpha_i^p & \text{, if } y_i \neq p \\ C - \alpha_i^p & \text{, if } y_i = p \end{cases} \tag{31}$$

and express α_i^p in terms of $\hat{\alpha}_i^p$ as follows:

$$\alpha_i^p = \hat{\alpha}_i^p \left(1 - 2\delta_{y_i}^p\right) + C\delta_{y_i}^p . \tag{32}$$

Since α_i^p and $\hat{\alpha}_i^p$ are linearly related, minimizing the quadratic function $W(\hat{\boldsymbol{\alpha}})$ obtained by substituting (32) into (10) as:

$$W(\hat{\boldsymbol{\alpha}}) = \frac{1}{2}\hat{\boldsymbol{\alpha}}^T \hat{\mathbf{H}} \hat{\boldsymbol{\alpha}} + \hat{\mathbf{b}}^T \hat{\boldsymbol{\alpha}} \tag{33}$$

is equivalent to minimizing $W(\boldsymbol{\alpha})$. The coefficients $\hat{H}_{ij}$ and $\hat{b}_i$ correspond to the coefficients of the terms $\hat{\alpha}_i^p \hat{\alpha}_j^p$ and $\hat{\alpha}_i^p$ in (33), respectively. These coefficients are defined as:

$$\begin{aligned} \hat{H}_{ij} &= \left(1 - 2\delta_{y_i}^p\right)\left(1 - 2\delta_{y_j}^p\right) H_{ij} \\ \hat{b}_i^p &= b_i^p \left(1 - 2\delta_{y_i}^p\right) + C \sum_j \left(1 - 2\delta_{y_i}^p\right) \delta_{y_j}^p H_{ij}. \end{aligned} \tag{34}$$

By defining the matrices $\hat{\bar{\mathbf{H}}}_n$ and $\hat{\bar{\mathbf{H}}}_m$ from $\hat{\mathbf{H}}$ using the same approach as in (14) and then performing the decomposition in (15), we can obtain $\hat{\bar{\mathbf{H}}}_n^-$, $\hat{\bar{\mathbf{H}}}_n^+$, $\hat{\mathbf{H}}_m^-$ and $\hat{\mathbf{H}}_m^+$ and define a minimization problem in terms of these matrices, which is equivalent to the enriched minimization problem formulated in (16). Thus we propose the following set of multiplicative update rules which operates separately on Lagrange multipliers with different constraints:

$$u_i = \begin{cases} \left(\dfrac{-[\mathbf{b}_{n+m}]_i - \sqrt{[\mathbf{b}_{n+m}]_i^2 + 4\left([\bar{\mathbf{H}}_n^+ \boldsymbol{\alpha}_{n+m}]_i + [\mathbf{H}_m^+ \boldsymbol{\alpha}_{n+m}]_i\right)\left([\bar{\mathbf{H}}_n^- \boldsymbol{\alpha}_{n+m}]_i + [\mathbf{H}_m^- \boldsymbol{\alpha}_{n+m}]_i\right)}}{2\left([\bar{\mathbf{H}}_n^+ \boldsymbol{\alpha}_{n+m}]_i + [\mathbf{H}_m^+ \boldsymbol{\alpha}_{n+m}]_i\right)} \right) [\boldsymbol{\alpha}_{n+m}]_i, \\ \text{if} \quad y_i \neq p. \\ \left(\dfrac{-[\hat{\mathbf{b}}_{n+m}]_i + \sqrt{[\hat{\mathbf{b}}_{n+m}]_i^2 + 4\left([\hat{\bar{\mathbf{H}}}_n^+ \hat{\boldsymbol{\alpha}}_{n+m}]_i + [\hat{\mathbf{H}}_m^+ \hat{\boldsymbol{\alpha}}_{n+m}]_i\right)\left([\hat{\bar{\mathbf{H}}}_n^- \hat{\boldsymbol{\alpha}}_{n+m}]_i + [\hat{\mathbf{H}}_m^- \hat{\boldsymbol{\alpha}}_{n+m}]_i\right)}}{2\left([\hat{\bar{\mathbf{H}}}_n^+ \hat{\boldsymbol{\alpha}}_{n+m}]_i + [\hat{\mathbf{H}}_m^+ \hat{\boldsymbol{\alpha}}_{n+m}]_i\right)} \right) [\hat{\boldsymbol{\alpha}}_{n+m}]_i, \\ \text{if} \quad y_i = p. \end{cases} \tag{35}$$

The proposed updates decrease $W(\boldsymbol{\alpha}_{n+m})$ by enforcing the non-positivity constraint on α_i^p, and at the same time, drive $\hat{\alpha}_i^p$ towards zero, thus increasing those Lagrange multipliers towards C.

3.3. Convergence Analysis of the Proposed Multiplicative Update Rules

We can show that the proposed updates converge to the global minimum by examining the derivative of the objective function. As will be shown, the derived multiplicative updates take steps proportional to the negative of the objective function gradient, moving each element $[\boldsymbol{\alpha}_{n+m}]_i$ to a direction opposite to that of its partial derivative. Recalling the decomposition of the objective function we have performed in (22), the gradient of $W(\boldsymbol{\alpha}_{n+m})$ can be similarly decomposed in terms of contributions of these five parts as follows:

$$\begin{aligned} c_i &= \frac{\partial W_c}{\partial \left[\boldsymbol{\alpha}_{n+m}\right]_i} = \left[\bar{\mathbf{H}}_n^{+}\boldsymbol{\alpha}_{n+m}\right]_i \leq 0 \quad , \quad & d_i &= \frac{\partial W_d}{\partial \left[\boldsymbol{\alpha}_{n+m}\right]_i} = \left[\bar{\mathbf{H}}_n^{-}\boldsymbol{\alpha}_{n+m}\right]_i \leq 0 \\ e_i &= \frac{\partial W_e}{\partial \left[\boldsymbol{\alpha}_{n+m}\right]_i} = \left[\mathbf{H}_m^{+}\boldsymbol{\alpha}_{n+m}\right]_i \leq 0 \quad , \quad & f_i &= \frac{\partial W_f}{\partial \left[\boldsymbol{\alpha}_{n+m}\right]_i} = \left[\mathbf{H}_m^{-}\boldsymbol{\alpha}_{n+m}\right]_i \leq 0 \\ b_i &= \frac{\partial W_b}{\partial \left[\boldsymbol{\alpha}_{n+m}\right]_i} = \left[\mathbf{b}_{n+m}\right]_i \geq 0. \end{aligned} \tag{36}$$

Since $\frac{\partial W}{\partial [\boldsymbol{\alpha}_{n+m}]_i} = c_i - d_i + e_i - f_i + b_i$ it arises directly that the respective update rule decreases $[\boldsymbol{\alpha}_{n+m}]_i$ if $\frac{\partial W}{\partial [\boldsymbol{\alpha}_{n+m}]_i} > 0$ and increase it if $\frac{\partial W}{\partial [\boldsymbol{\alpha}_{n+m}]_i} < 0$. Similarly, the opposite observation is valid for the update rule operating on $[\hat{\boldsymbol{\alpha}}_{n+m}]_i$.

3.4. Reoptimization Using a Warm-Start Strategy

Warm-start strategy is a popular approach in reoptimization problems where small perturbations in the form of the initial optimization problem occur. In the literature warm-start algorithms were found to efficiently handle two main cases of perturbation [34, 35]. The first considers that the initial optimization problem is modified by adding new constraints and/or new variables which is the actual case in our considered incremental training scenario, while the other, regards that the reoptimization problem has exactly the same constraints and variables with respect to the initial one but the underlying data are perturbed. In [36] the authors have investigated the extend of perturbation with respect to the problem size and proposed conditions in order to determine, whether the previous optimum solution is a convenient starting point to warm-start the optimization process for the perturbed problem. Theoretical analysis in [36] indicated the superiority of warm-start algorithms, to attain reduced computational effort compared with cold-start methods (i.e. performing optimization from scratch) especially for problems having undergone small degree of perturbation. These theoretical indications have been verified by extensive experimental results in [37].

The main motivation for applying a warm-start strategy for the optimization of the enriched problem, is the expectation that two closely related optimization problems, such as the batch training over a base training set $\mathcal{X}_n$ and the incremental training over an augmented training set $\mathcal{X}_{n+m}$), should, in general, share similar characteristics. Considering the classification task, this can be interpreted as the expectation that the new decision surface of the SVM classifier will have minimal disturbances with respect to its previous form, when a small number of new training samples is added to a base training dataset. Consequently, warm-start strategy can be applied as a mean to exploit the information gained during training over the base dataset. As a result, the proposed multiplicative update rules are expected to converge faster to the optimal solution, i.e. within fewer iterations, when starting from the previous global minimizer and initializing only the Lagrange multipliers $\boldsymbol{\alpha}_s$ related to the new training samples $\mathcal{X}_m$, compared to starting from an arbitrary initialization point when performing training from scratch.

Consider the minimization problem regarding the base n training samples:

$$\begin{aligned} &\min_{\boldsymbol{\alpha}_n} \quad W(\boldsymbol{\alpha}_n) = \frac{1}{2}\boldsymbol{\alpha}_n^T \left(\bar{\mathbf{H}}_n^+ - \bar{\mathbf{H}}_n^-\right) \boldsymbol{\alpha}_n + \mathbf{b}_n^T \boldsymbol{\alpha}_n \\ &\text{subject to the linear constraints defined in (11),} \end{aligned} \tag{37}$$

where $\bar{\mathbf{H}}_n^+$,$\bar{\mathbf{H}}_n^- \in \mathbb{R}_+^{nk \times nk}$, $\boldsymbol{\alpha}_n \in \mathbb{R}^{nk}$. Let $\boldsymbol{\alpha}_{n,o}$ be the optimal solution containing the Lagrange multipliers that minimize the objective function $W(\boldsymbol{\alpha}_n)$. We use the notation $\mathcal{D}_n = \{\boldsymbol{\alpha}_{n,o}, \bar{\mathbf{H}}_n^+, \bar{\mathbf{H}}_n^-, \mathbf{b}_n\}$ to denote the data instance related to the primal minimization problem applied to the n base training pairs found in $\mathcal{X}_n$. In order to perform incremental training and find the optimal solution of the augmented problem summarized by the objective function in (16) the data instance related to the primal minimization problem $\mathcal{D}_n$ is required in order to warm-start the optimization procedure. The reoptimization process using the warm-start strategy is outlined in Algorithm 1.

Algorithm 1 SVM reoptimization using warm-start algorithm

```
Require: Data instance $\mathcal{D}_n = \{\boldsymbol{\alpha}_{n,o}, \bar{\mathbf{H}}_n^+, \bar{\mathbf{H}}_n^-, \mathbf{b}_n\}$ evaluated from solving the base optimization problem.
Read $\boldsymbol{\alpha}_{n,o}$, $\bar{\mathbf{H}}_n^+$, $\bar{\mathbf{H}}_n^-$ and $\mathbf{b}_n$
Compute $\mathbf{b}_s$ and $\mathbf{H}_m$ via equations (3) and (14), respectively.
Obtain the nonnegative matrices $\mathbf{H}_m^+$ and $\mathbf{H}_m^-$.
Compute matrices $\hat{\bar{\mathbf{H}}}_n^-$, $\hat{\bar{\mathbf{H}}}_n^+$, $\hat{\mathbf{H}}_m^-$ and $\hat{\mathbf{H}}_m^+$.
Obtain $\mathbf{b}_{n+m} = \begin{bmatrix} \mathbf{b}_n \\ \mathbf{b}_s \end{bmatrix}$
Initialize $\boldsymbol{\alpha}_s = -\mathbf{1}$
Obtain $\boldsymbol{\alpha}_{n+m} = \begin{bmatrix} \boldsymbol{\alpha}_{n,o} \\ \boldsymbol{\alpha}_s \end{bmatrix}$
repeat
    for $i = 1$ up to $n + m$ do
        if $\Delta\alpha_i \geq \varepsilon$ then
            if $y_i \neq p$ then
                $[\boldsymbol{\alpha}_{n+m}]_i' = \left( \frac{-[\mathbf{b}_{n+m}]_i - \sqrt{[\mathbf{b}_{n+m}]_i^2 + 4\left([\bar{\mathbf{H}}_n^+\boldsymbol{\alpha}_{n+m}]_i + [\mathbf{H}_m^+\boldsymbol{\alpha}_{n+m}]_i\right)\left([\bar{\mathbf{H}}_n^-\boldsymbol{\alpha}_{n+m}]_i + [\mathbf{H}_m^-\boldsymbol{\alpha}_{n+m}]_i\right)}}{2\left([\bar{\mathbf{H}}_n^+\boldsymbol{\alpha}_{n+m}]_i + [\mathbf{H}_m^+\boldsymbol{\alpha}_{n+m}]_i\right)} \right) [\boldsymbol{\alpha}_{n+m}]_i$
            else
                $[\hat{\boldsymbol{\alpha}}_{n+m}]_i' = \left( \frac{-[\hat{\mathbf{b}}_{n+m}]_i + \sqrt{[\hat{\mathbf{b}}_{n+m}]_i^2 + 4\left([\hat{\bar{\mathbf{H}}}_n^+\hat{\boldsymbol{\alpha}}_{n+m}]_i + [\hat{\mathbf{H}}_m^+\hat{\boldsymbol{\alpha}}_{n+m}]_i\right)\left([\hat{\bar{\mathbf{H}}}_n^-\hat{\boldsymbol{\alpha}}_{n+m}]_i + [\hat{\mathbf{H}}_m^-\hat{\boldsymbol{\alpha}}_{n+m}]_i\right)}}{2\left([\hat{\bar{\mathbf{H}}}_n^+\hat{\boldsymbol{\alpha}}_{n+m}]_i + [\hat{\mathbf{H}}_m^+\hat{\boldsymbol{\alpha}}_{n+m}]_i\right)} \right) [\hat{\boldsymbol{\alpha}}_{n+m}]_i$
            end if
            Compute $\Delta\alpha_i = |[\boldsymbol{\alpha}_{n+m}]_i' - [\boldsymbol{\alpha}_{n+m}]_i|$ or $\Delta\alpha_i = |[\hat{\boldsymbol{\alpha}}_{n+m}]_i' - [\hat{\boldsymbol{\alpha}}_{n+m}]_i|$
        end if
    end for
until $\nexists\ \Delta\alpha_i \geq \varepsilon$
```

4. Experimental Study

In this section, we provide experimental validation regarding the computational efficiency of the proposed incremental SVM training algorithm. To comply with the standard terminology used in the related literature, throughout this Section we will use the term "online" training when referring to cases where the increment step size equals to one and "incremental" training when the training set is augmented by adding more than one new training data samples. We are primarily interested in comparing the computational cost of solving the reoptimization problem using an online/incremental training approach combined with a warm-start algorithm, with the cost of retraining the classifier from scratch, for the augmented training dataset $\mathcal{X}_{n+m}$. We investigate these settings that, on one hand, significantly downsize the required computational cost for updating the SVM classifier and, on the other hand, maintain the same classification accuracy achieved by brute force retraining from scratch. It should be mentioned that, whenever we are performing online or incremental training, it is always required that the SVM classifier has been previously trained over a base dataset $\mathcal{X}_n$ of n initial training samples, which is subsequently augmented by adding m new training pairs $\mathcal{X}_m$. Moreover, each time we incrementally update the SVM classifier, it is assumed that the data instance $\mathcal{D}_n = \{\boldsymbol{\alpha}_{n,o},\ \bar{\mathbf{H}}_n^+\ ,\ \bar{\mathbf{H}}_n^-,\ \mathbf{b}_n\}$ related to the previous optimization problem is available. Apart from the proposed multiplicative update rules, we can derive in a straightforward manner a batch training model by modifying the update rules in (35) setting $[\mathbf{H}_m^+\boldsymbol{\alpha}_{n+m}]_i = [\mathbf{H}_m^-\boldsymbol{\alpha}_{n+m}]_i = [\hat{\mathbf{H}}_m^+\hat{\boldsymbol{\alpha}}_{n+m}]_i = [\hat{\mathbf{H}}_m^-\hat{\boldsymbol{\alpha}}_{n+m}]_i = 0$, $\forall i$. This batch training approach is used as a test bed in order to experimentally verify that the proposed incremental training model converges to the same optimal solution and thus the obtained classification accuracy is also maintained.

Experiments have been conducted on three multiclass datasets, namely, the Satimage, the

Optical Recognition of Handwritten Digits (ORHD) and the Letter databases and on the two-class Mushroom collection. Each dataset has different content and size and were all obtained from the UCI Repository of machine learning databases [38]. Moreover, experiments for the frontal face view recognition problem have been conducted using the XM2VTS database [39]. Table 1 summarizes the characteristics of each examined dataset. Although the implemented SVM formulation has been designed for multiclass classification problems, we have selected to include in the experimental study two two-class datasets, in order to examine variations in the convergence process, since the number of Lagrange multipliers being optimized by the proposed method is proportional to the number of classes. Those collections that contain entries of qualitative attribute characteristics in non arithmetic form, such as the Mushroom and the Letter datasets, were first transformed to numerical form. Moreover, all training and testing data were randomly ordered and normalized so as to be in the $[-1, 1]$ range with zero mean. Finally, since for the Mushroom collection there is no available test set, we conducted a five-fold cross validation on the entire dataset in order to measure the mean achieved classification accuracy over all five folds.

Table 1: Characteristics of the examined datasets

Dataset	Training samples	Test samples	Classes	Features Dimensionality
Mushroom	5009	–	2	22
Satimage	4435	2000	6	36
ORHD	3823	1797	10	64
Letter	16000	4000	26	16
XM2VTS	3431	3431	2	1200

Since our main goal is to provide computational efficiency by achieving faster convergence to the optimal solution, when the underlying training data collection changes dynamically over time, we have chosen to train the SVM classifier on the examined datasets using only Radial Basis Functions (RBF) as kernel functions $K(\mathbf{x}_i, \mathbf{x}_j) = e^{\gamma||\mathbf{x}_i - \mathbf{x}_j||^2}$ considering different spreads of the Gaussian in the set $\gamma = \{2^{-8}, 2^{-7}, ..., 2^{7}, 2^{8}\}$, which generates a non-negative kernel. Moreover, in the test phase we applied the spread value γ for the Gaussian function that achieved the highest accuracy rate at the validation procedure. That is, we search for that value of γ, within the specified set that achieves the highest recognition rate on the training set.

Whenever we report the measured classification accuracy rates in the experimental results, we also provide the related γ parameter value. The rationale behind our decision to consider across all our experiments only the RBF kernel function is twofold. Firstly, the search space for parameter selection is significantly reduced and secondly, direct comparisons with the classification accuracy rates achieved by the proposed online training method are feasible, since the highest achieved classification accuracy rates using the multiclass SVM formulation by Crammer and Singer for two of the examined datasets and for the RBF kernel, are reported in [40].

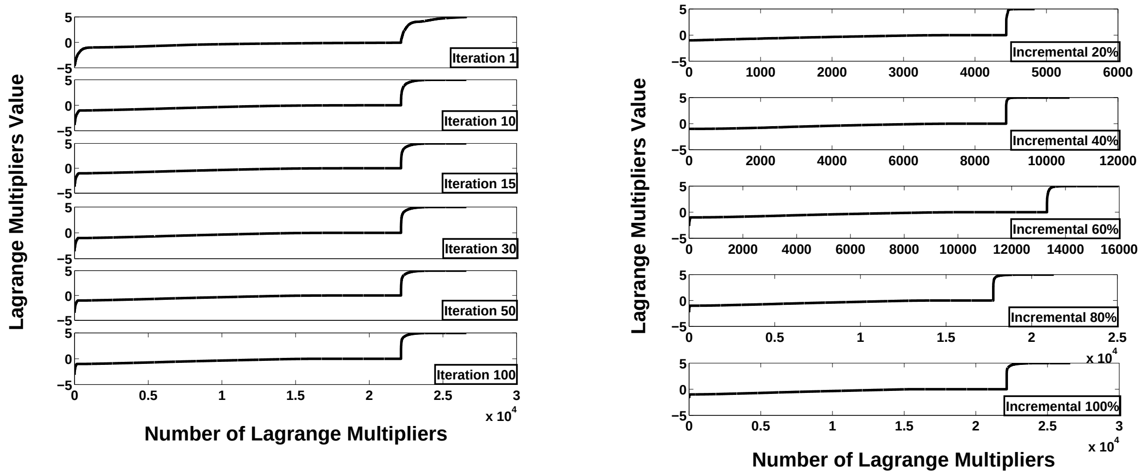
4.1. Multiplicative Update Rules Convergence Results

Here we demonstrate the optimization convergence of the Lagrange multipliers within the feasible optimization bounds using the proposed multiplicative update rules. First, we show convergence results obtained from the batch training model. Then we present convergence curves for the variables under optimization during the incremental training, while sequentially enriching the training set using parts of the complete dataset and show that they reach to the same optimal solution with the one obtained after batch processing the same training data. Regarding the provided plots of the optimized variables it should be noted that since we train the classifier with the RBF kernel the obtained Lagrange multipliers take values either close to zero or close to C and as a result the demonstrated plots do not have the spiky form as those presented in [30]. This is due to the fact that Lagrange multipliers corresponding to support vectors take values close to C, while the rest are attenuated close to zero, since Gaussian RBF SVMs of sufficiently small width can classify an arbitrarily large number of training points correctly [3].

It is important to terminate the optimization algorithm, when no significant alternations in the updated Lagrange multipliers values are observed. Terminating the optimization process before it starts oscillating around a convergence point with no improvement in the classification accuracy, significantly reduces the required computation cost. In our implementation, we stop the optimization process for those Lagrange multipliers whose value changes for less than 2×10^{-2} in an update, since we have strong evidence that they have reached convergence.

Figures 1 and 2 illustrate the convergence of the optimized Lagrange multipliers for the batch and the incremental training algorithms on the Satimage and Mushroom datasets, where cost parameter C measuring the penalty per unit slack, has been set to 5 for both experiments. The sub graphs demonstrate the convergence progress of the optimized Lagrange multipliers, where the horizontal axes index the Lagrange multipliers involved in each optimization step sorted in ascending order according to their value, while the vertical axes show their values. In order to verify that both the batch and the incremental training approaches converge to the same optimal solution we have applied the following procedure. First, we applied the batch training model using the entire training set of each collection. Second, in order to simulate an incremental training scenario we have split each dataset into five almost equally sized subsets, each containing approximately 20% of the total available training samples. We then applied the proposed incremental training method starting by training the classifier using one of these subsets and performing 5 steps by sequentially augmenting the training set by adding each time one of the remaining subsets.

In more detail, Figures 1(a) and 2(a) demonstrate the convergence progress of the optimized

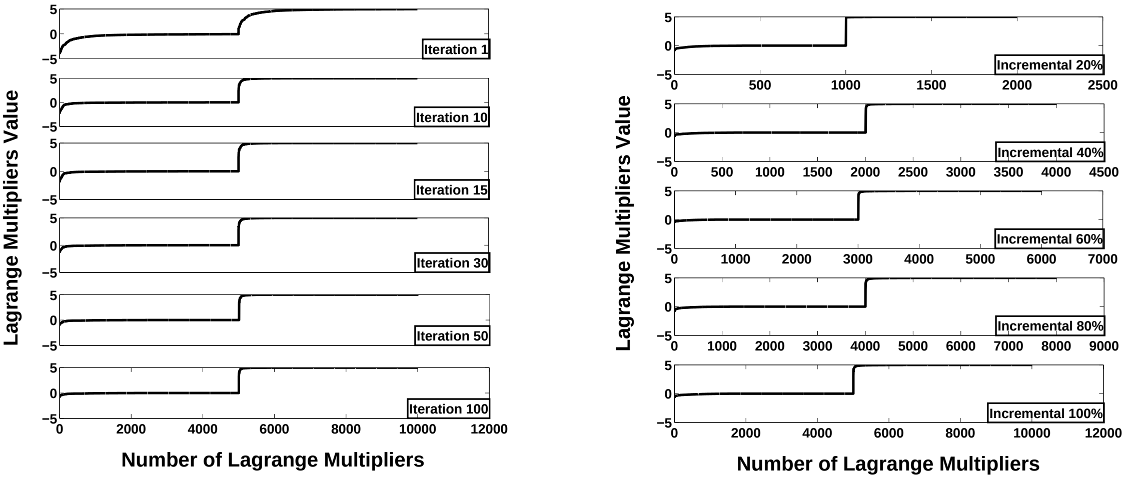


(a) (b)

Figure 1: Lagrance multiplier convergence for the Satimage dataset; (a) The solution found using the batch training approach for different iterations. (b) The solution found by performing incremental training each time adding 20% of the samples.

Lagrange multipliers during batch training. Each sub plot presents graphically the solution found after a number of iterations. Figures 1(b) and 2(b) present graphically the optimal solution found using the proposed incremental training method, while we sequentially augmented the training set size using each one of the five training subsets. It should be mentioned that each time we performed incremental training the optimal solution found at the previous step was used to warm-start the new optimization process, while the Lagrange multipliers corresponding to unseen training samples were uniformly initialized to $[\boldsymbol{\alpha}_s]_i = -1, \quad i = n+1 \ldots n+m$.

As it can be observed from the graphs, the proposed dual set of multiplicative update rules enforces the Lagrange multipliers of both datasets to converge inside the feasible optimization boundaries in the range $[-C, C]$. Except from the obvious visible resemblance between the sketched optimal solutions found using the batch and the incremental training models shown in the last plots in Figures 1 and 2, in order to experimentally verify that the optimized variables converge to the



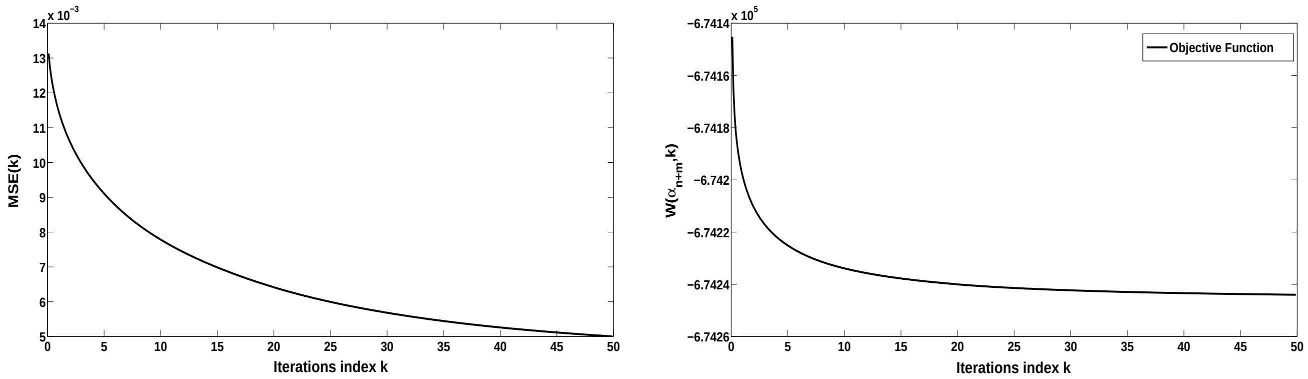
(a) (b)

Figure 2: Lagrange multiplier convergence for the two-class Mushroom collection; (a) The solution found using the batch training approach for different iterations. (b) The solution found by performing incremental training each time adding 20% of the samples.

same optimal solution for both the batch and the incremental training models, we have measured the Mean Square Error (MSE) between the two solutions. More precisely, we measured the MSE between each temporary solution found, while seeking for the optimal one, during incremental training the classifier where the training set has been augmented by adding the last training samples subset (the final 20% of the samples), thus resulting to the complete dataset, with the optimal solution obtained after performing batch training using the same complete dataset. Since, the solution of each optimization problem is determined by the values of the Lagrange multipliers, we evaluate the MSE at the k-th iteration as:

$$MSE(k) = \frac{1}{n} \sum_{i}^{n} \left([\boldsymbol{\alpha}_{n,o}]_i - [\boldsymbol{\alpha}_{n,k}]_i \right)^2 \tag{38}$$

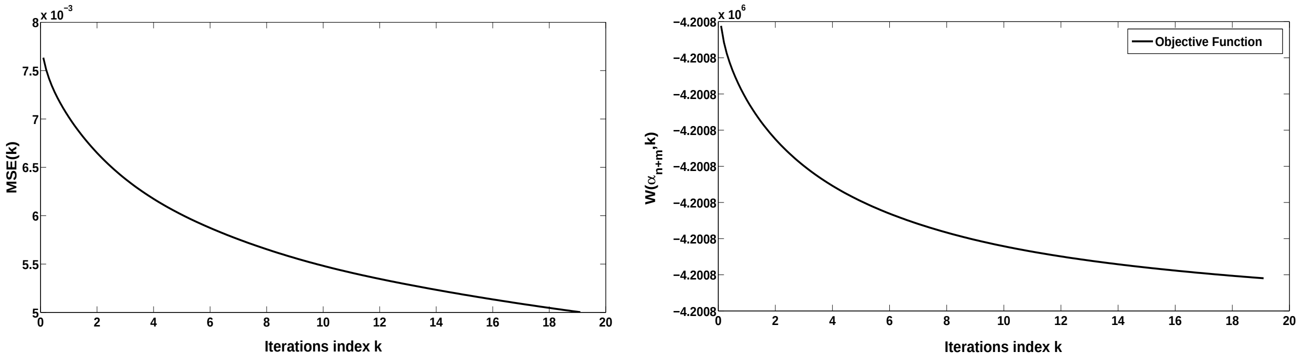
where $\boldsymbol{\alpha}_{n,o}$ corresponds to the optimal Lagrange multipliers vector found after batch training and $\boldsymbol{\alpha}_{n,k}$ denotes a temporary solution found after the k-th iteration of the proposed multiplicative update rules during incremental training. Figures 3(a) and 4(a) show the computed MSE for the Satimage and Mushroom collections, respectively. As it can be observed faster convergence is attained on the two-class Mushroom collection, since the proposed multiplicative update rules required 19 iterations compared with 50 iterations required in order the algorithm to reach convergence on the Satimage dataset. Figures 3(b) and 4(b) show the minimization of the objective function $W\left(\boldsymbol{\alpha}_{n+m}\right)$ formulated in (16), after each iteration of the proposed set of update rules during incremental training. As it can be seen the proposed updates decrease the objective function value while seeking for the global minimum. Moreover, as it can be observed continuing the optimization process beyond this point does not provide any significant gain regarding the minimization of the objective function value.



(a)

(b)

Figure 3: a) Computed MSE(k) and b) Objective function value $W(\boldsymbol{\alpha}_{n+m}, k)$ versus the iterations index k of the proposed update rules during incremental training the classifier where the training set has been augmented by adding the final training samples subset, thus resulting to the complete dataset. The results correspond to the Satimage dataset.



(a)

(b)

Figure 4: a) Computed MSE(k) and b) Objective function value $W(\boldsymbol{\alpha}_{n+m}, k)$ versus the iterations index k of the proposed update rules during incremental training the classifier where the training set has been augmented by adding the final training samples subset, thus resulting to the complete dataset. The results correspond to the Mushroom collection.

4.2. Computational Complexity Comparison Against Current State-of-the-art Multiclass SVM Solvers

In this subsection we compare our method against LaRank, in terms of computational complexity, since both methods optimize the same dual problem. For completeness in this comparison we have also included SMO method, which is a state-of-the-art algorithm for fast batch SVM training and the iterative optimization algorithm proposed in [23] solving the considered dual multiclass problem, which will be referred below as MCSVM algorithm.

A common characteristic across all the other considered optimizers is that they explicitly rely on the dual objective function gradient to find the optimum solution. Among the examined methods SMO and MCSVM exploit the full gradient information i.e. consider the dual objective functions gradient with respect to the Lagrange multipliers associated to every sample in the training set, while LaRank exploits this information only partially, arguing that computing and storing gradients that do not correspond to support patterns is computationally inefficient, since the optimum solution is determined only by a fraction of the kernel matrix that involves only the support patterns.

A common way to express the complexity of an algorithm is by using the big O notation [41], which will also facilitate our comparison. The attempted computational complexity comparison is performed at a single parameter optimization level, thus estimating the cost for finding the optimal value of a single Lagrange multiplier. The most expensive operation in the proposed multiplicative updates is the computation of the i-th element of vectors $[\bar{\mathbf{H}}_n^+ \boldsymbol{\alpha}_{n+m}]$, $[\mathbf{H}_m^+ \boldsymbol{\alpha}_{n+m}]$, $[\bar{\mathbf{H}}_n^- \boldsymbol{\alpha}_{n+m}]$ and $[\mathbf{H}_m^- \boldsymbol{\alpha}_{n+m}]$. However, since the Hessian matrices are too large to be stored (i.e. of dimensionality $lk \times lk$, where l is the number of training samples and k the number of classes), one can only calculate their i-th row when applying the updates. Recall, that Hessian matrices are sparse and initiated from the kernel matrix by performing a Kronecker multiplication with an appropriate dimensional identity matrix. For each kernel matrix element evaluation given that d is the dimensionality of the input feature vectors, $O(d)$ is required. Subsequently, calculating its i-th row takes $O(ld)$. Thus, the evaluation of the i-th element involving the multiplication of the i-th row with the column vector $\boldsymbol{\alpha}_{n+m}$, requires $O(l)$. Finally, since usually the feature vector dimensionality is much larger than the number of classes, that is $d \gg k$, the overall cost for updating a single Lagrange multiplier is of order $O\left(ld\right) + r_1 \times O\left(lk\right)$, where r_1 is the average number of iterations required to reach convergence.

SMO requires $O(\tilde{n}d)$ in order to compute the kernel matrix elements involved in the Lagrange multipliers update, where $\tilde{n}$ is the average number of candidate support vectors. As support vector candidates are considered those samples that violate a thresholded error function and thus, are amenable to update. Additionally, SMO involves in each update rule the computation of the prediction error generated by the updated samples. This operation requires $O(\tilde{n}k)$. Thus, the total cost for updating a single Lagrange multiplier is of order $O(\tilde{n}d) + r_2 \times O\left(\tilde{n}k\right)$, where r_2 is the average number of iterations required by SMO in order to converge. Typically the number of candidate support vectors is much larger than that of the actual ones ($\tilde{n} > n$). In other words, effectively reducing $\tilde{n}$ results in significant computational gain on the performance of SMO. This is addressed by LaRank algorithm which operates on support patterns, identified as those samples whose associated Lagrange coefficient in non zero for some class label. LaRank algorithm involves three different optimization routines which are invoked according to a computed probability. These strategies select for optimization those Lagrange multipliers that correspond either to a new unseen training sample or to an already identified support pattern and those two class labels that yield the maximum and the minimum derivative of the dual objective function. Consequently, LaRank requires $O(nd)$ to compute a single row of the kernel matrix where n is the number of support patterns with $n < \tilde{n} < l$. The derivative $g_i(y)$ of the dual objective function with respect to the Lagrange multiplier α_i^y associated with sample $\mathbf{x}_i$ and class y, is given by:

$$g_i(y) = (y - y_i) - \sum_{j=1}^{n} \alpha_j^y \mathbf{K}(\mathbf{x}_i, \mathbf{x}_j). \tag{39}$$

Consequently, the computational cost for evaluating the derivative with respect to the optimized Lagrange multiplier is $O(n)$ and finally the total computational cost is $O(nd) + r_3 \times O(n)$, where r_3 is the average number of iterations required by LaRank to reach convergence.

On the other hand, MCSVM algorithm decomposes the quadratic optimization problem into smaller sub problems and iteratively optimizes a single Lagrange multiplier. MCSVM learning resembles the SMO algorithm, although it optimizes a single variable at each round, in contrary to SMO which performs optimization in a pairwise manner. To do so, the algorithm selects a working set and chooses for optimization k variables associated with the same training sample in the working set. The involved update rule similarly with our approach, requires the evaluation of a single row of the kernel matrix considering the pairwise similarities between all the training samples and the sample under consideration. This operation requires $O(ld)$. The computed kernel row is subsequently multiplied by the Lagrange multipliers vector an operation that takes $O(l)$. We should also note that the algorithm needs to sort k values on each iteration which may become costly when k is large. The considered sorting algorithm as specified by the authors requires $O(klogk)$ to sort k values. Thus, the total computational cost of MCSVM is $O(ld) + r_4 \times (O(l) + O(k \log k))$

Finally, regarding the average number of iterations required by each examined method in order to reach convergence, based on empirical observations LaRank and our approach both require a few tens of iterations, while SMO and MCSVM require a few hundreds. Table 2 summarizes the estimated computational complexity of each examined method. According to the performed computational complexity analysis the required training time by LaRank is significantly smaller compared to that required by our approach, which is mainly attributed to the reduced training set size that is used by LaRank. More precisely, the estimated computational complexity of the LaRank training process is proportional to the number of support patterns, in contrary to that of our approach which scales proportionally to the number of training samples.

Table 2: Computational complexities of the examined methods

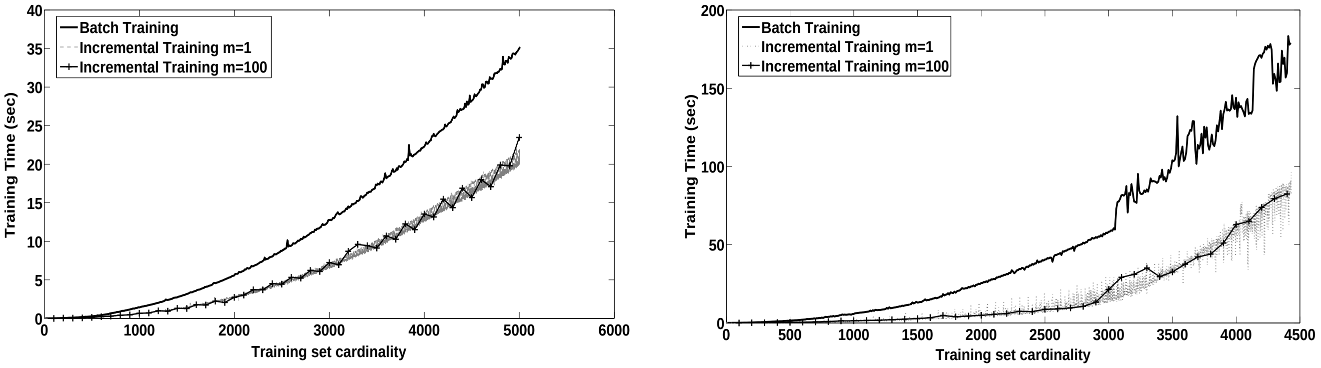
Method	Complexity
SMO	$O(\tilde{n}d) + r_2 \times O(\tilde{n}k)$
Proposed Method	$O(ld) + r_1 \times O(l)$
LaRank	$O(nd) + r_3 \times O(n)$
MCSVM	$O(ld) + r_4 \times (O(l) + O(k \log k))$

4.3. Computational Cost Comparison Between Online/Incremental and Batch Training

In order to examine the computational efficiency of the proposed incremental training method, we considered the case where the training set is sequentially augmented and we compare the computation time required in order to train the SVM classifier using the conventional batch training model with the one that is required by the proposed method. More precisely, for the online/incremental training approach, we have measured the required training time for the set $\mathcal{X}_{n+m}$ starting from the already known optimal solution for the set $\mathcal{X}_n$ and considering two different increment size steps $m = 1$ and $m = 100$ (i.e. set $\mathcal{X}_m$ has 1 or 100 training samples, respectively). For the batch training counterpart, we measured the time needed for training from scratch using $\mathcal{X}_{n+m}$.

Figures 5 and 6 demonstrate the recorded training time until convergence is reached, for both the batch training model and the online/incremental training approaches with respect to the number of training samples on each of the examined collections. The horizontal axis shows the cardinality of the training set $\mathcal{X}_{n+m}$. We assume that the optimal solution for the training set $\mathcal{X}_n$ is known and available to warm-start online or incremental training for the augmented training set $\mathcal{X}_{n+m}$. It is clearly seen that both the online and the incremental training algorithms are always much faster than batch training across all examined datasets.

To better illustrate the advantages of incremental training, consider two different training scenarios, where we have trained the SVM classifier over 3000 training samples of the Mushroom dataset ($n = 3000$) and we want to enrich the training set by including, in one case, a single new sample ($m = 1$) and in the other, 100 new samples ($m = 100$). Retraining the SVM classifier from scratch over $n + m = 3001$ samples requires 12.76 seconds, while for $n + m = 3100$ samples 13.57 seconds, whereas online/incremental training using the proposed algorithm requires 6.44 and 6.96 seconds, to incorporate the 1 or 100 new samples, respectively. In other words, the



(a)

(b)

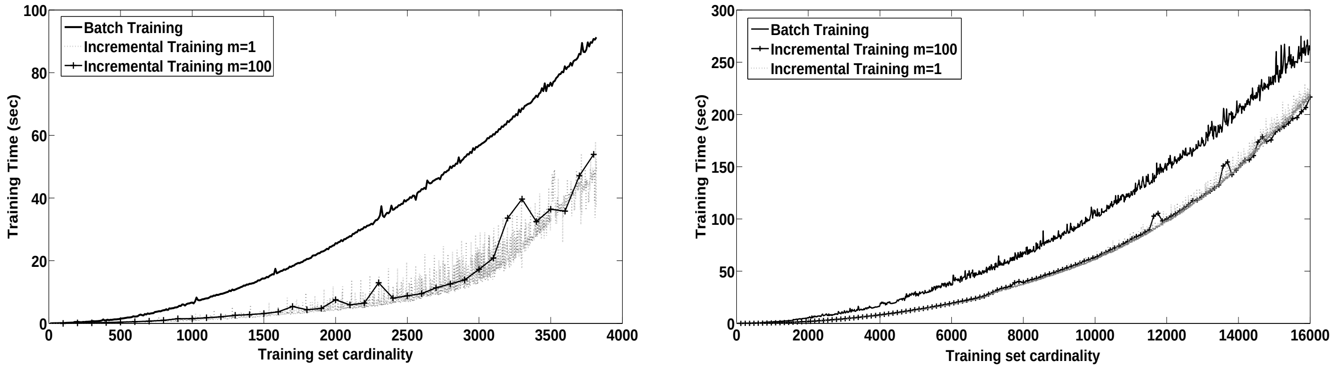
Figure 5: Computational time for online/incremental and batch training on the same training data measured over: a) the Mushroom and b) the Satimage datasets.

computational gain between the two approaches for the two examined scenarios is 6.32 and 6.61 seconds, respectively.

We have also measured the average, over all sample sizes, computation time gain on each dataset for the two examined increment step sizes $m = 1$ and $m = 100$ as:

$$\text{Average Time Gain} = \frac{1}{n}\sum_{i=1}^{n}(\text{Batch Training Time for } \mathcal{X}_i - \text{Incremental Training Time for } \mathcal{X}_i)$$

Table 3 summarizes the obtained gain results, as a percentage of the required time by the batch training model across the various examined collections for both applied increment step sizes. As it can be seen the higher is the feature vectors dimensionality and the less the number of involved data classes and the training set cardinality, the higher is the achieved computational gain. Consequently, the highest computational gain was attained in the ORHD dataset, which is a database of such characteristics. In contrary, the smallest improvement was attained in the Letter dataset, which is a large scale data collection with its samples partitioned into many classes (i.e $l = 16000$ and $k = 26$), while feature sample vectors dimensionality is small ($d = 16$).



(a) (b)

Figure 6: Computational time for online/incremental and batch training on the same training data measured over: a) the Optical Recognition of Handwritten Digits and b) the Letter datasets.

Table 3: Computational time reduction percentage

	Increment Step Size	
Dataset	$m = 1$	$m = 100$
Mushroom	44.62%	41.47%
Satimage	65.28%	60.02%
ORHD	70.64%	61.42%
Letter	33.13%	31.19%

4.4. Online Learning Using a Small Fraction of the Complete Training Set

For our third experiment we have exclusively considered online learning, where the underlying training set is sequentially enriched by adding a single new example, the classifier is adjusted such as to learn the additional information and its efficacy is subsequently assessed by predicting the class labels of the same standard testing sample set. Our aim is to investigate the behavior of the classifier, with respect to the classification accuracy, as the number of training samples increases and to derive useful insights regarding the effect of the new samples to the decision surface.

To perform online learning we initially trained the classifier by randomly selecting a single training sample from each class in order to define the initial decision hyperplanes. Subsequently, and as long as new training samples sequentially arrive, we performed online training and tested the classifier on the available standard testing set. To train in an online manner, a single unseen training sample was randomly selected and fed to the proposed algorithm which exploited the optimal solution found during the previous optimization to warm-start the new training process.

As can be observed from the provided graphs in Figure 7, for both the Mushroom and the Satimage datasets there exist some specific training samples that represent well the test data and, therefore, when included in the training set, boost the classification performance. The same observation regarding the Mushroom dataset has been reported in [14], where it has been shown that a carefully selected subset of around 100 samples from the complete training set is adequate in order to train well the SVM classifier and achieve 100% classification accuracy rate. On the other hand, as shown in Figure 8, the recognition rate for the Letter and the ORHD datasets increases smoothly, while we augment the training set by adding each time a single new sample.

The highest achieved recognition rate using the online training model is 100% for the Mushroom dataset with a Gaussian spread $\gamma = 2^{-6}$, 91.35% for the Satimage collection with $\gamma = 2^{3}$ and 98.5% and 93.66% with Gaussian spread $\gamma = 2^{-7}$ and $\gamma = 2^{-2}$ for the ORHD and Letter datasets, respectively. Although superiority on classification accuracy is not our primary concern, for completeness, we compare our method on the examined datasets against current state-of-the-art batch and online trained multiclass classifiers on accuracy. In this comparison we have considered DAG SVMs [19], the multiclass counterpart of the Weston and Watkins SVM classifier [21], MCSVM method and the LaRank algorithm. The highest measured classification accuracy rates achieved by each method in each examined database are summarized in Table 4. As it can be seen, with the exception of Letter dataset, the proposed method achieves equal or comparable performance with the best performing algorithms of the comparison.

Table 4: Comparison against other multiclass SVM classifiers using the RBF kernel on accuracy

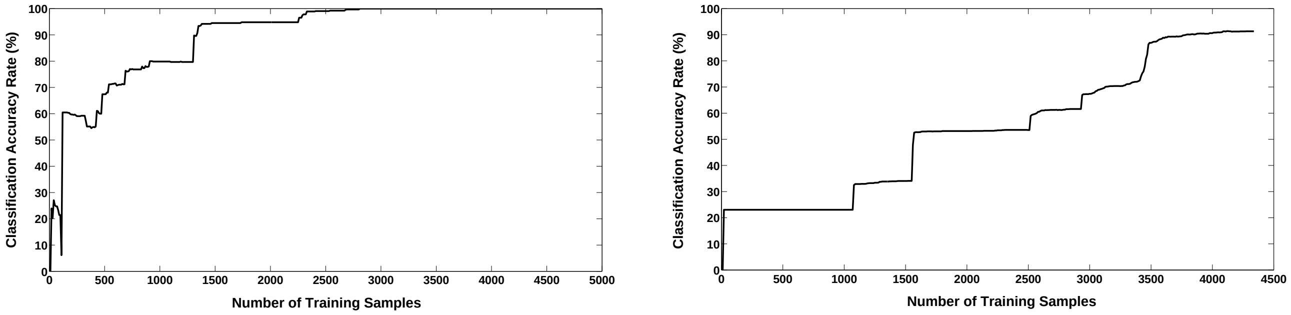
Dataset	DAG SVM	Weston Watkins	MCSVM	LaRank	Proposed Method
Mushroom	100%	100%	100%	100%	100%
Satimage	91.25%	91.25%	92.35%	91.15%	91.35%
Letter	97.98%	97.76%	97.68%	97.35%	93.66%

The minor degradation in the calculated recognition rate here is due to the tight box constraints that we have adopted. We should highlight that there is a trade off between the required training time and the achieved classification accuracy rate. Regarding our algorithm, the fastest training time is attained when considering the cost parameter C to be equal to zero, since this restricts the feasible optimization bounds of the optimized Lagrange multipliers and leads to faster convergence. Unfortunately, for some of the examined datasets, setting parameter C equal to zero is not the optimal selection in terms of the reached classification accuracy rate.

4.5. Frontal Face View Recognition

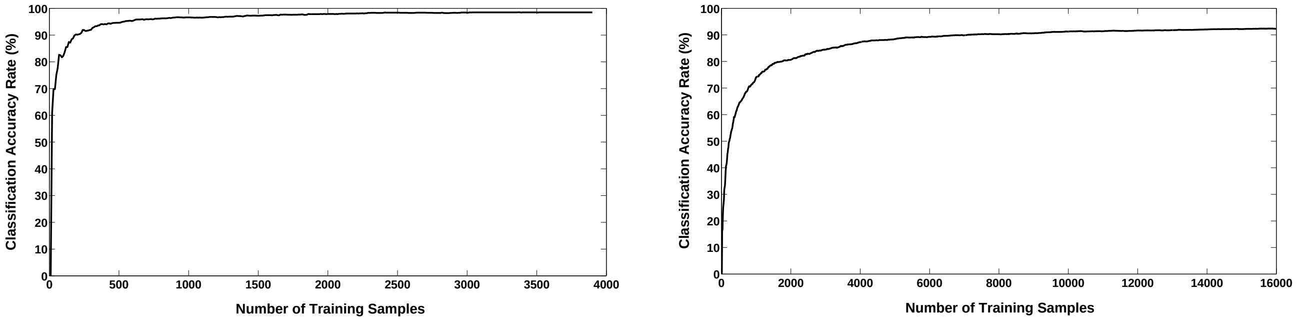
In the final set of experiments we have applied the online learning scenario to the frontal face view recognition problem, where our aim is to distinguish the frontal views of a person's face from the non frontal ones, while he/she is performing various head poses. For this experiment we have used data samples derived from the XM2VTS database, which has been widely applied for frontal face verification.

Experiments on frontal facial view recognition have been carried out similarly to those con-



(a) (b)

Figure 7: Classification accuracy rate versus the training set cardinality for online training on: a) the Mushroom and b) the Satimage datasets.



(a) (b)

Figure 8: Classification accuracy rate versus the training set cardinality for online training on: a) the ORHD and b) the Letter datasets.

cerning online learning with limited training samples, presented in subsection 4.4. That is, cost parameter C was set equal to zero, in order to achieve the fastest training time, while parameter γ was sought during a validation procedure. Initially two samples, one for each class, were randomly selected to start batch training the classifier. Subsequently, and as long as new training samples were sequentially arriving (one at a time) augmenting the training set, the classifier was trained in an online manner warm-starting the optimization process by exploiting the previously found optimal solution. Finally, after each training round we measured the classifiers efficacy by testing each time on the same standard test set.

In order to form the training and test sets, face detection and tracking were applied on the frames of the video sequences and the resulting Regions Of Interest (ROIs) were anisotropically scaled, so as to have fixed size of 30 × 40 pixels and were converted to grayscale. Each such fixed size facial image was scanned row-wise so as to form a feature vector $\mathbf{x} = [f_1 \dots f_{1200}]^T$ (f_i being the luminance of the i-th pixel) which was used to compose the training and test sets that are fed to the SVM classifier. In total 6862 facial images were extracted and divided in two equally sized parts for training and testing, each containing 1243 samples labelled as frontal face instances (according to the range of degrees we defined as an acceptable head rotation in each axis) and 2188 non frontal samples. Figure 9 shows the recognition percentage rate versus the number of samples used in order to train online the classifier. The highest achieved classification accuracy rate is 96.33% when considering the whole available training samples set, with a Gaussian spread parameter value $\gamma = 2^{-2}$. Equal performance was also achieved by our batch training algorithm which was trained on the whole training set, while training using the MCSVM method achieved a slightly better recognition performance rate of 96.67%. It is important to note that the classifier can

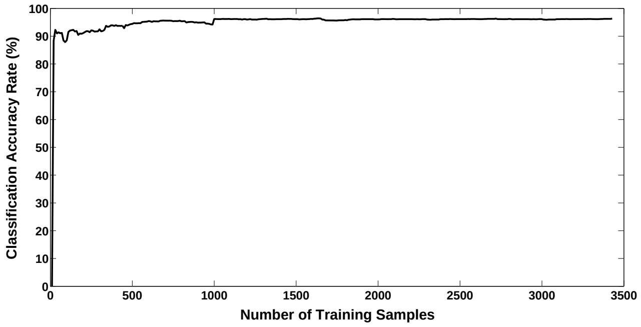


Figure 9: Frontal face view recognition rate versus the number of training samples on online training in the XM2VTS database.

effectively classify the facial images even when only a few tens of samples from each class are used for training. Moreover, the decision surface essentially remains static, as we keep augmenting the training set beyond 1000 training samples, since the achieved classification accuracy rate remains constantly over 96%.

5. Conclusions and Discussion

We have investigated incremental training of multiclass SVMs in order to provide computational efficiency for training problems, where the training data collection is sequentially enriched and dynamic adaptation of the classifier is required. We have introduced an auxiliary function designed for QP problems with non-positivity constraints which can be applied in both non-negative and sign insensitive kernels. A set of novel multiplicative update rules that guarantee convergence to the optimal solution is proposed. The proposed update rules have been applied using tight box constraints regarding the feasible optimization bounds and were incorporated within a warm-start framework in order to further enhance computational efficiency.

The proposed method has been tested on practical, nontrivial problems involving five datasets and showed that the computational benefits are significant, since our method is much faster than retraining the classifier from scratch, while the achieved classification accuracy is maintained close to the state-of-the-art recognition rate reported on the examined datasets, for the applied SVM classifier. Moreover, faster convergence is achieved, since only a few tens of iterations are required so that the optimized variables reach the convergence point, as we have experimentally verified. In our experimental study we have also considered online learning, which provided valuable insights regarding the training set samples and their ability to boost classification accuracy.

Various issues for future research came up through this study, such as reducing memory requirements by splitting the QP problem in smaller subproblems, the investigation about a feasible parallel implementation of the optimization procedure, dynamic data normalization and decremental learning. These are only some promising extensions of the proposed method which we plan to address in a future work.

Acknowledgment

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n^o 211471 (i3DPost).

References

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

[2] C. Cortes, V. Vapnik, Support-vector networks, Machine Learning 20 (3) (1995) 273–297.

[3] C. J. C. Burges, A tutorial on support vector machines for pattern recognition, Data Mining and Knowledge Discovery 2 (2) (1998) 121–167.

[4] R. Fletcher, Practical methods of optimization; (2nd ed.), Wiley-Interscience, New York, NY, USA, 1987.

[5] T.-T. Frieß, N. Cristianini, C. Campbell, The kernel-adatron algorithm: A fast and simple learning procedure for support vector machines, in: International Conference on Machine Learning (ICML), Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998, pp. 188–196.

[6] J. C. Platt, Fast training of support vector machines using sequential minimal optimization, in: B. Schölkopf, C. J. C. Burges, A. J. Smola (Eds.), Advances in Kernel Methods-Support Vector Learning, MIT Press, Cambridge, MA, USA, 1999, pp. 185–208.

[7] W. Krauth, M. Mezard, Learning algorithms with optimal stability in neural networks, Journal of Physics A: Mathematical and General 20 (1987) 745–752.

[8] A. Bordes, L. Bottou, The huller: A simple and efficient online svm, in: European Conference on Machine Learning (ECML), 2005, pp. 505–512.

[9] A. Bordes, S. Ertekin, J. Weston, L. Bottou, Fast kernel classifiers with online and active learning, Journal of Machine Learning Research 6 (2005) 1579–1619.

[10] A. Bordes, L. Bottou, P. Gallinari, J. Weston, Solving multiclass support vector machines with LaRank, in: International conference on Machine learning (ICML), 2007, pp. 89–96.

[11] G. Cauwenberghs, T. Poggio, Incremental and decremental support vector machine learning, in: Advances in Neural Information Processing Systems (NIPS), Vol. 13, MIT Press, 2000, pp. 409–415.

[12] P. Laskov, C. Gehl, S. Krüger, K.-R. Müller, Incremental support vector learning: Analysis, implementation and applications, Journal of Machine Learning Research 7 (2006) 1909–1936.

[13] P. J. Kim, H. J. Chang, J. Y. Choi, Fast incremental learning for one-class support vector classifier using sample margin information, in: International Conference on Pattern Recognition (ICPR), 2008, pp. 1–4.

[14] A. Shilton, M. Palaniswami, S. Member, D. Ralph, A. C. Tsoi, Incremental training of support vector machines, IEEE Transactions on Neural Networks 16 (1) (2005) 114–131.

[15] Z. Liang, Y. Li, Incremental support vector machine learning in the primal and applications, Neurocomputing 72 (10-12) (2009) 2249–2258.

[16] J.-L. An, Z.-O. Wang, Z.-P. Ma, An incremental learning algorithm for support vector machine, in: International Conference on Machine Learning and Cybernetics, Vol. 2, Xi'an, China, 2003, pp. 1153–1156.

[17] S. Katagiri, S. Abe, Incremental training of support vector machines using hyperspheres, Pattern Recognition Letters 27 (13) (2006) 1495–1507.

[18] K. Ikeda, T. Yamasaki, Incremental support vector machines and their geometrical analyses, Neurocomputing 70 (13-15) (2007) 2528–2533.

[19] J. C. Platt, N. Cristianini, J. Shawe-Taylor, Large margin DAG's for multiclass classification, in: Advances in Neural Information Processing Systems, Vol. 12, MIT Press, Cambridge, MA, 2000, pp. 547–553.

[20] H. Gâlmeanu, R. Andonie, A multi-class incremental and decremental svm approach using adaptive directed acyclic graphs, in: IEEE International Conference on Adaptive and Intelligent Systems, 2009, pp. 114–119.

[21] J. Weston, C. Watkins, Support vector machines for multi-class pattern recognition, in: European Symposium On Artificial Neural Networks, 1999.

[22] V. Vapnik, Statistical Learning Theory, Wiley, 1998.

[23] K. Crammer, Y. Singer, On the algorithmic implementation of multiclass kernel-based vector machines, Journal of Machine Learning Research 2 (2001) 265–292.

[24] K. Crammer, Y. Singer, On the learnability and design of output codes for multiclass problems, Machine

Learning 47 (2-3) (2002) 201–233.

[25] A. Aizerman, E. M. Braverman, L. I. Rozoner, Theoretical foundations of the potential function method in pattern recognition learning, Automation and Remote Control 25 (1964) 821–837.

[26] B. Schölkopf, A. J. Smola, Learning with Kernels, MIT Press, Cambridge, MA, 2002.

[27] S. Nikitidis, N. Nikolaidis, I. Pitas, Incremental Training of Multiclass Support Vector Machines, in: International Conference on Pattern Recognition (ICPR), Istanbul, Turkey, 2010, pp. 4267–4270.

[28] B. Schölkopf, S. Mika, C. J. C. Burges, P. Knirsch, K. R. Müller, G. Rätsch, A. J. Smola, Input space versus feature space in kernel-based methods, IEEE Transactions on Neural Networks 10 (5) (1999) 1000–1017.

[29] K. R. Müller, S. Mika, G. Rätsch, K. Tsuda, B. Schölkopf, An introduction to kernel-based learning algorithms, IEEE Transactions on Neural Networks 12 (2) (2001) 181–201.

[30] F. Sha, Y. Lin, L. K. Saul, D. D. Lee, Multiplicative updates for nonnegative quadratic programming, Neural Computation 19 (8) (2007) 2004–2031.

[31] F. Sha, L. K. Saul, D. D. Lee, Multiplicative updates for nonnegative quadratic programming in support vector machines, in: S. Becker, S. Thrun, K. Obermayer (Eds.), Advances in Neural Information Processing Systems (NIPS), MIT Press, 2002, pp. 1041–1048.

[32] A. P. Dempster, N. M. Laird, D. B. Rubin, Maximum likelihood from incomplete data via the em algorithm, Journal of the Royal Statistical Society, Series B 39 (1) (1977) 1–38.

[33] D. D. Lee, H. S. Seung, Algorithms for non-negative matrix factorization, in: Advances in Neural Information Processing Systems (NIPS), MIT Press, 2001, pp. 556–562.

[34] J. Gondzio, Warm start of the primal-dual method applied in the cutting-plane scheme, Mathematical Programming 83 (1) (1998) 125–143.

[35] J. Gondzio, A. Grothey, Reoptimization with the primal-dual interior point method, SIAM Journal on Optimization 13 (3) (2003) 842–864.

[36] E. A. Yildirim, S. Wright, Warm-start strategies in interior-point methods for linear programming, SIAM Journal on Optimization 12 (3) (2002) 782–810.

[37] E. John, E. A. Yildirim, Implementation of warm-start strategies in interior-point methods for linear programming in fixed dimension, Computational Optimization and Applications 41 (2008) 151–183.

[38] A. Asuncion, D. Newman, UCI Machine Learning Repository (2007).

URL http://www.ics.uci.edu/∼mlearn/MLRepository.html

[39] K. Messer, J. Matas, J. Kittler, J. Luttin, G. Maitre, XM2VTSDB: The extended M2VTS database, in: International Conference on Audio and Video-based Biometric Person Authentication (AVBPA), 1999, pp. 72–77.

[40] C.-W. Hsu, C.-J. Lin, A comparison of methods for multiclass support vector machines, IEEE Transactions on Neural Networks 13 (2) (2002) 415–425.

[41] T. Cormen, C. Leiserson, R. Rivest, C. Stein, Introduction to Algorithms, 2nd Edition, The MIT press, 2001.