Growing Adaptive Multi-Hyperplane Machines

Nemanja Djuric [1] **Zhuang Wang** [2] **Slobodan Vucetic** [3]

Abstract

Adaptive Multi-hyperplane Machine (AMM) is an online algorithm for learning Multi-hyperplane Machine (MM), a classification model which allows multiple hyperplanes per class. AMM is based on Stochastic Gradient Descent (SGD), with training time comparable to linear Support Vector Machine (SVM) and significantly higher accuracy. On the other hand, empirical results indicate there is a large accuracy gap between AMM and non-linear SVMs. In this paper we show that this performance gap is not due to limited representability of the MM model, as it can represent arbitrary concepts. We set to explain the connection between the AMM and Learning Vector Quantization (LVQ) algorithms, and introduce a novel Growing AMM (GAMM) classifier motivated by Growing LVQ, that imputes duplicate hyperplanes into the MM model during SGD training. We provide theoretical results showing that GAMM has favorable convergence properties, and analyze the generalization bound of the MM models. Experiments indicate that GAMM achieves significantly improved accuracy on non-linear problems, with only slightly slower training compared to AMM. On some tasks GAMM comes close to non-linear SVM, and outperforms other popular classifiers such as Neural Networks and Random Forests.

1. Introduction

Support Vector Machines (SVMs) with non-linear kernels (Cortes & Vapnik, 1995) can solve very complex, highly non-linear problems. Unfortunately, this strength of SVM is offset by high computational cost of SVM training, which led to development of a number of methods to make SVM more scalable, either by proposing algorithmic speed-ups (Platt, 1998; Kivinen et al., 2002; Vishwanathan et al., 2003; Tsang et al., 2005; Rai et al., 2009), or through parallelization approaches (Graf et al., 2004; Chang et al., 2007; Zhu et al., 2009). Representing kernel matrix through low-rank approximations is a popular line of research (Si et al., 2017; Horn et al., 2018), with a number of recent methods using Nÿstrom-based approximations to speed up training and inference time (Hsieh et al., 2014; Musco & Musco, 2017). However, scalability of non-linear SVM training is inherently constrained by the linear growth of model size with training data size (Steinwart, 2003), making the methods not easily applicable to extremely large problems. This is a seriously limiting constraint, and the trend of ever-growing data collections will further continue to pose new algorithmic challenges to large-scale machine learning.

Linear SVMs, which have a single hyperplane per class in multi-class classification, have been very popular when working with large-scale data (Joachims, 2006; Shalev-Shwartz et al., 2007). This is due to their very favorable properties of linear training time and constant memory scaling. However, scalability of linear SVMs comes at a price of a limited representational power. While linear SVMs have been successfully applied to many large-scale problems, there are many applications where learning a single separating hyperplane per class may not be sufficient. Thus, there is a significant gap in scalability and representational power between linear and non-linear SVMs. Multi-hyperplane Machine (MM) (Aiolli & Sperduti, 2005), which allows several hyperplanes per class, provides an intriguing avenue towards closing this gap. Along similar lines, Kantchelian et al. (2014) proposed a large-margin variant of the algorithm. The MM methods can be trained very efficiently, as described in the Adaptive MM (AMM) algorithm (Wang et al., 2011) that learns MMs in linear time using Stochastic Gradient Descent (SGD). The authors showed that the approach is only slightly slower than linear SVM, while achieving significantly higher accuracies (Wang et al., 2011).

In Wang et al. (2011) the authors show that the AMM algorithm converges to an optimal solution and prove the generalization bounds for separable problems. However, empirical results indicate that there is still a significant gap

[1]Uber ATG, Pittsburgh, PA, USA [2]Facebook, Menlo Park, CA, USA [3]Temple University, Philadelphia, PA, USA. Correspondence to: Nemanja Djuric <nemanja@temple.edu>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020.

between AMM and non-linear SVMs on many data sets. Considering this empirical evidence, it is an open question whether accuracy of MM algorithm is limited by representability of MM models or by optimization issues. In this paper we set to explore this question, and show that MM models can represent arbitrarily complex concepts, thus establishing that further improvements of the MM algorithms could be possible. We show a connection between MM and Learning Vector Quantization (LVQ) models (Kohonen, 1995), and use this insight to propose a novel AMM algorithm called Growing AMM (GAMM), motivated by the highly-influential work on Growing LVQ (Fritzke, 1994; 1995). The GAMM method imputes duplicates of existing hyperplanes into the MM model, leading to significant accuracy gains over AMM. We prove that the proposed method with weight duplication retains favorable convergence properties of the AMM algorithm. Moreover, by using theoretical results on Rademacher complexity applied to SVMs, we derive the generalization bounds of the MM models for non-separable problems as well.

2. Preliminaries and background

In this section we introduce the notation and provide background on hyperplane-based classification methods. In particular, we first describe the multi-class SVM, followed by discussion of the MM and AMM algorithms.

2.1. Multi-class SVM

Let us assume a training data $\mathcal{D} = \{(\mathbf{x}_t, y_t), t = 1, \ldots, T\}$ is given, where instance $\mathbf{x}_t \in \mathbb{R}^D$ is a D-dimensional feature vector and $y_t \in \mathcal{Y} = \{1, \ldots, M\}$ is a multi-class label. The goal is to learn a function $f : \mathbb{R}^D \to \mathcal{Y}$ that accurately predicts a label of new instances. In multi-class SVM (Crammer & Singer, 2001), model $f(\mathbf{x})$ is of the form

$$f(\mathbf{x}) = \arg\max_{i \in \mathcal{Y}} g(i, \mathbf{x}), \text{ where } g(i, \mathbf{x}) = \mathbf{w}_i^\top \mathbf{x}, \quad (1)$$

and is parameterized by the weight vector $\mathbf{w}_i \in \mathbb{R}^D$ of the i^{th} class. Thus, the predicted label of $\mathbf{x}$ is the class of weight vector that achieves the maximum value $g(i, \mathbf{x})$. By concatenating all class-specific weight vectors, we can construct the model matrix

$$\mathbf{W} = [\mathbf{w}_1 \ \mathbf{w}_2 \ \ldots \ \mathbf{w}_M]$$

as the $D \times M$ weight matrix representing $f(\mathbf{x})$. Under this setup, the multi-class SVM problem was defined in (Crammer & Singer, 2001) as

$$\min_{\mathbf{W}} \mathcal{L}(\mathbf{W}) \equiv \frac{\lambda}{2}||\mathbf{W}||_F^2 + \frac{1}{T}\sum_{t=1}^{T} l\big(\mathbf{W}; (\mathbf{x}_t, y_t)\big), \quad (2)$$

where $\lambda > 0$ is a regularization parameter that trades off the model complexity defined as the Frobenius norm on $\mathbf{W}$,

$$||\mathbf{W}||_F^2 = \sum\nolimits_{i \in \mathcal{Y}} ||\mathbf{w}_i||^2,$$

and the margin-based training loss is defined as

$$l\big(\mathbf{W}; (\mathbf{x}_t, y_t)\big) = \max\big(0, 1 + \max_{i \in \mathcal{Y} \setminus y_t} g(i, \mathbf{x}_t) - g(y_t, \mathbf{x}_t)\big). \quad (3)$$

It can be seen from (3) that the loss is zero if the prediction from the correct class is larger by a margin of at least one than the maximal prediction from the incorrect classes; otherwise, a linear penalty is used.

2.2. Multi-hyperplane Machines

In order to increase expressiveness of the classifier in (1), Aiolli & Sperduti (2005) extended the multi-class SVM by allowing multiple weights per class. They showed that this may lead to improved accuracy as compared to linear SVMs (Crammer & Singer, 2001).

Let us denote the j^{th} weight of the i^{th} class by $\mathbf{w}_{i,j}$, and let us assume that the i^{th} class has a total of b_i weights. By redefining $g(i, \mathbf{x})$ from (1) as

$$g(i, \mathbf{x}) = \max_j \mathbf{w}_{i,j}^\top \mathbf{x}, \quad (4)$$

the classifier $f(\mathbf{x})$ from (1) returns the associated class of the weight with the maximal prediction, and the resulting algorithm is called the Multi-hyperplane Machine (MM). With this modification to multi-class SVM, the training loss (3) equals zero if the maximal prediction from the weights of the correct class is by at least one larger than any prediction from weights of the incorrect classes. We can now concatenate all weights and define

$$\mathbf{W} = \Big[\mathbf{w}_{1,1} \ldots \mathbf{w}_{1,b_1} | \mathbf{w}_{2,1} \ldots \mathbf{w}_{2,b_2} | \ldots | \mathbf{w}_{M,1} \ldots \mathbf{w}_{M,b_M}\Big], \quad (5)$$

where $b_1, \ldots, b_M$ are the numbers of weights assigned to each of the classes, and each of M blocks in (5) corresponds to a set of class-specific weights.

The model is learned by solving the equation (2) where $g(i, \mathbf{x})$ is defined as in (4), with model complexity computed using $\mathbf{W}$ from (5). However, the cost function $\mathcal{L}(\mathbf{W})$ is non-convex, and finding the global optimum cannot be guaranteed. Thus, Aiolli & Sperduti (2005) proposed to solve an approximate convex problem,

$$\min_{\mathbf{W}} \mathcal{L}(\mathbf{W}|\mathbf{z}) \equiv \frac{\lambda}{2}||\mathbf{W}||_F^2 + \frac{1}{T}\sum\nolimits_{t=1}^{T} l_{cvx}\big(\mathbf{W}; (\mathbf{x}_t, y_t); z_t\big), \quad (6)$$

where the non-convex loss function $l\big(\mathbf{W}; (\mathbf{x}_t, y_t)\big)$ in (2) is replaced by its convex upper bound,

$$l_{cvx}\big(\mathbf{W}; (\mathbf{x}_t, y_t); z_t\big) = \max\big(0, 1 + \max_{i \in \mathcal{Y} \setminus y_t} g(i, \mathbf{x}_t) - \mathbf{w}_{y_t, z_t}^\top \mathbf{x}_t\big), \quad (7)$$

that replaces the non-convex term $-g(y_t, \mathbf{x}_t)$ in (3) with the convex term $-\mathbf{w}_{y_t, z_t}^\top \mathbf{x}_t$. Element z_t of vector $\mathbf{z} =$

$[z_1 \dots z_T]$ determines which weight belonging to the class of the t^{th} example is used to calculate (7).

Solving the convex problem (6) to train MM proceeds in rounds. At the r^{th} round, the problem (6) with fixed $\mathbf{z}^{(r)}$ is solved to find the optimal $\mathbf{W}^{*(r)}$. Then, matrix $\mathbf{W}^{*(r)}$ is fixed and vector $\mathbf{z}^{(r+1)}$ is calculated as

$$z_t^{(r+1)} = \arg\max_k \left(\mathbf{w}_{y_t,k}^{*(r)}\right)^\top \mathbf{x}_t, \tag{8}$$

which is an index of the correct-class weight with the highest prediction. The iterative procedure of optimizing $\mathbf{W}$ using SGD and reassigning $\mathbf{z}$ is repeated until convergence to a local optimum. However, the computational cost of the proposed batch MM algorithm is still the same as for non-linear SVM training, thus reducing its practical appeal.

2.3. Adaptive Multi-hyperplane Machine (AMM)

In order to address high computational cost of MM training from Aiolli & Sperduti (2005), Adaptive Multi-hyperplane Machine was proposed (Wang et al., 2011). The SGD-based training of AMM allows learning highly accurate MM models in time comparable to linear SVMs. Further, as defined in equation (4), the MM classifier allows using different number of weights (i.e., b_i) for different classes. However, due to practical difficulties in determining these numbers, in Aiolli & Sperduti (2005) all classes are assigned the same, pre-specified number of weights b. This can be suboptimal, since, depending on their distribution, different classes might require different numbers of weights. In AMM, the definition of weight matrix $\mathbf{W}$ in (5) is extended to allow infinite number of weights per class. The AMM algorithm starts by assigning an infinite number of zero-weights to each class, and lets zero-weights become non-zero during learning. To be practically implementable, AMM does not store the zero-weights. Experimental results show that the number of generated non-zero weights adapts to the complexity of the classification problem (Wang et al., 2011).

AMM algorithm uses SGD algorithm to solve the convex optimization problem (6). The SGD is initialized with the zero-matrix (i.e., $\mathbf{W}^{(0)} = \mathbf{0}$), followed by observing examples one by one and modifying the weight matrix accordingly. Upon receiving example $(\mathbf{x}_t, y_t) \in S$ at the t^{th} round, $\mathbf{W}^{(t)}$ is updated as

$$\mathbf{W}^{(t+1)} = \mathbf{W}^{(t)} - \eta^{(t)} \nabla^{(t)}, \tag{9}$$

where $\nabla^{(t)}$ is the sub-gradient of the instantaneous objective on the t^{th} example defined as

$$\mathcal{L}^{(t)}(\mathbf{W}|\mathbf{z}) \equiv \frac{\lambda}{2}||\mathbf{W}||_F^2 + l_{cvx}\left(\mathbf{W}; (\mathbf{x}_t, y_t); z_t\right), \tag{10}$$

and $\eta^{(t)} = 1/(\lambda t)$ is the learning rate. As can be seen, the instantaneous objective differs from equation (6) in that it only calculates the convex loss on the t^{th} example. The outcome of the update can be summarized as follows. At every round all model weights are reduced towards zero by multiplying them with $(1 - 1/t)$. In addition, if the convex loss on the t^{th} example is positive, then the class weight from the true class indexed by z_t (i.e., $\mathbf{w}_{y_t,z_t}^{(t)}$) is moved towards $\mathbf{x}_t$ by adding $\eta^{(t)}\mathbf{x}_t$ to the weight, while the class weight with the maximum prediction from the remaining classes $\mathbf{w}_{i_t,j_t}^{(t)}$ is moved away from $\mathbf{x}_t$ by subtracting $\eta^{(t)}\mathbf{x}_t$ from the weight. If the updated weight is zero-weight it becomes non-zero, thus increasing the number of active weights for that class by one. In this way, complexity of the AMM model adapts to the complexity of the problem at hand, and the number of weights b_i for each class is automatically learned during training.

3. Representability of MM models

The empirical results from Wang et al. (2011) indicate that AMM is more accurate than linear SVM, but less accurate than non-linear SVMs. The question is whether the reduced performance is due to limited representability of the MM model, or due to limitations of the MM learning algorithms. The following theorem answers this question by showing that MM can indeed represent any given concept.

Theorem 1. *Assume that we are given a training set $\mathcal{D} = \{(\mathbf{x}_t, y_t), t = 1, \dots, T\}$, in which there are no two examples with identical feature vectors and different labels. Then, there exists an MM classifier of the form (1) with $g(i, \mathbf{x})$ defined as in equation (4), that perfectly classifies the training set.*

Proof sketch. We give a proof by construction. Let us choose any strictly convex function $h : \mathbb{R}^D \to \mathbb{R}$. For each $\mathbf{x}_t$ find a tangent to $h(\mathbf{x})$ at $\mathbf{x}_t$, include the tangent into the weight matrix $\mathbf{W}$, and label it with y_t. Then, MM with weight matrix $\mathbf{W}$ perfectly classifies the training set, since, due to the strict convexity of $h(\cdot)$, among all other tangents the tangent at $\mathbf{x}_t$ has the maximal value for the t^{th} example and it returns the correct label y_t of the t^{th} example. □

Following the proof, Figure 1a illustrates how MM can be constructed given a training data with 1-D examples. However, MM trained using the construction from the proof is not practical, as it would require one weight vector per training example and would be prone to overfitting. Nevertheless, the result shows that the number of hyperplanes should be allowed to grow according to the data, and for more complex data distributions the model complexity should increase. In the extreme case where positive and negative examples keep alternating as shown in Figure 1a, the number of hyperplanes required to fit the training data would increase as $\mathcal{O}(T)$.

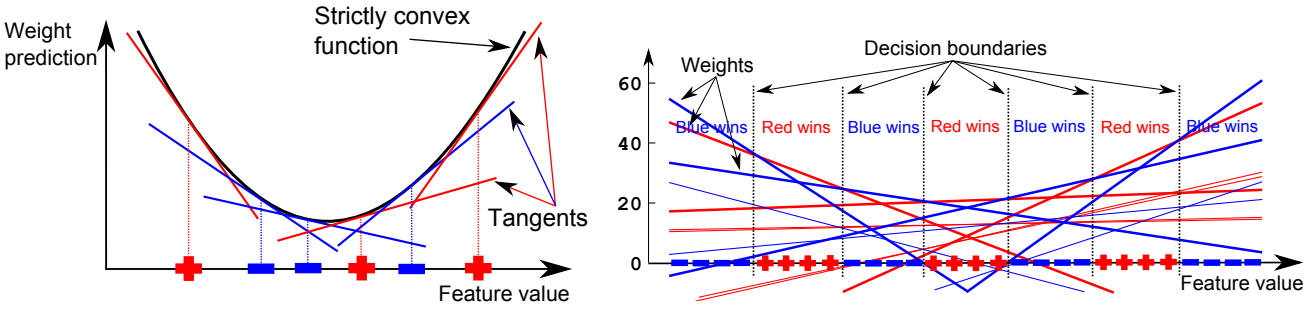


Figure 1. (a) Visualizing the proof of Theorem 1; (b) GAMM model trained on 1×7 XOR-like data (best seen in color)

Similar argument to Theorem 1 can be given to show that MM can represent an arbitrarily complex concept. For any partitioning of the feature space into classes, we can split the space into a fine grid, find a tangent to the center of each grid cell, and assign the tangent to a dominant class of its grid cell. The classification error could be made arbitrarily small by increasing the grid density. However, for high-dimensional feature spaces the size of the resulting MM model would be prohibitively large for practical purposes.

An interesting experimental observation is that the MM models learned by our proposed GAMM algorithm form a convex envelope whose minimum is close to the center of the training data. This is a desired outcome because, in addition to a good training loss, it also benefits the model size. To illustrate this, in Figure 1b we show the model trained by GAMM on 1×7 XOR-like data. As can be seen, active hyperplanes of the model indeed form a convex-shaped envelope with a minimum close to the central red block. For completeness, we note that the resulting model also contains several inactive hyperplanes below the envelope.

4. Connection between MM and LVQ models

By considering Figure 1b and the proof of Theorem 1, we can observe that the weights of MM model define a partition of the input space into disjoint one-class regions. This, along with the prototype-based nature of MM, indicates a connection between the MM models and the well-known LVQ algorithm (Kohonen, 1995) which finds a Voronoi tessellation of the input space. Interestingly, the connection between LVQ and large-margin classifiers has been established before by Crammer et al. (2002), where the authors showed that LVQ is a large-margin classifier that attempts to minimize loss function $\mathcal{L}^{(t)}_{LVQ}$ defined in terms of margin $\theta^{(t)}_{LVQ}$ for a current training example $\mathbf{x}_t$,

$$\mathcal{L}^{(t)}_{LVQ} = \max(0, 1 - \theta^{(t)}_{LVQ}),$$
$$\text{where } \theta^{(t)}_{LVQ} = \frac{1}{2}(\|\mathbf{x}_t - \boldsymbol{\mu}_-\|^2 - \|\mathbf{x}_t - \boldsymbol{\mu}_+\|^2), \quad (11)$$

and $\boldsymbol{\mu}_+$ and $\boldsymbol{\mu}_-$ are the closest LVQ prototypes to the example $\mathbf{x}_t$ belonging to correct and one of incorrect classes, respectively. Note that the loss function $\mathcal{L}^{(t)}_{LVQ}$ does not include regularization term, and, considering LVQ training scheme (Kohonen, 1995), it is not readily obvious how to regularize the LVQ model.

Let us introduce an alternative definition of a margin as $\theta^{(t)}_{MM} = \boldsymbol{\mu}_+^\top \mathbf{x}_t - \boldsymbol{\mu}_-^\top \mathbf{x}_t$. Then, if we choose $\boldsymbol{\mu}_+$ to be a correct-class MM weight assigned to $\mathbf{x}_t$ according to $\mathbf{z}$, and $\boldsymbol{\mu}_-$ is an incorrect-class weight maximizing equation (4), we can rewrite the loss (10) as

$$\mathcal{L}^{(t)}(\mathbf{W}|\mathbf{z}) = \max(0, 1 - \theta^{(t)}_{MM}) + \text{ regularization term.} \quad (12)$$

We can see that the instantaneous loss functions $\mathcal{L}^{(t)}_{LVQ}$ from (11) and $\mathcal{L}^{(t)}(\mathbf{W}|\mathbf{z})$ from (12) are equivalent up to a definition of margin θ and the addition of MM regularization. However, if we expand $\theta^{(t)}_{LVQ}$ as follows,

$$\begin{aligned}\theta^{(t)}_{LVQ} &= \frac{1}{2}\left(\|\mathbf{x}_t - \boldsymbol{\mu}_-\|^2 - \|\mathbf{x}_t - \boldsymbol{\mu}_+\|^2\right) \\ &= \left(\boldsymbol{\mu}_+^\top \mathbf{x}_t - \boldsymbol{\mu}_-^\top \mathbf{x}_t\right) + \frac{1}{2}\left(\|\boldsymbol{\mu}_-\|^2 - \|\boldsymbol{\mu}_+\|^2\right) \\ &= \theta^{(t)}_{MM} + \frac{1}{2}\left(\|\boldsymbol{\mu}_-\|^2 - \|\boldsymbol{\mu}_+\|^2\right),\end{aligned} \quad (13)$$

it becomes clear that the LVQ margin $\theta^{(t)}_{LVQ}$ can be expressed as the sum of MM-type margin $\theta^{(t)}_{MM}$ and an additional term involving norms of the prototypes. In other words, LVQ can be viewed as the MM model without *explicit* regularization that maximizes the margin $\theta^{(t)}_{MM}$ at each training step, where the regularization is *implicit* through the additional margin term forcing the prototypes to have similar norms. Having emphasized these ties between MM and LVQ models, we note that there remains a difference in a way the distance measure between prototypes and example $\mathbf{x}_t$ is defined. Unlike MM that uses linear kernel, LVQ uses Euclidean distance, thus exhibiting limited performance in high-D spaces due to the curse of dimensionality (Hastie et al., 1995; Verleysen et al., 2003).

Algorithm 1 Training algorithm for GAMM

Inputs: Train set $\mathcal{D}$, regularization param. λ, duplication params. p, β
Output: Trained model matrix $\mathbf{W}^{(t)}$

1. **Initialize** $\mathbf{W}^{(0)} \leftarrow \mathbf{0}, t \leftarrow 1, r \leftarrow 0$, and randomly initialize $\mathbf{z}^{(0)}$
2. **repeat**
3. **repeat**
4. **if** $\left(l_{cvx}\left(\mathbf{W}^{(t)}; (\mathbf{x}_t, y_t); z_t^{(r)}\right) > 0\right)$ **then**
5. **if** $\left(rand() < p\right)$ **then**
6. duplicate weight $\mathbf{w}^{(t)}_{y_t, z_t^{(r)}}$;
7. decrease duplication probability as $p \leftarrow \beta p$;
8. compute $\mathbf{W}^{(t++)}$ using (9);
9. **until** (enough epochs)
10. compute $\mathbf{z}^{(++r)}$ using (8), and set $t \leftarrow 1$;
11. **until** $\left(\mathbf{z}^{(r+1)} = \mathbf{z}^{(r)}\right.$ **or** enough epochs$\left.\right)$

5. Growing AMM model

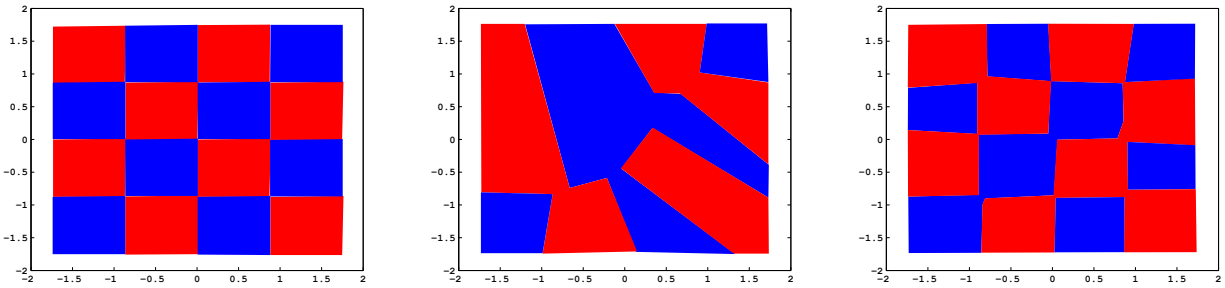
Both MM and LVQ can represent arbitrarily complex concepts, as shown in Theorem 1 and (Hornik et al., 1989), respectively. However, in practice they might obtain limited results due to constraints of their respective training procedures, as illustrated in the experimental section. Significant efforts have been invested to address this problem for LVQ, resulting in the highly-influential work on Growing Self-Organizing Networks (Fritzke, 1994; 1995). These growing networks obtained significant performance improvements over the competing LVQ methods through insertion of prototypes into the model that are a linear combination of the existing ones. Considering the correspondence between LVQ and MM and inspired by the ideas from (Fritzke, 1994; 1995) shown to increase representability of LVQ, we present a novel MM method with significantly improved performance over the aforementioned AMM approach. Higher accuracy is achieved by allowing MM weights to be duplicated during the SGD training. In particular, in addition to pushing the true-class weight closer to a misclassified example, a duplicate of that weight is made with some probability p. As we discuss later, the proposed method retains favorable convergence properties of the MM algorithms.

The pseudocode of the proposed Growing AMM (GAMM) algorithm is shown in Algorithm 1. The GAMM training closely follows the training procedure of AMM. However, unlike in the original AMM, when a convex classification loss is incurred at the t^{th} training iteration (line 4), with probability p we duplicate the true-class weight assigned to the observed example $(\mathbf{x}_t, y_t)$ (lines 5 and 6). Since we expect that the weights will eventually stabilize as the training continues, and to prevent unnecessary generation of duplicate weights in later stages of training, the algorithm gradually decreases the probability p by multiplying with a positive constant $\beta < 1$ after every insertion (line 7; in the experiments we used $\beta = 0.99$). GAMM is a generalization of AMM algorithm, since by setting $p = 0$ we obtain the original AMM algorithm. Note that we can trade-off between exploration and exploitation strategies by manipulating the value of p. By using higher p we explore a larger hypothesis space, possibly leading to discovery of more powerful classifiers but also to overfitting.

The SGD training from Algorithm 1 can result in many non-zero weights, leading to higher complexity of the model. This could negatively impact the generalization of AMM, as shown by Theorem 2 from (Wang et al., 2011) which indicates that the generalization error directly depends on the complexity of the model. In order to reduce the complexity, as well as training and prediction times, pruning of weights was introduced in (Wang et al., 2011). GAMM uses the same pruning approach, formulated as

$$\mathbf{W}^{(t+1)} \leftarrow \mathbf{W}^{(t+1)} - \Delta\mathbf{W}^{(t)}, \tag{14}$$

where $\Delta\mathbf{W}^{(t)}$ is a sparse matrix of the same size as $\mathbf{W}^{(t+1)}$, whose non-zero columns correspond to removed weights. For example, if $\mathbf{w}_{i,j}^{(t)}$ is removed, then $\Delta\mathbf{w}_{i,j}^{(t)} = \mathbf{w}_{i,j}^{(t)}$, while all other columns of $\Delta\mathbf{W}^{(t)}$ are zero. After defining pruning constant $c \geq 0$, pruning (14) is performed by sorting the weights by their norms and deleting as many as possible of those with smaller norms such that the condition $\|\Delta\mathbf{W}^{(t)}\| \leq c/\big((t-1)\lambda\big)$ is not violated. Note that the removal of small weights can be viewed as an online version of the additional ℓ_1-regularization of the MM model (Langford et al., 2009). Thus, by deleting rarely-winning weights when their norm becomes too small, as described by Wang et al. (2011), we adopt an intuitive, theoretically sound approach to lower the model complexity.



(a) Test data set (b) AMM (35.87% error rate) (c) GAMM (7.38% error rate)

Figure 2. Experiments on synthetic 4×4 *checkerboard* data

5.1. Theoretical analysis of GAMM

As shown by Wang et al. (2011), the pruning step described above does not considerably affect a convergence of the AMM algorithm to the solution of problem (6). If we treat weight duplication as model degradation, we can use a similar argument to Theorem 2 in Wang et al. (2011) to characterize convergence of GAMM that uses both weight pruning and weight duplication.

Theorem 2. *Let $\mathbf{W}^*$ be the solution of (6), and T be the total number of training iterations. Further, let the pruning be performed as described above, p be a starting probability of weight duplication, and $0 < \beta < 1$ is a multiplicative factor that reduces p after every weight duplication. Then,*

$$\frac{1}{T}\sum_{t=1}^{T}\big(\mathcal{L}^{(t)}(\mathbf{W}^{(t)}|\mathbf{z}) - \mathcal{L}^{(t)}(\mathbf{W}^{*}|\mathbf{z})\big) \leq \frac{(2+c)^2\big(2+p/(1-\beta)\big)}{\lambda} + \frac{(2+c)^2\big(2+p/(1-\beta)\big)^2}{2T\lambda}\Big(\frac{p(2\beta+3)}{(1-\beta)^2} + \ln(T) + 1\Big). \tag{15}$$

The proof is given in the Appendix. The theorem shows that weight duplication causes an additional constant regret of $(2+c)^2 p/\big((1-\beta)\lambda\big)$ (in the first term on r.h.s.), as well as initially larger regret that vanishes as the iteration count T grows (in the second term on r.h.s.). While the regret bound (15) is not tight, it nevertheless highlights a useful trade-off between model accuracy and its complexity.

Let us discuss the generalization error of a learned MM model. We assume that the product $\mathbb{R}^D \times \mathcal{Y}$ is a measurable space endowed with an unknown probability measure $\mathbb{P}$, and let random pair $(\mathbf{x}, y) \in \mathbb{R}^D \times \mathcal{Y}$ be distributed according to $\mathbb{P}$. Define the risk of function $f : \mathbb{R}^D \to \mathcal{Y}$ as

$$R(f) = \mathbb{E}[\ell\big(y, f(\mathbf{x})\big)], \tag{16}$$

where $\ell(y, f(\mathbf{x})) = \mathbb{1}_{y \neq f(\mathbf{x})}$, and let $\widetilde{R}_N$ denote the corresponding empirical risk on a data sample of size N. Then, the following theorem holds.

Theorem 3. *Let $\mathcal{F}$ be a class of functions that MM can implement, and $\|\mathbf{x}\| \leq 1$ without the loss of generality. Then, with probability of at least $1-\delta$, the risk of any function $f \in \mathcal{F}$ is bounded from above as*

$$R(f) \leq \widetilde{R}_N(f) + \frac{4 + 4K\|\mathbf{W}\|}{\sqrt{N}} + (\|\mathbf{W}\| + 1)\sqrt{\frac{\ln\frac{1}{\delta}}{2N}}, \tag{17}$$

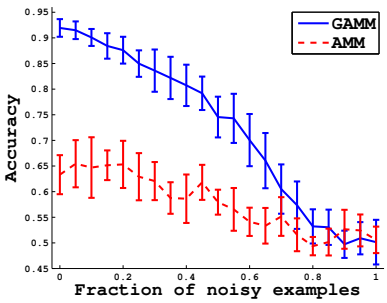
where $K = \sum_{i=1}^{M} b_i \sum_{j \neq i}^{M} b_j$, and b_i is the number of weights for the i^{th} class.

The proof, following Guermeur (2010), is given in the Appendix. Note that the generalization bound is tight as K is bounded because the norm of $\mathbf{W}$ is bounded, as shown in equation (3) of the proof of Theorem 2 in the Appendix. Theorem 3 shows that the true model risk $R(f)$ is closer to the empirical risk $\widetilde{R}(f)$ if the MM model is less complex, with smaller number of weights in a learned model (smaller K). This finding justifies the weight pruning strategy used both in Wang et al. (2011) and in our current work. We can also see that if pruning is too aggressive (resulting in smaller $\|\mathbf{W}\|$) the bound grows tighter, however the empirical risk of the learned model will suffer. Thus, the theorem indicates a trade-off between complexity and the generalization power of the MM models.

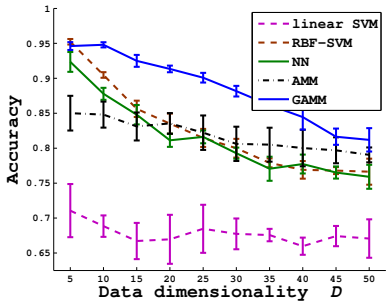
5.2. Online GAMM

Due to the requirement to compute $\mathbf{z}$ using (8) each epoch on the entire data set, the GAMM algorithm as presented in Algorithm 1 is applicable to cases where the data resides on the disk and multiple data scans are permitted. Moreover, additional memory is required in order to compute and store the assignment vector $\mathbf{z}$ at the end of each epoch. This leads to increased memory and latency overhead, as discussed in Wang et al. (2011) in the context of AMM.

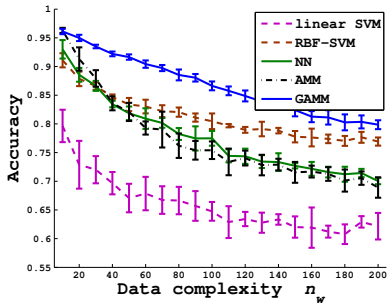
To address this issue, we follow Wang et al. (2011) and propose an online version of GAMM that achieves comparable



(a) AMM and GAMM on noisy 4×4 *checkerboard* data set



(b) Accuracy of the baseline classifiers on the *weights* data set ($n_w = 50$)



(c) Accuracy of the baseline classifiers on the *weights* data set ($D = 20$)

Figure 3. Accuracy of the baseline classifiers on the synthetic data sets of varying characteristics

accuracies with higher efficiency. In particular, for example $(\mathbf{x}_t, y_t)$ we compute an instantaneous assignment z_t as

$$z_t = \arg\max_k \mathbf{w}_{y_t,k}^\top \mathbf{x}_t, \tag{18}$$

thus using the best matching true-class weight to compute the loss (7). While this change renders the loss non-convex, it also results in faster and simpler training procedure, and allows a single-pass training. In addition, our experiments show that the model trained in such online fashion achieves similar performance to the original batched approach. As a result, we use online versions of both AMM and GAMM in the empirical evaluation presented in the following section.

6. Experiments

We implemented GAMM in C++ using open-source toolbox BudgetedSVM[1] (Djuric et al., 2013) and compared to linear SVM solver Pegasos (Shalev-Shwartz et al., 2007), kernel SVM solver LibSVM (Chang & Lin, 2011), as well as AMM (Wang et al., 2011). We also compared to LogitBoost (Friedman et al., 2000), Random Forest (RF) (Breiman, 2001), Neural Networks (NNs), and LVQ2.1 (Kohonen, 1995) using their scikit-learn implementations.

6.1. Detailed analysis of GAMM performance

In this section we present results on synthetic data sets that provide detailed characterization of and insight into the MM models. We run experiments on data of varying characteristics to measure accuracy of the proposed method, and to estimate its robustness to noise.

6.1.1. Experiments on the *checkerboard* data

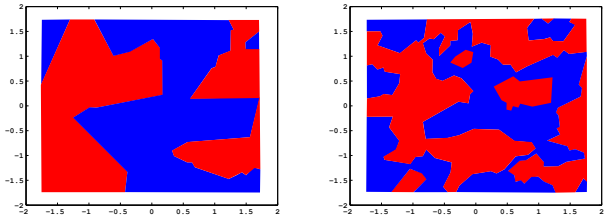
We first considered 4×4 *checkerboard* data, and compared performance of AMM and GAMM. The data is not linearly separable, providing an excellent benchmark for showing the benefits of the proposed GAMM. We created a balanced, two-class training set with 15,000 examples, as well as a test set with 5,000 examples illustrated in Figure 2a. We set parameters to their default values, $c = 10$ for AMM and $c = 50$ for GAMM (due to more frequent introduction of new weights), $p = 0.2$, $\beta = 0.99$, set λ through cross-validation, and trained the models for 15 epochs. As can be seen from the results in Figures 2b and 2c, AMM could not solve this difficult non-linear problem, while GAMM achieved near-perfect class separation. We note that SVM with RBF kernel (RBF-SVM) is very accurate on this data, reaching nearly 99.9% accuracy (Wang et al., 2012).

Table 1. Error rate as a function of the *checkerboard* pattern

Pattern	AMM	GAMM
1×2	0.48 ± 0.33	0.42 ± 0.29
1×3	1.39 ± 0.72	1.33 ± 0.62
1×4	3.23 ± 0.70	2.45 ± 0.51
1×5	4.90 ± 0.97	3.73 ± 0.71
2×2	1.49 ± 0.57	1.08 ± 0.40
2×3	2.98 ± 0.85	2.20 ± 0.52
2×4	10.33 ± 6.77	3.77 ± 0.89
2×5	20.79 ± 7.42	5.90 ± 1.61
3×3	16.92 ± 9.72	4.17 ± 0.77
3×4	24.47 ± 7.42	5.69 ± 0.93
3×5	34.87 ± 4.84	7.97 ± 1.43
4×4	35.87 ± 3.39	7.38 ± 1.53
4×5	35.13 ± 5.26	13.11 ± 1.84
5×5	43.27 ± 3.71	22.15 ± 2.08

To get a better insight into representational power and stability of AMM, we ran experiments on *checkerboard* data sets of different complexity. We changed the number of rows and columns of the grid from 1 to 5, and in Table 1 report mean and standard deviation of classification error rate after 10 repetitions. We can see that AMM could not solve

[1]The GAMM implementation is available for download at https://github.com/djurikom/BudgetedSVM.



(a) $n_w = 50$ (b) $n_w = 200$

Figure 4. Examples of 2-D *weights* data of different complexity

2×4 and 3×3 checkerboards. On the other hand, GAMM achieved significantly higher accuracy and was much more stable than AMM in all the cases.

To further explore robustness of AMM, we added label noise to 4×4 *checkerboard* training data. We increased the fraction of noisy examples (to which we assigned a class label at random) from 0 to 1 in increments of 0.05, repeating each experiment 10 times. The accuracy and confidence intervals indicating two standard deviations computed on noise-free test data are shown in Figure 3a. We see that GAMM is very robust to noise, significantly outperforming AMM until noise levels become prohibitively high.

6.1.2. Experiments on the *weights* data

In the next set of experiments we compared the MM models to a wide range of classifiers on a number of synthetic *weights* data sets of varying complexity. The data set was generated as follows: for a given number of features D and complexity n_w, we first generated n_w weights of dimensionality $D+1$ (including the bias term) by uniformly sampling each dimension from a $[0, 1]$ range, normalized each weight to unit length, and uniformly at random assigned each weight to either positive or negative class. Then, we created 15,000 training and 5,000 test examples by uniformly sampling each dimension from a $[0, 1]$ range. We assigned each example a class using equation (4); see Figures 4a and 4b for examples of the *weights* data. It is worth noting that although the synthetic data sets were generated according to the MM models, similar data could be generated according to the SVM model by choosing a number of support vectors and randomly assigning a class label to each. We set AMM and GAMM parameters as in the previous experiments, λ for Pegasos and C and γ parameters of SVM through cross-validation, and used NN with one hidden layer comprising $2.5D$ nodes (found through cross-validation to give the best NN results for this specific task).

We observed that for larger D and n_w the baselines LogitBoost, RF, and LVQ2.1 had accuracy slightly better than a random guess (the results are not listed). Thus, in Figures 3b and 3c we show accuracies of NN, linear SVM, and RBF-SVM. In Figure 3b we give results on the *weights* data where we set $n_w = 50$, and increased D from 5 to 50 in increments of 5, showing mean accuracy and error bars representing two standard deviations after 10 repetitions. We can see that AMM, NN, and RBF-SVM classifiers performed much better than linear SVM, which was expected because the problems were highly non-linear. For $D = 5$ RBF-SVM performed slightly better than GAMM; however, as the dimensionality increased, both NN and RBF-SVM performance dropped sharply, while GAMM still attained very stable, high accuracy. Interestingly, for higher D the AMM accuracy slowly reached that of GAMM. This can be explained by the fact that as the values of n_w and D become closer, the number of alternating classes along a randomly chosen direction decreases, which poses an easier task for AMM. For a similar effect compare the results of AMM and GAMM for 2×2 vs. 3×3 vs. 4×4 checkerboard data in Table 1, where the gap increases superlinearly with larger complexity. In Figure 3c we provide results on the *weights* data where we set $D = 20$, and increased the number of weights n_w from 10 to 200 in increments of 10 (i.e., we increased "non-linearity" of the data set), showing mean accuracy and error bars representing two standard deviations after 10 repetitions. We can see that GAMM consistently outperformed NN and SVM, while AMM performance dropped sharply as the data complexity increased.

6.2. Performance on real-world data sets

We evaluated the algorithms on 5 real-world data sets[2] of very different sizes, dimensions, and complexities[3]. The results are presented in Table 2. We used the same parameter settings as before, and trained for 15 epochs on the first three smaller data sets, for 5 epochs on medium-sized $rcv1$, and 1 epoch on large $mnist$ data set. To better illustrate scalability we evaluated the algorithms on the lower-end Intel® E7400 with 2.80GHz processor and 4GB RAM, and we excluded data loading time when reporting training times. We can see that GAMM consistently outperformed the AMM algorithm, in all cases by a significant margin. GAMM inherits very favorable scalability of AMM, and is only slightly slower due to weight duplication. Compared to RBF-SVM, on some tasks GAMM comes close while requiring significantly less time to train, thus further closing the representability and scalability gap between MM and SVM models. For example, on $rcv1$ data set it took nearly a whole day to train RBF-SVM on our machine, while GAMM achieved similar accuracy after training of only half a minute. Furthermore, RBF-SVM could not be trained in a reasonable time on the very large, high-

[2] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/, last accessed June 2020.

[3] Binary $mnist$ data set was obtained by classifying round $\{3, 6, 8, 9, 0\}$ vs. non-round digits $\{1, 2, 4, 5, 7\}$.

Table 2. Error rates of the baseline classifiers on real-world data sets

	# classes	# train	# dim	linear SVM	RBF-SVM	AMM	GAMM
$usps$	10	7,291	256	8.38 ± 0.23	4.81	7.32 ± 0.35	6.28 ± 0.19
$letter$	26	15,000	16	25.84 ± 1.12	2.02	17.47 ± 0.93	11.69 ± 0.47
$ijcnn1$	2	49,990	22	7.76 ± 0.19	1.31	2.58 ± 0.21	2.16 ± 0.20
$rcv1$	2	677,399	47,236	2.31 ± 0.03[1]	2.17[1]	2.29 ± 0.09[1]	2.11 ± 0.09[1]
$mnist$	2	8,000,000	784	24.18 ± 0.39[2]	0.43[2]	3.40 ± 0.33[2]	2.84 ± 0.02[2]

[1] $rcv1$ training times: 1.5s (linear SVM); 20.2h (RBF-SVM); 9s (AMM); 32s (GAMM)

[2] $mnist$ training times: 1.1min (linear SVM); 4min (AMM); 19min (GAMM); RBF-SVM accuracy reported after 2 days of P-packSVM training on 512 processors (Zhu et al., 2009)

dimensional $mnist$ data set (the reported accuracy was obtained after 2 days of P-packSVM training on 512 processors (Zhu et al., 2009)), while GAMM was trained within minutes and reached reasonable accuracy.

Lastly, let us consider complexity of the learned MM models, taking the smaller data sets trained for the same number of epochs as an example. After completing 15 epochs of GAMM training the number of hyperplanes for $usps$ was on average 36 per class, 11 per class for $letter$, and 140 per class for $ijcnn1$, roughly correlated with the training times of $8.6s$, $2.7s$, and $9.1s$, respectively. For AMM the average hyperplane counts per class were 3, 2, and 6, with the training times of $1.2s$, $0.6s$, and $0.7s$, respectively. While AMM had somewhat faster training and less complex model, it was also significantly outperformed by the proposed GAMM. As discussed previously, we can also see that model complexity adapts to complexity of the problem at hand, where higher-dimensional, bigger data sets resulted in larger models for both AMM and GAMM.

7. Conclusion

We presented a novel GAMM algorithm that significantly improves accuracy of learned MM models, while retaining very favorable scalability of AMM. We showed that by duplication of hyperplanes during SGD training the resulting MM model can solve highly non-linear problems. In addition, we provided theoretical proofs indicating that MM models can represent arbitrarily complex concepts. Moreover, we showed that GAMM convergence does not significantly suffer due to weight duplication, and provided insightful generalization bounds for non-separable problems for a general class of MM models. Experiments indicate that GAMM achieves significantly improved accuracy over AMM on non-linear problems, with slightly slower training. On some tasks GAMM performed similarly to the state-of-the-art RBF-SVM, suggesting that GAMM represents an efficient and scalable alternative to non-linear SVMs. The results indicate that GAMM occupies an important niche: having computational cost comparable to linear SVMs with accuracy approaching kernel SVMs.

References

Aiolli, F. and Sperduti, A. Multi-class classification with multi-prototype Support Vector Machines. *Journal of Machine Learning Research*, 6(1):817, 2005.

Breiman, L. Random forests. *Machine learning*, 45(1): 5–32, 2001.

Chang, C.-C. and Lin, C.-J. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

Chang, E., Zhu, K., Wang, H., Bai, H., Li, J., Qiu, Z., and Cui, H. Psvm: Parallelizing support vector machines on distributed computers. *Advances in Neural Information Processing Systems*, 20:16, 2007.

Cortes, C. and Vapnik, V. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

Crammer, K. and Singer, Y. On the algorithmic implementation of multi-class kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292, 2001.

Crammer, K., Gilad-Bachrach, R., Navot, A., and Tishby, N. Margin analysis of the LVQ algorithm. *Advances in Neural Information Processing Systems*, 15:462–469, 2002.

Djuric, N., Lan, L., Vucetic, S., and Wang, Z. Budgetedsvm: A toolbox for scalable svm approximations. *The Journal of Machine Learning Research*, 14(1):3813–3817, 2013.

Friedman, J., Hastie, T., and Tibshirani, R. Additive logistic regression: A statistical view of boosting. *The Annals of Statistics*, 28(2):337–407, 2000.

Fritzke, B. Growing cell structures - A self-organizing network for unsupervised and supervised learning. *Neural Networks*, 7(9):1441–1460, 1994.

Fritzke, B. A growing neural gas network learns topologies. *Advances in Neural Information Processing Systems*, 7:625–632, 1995.

Graf, H., Cosatto, E., Bottou, L., Dourdanovic, I., and Vapnik, V. Parallel Support Vector Machines: The cascade SVM. *Advances in Neural Information Processing Systems*, 17:521–528, 2004.

Guermeur, Y. Sample complexity of classifiers taking values in $\mathbb{R}^Q$, application to multi-class SVMs. *Communications in Statistics - Theory and Methods*, 39(3):543–557, 2010.

Hastie, T., Simard, P., and Säckinger, E. Learning prototype models for tangent distance. *Advances in Neural Information Processing Systems*, pp. 999–1006, 1995.

Horn, D., Demircioğlu, A., Bischl, B., Glasmachers, T., and Weihs, C. A comparative study on large scale kernelized support vector machines. *Advances in Data Analysis and Classification*, 12(4):867–883, 2018.

Hornik, K., Stinchcombe, M., and White, H. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

Hsieh, C.-J., Si, S., and Dhillon, I. S. Fast prediction for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, pp. 3689–3697, 2014.

Joachims, T. Training linear SVMs in linear time. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 217–226, 2006.

Kantchelian, A., Tschantz, M. C., Huang, L., Bartlett, P. L., Joseph, A. D., and Tygar, J. D. Large-margin convex polytope machine. In *Advances in Neural Information Processing Systems*, pp. 3248–3256, 2014.

Kivinen, J., Smola, A. J., and Williamson, R. C. Online learning with kernels. *IEEE Transactions on Signal Processing*, 52(8):2165–2176, 2002.

Kohonen, T. Learning vector quantization. In Arbib, M. (ed.), *The Handbook of Brain Theory and Neural Networks*, pp. 537–540. MIT Press, Cambridge, MA, 1995.

Langford, J., Li, L., and Zhang, T. Sparse online learning via truncated gradient. *Journal of Machine Learning Research*, 10(Mar):777–801, 2009.

Musco, C. and Musco, C. Recursive sampling for the nystrom method. In *Advances in Neural Information Processing Systems*, pp. 3833–3845, 2017.

Platt, J. Fast training of Support Vector Machines using Sequential Minimal Optimization. *Advances in kernel methods - support vector learning, MIT Press*, 1998.

Rai, P., Daumé III, H., and Venkatasubramanian, S. Streamed learning: one-pass SVMs. In *International Joint Conference on Artificial Intelligence*, pp. 1211–1216. Morgan Kaufmann Publishers Inc., 2009.

Shalev-Shwartz, S., Singer, Y., and Srebro, N. Pegasos: Primal estimated sub-gradient solver for SVM. In *International Conference on Machine Learning*, pp. 807–814, 2007.

Si, S., Hsieh, C.-J., and Dhillon, I. S. Memory efficient kernel approximation. *The Journal of Machine Learning Research*, 18(1):682–713, 2017.

Steinwart, I. Sparseness of support vector machines. *Journal of Machine Learning Research*, 4:1071–1105, 2003.

Tsang, I. W., Kwok, J. T., and Cheung, P.-M. Core vector machines: Fast SVM training on very large data sets. *Journal of Machine Learning Research*, 6(1):363, 2005.

Verleysen, M., François, D., Simon, G., and Wertz, V. On the effects of dimensionality on data analysis with neural networks. In *Artificial Neural Nets Problem solving methods*, pp. 105–112. Springer, 2003.

Vishwanathan, S. V. N., Smola, A. J., and Murty, M. N. SimpleSVM. In *International Conference on Machine Learning*, 2003.

Wang, Z., Djuric, N., Crammer, K., and Vucetic, S. Trading representability for scalability: Adaptive multi-hyperplane machine for nonlinear classification. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2011.

Wang, Z., Crammer, K., and Vucetic, S. Breaking the curse of kernelization: Budgeted stochastic gradient descent for large-scale svm training. *Journal of Machine Learning Research*, 13(Oct):3103–3131, 2012.

Zhu, Z. A., Chen, W., Wang, G., Zhu, C., and Chen, Z. P-packSVM: parallel primal gradient descent kernel SVM. In *IEEE International Conference on Data Mining*, pp. 677–686, 2009.