NOVEL SUPPORT VECTOR MACHINES FOR DIVERSE LEARNING PARADIGMS

A Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

by

GABRIELLA ANGELA MELKI

Ph.D. Candidate

Director: Alberto Cano,

Assistant Professor, Department of Computer Science,

Virginia Commonwealth University

Director: Sebastian Ventura,

Professor, Department of Computer Science & Numerical Analysis,

University of Córdoba

Virginia Commonwealth University

Richmond, Virginia

September 2018

CHAPTER 1

NOVEL ONLINE SVM USING WORST-VIOLATORS

Due to the ever-growing nature of dataset sizes, the need for scalable and accurate machine learning algorithms has become evident. Stochastic gradient descent methods are popular tools used to optimize large-scale learning problems because of their generalization performance, simplicity, and scalability. This chapter proposes a novel stochastic, also known as online, learning algorithm for solving the L1 support vector machine (SVM) problem, named *OnLine Learning Algorithm using Worst-Violators* (OLLAWV). Unlike other stochastic methods, OLLAWV eliminates the need for specifying the maximum number of iterations and the use of a regularization term. OLLAWV uses early stopping for controlling the size of the margin instead of the regularization term. The experimental study, performed under very strict nested cross-validation (a.k.a., double resampling), evaluates and compares the performance of this proposal with state-of-the-art SVM kernel methods that have been shown to outperform traditional and widely used approaches for solving L1-SVMs such as *Sequential Minimal Optimization*. OLLAWV is also compared to 5 other traditional non-SVM algorithms. The results over 23 datasets show OLLAWV's superior performance in terms of accuracy, scalability, and model sparseness, making it suitable for large-scale learning.

1.1 Online Learning Background

This section defines the notation that will be used throughout the chapter, describes the support vector machine problem, and reviews related works on current popular solvers and online learning algorithms aimed at training support vector machines.

Table 1.1.: Summary of notation used throughout the paper

Definition	Notation
Number of Samples	n
Number of Input Attributes	d
Input Space	$\boldsymbol{X} \in \mathbb{R}^{n \times d}$
Labels	$\boldsymbol{Y} \in \{-1, 1\}^n$
Sample i	$\boldsymbol{x}_i = (x_{i1}, \ldots, x_{id}), \forall i \in \{1, \ldots, n\}$
Sample Label i	$y_i \in \{-1, 1\}, \forall i \in \{1, \ldots, n\}$
Full Training Dataset	$\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \ldots, \{\boldsymbol{x}_i, y_i\}, \ldots, (\boldsymbol{x}_n, y_n)\}$

1.1.1 Notation

Let $\mathcal{D}$ be the full training dataset of n d-dimensional samples. Let $\boldsymbol{Y} \in \mathcal{D}$ be a vector of n labels corresponding to each sample, such that $\boldsymbol{Y} \in \{-1, 1\}^n$. In the non-binary (more than two classes) classification cases, $\boldsymbol{Y} \in \mathbb{Z}^n$. Let $\boldsymbol{X} \in \mathcal{D}$ be a matrix consisting of n samples that are d-dimensional, $\boldsymbol{X} \in \mathbb{R}^{n \times d}$. Table 1.1 provides a summary of the notation used in this paper.

1.1.2 Online Learning Methods

Although support vector machines represent a major development in machine learning algorithms, in the case of large-scale problems (hundreds of thousands to several millions of samples), the design of SVM training algorithms still has room for improvement. So far, there have been several different approaches for tackling large-scale SVM classification problems.

Recently, several authors have proposed the use of a standard stochastic gradient descent (SGD) approach for SVMs to optimize large-scale learning problems [14, 13, 12, 9, 5]. Kivinen et al.[10] and Bousquet and Bottou [2] showed that stochastic algorithms can be both the fastest, and have the best generalization performances. Shalev-Shwartz and Ben-David [13] have also demonstrated that the basic SGD algorithm is very effective when data are sparse, taking less than linear $[O(d)]$ time and space per iteration to optimize a system with d

parameters. It can greatly surpass the performance of more sophisticated batch methods on large data sets. The previously mentioned approaches are extended variants of a classic kernel perceptron algorithm [4].

Notable representatives of this method of learning include the *Naïve Online R Minimization Algorithm* (NORMA) by Kivinen et al. [10] and the *Primal Estimated Sub-Gradient SOlver for SVM* (PEGASOS) by Shalev-Shwartz et al.[14]. NORMA is an online kernel based algorithm designed to utilize SGD for solving the SVM problem, exploiting the kernel trick in an online setting. It can be regarded as a generalization of the kernel perceptron algorithm with regularization [10]. PEGASOS solves the primal SVM problem using stochastic sub-gradient descent, implementing both linear and non-linear kernels, and showed that the algorithm does not directly depend on the size of the data, making it suitable for large-scale learning problems.

A more recent approach that is the inspiration behind this contribution, named *OnLine Learning Algorithm* (OLLA) [6] is a unification, simplification, and expansion of the somewhat similar approaches presented in [14, 13, 12, 9, 5] and [8, 11]. This algorithm is unique because it is not only designed to optimize the SVM cost function, but also the cost functions of several other popular nonlinear (kernel) classifiers using SGD in the primal domain. Collobert and Bengio [4] provided justification for not using regularization, and thus OLLA was designed to handle cost functions with and without the regularization term. Comparisons of performances of OLLA with the popular SMO algorithm highlighted the merits of OLLA in terms of speed, as well as accuracy, when the number of samples was increased, making it suitable for large-scale learning. Comparisons using various different classifiers against SMO were also shown, but for the scope of this paper the L1-SVM was mentioned [6].

Although the SGD approaches mentioned above have many merits when it comes to solving large-scale machine learning problems, stochastic procedures also have their disadvantages. One of them stems from the lack of meaningful stopping criteria. The only specified stopping criteria is a user defined input for the number of iterations, which gives

rise to the question of what it should be set to. Another disadvantage of kernelized online algorithms is that the training time for each update increases superlinearly with the number of samples. This paper aims to deal with these limitations without sacrificing accuracy for scalability with respect to large-scale problems, while maintaining applicability to small and medium sized datasets.

1.2 OnLine Learning Algorithm using Worst-Violator

OLLAWV is an iterative, online learning algorithm for solving the L1-SVM problem using a novel model update procedure while implementing a self-stopping condition. The inspiration behind OLLAWV came from [6] which presented a generic online learning algorithm tailored not only for SVMs, but also for various other popular classifiers that use different risk functions, with or without a regularization term. The difference, novelty, and advantage of OLLAWV resides in its iterative method, where the weight α_i of the most violating sample i.e., of the *worst-violator*, is only updated in each iteration. A worst violating sample is defined as the sample that has the largest error with respect to the current decision function. Rather than randomly selecting samples to update per iteration, OLLAWV selects (without replacement) the most incorrectly classified sample and updates the model accordingly. By iteratively updating the model using only the worst-violator, the model is essentially finding its support vectors, as well as implicitly defining a stopping criterion. If there are no more violating samples, the algorithm terminates, eliminating the need to define the number of iterations for an algorithm to perform before returning the model, as is the case with most state-of-the-art online algorithms.

At every iteration, the algorithm selects a worst violating sample that has not been previously chosen, stores its index in vector $\boldsymbol{S}$, and then updates the model. Equation 1.1 shows the method for selecting the worst-violator, where $yo \in \mathbb{R}$ is the error value, $wv \in \{1, \ldots, n\}$ is the error value's index, $\boldsymbol{o} \in \mathbb{R}^n$ is the decision function output, and $\neg$ is the 'not' symbol. For the L1-SVM, an error value will always be negative which is why the minimum

function is used (i.e. the most negative output value or incorrectly classified sample). The worst violating sample becomes the model's support vector because its weight is updated and non-zero. Therefore, the number of iterations of OLLAWV is equal to the number of support vectors that the resulting model has. This is an interesting property of OLLAWV; if the number of iterations is set beforehand, one is implicitly setting a bound on the number of support vectors.

$$[yo, wv] = \min\{y_{wv} \cdot o_{wv}\},\ \forall wv \in \{\neg \boldsymbol{S}\} \tag{1.1}$$

Algorithm 1.1 lists OLLAWV's pseudocode and Figure 1.2 illustrates the steps taken by OLLAWV. First, the model parameters ($\boldsymbol{\alpha}$, b, $\boldsymbol{S}$) and the algorithm variables ($\boldsymbol{o}$, iteration counter (t), initial worst-violator index wv and it's error yo) are first initialized. The worst-violator with respect to the current hyperplane is then found and the model parameters are updated. Once no more violating samples are found or the maximum number of iterations is reached, the model is returned.

OLLAWV performs stochastic gradient descent on the primal L1-SVM objective function given below:

$$\min_{\boldsymbol{w}\in\mathcal{H}_o\times\mathbb{R}} R \ = \ \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^{n}\max\left\{0, 1 - y_i o_{(w)}(\boldsymbol{x}_i)\right\}, \tag{1.2}$$

where $o_{(w)}(\boldsymbol{x}_i) = \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle$ is a linear predictor, with the bias term excluded for simplicity. Note that the loss function used in Equation 1.2 is non-differentiable, but it has a subgradient due to a knick-point at $yo = 1$. The loss function's gradient after the knick-point equals zero, which leads to a sparse model. Hence, when the value of $yo \geq 1$ the loss is zero, and for $yo < 1$ the loss increases linearly. The subgradient of the above cost function is given by:

$$\frac{\partial R}{\partial \boldsymbol{w}} = \begin{cases} \boldsymbol{w} - C\sum_{i=1}^{n} y_i \boldsymbol{x_i} & y_i o_i < 1 \\ \boldsymbol{w} & \text{otherwise.} \end{cases} \tag{1.3}$$

In the stochastic case, the calculation of the gradient needed for the weight update, is *pattern based*, not *epoch based* as in batch gradient descent. It has been shown [7] that the ideal

gradient is equal to the sum of the gradients calculated after each sample is presented for fixed weights during the whole epoch. Thus, the stochastic update of $\boldsymbol{w}$ from the subgradient shown in Equation 1.3 becomes:

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \frac{\partial R}{\partial \boldsymbol{w}}$$

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta \begin{cases} C y_i \boldsymbol{x_i} - \boldsymbol{w} & y_i o_i < 1 \\ -\boldsymbol{w} & \text{otherwise,} \end{cases}$$

where $\eta > 0 \in \mathbb{R}$ is the learning rate. According to the Representer Theorem, a vector $\boldsymbol{\alpha} \in \mathbb{R}^n$ exists such that $\boldsymbol{w} = \sum_{i=1}^{n} \alpha_i \phi(\boldsymbol{x}_i)$ is an optimal solution to Equation 1.2, where $\phi(\cdot)$ is a mapping from feature space to Hilbert space [13]. On the basis of the Representer Theorem, Equation 1.2 can be optimized with respect to $\boldsymbol{\alpha}$ instead of $\boldsymbol{w}$. By expressing $\boldsymbol{w}$ this way and mapping input sample $\boldsymbol{x_i}$ to $\phi(\boldsymbol{x_i})$, the kernelized SGD update becomes:

$$\sum_{i=1}^{n} \alpha_i \phi(\boldsymbol{x}_i) \leftarrow \sum_{i=1}^{n} \alpha_i \phi(\boldsymbol{x}_i) + \eta \begin{cases} C y_i \phi(\boldsymbol{x_i}) - \sum_{i=1}^{n} \alpha_i \phi(\boldsymbol{x}_i) & y_i o_i < 1 \\ -\sum_{i=1}^{n} \alpha_i \phi(\boldsymbol{x}_i) & \text{otherwise.} \end{cases}$$

However, OLLAWV optimizes in a stochastic manner, resulting in the following update:

$$\forall i : \alpha_i \phi(\boldsymbol{x}_i) \leftarrow \alpha_i \phi(\boldsymbol{x}_i) + \eta \begin{cases} (C y_i \phi(\boldsymbol{x_i}) - \alpha_i \phi(\boldsymbol{x}_i)) & y_i o_i < 1 \\ (-\alpha_i \phi(\boldsymbol{x}_i)) & \text{otherwise} \end{cases}$$

$$\forall i : \alpha_i \leftarrow \alpha_i + \eta \begin{cases} (C y_i - \alpha_i) & y_i o_i < 1 \\ (-\alpha_i) & \text{otherwise.} \end{cases}$$

The case when the worst violating sample is correctly classified $yo \geq 1$ is OLLAWV's termination condition, i.e. is used as the stopping criterion in the algorithm. Hence the update for $\boldsymbol{\alpha}$ is reduced to the following:

$$\forall i : \alpha_i \leftarrow \alpha_i + \eta(C y_i - \alpha_i) \tag{1.4}$$

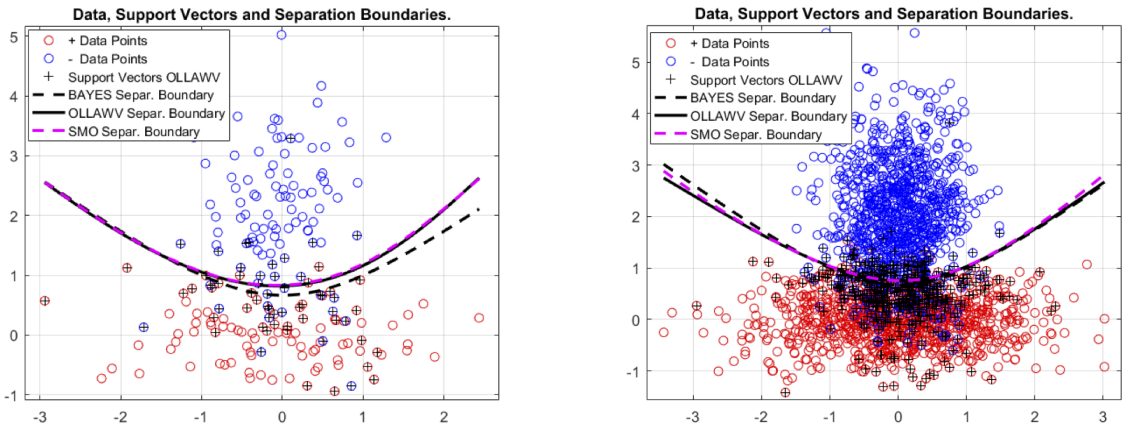

Fig. 1.1.: A case of classifying 2-dimensional normally distributed data with different covariance matrices, (left) for 200 and (right) 2000 data points. The theoretical separation boundary (denoted as the Bayes Separation Boundary) is quadratic and is shown as the dashed black curve. The other two separation boundaries shown are the ones obtained by OLLAWV and SMO (implemented within LIBSVM), respectively. In this particular case (left), the difference between the OLLAWV boundary and the SMO boundary is hardly visible. The case presented on the right shows that, with an increase of training samples, the OLLAWV and SMO boundaries converge to the theoretical Bayesian solution.

If the bias term b is included in Equation 1.2, it's stochastic update is as follows:

$$\forall i : b \leftarrow b + \eta \frac{C y_i}{n} \tag{1.5}$$

In this experimental study, $\eta = 2/\sqrt{t}$ is used, where t is the current iteration; however, other learning rates such as $\eta = 1/t$ can also be used. Let $\Lambda = \eta C y_i$ and $\mathrm{P} = \eta \alpha_i$ be the update parameters for OLLAWV, and the $\boldsymbol{\alpha}$ update can be expressed as: $\alpha_i \leftarrow \alpha_i + (\Lambda - \mathrm{P})$. Note that Λ is the update resulting from the loss function and P is derived from the regularizer term in Equation 1.2. In the case of OLLAWV, $\mathrm{P} = 0$ because samples are never updated more than once and their initial α value is always 0. It is important to note that in OLLAWV's case, Λ never equals 0 because the samples being updated are worst-violators, meaning they are misclassified or incur some loss. The values of the decision function (output

vector $\boldsymbol{o} \in \mathbb{R}^n$ in Algorithm 1.1), from which a worst-violator is found, changes per iteration based on the influence of the support-vectors that have been previously updated. From Equation **??**, the output vector update becomes the following:

$$\boldsymbol{o} \leftarrow \boldsymbol{o} + \Lambda * \mathcal{K}\left(\boldsymbol{x}_{\neg \boldsymbol{S}}, \boldsymbol{x}_{wv}, \gamma\right) + B \tag{1.6}$$

where $\mathcal{K}(\cdot)$ is the Gaussian radial basis function (RBF) kernel, $\gamma \in \mathbb{R}$ is it's parameter, and $B = (\Lambda * \beta)/n$ denotes the bias update. Only non-support vector output values are calculated per iteration, as denoted by $\boldsymbol{x}_{\neg \boldsymbol{S}}$ in the kernel function, because samples are never selected to be updated more than once. Because the output values scale with the C value, the stopping criteria for OLLAWV is also set to scale with C, rather than the classic formulation $y_i o_i \geq 1$. If the value of C is very large, $y_i o_i$ will never be greater than 1 and the algorithm will never terminate. Therefore, the stopping criteria is set to be $y_i o_i \geq M$, where $M \in \mathbb{R}$ is a scaled value of C. For the B calculation in Equation 1.6, $\beta \in \{0, 1\}$ indicates whether the bias term is to be used. If b is not a part of the model, it should be omitted from Equations **??** and 1.6 by setting $\beta = 0$, otherwise $\beta = 1$.

OLLAWV is a stochastic gradient method (SGM) that has a convex cost function. Its learning rate coefficient can decrease linearly or semi-linearly during the learning stage. Hence, OLLAWV shares the complexity characteristics of SGM methods. Primarily, it can achieve linear convergence, making it a particularly convenient and practical method for solving very large machine learning problems. OLLAWV also works over a cost function without local minima, always leading towards the global minimum, even though it stops the learning process as soon as all samples are outside the prescribed margin. Figure 1.1 shows the decision boundaries achieved by OLLAWV versus SMO (implemented within LIBSVM) and Bayes for toy datasets.

Algorithm 1.1 OnLine Learning Algorithm using Worst-Violators (OLLAWV)

Input: $\mathcal{D}$, C, γ, β, M
Output: $\boldsymbol{\alpha}$, b, $\boldsymbol{S}$

$\boldsymbol{\alpha} \leftarrow \mathbf{0}$, $b \leftarrow 0$, $\boldsymbol{S} \leftarrow \mathbf{0}$ ▷ Initialize OLLAWV model parameters
$\boldsymbol{o} \leftarrow \mathbf{0}$, $t \leftarrow 0$ ▷ Initialize the output vector and iteration counter
$wv \leftarrow 0$, $yo \leftarrow y_{wv} * \boldsymbol{o}_{wv}$ ▷ Initialize hinge loss error and worst-violator index
while $yo < M$ **do**
    $t \leftarrow t + 1$
    $\eta \leftarrow 2/\sqrt{t}$ ▷ Learning rate

    $\Lambda \leftarrow \eta * C * y_{wv}$ ▷ Calculate hinge loss update
    $B \leftarrow (\Lambda * \beta)/n$ ▷ Calculate bias update
    $\boldsymbol{o} \leftarrow \boldsymbol{o} + \Lambda * \mathcal{K}(\boldsymbol{x}_{\neg \boldsymbol{S}}, \boldsymbol{x}_{wv}, \gamma) + B$ ▷ Update output vector
    $\boldsymbol{\alpha}_{wv} \leftarrow \boldsymbol{\alpha}_{wv} + \Lambda$ ▷ Update worst-violator's alpha value
    $b \leftarrow b + B$ ▷ Update bias term

    $\boldsymbol{S}_t \leftarrow wv$ ▷ Save index of worst-violator
    $[yo, wv] \leftarrow \min_{wv \in \{\neg \boldsymbol{S}\}} \{y_{wv} \cdot o_{wv}\}$ ▷ Find the worst-violator
end while

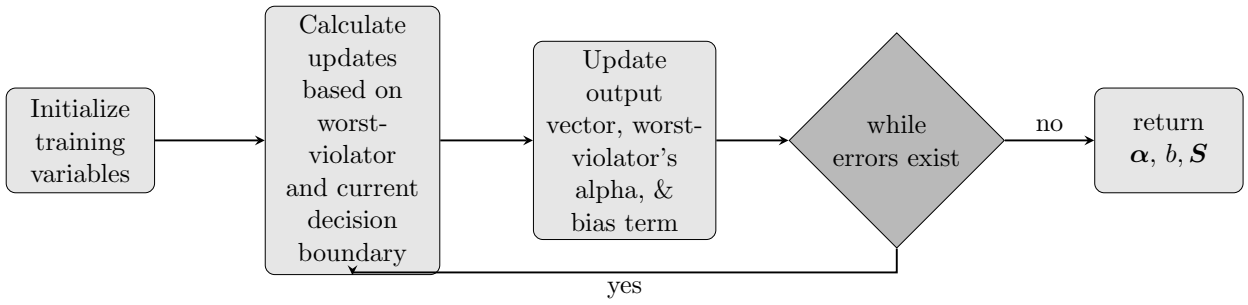

Fig. 1.2.: A summary of the steps performed by OLLAWV. The model parameters ($\boldsymbol{\alpha}$, b, $\boldsymbol{S}$) and the algorithm variables ($\boldsymbol{o}$, t, wv, and yo) are first initialized. The worst-violator with respect to the current hyperplane is then found and the model parameters are then updated. Once no more violating samples are found, the model is returned.

1.3 Experimental Environment, Results, and Analysis

This section presents two experimental setups of our contribution against other state-of-the-art algorithms on 23 different benchmark datasets. The first study, presented in Section 1.3.1, compares OLLAWV to two other SVM kernel methods, and the second compares OLLAWV to 5 non-SVM methods, shown in Section 1.3.2. In each section, the experimental setups are first described and the state-of-the-art methods are listed. The results and statistical analysis [**Derrac2011**] are then presented and analyzed. The main aim of the experiments is to compare our contribution to other support vector machine solvers that have been shown to surpass popular and widely used SVM kernel methods in terms of memory consumption, runtime, and accuracy. The supplemental experimental study in 1.3.2 was conducted to emphasizes the better performance of OLLAWV against non-SVM algorithms.

Table 1.2 presents a summary of the 23 datasets used throughout the experiments, where the number of attributes (dimensionality), classes, and samples are shown. The datasets used and the results obtained are divided into three groups: *small*, *medium* and *large*. The datasets were obtained from the UCI Machine Learning repository[1] [1], and the LibCVM[2] [16] and LIBSVM[3] [3] sites.

1.3.1 SVM Experimental Setup

The experimental setup was designed to evaluate differences in performance of the proposed OLLAWV method against the state-of-the-art algorithms: *Minimal Norm SVM* (MNSVM) [15] and *Non-Negative Iterative Single Data Algorithm* (NNISDA) [17]. These algorithms were chosen because they have shown considerable performance in runtime, memory consumption, and accuracy against the popular and widely used LIBSVM and LibCVM

[1]http://archive.ics.uci.edu/ml/index.php

[2]http://c2inet.sce.ntu.edu.sg/ivor/cvm.html

[3]https://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/

Table 1.2.: Datasets

Dataset	# Samples	# Attributes	# Classes
small datasets			
iris	150	4	3
teach	151	5	3
wine	178	13	3
cancer	198	32	2
sonar	208	60	2
glass	214	9	6
vote	232	16	2
heart	270	13	2
dermatology	366	33	6
prokaryotic	997	20	3
eukaryotic	2,427	20	2
medium datasets			
optdigits	5,620	64	10
satimage	6,435	36	6
usps	9,298	256	10
pendigits	10,992	16	10
reuters	11,069	8,315	2
letter	20,000	16	26
large datasets			
adult	48,842	123	2
w3a	49,749	300	2
shuttle	58,000	7	7
web (w8a)	64,700	300	2
ijcnn1	141,691	22	2
intrusion	5,209,460	127	2

pagackes. In [15], it was shown that MNSVM outperforms both the L1 and L2 implementations of LIBSVM, and BVM embedded in LibCVM. NNISDA was then compared to MNSVM in [17], and showed an added improvement in runtime performance. MNSVM was implemented in an open source C++ framework called "GSVM Command Line Tool for Geometric SVM Training[4]". Both, NNISDA and OLLAWV were implemented as additional modules within Strack-Kecman's code, keeping the experimental environment controlled for all three algorithms. The experiments for all methods were run on the same computer containing two Intel Xeon X5680 CPUs (6-core, 3.33 GHz) and 96 GB of RAM.

Experiments were performed using double, or nested, 5-fold cross-validation in order to

[4]Strack-Kecman, https://github.com/strackr/gsvm

objectively evaluate the models' performances and tune hyper-parameters [**varma2006bias**, **wu2017two**]. In the outer loop, the data are separated into 5 equally sized folds and each part is held out in turn as the test set, and the remaining four parts are used as the training set. In the inner loop, 5-fold cross-validation is also used over the training set assigned by the outer loop, where the best hyper-parameters are chosen. The best model obtained by the inner loop is then applied on the outer loop's test set. This procedure ensures the model's performance is not optimistically biased as when using a single loop of k-fold cross-validation. It ensures the class labels of the test data will not be seen when tuning the hyper-parameters, which is consistent with real-world applications. Obviously, such a rigorous procedure is computationally expensive, but the goal is to fairly compare different classification models on the same data sets, with the same cross-validation procedure, and hyper-parameters. First, the datasets were normalized by linear transformation of the feature values to the range $[0, 1]$. Then, the training process, also involving model selection using pattern search, was performed. The best hyper-parameters were chosen from the following 8×8 possible combinations, shown in Equations (1.7a) and (1.7b), and were also used for the competing SVM methods.

$$C \in \{4^n\}, \quad n = \{-2, \ldots, 5\} \tag{1.7a}$$

$$\gamma \in \{4^n\}, \quad n = \{-5, \ldots, 2\} \tag{1.7b}$$

The γ parameter refers to that of the Gaussian RBF kernel, given by:

$$\mathcal{K}(\boldsymbol{x_i}, \boldsymbol{x_j}) = e^{-\gamma ||\boldsymbol{x_i} - \boldsymbol{x_j}||^2}. \tag{1.8}$$

To deal with multi-class classification problems, the one-vs-one, or pairwise, approach was used. The pairwise training procedure trains $c(c-1)/2$ binary classifiers, a classifier for each possible pair of classes, where c is the number of classes. During the prediction phase, a voting scheme is used where all $c(c-1)/2$ models predict an unseen data sample and the class that received the highest number of votes is considered to be the samples true class.

Table 1.3.: Comparison of OLLAWV vs. NNISDA and MNSVM

Dataset	Accuracy (%)			Runtime (s)			Support Vectors (%)		
	OLLAWV	NNISDA	MNSVM	OLLAWV	NNISDA	MNSVM	OLLAWV	NNISDA	MNSVM
small datasets									
iris	**97.33**	94.00	96.67	**0.05**	0.27	3.57	**13.50**	40.20	29.80
teach	52.32	52.31	**52.95**	**0.12**	0.44	8.85	**69.19**	99.80	87.40
wine	**98.87**	96.60	96.60	**0.28**	0.43	4.84	**15.02**	44.40	48.60
cancer	80.36	**81.86**	81.38	**0.49**	0.85	4.46	**42.79**	83.80	89.60
sonar	**92.32**	89.48	87.57	**0.59**	0.98	3.03	**31.26**	73.00	66.00
glass	**72.41**	67.81	69.30	**0.46**	1.01	11.94	**62.84**	90.80	87.60
vote	**96.54**	96.11	93.99	**0.26**	0.46	1.49	**13.36**	33.20	34.00
heart	82.22	**83.33**	**83.33**	**0.50**	0.91	6.45	**37.69**	73.00	82.00
dermatology	97.82	**98.36**	**98.36**	**1.62**	2.47	11.68	**36.94**	59.00	59.80
prokaryotic	88.96	88.86	**88.97**	**6.09**	10.64	50.86	**29.01**	51.20	49.00
eukaryotic	77.38	79.56	**81.21**	61.95	**49.16**	342.76	**54.11**	76.40	72.60
medium datasets									
optdigits	99.11	99.29	**99.31**	**411**	528	787	**28.64**	31.60	30.60
satimage	91.66	**92.39**	92.35	1,334	**687**	1,094	**20.72**	45.00	44.80
usps	97.49	98.05	**98.24**	10,214	**5,245**	7,777	**11.22**	29.40	28.00
pendigits	99.56	**99.62**	99.61	**723**	909	1,500	**10.27**	17.60	16.60
reuters	98.03	**98.08**	97.99	**954**	1,368	1,657	**8.770**	18.20	18.60
letter	96.99	99.11	**99.13**	**5,259**	12,009	26,551	**43.56**	57.60	56.60
large datasets									
adult	84.75	85.07	**85.13**	**21,025**	72,552	123,067	**34.66**	56.00	56.60
w3a	**98.86**	98.82	98.82	**6,532**	15,951	24,562	**3.270**	14.60	12.40
shuttle	99.77	99.83	**99.87**	**2,833**	7,420	45,062	**2.010**	6.00	16.40
web	98.94	**99.00**	99.00	**12,067**	30,583	38,040	**4.320**	13.20	10.80
ijcnn1	98.31	99.34	**99.41**	**162,587**	296,917	370,144	16.36	11.00	**7.600**
intrusion	**99.77**	99.67	99.66	**2,402,804**	4,646,810	3,772,113	**0.780**	2.000	1.700
Average	**91.29**	91.15	91.25	**114,209**	221,350	191,861	**25.66**	44.65	43.79
Ranks	**1.739**	2.022	2.239	**1.217**	1.913	2.869	**1.087**	2.609	2.304

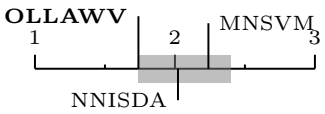

Fig. 1.3.: Bonferroni-Dunn test for Accuracy

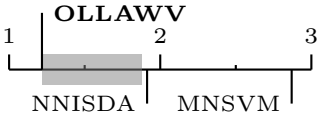

Fig. 1.4.: Bonferroni-Dunn test for Runtime

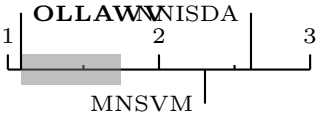

Fig. 1.5.: Bonferroni-Dunn test for % Support Vectors

Comparison Results and Analysis

The classification performance was measured using the following metrics: accuracy, runtime, and the percentage of support vectors (size of the model). Table 1.3 displays the results for OLLAWV and the two state-of-the-art methods. The percentage of support vectors was reported for analyzing the complexities of the resulting models over the variously

sized datasets. In order to analyze the performances of the multiple models, non-parametric statistical tests are used to validate the experimental results obtained [**Derrac2011**]. The Iman-Davenport non-parametric test is run to investigate whether significant differences exist among the performance of the algorithms by ranking them over the datasets used, using the Friedman test. The algorithm ranks for each metric are presented in the last row of Table 1.3, and the lowest (best) rank value is typeset in bold. After the Iman-Davenport test indicates significant differences (with p-value = 0.2397 for accuracy, and p-value = 0 for runtime and percent support vectors), the Bonferroni-Dunn post-hoc test is then used to find where they occur between algorithms by assuming the classifiers' performances are different by at least some critical value (critical distance is 0.66 for $\alpha = 0.05$). Below Table 1.3, Figures 1.3, 1.4, and 1.5 highlight the critical distance (in gray) from the best ranking algorithm to the rest. The algorithms to the right of the critical distance bar perform statistically significantly worse than the control algorithm, OLLAWV.

The results in Table 1.3 indicate that OLLAWV outperforms NNISDA and MNSVM in terms of accuracy, runtime, and model complexity. Although the differences in accuracy between the methods is not very large, on average, OLLAWV is about 2 times faster than NNISDA and MNSVM. As mentioned previously, OLLAWV aims to speed up the learning process without sacrificing the model's accuracy. This stems from OLLAWV's ability to produce sparse models, as is shown by the averaged percentage of support vectors. The speedup that OLLAWV presents is proportional to the model complexity and the experimental results show that OLLAWV produces, on average, models that are 1.7 times smaller than the two state-of-the-art methods used. This highlights the applicability and advantage that OLLAWV has for learning from large datasets.

Figures 1.3, 1.4, and 1.5 show the results of the statistical analysis for accuracy, runtime, and percentage of support vectors. Figures 1.4 and 1.5 show that OLLAWV is statistically significantly better than MNSVM and NNISDA for runtime and percentage of support vectors (model size). At the same time, Figure 1.3 emphasizes what was mentioned earlier:

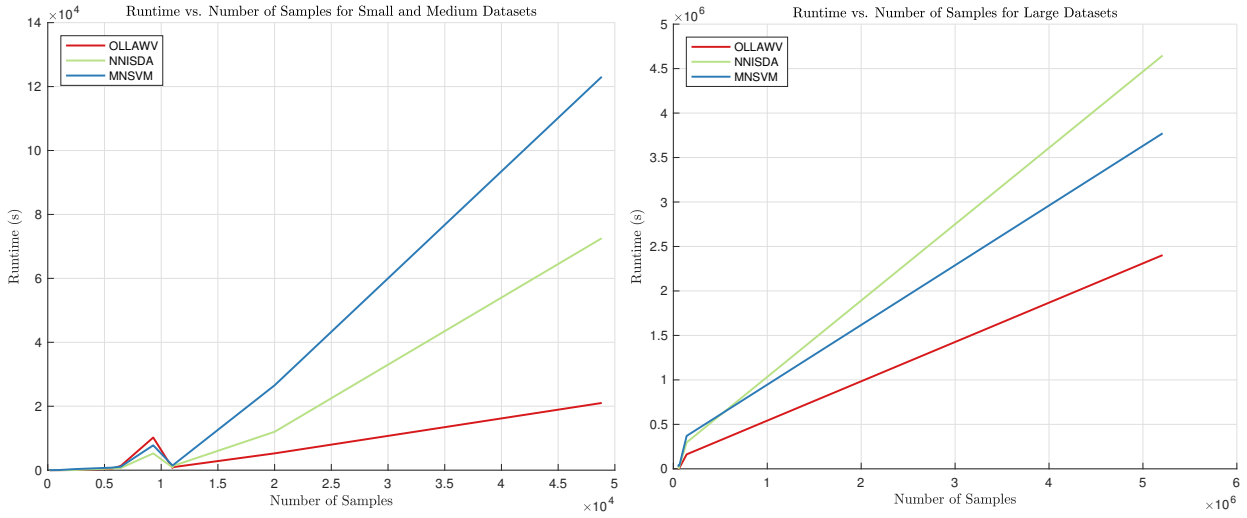

Fig. 1.6.: Runtime in seconds versus the number of samples, divided into two groups: small & medium (left) versus large (right). Note OLLAWV's gradual increase in runtime as the number of samples increases compared to NNISDA and MNSVM's steeper change. In almost all cases, OLLAWV displays superior runtime over state-of-the-art. Runtime depends upon many characteristics: dimensionality, class-overlapping, complexity of the separation boundary, number of classes as well as upon the number of support vectors, which partly explains the tiny bump in the left figure.

OLLAWV is shown to speed up the learning process without sacrificing model accuracy against the state-of-the-art methods used.

Figure 1.6 plots the correlation of OLLAWV, NNISDA, and MNSVM's runtime versus number of samples for the small, medium, and large datasets. The figure clearly emphasizes the benefit of using OLLAWV for large-scale learning due to it's gradual increase in runtime as the number of samples increases in comparison to NNISDA and MNSVM. Figure 1.7 shows the correlation between OLLAWV, NNISDA, and MNSVM's percentage of support vectors and the number of samples for all datasets. It highlights OLLAWV's model sparseness in comparison to the competing methods, while mirroring the runtime results.

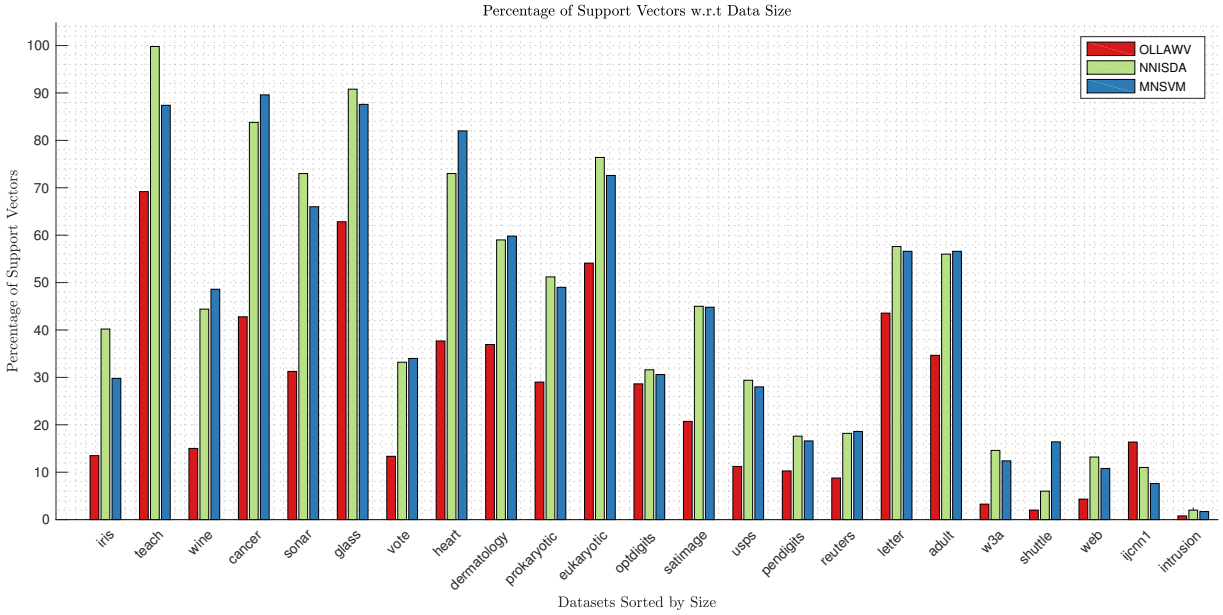

Fig. 1.7.: Size of the model given as percentage of support vectors with respect to the number of samples versus the number of samples. Note that OLLAWVs percentage of support vectors is always smaller (except in one case) than NNISDA's and MNSVM's ones.

1.3.2 Non-SVM Experimental Study

The supplemental experimental setup was designed to compare the performance of the proposed OLLAWV against the following 5 popular and widely-used non-SVM algorithms: *k-Nearest Neighbors (KNN)*, *J48*, *JRip*, *Naïve Bayes*, and *Logistic*. These methods have been implemented within the Weka framework [**eibe2016weka**]. The experiments were performed under the same nested 5-fold cross-validation framework as the SVM algorithm experimental study which was described in Section 1.3.1. The following hyper-parameters shown in Table 1.4 were used for the non-SVM algorithms.

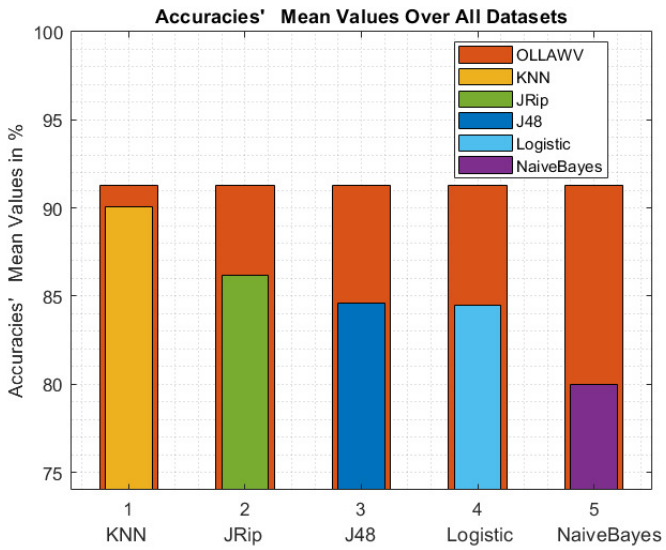

Fig. 1.9.: Mean accuracy over all datasets for OLLAWV and the 5 non-SVM state-of-the-art methods.

Table 1.4.: Non-SVM Algorithm Hyper-parameters

Algorithm	Parameters
k-NN	Number of neighbors: $k \in \{1, 3, 5, 7\}$
J48	Pruning: {True, False}, Pruning Confidence: {0.1, 0.25, 0.5}
JRip	Pruning: {True, False}
Naive Bayes	Use kernel estimation: {True, False}
Logistic	Log-likelihood: $\{1e^{-7}, 1e^{-8}, 1e^{-9}\}$

Results and Statistical Analysis

Table 1.5 displays the accuracy results for OLLAWV and five state-of-the-art methods. The table also shows the standard deviation for accuracy per outer fold, the average values accross all datasets, and the algorithm ranks. As the results indicate, OLLAWV outperforms all other state-of-the-art methods. Figure 1.9 displays the average accuracy results for

Table 1.5.: Accuracy Comparison for Non-SVM Methods versus OLLAWV

Dataset	OLLAWV	KNN	J48	JRip	Naïve Bayes	Logistic
iris	**97.33 ± 1.49**	96.00 ± 3.65	94.00 ± 2.79	90.67 ± 4.35	96.00 ± 2.79	97.33 ± 2.7
teach	52.32 ± 3.46	**59.64 ± 2.89**	49.72 ± 7.58	56.75 ± 9.60	53.75 ± 6.46	51.77 ± 6.6
wine	**98.87 ± 1.54**	97.73 ± 3.72	90.43 ± 5.83	93.24 ± 3.27	96.60 ± 3.14	96.05 ± 2.5
cancer	**80.36 ± 5.80**	77.32 ± 6.93	73.81 ± 8.57	73.78 ± 5.81	67.73 ± 5.07	77.32 ± 7.7
sonar	**92.32 ± 3.11**	88.99 ± 4.59	76.16 ± 10.6	75.18 ± 6.77	73.69 ± 7.65	75.18 ± 7.3
glass	**72.41 ± 2.28**	67.73 ± 5.91	65.06 ± 5.51	65.59 ± 9.66	49.46 ± 5.19	62.04 ± 5.7
vote	**96.54 ± 1.87**	92.26 ± 3.19	95.70 ± 2.12	96.54 ± 2.45	92.24 ± 3.24	93.54 ± 2.5
heart2	82.22 ± 2.93	79.63 ± 5.71	78.52 ± 2.81	80.74 ± 4.06	**84.44 ± 4.46**	83.33 ± 3.9
dermatology	**97.82 ± 0.05**	96.18 ± 1.78	94.52 ± 2.21	91.27 ± 5.08	97.28 ± 1.64	96.98 ± 2.2
pro	**88.96 ± 2.14**	87.96 ± 3.01	78.54 ± 1.62	79.13 ± 2.78	62.38 ± 3.54	87.57 ± 2.5
euk	77.38 ± 1.96	**81.42 ± 2.06**	65.27 ± 2.92	66.42 ± 3.47	39.27 ± 3.43	69.55 ± 1.3
optdigits	**99.11 ± 0.38**	98.74 ± 0.39	90.87 ± 1.09	91.28 ± 0.40	92.42 ± 0.75	95.05 ± 0.9
satimage	**91.66 ± 0.80**	90.38 ± 0.72	85.64 ± 1.21	85.33 ± 0.77	85.41 ± 0.92	88.14 ± 1.1
usps	**97.49 ± 0.22**	97.04 ± 0.47	88.73 ± 0.46	89.20 ± 1.00	79.45 ± 0.59	91.88 ± 0.6
pendigits	**99.56 ± 0.12**	99.33 ± 0.17	96.24 ± 0.31	96.34 ± 0.41	88.34 ± 0.65	95.59 ± 0.1
reuters	**98.03 ± 0.22**	97.15 ± 0.43	96.90 ± 0.32	97.18 ± 0.44	93.52 ± 0.02	69.54 ± 0.2
letter	**96.99 ± 0.21**	95.71 ± 0.19	87.34 ± 0.68	87.02 ± 0.66	74.12 ± 0.97	77.45 ± 0.1
adult	**84.75 ± 0.26**	83.85 ± 0.28	84.38 ± 0.28	83.73 ± 0.17	80.57 ± 0.09	82.46 ± 0.1
w3a	**98.86 ± 0.04**	98.60 ± 0.06	98.71 ± 0.05	98.41 ± 0.10	96.71 ± 0.20	98.61 ± 0.1
shuttle	99.77 ± 0.03	99.93 ± 0.03	**99.97 ± 0.02**	99.96 ± 0.02	98.57 ± 0.24	96.83 ± 0.1
web	**98.94 ± 0.05**	98.89 ± 0.06	98.79 ± 0.09	98.50 ± 0.13	96.71 ± 0.21	98.70 ± 0.0
ijcnn1	98.31 ± 0.07	**98.48 ± 0.04**	98.40 ± 0.09	98.11 ± 0.10	90.69 ± 0.26	92.29 ± 0.1
intrusion	**99.77 ± 0.02**	88.20 ± 1.06	58.01 ± 26.6	87.66 ± 3.79	49.75 ± 30.7	65.15 ± 15
Average	**91.29 ± 1.26**	90.05 ± 2.06	84.60 ± 3.64	86.18 ± 2.84	79.96 ± 3.58	84.45 ± 2.8
Ranks	**1.500**	2.500	4.041	3.9583	5.0625	3.9375

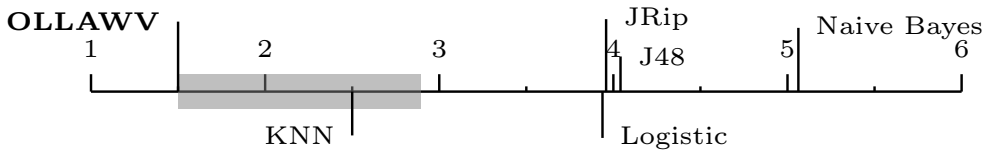

Fig. 1.8.: Bonferroni-Dunn test for Accuracy

OLLAWV and the non-SVM methods accross all datasets and highlights OLLAWV's better performance. The Friedman test indicates that OLLAWV performs significantly better than the competing methods for $\alpha = 0.05$ and is ranked first. Figure 1.8 shows the critical distance bar (which is 1.391), and indicates that all other algorithms perform statistically significantly worse than OLLAWV, except for KNN.

1.4 Conclusion

This paper proposed a novel online learning procedure and algorithm for solving the L1-SVM problem, which is a unique method in terms of both iterating over samples and updating the model. A new stopping criterion for the stochastic gradient procedure is also proposed. The model is updated by changing the weight α_i of a single worst-violator per iteration and stops when all violating samples i.e., support vectors, are found. Finding the *worst-violators* is done without replacement. Such an approach results in a significant shortening of training time, as well as in a huge decrease in the resulting model size. The key features of the proposed algorithm, OLLAWV, stem from it's implicit ability of finding support vectors and it's self-stopping condition. This design was devised to address the limitations presented by current SVM solvers.

The first experimental study demonstrates the better performance of OLLAWV compared with state-of-the-art SVM solvers (MNSVM and NNISDA) which have been shown to outperform the popular SMO implementation in the LIBSVM package. The results for accuracy, runtime, and percentage of support-vectors, obtained by the strict nested cross-validation procedure, were compared and further validated using statistical analysis with non-parametric tests. They highlighted the advantages and major speedup achieved by OLLAWV against the competing MNSVM and NNISDA. The second, supplemental experimental study evaluated the performance of OLLAWV against 5 popular non-SVM methods, showing the better performance of OLLAWV against all five non-SVM algorithms (k-Nearest Neighbors (KNN), J48, JRip, Naïve Bayes, and Logistic). The proposal, OLLAWV, performs statistically better in terms of runtime and model size across all 23 evaluated benchmark datasets, without compromising accuracy.