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: Sebastián Ventura,

Professor, Department of Computer Science & Numerical Analysis,

University of Córdoba

Virginia Commonwealth University

Richmond, Virginia

September 2018

Abstract

NOVEL SUPPORT VECTOR MACHINES FOR DIVERSE LEARNING PARADIGMS

This dissertation introduces novel support vector machines (SVM) for the following traditional and non-traditional learning paradigms: *Online* classification, *Multi-Target* Regression, *Multiple-Instance* classification, and *Data Stream* classification.

Three multi-target support vector regression (SVR) models are first presented. The first involves building independent, single-target SVR models for each target. The second builds an ensemble of randomly chained models using the first single-target method as a base model. The third calculates the targets' correlations and forms a maximum correlation chain, which is used to build a single chained SVR model, improving the model's prediction performance, while reducing computational complexity.

Under the multi-instance paradigm, a novel SVM multiple-instance formulation and an algorithm with a bag-representative selector, named *Multi-Instance Representative SVM* (MIRSVM), are presented. The contribution trains the SVM based on bag-level information and is able to identify instances that highly impact classification, i.e. bag-representatives, for both positive and negative bags, while finding the optimal class separation hyperplane. Unlike other multi-instance SVM methods, this approach eliminates possible class imbalance issues by allowing both positive and negative bags to have at most one representative, which constitute as the most contributing instances to the model.

Due to the shortcomings of current popular SVM solvers, especially in the context of large-scale learning, the third contribution presents a novel stochastic, i.e. online, learning algorithm for solving the L1-SVM problem in the primal domain, dubbed *OnLine Learning Algorithm using Worst-Violators* (OLLAWV). This algorithm, unlike other stochastic methods, provides a novel stopping criteria and eliminates the need for using a regularization term. It instead uses early stopping. Because of these characteristics, OLLAWV was proven to efficiently produce sparse models, while maintaining a competitive accuracy.

OLLAWV's online nature and success for traditional classification inspired its implementation, as well as its predecessor named *OnLine Learning Algorithm - List 2* (OLLA-L2), under the batch data stream classification setting. Unlike other existing methods, these two algorithms were chosen because their properties are a natural remedy for the time and memory constraints that arise from the data stream problem. OLLA-L2's low spacial complexity deals with memory constraints imposed by the data stream setting, and OLLAWV's fast run time, early self-stopping capability, as well as the ability to produce sparse models, agrees with both memory and time constraints. The preliminary results for OLLAWV showed a superior performance to its predecessor and was chosen to be used in the final set of experiments against current popular data stream methods.

Rigorous experimental studies and statistical analyses over various metrics and datasets were conducted in order to comprehensively compare the proposed solutions against modern, widely-used methods from all paradigms. The experimental studies and analyses confirm that the proposals achieve better performances and more scalable solutions than the methods compared, making them competitive in their respected fields.

TABLE OF CONTENTS

LIST OF ALGORITHMS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In traditional classification and regression problems, learning algorithms uncover dependencies and patterns that exist between given inputs (samples) and their outputs (categorical or continuous), using training data. Identifying these patterns is a non-trivial task due to many factors, such as the high dimensionality of the data, as well as dataset size. Over the past decade, dataset sizes have grown disproportionately to the speed of processors and memory capacity, limiting machine learning methods to computational time. Many real-world applications, such as human activity recognition, operations research, and video/signal processing, require algorithms that are scalable and accurate, while being able to provide insightful information in a timely fashion.

More recently, these classic methods have been extended to accommodate various types of data paradigms [163]. Examples include *Multiple Target* (MT) learning, *Multiple Instance* (MI) learning, and *Data Stream* learning. These emerging paradigms require algorithms to be robust, while accommodating and exploiting their different and complex data representations. In this thesis, various approaches are devised for solving classification and regression problems within the traditional and non-traditional learning paradigms mentioned, using support vector machines.

Multi-target learning is a challenging task that consists of creating predictive models for problems with multiple simultaneous outputs [12, 45, 164]. Learning under this paradigm has the capacity to generate models representing a wide variety of real-world applications, ranging from natural language processing [100] to bioinformatics [124]. MT learning includes *multi-target regression* (MTR), which addresses the prediction of continuous targets, *multi-label classification* [192] which focuses on binary targets, and *multi-dimensional classification* which describes the prediction of discrete targets [23]. One contribution of this dissertation

will be focused on tackling the multi-target regression problem, also known as *multi-output*, *multi-variate*, or *multi-response regression* [23].

A characteristic of multi-target data is that the outputs have some structure, in the form of inter-relationships, correlations, and dependencies. Although modeling the multi-variate nature and possible complex relationships between the target variables simultaneously is challenging, past empirical work has shown that the targets are more accurately represented by a single multi-target model [45, 68]. The most valuable advantage of using multi-target techniques is that, not only are the relationships between the sample variables and the targets exploited, but the relationships between the targets amongst themselves are as well [12, 45]. This guarantees a better representation and interpretability of real-world problems that produce multiple outputs, unlike a series of *single-target* (or traditional) models [13]. In addition, MT models could also be considerably more computationally efficient to train, rather than training multiple single-target models individually [69].

Several methods have been proposed for solving such multi-target tasks and can be categorized into two groups. The first being *problem transformation* methods, also known as *local* methods, in which the multi-target problem is transformed into multiple single-target (ST) problems, each solved separately using standard classification and regression algorithms. The second being *algorithm adaptation* methods, also known as *global* or *big-bang* methods, which adapt existing traditional algorithms to predict all the target variables simultaneously [23]. It is known that algorithm adaptation methods outperform problem transformation methods, however they are deemed to be more challenging since they predict, model, and interpret multiple outputs simultaneously.

Multi-instance learning (MIL) is a generalization of supervised learning that has been recently been gaining interest because of its applicability to many real-world problems such as image classification and annotation [91], human action recognition [187], predicting student performance [189], and drug activity prediction [58]. The difference between MIL and traditional learning is the nature of the data. In the multi-instance classification setting, a

sample is considered a *bag* that contains multiple *instances* and is associated with a single label. The individual instance labels within a bag are unknown and bag labels are assigned based on a multi-instance assumption, or hypothesis. Introduced by Dietterich et. al. [58], the *standard MI assumption* states that a bag is labeled positive if and only if it contains at least one positive instance, and is negative otherwise. In other words, the bag-level class label is decided by the disjunction of the instance-level class labels. Other hypotheses have been proposed by Foulds and Frank [70] to encompass a wider range of applications with MI data, but for the scope of this thesis, the focus will be on the standard MI assumption.

Multi-instance classification methods are typically categorized on how the information within the data is exploited. Under the *Instance-Space* (IS) paradigm, discriminative information is considered to be at the instance level, where instance-level classifiers aim to separate the instances from positive bags from those in negative ones. Given a new bag, the classifier will predict the bag-label by aggregating the instance-level scores using some MI assumption. The IS paradigm is based on *local*, or instance-level information, where learning is not concerned with global characteristics of the entire bag. Unlike the IS paradigm, the *Bag-Space* (BS) paradigm considers the information provided of the bag as whole, also known as *global*, or *bag-level* information. Another approach for dealing with multi-instance data falls under the *Embedded-Space* (ES) paradigm, where each bag is mapped to a single feature vector, which summarizes the information contained within each bag. The original bag space is mapped to a vector space, where a classifier is then trained. Under this paradigm, the multi-instance problem is transformed into a traditional supervised learning problem, where any classifier can then be applied.

A recent survey [43] organized the various problems and complexities associated with MIL into four broad categories: *Prediction level*, *Bag composition*, *Label ambiguity* and *Data distribution*; each raising different challenges. As mentioned previously, when instances are grouped into bags, predictions can be performed at two levels: the bag-level or the instance level. Certain types of algorithms are often better suited for one of these two types of

predictions. The composition of each bag, such as the proportion of instances of each class and the relationship between instances also affects the performance of MIL methods. The ambiguity amongst the instance labels stemming from label noise and unclear relationships between an instance and its class is another complexity that should be considered. Finally, the underlying distributions of positive and negative classes affects MIL algorithms depending on their assumptions about the data.

One of the major complexities that this thesis will be tacking is dealing with the ambiguity of the relationship between a bag label and the instances within the bag. This issue stems from the standard MI assumption, where the underlying distribution among instances within positive bags is unknown. There have been different attempts to overcome this complexity, such as "flattening" the MIL datasets, meaning instances contained in positive bags each adopt a positive label, allowing the use of classical supervised learning techniques [142]. This approach assumes that positive bags contain a significant number of positive instances, which may not be the case, causing the classifier to mislabel negative instances within the bag, decreasing the power of the MI model. To overcome this issue, a different MIL approach was proposed, where subsets of instances are selected from positive bags for classifier training [126]. One drawback of this type of method is that the resulting training datasets become imbalanced towards positive instances. Model performance further deteriorates when more instances are selected as subsets than needed [44]. The MIL contribution of this thesis aims to deal with these drawbacks by minimizing class imbalance, which is achieved by optimally selecting bag-representatives from both classes.

The Data Stream learning paradigm has become a more pragmatic area of research recently with the prevalence and advancements in software and hardware technologies, which have vastly increased the amount and frequency of available data [78]. The classic machine learning process, whether it be for traditional supervised learning or non-traditional learning paradigms, is divided into two phases: model building and model testing from static datasets. It is often assumed that the data generating process is stationary, i.e. the data are drawn from

a fixed, yet unknown probability distribution, however, in many real-world scenarios, this might not be the case. Rather, data are now being made available in an online, or streamed fashion, and are usually generated by an evolving, or drifting, phenomenon [55, 59, 197]. The former is described as a *stationary* stream and the latter, *non-stationary*. These drifts can be due to a number of events, such as seasonality effects, changes in user preferences, or hardware/software faults. Due to the fluid nature of the data generation environment, where the probabilistic characteristics of data can change over time, traditional machine learning methods will be bound to perform sub-optimally at best or completely fail at worst. These data complexities prompted the need for effective, efficient, and accurate algorithms for learning from stationary streamed data, as well as adapting to, drifting environments.

Applications of data stream classification can vary from astronomical and geophysical operations [40] to real-time recommender systems for business and industrial uses [104, 105]. Adapting traditional classification methods to these types of scenarios is usually a non-trivial task. The algorithms need to perform classification immediately upon request, since it may not be possible to control the rate at which test samples arrive. Another hurdle stems from the possibility of drifting concepts, and if they occur, the classifier will most likely become outdated after a period of time [82]. There are two popular strategies commonly used when learning from non-stationary data streams, commonly referred to as *active* and *passive* approaches [59, 67]. They differ in their employed methods for adapting to a possibly evolving data stream. The active approach relies on an explicit drift detector in order to utilize an appropriate adaptation mechanism, while the passive approach continuously updates the model over time when data is received, without the need for an explicit drift detector. Deciding which approach to utilize depends on the application (whether there are sudden concept drifts, or if the data arrive online or in batches), the computational resources that are available, and any prior assumptions about the data distribution [4].

Generally, passive approaches have been shown to be more effective in classification settings where there are gradual drifts [67]. Although detecting and deal with gradual drifts can

16

be done by active approaches, the change detection is considerably more difficult [5]. Active approaches work well in settings where the concept drift is abrupt. Additionally, passive approaches generally perform better in batch learning settings, whereas active approaches have been shown to work well in the online setting [15, 83].

The contributions of this thesis aim to deal with the drawbacks that exist within these non-traditional learning paradigms, using traditional and a novel solvers for support vector machines. *Support vector machines* (SVMs), proposed by Cortes and Vapnik [54], represent popular linear and non-linear (kernelized) learning algorithms based on the idea of a large-margin classifier. They have been shown to improve generalization performance for binary classification problems. SVMs are similar to other machine learning techniques, but literature shows that they usually outperform them in terms of scalability, computational efficiency, and robustness against outliers. They are known for creating sparse and non-linear classifiers, making them suitable for handling large datasets.

A traditional approach for training SVMs is the *Sequential Minimal Optimization* (SMO) algorithm [138], a method for solving the L1-SVM's *Quadratic Programming* (QP) task. Although SMO provides an exact solution to the SVM QP problem, its performance is highly dependent on the SVM hyperparameters. More recent approaches, which have been shown to surpass SMO in terms of scalability while remaining competitive in accuracy, include the LASVM algorithm [24], the *Minimal Norm SVM* (MNSVM) [159] and the *Non-Negative Iterative Single Data Algorithm* (NNISDA) [110, 198]. With all the various efforts aimed at solving the SVM task efficiently, this area is still requires investigation.

To deal with issues of scalability, this thesis introduces a different approach which focuses on the minimization of the regularized L1-SVM through *Stochastic Gradient Descent* (SGD), a well-known simple, yet efficient technique for learning classifiers under convex loss functions. Recently, SGD algorithms have been shown to have considerable performance and generalization capabilities in the context of large-scale learning [27], and have been used to solve the SVM problem, such as *NORMA* [113] and *PEGASOS* [153, 194].

17

Although stochastic and iterative algorithms are very simple to implement and efficient, they also have their limitations. One of these limitations is the lack of meaningful stopping criteria for the algorithm; without a pre-specified number of iterations to train, the algorithms continue running [135]. Another limitation stems from the superlinear increase in training time as the number of samples increases. Incremental algorithms attempt to alleviate this issue, but they cannot guarantee a bound on the number of updates per iteration.

1.1 Contributions of the Dissertation

The current leading MT models are based on ensembles of regressor chains, where random, differently ordered chains of the target variables are created and used to build separate regression models, using the previous target predictions in the chain. The challenges of building MT models stem from trying to capture and exploit possible correlations among the target variables during training, at the expense of increasing the computational complexity of model training. One of the contributions of this thesis aims to investigate the performance changes when building a regression model using two distinct chaining methods versus building independent single-target models for each target variable using a novel framework. Specifically, this MTR contribution includes:

- Evaluating the performance of a *Support Vector Regressor* (SVR) as a multi-target to single-target *problem transformation* method to determine whether it outperforms current popular ST algorithms. Its performance is analyzed as a base-line model for MT chaining methods due to the fact that ST methods do not account for any correlation among the target variables.

- Building an MT ensemble of randomly chained SVR models (SVRRC), an approach inspired by the chaining classification method in [157], to investigate the effects of exploiting correlations among target variables during model training. The main issues to be investigated with this approach are the *randomness* of the created chains: they might not capture all targets' correlations, and the time taken to build the ensemble.

18

- Proposing an MT *algorithm adaptation* model of SVRs that builds a unique chain, capturing the maximum correlation among target outputs, named *SVR Correlation Chains* (SVRCC). The advantages of using this approach include exploiting the correlations among the targets which leads to an improvement in model prediction performance, and a reduction in computational complexity because a single SVR-chain model is trained, rather than building an ensemble of 10 base regressors.

To address the limitations presented by MIL algorithms, this thesis proposes a novel SVM formulation with a bag-representative selector, called *Multiple-Instance Representative Support Vector Machine* (MIRSVM). The algorithm does not assume any distribution of the instances and is not affected by the number of instances within a bag, making it applicable to a variety of contexts. The key contributions of this work include:

- Reformulating the traditional primal L1-SVM problem to optimize over bags, rather than instances, ensuring all the information contained within each bag is utilized during training, while defining bag representative selector criteria.

- Deriving the dual multi-instance SVM problem, with the Karush-Kuhn-Tucker necessary and sufficient conditions for optimality. The dual is maximized with respect to the Lagrange multipliers and provides insightful information about the resulting sparse model. The dual formulation is kernelized with a Gaussian radial basis function, which calculates the distances between bag representatives.

- Devising a unique bag-representative selection method that makes no presumptions about the underlying distributions of the instances within each bag, while maintaining the default MI assumption. This approach eliminates the issue of class imbalance caused by techniques such as flattening or subsetting positive instances from each bag. The key feature of MIRSVM is its ability to identify instances (support vectors) within positive and negative bags that highly impact the model.

To address the limitations presented by current popular SVM solvers, this thesis proposes a novel *OnLine Learning Algorithm using Worst-Violators* (OLLAWV). This unique method iterates over samples, updates the model, and utilizes a novel stopping criterion. The model is updated by iteratively selecting (without replacement) the worst violating sample, i.e. the sample with the largest error according to the current decision function, and stops training when there are no more violating samples left to update. In other words, the algorithm is implicitly identifying support vectors and stopping when it has found them all. Because samples are selected and updated without replacement, coupled with the fact that the maximum number of iterations never exceeds the size of the dataset, OLLAWV does not use the regularization updating term. Instead, the regularization is achieved by early stopping. In [53], it has been shown that the smaller the number of updates (determined here by the proposed stopping criterion) is, the larger will be the margin. On the other hand, the larger the margin, the better generalization of the model is. The experimental results presented here confirm both the theoretical statements in [53] and the validity of the approach proposed and taken in OLLAWV algorithm. Combining this method of updating the model and stopping criteria with the fact that SVMs are known for creating sparse kernel classifiers, the contribution aims to speed up the model training time without sacrificing the model's accuracy. The key contributions of this work include:

- Devising a unique iterative procedure for solving the L1-SVM problem, as well as a novel method for identifying support vectors, or *worst-violators*. Rather than randomly iterating over the data samples, OLLAWV aims to reduce training time by selecting and updating the samples that are most incorrectly classified with respect to the current decision hyper-plane.

- Designing a novel stopping criteria by utilizing the worst-violator identification method. This aims to eliminate the added parameterization that is included with most online methods, where the number of iterations of the algorithm needs to be set in advance. Once there are no incorrectly classified samples left, the algorithm terminates.

The current leading data stream approaches are based on ensembles of various classification algorithms, used in conjunction with drift detection mechanisms [18, 121]. To address the limitations of these current popular solvers and due to the infrequent investigation of SVMs within the data stream environment [1], this thesis proposes a novel implementation and experimental study of two online algorithms: *OnLine Learning Algorithm - List 2* (OLLA-L2) [106] and its successor OLLAWV, within the batched data stream classification environment (implemented in the *Massive Online Analysis* (MOA) [17] framework). One contribution aims to investigate the performance of OLLA-L2 because of its simplicity, online nature, and superior performance over other popular SVM solver. A novel preliminary drift detection mechanism was also designed and implemented to try and improve the performance of OLLA-L2 when encountering drifting streams. The second contribution was to compare the novel and newly completed OLLAWV with OLLA-L2 and its drift detector. The main aim of these contributions is to provide a baseline streamed, online SVM that satisfies the computational, memory, and time complexities that come with the data stream classification problem. The key contributions include:

- Implementing OLLA-L2 for solving the L1-SVM problem within for batched data streams, and a novel, preliminary change-detection tool. Rather than blindly rebuild the model when a new batch arrives from the stream, the algorithm monitors the change in error and determines whether to update/rebuild/retain the model.

- Developing the novel OLLAWV for batch data stream learning. Due to OLLAWV's speedy convergence, novel stopping criteria, worst-violator identification mechanism, and its ability to produce sparse models, OLLAWV seemed like a prime contender for being used in a batch streamed environment.

CHAPTER 2

BACKGROUND

Support vector machines represent a popular set of learning techniques that have been introduced under Vapnik-Chervonenkis theory of *structured risk minimization* (SRM) [26, 54, 107, 150, 151]. SRM is an inductive principle for the purpose of model selection. It minimizes the expected probability of error, resulting in a generalized model, without making assumptions about the data distribution [151, 173]. This is the basis for developing the maximal margin classifier [173]. Based on the work of Aizerman et. al. [2], Boser et. al. [26] generalized the linear algorithm to the non-linear case. Then, Cortes and Vapnik [54] proposed the *soft-margin* SVM; a modification that not only allowed maximal margin classifiers to be applied to non-linearly separable data, but also introduced a regularization parameter to prevent overfitting and gauge generalizability. That same year, the algorithm was extended by Vapnik and his coworkers [171] to the regression case.

This chapter presents a theoretical background of support vector machines. First, the SVM paradigm is discussed in the context of classification, introducing the concepts of the maximal margin, the linear soft-margin SVM for overlapping classes, and its kernelized version. Next, Vapnik's ϵ-insensitivity loss function and the concept of *support vector regression* (SVR) are introduced. Afterwards, popular methods for solving the SVM problem are presented and their advantages and problems are discussed. Finally, insights into the benefits of using SVMs are presented, along with a short comparison of SVMs to the classical statistical learning paradigm.

Fig. 2.1.: A 2-dimensional example of different possible separating hyperplanes that correctly classify all the toy data points.

Fig. 2.2.: An illustration of the soft-margin L1-SVM solution on an example 2-dimensional non-linearly separable dataset.

2.1 Support Vector Machine Classification

Supervised learning is the process of determining a relationship $f(\boldsymbol{x})$ by using a training dataset $\mathcal{S} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_i, y_i), \ldots, (\boldsymbol{x}_n, y_n)\}$, which contains n inputs of d-dimensionality, $\boldsymbol{x}_i \in \mathbb{R}^d$, and their class labels y_i. In the case of binary classification, $y_i \in \{+1, -1\}$, where $+1$ and -1 are the two class labels.

The goal of the soft-margin SVM classifier is to find a classification function

$$f(\boldsymbol{x}) = \text{SIGN } o_{\boldsymbol{w},b}(\boldsymbol{x}), \tag{2.1}$$

where $o_{\boldsymbol{w},b}(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x}_i + b$ is a linear decision (output) function representing an affine mapping function $o : \mathbb{R}^d \to \mathbb{R}$ and is parameterized by $\boldsymbol{w} \in \mathbb{R}^d$, the weight vector, and $b \in \mathbb{R}$, the bias term. In addition, $\boldsymbol{w}$ and b must satisfy the following,

$$y_i (\boldsymbol{w} \cdot \boldsymbol{x}_i + b) \geq 1 - \xi_i, \forall i \in \{1, \ldots, n\}, \tag{2.2}$$

where $\boldsymbol{\xi} \in \mathbb{R}^n$ are the non-negative slack variables that allow for some classification error

to account for overlapping datasets. The minimal distance between points belonging to opposite classes and the hyperplane is defined as the *margin* and has a width equal to $\frac{2}{||\boldsymbol{w}||}$, which is why the $||\boldsymbol{w}||$ must be minimal in order to maximize the margin.

In the example shown in Figure 2.1, if the training data points are slightly moved, the solid line (with the larger margin) will still correctly classify all the instances, whereas the dotted line (with a much smaller margin, comparatively) will not. This illustrates that the location of the hyperplane has a direct impact on the classifiers generalization capabilities. The hyperplane with the largest margin is called the *optimal separating hyperplane*. Figure 2.2 shows the optimal separating hyperplane for overlapping training data points, where the filled data points are from the $+1$ class and the non-filled data points are from the -1 class. The training data points on the separating hyperplane (the circled data points), whose decision function value equals $+1$ or -1, are called the *support vectors*.

The soft-margin SVM is a result of the following optimization problem:

$$
\begin{aligned}
\min_{(\boldsymbol{w},b)} \quad & \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^{n}\xi_i \\
\text{s.t.} \quad & y_i\left(\boldsymbol{w}\cdot\boldsymbol{x}_i + b\right) \geq 1 - \xi_i, \ \forall i \in \{1,\ldots,n\} \\
& \xi_i \geq 0, \ \forall i \in \{1,\ldots,n\},
\end{aligned}
\tag{2.3}
$$

where the penalty parameter $C \in \mathbb{R}$ controls the trade-off between margin maximization and classification error minimization, penalizing large norms and errors. Note that Equation 2.3 is a classic quadratic optimization problem with linear constraints, and consequently, it has a unique solution. In geometric terms of SVM, this means that there is a one unique separation boundary in the input space with a maximal margin.

Equation 2.3 can be rewritten as a regularized loss minimization problem by representing the constraints as the Hinge loss, given by:

$$
L(y_i, o_{(w,b)}(\boldsymbol{x}_i)) = \max\left\{0, 1 - y_i o_{(w,b)}(\boldsymbol{x}_i)\right\},
\tag{2.4}
$$

which penalizes errors satisfying the following: $y_i o_{(w,b)}(\boldsymbol{x}_i) < 1$ and is a crucial element that

facilitates the SVM model's sparseness. The soft-margin SVM represented as a regularized loss minimization problem becomes:

$$\min_{(\boldsymbol{w},b)\in\mathcal{H}_o\times\mathbb{R}} R = \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^{n} L(y_i, o_{(w,b)}(\boldsymbol{x}_i)), \tag{2.5}$$

where $\mathcal{H}_o$ is a general Hilbert space. To handle cases when the data are non-linearly separable, while enhancing the classifier's generalization capabilities, a kernel function can be used [2], as shown in Equation 2.6:

$$\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j)\rangle, \tag{2.6}$$

where $\phi(\cdot)$ represents a mapping function from the original feature space to a higher dimensional space. The advantage of utilizing kernels is being able to calculate the inner product in the input space rather than in the very high feature dimensional space (including the infinite dimensional ones). The SVM model output, o shown in Equation 2.7, for a given input vector $\boldsymbol{x}$ is defined by the kernel as given below:

$$o(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i \mathcal{K}(\boldsymbol{x}, \boldsymbol{x}_i) + b, \tag{2.7}$$

where $\alpha_i \in \mathbb{R}$ are the coefficients, or weights, of the expansion in feature space, and $b \in \mathbb{R}$ is the so-called bias term. Note that if a positive definite kernel is used, there is no need for a bias term b, but b can nevertheless be used. The two terms, $\boldsymbol{\alpha}$ and b, parametrize the SVM model. A model is called *dense* if the absolute value of all its weights are greater than 0, while a *sparse* model would be one that contains some $\alpha_i = 0$. The level of sparseness may vary, but the sparser the model, the more scalable the applications.

2.2 Support Vector Regression

The support vector machine was applied to the regression case [64, 172], maintaining all the maximal margin algorithmic features. Unlike pattern recognition problems where the desired outputs y_i are discrete, for the regression case they are continuous, real-valued,

function outputs. Given training dataset $\mathcal{S} = \left\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n) \in \mathbb{R}^d \times \mathbb{R}\right\}$, where $y_i \in \mathbb{R}$ is the continuous output of input $\boldsymbol{x}_i \in \mathbb{R}^d$, the goal is to learn a function $f(\boldsymbol{x})$ with at most ϵ deviation from the true targets y_i for all the training data, while being as flat as possible. This was introduced by Vapnik's linear loss function with ϵ-insensitivity zone, illustrated in Figure 2.3 and given by:

$$|y_i - o_{(w,b)}(\boldsymbol{x_i})|_\epsilon = \begin{cases} 0 & if |y_i - o_{(w,b)}(\boldsymbol{x_i})| \leq \epsilon \\ |y_i - o_{(w,b)}(\boldsymbol{x_i})| - \epsilon & \text{otherwise.} \end{cases} \tag{2.8}$$

The loss is equal to 0 if the difference between the predicted and true output values is less than ϵ. Vapnik's ϵ-insensitivity function, shown in Equation 2.8, defines an ϵ-tube, illustrated in Figure 2.4. If the predicted value is within the tube, no loss is incurred [107]. Estimating a linear regression hyperplane is achieved by minimizing:

$$\min_{(\boldsymbol{w},b)\in\mathcal{H}_o\times\mathbb{R}} R = \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^{n}(|y_i - o_{(w,b)}(\boldsymbol{x_i})|_\epsilon). \tag{2.9}$$

Equation 2.9 is equivalent to the following, where non-negative slack variables are introduced:

$$\min_{(\boldsymbol{w},b,\boldsymbol{\xi},\boldsymbol{\xi^*})} \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*)$$

$$\text{s.t.} \quad y_i - \boldsymbol{w}\cdot\boldsymbol{x}_i - b \leq \xi_i + \epsilon, \ \forall i = \{1,\ldots,n\} \tag{2.10}$$

$$\boldsymbol{w}\cdot\boldsymbol{x}_i + b - y_i \leq \xi_i^* + \epsilon, \ \forall i = \{1,\ldots,n\}$$

$$\xi_i, \xi_i^* \geq 0, \ \forall i = \{1,\ldots,n\}.$$

Note that the constant C influences the trade-off between approximation error and the model generalizability, similar to the classification setting. The optimization function given in 2.10 can be solved more easily in its dual formulation and is key to extending the SVR to learn from non-linear functions [150].
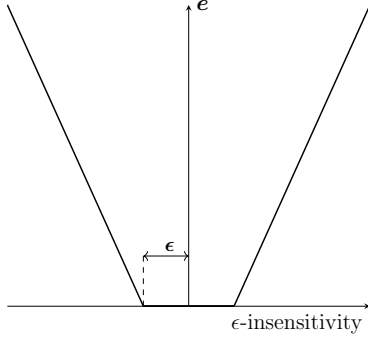
Fig. 2.3.: Vapnik's ϵ-insensitivity loss function.

Fig. 2.4.: Linear support vector regression example solution on a toy 2D dataset.

The dual SVR kernelized cost function is as follows:

$$\max_{(\boldsymbol{\alpha},\boldsymbol{\alpha}^*)} \quad -\frac{1}{2}\sum_{i,j=1}^{n}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) - \epsilon\sum_{i=1}^{n}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{n}y_i(\alpha_i - \alpha_i^*)$$
$$\text{s.t.} \quad \sum_{i=1}^{n}(\alpha_i - \alpha_i^*) = 0 \tag{2.11}$$
$$\alpha_i, \alpha_i^* \in [0, C], \forall i = \{1, \ldots, n\},$$

where $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}^*$ correspond to the SVR dual variables.

2.3 Support Vector Machine Solvers

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.

Interior Point (IP) methods pose the SVM learning problem as a quadratic optimization problem subject to linear constraints, which are then replaced with a barrier function [33]. The resulting unconstrained problem can then be optimized using Newton or Quasi-Newton methods. Generally, interior point methods are good choices for small sized learning problems. However, because IP methods typically involve Cholesky decomposition, they require a run time of $[O(n^3)]$. Moreover, their memory requirements are $[O(n^2)]$, rendering the use of IP methods impractical when the training set consists of a large number of samples [154].

The first attempts of overcoming the quadratic memory requirement of IP methods and speeding up their training time were aimed at *decomposing* the underlying SVMs quadratic programming problem. First, Boser et al. [26] implemented Vapnik's *chunking* method. Osuna et al. [132] later introduced a theorem on the convergence of dividing large QP problems into a series of smaller sub-problems. The theorem states that the chunking method will converge to the globally optimal solution.

Sequential Minimal Optimization (SMO) by Platt [138], its improvement by Keerthie et al. [111], and SVM-Light [101] are alternative approaches to decomposing the QP problem. SMO, implemented in the popular, widely used software package LIBSVM [47], is an iterative procedure that divides the SVM dual problem into a series of sub-problems, which are solved analytically by finding the optimal α values that satisfy the Karush-Kuhn-Tucker conditions [33]. Although SMO is guaranteed to converge, heuristics are used to choose α values in order to accelerate the convergence rate. This is a critical step because the convergence speed of the SMO algorithm is highly dependent on the dataset size and SVM hyperparameters [150].

Most existing approaches, including the methods described above, focus on solving the dual of Equation 2.3, for two main reasons: firstly, the dual formulation provides a convenient way of dealing with the constraints. Secondly, the dual formulation can be written in terms of dot products, allowing the use of kernel functions [31]. However, these two conveniences are not restrictions for solving *primal* SVM problem. For example, Chappelle [48] showed that even though optimizing the primal and dual are equivalent in terms of solutions and time, optimizing the primal when dealing with approximate solutions is far more superior. The primal problem can be cast as an unconstrained problem by using linear or non-linear kernels and the Representer theorem [149]; mainly, reparametrizing the weight vector. Chapelle [48] investigated solving the primal objective with smooth loss functions, rather than using the hinge loss function, and suggested using methods such as conjugate gradient descent and Newton's method.

Among typical techniques for solving QP problems are *active set* methods [181], which have also been applied for solving the SVM problem [46, 89, 148]. These are iterative methods, where constraints are divided into the sets of active and inactive constraints. They iteratively update the active set until the optimal solution is found, i.e. when the variables are no longer actively constrained. Although most active set methods have theoretical guarantees to converge in finite time [148, 155], they are typically computationally expensive, especially when the number of support vectors is large [154].

Some advancements in handling large scale problems are based on a *geometric* interpretation of SVM problem. Some of these geometric SVMs include approaches that use convex hulls [14] and minimum enclosing balls such as *Core Vector Machines* (CVM) [167]. Tsang et al. [166] later improved the scalability of CVMs by introducing *Ball Vector Machines* (BVM) which do not require a QP solver. Other geometric approaches include the novel algorithms introduced by Strack [159], known as the *Sphere Support Vector Machine* (SphereSVM) and *Minimal Norm Support Vector Machine* (MNSVM), which utilize the connection between minimal enclosing balls and convex hull problems, while demonstrating a high capability for learning from large datasets.

The *Non-Negative Iterative Single Data Algorithm* (NNISDA) [198] is an efficient approach for solving the SVM problem, shown to be faster than SMO and equal in terms of accuracy [110]. NNISDA is an iterative algorithm that finds a solution to the L2-SVM using *coordinate descent*, inspired by *Iterative Single Data Algorithm* (ISDA) [96], which was originally introduced in [108]. Coordinate descent is a popular optimization method that has been widely used for solving the SVM problem [25, 74, 95, 125]. Note, the SMO algorithm also utilizes a coordinate descent method, however it is done through coordinate pairs [154].

Recently, several authors have proposed the use of a standard *stochastic* (or *online*) *gradient descent* (SGD) approach for SVMs to optimize large-scale learning problems [90, 112, 150, 151, 153]. Kivinen et al.[113] and Bousquet and Bottou [32] showed that stochastic algorithms can be both the fastest, and have the best generalization performances. It has

also been shown that the SGD run time for solving the SVM unconstrained primal problem is inversely proportional to the size of the training set [135, 152]. Shalev-Shwartz and Ben-David [151] have 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 the classic *kernel perceptron* algorithm [53].

Notable representatives of this method of learning include the *Naïve Online R Minimization Algorithm* (NORMA) by Kivinen et al. [113] and the *Primal Estimated Sub-Gradient SOlver for Support vector machines* (PEGASOS) by Shalev-Shwartz et al. [153]. 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 [113]. 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, named *OnLine Learning Algorithm* (OLLA) [106] is a unification, simplification, and expansion of the somewhat similar approaches presented in [90, 112, 150, 151, 153] and [109, 129, 127]. 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 [53] 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 in [106], but for the scope of this thesis the L1-SVM was mentioned.

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 unknown parameter that requires tuning is the gradient step size, which in some cases, directly affects the algorithm convergence rate. Moreover, a third disadvantage of kernelized online algorithms is that the training time for each update increases superlinearly with the number of samples [28].

2.4 Why Support Vector Machines: Form & Norm

The support vector machine problem shares similarities to classical statistical inference such as *Neural Networks* (NNs), however, there are several very important differences between their approaches and assumptions.

One of the major differences is that these traditional classification and regression statistical techniques are based on the strict assumption that the data distribution is known and is that of a Gaussian distribution. Another assumption is that this data can be modeled by a set of linear parameter functions. Following this, the induction paradigm for parameter estimation is the maximum likelihood estimation method, which can be reduced to the minimization of the sum-of-errors-squared cost function [107].

The previously stated assumptions, on which the classic statistical paradigm relied, turned out to be inappropriate for many contemporary problems for a couple of reasons [172] . Real-world problems are, more often than not, high-dimensional. If the underlying mapping is not smooth, the linear paradigm needs an exponentially increasing number of terms, thus increasing the dimensionality of the input space, also known as '*the curse of dimensionality*'. Another issue is that the data generation process might be very different from the normal distribution. Due to these grave concerns, the maximum likelihood estimator or sum-of-errors-squared cost function, should be replaced by a new induction paradigm to be

able to model non-Gaussian distributions, or rather, to have a distribution-free method of classification or regression for high-dimensional, sparse data [107]. This was the foundational reason for developing support vector machines.

The main differences between support vector machines and classical statistical techniques, such as neural networks, can be identified by analyzing their *form* and *norm*. With respect to both types of models' *form*, the greatest majority of machine learning models are the same, i.e. they are represented as the sum of weighted basis functions. The difference in the two approaches stems from their *norm* (cost functions) and how these models learn their function parameters (e.g. how many functions should be used, what their parameters are, what the value of their weights should be).

For example, neural networks minimize the sum-of-errors-squared in output space, i.e. the L2 norm, and SVMs maximize the margin in input space by minimizing the L2 norm of the weight vector. For SVMs, the model parameters are not predefined and their number depends on the training data used. Rather than choosing the appropriate structure of the model, keeping the estimation error fixed, and minimizing the training error, as done by classical techniques, SVMs keep the training error fixed or set to some appropriate level and minimize the estimation error. This is the paradigm of *structural risk minimization* (SRM) introduced by Vapnik and Chervonenkis and their colleagues, which led to the new learning algorithm. This approach has been proven, both experimentally and theoretically, to be superior (or comparable) to NNs and other statistical methods for contemporary real-world problems, which are often very sparse datasets (i.e. small number of samples in high dimensional spaces).

CHAPTER 3

MULTI-TARGET SVR USING MAXIMUM CORRELATION CHAINS

This chapter presents three multi-target *support vector regression* (SVR) models. The first involves building independent, single-target SVR models for each output variable. The second builds an ensemble of random chains using the first method as a base model, named *SVR with Random Chains* (SVRRC), inspired by the classification MT method, *Ensemble of Random Chains Corrected* (ERCC) [157]. The third calculates the targets' correlations and forms a maximum correlation chain, which is used to build a single chained model named *SVR with Correlation Chaining* (SVRCC). The experimental study compares the performance of the three approaches with six other prominent MT regressors. The experimental results are then analyzed using non-parametric statistical tests. The results show that the maximum correlation SVR approach improves the performance of using ensembles of random chains.

This chapter is organized as follows: Section 3.1 describes the notation used throughout this chapter and reviews related works on multi-target regression. Section 3.2 presents the three multi-target support vector regression approaches. Section 3.3 presents the experimental study. Section 3.4 discusses the results and the statistical analysis. Finally, Section 3.5 shows the main conclusions of this work.

3.1 Multi-Target Regression Background

This section first defines the notation that will be used throughout this chapter, and then formally describes the multi-target regression problem along with relevant popular algorithms used within this paradigm.

Table 3.1.: Summary of Multiple-Target Learning Notation

Definition	Notation
Number of Samples	$\mathcal{N}$
Number of Input Attributes	d
Input Space	$\boldsymbol{X} \in \mathbb{R}^{\mathcal{N} \times d}$, $1 \leq i \leq d$
Input Instance	$\boldsymbol{x}^{(l)} = (x_1^{(l)}, \ldots, x_d^{(l)}) \in \boldsymbol{X}$, $1 \leq l \leq \mathcal{N}$
Number of Dataset Targets/Outputs	m
Target Space	$\boldsymbol{Y} = \{\boldsymbol{Y}_1, \ldots, \boldsymbol{Y}_j, \ldots, \boldsymbol{Y}_m\} \in \mathbb{R}^{\mathcal{N} \times m}$, $1 \leq j \leq m$
Predicted Target Space	$\hat{\boldsymbol{Y}} = \{\hat{\boldsymbol{Y}}_1, \ldots, \hat{\boldsymbol{Y}}_j, \ldots, \hat{\boldsymbol{Y}}_m\} \in \mathbb{R}^{\mathcal{N} \times m}$, $1 \leq j \leq m$
Target Instance	$\boldsymbol{y}^{(l)} = (y_1^{(l)}, \ldots, y_m^{(l)}) \in \boldsymbol{Y}$, $1 \leq l \leq \mathcal{N}$
Full Multi-Target (MT) Training Dataset	$\mathcal{D} = \{(x_1^{(1)}, y_1^{(1)}), \ldots, (x_d^{(\mathcal{N})}, y_m^{(\mathcal{N})})\}$
Single-Target (ST) Dataset with j^{th} Target	$\mathcal{D}_j = \{(x_1^{(1)}, y_j^{(1)}), \ldots, (x_d^{(\mathcal{N})}, y_j^{(\mathcal{N})})\} \in \mathcal{D}$, $1 \leq j \leq m$
Number of Cross-Validation (CV) Sets	k
ST Test Dataset with j^{th} Target, i^{th} CV Fold	$\mathcal{D}_j^{(i)} = \{(x_1^{(i)}, y_j^{(i)}), \ldots, (x_d^{(i)}, y_j^{(i)})\} \in \mathcal{D}_j$, $i \in \{1, \ldots, \mathcal{N}\}$
ST Training Dataset with j^{th} Target, Excluding the ith CV Fold	$\mathcal{D}_j^{(k-i)} = \mathcal{D}_j \setminus \mathcal{D}_j^{(i)}$
ST Regression Model	$h : \boldsymbol{X} \times \boldsymbol{Y}$
MT Regression Model	$h_j : \boldsymbol{X} \times \boldsymbol{Y}_j$, $1 \leq j \leq m$
Unknown Sample	$\boldsymbol{x}^{(\mathcal{N}')} = \{\boldsymbol{x}^{(\mathcal{N}+1)}, \ldots, \boldsymbol{x}^{(N')}\}$
Predicted Values for Unknown Sample	$\boldsymbol{y}^{(\mathcal{N}')} = \{\boldsymbol{y}^{(\mathcal{N}+1)}, \ldots, \boldsymbol{y}^{(N')}\}$

3.1.1 Notation

Let $\mathcal{D}$ be a training dataset of n instances. Let $\boldsymbol{X} \in \mathcal{D}$ be a matrix consisting of d input variables and n samples, such that $\boldsymbol{X} \in \mathbb{R}^{n \times d}$. Let $\boldsymbol{Y} \in \mathcal{D}$ be a matrix consisting of m continuous target variables and n samples, where $\boldsymbol{Y} \in \mathbb{R}^{n \times m}$. Table 3.1 summarizes the notation used throughout this chapter.

3.1.2 Multi-Target Regression Methods

As mentioned previously, there are two main approaches to solving multiple-output problems: *problem transformation* and *algorithm adaptation.* This section will present the theory behind both approaches, their advantages and disadvantages, as well as current popular solvers.

Problem transformation methods are mainly based on training m independent, single-target models for each target output on datasets $\mathcal{D}_j = \{\boldsymbol{X}, \boldsymbol{Y}_j\}$, $\forall j \in \{1, \ldots, m\}$ and concetenating all m predictions. The *single-target* method [157], also known as binary relevance in literature [192], simply does exactly that, and is considered as a baseline for measuring

the performance of other problem transformation approaches. Since this approach divides the multi-target problem into m single-target ones, any off-the-shelf traditional regression algorithm can be used. Examples include *ridge regression* [93], *regression trees* [34], and *support vector regression* [64].

The main drawback with single-target approaches in the multi-target setting, is that the relationships between the targets are lost once independent models are built for each target. This in turn may affect the overall quality of the m predictions [23]. Another drawback of this type of approach is computational complexity: prediction for an unseen sample would be obtained by running each of the m single-target models and concatenating their results.

Recently, Spyromitros-Xioufis et al. [157] proposed extending well-known multi-label classification methods to deal with the multi-target regression problem, while modeling the targets' dependencies. Inspired by their successful classification counterparts, Spyromitros-Xioufis et al. [157] introduced two novel approaches for multi-target regression: *multi-target regressor stacking* (MTS) and *regressor chains* (RC). These methods involve two stages of learning, the first being building ST models; and the second uses the knowledge gained by the first step to predict the target variables while using possible relationships the targets might have with one another.

Multi-target regressor stacking was inspired by its multi-label classification counterpart [85] and involves two stages of training. The first stage consists of training m independent single-target models, like in ST. In the second step, a second set of m meta models are learned for each target variable, $\boldsymbol{Y}_j$, $1 \leq j \leq m$. These meta models are learned on a transformed dataset, where the input attributes space is expanded by adding the approximated target variables obtained in the first stage, excluding the jth target being predicted.

The regressor chains method, also inspired by an equivalent multi-label classification method [143], is another problem transformation method, based on the idea of chaining a sequence of single-target models. In the training of RC, a random chain (sequence) of the set of target variables is selected and for each target in the chain, models are built sequentially

by using the output of the previous model as input for the next [158], following the order of the chain.

If the default, ordered chain is $C = \{\boldsymbol{Y}_1, \boldsymbol{Y}_2, \ldots, \boldsymbol{Y}_m\}$, the first model $h_1 : \boldsymbol{X} \to \mathbb{R}$ is trained for $\boldsymbol{Y}_1$, as in ST. For the subsequent models $h_{j,j>1}$, the dataset is transformed by sequentially appending the true values of each of the previous targets in the chain to the input vectors. For a new input vector, the target values are unknown. Once the models are trained, the unseen input vector will be appended with the approximated target values, making the models dependent on the approximated values obtained at each step. One of the issues associated with this method is that, if a single random chain is used, the possible relationships between the targets at the head of the chain and the end of the chain are not exploited due to the algorithm's sequential nature.

In the methods described above, the estimated target variables (meta-variables) are used as input in the second stage of training. In both methods, the models are trained using these meta-variables that become noisy at prediction time, and thus the relationship between the meta-variables and target variable is muddied. Dividing the training set into sets, one for each stage, would not help this situation because both methods would be trained on training sets of decreasing size. Due to these issues, *Spyromitros et. al.* proposed modifications, in [157], to both methods that resembles k-fold cross-validation (CV) to be able to obtain unbiased estimates of the meta-variables. These methods are called *Regressor Chains Corrected* (RCC) and *Multi-Target Stacking Corrected* (MTSC).

However, these corrections did not solve all the methods' problems. One problem with the RC and RCC methods was that they are sensitive to the chain ordering. To remedy this issue, Spyromitros-Xioufis et al. [157] proposed the *Ensemble of Regressor Chains* (ERC) and *Ensemble of Regressor Chains Corrected* (ERCC). Instead of a single chain, $k \leq 10$ chains are created at random, and the final prediction values are obtained by taking the mean values of the k predicted values for each target. The ERC, ERCC and MTSC procedures involve repeating the RCC and MTS procedures k times, respectively, with k randomly ordered

chains for ERCC, and k different modified training sets for MTSC. The corrected methods exhibited better performance than their original variants, as well as ST models. The ERCC algorithm had the best overall performance, as well as being statistically significantly more accurate of all the methods tested[157].

Many authors have proposed using support vector machines for multi-target learning [23, 184, 185]. One example is that of Zhang et al. [195], who presented a multi-output support vector regression approach based on problem transformation. It builds a multi-output model that considers the correlations between all the targets using the vector virtualization method. Basically, it extends the original feature space and expresses the multi-output problem as an equivalent single-output problem, so that it can then be solved using the single-output *least squares support vector regression machines* (LS-SVR) algorithm [161]. Moreover, other contemporary problem transformation approaches include *Linear Target Combinations for MT Regression* [170].

Algorithm adaptation methods are based on the principle of simultaneously predicting all outputs using a single model which captures all dependencies and internal relationships among them. Using this type of approach provides several advantages over problem transformation methods [36, 117, 156]:

- A single taget model is more interpretable than several single-target models.

- When the targets are correlated, algorithm adaptation methods ensure better predictive performance.

The first attempts at dealing with predicting multiple real-valued targets at the same time are statistical approaches which aim to capture the possible correlations amongst target variables. One example is reduced-rank regression, proposed by Izenman [98], which places a constraint on the elements of estimated *reduced-rank regression* coefficient matrices. Brown and Zidek [38] then proposed a multivariate version of the Hoerl-Kennard ridge regression rule. Recently Similä and Tikka [156] considered the regression problem of modeling several

output variables using the same set of input variables, chosen by their simultaneous variable selection method, named L2-SVS. The importance of an input attribute is measured by the L2-norm of their corresponding regression weights, which are found by minimizing the sum-of-errors-squared.

Maximal margin classifiers have also been transformed to accommodate the multi-output case [168]. Rather than having a single-output support vector regressor be applied independently to each target, several approaches have been proposed to extend the traditional SVR to the multi-output case. One example is that of Vazquez and Walter [174]. They extended the traditional SVR by considering the multi-output version of Kringing called *Cokringing* [52]. The authors show that their multi-target SVR produced better results than building independent SVRs. Another example is Brudnak's [39] proposal of a *vector-valued SVR* (VVSVR). Their method generalizes Vapnik's ϵ-insensitive loss function and regularization function from the scalar-valued case to that of vector-value.

In addition to maximal margin classifiers being extended to the multi-output case, the use of multi-output kernels has also been investigated. Evgeniou and Pontil [69] presented an approach to multi-output learning based on minimizing regularized risk functionals, such as SVMs. They proposed a novel kernel function that uses a parameter μ that couples the targets. Their experiments also supported the fact that using an algorithm adaptation approach does perform well when targets are correlated. However, when the targets are not correlated, Evengiou and Pontil showed that their proposed method reduces to single-target learning when the parameter μ is set to be very large, posing no risk to using their multi-target kernel. Choosing the right value for μ must be found by cross-validation.

Due to the success of using this multi-output kernel approach, Evgeniou et al. [68] extended their earlier results and developed a framework for multi-task learning within the context of regularization in reproducing kernel Hilbert spaces. A drawback of their proposed kernel method is that its computational complexity time is worse than the complexity of solving m independent kernel methods.

Multiple-target regression trees, also known as *multi-variate* or *multi-objective* regression trees, are extensions of the traditional regression tree to the multi-output case. One of the first approaches for building multi-target regression trees was that of De'ath [56], who proposed an extension of the univariate method CART [37] to the multi-output case, dubbed *multi-variate regression trees* (MRTs). The main difference between the traditional CART and its multi-variate extension is the redefinition of the impurity measure of a node to the multi-variate sum-of-squared-errors.

Blockeel et al. proposed *multi-objective decision trees* (MODTs) [21, 117], which are decision trees capable of predicting multiple target attributes at once and are used for multi-objective prediction. Struyf and Džeroski [160] proposed a constraint-based framework for building *multi-objective regression trees* (MORTs). Later, Kocev et al. [115] investigated whether ensembles of multi-objective decision trees could be used to improve the performance of using multiple single-target trees or a single multi-target tree. The ensemble learning techniques used were bagging [34] and random forests [35].

The methods described above were all designed to try to analyze and improve the performance of predicting multiple outputs at once, however there are still outstanding issues to be addressed. Considering the models' predictive performances, the benefits of using MTSC and ERCC instead of the baseline ST are not apparent. In the experimental study performed by Spyromitros-Xioufis et al. [157], the ST method sometimes outperformed their proposed problem transformation approaches. The best explanation for this would be that the targets correlations were not captured due to the *randomized* learning process (chain order). Another issue that these approaches face comes with having a large number of output variables. Due to the ensemble based approach of up to 10 random chains, or solving a large number of single-target problems, the algorithms' computational complexity would suffer. Furthermore, these models do not provide a clear description of the relationship between the input and output variables, as well as the outputs amongst themselves. The contributions of this chapter aim to remedy the mentioned disadvantages.

3.2 Three Novel SVMs for Multi-Target Regression

Three novel models have been implemented for the purposes of multi-target regression. The base model is the SVR model, where m single-target soft-margin *non-linear support vector regressors* (NL-SVR) are built for each target variable $\boldsymbol{Y}_j$.

For NL-SVR, the regularized soft-margin loss function given in equation (2.10) is minimized. This contribution involves solving the dual of this formulation given by (2.11). Using the dual formulation, the multi-target problem is solved by transforming it into m single-target problems, as shown in Algorithm 3.1 and Figure 3.1. This algorithm will output m single-target models, $h_j, \forall j = 1, \ldots, m$, for a given dataset $\mathcal{D}$. It first splits the dataset into m separate ones, $\mathcal{D}_j$, each with a single-target variable $\boldsymbol{Y}_j$, and then builds a distinct SVR model for each of the datasets.

Building m ST models is a good base-line, but as mentioned previously, it does not capture possible correlations between the target attributes during training. If these correlations

$$\mathcal{D}_1 : [\boldsymbol{X}][\boldsymbol{Y}_1] \longrightarrow h_1 : \mathcal{D}_1 \to \hat{\boldsymbol{Y}_1}$$

$$\mathcal{D}_2 : [\boldsymbol{X}][\boldsymbol{Y}_2] \longrightarrow h_2 : \mathcal{D}_2 \to \hat{\boldsymbol{Y}_2}$$

$$\mathcal{D} : [\boldsymbol{X}][\boldsymbol{Y}] \qquad \vdots$$

$$\mathcal{D}_m : [\boldsymbol{X}][\boldsymbol{Y}_m] \longrightarrow h_m : \mathcal{D}_m \to \hat{\boldsymbol{Y}_m}$$
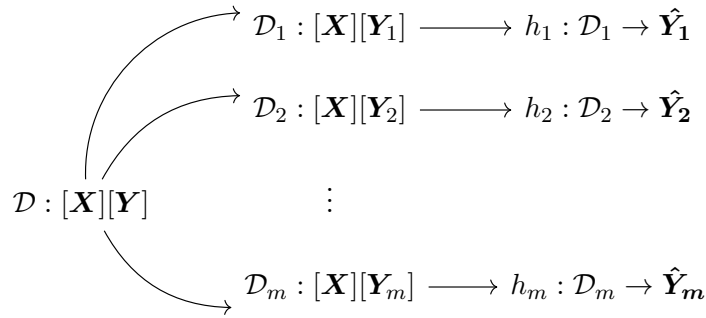
Fig. 3.1.: SVR Flow Diagram. Firstly, the multi-target dataset is divided into m ST datasets, $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_m$. Then m models, $h_1, h_2, \ldots, h_m$, are independently trained for each ST dataset.

Algorithm 3.1 Multi-Target Support Vector Regression (SVR)

Input: Training dataset $\mathcal{D}$
Output: ST models $h_j, j = 1, \ldots, m$
1: **for** $j = 1$ to m **do**
2:     $\mathcal{D}_j = \{\boldsymbol{X}, \boldsymbol{Y}_j\}$ ▷ Get ST data
3:     $h_j : \boldsymbol{X} \to \mathbb{R}$ ▷ Build ST model for the jth target
4: **end for**

Algorithm 3.2 Build Chained Model

Input: Training dataset $\mathcal{D}$, random chain $\boldsymbol{C}$
Output: A chained model $h_j, j = \{1, \ldots, m\}$
1: $\mathcal{D}_1 = \{\boldsymbol{X}, \boldsymbol{Y}_{C_1}\}$ ▷ Initialize first dataset
2: **for** $j = 1$ to m **do**   ▷ For each target in chain $\boldsymbol{C}$
3:      $h_j : \mathcal{D}_j \to \mathbb{R}$ ▷ Train model on appended dataset
4:      **if** $j < m$ **then**
5:          $\mathcal{D}_{j+1} = \{\mathcal{D}_j, \boldsymbol{Y}_{C_j}\}$ ▷ Append new target in chain to dataset
6:      **end if**
7: **end for**

---

are not exploited, this could retract from the model's potential performance. Therefore, creating an ensemble model using a series of random chains was proposed, using the base-line SVR method, named *SVR Random Chains* (SVRRC).

For SVRRC, ensembles of at most 10 m-sized random chains, $\mathcal{C}$, are built from different and distinct permutations of the target variable indices. When chaining target values, there are two main options: using the predicted value as input for the following target, or using the true value of the target variable as input of the subsequent targets. The main problem with the former approach is that errors are propagated throughout the chained model, therefore SVRRC employs chaining of the true values.

For each random chain, a new model is trained by predicting the first target variable in the chain. Next, the first target's true value, $\boldsymbol{Y}_j$, is appended to the training set. This chaining process is repeated for all the target indices in the chains, $\{\boldsymbol{C}_1, \ldots, \boldsymbol{C}_c\} \in \mathcal{C}$, $c \leq 10$. This process will be repeated for each random chain generated, returning an ensemble of chained SVRs. Algorithm 3.2 describes the process of building a chained model given chain $\boldsymbol{C} \in \mathcal{C}$, and Algorithm 3.3 shows the steps taken by SVRRC.

Given this ensemble of chained models, the predicted values for a given unseen instance are calculated by taking the mean of the multiple models generated using different random chains. Since the unseen input has no known target value, the predicted value at each step of the chain $\hat{\boldsymbol{Y}}_j$ is appended to the input at each step of the chain.

Due to the computational complexity of building $m!$ distinct chains and training $(m!) \times m$

41

$$\mathcal{D} : [\boldsymbol{X}][\boldsymbol{Y_1 Y_2 Y_3}]$$
$$\downarrow$$

$$[1, 2, 3] \qquad\qquad [1, 3, 2] \qquad\qquad \ldots \qquad\qquad [3, 2, 1]$$
$$\downarrow \qquad\qquad\qquad \downarrow \qquad\qquad\qquad\qquad\qquad \downarrow$$
$$h_1 : [\boldsymbol{X}] \to \hat{\boldsymbol{Y_1}} \qquad h_1 : [\boldsymbol{X}] \to \hat{\boldsymbol{Y_1}} \qquad \ldots \qquad h_1 : [\boldsymbol{X}] \to \hat{\boldsymbol{Y_1}}$$
$$\downarrow \qquad\qquad\qquad \downarrow \qquad\qquad\qquad\qquad\qquad \downarrow$$
$$h_2 : [\boldsymbol{XY_1}] \to \hat{\boldsymbol{Y_2}} \qquad h_2 : [\boldsymbol{XY_1}] \to \hat{\boldsymbol{Y_3}} \qquad \ldots \qquad h_2 : [\boldsymbol{XY_3}] \to \hat{\boldsymbol{Y_2}}$$
$$\downarrow \qquad\qquad\qquad \downarrow \qquad\qquad\qquad\qquad\qquad \downarrow$$
$$h_3 : [\boldsymbol{XY_1 Y_2}] \to \hat{\boldsymbol{Y_3}} \qquad h_3 : [\boldsymbol{XY_1 Y_3}] \to \hat{\boldsymbol{Y_2}} \qquad \ldots \qquad h_3 : [\boldsymbol{XY_3 Y_2}] \to \hat{\boldsymbol{Y_1}}$$

Fig. 3.2.: SVRRC Flow Diagram on a dataset with three targets. SVRRC first builds the six random chains of the target's indices (three examples are shown). It then constructs a chained model by proceeding recursively over the chain, building a model, and appending the current target to the input space to predict the next target in the chain.

Algorithm 3.3 Multi-Target SVR with Random-Chains (SVRRC)

Input: Training dataset $\mathcal{D}$, c random chains $\mathcal{C}$
Output: An ensemble of chained models $h_\mathcal{C}$
1: **for each** $\boldsymbol{C} \in \mathcal{C}$ **do**   ▷ For each random chain
2:    $h_C \leftarrow \texttt{buildChainedModel}(\mathcal{D}, \boldsymbol{C})$ ▷ build a chained model for chain $\boldsymbol{C}$
3: **end for**

---

models, the number of ensembles and chains are limited to a maximum of 10. However, if the number of target variables is less than 3, i.e. $m! \leq 10$, all $m!$ random chains are constructed.

A disadvantage of building an ensemble of 10 random chains stems from the fact that: when the number of output variables increases, the number of possible chains increases factorially. Therefore, there is no guarantee that the 10 random chains generated will truly reflect the relationships among the target variables. Additionally, building an ensemble of regressors is computationally expensive. Finding a heuristic that allows the identification of a single, most appropriate chain, which fully reflects the output variable interrelations would improve the scalability of training the ensemble.

$$\mathcal{D} : [\boldsymbol{X}][\boldsymbol{Y_1 Y_2 Y_3}] \xrightarrow[\dfrac{\mathbf{E}\left[(Y_i-\mu_i)(Y_j-\mu_j)\right]}{\sqrt{\mathbf{E}\left[(Y_i-\mu_i)(Y_i-\mu_i)\right]\mathbf{E}\left[(Y_j-\mu_j)(Y_j-\mu_j)\right]}}]{\textit{generate maximum correlation chain}} [1,2,3]$$

$$\hookrightarrow h_1 : [\boldsymbol{X}] \to \hat{\boldsymbol{Y_1}} \longrightarrow h_2 : [\boldsymbol{XY_1}] \to \hat{\boldsymbol{Y_2}} \longrightarrow h_3 : [\boldsymbol{XY_1Y_2}] \to \hat{\boldsymbol{Y_3}}$$

Fig. 3.3.: SVRCC Flow Diagram on a sample dataset with three targets. SVRCC first finds the direction of maximum correlation among the targets and uses that order as the only chain. It then constructs the chained model, as done in SVRRC.

Algorithm 3.4 Multi-Target SVR with max-Correlation Chain (SVRCC)

1: $\mathbf{P} = corrcoef(\boldsymbol{Y})$ ▷ Find correlation coefficient matrix for target variables
2: $\boldsymbol{C} = \sum_{i=1}^{n} \mathbf{P}_{ij}, \forall j = 1, \ldots, m$ ▷ Sum row elements of the correlation coefficient matrix
3: $\boldsymbol{C} = \texttt{sort}\,(\boldsymbol{C}, \textbf{decreasing})$ ▷ Sort sums in decreasing order
4: $h_{\boldsymbol{C}} = \texttt{buildChainedModel}(\mathcal{D}, \boldsymbol{C})$ ▷ build a chained model for max correlation chain $\boldsymbol{C}$

The third proposal was designed to remedy this issue. It builds a single chain based on the maximization of the correlations among the target variables. By calculating the correlation of the target variables and imposing it on the order of the chain, this ensures that each appended target provides some additional knowledge on the training of the next. With SVRRC, there is no reasoning behind the generation of these chains, and since the number of random chains generated is limited to 10, there is no way of ensuring that the 10 chains fully represent the targets' dependencies. Calculating and using the correlations of the targets would break this uncertainty. Algorithm 3.4 presents the *SVR maximum Correlation Chain* (SVRCC) method. The computational complexity and hardware constraints (memory size) are negligible during the construction of the targets' correlation matrix, since the correlation matrix would be an $(m \times m)$ matrix, and the likelihood that the number of targets is large enough to cause a memory issue is minimal.

To calculate the correlation coefficients of the targets, the targets' co-variance matrix, $\boldsymbol{\Sigma}$, is first calculated as shown in Equation 3.1:

$$\boldsymbol{\Sigma}_{ij} = cov(\boldsymbol{Y}_i, \boldsymbol{Y}_j) = \mathbf{E}\left[(\boldsymbol{Y}_i - \mu_i)(\boldsymbol{Y}_j - \mu_j)\right], \tag{3.1}$$

where $\mu_i = \mathbf{E}(\boldsymbol{Y}_i)$, and $\mathbf{E}(\boldsymbol{Y}_i)$ is the expected value of $\boldsymbol{Y}_i$, $\forall i, j \in \{1, \ldots, m\}$. This matrix

will show how the targets change together.

The correlation coefficients matrix, $\mathbf{P}$, is then calculated as shown in Equation 3.2:

$$\mathbf{P} = corrcoef(\boldsymbol{Y}) = \frac{\boldsymbol{\Sigma}_{ij}}{\sqrt{\boldsymbol{\Sigma}_{ii}\boldsymbol{\Sigma}_{jj}}}, \ \forall i, j \in \{1, \ldots, m\}, \tag{3.2}$$

which describes the linear relationship among the target variables. The coefficients are then sorted in decreasing order, creating the maximum correlation chain.

3.3 Experimental Environment

Although many interesting applications of multi-target regression exist, there are not many publicly available datasets to use. The datasets used in the experimental study were collected from the Mulan website [169], as well as the UCI Machine Learning Repository [8]. Information on the 24 datasets used is summarized in Table 3.2.

Experiments were performed over the RC, ST, MTS, MTSC, ERC, ERCC, and MORF algorithms, which have also been used in the experimental study conducted in [157]. These algorithms were chosen because they have shown considerable performance in training multi-target models. The have also made their framework readily available for reproducing their results. All three SVR algorithms are implemented within the general framework of Mulan's MTRegressor[1] [169], which was built on top of Weka[2] [88]. LIBSVM's Epsilon-SVR [47] implementation was used as the base SVR model. The parameters experimented with for the SVR regression task are the penalty parameter C, the Gaussian kernel parameter γ, and the error or tube parameter ϵ given by Equations (3.3a) to (3.3c), referred to as (3.3).

---

[1]http://mulan.sourceforge.net

[2]http://www.cs.waikato.ac.nz/ml/weka

$$C \in \{1, 10, 100\} \tag{3.3a}$$

$$\gamma \in \{1^{-9}, 1^{-7}, 1^{-5}, 1^{-3}, 1^{-1}, 1, 5, 10\} \tag{3.3b}$$

$$\epsilon \in \{0.01, 0.1, 0.2\} \tag{3.3c}$$

To ensure a controlled environment when conducting the performance comparisons, the experimental environment for running the competing algorithms was the same as what was done in [157]. This includes the following. The ST base-line model used was Bagging [34] of 100 regression trees [182]. The MTSC and ERCC methods are run using 10-fold cross-validation, and the ensemble size for the ERC and ERCC methods was set to 10. The

Table 3.2.: Multi-Target (MT) Regression datasets

Dataset	# Samples	# Attributes	# Targets
EDM	145	16	2
Enb	768	8	2
Jura	359	11	7
Osales	639	413	12
Scpf	1137	23	3
Slump	103	7	3
Solar Flare 1	323	10	3
Solar Flare 2	1,066	10	3
Water Quality	1,060	16	14
OES97	323	263	16
OES10	403	298	16
ATP1d	201	411	6
ATP7d	188	411	6
Andro	49	30	6
Wisconsin Cancer	198	34	2
Stock	950	10	3
California Housing	20,640	7	2
Puma8NH	8,192	8	3
Puma32H	8,192	32	6
Friedman	500	25	6
Polymer	41	10	4
M5SPEC	80	700	3
MP5SPEC	80	700	3
MP6SPEC	80	700	3

ensemble size of 100 trees was used for MORF, and the rest of its parameters were set as recommended by [116].

The performance metrics used to analyze our contributions' performances are shown in Equations 3.4 to 3.7. For unseen or test datasets of size $\mathcal{N}_{test}$, the performances are evaluated by taking the run time (seconds) each algorithm takes to build a classifier, as well as the following metrics, where the upwards arrow $\uparrow$ indicates maximizing the metric and the downwards arrow $\downarrow$ indicates minimizing the metric.

- The average correlation coefficient (aCC $\uparrow$):

$$\frac{1}{m} \sum_{j=1}^{m} \frac{\sum_{l=1}^{\mathcal{N}_{test}} (y_j^{(l)} - \bar{y}_j)(\hat{y}_j^{(l)} - \bar{\hat{y}}_j)}{\sqrt{\sum_{l=1}^{\mathcal{N}_{test}} (y_j^{(l)} - \bar{y}_j)^2 \sum_{l=1}^{\mathcal{N}_{test}} (\hat{y}_j^{(l)} - \bar{\hat{y}}_j)^2}} \tag{3.4}$$

- The mean squared error (MSE $\downarrow$):

$$\frac{1}{m} \sum_{j=1}^{m} \frac{1}{\mathcal{N}_{test}} \sum_{l=1}^{\mathcal{N}_{test}} (y_j^{(l)} - \hat{y}_j^{(l)})^2 \tag{3.5}$$

- The average root mean squared error (aRMSE $\downarrow$):

$$\frac{1}{m} \sum_{j=1}^{m} \sqrt{\frac{\sum_{l=1}^{\mathcal{N}_{test}} (y_j^{(l)} - \hat{y}_j^{(l)})^2}{\mathcal{N}_{test}}} \tag{3.6}$$

- The average relative root mean squared error (aRRMSE $\downarrow$):

$$\frac{1}{m} \sum_{j=1}^{m} \sqrt{\frac{\sum_{l=1}^{\mathcal{N}_{test}} (y_j^{(l)} - \hat{y}_j^{(l)})^2}{\sum_{l=1}^{\mathcal{N}_{test}} (y_j^{(l)} - \bar{y}_j)^2}} \tag{3.7}$$

The predicted output is represented by $\hat{\mathbf{y}}$, the average of the predicted output is $\bar{\hat{\mathbf{y}}}$, and the average of the true output target variable is $\bar{\mathbf{y}}$. The test dataset is the hold-out set during cross-validation. This ensures our model is evaluated on data that it has not been trained on, and thus unbiased towards the training datasets. It also contributes to the generalizability and robustness of the model.

3.4 Results & Statistical Analysis

Tables 3.3, 3.5, 3.7, 3.9, and 3.11 show the results of our algorithm implementations compared with those of *RC, MORF, ST, MTS, MTSC, ERC,* and *ERCC*. Each subsection discusses a single metric along with the statistical analysis of the results. The best metric value obtained on each dataset is typeset in bold. Non-parametric statistical tests are then used to validate the experiments results obtained. To determine whether significant differences exist among the performance and results of the algorithms, the Iman-Davenport non-parametric test is run to rank the algorithms over the datasets used, according to the Friedman test. The average ranks are presented in the last row of the results tables. The Bonferroni-Dunn post-hoc test [65] is then used to find these differences that occur between the algorithms. Below each result table, a diagram highlighting the critical distance (in gray) between each algorithm is shown. The Wilcoxon, Nemenyi, and Holm [180] tests were run for each of the result metrics to compute multiple pairwise comparisons among the algorithms used in the experimental study. Tables 3.4, 3.6, 3.8, 3.10, and 3.12 show the sum of ranks R^+ and R^- of the Wilcoxon rank-sum test, and the p-values for the 3 tests, which show the statistical confidence rather than using a fixed α value.

3.4.1 Average Correlation Coefficient

Table 3.3 shows that our proposed methods perform the best on 15 out of the 24 datasets. Specifically, the maximum correlation chain method, SVRCC, performs the best on 11, which is better than the total number of datasets the competing methods performed better at (9). The Iman-Davenport statistic, distributed according to the F-distribution with 9 and 207 degrees of freedom is 6.72, with a p-value of $1.9E^{-8}$ which is significantly less than 0.01, implying a statistical confidence larger than 99%. Therefore, we can conclude that there exist statistically significant differences between the aCC results of the algorithms.

Figure 3.4 shows the mean rank values of each algorithm along with the critical difference value, 2.4236, for $\alpha = 0.05$. The algorithms that are to the right of the critical

difference rectangle are the ones with significantly different results. Therefore, the 6 out of 10 algorithms beyond the critical difference perform significantly worse than our control algorithm, SVRCC. Table 3.4 provides complementary analysis of the results.

Table 3.3.: Average Correlation Coefficient (aCC) for MT regressors

Datasets	MORF	ST	MTS	MTSC	RC	ERC	ERCC	SVR	SVRRC	SVRCC
Slump	0.6965	0.7062	0.7163	0.6977	0.6956	0.6977	0.7023	0.7245	0.7339	**0.7457**
Polymer	0.7305	0.7336	0.7371	0.7228	0.7015	0.7029	0.7222	0.7634	0.7857	**0.7905**
Andro	**0.7349**	0.6454	0.6793	0.6581	0.6915	0.6806	0.6653	0.6880	0.6951	0.7056
EDM	**0.6722**	0.6352	0.6412	0.6354	0.6355	0.6379	0.6354	0.6484	0.6565	0.6567
Solar Flare 1	0.1083	0.1258	0.1034	0.1193	**0.1492**	0.1387	0.1292	0.1066	0.0857	0.1152
Jura	0.7854	0.7907	0.7880	0.7882	0.7877	0.7884	0.7897	0.7789	0.7921	**0.7983**
Enb	0.9828	0.9832	0.9822	0.9829	0.9813	0.9823	0.9837	0.9858	0.9867	**0.9868**
Solar Flare 2	0.2357	0.2295	0.2375	0.2343	0.2302	0.2351	**0.2432**	0.1470	0.1648	0.1656
Wisconsin Cancer	0.3362	0.3587	**0.3652**	0.3588	0.3628	0.3609	0.3590	0.3187	0.3208	0.3373
California Housing	0.7705	0.7720	0.7149	0.7451	0.7007	0.7844	**0.8065**	0.7847	0.7949	0.8007
Stock	0.9785	0.9747	0.9755	0.9752	0.9753	0.9757	0.9763	0.9825	**0.9829**	0.9822
SCPF	0.5827	0.5508	0.5503	0.5477	0.5569	0.5656	0.5515	0.5891	**0.5975**	0.5946
Puma8NH	0.5424	0.4828	0.4942	0.4205	0.4677	0.4656	0.4650	**0.6041**	0.5975	0.6038
Friedman	0.1507	0.1609	0.1548	0.1667	0.1558	0.1608	0.1632	0.1710	0.1748	**0.1752**
Puma32H	0.3085	0.2934	0.2890	0.2504	0.2754	0.2870	0.2797	0.3358	0.3351	**0.3385**
Water Quality	**0.4303**	0.4063	0.4019	0.4051	0.3992	0.4052	0.4147	0.3545	0.3828	0.3857
M5SPEC	0.8161	0.8346	0.8134	0.8228	0.8333	0.8340	0.8308	0.9451	0.9452	**0.9472**
MP5SPEC	0.8315	0.8536	0.8244	0.8535	0.8524	0.8526	0.8542	0.9560	0.9602	**0.9633**
MP6SPEC	0.8317	0.8531	0.8231	0.8531	0.8507	0.8515	0.8541	0.9444	0.9500	**0.9528**
ATP7d	0.8260	0.8408	0.8422	**0.8474**	0.8273	0.8351	0.8464	0.8305	0.8407	0.8400
OES97	0.7829	0.7995	0.7990	0.8001	0.7986	0.7990	0.7999	0.8116	0.8134	**0.8137**
Osales	0.7186	0.6912	0.7104	0.7076	0.6357	0.7136	**0.7193**	0.6511	0.6433	0.6677
ATP1d	0.8961	0.9066	0.9051	0.9075	0.9048	0.9081	0.9071	0.9092	**0.9130**	0.9100
OES10	0.8708	0.8808	0.8805	0.8806	0.8804	0.8804	0.8809	0.8911	0.8924	**0.8963**
Average	0.6508	0.6462	0.6429	0.6409	0.6396	0.6476	0.6492	0.6634	0.6685	**0.6739**
Ranks	6.4167	5.8958	6.6042	6.4792	7.5208	5.8958	4.8542	4.7917	3.7083	**2.8333**

Fig. 3.4.: Bonferroni-Dunn test for aCC

Table 3.4.: Wilcoxon, Nemenyi, and Holm tests for aCC

SVRCC vs.	Wilcoxon R^+	Wilcoxon R^-	Wilcoxon p-value	Nemenyi p-value	Holm p-value
MORF	224.0	76.0	$3.4E^{-2}$	$4.1E^{-5}$	$8.3E^{-3}$
ST	239.0	61.0	$9.6E^{-3}$	$4.6E^{-4}$	$1.3E^{-2}$
MTS	242.0	58.0	$7.2E^{-3}$	$1.6E^{-5}$	$6.3E^{-3}$
MTSC	238.0	62.0	$1.1E^{-2}$	$3.0E^{-5}$	$7.1E^{-3}$
RC	250.0	50.0	$3.1E^{-3}$	0.0000	$5.6E^{-3}$
ERC	229.0	71.0	$2.3E^{-2}$	$4.6E^{-4}$	$1.0E^{-2}$
ERCC	221.0	79.0	$4.3E^{-2}$	$2.1E^{-2}$	$1.7E^{-2}$
SVR	297.0	3.00	$6.0E^{-7}$	$2.5E^{-2}$	$2.5E^{-2}$
SVRRC	266.5	33.5	$4.0E^{-4}$	$3.2E^{-1}$	$5.0E^{-2}$

According to the Wilcoxon test, SVRCC is shown to have significantly better performance over all algorithms with p-value < 0.05. The Nemenyi and Holm tests show that SVRCC performs better than 6 out of the 9 algorithms with p-value $\leq 5.6E^{-3}$ and $\leq 1.7E^{-2}$, respectively. The exact confidence for algorithm SVRCC against all others is 0.95.

Table 3.5.: Mean Square Error (MSE) for MT regressors

Datasets	MORF	ST	MTS	MTSC	RC	ERC	ERCC	SVR	SVRRC	SVRCC
Slump	1.4388	1.4161	1.3667	1.4414	1.4602	1.4727	1.4183	1.2991	1.1726	**1.1614**
Polymer	1.6718	1.8120	1.5446	1.6726	1.8259	1.9999	1.6873	1.1874	1.1068	**1.0796**
Andro	1.4930	2.1467	1.4714	1.7525	2.2603	2.0812	1.8707	1.5406	1.2847	**1.2187**
EDM	**0.8342**	0.9373	0.9352	0.9418	0.9389	0.9326	0.9393	0.9092	0.8650	0.8817
Solar Flare 1	3.3458	3.1196	3.1193	3.0524	3.0357	3.0381	3.0594	**2.9912**	3.0176	3.0129
Jura	1.0973	1.0595	1.0732	1.0695	1.0744	1.0694	1.0632	1.1167	1.0435	**1.0315**
Enb	0.0381	0.0361	0.0407	0.0377	0.0452	0.0403	0.0343	0.0255	0.0216	**0.0214**
Solar Flare 2	2.9619	2.8532	**2.7732**	2.8282	2.8510	2.8273	2.8110	2.9518	2.9204	2.8713
Wisconsin Cancer	1.7666	1.7155	1.7156	1.7256	**1.7119**	1.7146	1.7195	1.8171	1.7915	1.7692
California Housing	0.8665	0.8221	0.9642	0.8673	1.0125	0.8952	0.7513	0.7477	0.6987	**0.6726**
Stock	0.0841	0.1039	0.0990	0.1008	0.0998	0.0987	0.0949	0.0578	0.0596	**0.0554**
SCPF	**2.2244**	2.3173	2.3661	2.3517	2.3923	2.3025	2.3295	2.2960	2.2510	2.3179
Puma8NH	1.9678	2.1133	2.0989	2.2024	2.1413	2.1473	2.1467	**1.8242**	1.8728	1.8299
Friedman	5.4573	5.3357	5.3478	5.3260	5.3482	5.3253	5.3210	5.3038	5.2942	**5.2812**
Puma32H	5.3419	**4.9499**	4.9627	5.0405	4.9905	4.9662	4.9805	5.2711	5.2749	5.1306
Water Quality	**11.3143**	11.5621	11.6276	11.5931	11.6495	11.6022	11.5004	12.2974	12.2042	12.0593
M5SPEC	1.0081	0.8754	1.0336	0.9421	0.8847	0.8824	0.8903	0.2578	0.2597	**0.2575**
MP5SPEC	1.1483	0.9817	1.1953	0.9970	0.9886	0.9880	0.9882	0.2261	**0.1979**	0.2136
MP6SPEC	1.1626	0.9928	1.1906	0.9992	1.0115	1.0045	0.9905	0.2926	**0.2903**	0.2954
ATP7d	1.7859	1.7348	**1.6435**	1.6460	1.8521	1.7888	1.6739	1.7820	1.7433	1.7098
OES97	4.6331	4.8340	4.8379	4.8082	4.8573	4.8591	4.8187	3.1440	3.0633	**3.0499**
Osales	7.3631	6.6850	**5.8848**	6.0850	7.8575	6.4746	5.9155	7.0727	7.3153	7.1374
ATP1d	1.0589	0.9056	0.9053	0.8982	0.9125	**0.8783**	0.9004	0.9091	0.8837	0.8922
OES10	3.6471	3.8931	3.8952	3.8909	3.9031	3.9063	3.8869	2.2623	2.1608	**2.1320**
Average	2.6546	2.6334	2.5872	2.5946	2.7127	2.6373	2.5747	2.3993	2.3664	**2.3368**
Ranks	6.5833	5.6667	6.0833	6.2500	7.8333	6.1250	5.1250	4.6667	3.6250	**3.0417**
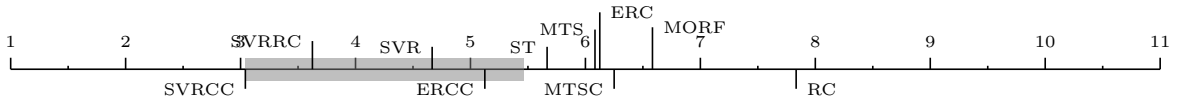
Fig. 3.5.: Bonferroni-Dunn test for MSE

Table 3.6.: Wilcoxon, Nemenyi, and Holm tests for MSE

SVRCC vs.	Wilcoxon R^+	Wilcoxon R^-	Wilcoxon p-value	Nemenyi p-value	Holm p-value
MORF	268.0	32.0	$3.2E^{-4}$	$5.1E^{-5}$	$6.3E^{-3}$
ST	241.0	59.0	$7.9E^{-3}$	$2.7E^{-3}$	$1.3E^{-2}$
MTS	224.0	76.0	$3.4E^{-2}$	$5.0E^{-4}$	$1.0E^{-2}$
MTSC	226.0	74.0	$2.9E^{-2}$	$2.4E^{-4}$	$7.1E^{-3}$
RC	263.0	37.0	$6.5E^{-4}$	0.0000	$5.6E^{-3}$
ERC	234.0	66.0	$1.5E^{-2}$	$4.2E^{-4}$	$8.3E^{-3}$
ERCC	224.0	76.0	$3.4E^{-2}$	$1.7E^{-2}$	$1.7E^{-2}$
SVR	262.0	38.0	$7.4E^{-4}$	$6.3E^{-2}$	$2.5E^{-2}$
SVRRC	245.0	55.0	$5.3E^{-3}$	$5.1E^{-1}$	$5.0E^{-2}$

3.4.2 Mean Square Error

Table 3.5 shows that our proposed methods perform the best on 15 out of the 24 datasets. In this case, SVRCC also performs the best on 11 versus the 9 that the competing methods performed better at. The Iman-Davenport statistic, distributed according to the F-distribution with 9 and 207 degrees of freedom is 6.57, with a p-value of $3.1E^{-8}$, implying statistically significant differences among the MSE results.

Figure 3.5 shows the mean rank values of each algorithm along with the critical difference value, 2.4236, for $\alpha = 0.05$. According to the critical difference bar, there are 6 out of 10 algorithms beyond that perform significantly worse than our control algorithm, SVRCC. According to the Wilcoxon test, shown in Table 3.6, SVRCC is shown to have significantly better performance over all algorithms with p-value < 0.05. The Nemenyi and Holm tests show that SVRCC performs significantly better than 6 out of the 9 algorithms with p-values $\leq 5.6E^{-3}$ and $\leq 1.7E^{-2}$ respectively, and has an exact confidence of 0.95 against all others.

3.4.3 Average Root Mean Square Error

Table 3.7 shows that our proposed methods perform the best on 18 out of the 24 datasets. In this case, SVRCC performs the best on 15 versus the 6 that the methods compared performed better at. The Iman-Davenport statistic is 7.6, with a p-value of $1.3E^{-9}$, implying statistically significant differences in the aRMSE results.

Figure 3.6 shows the mean rank values of each algorithm along with the critical difference value, 2.4236, for $\alpha = 0.05$. According to the critical difference bar, there are 7 out of 10 algorithms that perform significantly worse than our control algorithm, SVRCC.

According to the Wilcoxon test, shown in Table 3.8, SVRCC is shown to have significantly better performance over all algorithms with p-value < 0.01. The Nemenyi test shows that SVRCC performs significantly better than 7 out of the 9 algorithms with p-value $\leq 5.6E^{-3}$, while the stricter Holm test shows that it performs significantly better than 8 out of the 9 algorithms with p-value ≤ 0.05.

Table 3.7.: Average Root Mean Square Error (aRMSE) for MT regressors

Datasets	MORF	ST	MTS	MTSC	RC	ERC	ERCC	SVR	SVRRC	SVRCC
Slump	0.6711	0.6652	0.6456	0.6699	0.6787	0.6793	0.6649	0.5561	0.5345	**0.5337**
Polymer	0.5277	0.5409	0.5042	0.5336	0.5536	0.5803	0.5319	0.4403	0.4062	**0.4060**
Andro	0.4649	0.5420	0.4414	0.4871	0.5390	0.5317	0.5039	0.4326	0.4061	**0.3989**
EDM	0.6372	0.6715	0.6705	0.6729	0.6722	0.6704	0.6721	0.6449	0.6411	**0.6366**
Solar Flare 1	0.9777	0.9274	0.9271	0.9089	0.8921	0.9016	0.9121	0.8856	0.8844	**0.8801**
Jura	0.5800	0.5686	0.5720	0.5706	0.5726	0.5712	0.5693	0.5794	0.5687	**0.5622**
Enb	0.1212	0.1166	0.1237	0.1214	0.1272	0.1253	0.1140	0.0981	0.0914	**0.0903**
Solar Flare 2	0.8725	0.8420	**0.8127**	0.8305	0.8313	0.8300	0.8304	0.8418	0.8349	0.8345
Wisconsin Cancer	0.9290	0.9163	0.9158	0.9187	**0.9153**	0.9160	0.9173	0.9422	0.9362	0.9306
California Housing	0.6541	0.6366	0.6889	0.6530	0.7053	0.6632	0.6079	0.6038	0.5859	**0.5755**
Stock	0.1643	0.1830	0.1774	0.1790	0.1790	0.1777	0.1739	0.1357	0.1329	**0.1308**
SCPF	0.7113	0.7235	0.7342	0.7255	0.7285	0.7143	0.7227	0.7155	0.7081	**0.7048**
Puma8NH	0.7855	0.8139	0.8114	0.8307	0.8196	0.8202	0.8203	**0.7650**	0.7740	0.7671
Friedman	0.9382	0.9203	0.9219	0.9199	0.9219	0.9197	0.9193	0.9203	0.9195	**0.9183**
Puma32H	0.9395	**0.8700**	0.8713	0.8778	0.8739	0.8716	0.8727	0.9353	0.9356	0.9331
Water Quality	**0.8921**	0.9015	0.9041	0.9025	0.9051	0.9030	0.8990	0.9284	0.9293	0.9271
M5SPEC	0.5707	0.5324	0.5761	0.5515	0.5347	0.5339	0.5376	0.2745	0.2744	**0.2740**
MP5SPEC	0.5315	0.4914	0.5426	0.4947	0.4930	0.4928	0.4928	0.2337	**0.2176**	0.2177
MP6SPEC	0.5344	0.4939	0.5416	0.4943	0.4982	0.4967	0.4927	0.2627	**0.2460**	0.2497
ATP7d	0.5216	0.4956	**0.4752**	0.4765	0.5194	0.5024	0.4824	0.5141	0.5066	0.5018
OES97	0.4652	0.4634	0.4635	0.4622	0.4643	0.4644	0.4627	0.3794	0.3768	**0.3749**
Osales	0.7190	0.6912	**0.6496**	0.6615	0.7591	0.6772	0.6515	0.7212	0.7343	0.7121
ATP1d	0.4053	0.3608	0.3587	0.3591	0.3653	0.3562	0.3596	0.3693	0.3638	**0.3507**
OES10	0.3954	0.3896	0.3897	0.3892	0.3901	0.3903	0.3889	0.3085	0.3039	**0.3038**
Average	0.6254	0.6149	0.6133	0.6121	0.6225	0.6162	0.6083	0.5620	0.5547	**0.5506**
Ranks	7.3333	5.7708	5.8125	6.0625	7.6250	6.0208	4.8542	5.0625	3.9167	**2.5417**

Fig. 3.6.: Bonferroni-Dunn test for aRMSE

Table 3.8.: Wilcoxon, Nemenyi, and Holm tests for aRMSE

SVRCC vs.	Wilcoxon R^+	Wilcoxon R^-	Wilcoxon p-value	Nemenyi p-value	Holm p-value
MORF	286.0	14.0	$1.3E^{-5}$	0.0000	$6.3E^{-3}$
ST	259.0	41.0	$1.1E^{-3}$	$2.2E^{-4}$	$1.3E^{-2}$
MTS	247.0	53.0	$4.3E^{-3}$	$1.8E^{-5}$	$1.0E^{-2}$
MTSC	251.0	49.0	$2.8E^{-3}$	$5.6E^{-5}$	$7.1E^{-3}$
RC	270.0	30.0	$2.4E^{-4}$	0.0000	$5.6E^{-3}$
ERC	255.0	45.0	$1.8E^{-3}$	$6.9E^{-5}$	$8.3E^{-3}$
ERCC	246.0	54.0	$4.8E^{-3}$	$8.2E^{-3}$	$2.5E^{-2}$
SVR	296.0	4.00	$8.3E^{-7}$	$3.9E^{-3}$	$1.7E^{-2}$
SVRRC	284.0	16.0	$2.0E^{-5}$	$1.2E^{-1}$	$5.0E^{-2}$

Table 3.9.: Average Relative Root Mean Square Error (aRRMSE) for MT regressors

| Datasets	MORF	ST	MTS	MTSC	RC	ERC	ERCC	SVR	SVRRC	SVRCC
Slump	0.6939	0.6886	0.6690	0.6938	0.7019	0.7022	0.6886	0.5765	**0.5545**	0.5560
Polymer	0.6159	0.5971	0.5778	0.6493	0.6270	0.6544	0.6131	0.5573	0.5253	**0.5116**
Andro	0.5097	0.5979	0.5155	0.5633	0.5924	0.5885	0.5666	0.4856	0.4651	**0.4455**
EDM	0.7337	0.7442	0.7413	0.7446	0.7449	0.7452	0.7443	0.7058	0.7070	**0.6978**
Solar Flare 1	1.3046	1.1357	1.1168	1.0758	0.9951	1.0457	1.0887	0.9917	0.9455	**0.9320**
Jura	0.5969	0.5874	0.5906	0.5892	0.5910	0.5896	0.5880	0.5952	**0.5764**	0.5885
Enb	0.1210	0.1165	0.1231	0.1211	0.1268	0.1250	0.1139	0.0977	0.0910	**0.0899**
Solar Flare 2	1.4167	1.1503	**0.9483**	1.0840	1.0092	1.0522	1.0928	1.0385	1.0253	1.0298
Wisconsin Cancer	0.9413	0.9314	0.9308	0.9336	**0.9305**	0.9313	0.9323	0.9555	0.9483	0.9427
California Housing	0.6611	0.6447	0.6974	0.6630	0.7131	0.6690	0.6146	0.6130	0.5945	**0.5852**
Stock	0.1653	0.1844	0.1787	0.1803	0.1802	0.1789	0.1752	0.1364	**0.1337**	0.1388
SCPF	0.8273	0.8348	0.8436	0.8308	0.8263	0.8105	0.8290	0.8164	0.8037	**0.8013**
Puma8NH	0.7858	0.8142	0.8118	0.8311	0.8199	0.8205	0.8207	**0.7655**	0.7744	0.7676
Friedman	0.9394	0.9214	0.9231	0.9210	0.9231	0.9209	0.9204	0.9218	0.9208	**0.9196**
Puma32H	0.9406	**0.8713**	0.8727	0.8791	0.8752	0.8729	0.8740	0.9364	0.9367	0.9319
Water Quality	**0.8994**	0.9085	0.9109	0.9093	0.9121	0.9097	0.9057	0.9343	0.9310	0.9045
M5SPEC	0.5910	0.5523	0.5974	0.5671	0.5552	0.5542	0.5558	0.2951	0.2935	**0.2925**
MP5SPEC	0.5522	0.5120	0.5683	0.5133	0.5145	0.5143	0.5119	0.2484	**0.2323**	0.2358
MP6SPEC	0.5553	0.5152	0.5686	0.5119	0.5198	0.5187	0.5109	0.2850	0.2669	**0.2623**
ATP7d	0.5563	0.5308	**0.5141**	0.5142	0.5558	0.5397	0.5182	0.5455	0.5371	0.5342
OES97	0.5490	0.5230	0.5229	0.5217	0.5239	0.5237	0.5222	0.4641	**0.4618**	0.4635
Osales	0.7596	0.7471	**0.7086**	0.7268	0.8318	0.7258	0.7101	0.7924	0.7924	0.7811
ATP1d	0.4173	0.3732	0.3733	0.3712	0.3790	**0.3696**	0.3721	0.3773	0.3707	0.3775
OES10	0.4518	0.4174	0.4176	0.4171	0.4178	0.4180	0.4166	0.3570	0.3555	**0.3538**
Average	0.6910	0.6625	0.6551	0.6589	0.6611	0.6575	0.6536	0.6039	0.5935	**0.5893**
Ranks	7.5000	5.7708	5.9375	6.1667	7.4375	6.3750	4.9792	4.7708	3.2708	**2.7917**

Fig. 3.7.: Bonferroni-Dunn test for aRRMSE

Table 3.10.: Wilcoxon, Nemenyi, and Holm tests for aRRMSE

SVRCC vs.	Wilcoxon R^+	Wilcoxon R^-	Wilcoxon p-value	Nemenyi p-value	Holm p-value
MORF	290.0	10.0	$5.1E^{-6}$	0.0000	$5.6E^{-3}$
ST	261.0	39.0	$8.5E^{-4}$	$6.5E^{-4}$	$1.3E^{-2}$
MTS	239.0	61.0	$9.6E^{-3}$	$3.2E^{-3}$	$1.0E^{-2}$
MTSC	261.0	39.0	$8.5E^{-4}$	$1.1E^{-3}$	$8.3E^{-3}$
RC	275.0	25.0	$1.1E^{-4}$	0.0000	$6.3E^{-3}$
ERC	261.0	39.0	$8.5E^{-4}$	$4.1E^{-5}$	$7.1E^{-3}$
ERCC	254.0	46.0	$2.0E^{-3}$	$1.2E^{-2}$	$1.7E^{-2}$
SVR	291.0	9.00	$3.9E^{-6}$	$2.4E^{-2}$	$2.5E^{-2}$
SVRRC	222.5	77.5	$3.8E^{-2}$	$5.8E^{-1}$	$5.0E^{-2}$

3.4.4 Average Relative Root Mean Square Error

Table 3.9 shows that our proposed methods perform the best on 16 out of the 24 datasets. In this case, SVRCC performs the best on 11 versus the 6 that the competing methods

performed better at. The Iman-Davenport statistic is 8.54, with a p-value of $7.6E^{-11}$.

Figure 3.7 shows the mean rank values of each algorithm along with the critical difference value, 2.4236, for $\alpha = 0.05$. According to the critical difference bar, there are 6 out of 10 algorithms beyond that perform significantly worse than our control algorithm, SVRCC.

According to the Wilcoxon test, shown in Table 3.10, SVRCC is shown to have significantly better performance over all algorithms with p-value < 0.05, and 8 out of the 9 algorithms for p-value < 0.01. The Nemenyi test shows that SVRCC performs significantly better than 6 out of the 9 algorithms with p-value $\leq 5.6E^{-3}$, and the Holm test shows its performance is significantly better than 8 out of the 9 algorithms with p-value ≤ 0.05.

3.4.5 Run Time

Table 3.11 shows that our proposed methods perform faster on 16 out of the 24 datasets. In this case, SVR performs the best on 12 versus the 6 of the state-of-the-art methods. The Iman-Davenport statistic 64.41, with a p-value of 0.0 which implies a statistical confidence of 100%. Figure 3.8 shows the mean rank values of each algorithm along with the critical difference value, 2.4236, for $\alpha = 0.05$. According to the critical difference bar, there are 6 out of 10 algorithms beyond that perform significantly worse than our control algorithm, SVR. According to the Wilcoxon test, shown in Table 3.12, SVR is shown to have significantly better performance over all algorithms with p-value < 0.01. The Nemenyi and Holm tests show that SVRCC performs significantly better than 6 out of the 9 algorithms and 8 out of the 9 algorithms with p-value $\leq 5.6E^{-3}$ and p-value $\leq 1.6E^{-2}$, respectively.

3.4.6 Discussion

Results indicate that our proposed methods perform competitively against the current contemporary methods, specifically SVRCC which exploits relationships among the targets. Firstly, they show that using SVR as a base-line method for multi-target chaining causes a performance improvement in model prediction, compared to other ST base-line models,

Table 3.11.: Run Time (seconds) for MT regressors

Datasets	MORF	ST	MTS	MTSC	RC	ERC	ERCC	SVR	SVRRC	SVRCC
Slump	38.1	2.6	9.9	15.9	1.8	11.1	50.5	**0.6**	1.9	0.7
Polymer	7.6	2.7	9.1	15.5	1.9	14.9	80.5	**0.5**	2.6	**0.5**
Andro	25.7	4.4	15.0	34.2	3.4	33.2	197.9	**1.1**	6.2	**1.1**
EDM	24.8	2.8	9.4	18.1	2.1	5.8	19.0	**0.9**	1.0	**0.9**
Solar Flare 1	34.1	3.5	13.6	26.7	2.7	17.7	86.9	**2.3**	9.3	2.6
Jura	64.3	7.9	31.8	74.3	6.4	43.5	254.2	**4.7**	18.7	5.3
Enb	71.4	6.6	26.1	63.6	**5.4**	15.6	69.6	11.3	17.7	15.9
Solar Flare 2	55.4	7.4	30.7	68.0	**6.3**	42.9	241.5	9.4	53.5	15.6
Wisconsin Cancer	51.4	6.1	21.9	53.7	4.9	14.8	61.6	**2.0**	2.4	**2.0**
California Housing	93.0	9.7	34.8	75.9	**8.2**	21.3	102.0	15.8	25.2	23.6
Stock	93.7	11.7	46.8	96.7	**11.0**	75.4	427.3	18.5	90.5	26.3
SCPF	66.3	19.3	65.9	176.3	**15.0**	104.2	734.2	32.8	162.8	48.8
Puma8NH	130.4	29.7	106.7	288.6	**27.9**	201.6	1227.7	94.1	516.6	177.1
Friedman	79.5	27.0	81.2	258.3	25.0	273.7	2871.6	**12.3**	322.3	18.8
Puma32H	93.9	68.1	181.0	635.0	87.7	667.9	6087.0	**32.2**	1018.7	53.1
Water Quality	108.4	**93.1**	262.1	912.3	127.2	925.4	10993.3	110.2	2567.9	189.5
M5SPEC	89.8	68.9	166.3	604.6	73.7	262.3	3132.1	**39.2**	546.7	45.1
MP5SPEC	84.5	94.6	221.2	888.3	91.5	557.0	6864.1	**49.3**	1132.1	58.4
MP6SPEC	90.3	93.4	212.6	871.0	89.1	557.6	6761.3	**47.2**	1227.1	58.5
ATP7d	**70.5**	262.6	452.1	2319.8	242.1	1779.2	24373.8	80.0	1897.4	136.5
OES97	**83.4**	485.3	1146.6	4928.9	499.8	5315.0	58072.1	148.2	3759.1	342.6
Osales	**92.0**	1094.8	2340.7	8322.2	986.5	11361.2	122265.3	437.0	4830.1	843.6
ATP1d	**70.7**	272.9	476.5	2568.9	261.9	2138.9	26768.9	95.0	2127.8	174.4
OES10	**90.0**	738.9	1633.6	6682.9	688.5	7150.8	83533.1	229.1	5419.4	577.1
Average	71.2	142.2	316.5	1250.0	136.2	1316.3	14803.2	**61.4**	1073.2	117.4
Ranks	5.5	3.71	6.0	8.29	3.0	7.08	9.92	**1.88**	6.71	2.92

Fig. 3.8.: Bonferroni-Dunn test for Run Time

Table 3.12.: Wilcoxon, Nemenyi, and Holm tests for Run Time

SVRCC vs.	Wilcoxon R^+	Wilcoxon R^-	Wilcoxon p-value	Nemenyi p-value	Holm p-value
SVRCC	295.0	5.00	$1.2E^{-6}$	$2.3E^{-1}$	$5.0E^{-2}$
MORF	225.0	75.0	$3.2E^{-2}$	$3.4E^{-5}$	$1.3E^{-2}$
ST	221.5	78.5	$4.1E^{-2}$	$3.6E^{-2}$	$1.7E^{-2}$
MTS	300.0	0.00	$1.2E^{-7}$	$2.0E^{-6}$	$1.0E^{-2}$
MTSC	300.0	0.00	$1.2E^{-7}$	0.0000	$6.3E^{-3}$
RC	229.0	71.0	$2.3E^{-2}$	$2.0E^{-1}$	$2.5E^{-2}$
ERC	300.0	0.00	$1.2E^{-7}$	0.0000	$7.1E^{-3}$
ERCC	300.0	0.00	$1.2E^{-7}$	0.0000	$5.6E^{-3}$
SVRRC	300.0	0.00	$1.2E^{-7}$	0.0000	$8.3E^{-3}$

as well as most MT methods. This demonstrates the advantages of using the SVR method as a base-line for multi-target learning, thus increasing the performance of the ensemble of regressor chains, SVRRC, compared to ERCC. More importantly, the results highlight the

major advantage of capturing and exploiting the targets' relationships during model training. Using an ensemble of randomly generated chains does not ensure the targets' correlations are fully captured; however, using a maximum correlation chain improves the performance in terms of quality metrics as well as run time. The run time of SVR was shown to be the fastest, due to the fact that its complexity is mostly dependent on the number of targets. However, this method does not consider any of the correlations that might exist among the target variables, but SVRCC does take them into account and does not have a significant impact on run time. The most noteworthy finding that highlights advantage of using the base-line SVR and the maximum correlation method, SVRCC, rather than random chaining as done in ERCC, are the run time results and their analysis. ERCC had the worst run time across all datasets, whereas our proposals, SVR and SVRCC, performed the fastest. This emphasizes the advantage of using a single chain rather an ensemble of random chains, especially when the single chain is ordered in the direction of the targets maximum correlation.

3.5 Conclusions

This contribution proposed three novel methods for solving multi-target regression problems. The first method takes a problem transformation approach, which generates m ST models, each trained independently. This base-line approach was shown to perform the best in terms of run time, but its drawback is that it does not take the possible correlations between the target variables into account during training. The second implements SVR as an ensemble model of randomly generated chains, inspired by the classification method ERCC. This was done to investigate the effects of exploiting correlations among the target variables during model training. Due to the random nature of this method, capturing target correlations is not guaranteed. The third proposal, SVRCC, generates a single chain that is ordered in the direction of the targets' maximum correlation, ensuring the correlations among targets are taken into account within the learning process.

The experimental study compared the proposed methods' performances to 7 popular, contemporary methods on 24 MT regression datasets. Firstly, the results show the superior performance of using the SVR method as a base-line model, rather than regression trees as used in MORF. The results for SVRRC show an increase in performance when random chaining is used to develop an ensemble model. This indicates the importance of the relationship among the target variables during training. Finally, the results show the superiority of using the SVRCC method, which was ranked the best in all quality metrics and second best in terms of run time. SVRCC performed better than the single-target SVR model and the randomly chained ensemble model SVRRC, showing that the targets' maximum correlation does positively contribute toward model training. The statistical analysis supports and shows the significance of the results obtained by our experiments. They demonstrated that statistically significant differences exist between the proposed algorithms against the methods compared. SVRCCs competitive performance, as well as speed, shows that it is a powerful learning algorithm for multi-target problems. The research outcomes of this chapter have been published in [130].

CHAPTER 4

MULTI-INSTANCE SVM USING BAG-REPRESENTATIVES

This chapter proposes a novel *support vector machine* (SVM) formulation under the *multiple-instance learning* (MIL) paradigm. It also presents a novel algorithm and bag representative selector that train the SVM using bag-level information, named *Multi-Instance Representative Support Vector Machine* (MIRSVM). The contribution is able to identify instances that highly impact classification, i.e. the *bag-representatives*, for both positive and negative bags, while finding the optimal class separation hyperplane. Unlike other multi-instance SVM methods, this approach eliminates possible class imbalance issues by allowing both positive and negative bags to have at most one representative, which constitute as the most contributing instances to the model. The experimental study evaluates and compares the performance of this proposal against 11 popular and widely used multi-instance methods over 15 datasets, and the results are validated through non-parametric statistical analysis. The results indicate that bag-based learners outperform the instance-based and wrapper methods, and emphasize this proposal's overall superior performance against other multi-instance SVM models.

4.1 Multi-Instance Classification Background

This section defines the notation that will be used throughout this chapter and reviews related works pertaining to multiple-instance learning, specifically the concepts of instance-based and bag-based learners are discussed and compared, along with algorithms within those paradigms.

Table 4.1.: Summary of Multiple-Instance Learning Notation

Definition	Notation
Number of Bags	n
Number of Instances	m
Number of Input Attributes	d
Set of Bags	$\mathcal{B} = \{\mathcal{B}_1, \ldots, \mathcal{B}_n\}$
Bag Index Set	$I \in \mathbb{Z}_+^n$
Input Space	$\boldsymbol{X} \in \mathbb{R}^{m \times d}$
Bag Labels	$\boldsymbol{Y} \in \{-1, 1\}^n$
Input Instance i from Bag I	$\boldsymbol{x}_i \in \boldsymbol{x}_i = (x_{i1}, \ldots, x_{id}), \forall i \in \{1, \ldots, n\}, i \in I$
Unknown Individual Instance Label i	$y_i \in \{-1, 1\}$
Bag I	$\mathcal{B}_I = \{\boldsymbol{x}_i \,|\, \forall i \in I\}$
Full Multi-Instance Training Dataset	$\mathcal{D} = \{(\mathcal{B}_1, Y_1), \ldots, (\mathcal{B}_n, Y_n)\}$

4.1.1 Notation

Let $\mathcal{D}$ be a training dataset of n bags. Let $\boldsymbol{Y} \in \mathcal{D}$ be a vector of n labels corresponding to each bag, having a domain of $\boldsymbol{Y} \in \{-1, 1\}^n$. Let $\boldsymbol{X} \in \mathcal{D}$ be a matrix consisting of d input variables and m instances, $\boldsymbol{x}_i \in \boldsymbol{X}$, $\forall i \in \{1, \ldots, m\}$, having a domain of $\boldsymbol{X} \in \mathbb{R}^{m \times d}$. Let $\mathcal{B}$ be the set of bags which contain $|\mathcal{B}_I|$ number of instances, sometimes of different size and usually non-overlapping, such that $\mathcal{B}_I = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{|\mathcal{B}_I|}\}$ for index set $I \in \{1, \ldots, n\}$.

4.1.2 Multi-Instance Classification Methods

The difference between MIL and traditional learning lies in the nature of the data. In the traditional binary classification setting, the goal is to learn a model that maps input samples to labels, $f : \mathbb{R}^{m \times d} \to Y^m \in \{-1, +1\}$. In the multi-instance setting, samples are called *bags* and each bag contains one or more input instances and is assigned a single bag-level label. Input instances, $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m\}$, are grouped into bags with unique identifiers, $\mathcal{B} = \{\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_n \,|\, \mathcal{B}_I = \{\boldsymbol{x}_i \,|\, \forall i \in I\}, \forall I \in \{1, \ldots, n\}\}$ and assigned a label, Y_I. The individual instance labels within a bag are unknown. Traditionally, MIL has focused on binary classification problems, however, there are cases where the number of classes can be larger. The scope of this chapter will focus on binary multi-instance classification.

Using the training dataset $\mathcal{D} = \{(\mathcal{B}_1, Y_1), \ldots, (\mathcal{B}_n, Y_n)\}$, the goal is to train a classifier that predicts the label of an unseen bag, $f(\mathcal{B}_{n+1}) \to Y_{n+1}$ [6]. In order to build a classifier without any knowledge of the individual training instance labels, Dietterich et. al. [58] proposed the *standard MI* (SMI) hypothesis, shown in Equation (4.1), which states that a bag is labeled positive if and only if at least one of the instances in the bag is positive, and is labeled negative otherwise.

$$Y_I = \begin{cases} +1 & \text{if } \exists\, y_i = +1, \forall i \in I \\ -1 & \text{otherwise.} \end{cases} \tag{4.1}$$

This implies that individual instance labels y_i exist, but are not known (for positive bags) during training. Equation (4.1) can also be rewritten as Equation (4.2) for simplicity:

$$Y_I = \text{argmax}_{\forall i \in I}\ y_i. \tag{4.2}$$

In addition to the SMI assumption, alternative MI assumptions have been proposed to date [70]. A recent review [6] describing the taxonomy of multi-instance classification presents various methods and algorithms used in literature, which are categorized based on their approach to handling the MI input space. Instance-based classifiers that fall under the *instance-space paradigm*, aim to separate instances in positive bags from those in negative ones. Bag-level classifiers (*bag-space paradigm*) treat each bag as a whole entity, implicitly extracting information from each bag in order to accurately predict their labels. Methods that fall under the *embedded-space paradigm* map the input bags to a single feature vector that explicitly encapsulates all the relevant information contained within each bag.

Instance-based methods that follow the SMI assumption attempt to identify desirable instance properties that make a bag positive. A simple and natural way of addressing this type of learning problem is to assume that each instance in a bag has the same label as the bag itself. After that, a single-instance classifier can be trained on the transformed dataset, and finally, the SMI assumption can be applied over the predicted instance labels of unseen

bags [92]. Methods that employ this type of approach are called *Wrapper* methods; they act as an interface between the instance and bag levels. One algorithm that employs this approach is called Simple-MI [62]. Simple-MI represents each bag with the mean vector of the instances within it. Another example is MIWrapper [71], which introduces weights to treat instances from different bags differently. The major disadvantage of wrapper techniques is that they assume the distribution the instances in positive bags is positive, when it may not be, thus imposing noise over the positive class.

One traditional instance-based method that takes a different approach is the Axis-Parallel Rectangle (APR) [58], which trains a model that assigns a positive label to an instance if it belongs to an axis-parallel rectangle in feature space, and assigns a negative label otherwise. The APR method is optimized by maximizing the number of positive bags in the training set containing at least one instance in the APR, while concurrently maximizing the number of negative bags that do not contain any instance in the APR.

Another similar and popular approach that falls under maximum likelihood-based methods, is the Diverse Density (DD) [126] framework. The DD metric is maximized for instances in feature space that are near at least one instance in a positive bag and far from all instances in negative bags. In the Expectation-Maximization Diverse Density (EM-DD) algorithm, Zhang and Goldman [193] propose a similar framework that iteratively maximizes the DD measure. Auer and Ortner [9] present a boosting approach that uses balls centered around positive bags to solve the MI problem called Multi-Instance Optimal Ball (MIOptimalBall). This approach is similar to that of APR and DD, except that Auer and Ortner [9] propose computing optimal balls per positive bags. A major challenge affecting these methods is that the distributions of the positive and negative bags affect their performance. Methods based on the DD metric [44, 49, 50] assume the positive instances form a cluster, which may not be the case. Alternatively, Fu et al.[75] models the distribution of negative bags with Gaussian kernels, which can prove difficult when the quantity of data is limited.

As mentioned previously in brief, the classical method of boosting [72, 147] has been adapted to a multi-instance instance-based algorithm. For example, Viola et al. [191] proposed an adaptation of boosting to the multi-instance paradigm based on the standard MI assumption, named MILBoost. Later, inspired by the MILBoost [191] and Online-ADABoost [134] algorithms, Babenko et al. [10] developed a novel online MI boosting method which lead to a robust and stable model in the area of object tracking. All the works mentioned, including many other such as [140, 183], have positively contributed and inspired many approaches in the area of visual detection and tracking.

An extension of traditional single-instance k-nearest neighbors method (k-NN) was proposed by Wang and Zucker [176] to be applied to the bag-level, named CitationKNN. This method uses a distance function between bags in order to determine bag similarities. Not only are the set of closest bags to a single bag considered, but also how many bags is the single bag closest to. A voting scheme is then used to determine the bag class labels. Note, any bag-based distance function can be used by CitationKNN.

Rule based methods have also been adapted to the multi-instance learning environment. Zafra et al. [188] proposed a novel multi-objective multi-instance genetic programming algorithm (MOG3P-MI) for rule-based systems, which optimizes two objectives: sensitivity and specificity. This evolutionary algorithm was then implemented in a distributed GPU environment in [42] to further enhance its performance over large-dimensional MIL problems.

Blockeel et al. [22] introduced the Multi-Instance Tree Inducer (MITI), based on the standard MI assumption, which uses decision trees as a heuristic to solve the MI problem. This approach aims to identify whether an instance within a bag is truly positive and eliminate false positives within the same bag. The disadvantage of this approach stems from removing instances considered as false positives from partially grown trees without updating the existing tree structure. Bjerring and Frank [19] then enhanced this approach by creating the method Multi-Instance Rule Induction (MIRI). The algorithm aims to eliminate any possibility of a suboptimal split because the tree is discarded and regrown.

The MI adaptation of the SVM presents two contexts for solving the problem: the instance-level and the bag-level. The first tries to identify instances, either all within a bag or just key instances, that help find the optimal separating hyperplane between positive and negative bags. The latter uses kernels defined over whole bags to optimize the margin [63].

Andrews et. al. [7] proposed a mixed-integer quadratic program that solves the MI problem at an instance-level, using a support vector machine, named MISVM, that can be solved heuristically. Rather than maximizing the margin of separability between instances of different classes, this instance-based method maximizes the margin between bags. It tries to identify the key instance from each positive bag that makes the bag positive by assuming it has the highest margin value. Instances from positive bags are selected as bag-representatives, and the algorithm iteratively creates a classifier that separates those representatives from all instances from the negative bags. Using bag-representatives from one class and all instances from the other is an example of an approach that combines rules from the SMI assumption and the collective assumption. A disadvantage of this approach stems from the assumption that all instances within positive bags are also positive, which is an implicit step in the initialization of MISVM. Andrews et. al. [7] also proposed a second mixed-integer instance-level approach, named mi-SVM, which does not discard the negative instances of the positive bags. It rather tries to identify the instances within positive bags that are negative and utilize them in the construction of the negative margin. The main disadvantage of these approaches is that they create an imbalanced class problem that favors the negative class, resulting in a biased classifier.

Tomar and Agarwal [165] proposed a bag-level multi-instance SVM based on Twin Support Vector Machine, named MIL-TWSVM. Rather than constructing a single hyperplane, as done by the traditional SVM, the Twin SVM constructs two nonparallel hyperplanes. In this implementation, each bag is transformed into a single vector of dissimilarities to other bags, using various distance metrics.

Asymmetric SVM (ASVM), presented by Yang et al. [91], was designed to use an asymmetric loss function under the SMI assumption. This approach was based on the idea that the cost of misclassification is different for positive and negative bags. For example, a false negative instance in a positive bag would not necessarily introduce an error on the bag label, assuming there are many positive instances within the bag. However, a false positive instance in a negative bag would definitely lead to an error. Using rules such as these, ASVM aims to minimize false positives, while ensuring all negative instances are on the negative side of the hyperplane.

The approach presented by Cheung et al. [51] presented a loss function that takes the cost associated with bag labels and the cost (under the SMI assumption) between prediction of beach bag and its instances into account. They also presented an SVM regularization scheme as well which, rather than using a heuristic method, used concave-convex optimization, ensuring local-optimum convergence.

An example of using the approach of a bag-level kernel would coincidentally be one of the first bag-level approaches to the multi-instance SVM problem, proposed by Gärtner et. al. [84]. A bag-level kernel determines the similarity between two bags in a higher dimensional space. Blaschko et. al. [20] proposed conformal kernels which manipulate each attribute's dimension based on its importance, without affecting the angles between vectors in the transformed space. These type of bag level kernels transform the bags into a single-instance representation which enables standard SVMs to be directly applied to multi-instance data.

Unlike the bag-based methods mentioned previously, which have a the SMI assumption embedded in their design, mapping-based algorithms do not assume a specific relationship exists between the labels of each bag and its instances. Rather, the relationship is learned from the data. An example of such methods is the Two-Level Classifier (TLC) [178]. Another example, which includes the use of kernels in the multi-instance bag-space setting, would be approach taken by Zhou et al. [196]. The basic idea behind their method is to treat instances in an non-i.i.d. manner, thus exploiting relationships among instances using a graph kernel.

For most of the methods described above, implicit or explicit assumptions have been made about the distribution of the data. Selecting a method that is robust for a problem such as MIL can be difficult when little is known about the nature of the data, especially considering the unknown distribution of the instances within bags [6]. The proposed method, MIRSVM, is a general method that uses support vector machines to design a MIL model without making prior assumptions about the data. Classifiers of this type are known to provide better generalization capabilities and performance, as well as sparser models.

4.2 MIRSVM: A Novel SVM for Multi-Instance Classification

MIRSVM is based on the idea of selecting representative instances from both positive and negative bags which are used to find an unbiased, optimal separating hyperplane. A representative is iteratively chosen from each bag, and a new hyperplane is formed according to the representatives until they converge. Based on the SMI hypothesis, only one instance in a bag is required to be positive for the bag to adopt a positive label. Due to the unknown distribution of instances within positive bags, MIRSVM is designed to give preference to negative bags during training, because their distribution is known, i.e. all instances are guaranteed to be negative. This is evident during the representative selection process, by taking the index of the maximum output value within each bag based on the current hyperplane using the following rule, $s_I = \text{argmax}_{i \in I}(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b)$, $\forall I \in \{1, \ldots, n\}$. In other words, the most positive instance is chosen from each positive bag and the least negative instance is chosen from each negative bag (instances with the largest output value based on the current hyperplane), pushing the decision boundary towards the positive bags.

Equation (4.3) presents the primal MIRSVM optimization problem:

$$\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_I \xi_I, \tag{4.3}$$

$$\text{s.t. } Y_I(\langle \boldsymbol{w}, \boldsymbol{x}_{s_I} \rangle + b) \geq 1 - \xi_I, \ \forall I \in \{1, \ldots, n\} \tag{4.3a}$$

$$\xi_I \geq 0, \ \forall I \in \{1, \ldots, n\}, \tag{4.3b}$$

64

where $\boldsymbol{S}_I$ is the set of the bag representatives' indices and $\boldsymbol{x}_{s_I}$ is the instance representative of bag $\mathcal{B}_I$. Note the variables in MIRSVMs formulation are the similar to those of the traditional SVM, except they are now representing each bag as an instance. Solving the optimization problem given in Equation (4.3) using a quadratic programming solver is a computationally expensive task due to the number of constraints, which scales by the number of bags n, as well as the calculation of the inner product between two d-dimensional vectors in constraint (4.3a). The proposed solution for these problems was deriving and solving the dual of the optimization problem given by Equation (4.3).

The dual can be formed by taking the Lagrangian of (4.3), given by Equation (4.4), where $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are the non-negative Lagrange multipliers.

$$\mathcal{L}\left(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}\right) = \frac{1}{2} \sum_{j=1}^{d} w_j^2 + C \sum_I \xi_I - \sum_I \beta_I \xi_I - \sum_I \alpha_I \left(Y_I \left(\sum_{j=1}^{d} w_j x_{s_I j} + b \right) - 1 + \xi_I \right) \quad (4.4)$$

At optimality [33], $\bigtriangledown_{w,b,\xi} \mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = 0$ and the following conditions are met:

$$\frac{\partial \mathcal{L}}{\partial w_j} : w_j = \sum_I \alpha_I Y_I x_{s_I j}, \ \forall j \in \{1, \ldots, d\} \quad (4.5)$$

$$\frac{\partial \mathcal{L}}{\partial b} : \sum_I \alpha_I Y_I = 0, \quad (4.6)$$

$$\frac{\partial \mathcal{L}}{\partial \xi_I} : \alpha_I + \beta_I = C, \ \forall I \in \{1, \ldots, n\} \quad (4.7)$$

and the KKT complementary conditions below,

$$\alpha_I(Y_I(\langle \boldsymbol{w}, \boldsymbol{x}_{s_I} \rangle + b) - 1 + \xi_I) = 0, \ \forall I \in \{1, \ldots, n\} \quad (4.8)$$

$$\beta_I \xi_I = (C - \alpha_I)\xi_I = 0, \ \forall I \in \{1, \ldots, n\}. \quad (4.9)$$

At the optimal solution and due to the KKT conditions 4.8 and 4.9, the dual Lagrangian becomes in terms of $\boldsymbol{\alpha}$, and the dual MIRSVM formulation becomes the following:

$$\max_{\boldsymbol{\alpha}} \sum_I \alpha_I - \frac{1}{2} \sum_I \sum_{K \in I} \sum_{j=1}^d \alpha_I \alpha_K Y_I Y_K x_{s_I j} x_{s_K j} \tag{4.10}$$

$$\text{s.t.} \sum_I \alpha_I Y_I = 0 \tag{4.10a}$$

$$0 \leq \alpha_I \leq C, \forall I \in \{1, \dots, n\}, \tag{4.10b}$$

where s_I is computed for each bag, as shown in Equation (4.11):

$$s_I = \operatorname{argmax}_{i \in I} (\sum_{K \in I} \sum_{j=1}^d \alpha_K Y_K x_{s_K j} x_{ij} + b), \forall I \in \{1, \dots, n\}. \tag{4.11}$$

Equations (4.8) and (4.9) imply three possible solutions for α_I values:

1. If $\alpha_I = 0$ and $\xi_I = 0$, then the instance is correctly classified and outside the margin.

2. If $0 < \alpha_I < C$, then, from the complementary conditions, $Y_I(\langle \boldsymbol{w}, \boldsymbol{x}_{s_I} \rangle + b) - 1 + \xi_I = 0$ and $\xi_I = 0$. Thus $Y_I(\langle \boldsymbol{w}, \boldsymbol{x}_{s_I} \rangle + b) = 1$, and the instance is a support vector. Support vectors with $0 < \alpha_I < C$ are called *unbounded* support vectors and lie on the margins.

3. If $\alpha_I = C$, then there is no restriction for $\xi_I > 0$. This also indicates that the instance is a support vector, but one that is *bounded*. If $0 \leq \xi_I < 1$, then the instance is correctly classified, and is misclassified if $\xi \geq 1$.

The dual is then kernelized by replacing the inner product of samples in feature space with their corresponding kernel value, $\mathcal{K}(\boldsymbol{x}_{s_I}, \boldsymbol{x}_{s_K})$. The dual function is now written as:

$$\max_{\boldsymbol{\alpha}} \sum_I \alpha_I - \frac{1}{2} \sum_I \sum_{K \in I} \alpha_I \alpha_K Y_I Y_K \mathcal{K}(\boldsymbol{x}_{s_I}, \boldsymbol{x}_{s_K}) \tag{4.12}$$

$$\text{s.t.} \sum_I \alpha_I Y_I = 0 \tag{4.12a}$$

$$0 \leq \alpha_I \leq C, \forall I \in \{1, \dots, n\}. \tag{4.12b}$$

One of the biggest advantages of the dual SVM formulation is the sparseness of the resulting model. This is because support vectors, instances that have their corresponding $\alpha_I \neq 0$, are only considered when forming the decision boundary. MIRSVM uses a Gaussian RBF kernel, given by Equation (4.13), where σ is the Gaussian shape parameter.

$$\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||^2}{2\sigma^2}} \tag{4.13}$$

To evaluate the output vector, $\boldsymbol{o}_I$, of bag I using the kernel, the following equation is used, where $\mathcal{B}_I$ are the instances of bag I, $\boldsymbol{X}_S$ are the optimal bag representatives, and $\boldsymbol{Y}_S$ are the representative bag labels.

$$\boldsymbol{o}_I = \mathcal{K}(\mathcal{B}_I, \boldsymbol{X}_S) * (\boldsymbol{\alpha} \cdot \boldsymbol{Y}_S) + b \tag{4.14}$$

The bias term b is calculated as shown in Equation (4.15), where $\boldsymbol{sv}$ is the vector of support vector indices and n_{sv} is the number of support vectors [96].

$$b = \frac{1}{n_{\boldsymbol{sv}}} \sum_{\boldsymbol{sv}} Y_{\boldsymbol{sv}} - \mathcal{K}(\boldsymbol{X}_{\boldsymbol{sv}}, \boldsymbol{X}_{\boldsymbol{sv}}) * (\boldsymbol{\alpha}_{\boldsymbol{sv}} \cdot Y_{\boldsymbol{sv}}) \tag{4.15}$$

Algorithm 4.1 shows the procedure for training the multi-instance representative SVM classifier and obtaining the optimal representatives from each bag. During training, the representatives, $\boldsymbol{S}$, are first initialized by randomly selecting an instance from each bag. A hyperplane is then obtained using the representative instances, and new optimal representatives are found with respect to the current hyperplane, by using the rule given in Equation (4.11). At each step, the previous values in $\boldsymbol{S}$ are stored in $\boldsymbol{S}_{old}$. The training procedure ends when the bag representatives stop changing from one iteration to the next ($\boldsymbol{S} = \boldsymbol{S}_{old}$). Examples of the convergence of bag-representatives are shown in Figure 4.2. During the testing procedure, each bag produces an output vector based on the hyperplane found in the training procedure. The bag label is then assigned by taking the sign of the output vector's maximum value, following the SMI assumption.

This formulation is designed to utilize and select representatives from positive and negative bags, unlike MISVM, which only optimizes over representatives from positive bags, while flattening the negative bag instances. MISVM allows multiple representatives to be chosen from negative bags and limits positive bag-representatives to be one, while MIRSVM allows for balanced bag-representative selection, where each bag is allowed one. MISVM also uses a wrapper method to initialize the positive bag-representatives by taking the mean vector of the instances within each positive bag. This is an implicit assumption that the

Fig. 4.1.: A summary of the steps performed by MIRSVM. The representatives are first randomly initialized and continuously updated according to the current hyperplane. Upon completion, the model is returned along with the optimal bag-representatives.

Algorithm 4.1 Multi-Instance Representative SVM (MIRSVM)

Input: Training dataset $\mathcal{D}$, SVM Parameters C and σ
Output: SVM model parameters $\boldsymbol{\alpha}$ and b, Bag Representative IDs $\boldsymbol{S}$
1: **for** $I \in \{1, \ldots, n\}$ **do**
2: $\quad \boldsymbol{S}_I \leftarrow \operatorname{rand}(|\mathcal{B}_I|, 1, 1)$ $\qquad\qquad\qquad\qquad$ ▷ Assign each bag a random instance
3: **end for**
4: **while** $\boldsymbol{S} \neq \boldsymbol{S}_{old}$ **do**
5: $\quad \boldsymbol{S}_{old} \leftarrow \boldsymbol{S}$
6: $\quad \boldsymbol{X}_S \leftarrow \boldsymbol{X}(\boldsymbol{S}), \boldsymbol{Y}_S \leftarrow \boldsymbol{Y}(\boldsymbol{S})$ $\qquad\qquad$ ▷ Initialize the representative dataset
7: $\quad \boldsymbol{G} \leftarrow (\boldsymbol{Y}_S \times \boldsymbol{Y}_S) \cdot \mathcal{K}(\boldsymbol{X}_S, \boldsymbol{X}_S, \sigma)$ $\qquad\qquad\qquad$ ▷ Build Gram matrix
8: $\quad \boldsymbol{\alpha} \leftarrow \operatorname{quadprog}(\boldsymbol{G}, -\mathbf{1}^n, \boldsymbol{Y}_S, \mathbf{0}^n, \mathbf{0}^n, \boldsymbol{C}^n)$ $\qquad\qquad$ ▷ Solve QP Problem
9: $\quad \boldsymbol{sv} \leftarrow \operatorname{find}(0 < \boldsymbol{\alpha} \leq C)$ $\qquad\qquad$ ▷ Get the support vector indices
10: $\quad n_{\boldsymbol{sv}} \leftarrow \operatorname{count}(0 < \boldsymbol{\alpha} \leq C)$ $\qquad\qquad$ ▷ Get the number of support vectors
11: $\quad b \leftarrow \frac{1}{n_{\boldsymbol{sv}}} \sum_{i=1}^{n_{sv}} (\boldsymbol{Y}_{\boldsymbol{sv}} - \boldsymbol{G}_{\boldsymbol{sv}} * (\boldsymbol{\alpha}_{\boldsymbol{sv}} \cdot \boldsymbol{Y}_{\boldsymbol{sv}}))$ $\qquad\qquad$ ▷ Calculate the bias term
12: $\quad$ **for** $I \in \{1, \ldots, n\}$ **do**
13: $\quad\quad \boldsymbol{G}_I \leftarrow (Y_I \times \boldsymbol{Y}_S) \cdot \mathcal{K}(\mathcal{B}_I, \boldsymbol{X}_S, \sigma)$
14: $\quad\quad \boldsymbol{S}_I \leftarrow \operatorname{argmax}_{i \in I} (\boldsymbol{G}_I * \boldsymbol{\alpha} + b)$ $\qquad\qquad$ ▷ Select optimal bag-representatives
15: $\quad$ **end for**
16: **end while**

Fig. 4.2.: Bag representative convergence plots on 9 datasets. The blue line shows the number of bag representatives that are equal from one iteration to the next. The red dashed line represents the total number of bags.

instances within the positive bags are all positive, whereas MIRSVM's initialization procedure selects an instance from all bags at random, ensuring no noise is added by any wrapper techniques during initialization and no assumptions are made about the instances. Due to the constraints on the representatives, MIRSVM produces sparser models while MISVM has the freedom to select as many negative support vectors as it needs and restricts the support vectors chosen from positive bags to be one. Figure 4.3 shows the decision boundaries produced by MIRSVM and MISVM to highlight the differences in their solutions. As Figure 4.3 shows, MISVM produces a larger number of support vectors from the negative bags, which greatly influences the final decision boundary in favor of the negative class.
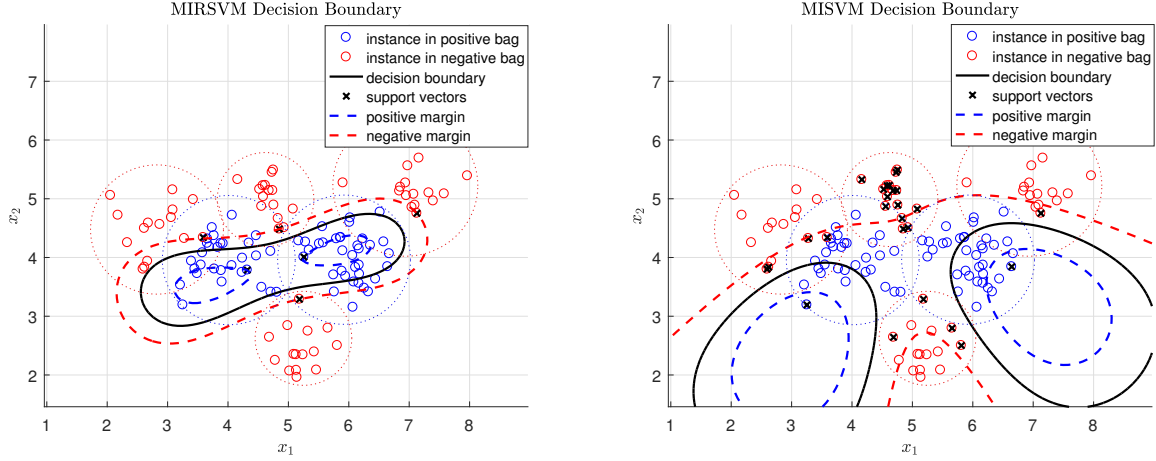
Fig. 4.3.: Difference between MIRSVM and MISVM on a random 2-dimensional toy dataset. Note the differing number of support vectors produced by the two methods. MIRSVM has 6, one for each bag, and MISVM has 29. Also note the smoother representation of the data distribution given by MIRSVM's decision boundary, unlike MISVM whose decision boundary was greatly influenced by the larger number of support vectors belonging to the negative class with respect to the only 2 positive support vectors.

4.3   Experimental Environment

This section presents the experimental setup and comparison of our contribution, as well as 11 other widely used methods on 15 different benchmark datasets. The main aim of the experiments is to compare our contribution to other multi-instance support vector machines, contemporary multi-instance learners, and ensemble methods.

Table 4.2 presents a summary of the 15 datasets used throughout the experiments, where the number of attributes, bags, and total number of instances are shown. The datasets were obtained from the Weka[1] [88] and KEEL[2] [3] dataset repositories.

The experimental environment was designed to test the difference in performance of the proposed method against 11 competing algorithms, contrasting instance-level, bag-level, and ensemble methods. Instance-level methods include MIOptimalBall, MIBoost, MISVM, MIDD, and MIWrapper. Bag-level methods include MISMO, SimpleMI, miGraph, and TLC.

---

[1]http://www.cs.waikato.ac.nz/ml/weka

[2]http://sci2s.ugr.es/keel/datasets.php

Table 4.2.: Multi-Instance (MI) Classification datasets

Dataset	Attributes	Positive Bags	Negative Bags	Total	Instances	Avg. Bag Size
Suramin	20	7	6	13	2898	222.92
EastWest	24	10	10	20	213	10.65
WestEast	24	10	10	20	213	10.65
Musk1	166	47	45	92	476	5.17
Musk2	166	39	62	101	6728	66.61
Webmining	5863	21	92	113	3423	30.29
Mutagenesis-atoms	10	125	63	188	1618	8.61
Mutagenesis-bonds	16	125	63	188	4081	21.71
Mutagenesis-chains	24	125	63	188	5424	28.85
TRX	8	25	168	193	26611	137.88
Elephant	230	100	100	200	1391	6.96
Fox	230	100	100	200	1320	6.60
Tiger	230	100	100	200	1188	5.94
Component	200	423	2707	3130	36894	11.79
Function	200	443	4799	5242	55536	10.59

The ensemble-based bag-space methods, Bagging and Stacking, were also used. The base algorithms selected for the ensembles Bagging and Stacking were TLC, and TLC and SimpleMI, respectively. These algorithms were chosen because they have shown considerable performance in learning multi-instance models, while also having their frameworks readily available for reproducing their results through MILK, the Multi-Instance Learning Kit[3] [186], used in conjunction with the Weka framework. Experiments were run on an Intel i7-6700k CPU with 32GB RAM. MIRSVM was implemented in MATLAB while the referenced algorithms are available in the Java implementation of Weka with the exception of miGraph which was made available by Zhou et al.[4] and tested in MATLAB.

We have compared the models trained on the different hyperparameters using the cross-validation (CV) procedure which ensures that the models performances are accurately assessed and the model built is not biased towards the full dataset. The tuning of the model includes finding the best penalty parameter, C, as well as the best shape parameter for the Gaussian radial basis function kernel, σ. The best hyperparameters were chosen from the

---

[3]http://www.cs.waikato.ac.nz/ml/milk

[4]http://lamda.nju.edu.cn/code_miGraph.ashx

following 6×6 possible combination runs, shown in Equations (4.16a) and (4.16b), referred to as (4.16).

$$C \in \{0.1, 1, 10, 100, 1000, 10000\} \tag{4.16a}$$

$$\sigma \in \{0.1, 0.5, 1, 2, 5, 10\} \tag{4.16b}$$

These parameters were also used for the compared SVM methods. This was done in order to keep the experimental environment controlled and ensure fair evaluation of the multi-instance SVM algorithms. The parameters for the referenced algorithms used throughout the experiments were those specified by their authors.

4.4 Results & Statistical Analysis

The classification performance was measured using five metrics: Accuracy (4.17a), Precision (4.17b), Recall (4.17c), Cohen's kappa rate (4.17d), and Area under ROC curve (AUC) (4.17e). The Precision and Recall measures were reported because Accuracy alone can be misleading when classes are imbalanced, as is the case with the *component* and *function* datasets, which respectively have six and ten times as many negative bags than positive. Cohen's Kappa Rate and the AUC measures are used as complementary measures in order to evaluate the algorithms comprehensively. Cohen's kappa rate, shown in Equation (4.17d), evaluates classifier merit according to the class distribution and ranges between -1 (full disagreement), 0 (random classification), and 1 (full agreement). The AUC metric highlights the trade-off between the true positive rate, or recall, and the false positive rate, as shown in Equation (4.17e). The values of the true positive (TP), true negative (TN), false positive (FP), and false negative samples (FN) were first collected for each of the classifiers, then the metrics were computed using the equations shown in (4.17) on the n' bags of the test data, where $n' = TP + FP + TN + FN$. The run times (training and testing times) of each algorithm are also reported to analyze the scalability and speed of each of the algorithms across differently sized datasets.

72

The results for the following are shown in Tables 4.3, 4.5, 4.7, 4.9, 4.11, and 4.13.

$$\text{Accuracy} \qquad \frac{TP + TN}{n'} \qquad (4.17a)$$

$$\text{Precision} \qquad \frac{TP}{TP + FP} \qquad (4.17b)$$

$$\text{Recall} \qquad \frac{TP}{TP + FN} \qquad (4.17c)$$

$$\text{Cohen's Kappa Rate} \qquad \frac{n' - \frac{(TP + FN) * (TP + FP)}{n'}}{1 - \frac{(TP + FN) * (TP + FP)}{n'}} \qquad (4.17d)$$

$$\text{Area Under ROC Curve} \qquad \frac{1 + \frac{TP}{TP + FN} - \frac{FP}{FP + TN}}{2} \qquad (4.17e)$$

In order to analyze the performances of the multiple models, non-parametric statistical tests are used to validate the experimental results obtained. 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 in Equations (4.17) are presented in the last row of the results tables, and the lowest (best) rank value is typeset in bold. Table 4.14 contains the ranks and meta-rank of all methods, which helps determine and visualize the best performing algorithms across all datasets and metrics.

After the Iman-Davenport test indicates significant differences, the Bonferroni-Dunn post-hoc test [65] is then used to find where they occur between algorithms by assuming the classifiers' performances are different by at least some critical value. Below each result table, a figure highlighting the critical distance (in gray), from the best ranking algorithm to the rest, is shown. The algorithms to the right of the critical distance bar perform statistically significantly worse than the control algorithm, MIRSVM. Figures 4.4, 4.5, 4.6, 4.7, 4.8, 4.9 show the results of the Bonferroni-Dunn post-hoc procedure over the metrics in (4.17), as well as the meta-rank results in Table 4.14. The Holm (multiple) and Wilcoxon

(pairwise) rank-sum post-hoc tests [94] were then run for each of the metrics to compute multiple and pairwise comparisons between the proposed algorithm and the other methods compared, investigating whether statistical differences exist among the algorithms' results. Tables 4.4, 4.6, 4.8, 4.10, and 4.12 show the p-values for the Holm test for $\alpha = 0.05$, and the rank-sums and adjusted p-values for the Wilcoxon test.

4.4.1 Accuracy

The results for accuracy indicate that the bag-based and ensemble learners perform better than the instance-based and wrapper methods. MIRSVM achieves the best accuracy over 5 of the 15 datasets with a competitive average against miGraph, Bagging, Stacking, and TLC. Note that MIRSVM performs better than MISVM for all datasets, indicating that using representatives from each bag and limiting the number of support vectors per negative bag improves the classification performance. The instance-level classifiers and wrapper methods,

Table 4.3.: Accuracy for MI classifiers

Datasets	MIRSVM	miGraph	MIBoost	MIOptimalBall	MIDD	MIWrapper	MISMO	MISVM	SimpleMI	TLC	Bagging	Stacking
suramin	0.8000	**0.8462**	0.5000	0.7250	0.4250	0.5000	0.7250	0.5000	0.5000	0.6000	0.6650	0.4615
eastWest	**0.8000**	0.7000	0.5000	0.7250	0.6125	0.5000	0.7125	0.5625	0.5000	0.6000	0.6000	0.4500
westEast	0.7500	0.7500	0.5000	0.3750	0.4500	0.5000	0.7375	0.4125	0.5000	**0.9649**	0.5625	0.6375
musk1	**0.9022**	0.8152	0.5109	0.7717	0.8804	0.5109	0.7826	0.7609	0.5109	0.8587	0.8142	0.8587
musk2	0.8218	0.7426	0.6139	0.7723	0.7228	0.6139	0.7030	0.7129	0.6139	0.6238	**0.8756**	0.6733
webmining	0.8500	0.8142	0.8142	0.7699	0.8142	0.8142	0.8407	0.6903	0.8142	0.8142	**0.9358**	0.8053
trx	0.8860	0.8964	0.8705	**0.9016**	0.8808	0.8705	0.8705	0.8705	0.8705	0.8756	0.6450	0.8860
mutagenesis-atoms	0.7714	0.7606	0.6649	0.6436	0.7074	0.6649	0.6915	0.6649	0.6649	**0.7766**	**0.7766**	0.7606
mutagenesis-bonds	0.8252	0.7872	0.6649	0.6915	0.7713	0.6649	0.7979	0.6649	0.6649	0.8351	0.8351	**0.8564**
mutagenesis-chains	**0.8411**	0.7926	0.6649	0.6702	0.7766	0.6649	0.8351	0.6649	0.6649	0.8404	0.8404	0.8351
tiger	0.7750	0.7950	0.5000	0.5000	0.7100	0.5000	0.7200	0.7550	0.5000	0.6650	**0.8000**	0.7250
elephant	0.8300	0.8300	0.5000	0.5000	0.7900	0.5000	0.8100	0.8000	0.5000	0.8000	0.5625	0.8250
fox	0.6550	0.6300	0.5000	0.5000	0.5800	0.5000	0.5250	0.4750	0.5000	0.6450	**0.8587**	0.6500
component	**0.9366**	0.9153	0.8649	0.8696	0.8780	0.8649	0.8968	0.8703	0.8649	0.9358	0.6000	0.9355
function	0.9523	0.9405	0.9155	0.9138	0.9193	0.9155	0.9376	0.9195	0.9155	**0.9649**	0.6238	0.9647
Average	**0.8264**	0.8010	0.6390	0.6886	0.7279	0.6390	0.7724	0.6883	0.6390	0.7598	0.7598	0.7550
Rank	**2.2000**	3.8667	9.6000	7.8667	6.5667	9.6000	5.3333	8.5667	9.6000	4.7000	4.8667	5.2333

Fig. 4.4.: Bonferroni-Dunn test for Accuracy

Table 4.4.: Holm and Wilcoxon tests for Accuracy

MIRSVM vs.	miGraph	MIBoost	MIOptimalBall	MIDD	MIWrapper	MISMO	MISVM	SimpleMI	TLC	Bagging	Stacking
Holm p-value	0.0500	0.0045	0.0071	0.0083	0.0050	0.0100	0.0063	0.0056	0.0250	0.0167	0.0125
Wilcoxon p-value	0.0279	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0067	0.3028	0.0103
Wilcoxon R^+	98.500	120.00	119.00	120.00	120.00	120.00	120.00	120.00	106.00	79.000	104.00
Wilcoxon R^-	21.500	0.0000	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000	14.000	41.000	16.000

such as MIBoost, MIWrapper, and SimpleMI perform the worst. This behavior emphasizes the importance of not making prior assumptions about the positive bags' distributions.

Figure 4.4 and Table 3.4 show the results for the statistical analysis on the accuracy results. The algorithms with ranking higher than 5.63 (MIRSVM rank + Bonferroni-Dunn critical value), to the right of the gray bar in Figure 4.4, perform statistically worse than MIRSVM. Table 3.4 shows the p-values of the Holm and Wilcoxon tests and their results complement one another. Holm's procedure rejects those hypotheses having a p-value ≤ 0.01, thus indicating that MIRSVM performs significantly better than all methods except mi-Graph, Bagging, Stacking, and TLC. The Wilcoxon p-values show significant differences exist among all algorithms except miGraph, Bagging, and Stacking. They also show that MIRSVM has significantly better accuracy than MIBoost, MIOptimalBall, MIDD, MIWrapper, MISMO, MISVM, and SimpleMI, each having respectively small p-values, highlighting MIRSVM's superior classification accuracy.

4.4.2 Precision & Recall

Precision and recall are conflicting metrics that must be evaluated together in order to observe their behavior, since they are both used to measure relevance. The results for MIWrapper and SimpleMI indicate that they are unstable classifiers, exhibiting extreme variance in behavior, making them unsuitable for real-world applications. It is also interesting to analyze the performance on the mutagenesis datasets which have a larger number of positive bags than negative, where MISVM, MIBoost, MIWrapper, and SimpleMI predict all bags as negative. Additionally, while MISMO obtains unbiased results on these datasets, MIRSVM significantly outperforms it over both precision and recall, achieving a better trade-off.

Table 4.5.: Precision for MI classifiers

Datasets	MIRSVM	miGraph	MIBoost	MIOptimalBall	MIDD	MIWrapper	MISMO	MISVM	SimpleMI	TLC	Bagging	Stacking
suramin	0.7778	0.7778	**1.0000**	**1.0000**	0.2857	**1.0000**	**1.0000**	0.5000	**1.0000**	0.6429	0.6514	0.4000
eastWest	0.7143	0.7000	0.5000	0.8750	0.5882	0.5000	0.7429	**1.0000**	0.5000	0.6053	0.6053	0.4444
westEast	0.7272	0.7273	0.5000	0.2727	0.4600	0.5000	0.6939	0.3600	0.5000	0.5581	**0.9729**	0.6038
musk1	0.8519	0.7778	**1.0000**	0.9286	0.9048	**1.0000**	0.8049	0.8108	**1.0000**	0.8478	0.8817	0.8478
musk2	0.7059	0.7826	0.6139	0.7826	0.7576	0.6139	0.7424	0.7538	0.6139	0.7400	**0.9138**	0.7164
webmining	0.7500	**1.0000**	0.8142	0.8173	0.8142	0.8142	0.8936	**1.0000**	0.8142	0.8817	0.9462	0.8500
trx	**1.0000**	0.8571	0.8705	0.9306	0.9191	0.8705	0.8705	0.8705	0.8705	0.9138	0.6747	0.9011
mutagenesis-atoms	0.7872	0.7985	**1.0000**	0.4630	0.6111	**1.0000**	0.5439	**1.0000**	**1.0000**	0.7059	0.7059	0.6667
mutagenesis-bonds	0.8468	0.8195	**1.0000**	0.5385	0.7500	**1.0000**	0.6812	**1.0000**	**1.0000**	0.7857	0.7857	0.8333
mutagenesis-chains	0.8571	0.8116	**1.0000**	0.5091	0.7059	**1.0000**	0.7759	**1.0000**	**1.0000**	0.7705	0.7705	0.7581
tiger	0.7365	0.7323	0.5000	0.5000	0.6944	0.5000	0.7444	0.7802	0.5000	0.6514	**0.8000**	0.7320
elephant	0.8576	**0.8750**	0.5000	0.5000	0.7959	0.5000	0.8444	0.7679	0.5000	0.8000	0.5581	0.8283
fox	0.6040	0.6275	0.5000	0.5000	0.5833	0.5000	0.5287	0.4854	0.5000	0.6747	**0.8478**	0.6705
component	**0.9866**	0.7782	0.8649	0.8778	0.8902	0.8649	0.8958	0.8696	0.8649	0.9462	0.6429	0.9449
function	0.8459	0.6775	0.9155	0.9202	0.9317	0.9155	0.9376	0.9197	0.9155	**0.9729**	0.7400	0.9726
Average	0.8033	0.7828	0.7719	0.6944	0.7128	0.7719	0.7800	**0.8079**	0.7719	0.7665	0.7665	0.7447
Rank	**5.3333**	6.1333	7.1000	7.3333	7.3667	7.1000	5.8667	5.8667	7.1000	5.9000	6.3333	6.5667

Fig. 4.5.: Bonferroni-Dunn test for Precision

Table 4.6.: Holm and Wilcoxon tests for Precision

MIRSVM vs.	miGraph	MIBoost	MIOptimalBall	MIDD	MIWrapper	MISMO	MISVM	SimpleMI	TLC	Bagging	Stacking
Holm p-value	0.0125	0.0056	0.0050	0.0045	0.0063	0.0250	0.0500	0.0071	0.0167	0.0100	0.0083
Wilcoxon p-value	0.4212	0.5614	0.0946	0.0256	0.5614	0.4212	0.8039	0.5614	0.1354	0.4543	0.1354
Wilcoxon R$^+$	75.000	71.000	90.000	99.000	71.000	75.000	55.000	71.000	87.000	74.000	87.000
Wilcoxon R$^-$	45.000	49.000	30.000	21.000	49.000	45.000	65.000	49.000	33.000	46.000	33.000

Table 4.7.: Recall for MI classifiers

Datasets	MIRSVM	miGraph	MIBoost	MIOptimalBall	MIDD	MIWrapper	MISMO	MISVM	SimpleMI	TLC	Bagging	Stacking
suramin	**1.0000**	**1.0000**	0.0000	0.4500	0.1000	0.0000	0.4500	0.5000	0.0000	0.4500	0.7100	0.3333
eastWest	**1.0000**	0.7000	0.7000	0.5250	0.7500	0.7000	0.6500	0.1250	**1.0000**	0.5750	0.5750	0.4000
westEast	0.8000	0.8000	0.9000	0.1500	0.5750	0.9000	0.8500	0.2250	**1.0000**	0.6000	0.9892	0.8000
musk1	**0.9787**	0.8936	0.0000	0.5778	0.8444	0.0000	0.7333	0.6667	0.0000	0.8667	0.8913	0.8667
musk2	0.9231	0.4615	**1.0000**	0.8710	0.8065	**1.0000**	0.7903	0.7903	**1.0000**	0.5968	0.9464	0.7742
webmining	0.2857	0.0000	**1.0000**	0.9239	**1.0000**	**1.0000**	0.9130	0.6196	**1.0000**	0.8913	0.9815	0.9239
trx	0.4833	0.2400	**1.0000**	0.9583	0.9464	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.9464	0.5600	0.9762
mutagenesis-atoms	**0.8880**	0.8560	0.0000	0.3968	0.3492	0.0000	0.4921	0.0000	0.0000	0.5714	0.5714	0.5714
mutagenesis-bonds	**0.8960**	0.8720	0.0000	0.5556	0.4762	0.0000	0.7460	0.0000	0.0000	0.6984	0.6984	0.7143
mutagenesis-chains	**0.9120**	0.8960	0.0000	0.4444	0.5714	0.0000	0.7143	0.0000	0.0000	0.7460	0.7460	0.7460
tiger	0.8700	0.9300	0.5000	**1.0000**	0.7500	0.5000	0.6700	0.7100	**1.0000**	0.7100	0.8000	0.7100
elephant	0.9100	0.7700	0.6000	**1.0000**	0.7800	0.6000	0.7600	0.8600	**1.0000**	0.8000	0.6000	0.8200
fox	0.9000	0.6400	0.7000	**1.0000**	0.5600	0.7000	0.4600	0.8300	**1.0000**	0.5600	0.8667	0.5900
component	0.5839	0.5225	**1.0000**	0.9867	0.9797	**1.0000**	0.9967	**1.0000**	**1.0000**	0.9815	0.4500	0.9826
function	0.5327	0.5643	**1.0000**	0.9919	0.9840	**1.0000**	0.9983	0.9994	**1.0000**	0.9892	0.5968	0.9894
Average	**0.7976**	0.6764	0.5600	0.7221	0.6982	0.5600	0.7483	0.5551	0.6667	0.7322	0.7322	0.7465
Rank	4.8667	6.5667	6.8667	6.3333	7.3667	6.8667	6.7000	7.4333	**4.8333**	7.3667	6.0667	6.7333
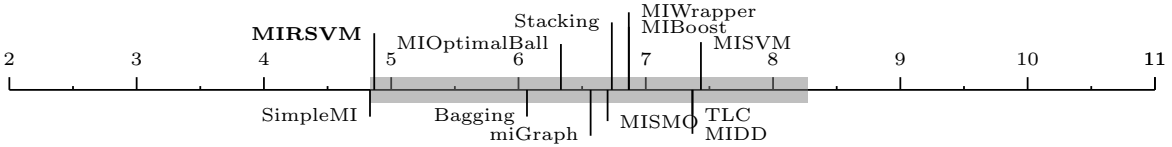
Fig. 4.6.: Bonferroni-Dunn test for Recall

Table 4.8.: Holm and Wilcoxon tests for Recall

MIRSVM vs.	miGraph	MIBoost	MIOptimalBall	MIDD	MIWrapper	MISMO	MISVM	SimpleMI	TLC	Bagging	Stacking
Holm p-value	0.0125	0.0063	0.0167	0.0050	0.0071	0.0100	0.0045	0.0500	0.0056	0.0250	0.0083
Wilcoxon p-value	0.0060	0.2077	0.5614	0.4543	0.2077	0.6387	0.1070	0.6603	0.5995	0.1354	0.5721
Wilcoxon R$^+$	106.50	83.000	71.000	74.000	83.000	69.000	89.000	60.000	70.000	87.000	62.000
Wilcoxon R$^-$	13.500	37.000	49.000	46.000	37.000	51.000	31.000	45.000	50.000	33.000	43.000

Figure 4.5 and 4.6 show that there are no significant differences between the precision and recall results obtained by all algorithms. Note, MIRSVM outperforms both ensemble methods according to recall, despite them exhibiting good accuracy and precision, indicating they are strongly conservative towards predicting positive bags. Holm's test indicates significant differences exist between MIRSVM and all algorithms except miGraph, MISMO, MISVM, and TLC for precision, and all the above along with SimpleMI, MIOptimalBall, and Bagging for recall. The Wilcoxon test does not reflect significant differences for precision, does for recall. The tests are severely biased due to the classifier's extreme unbalanced behavior, whereas MIRSVM demonstrates proper balance of the precision-recall trade-off.

4.4.3 Cohen's Kappa Rate

Table 4.9 shows the Cohen's Kappa rate results obtained by the algorithms. These results support the accuracy achieved by the algorithms, in the sense that the instance-based

Table 4.9.: Cohen's Kappa Rate for MI classifiers

Datasets	MIRSVM	miGraph	MIBoost	MIOptimalBall	MIDD	MIWrapper	MISMO	MISVM	SimpleMI	TLC	Bagging	Stacking
suramin	**0.6829**	**0.6829**	0.0000	0.4500	-0.1500	0.0000	0.4500	0.0000	0.0000	0.2000	0.3300	-0.0964
eastWest	**0.6000**	0.4000	0.0000	0.4500	0.2250	0.0000	0.4250	0.1250	0.0000	0.2000	0.2000	-0.1000
westEast	0.5000	0.5000	0.0000	-0.2500	-0.1000	0.0000	0.4750	-0.1750	0.0000	0.1250	**0.7529**	0.2750
musk1	**0.8036**	0.6290	0.0000	0.5396	0.7604	0.0000	0.5642	0.5197	0.0000	0.7174	0.3744	0.7174
musk2	**0.6540**	0.4123	0.0000	0.5031	0.4039	0.0000	0.3613	0.3856	0.0000	0.2492	0.3858	0.2940
webmining	0.3468	0.0000	0.0000	0.0246	0.0000	0.0000	0.4535	0.3771	0.0000	0.3744	**0.6945**	0.2458
trx	0.2100	0.3375	0.0000	**0.5228**	0.4224	0.0000	0.0000	0.0000	0.0000	0.3858	0.2900	0.3364
mutagenesis-atoms	**0.5395**	0.4431	0.0000	0.1709	0.2654	0.0000	0.2909	0.0000	0.0000	0.4738	0.4738	0.4431
mutagenesis-bonds	0.5699	0.5070	0.0000	0.3131	0.4356	0.0000	0.5569	0.0000	0.0000	0.6195	0.6195	**0.6659**
mutagenesis-chains	0.6303	0.5094	0.0000	0.2359	0.4738	0.0000	0.6225	0.0000	0.0000	**0.6391**	**0.6391**	0.6285
tiger	0.5500	0.5000	0.0000	0.0000	0.4200	0.0000	0.4400	0.5100	0.0000	0.3300	**0.6000**	0.4500
elephant	**0.7000**	0.6600	0.0000	0.0000	0.5800	0.0000	0.6200	0.6000	0.0000	0.6000	0.1250	0.6500
fox	0.3100	0.2600	0.0000	0.0000	0.1600	0.0000	0.0500	-0.0500	0.0000	0.2900	**0.7174**	0.3000
component	0.6644	0.5795	0.0000	0.1613	0.2836	0.0000	0.3656	0.0675	0.0000	**0.6945**	0.2000	0.6906
function	0.6292	0.5838	0.0000	0.0966	0.2801	0.0000	0.4083	0.0933	0.0000	**0.7529**	0.2492	0.7507
Average	**0.5594**	0.4730	0.0000	0.2145	0.2973	0.0000	0.4056	0.1635	0.0000	0.4434	0.4434	0.4167
Rank	**2.6333**	4.2000	10.1667	7.0000	6.5333	10.1667	5.2333	8.3667	10.1667	4.2667	4.2667	5.0000

Fig. 4.7.: Bonferroni-Dunn test for Cohen's Kappa rate

Table 4.10.: Holm and Wilcoxon tests for Cohen's Kappa rate

MIRSVM vs.	miGraph	MIBoost	MIOptimalBall	MIDD	MIWrapper	MISMO	MISVM	SimpleMI	TLC	Bagging	Stacking
Holm p-value	0.0500	0.0045	0.0071	0.0083	0.0050	0.0100	0.0063	0.0056	0.0167	0.0250	0.0125
Wilcoxon p-value	0.0121	0.0001	0.0012	0.0012	0.0001	0.0006	0.0001	0.0001	0.1205	0.2077	0.0946
Wilcoxon R^+	91.500	120.00	113.00	113.00	120.00	115.00	119.00	120.00	88.000	83.000	90.000
Wilcoxon R^-	13.500	0.0000	7.0000	7.0000	0.0000	5.0000	1.0000	0.0000	32.000	37.000	30.000

and wrapper methods perform worse than bag-based and ensemble learners. MIRSVM's kappa values all fall within the range (0.5-1], indicating that its merit as a classifier agrees with the class distribution and is not random. Note that MIOptimalBall, MIDD, MISVM, MISMO, and Stacking contain some negative kappa values, indicating performance worse than the default-hypothesis. MIBoost, SimpleMI, and MIWrapper are shown to randomly classify all 15 datasets. Figure 4.7 and Table 4.10 show the results of the statistical analysis on the Cohen's Kappa Rate results. The Holm and Wilcoxon procedures reflect results similar to the Bonferroni-Dunn test, where MIRSVM performs significantly better than MIOptimalBall, MIDD, MISVM, MIWrapper, MIBoost, and SimpleMI, having p-values < 0.01. This supports MIRSVM's performance as a competitive classifier.

Table 4.11.: AUC for MI classifiers

Datasets	MIRSVM	miGraph	MIBoost	MIOptimalBall	MIDD	MIWrapper	MISMO	MISVM	SimpleMI	TLC	Bagging	Stacking
suramin	**0.8333**	**0.8333**	0.5000	0.7250	0.4250	0.5000	0.7250	0.5000	0.5000	0.6000	0.6650	0.4524
eastWest	**0.8000**	0.7000	0.5000	0.7250	0.6125	0.5000	0.7125	0.5625	0.5000	0.6000	0.6000	0.4500
westEast	0.7500	0.7500	0.5000	0.3750	0.4500	0.5000	0.7375	0.4125	0.5000	0.5625	**0.8456**	0.6375
musk1	**0.9005**	0.8135	0.5000	0.7676	0.8797	0.5000	0.7816	0.7589	0.5000	0.8589	0.6837	0.8589
musk2	**0.8406**	0.6904	0.5000	0.7432	0.6981	0.5000	0.6772	0.6900	0.5000	0.6317	0.6732	0.6435
webmining	0.6320	0.5000	0.5000	0.5096	0.5000	0.5000	0.7184	0.8098	0.5000	0.6837	**0.8123**	0.6048
trx	0.6500	0.6170	0.5000	**0.7392**	0.6932	0.5000	0.5000	0.5000	0.5000	0.6732	0.6450	0.6281
mutagenesis-atoms	0.7106	0.7137	0.5000	0.5824	0.6186	0.5000	0.6420	0.5000	0.5000	**0.7257**	**0.7257**	0.7137
mutagenesis-bonds	0.7856	0.7455	0.5000	0.6578	0.6981	0.5000	0.7850	0.5000	0.5000	0.8012	0.8012	**0.8211**
mutagenesis-chains	**0.8252**	0.7417	0.5000	0.6142	0.7257	0.5000	0.8051	0.5000	0.5000	0.8170	0.8170	0.8130
tiger	0.7750	0.7950	0.5000	0.5000	0.7100	0.5000	0.7200	0.7550	0.5000	0.6650	**0.8000**	0.7250
elephant	0.8200	**0.8300**	0.5000	0.5000	0.7900	0.5000	0.8100	0.8000	0.5000	0.8000	0.5625	0.8250
fox	0.6550	0.6300	0.5000	0.5000	0.5800	0.5000	0.5250	0.4750	0.5000	0.6450	**0.8589**	0.6500
component	0.7855	0.7496	0.5000	0.5536	0.6033	0.5000	0.6272	0.5201	0.5000	**0.8123**	0.6000	0.8081
function	0.7563	0.7698	0.5000	0.5298	0.6015	0.5000	0.6391	0.5268	0.5000	**0.8456**	0.6317	0.8434
Average	**0.7680**	0.7253	0.5000	0.6015	0.6390	0.5000	0.6937	0.5874	0.5000	0.7148	0.7148	0.6983
Rank	**2.7667**	4.2667	10.1667	7.0000	6.5333	10.1667	5.2333	8.2333	10.1667	4.2667	4.2667	4.9333
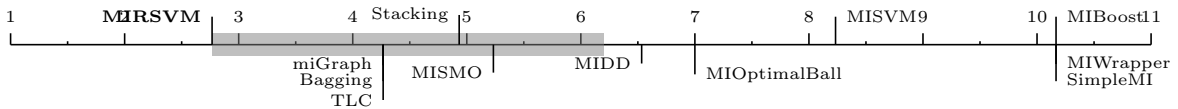
Fig. 4.8.: Bonferroni-Dunn test for AUC

Table 4.12.: Holm and Wilcoxon tests for AUC

MIRSVM vs.	miGraph	MIBoost	MIOptimalBall	MIDD	MIWrapper	MISMO	MISVM	SimpleMI	TLC	Bagging	Stacking
Holm p-value	0.0167	0.0045	0.0071	0.0083	0.0050	0.0100	0.0063	0.0056	0.0250	0.0500	0.0125
Wilcoxon p-value	0.0166	0.0001	0.0002	0.0003	0.0001	0.0012	0.0009	0.0001	0.2523	0.3028	0.0781
Wilcoxon R^+	90.000	120.00	118.00	117.00	120.00	113.00	114.00	120.00	81.000	79.000	91.500
Wilcoxon R^-	15.000	0.0000	2.0000	3.0000	0.0000	7.0000	6.0000	0.0000	39.000	41.000	28.500

4.4.4   Area Under ROC Curve

Table 4.11 shows AUC results obtained by the algorithms, which complement the accuracy and kappa rate, emphasizing the better performance of bag-based methods. MIRSVM achieves the best AUC score on 5 of the 15 datasets, while MIBoost, SimpleMI, and MI-Wrapper obtain the worst results. Their AUC score indicates random predictor behavior, having values = 0.5. Bag-level methods all obtain scores between 0.7 and 0.77 indicating a high true positive rate and a low false positive rate, which is reflected by the precision and recall results. Figure 4.8 and Table 4.12 show that MIRSVM performs significantly better than 6 out of the 11 competing algorithms. Holm's procedure indicates that significant differences exist between MIRSVM and all algorithms except miGraph, TLC, Bagging, and Stacking. MISVM's true positive rate could be affected because of the possible imbalance of support vectors from the positive and negative classes (favoring the negative). Note that the Wilcoxon p-values for MIWrapper, MIBoost, and SimpleMI are 0.0001.

4.4.5   Overall Comparison

Table 4.13 shows the run times, in seconds, for each algorithm. MIRSVM has the fastest run time and is ranked second. MIRSVM shows very good scalability considering the number of features, such as in the webmining dataset which comprises of 5863 attributes. Additionally, taking into account the number of instances as seen in the two largest datasets, component and function, MIRSVM displays superior scalability. It is important to note that quadratic programming solvers are not the most efficient tools for solving optimization problems in terms of run time, and yet MIRSVM still is shown to perform competitively against the current widely used algorithms. The scalability of MIRSVM is founded on the speedy rate of bag-representative convergence, as shown previously in Figure 4.2.

SimpleMI achieves the highest rank and competitive run times because, rather than use the instances in each bag to train a model, it takes the mean value of the instances in a bag and uses that for training. Even though SimpleMI has fast run-times, its performance over

the previous metrics is shown to be random and not as effective as the bag-level methods.

Table 4.14 shows the ranks achieved by each of the metrics along with the average and meta-ranks, to illustrate the overall performance across all metrics. MIRSVM has the best meta-rank (rank of the ranks) and the miGraph method has the second best. The meta-ranks also highlight the better performance of bag-level methods over instance-level and wrapper methods, emphasizing the importance of training at the bag-level. Not only does MIRSVM use bag-level information during classification, but it also optimizes over the instances within the bag, which helps determine which instances contribute the most information about the bags label. SimpleMI, MIWrapper, MIBoost, MISVM, and MDD have the worst performance compared to MIRSVM and miGraph. Specifically, it is evident from the precision and recall results that MIBoost, MIWrapper, and SimpleMI, for example, classify all bags as negative

Table 4.13.: Run Time (seconds) for MI classifiers

Datasets	MIRSVM	miGraph	MIBoost	MIOptimalBall	MIDD	MIWrapper	MISMO	MISVM	SimpleMI	TLC	Bagging	Stacking
suramin	**0.1**	19.7	8.8	30.5	7922.0	9.5	52.3	333.9	7.2	35.5	183.0	90.6
eastWest	**0.1**	3.0	5.5	9.4	217.1	6.3	14.8	21.4	5.8	15.4	15.4	15.2
westEast	**0.1**	2.8	6.5	7.8	79.7	6.5	14.7	99.5	6.0	16.6	12128.1	10.8
musk1	**0.4**	56.8	13.4	32.1	3542.6	20.6	89.7	198.4	11.1	93.0	86272.6	759.5
musk2	**2.3**	452.3	97.3	782.9	126016.8	208.3	1799.4	26093.5	16.1	1772.2	2229.3	16759.0
webmining	**300.6**	302.5	45745.4	60474.8	47601.4	68736.7	51923.6	105622.3	2685.9	86272.6	9861.5	592948.9
trx	61.8	2206.4	17.6	682.3	339110.5	19.3	8670.3	134622.1	**7.4**	2229.3	243.3	11927.9
mutagenesis-atoms	9.8	193.1	8.8	99.2	2623.0	8.0	55.0	53.5	**6.4**	44.0	44.0	153.9
mutagenesis-bonds	**8.3**	410.3	10.2	310.2	17538.7	12.3	457.4	2794.8	8.4	131.1	131.1	853.1
mutagenesis-chains	19.3	513.4	12.0	525.0	48982.7	14.8	2451.9	6637.4	**7.2**	224.4	224.4	1619.0
tiger	29.5	302.8	44.5	157.8	23220.5	56.2	208.0	608.8	**16.2**	183.0	212.1	1085.0
elephant	47.7	306.7	45.5	243.9	56456.2	69.7	232.1	1114.3	20.8	212.1	**16.6**	1462.2
fox	81.0	303.1	44.2	206.1	27773.8	66.0	369.6	891.5	**23.5**	243.3	93.0	1729.1
component	231.7	3091.0	572.5	228209.6	96263.9	1096.9	629366.4	37224.6	144.0	9861.5	**35.5**	79149.8
function	740.3	8162.7	935.5	768458.0	350124.7	1887.5	1052225.3	565026.4	**232.8**	12128.1	1772.2	185918.5
Average	**102.2**	1088.4	3171.2	70682.0	76498.2	4814.6	116528.7	58756.2	213.3	7564.1	7564.1	59632.2
Rank	2.3	6.2	3.1	7.2	11.1	4.3	8.5	10.1	**1.9**	7.2	6.5	9.7

Table 4.14.: Overall ranks comparison for MI classifiers

| Ranks	MIRSVM	miGraph	MIBoost	MIOptimalBall	MIDD	MIWrapper	MISMO	MISVM	SimpleMI	TLC	Bagging	Stacking
Accuracy	**2.2000**	3.8667	9.6000	7.8667	6.5667	9.6000	5.3333	8.5667	9.6000	4.7000	4.8667	5.2333
Precision	5.3333	6.1333	7.1000	7.3333	7.3667	7.1000	5.8667	5.8667	7.1000	5.9000	6.3333	6.5667
Recall	4.8667	6.5667	6.8667	6.3333	7.3667	6.8667	6.7000	7.4333	**4.8333**	7.3667	6.0667	6.7333
Kappa	**2.6333**	4.2000	10.1667	7.0000	6.5333	10.1667	5.2333	8.3667	10.1667	4.2667	4.2667	5.0000
AUC	**2.7667**	4.2667	10.1667	7.0000	6.5333	10.1667	5.2333	8.2333	10.1667	4.2667	4.2667	4.9333
Time	2.2667	6.2000	3.1000	7.2000	11.0667	4.3000	8.5333	10.1333	**1.8667**	7.2000	6.4667	9.6667
Average	**3.3444**	5.2056	7.8333	7.1222	7.5722	8.0333	6.1500	8.1000	7.2889	5.6167	5.3778	6.3556
Rank	**1.3333**	3.6667	8.9167	7.7500	9.2500	9.0833	5.9167	8.7500	7.3333	5.2500	4.2500	6.5000
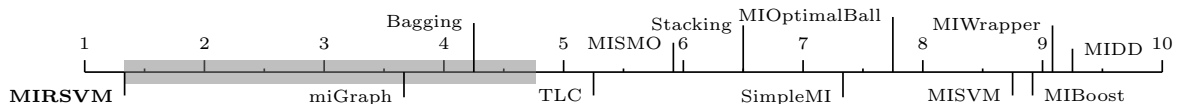
Fig. 4.9.: Bonferroni-Dunn test for overall ranks comparison

for datasets that have imbalanced class distributions which favor the negative class. This emphasizes the disadvantage of using wrapper methods and assuming the data distribution of the instances within positive bags. Although these algorithms are popular in literature, the experimental study clearly shows that recent bag-level and ensemble methods easily overcome traditional multi-instance learning algorithms.

In summary, MIRSVM offers improvement in terms of both accuracy and run-time when compared to referenced methods, especially those utilizing SVM-based algorithms.

4.5   Conclusions

This proposal consisted of a novel formulation and algorithm for the multiple-instance support vector machine problem, which optimizes bag classification via bag-representative selection. First, the primal formulation was posed and its dual was then derived and solution computed using a quadratic programming solver. This formulation was designed to utilize bag-level information and find an optimal separating hyperplane between bags, rather than individual instances, using the standard multi-instance assumption. The SMI assumption states that a bag is labeled positive if and only if at least one instance within a bag is positive, and is negative otherwise. The key features of the proposed algorithm MIRSVM are its ability to identify instances within positive and negative bags, i.e. the support vectors or representatives, that highly impact the decision boundary and margin, as well as avoiding uncertainties and issues caused by techniques that flatten, subset, or under-represent positive instances within positively labeled bags. Additionally, it exhibits desirable convergence and scalability, making it suitable for large-scale learning tasks.

The experimental study showed the better performance of MIRSVM compared with existing multi-instance support vector machines, traditional multi-instance learners, as well as ensemble methods. The results, according to a variety of performance metrics, were compared and further validated using statistical analysis with non-parametric tests which highlight the advantages of using bag-level based and ensemble learners, such as miGraph,

Bagging, and Stacking, while showing the instance-level based learners performed poorly in comparison or were deemed as strongly biased and unstable classifiers. Our proposal, MIRSVM, performs statistically better, neither compromising accuracy nor run-time while displaying a robust performance across all of the evaluated datasets. The research outcomes of this chapter have been published in [128].

CHAPTER 5

NOVEL ONLINE SVM USING WORST-VIOLATORS

Due to the ever-growing nature of dataset sizes, the need for scalable and accurate learning algorithms has become evident. Stochastic gradient descent methods are popular tools used to optimize large-scale learning problems because of their generalizability, simplicity, and scalability. This chapter proposes a novel stochastic, i.e. online, learning algorithm for solving the L1 support vector machine (SVM) problem: *OnLine Learning Algorithm using Worst-Violators* (OLLAWV). This chapter's scope is concerned with developing a unique algorithm for large data problems without parallelization and distributed techniques. Unlike other stochastic methods, OLLAWV eliminates the need for specifying a maximum number of iterations and the use of a regularization term. Rather than using a regularizer, OLLAWV uses early stopping for controlling the size of the margin, via a novel stopping criterion. The experimental study, performed under strict nested cross-validation, evaluates and compares the performance of OLLAWV with two modern SVM kernel methods that have been shown to outperform traditional, widely used approaches for solving L1-SVMs, such as *Sequential Minimal Optimization*. OLLAWV is also compared to five classic 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.

5.1   Online Learning Background

Since a comprehensive description of contemporary online learning algorithms was presented in Chapter 2, this section will introduce the concept of *Stochastic Gradient Descent* (SGD) for the unconstrained primal L1-SVM optimization problem, given by Equation 2.5. First, the notation used throughout the chapter is given, then a brief background on stochastic gradient descent and its use for the L1-SVM problem will be presented.

Table 5.1.: Summary of Online Learning Notation

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	$\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \ldots, \{\boldsymbol{x}_i, y_i\}, \ldots, (\boldsymbol{x}_n, y_n)\}$

5.1.1 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, $\boldsymbol{Y} \in \{-1, 1\}^n$. 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 5.1 summarizes the notation used throughout the chapter.

5.1.2 Stochastic Gradient Descent

Many learning algorithms function by repeatedly selecting a sample and updating the model's parameters based on only this chosen sample. These algorithms are called *stochastic*, or *online*, methods, which have been used for various real-world scenarios dating back many years ago, when algorithmic simplicity was a practical requirement [30, 144, 179]. Stochastic algorithms are still extremely useful nowadays because of their performance advantages, simplicity, and desirable sample complexity, in the context of large-scale learning problems.

The goal, for many learning algorithms, is to find an optimal weight vector $\boldsymbol{w}$ that minimizes some convex, differentiable risk function $f(\boldsymbol{w})$ over all samples. One approach for achieving this, is through *gradient descent*: an iterative procedure where the gradient of f, $\bigtriangledown f(\boldsymbol{w}) = \left(\frac{\partial f(\boldsymbol{w})}{\partial w_1}, \ldots, \frac{\partial f(\boldsymbol{w})}{\partial w_d} \right)$, is taken at each step and used to update the weights. At each iteration, the *batch* gradient descent algorithm involves computing the average gradient

across all training samples as shown in Equation 5.1,

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \eta \frac{1}{n} \sum_{i=1}^{n} \nabla_w f_i(\boldsymbol{w}^{(t)}), \tag{5.1}$$

where $\eta > 0 \in \mathbb{R}$ is the learning rate and usually decreases with the number of iterations, t. After T iterations, the algorithm returns the averaged weight vector $\bar{\boldsymbol{w}} = \frac{1}{T} \sum_{t=1}^{T} \boldsymbol{w}^{(t)}$. Different versions of the weight vector could also be returned, such as the last weight vector, $\boldsymbol{w}^{(T)}$, or the average of the last 25% of updates. More information about this can be found in [129]. Under large-scale learning conditions however, taking the gradient across all samples might be an impractical and computationally expensive task.

The *stochastic gradient descent* (SGD) algorithm is a drastic simplification of the gradient descent procedure. Rather than computing the gradient of f exactly, at each iteration it is instead an estimation of the gradient on the basis of a single, randomly picked sample $\boldsymbol{x}_i$, $\forall i \in \{1, \ldots, n\}$, illustrating the concept known as *stochastic approximation*. In the stochastic case, the model update then becomes,

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \eta \nabla_w f_i(\boldsymbol{w}^{(t)}). \tag{5.2}$$

Thus, each iteration of SGD is very cheap in terms of computation since it only involves the gradient at one sample and does not need to keep track of which samples were used during previous iterations. Note, the iterative sequence is not determined uniquely by the optimization function, starting point of $\boldsymbol{w}_0$, and sequence of step sizes; rather, it is a stochastic process whose behavior is determined by the sequence of random examples chosen at each iteration [27, 29]. If the samples are randomly selected from an online base system (ground truth distribution), SGD directly minimizes the expected risk. If the samples are drawn from a finite training set, the procedure minimizes the empirical risk [30].

The gradient descent procedure, stochastic or not, requires the function being minimized to be differentiable. Some loss functions are not fully differentiable (such as the Hinge loss function 2.4). In these cases, the subgradient of f at $\boldsymbol{w}$ can be used instead of the gradient.

5.1.3 Stochastic Gradient Descent for the Primal L1-SVM Problem

SVMs were typically treated and solved as a constrained quadratic optimization in dual space. Their early development was hindered because of the quadratic dependence on the number of samples and of the memory required to efficiently solve them. This led to the idea of optimizing over subsets of the data, also known as decomposition methods [26, 101, 111, 138]. Although these methods improved convergence rates, in practice, their superlinear (and sometimes cubic) dependence on the number of samples still was an issue in terms of slow run times when learning from massive datasets. Linear SVMs, taking advantage of linear kernels, were shown to outperform decomposition SVMs, motivating research for solving the SVM problem in the primal [48, 135]. It has been shown that when finding an approximate solution, primal optimization is superior [48]. These algorithms for the linear SVM are mostly based on the perceptron [73, 145] - the simplest stochastic learning algorithm for binary linear classification in the primal space. The perceptron cycles repeatedly, one sample at a time, updating the weight vector accordingly, until an appropriate condition is satisfied. By updating cyclically, these types of algorithms are able to process large amounts of data at a much faster speeds with low memory resources, consequently making them suitable for handling large datasets.

An approach similar to that of the perceptron aims to solve the regularized soft-margin loss through stochastic gradient descent, which allows a perceptron-like update while managing the model capacity [135]. This update is the only modification performed by the algorithm when a new sample is given, only occurs if some loss is incurred, and continues to be performed until some user intervention. Due to a lack of meaningful stopping criteria, the algorithm would keep running indefinitely unless some user intervenes. Notable representations of algorithms that use this type of approach can be found in Chapter 2.

Due to these characteristics, algorithms that use SGD are fundamentally different than methods such as mistake-driven perceptron-like approaches. However, Collobert and Bengio

later showed that with early stopping, the capacity of the perceptron could be controlled, using the idea of the margin. This was shown in a study comparing perceptrons, multi-layer perceptrons, and SVMs [53]. They also showed that it can be computationally expensive to train SVMs and perceptrons using SGD due to the regularization term, i.e. the term that controls the models capacity, and then showed alternative methods for remedying this issue. Namely, early stopping, and the removal of the regularization term or the learning rate, can control the size of the margin. However, this then raises the question about knowing when it is early enough to stop. This issue, along with the fact that solving the L1-SVM problem in the primal provides a better approximate solution than solving the dual, inspired the investigation for OLLAWV.

5.2   OLLAWV: OnLine Learning Algorithm using Worst-Violators

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 [106] 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 5.3 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, OLLAWV's number of iterations is equal to the final model's number of support vectors. 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.

$$[yo, wv] = \min \{y_{wv} \cdot o_{wv}\}, \; \forall wv \in \{\neg \boldsymbol{S}\} \tag{5.3}$$

Algorithm 5.1 lists OLLAWV's pseudocode and Figure 5.1 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 its 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:

$$\min_{\boldsymbol{w} \in \mathcal{H}_o \times \mathbb{R}} R \;=\; \frac{1}{2}||\boldsymbol{w}||^2 + C \sum_{i=1}^{n} \max \{0, 1 - y_i o_{(w)}(\boldsymbol{x}_i)\}, \tag{5.4}$$

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 5.4 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:

$$\frac{\partial R}{\partial \boldsymbol{w}} = \begin{cases} \boldsymbol{w} - C \sum_{i=1}^{n} y_i \boldsymbol{x_i} & y_i o_i < 1 \\ \boldsymbol{w} & \text{otherwise.} \end{cases} \qquad (5.5)$$

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 [107] 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 5.5 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 [149], 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 5.4, where $\phi(\cdot)$ is a mapping from feature space to Hilbert space [151]. On the basis of the Representer theorem, Equation 5.4 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:

$$\forall i : \alpha_i \phi(\boldsymbol{x_i}) \leftarrow \alpha_i \phi(\boldsymbol{x_i}) + \eta \begin{cases} (Cy_i\phi(\boldsymbol{x_i}) - \alpha_i\phi(\boldsymbol{x_i})) & y_io_i < 1 \\ (-\alpha_i\phi(\boldsymbol{x_i})) & \text{otherwise} \end{cases}$$

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

The case when the worst violating sample is correctly classified, i.e. $y_io_i \geq 1$, 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:

$$\forall i : \alpha_i \leftarrow \alpha_i + \eta(Cy_i - \alpha_i). \tag{5.6}$$

If the bias term b is included in Equation 5.4, its stochastic update is as follows:

$$\forall i : b \leftarrow b + \eta \frac{Cy_i}{n}. \tag{5.7}$$

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 $\Lambda = \eta Cy_i$ and $P = \eta\alpha_i$ be the update parameters for OLLAWV, and the $\boldsymbol{\alpha}$ update can be expressed as: $\alpha_i \leftarrow \alpha_i + (\Lambda - P)$. Note that Λ is the update resulting from the loss function and P is derived from the regularizer term in Equation 5.4. In the case of OLLAWV, $P = 0$ 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 5.1), from which a worst-violator is found, changes per iteration based on the influence of the support-vectors that have been previously updated.

Algorithm 5.1 OnLine Learning Algorithm using Worst-Violators (OLLAWV)

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

1: $\boldsymbol{\alpha} \leftarrow \boldsymbol{0}$, $b \leftarrow 0$, $\boldsymbol{S} \leftarrow \boldsymbol{0}$ ▷ Initialize OLLAWV model parameters
2: $\boldsymbol{o} \leftarrow \boldsymbol{0}$, $t \leftarrow 0$ ▷ Initialize the output vector and iteration counter
3: $wv \leftarrow 0$, $yo \leftarrow y_{wv} * \boldsymbol{o}_{wv}$ ▷ Initialize hinge loss error and worst-violator index
4: **while** $yo < M$ **do**
5:     $t \leftarrow t + 1$
6:     $\eta \leftarrow 2/\sqrt{t}$ ▷ Learning rate
7:
8:     $\Lambda \leftarrow \eta * C * y_{wv}$ ▷ Calculate hinge loss update
9:     $B \leftarrow (\Lambda * \beta) / n$ ▷ Calculate bias update
10:     $\boldsymbol{o} \leftarrow \boldsymbol{o} + \Lambda * \mathcal{K}\left(\boldsymbol{x}_{\neg \boldsymbol{S}}, \boldsymbol{x}_{wv}, \gamma\right) + B$ ▷ Update output vector
11:     $\boldsymbol{\alpha}_{wv} \leftarrow \boldsymbol{\alpha}_{wv} + \Lambda$ ▷ Update worst-violator's alpha value
12:     $b \leftarrow b + B$ ▷ Update bias term
13:
14:     $\boldsymbol{S}_t \leftarrow wv$ ▷ Save index of worst-violator
15:     $[yo, wv] \leftarrow \min_{wv \in \{\neg \boldsymbol{S}\}} \{y_{wv} \cdot o_{wv}\}$ ▷ Find the worst-violator
16: **end while**

Fig. 5.1.: 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.

Fig. 5.2.: 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.

From Equation 2.7, the output vector update becomes the following:

$$\boldsymbol{o} \leftarrow \boldsymbol{o} + \Lambda * \mathcal{K}\left(\boldsymbol{x}_{\neg \boldsymbol{S}},\, \boldsymbol{x}_{wv}, \gamma\right) + B, \tag{5.8}$$

where $\mathcal{K}(\cdot)$ is the Gaussian radial basis function (RBF) kernel, $\gamma \in \mathbb{R}$ is its 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 5.8, $\beta \in \{0, 1\}$ indicates whether the bias term is to be used. If b is not a part of the model, it should be omitted from Equations 2.7 and 5.8 by setting $\beta = 0$, otherwise $\beta = 1$.

OLLAWV is an SGD method 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 SGD 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 5.2 shows the decision boundaries achieved by OLLAWV versus SMO (implemented within LIBSVM) and Bayes for toy datasets.

5.3 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 5.3.1, compares OLLAWV to two other SVM kernel methods, and the second compares OLLAWV to 5 non-SVM methods, shown in Section 5.3.3. In each section, the experimental setups are first described and the state-of-the-art methods are listed. The results and statistical analysis 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, run time, and accuracy. The supplemental experimental study in 5.3.3 was conducted to emphasize the better performance of OLLAWV against non-SVM algorithms.

Table 5.2 presents a summary of the 23 datasets used throughout the experiments, where the number of attributes, classes, and samples are shown. They are divided into three groups: *small*, *medium* and *large*. The datasets were acquired from the UCI Machine Learning repository[1], and the LIBSVM[2] and LibCVM[3] sites [8, 47, 167].

---

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

[2]`https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`

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

Table 5.2.: Classification 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	4
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

5.3.1 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) [159] and *Non-Negative Iterative Single Data Algorithm* (NNISDA) [198]. These algorithms were chosen because they have shown considerable performance in run time, memory consumption, and accuracy against the popular and widely used LIBSVM and LibCVM packages. In [159], 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 [198], and showed an added improvement in run time 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 machine 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 objectively evaluate the models' performances and tune hyperparameters. 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 hyperparameters 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 hyperparameters, 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 hyperparameters. 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 hyperparameters were chosen from the following 8×8 possible combinations, shown in Equations (5.9a) and (5.9b), and were also used for the competing SVM methods.

$$C \in \{4^n\}, \quad n = \{-2, \dots, 5\} \tag{5.9a}$$

$$\gamma \in \{4^n\}, \quad n = \{-5, \dots, 2\} \tag{5.9b}$$

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

$$\mathcal{K}(\boldsymbol{x_i}, \boldsymbol{x_j}) = e^{-\gamma||\boldsymbol{x_i} - \boldsymbol{x_j}||^2}. \tag{5.10}$$

---

[4]https://github.com/strackr/gsvm

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.

5.3.2 SVM Comparison Results and Statistical Analysis

The classification performance was measured using the following metrics: accuracy, run time, and the percentage of support vectors (size of the model). Table 5.3 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 [57]. 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 5.3, 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 run time 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 5.3, Figures 5.3, 5.4, and 5.5 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 5.3 indicate that OLLAWV outperforms NNISDA and MNSVM in terms of accuracy, run time, and model complexity. Although the differences in accuracy between the methods is not very large, on average, OLLAWV is about 2 times faster than

Table 5.3.: Comparison of OLLAWV vs. NNISDA and MNSVM

Dataset	Accuracy (%)			Run Time (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
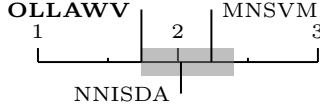
Fig. 5.3.: Bonferroni-Dunn test for Accuracy
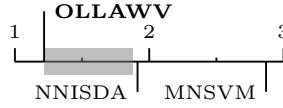
Fig. 5.4.: Bonferroni-Dunn test for Run Time
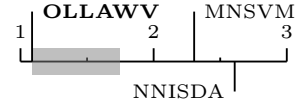
Fig. 5.5.: Bonferroni-Dunn test for % Support Vectors

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.
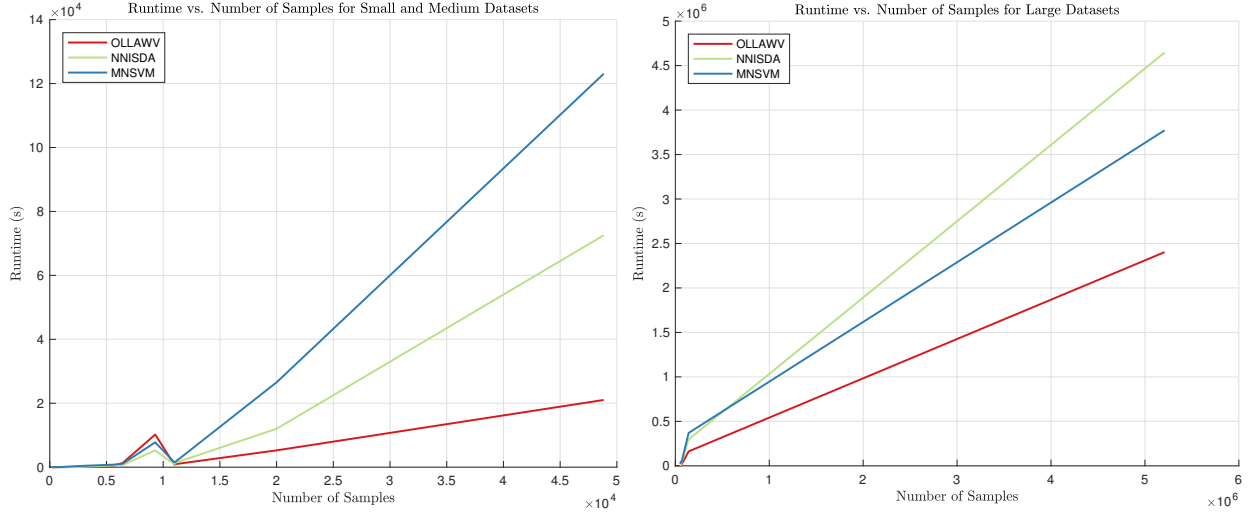
Fig. 5.6.: Run time in seconds versus the number of samples, divided into two groups: small & medium (left) versus large (right). Note OLLAWV's gradual increase in run time as the number of samples increases compared to NNISDA and MNSVM's steeper change. In almost all cases, OLLAWV displays superior run time over state-of-the-art. Run time depends upon many characteristics: dimensionality, class-overlapping, complexity of the separation boundary, number of classes, as well as the number of support vectors, which partly explains the tiny bump in the left figure.
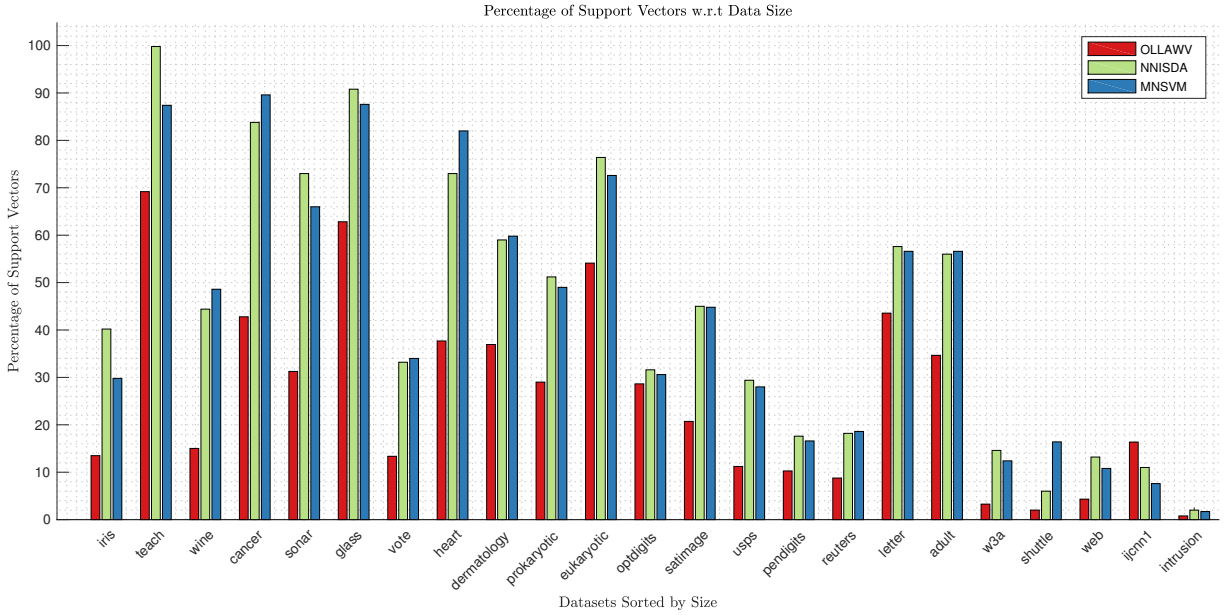
Fig. 5.7.: Size of the model given as percentage of support vectors with respect to the number of samples versus the number of samples. Note that OLLAWVs percentage of support vectors is always smaller (except in one case) than NNISDA's and MNSVM's ones.

Figures 5.3, 5.4, and 5.5 show the results of the statistical analysis for accuracy, run time, and percentage of support vectors. Figures 5.4 and 5.5 show that OLLAWV is statistically significantly better than MNSVM and NNISDA for run time and percentage of support vectors (model size). At the same time, Figure 4.4 emphasizes what was mentioned earlier: OLLAWV is shown to speed up the learning process without sacrificing model accuracy against the state-of-the-art methods used.

Figure 5.6 plots the correlation of OLLAWV, NNISDA, and MNSVM's run time 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 its gradual increase in run time as the number of samples increases in comparison to NNISDA and MNSVM. Figure 5.7 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 run time results.

5.3.3 Non-SVM Experimental Setup

The supplemental experimental setup was designed to compare the performance of the proposed OLLAWV against the following 5 popular non-SVM algorithms: *k-Nearest Neighbors (k-NN)*, *J48*, *JRip*, *Naïve Bayes*, and *Logistic*. These methods have been implemented within the Weka framework [66]. The experiments were performed under the same nested 5-fold cross-validation framework as the SVM experimental study in Section 5.3.1. The following hyperparameters, shown in Table 5.4, were used for the non-SVM algorithms.

Table 5.4.: Non-SVM Algorithm Hyperparameters

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: $\left\{1\mathrm{e}^{-7}, 1\mathrm{e}^{-8}, 1\mathrm{e}^{-9}\right\}$

Table 5.5.: Accuracy (%) for Non-SVM Methods vs. OLLAWV

Dataset	OLLAWV	k-NN	J48	JRip	Naïve Bayes	Logistic
small datasets						
iris	**97.33 ± 1.49**	96.00 ± 3.65	94.00 ± 2.79	90.67 ± 4.35	96.00 ± 2.79	97.33 ± 2.79
teach	52.32 ± 3.46	**59.64 ± 2.89**	49.72 ± 7.58	56.75 ± 9.60	53.75 ± 6.46	51.77 ± 6.68
wine	**98.87 ± 1.54**	97.73 ± 3.72	90.43 ± 5.83	93.24 ± 3.27	96.60 ± 3.14	96.05 ± 2.58
cancer	**80.36 ± 5.80**	77.32 ± 6.93	73.81 ± 8.57	73.78 ± 5.81	67.73 ± 5.07	77.32 ± 7.78
sonar	**92.32 ± 3.11**	88.99 ± 4.59	76.16 ± 10.6	75.18 ± 6.77	73.69 ± 7.65	75.18 ± 7.31
glass	**72.41 ± 2.28**	67.73 ± 5.91	65.06 ± 5.51	65.59 ± 9.66	49.46 ± 5.19	62.04 ± 5.75
vote	**96.54 ± 1.87**	92.26 ± 3.19	95.70 ± 2.12	96.54 ± 2.45	92.24 ± 3.24	93.54 ± 2.59
heart	82.22 ± 2.93	79.63 ± 5.71	78.52 ± 2.81	80.74 ± 4.06	**84.44 ± 4.46**	83.33 ± 3.93
dermatology	**97.82 ± 0.05**	96.18 ± 1.78	94.52 ± 2.21	91.27 ± 5.08	97.28 ± 1.64	96.98 ± 2.28
prokaryotic	**88.96 ± 2.14**	87.96 ± 3.01	78.54 ± 1.62	79.13 ± 2.78	62.38 ± 3.54	87.57 ± 2.56
eukaryotic	77.38 ± 1.96	**81.42 ± 2.06**	65.27 ± 2.92	66.42 ± 3.47	39.27 ± 3.43	69.55 ± 1.34
medium datasets						
optdigits	**99.11 ± 0.38**	98.74 ± 0.39	90.87 ± 1.09	91.28 ± 0.40	92.42 ± 0.75	95.05 ± 0.91
satimage	**91.66 ± 0.80**	90.38 ± 0.72	85.64 ± 1.21	85.33 ± 0.77	85.41 ± 0.92	88.14 ± 1.11
usps	**97.49 ± 0.22**	97.04 ± 0.47	88.73 ± 0.46	89.20 ± 1.00	79.45 ± 0.59	91.88 ± 0.65
pendigits	**99.56 ± 0.12**	99.33 ± 0.17	96.24 ± 0.31	96.34 ± 0.41	88.34 ± 0.65	95.59 ± 0.18
reuters	**98.03 ± 0.22**	97.15 ± 0.43	96.90 ± 0.32	97.18 ± 0.44	93.52 ± 0.02	69.54 ± 0.28
letter	**96.99 ± 0.21**	95.71 ± 0.19	87.34 ± 0.68	87.02 ± 0.66	74.12 ± 0.97	77.45 ± 0.16
large datasets						
adult	**84.75 ± 0.26**	83.85 ± 0.28	84.38 ± 0.28	83.73 ± 0.17	80.57 ± 0.09	82.46 ± 0.14
w3a	**98.86 ± 0.04**	98.60 ± 0.06	98.71 ± 0.05	98.41 ± 0.10	96.71 ± 0.20	98.61 ± 0.12
shuttle	99.77 ± 0.03	99.93 ± 0.03	**99.97 ± 0.02**	99.96 ± 0.02	98.57 ± 0.24	96.83 ± 0.12
web	**98.94 ± 0.05**	98.89 ± 0.06	98.79 ± 0.09	98.50 ± 0.13	96.71 ± 0.21	98.70 ± 0.08
ijcnn1	98.31 ± 0.07	**98.48 ± 0.04**	98.40 ± 0.09	98.11 ± 0.10	90.69 ± 0.26	92.29 ± 0.16
intrusion	**99.77 ± 0.02**	88.20 ± 1.06	58.01 ± 26.6	87.66 ± 3.79	49.75 ± 30.7	65.15 ± 15.7
Average	**91.29 ± 1.26**	90.05 ± 2.06	84.60 ± 3.64	86.18 ± 2.84	79.96 ± 3.58	84.45 ± 2.83
Ranks	**1.500**	2.500	4.041	3.9583	5.0625	3.9375
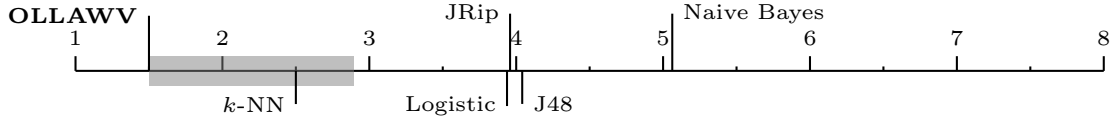
Fig. 5.8.: Bonferroni-Dunn test for Accuracy

5.3.4 Non-SVM Results and Statistical Analysis

Table 5.5 displays the accuracy results for all algorithms. The table also shows the standard deviation for accuracy per outer fold, the average values across all datasets, and the algorithm ranks. As the results indicate, OLLAWV outperforms all other methods. Figure 5.9 displays the average accuracy results for OLLAWV and the non-SVM methods across 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 5.8 shows the critical distance bar (which is 1.391), and indicates that all other algorithms perform statistically worse than OLLAWV, except for k-NN.

Fig. 5.9.: Mean accuracy over all datasets for OLLAWV and the 5 non-SVM competing methods.

5.4 Conclusions

This chapter 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 its implicit ability of finding support vectors and its 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, run time, 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 (k-NN), J48, JRip, Naïve Bayes, and Logistic). The proposal, OLLAWV, performs statistically better in terms of run time and model size across all 23 evaluated benchmark datasets, without compromising accuracy. The research outcomes of this chapter have been published in [131].

CHAPTER 6

OLLAWV FOR BATCHED DATA STREAMS

Due to advances in hardware/software technologies and information systems, data generation and automated processing has become fast, voluminous, and continuous; usually referred to as streams of data. Nowadays, data streams are ubiquitous and notoriously difficult to store, analyze, visualize, and learn from. Most traditional machine learning techniques need to be adapted to accommodate the streamed, also known as online, environment due to underlying resource constraints, i.e. memory consumption and run time, as well as the possibility of concept drift [82]. This chapter presents a novel implementation and experimental environment for two online support vector machines (SVMs) in the batched data stream setting. Unlike other existing methods, these two stochastic, i.e. online, algorithms were chosen because their characteristics are a natural remedy for the time and memory constraints that come with the data stream problem. The first algorithm's low memory complexity deals with the memory constraints, and the second methods' fast run time and self-stopping capability remedies the time constraint. The results for the latter, *OnLine Learning Algorithm using Worst-Violators* (OLLAWV), showed a superior performance to the former, *OnLine Learning Algorithm - List 2* (OLLA-L2). OLLAWV was then compared to 12 popular data stream algorithms against 24 datasets and stream generators. The results and statistical analysis showed OLLAWV's better performance, making it suitable for streamed learning.

6.1 Data Stream Classification Background

This section first describes the notation that will be used throughout the chapter, and then formally describes the data stream classification problem along with relevant popular algorithms used within this paradigm.

Table 6.1.: Summary of Data Stream Notation

Definition	Notation
Number of Samples	n
Number of Input Attributes	d
Time Step	t
Input Space at time t	$\boldsymbol{X}_t \in \mathbb{R}^{n \times d}$
Labels at time t	$\boldsymbol{Y}_t \in \mathbb{Z}^n$
Sample i at time t	$\boldsymbol{x}_t^i = (x_t^{i1}, \ldots, x_t^{id}) \in \boldsymbol{X}_t, \forall i \in \{1, \ldots, n\}$
Sample Label i at time t	$y_t^i \in \boldsymbol{Y}_t, \forall i \in \{1, \ldots, n\}$
Data Stream	$\mathcal{D} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_t, y_t), \ldots\}$
Set of Samples Provided to the Algorithm	$\mathcal{S}_t = \left\{ \left(\boldsymbol{x}_t^1, y_t^1\right), \ldots, (\boldsymbol{x}_t^n, y_t^n) \right\} = \{(\boldsymbol{X}_t, \boldsymbol{Y}_t)\} \in \mathcal{D}$

6.1.1   Notation

Let $\mathcal{D}$ be a data stream which provides a sequence of samples $\mathcal{S}_t = \{(\boldsymbol{x}_t, y_t)\}^n$ that are provided from an unknown probability distribution at some arbitrary time t, where n is the number of tuples. The sub-script, time t, is used to explicitly assert the data's possible time-varying nature. $\boldsymbol{x}_t \in \mathbb{R}^d$ represents a feature vector, and $y_t \in \mathbb{Z}$ is its discrete class label (in the non-binary classification case), both at time time t. The data may arrive to the algorithm in an online manner, i.e. one sample at a time: $\mathcal{S}_t = \{(\boldsymbol{x}_t, y_t)\}$, or in a batch setting: $\mathcal{S}_t = \{(\boldsymbol{x}_t^1, y_t^1), \ldots, (\boldsymbol{x}_t^n, y_t^n)\}$. The single-sample setting occurs when $n = 1$. Table 6.1 summarizes the notation used throughout the chapter.

6.1.2   Data Stream Classification Methods

In the context of supervised data stream classification, the goal is to learn the relationship between a set of attributes and a class label from continuous and rapidly arriving data. Examples of real-world data stream applications range between network analysis, financial data prediction, traffic control, sensor measurement, GPS tracking, as well as many others [17, 77, 121, 123]. A data stream, $\mathcal{D}$, produces a rapid sequence of data, $\mathcal{S}_t$, either in *batches* or in an *online* format, from an unknown distribution, over time t that is possibly infinite in length and cannot be accessed more than once [190]. In real-world scenarios, $\mathcal{S}_t$

is only available for learning at the time it is first presented to the algorithm [18]. This requires machine learning algorithms to be robust, alert, and perform classification in a resource adaptive way. The stream would need to be classified on demand, since there is no way to control the arrival of test samples.

Typically, there are two main core categories of data streams: *stationary* streams, where the samples are drawn from a fixed, unknown probability distribution, and *non-stationary* streams, where data distribution can change over time. In the non-stationary case, the class labels and/or attribute distributions can evolve, abruptly or gradually. This phenomenon is known as *concept drift* [82]. These drifts are only apparent in incoming samples, which will eventually cause the classifiers performance to deteriorate, due to being trained on outdated training data. Besides gradual and abrupt (i.e. sudden) changes, concept drifts can also be described by their continuity, predictability, and recurrence [121, 177]. Stationary stream classifiers lack any mechanism to adapt quickly when a drift occurs. Due to these complexities, in both the stationary and non-stationary case, fast, robust, and adaptive algorithms need to be designed for the data stream environment.

There are two main approaches for data stream learning problems in non-stationary environments: *active* and *passive*. To accommodate possible changes in the data stream, as well as ensure prompt and robust classification, passive approaches, also known as *trigger*, perform continuous updates of the model whenever new data arrive. The active approach relies on a *change-detection* mechanism which triggers an update to remedy or react to the change encountered [120]. Both approaches attempt to maintain an updated model at all times, however, passive approaches can avoid potential errors that might arise by a faulty change-detection mechanism (false alarms or failures to detect changes) [59]. Methods that fall under the passive approach are typically divided into two main categories: those that focus on updating a single classifier (*Single Classifier Models*), and those that add/remove/modify components of ensemble models (*Ensemble Classifier Models*). Single classifier models usually provide a lower computational cost, in comparison to ensemble models, making them a

more attractive and simple solution for mining ultra-large data streams. However, ensemble methods are very popular due to their natural way of handling drifting streams [121].

One of the earliest passive methods, specifically designed for the streaming environment, is the popular *Very Fast Decision Tree* (VFDT) [61], or *Hoeffding Tree.* It essentially sub-samples data to achieve scalability when constructing the tree. The main idea behind this approach is to determine a representative sample of sufficient size for constructing the tree, as though it were constructed on the entire dataset. It was transformed to accommodate the data stream format by building the tree incrementally. In order to accommodate concept-drifting streams, the method was extended in [97] and is called *Concept-adapting Very Fast Decision Tree* (CVFDT). It was also extended to handle multiple sub-trees at each node. Gama et. al. [79] also modified the VFDT method to handle continuous and drifting data by applying Bayes classifiers at leaf nodes [1]. More recent adaptations of VFDTs are the *Hoeffding Option Tree* (HOT) and *Adapting Hoeffding Option Tree* (AdaHOT) [137]. HOT enables each training sample to update a set of option nodes, instead of a single leaf, and AdaHOT incorporates Naïve Bayes classification at the leaves. One of the major disadvantages of the VFDT family is that they are naturally suited for categorical data. Numerical data would need to be discretized; a task that is usually done offline. It is possible to test all possible splits, however, when dealing with numerical data, the number of splits may be very large.

Rule-based methods have also been adapted for the data stream environment because of their high level of interpretability. One popular example of a rule-based method is the *Very Fast Decision Rules* (VFDR) [80] algorithm. This approach uses Naïve Bayes to help exploit information available in each rule. Another recent example of rule-based methods for concept-drifting data streams is a parallelized differential evolutionary algorithm for learning classification rules [41]. Rule classifier's major disadvantage lies when data evolves significantly because of their inability to adapt quickly [59].

Ensemble classifiers are popular approaches for learning in streamed environments and provide some advantages: they tend to have more flexibility in easily incorporating newly presented data into a model by adding new members to the ensemble. They also obtain a natural way of "*forgetting*" irrelevant knowledge by removing members from the ensemble [59]. The two most common ensemble methods are bagging and boosting [133].

One well known, online approach that uses sample and model weighting/selection to obtain the best accuracy over different sections of a drifting stream is the *Dynamic Weighted Majority* (DWM) [118] algorithm. The main idea is to choose the best weighted classifier that performs well for an unseen sample, or portion of the training stream, while having the ability to add/remove classifiers to accelerate reacting to a possible drift. In [119], an ensemble approach with a self-adapting confidence level for reacting to drifting streams is introduced. Instead of a traditional voting, the dynamic threshold allows for an additional functionality for ensemble classifiers: an option to abstain from contributing to the final classification model.

Another ensemble method is the *Social Adaptive Ensemble 2* (SAE2) [87], an improved version of SAE [86] classifier. The main difference between the two is that SAE2 implements a more scalable adaptation method. A relatively new online ensemble is the *Anticipative Dynmic Adaptation to Concept Changes* (ADACC) [99] algorithm, which recognizes concepts from incoming samples and optimizes control over the incremental classifiers. The main disadvantages of such ensemble approaches stem from their high computational and spatial complexity, as well as their low inter-classifier interpretability.

Unlike the previously mentioned ensemble approaches, the *Learn++ for Non-Stationary Environments* (Learn++.NSE) [67] method, which belongs to the Learn++ family of algorithms [139], operates in batch mode and has the ability to learn from environments that exhibit constant, variable, or cyclical drifts. It uses a time-adjusted loss function to favor newly trained classifiers, as well as classifiers that have performed will in the past (in case of a recurring concept).

k-Nearest neighbor (*k*-NN) methods comprise another set of algorithms that have been adapted to handle non-stationary streams of data, however, they use a *change-detection* mechanism. Unlike the VFDTs and Naïve Bayes classifiers, *k*-NN cannot learn from a stream without discarding data. These methods maintain a dynamic sample, or *sliding window*, of the data stream. Finding this representative sample of data is referred to as *reservoir sampling* [175], a technique used by many streaming algorithms [59]. Basic implementations of *k*-NN maintain a sliding window of fixed sizes because storing all samples in a stream is not feasible due to memory and processing constraints. The class label is then assigned by the majority label present among the *k*-nearest neighbors of the unknown test sample.

An example of such windowing methods is the *Probabilistic Adaptive Window* (PAW) algorithm, used with *k*-NN in [16]. Rather than storing all samples, or a limited sized window, PAW keeps a dynamically sized window with a logarithmic number of samples, storing those with higher probability than the rest. Another example of windowing techniques is that of the *ADaptive WINdowing* (ADWIN) algorithm [15], which increases the size of the window when no change is apparent, and shrinks it when there is a drift. The main disadvantage with these change detection techniques is their difficulty to set a threshold for when a change occurs; too low of a threshold value would cause many false positives, and too large a threshold would cause many false negatives.

Other widely used examples of change-detection techniques include the *Drift Detection Method* (DDM) [83] and its successor *Early Drift Detection Method* (EDDM) [11]. DDM estimates the classifier error, which must decrease as more training samples arrive. However, if the classification error increases above some threshold, a warning signal is generated and new incoming samples are then stored in a window. If the error later falls below the threshold, the warning is considered a false alarm and the window is discarded. If the error rate increases, the classifier is then rebuilt on the samples stored in the window. EDDM is similar to its predecessor, however offers an improvement for detecting gradual drifts by proposing a novel method for comparing distances of error rates.

Support vector machine (SVM) methods have infrequently been used for data stream classification due to challenges that stem from incrementally updating the SVM model as the number of samples increases. Some methods, such as *SVMLight* [101] and *SVMPerf* [102], have been proposed for speeding up SVM classification. It is worth noting that some of the earliest work in streamed SVM learning [114, 162] precedes the earliest work in stream decision tree learning. In fact, the contributions made by Klinkenberg and Joachims in [114], are one of the earliest window-based methods for detecting concept drift. Other significant contributions on incremental [24, 46, 141, 146, 155] and window-based [60, 76] support vector machines have also been investigated. However, the biggest issue with these algorithms is that most were not designed for streaming environments.

Adjusting SVMs for the data stream setting is a particularly critical task because of their generalization capabilities, scalability, and robustness against outliers. Most of the methods mentioned employ a quadratic programming formulation, with equal number of constraints as the number of data, making the problem computationally expensive for large amounts of data. In the kernelized case, the size of the kernel scales with the number of samples squared, an impractical solution in the case of data streams. Even when using decomposition methods, the disadvantage then stems from receiving an approximate solution; which may require the algorithm to make many passes over the data to achieve a reasonable level of convergence. Considering the event of a concept drift, the Karush-Kuhn-Tucker (KKT) optimality conditions must be maintained if/when adding or removing samples for the algorithm to succeed [1, 122]. The most interesting SVM methods for data stream learning are that of Cauwenberghs and Poggio [46], and Bordes et al. [24], because both utilize decremental unlearning methods which provide insights into the relationship between the geometry of the data and generalization capabilities.

These computational, spatial, and time complexities are what prompted researchers to investigate large-scale learning solutions using *Stochastic Gradient Descent* (SGD) before investigating how to deal with concept-drifts and their detection [90, 112, 145, 150, 151,

153]. Stochastic algorithms can be both the fastest and have superior generalization performances [32, 113] against batch methods. This (and the work done in [129, 109]) prompted the investigation of the novel *OnLine Learning Algorithm* (OLLA) [106] and its younger counterpart *OnLine Learning Algorithm using Worst-Violators* (OLLAWV) [131] under a stationary and non-stationary data stream environment. Due to their online, iterative nature, and that they have shown significant improvements over popular SVM solvers in terms of speed, memory consumption, and accuracy; OLLA and OLLAWV are a natural extension for SVMs to the data stream environment.

6.2   OnLine Learning Algorithms for Batched Data Streams

OLLA-L2 [106] and its successor OLLAWV [131] are two online, iterative learning algorithms that are implemented and investigated within the context of a batched, stationary and non-stationary, data stream environment. OLLA-L2 stems from the same core algorithm as OLLAWV, named *OnLine Learning Algorithm* (OLLA) [106]. It is a generic stochastic learning algorithm tailored and designed for various non-linear (i.e. kernel) classifiers in the primal domain. In this implementation of OLLA-L2, similar to that of OLLAWV described in Chapter 5, the online learning algorithm is derived and used for solving the primal L1-SVM cost function, given by Equation 2.5. The online nature of the core OLLA algorithm and its superior performance in terms of speed and accuracy against popular and widely used SVM solvers merited investigation in the data stream setting. The main objective is to design a novel base-line incremental support vector machine that is capable of handling large volumes of streamed data in a short amount of time, while maintaining a competitive accuracy. By beginning with this approach, in the event of a concept drift, the classifier would simply be able to rebuild itself, without violating any time constraints.

The algorithm that was first implemented and investigated under the data stream environment was OLLA-L2, listed in Algorithm 6.1. At every iteration, OLLA-L2 proceeds cyclically over the data, updating the model if the chosen sample, i, is a violator, i.e. has an

error $y_i o_i \leq 1$, where y_i and o_i are the sample's class label and corresponding model output, respectively. If the sample has no error ($y_i o_i > 1$), the model remains the same and no update is performed. Note, this error rule is that of the Hinge loss, shown in Equation 2.4. In the context of batched streams, the algorithm would cycle through the samples in each arriving batch. For a pure online scenario, the algorithm would proceed with its updates, as long as samples are provided by the stream.

The update parameters $\Lambda \in \mathbb{R}$ (lambda) and $P \in \mathbb{R}$ (rho), as seen in Algorithm 6.1, describe scalars which are used for updating the output vector $\boldsymbol{o} \in \mathbb{R}^n$, the weight vector in kernel feature space $\boldsymbol{\alpha} \in \mathbb{R}^n$, and the bias term $b \in \mathbb{R}$, if used. The input parameter $\beta \in \{0, 1\}$ describes whether the bias term will be used ($\beta = 1$) or not ($\beta = 0$). The update parameters for Λ and P for the L1-SVM with regularizer $||\boldsymbol{w}||^2$ are given by:

$$\Lambda = \eta C y_i \quad P = \eta \alpha_i$$

The advantage of using OLLA-L2's update procedure is most apparent in lines 8 and 15 in Algorithm 6.1. The model's $\boldsymbol{\alpha}$ parameter update is simply the sum of two scalars values: the previous value of $\boldsymbol{\alpha}$ and Λ. Then, to determine whether an update is even needed, the next sample's output value, o_i, is needed. It is obtained by using the scalar product of kernel values and weight vector $\boldsymbol{\alpha}$ for support vectors (stored in $\boldsymbol{S}$) only. Hence, the sparser the model, the more OLLA-L2 exhibits a speed-up [106].

In the batch setting, the quantity of iterations can be controlled by the parameter e, which represents the number of epochs, or cycles, over the data. As mentioned in the previous chapter, early stopping acts as a regularization technique [53]. It is worth noting that, if the number if epochs is set to be less than the number of samples, or in the pure online stream learning setting, where samples arrive one at a time with no cycling involved, the parameter P is always equal to 0 because $\boldsymbol{\alpha}$ is initialized to 0. $P > 0$ only if the algorithm cycles over samples that have already had their respective $\boldsymbol{\alpha}$ updated.

To try and maintain a sparse model, especially with evolving data streams, is a difficult

Algorithm 6.1 OnLine Learning Algorithm - List 2 (OLLA-L2)

Input: $\boldsymbol{X}, \boldsymbol{Y}, \beta, n, e$
Output: $\boldsymbol{\alpha}, b, \boldsymbol{S}$

1: $\boldsymbol{\alpha} \leftarrow \boldsymbol{0}, b \leftarrow 0, \boldsymbol{S} \leftarrow \boldsymbol{0}, \boldsymbol{o} \leftarrow \boldsymbol{0}, i \leftarrow 0$ ▷ Initialize model and algorithm parameters
2: **for** $t = 1, \ldots, n * e$ **do**
3:     $\eta \leftarrow 2/\sqrt{t}$ ▷ Learning rate in function of time
4:     **if** $y_i \, o_i \leq 1$ **then**   ▷ Check if current sample is a violator
5:        *Calculate* Λ *and* P   ▷ Calculate update parameters
6:        $\boldsymbol{S} \leftarrow [\boldsymbol{S} \cup i]$ ▷ Save index of current violator
7:        $\alpha_i \leftarrow \alpha_i + (\Lambda - P)$ ▷ Update violator's alpha value
8:        $b \leftarrow b + (\Lambda - P)\beta$ ▷ Update bias term
9:     **end if**
10:    $i \leftarrow i + 1$ ▷ Get new sample
11:    **if** $i = n$ **then**   ▷ If the sample index exceeds the number of samples
12:       $i = 0$ ▷ Reset sample index
13:    **end if**
14:    $o_i \leftarrow \boldsymbol{K}(\boldsymbol{x}_i, \boldsymbol{x_S}) \boldsymbol{\alpha_S} + b$ ▷ Calculate the new sample's output value
15: **end for**

task. In the context of batch data streams, building a new model from scratch might be too expensive, especially if there are many support vectors. The sample principle applies to the pure online scenario. Because of these issues, a simple change detection mechanism was designed and tested. OLLA-L2 was first updated to store the classification error, $\boldsymbol{\xi} = \sum_{i=1}^{n} y_i \, o_i$, for each previous batch trained on. If the mechanism is used, upon receiving a new batch of data, the algorithm will calculate the difference in error between subsequent batches and make a decision to either do nothing, update, or completely rebuild the model. The model update procedure involves firstly training the model on the new batch, then eliminating the support vectors with the smallest $\boldsymbol{\alpha}$ values, and then appending the newly added support vectors. Model variables from previous batches are scaled down by a 'learning rate' that set to be large for the older batches and smaller for the more recent ones. This gives preference to newly arriving samples in the case of a drift.

Although OLLA-L2 has many advantages, it also has complexities, some of which are shared by other stochastic solvers. One example would be how/what to set the algorithm's maximum number of iterations, i.e. the model's optimal point to stop training. Stopping the training process prematurely might cause a missed opportunity for finding all possible

Table 6.2.: Comparison of OLLAWV vs. OLLA-L2

Dataset	Accuracy (%)		Run Time (s)	
	OLLA-L2	OLLAWV	OLLA-L2	OLLAWV
RBFNoDrift	93.07	**94.21**	**0.0238**	0.0329
HyperplaneSlow	87.40	**90.09**	**0.0261**	0.0353
HyperplaneFaster	87.40	**89.51**	**0.0256**	0.0263
STAGGERGeneratorF1	**100.0**	100.0	0.0034	**0.0021**
HyperplaneFaster02	87.41	**89.49**	**0.0257**	0.0268
MixedGeneratorBT	92.45	**98.00**	**0.0108**	0.0205
MixedGeneratorBF	92.55	**98.03**	**0.0107**	0.0299
SineGeneratorF1BF	97.37	**97.79**	**0.0091**	0.0122
SineGeneratorF2BF	97.37	**97.79**	**0.0091**	0.0121
STAGGERGeneratorF1BF	**100.0**	100.0	0.0035	**0.0021**
STAGGERGeneratorF2BF	**100.0**	100.0	0.0039	**0.0022**
HyperplaneFasterAN0	87.40	**89.51**	**0.0255**	0.0263
HyperplaneFasterAN5	87.29	**89.29**	**0.0258**	0.0264
SEASuddenAN0	84.01	**87.80**	0.0494	**0.0208**
SEASuddenAN05	83.69	**87.53**	0.0494	**0.0284**
Average	91.83	93.94	**0.0201**	0.0203
Rank	1.90	**1.10**	**1.3333**	1.6667

support vectors. Stopping too late might cause the algorithm to diverge from its optimum point, while leading to a longer run time. Setting a stopping point for training would be especially difficult for the pure online scenario. This complexity inspired the work in Chapter 5 on OLLAWV. The novelty and advantage of OLLAWV resides in the way it iterates and updates the model. At each step, the algorithm chooses and updates the most violating sample, i.e. *worst-violator*. A worst-violator is defined as the sample that produces the largest error with respect to the current decision function. Rather than shuffling the data and cycling through each sample, as done in OLLA-L2, OLLAWV selects (without replacement) the most incorrectly classified sample and updates the model accordingly. This procedure essentially finds the model's support vectors while implicitly defining a stopping criterion: if there are no more violating samples, the algorithm terminates. These characteristics are especially useful in the context of learning from a batch streamed environment since the resulting model is speedily trained and is always sparse [131].

Preliminary experiments were conducted to test the capabilities of both approaches, and the results for percent accuracy and run time are listed in Table 6.2. The results show the better performance of OLLAWV in terms of accuracy. Both algorithms scale similarly

in terms of run time. The fast comparative run time of OLLAWV-L2 is dependent on its change detection mechanism, since models have the opportunity to be minimally updated, or even remain the same over new incoming batches of data. However, OLLAWV achieves a competitive run time without the use of a change detector. The main bottleneck in terms of run time for OLLAWV is the kernel vector calculation needed for calculating the model output o, i.e. current decision function output. Only non-support vector output values are calculated per iteration because samples are only allowed to be updated once. Therefore, the number of kernel calculations decreases by 1 per iteration. This phenomenon, as well as early stopping are contributing factors to OLLAWV's competitive run time and model sparsity. These characteristics motivated OLLAWV's implementation and further experimentation in the context of batch data streams.

6.3 Experimental Study

This section presents the experimental setup of OLLAWV against 12 popular data stream algorithms on 24 different benchmark datasets and stream generators. The experimental setup is first described, the contemporary comparison methods are then listed, then a brief description of the datasets and stream generators is presented. The main purpose of this study is to investigate whether OLLAWV is capable of learning from streams efficiently, ensuring a strong base algorithm, before tackling streams with abrupt concept drift.

6.3.1 Experimental Environment

The experimental environment tests the difference in performance of OLLAWV against 12 contemporary algorithms, designed to handle stationary and non-stationary data streams. Some use explicit drift-detectors and others take a passive approach. The majority of the classifiers are without an explicit drift-detector because OLLAWV was implemented without one, ensuring a fair comparison. However, classifiers with a detector mechanisms were included for a more comprehensive understanding of how OLLAWV will compare.

Table 6.3.: Data Stream Algorithms Used in Experimental Study

Algorithm	Description
HOT [137]	Hoeffding Option Tree
AdaHOT [137]	Adaptive Hoeffding Option Tree
NB [103]	Naïve Bayes
k-NNPAW [16]	k-NN with Probabilistic Adaptive Windows
DDM [83]	Drift Detection Method with HOT
VFDR [80]	Very Fast Decision Rules
VFDR-NB [80]	VFDR with Naïve Bayes
SAE2 [87]	Social Adaptive Ensemble 2
Learn.NSE [67]	Learn++ for Non-Stationary Environments
DWM [118]	Dynamic Weighted Majority
DACC [99]	Dynamic Adaptation to Concept Changes
OCBoost [136]	Online Coordinate Boosting

A summary of the 12 competing algorithms used throughout the experimental study is given in Table 6.3. They are the following: *Hoeffding Option Tree* [137], *Adaptive Hoeffding Option Tree* [137], *Naïve Bayes* [103], *k-NN with Probabilistic Adaptive Windows* [16], *Drift Detection Method* [83], *Very Fast Decision Rules* [80], *VFDR with Naïve Bayes* [80], *Social Adaptive Ensemble 2* [87], *Learn++ for Non-Stationary Environments* [67], *Dynamic Weighted Majority* [118], *Dynamic Adaptation to Concept Changes* [99], and *Online Coordinate Boosting* [136]. These algorithms were chosen because they have shown considerable performance in learning in a stream environment, while also being readily available for use and reproducing their results through MOA, *Massive Online Analysis*[1] [17] framework. The experiments were run on a server with 2 Intel Xeon CPU E5-2690v4 with 28 cores (56 threads), 128 GB of memory, and CentOS 7.4. OLLAWV was also implemented in Java within MOA framework.

In the context of traditional batch learning, the most popular and unbiased method for estimating the performance of an algorithm is the *nested cross-validation* procedure. However, in the online learning setting which has computationally strict requirements and drifting concepts, the nested cross-validation procedure is not straightforwardly applicable.

---

[1]https://moa.cms.waikato.ac.nz/

Algorithm 6.2 Evaluate Interleaved Chunks

Input: $\mathcal{D} = \{\mathcal{S}_1, \ldots, \mathcal{S}_T\}$
Output: `evaluator`
1: `firstChunk` = TRUE
2: Initialize the `classifier` and `evaluator`
3: **while** $\mathcal{D} \neq \emptyset$ **do**
4:     From $\mathcal{D}$ generate new batch of instances, $\mathcal{S}_t$
5:     **if** `!firstChunk` **then**
6:         **for** $\forall (\boldsymbol{x}_t, y_t) \in \mathcal{S}_t$ **do**
7:             `predictions` $\leftarrow$ `classifier.classify(`$(\boldsymbol{x}_t, y_t)$`)`
8:             `evaluator.addResult(`$(\boldsymbol{x}_t, y_t)$`, predictions)`
9:         **end for**
10:     **else**
11:         `firstChunk` $\leftarrow$ FALSE
12:     **end if**
13:     `classifier.trainOnInstances(`$\mathcal{S}_t$`)`
14: **end while**

The approach taken for this experimental environment is the evaluation method named *Interleaved Test-Then-Train*, also known as *Prequential* [81] (predictive sequential evaluation), listed in Algorithm 6.2. It is used to describe the change in accuracy (or any other evaluation metric) of algorithms over time. It follows the online learning protocol: when a sample is received, the current model makes a prediction, and when the system receives the sample's true label, the classifier's loss can be computed. In other words, each sample can be used to test the model before it is used during training, and using this, the evaluation metric can be incrementally updated. Using this order, the algorithms are always being tested on unseen samples. This scheme ensures that no hold-out test set is needed, thus utilizing all available data. It also provides a smooth and unbiased plot of the evaluation metric over time, where each sample becomes increasingly less significant to the overall average.

Along with the prequential evaluation method, traditional k-fold cross-validation was used in the OLLAWV experiments for the purpose of model selection. Once the first batch is received by the algorithm, 5-fold cross-validation is performed to select the best representative parameters for the current batch, and once they are chosen, they will be used for the remaining batches. This rigorous procedure is computationally expensive, however, the

goal is to optimally and fairly find the best representative SVM parameters for the incoming stream. Each batch of samples is first normalized by linear transformation of the feature values to the range [0,1]. Then the training process, also involving cross-validation and prequential evaluation, are performed. To deal with multi-class classification problems, the one-vs-one, or pairwise, approach was used. The best hyperparameters were chosen from the following 6×7 possible combinations, shown in Equations (6.1a) and (6.1b).

$$C \in \{0.01, 0.1, 1.0, 10.0, 100.0, 1000.0\} \tag{6.1a}$$

$$\gamma \in \{0.01, 0.1, 0.5, 1.0, 2.0, 4.0, 16.0\} \tag{6.1b}$$

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

$$\mathcal{K}(\boldsymbol{x_i}, \boldsymbol{x_j}) = e^{-\gamma||\boldsymbol{x_i} - \boldsymbol{x_j}||^2}. \tag{6.2}$$

6.3.2 Static Datasets & Stream Generators

Table 6.4 presents a summary of the static datasets and the stream generators used throughout the experiments, where the number of attributes, classes, and samples are shown.

Table 6.4.: Base Streamed Datasets & Generators

Dataset	# Samples	# Attributes	# Classes
Static			
Shuttle	57,999	10	7
Census	299,284	42	2
CovType	581,012	55	7
Generators			
RandomRBFGenerator	1,000,000	10	2
LEDGenerator	1,000,000	2	10
HyperplaneGenerator	1,000,000	10	2
WaveformGenerator	1,000,000	40	3
STAGGERGenerator	1,000,000	3	2
MixedGenerator	1,000,000	4	2
SineGenerator	1,000,000	2	2
SEAGenerator	1,000,000	2	2

The static datasets were obtained from the UCI Machine Learning repository[2] [8] and were simulated as streams using a stream simulator from the MOA framework. The stream generators listed were also provided by the MOA [17] framework. Both the streamed-static data and the generators provided 1000-sample sized batches to the algorithms. Variations of the generators were also used to simulate the concept drift rate, as well as varying number of attributes, classes, and functions (pertaining to the functional generators). These variations are suffixed with the generator name in the results tables. For the purposes of this experimental study, the variations used on the generators were mild in terms of their drifting capabilities. This was done intentionally in order to test the performance of OLLAWV as a base-line data stream classifier. The following are descriptions of the types of base generators used throughout the experimental study:

- **Random RBF Generator**: This generator produces samples that form normally distributed hyperspheres surrounding randomly selected centroids, with varying densities. This generator also has the option to simulate data evolution, i.e. concept drift.

- **LED Generator**: This stream generator's goal is to predict the digit displayed on a seven-segment LED display, where each segment has a 10% chance of being inverted.

- **Rotating Hyperplane**: This generator produces samples of different classes that are separated by a hyperplane. Their orientation and position may be smoothly changed by modifying the size of the weights.

- **Waveform Generator**: This generator constructs three different types of waves which are combinations of two/three base waveforms. The goal is to find the final wave type.

- **STAGGER Concepts Generator**: This generator creates a concepts which are described by a collection of samples, where each sample is a Boolean function of attribute-valued pairs and is described by a disjunct of conjuncts.

---

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

- **MIXED Concepts Generator**: This stream generator has an abrupt concept drift, where the class label is selected based on some functional conditional, and is reversed after a context change.

- **Sine Concepts Generator**: This generator involves two relevant attributes, each having values uniformly distributed between [0,1], with an abrupt concept drift change. In the first context, all samples below a sine curve are classified as positive. However, after the context change, the class label is reversed.

- **SEA Concepts Generator**: This generator involves an abrupt concept drift change and three attributes, of which only two are relevant, ranging between [0, 10]. Samples belong to a certain class label if the sum of the relevant attributes is less than or equal to some threshold value, which varies based on the concept.

6.4 Results & Analysis

The classification performance was measured using two metrics: Accuracy and Cohen's kappa rate, and the results are shown in Tables 6.5 and 6.6, respectively. The metrics are calculated as shown in Equations 6.3a and 6.3b, where TP is the true positive value, TN is the true negative value, FP is the false positive value, FN is the false negative value, and $n' = TP + FP + TN + FN$. The training times of each algorithm are also given in Table 6.7 to analyze their scalability. All metrics were calculated using the prequential procedure.

$$\text{Accuracy} \qquad \frac{TP + TN}{n'} \qquad\qquad (6.3a)$$

$$\text{Cohen's Kappa Rate} \qquad \frac{n' - \dfrac{(TP + FN) * (TP + FP)}{n'}}{1 - \dfrac{(TP + FN) * (TP + FP)}{n'}} \qquad (6.3b)$$

In order to analyze the performances of the multiple models, non-parametric statistical tests are used to validate the experimental results obtained. The Iman-Davenport non-parametric test is run to investigate whether significant differences exist among the per-

formance of the algorithms by ranking them over the datasets used, using the Friedman test. The algorithm ranks for each metric in Equations (4.17) are presented in the last row of the results tables, and the lowest (best) rank value is typeset in bold. After the Iman-Davenport test indicates significant differences, the Bonferroni-Dunn post-hoc test [65] (for $\alpha = 0.05$) is then used to find where they occur between algorithms by assuming the classifiers' performances are different by at least some critical value. Below each result table, a figure highlighting the critical distance (in gray), from the best ranking algorithm to the rest, is shown. The algorithms to the right of the critical distance bar perform statistically significantly worse than the control algorithm, OLLAWV. Figures 6.1, 6.2, and 6.3 show the results of the Bonferroni-Dunn post-hoc procedure over the metrics in Equation (6.3).

6.4.1   Accuracy

Table 6.5 shows the accuracy results of the 13 algorithms over 24 data streams, along with their average and rank. Specifically, OLLAWV achieves the best accuracy over 14 of the 24 datasets, with a competitive average against k-NNPAW, AdaHOT, DDM, and HOT. Note that OLLAWV performs competitively with the algorithms that employ some form of change detection/adaptation, even on more sudden drifting streams, indicating its value as a base-line batch stream learning algorithm (without a detection/adaptation mechanism). This is most likely due to the method in which OLLAWV performs its model updates: by iteratively selecting and updating the worst-violating sample. If a gradual drift occurs when a new batch is received, there are bound to be misclassified samples, which will in turn be selected first by OLLAWV to be updated. However, for sudden drifts, OLLAWV would not be able to update and adjust the model in time; it would always be one batch behind. This is evident through OLLAWV's accuracy results achieved over the two variant SEA generators: SEASuddenAN0 and SEASuddenAN05.

Table 6.5.: Accuracy (%) for Data Stream Classifiers

| Dataset	OLLAWV	HOT	AdaHOT	NB	k-NNPAW	DDM	VFDR	VFDR-NB	SAE2	LearnNSE	DWM	DACC	OCBoost
CovType	**90.28**	85.34	86.22	60.04	87.89	59.56	60.32	75.58	76.21	69.97	71.96	61.70	71.29
Census	93.76	94.70	**94.74**	87.02	93.65	91.85	93.65	84.06	90.13	84.14	91.40	90.37	93.47
Shuttle	**99.67**	98.18	98.52	90.02	99.26	98.49	88.40	96.06	90.24	93.79	89.91	92.03	74.21
RBFNoDrift	**94.21**	92.94	92.96	71.99	93.75	92.48	77.53	81.71	89.16	70.28	70.43	65.01	92.08
LEDNoDrift	73.83	73.85	73.84	**73.94**	65.83	73.64	41.16	73.75	67.60	67.84	71.15	48.27	17.44
HyperplaneSlow	**90.09**	82.10	82.42	77.69	84.03	81.57	68.88	85.19	82.67	86.20	88.06	80.66	85.78
HyperplaneFaster	**89.51**	82.72	85.34	77.23	84.27	84.33	78.63	85.18	83.01	86.50	86.76	81.15	87.80
RBFGradualRecurring	98.41	94.63	94.44	58.33	**98.43**	93.47	60.08	86.16	88.48	72.87	74.96	61.91	49.62
RBFBlips	**99.07**	95.67	95.60	60.83	98.94	94.92	66.90	88.35	89.04	77.53	79.98	68.27	47.46
WaveformGenerator	83.94	82.99	**84.14**	80.41	80.13	83.61	63.88	75.84	80.18	80.19	78.39	73.59	55.05
STAGGERGeneratorF1	100.0	99.99	99.99	100.0	100.0	100.0	99.91	100.0	95.04	89.82	100.0	99.96	100.0
HyperplaneFaster02	**89.49**	82.77	85.36	77.25	84.27	87.64	78.89	85.11	83.04	86.51	86.76	81.23	87.59
RBFGradualRecurringv2	**97.18**	93.29	93.00	57.47	95.73	93.19	57.96	80.71	84.45	62.41	63.79	49.42	48.88
MixedGeneratorBT	98.00	99.11	**99.32**	91.93	97.67	99.11	83.12	93.28	93.16	90.91	91.19	89.16	98.98
MixedGeneratorBF	98.03	99.18	**99.36**	92.04	97.59	99.20	89.96	94.30	93.41	90.76	91.46	88.61	98.94
RandomRBFGeneratorC4A25	**99.12**	97.43	97.14	81.59	98.69	96.89	72.33	89.03	90.61	78.23	79.67	63.02	52.16
RandomRBFGeneratorC4A50	**99.74**	99.16	99.14	91.90	99.17	98.99	80.63	95.63	92.00	86.03	90.45	73.78	50.66
SineGeneratoF1BF	97.79	**99.75**	99.73	93.55	95.54	99.66	94.83	96.09	94.51	92.55	93.30	92.20	99.51
SineGeneratoF2BF	97.79	**99.75**	99.73	93.55	95.41	99.66	95.26	96.10	94.57	92.55	93.34	92.20	99.48
STAGGERGeneratorF1BF	100.0	99.99	99.99	100.0	100.0	100.0	99.91	100.0	95.04	89.82	100.0	99.96	100.0
STAGGERGeneratorF2BF	100.0	99.98	99.98	100.0	100.0	99.98	99.87	100.0	95.02	44.41	100.0	100.0	0.61
HyperplaneFasterAN5	**89.29**	82.69	85.24	87.38	84.19	88.74	79.14	84.89	82.92	86.41	86.67	81.12	87.38
SEASuddenAN0	87.80	84.92	85.18	88.23	87.22	**88.97**	81.56	85.17	85.11	85.77	86.93	83.73	88.23
SEASuddenAN05	87.53	84.57	84.82	87.53	86.96	**88.28**	81.55	85.11	84.57	85.63	86.54	83.53	87.53
Average	**93.94**	91.90	92.34	82.50	92.03	91.43	78.93	88.21	87.51	81.30	85.55	79.20	73.92
Rank	**2.52**	5.42	4.54	8.19	4.8542	4.63	10.96	6.77	8.46	9.04	7.27	10.90	7.46
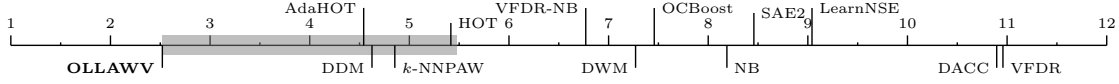
Fig. 6.1.: Bonferroni-Dunn test for Accuracy

Figure 6.1 shows the results for the statistical analysis on the accuracy results. The algorithms with rank higher than 5.48 (OLLAWV's rank + the Bonferroni-Dunn critical value), i.e. to the right of the gray bar in Figure 6.1, perform statistically worse than OLLAWV. The Bonferroni-Dunn test shows that there are no statistically significant differences in the accuracy results achieved by OLLAWV and AdaHOT, DDM, k-NNPAW, and HOT. This is expected firstly, since the AdaHOT and DDM methods utilize HOT as their base classifier and the HOT classifier performed competitively; and secondly, because of k-NNPAW's use of a probabilistic sliding window.

6.4.2 Cohen's Kappa Rate

Table 6.6 shows the Cohen's Kappa rate results obtained by the algorithms. These results complement the accuracy results achieved by the algorithms, where OLLAWV outperforms all compared classifiers on 13 out of the 24 datasets, and is competitive with the popular algorithms: AdaHOT, DDM, k-NNPAW, and HOT. OLLAWV's kappa values all

fall within the (50%, 100%] range, with the exception of the Census dataset, indicating that its merit as a classifier that agrees with the class distribution, and is not random. Concerning the Census dataset, the best result achieved was by AdaHOT, which obtained 37.39%. This indicates that all algorithms performed approximately randomly over the Census dataset. Classifiers OCBoost, DACC, Learn++.NSE, and VFDR all achieved the lowest rates on average. Note, for STAGGERGeneratorF2BF, Learn++.NSE obtained 0% indicating complete random classifier behavior, and OCBoost obtained −96.47 indicating a performance worse than the default-hypothesis. Figure 6.2 shows the results of the statistical analysis on the Cohen's Kappa rate results. They show that OLLAWV performs significantly better than 8 out of the 12 competing classifiers. These results also support OLLAWV's performance as a competitive base classifier and supplement the statistical analysis over the accuracy metric.

Table 6.6.: Cohen's Kappa Rate (%) for Data Stream Classifiers

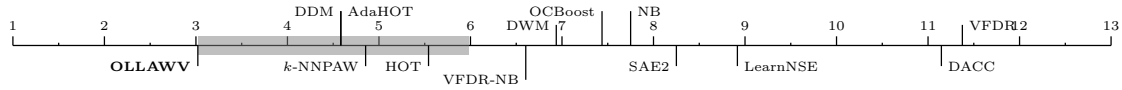| Dataset	OLLAWV	HOT	AdaHOT	NB	k-NNPAW	DDM	VFDR	VFDR-NB	SAE2	LearnNSE	DWM	DACC	OCBoost
CovType	**84.30**	76.44	77.69	40.16	80.43	34.11	31.47	61.16	62.98	50.63	55.53	36.64	48.24
Census	5.06	34.37	**37.39**	35.96	22.11	20.39	6.97	18.90	18.66	17.48	36.18	16.68	33.66
Shuttle	96.17	94.91	95.85	75.04	**97.92**	95.75	60.82	89.53	75.32	81.56	75.10	74.98	36.69
RBFNoDrift	**88.42**	85.88	85.92	43.98	87.49	84.96	55.07	63.41	78.32	40.57	40.84	30.02	84.17
LEDNoDrift	70.92	70.94	70.93	**71.05**	62.03	70.71	34.62	70.84	64.00	64.27	67.94	42.52	8.27
HyperplaneSlow	**80.18**	64.19	64.84	55.38	68.05	63.13	37.77	70.38	66.97	72.41	76.12	61.32	71.55
HyperplaneFaster	**79.02**	65.43	70.68	54.46	68.54	68.65	57.27	70.36	67.63	73.00	73.52	62.31	75.59
RBFGradualRecurring	97.88	92.82	92.58	44.28	**97.90**	91.27	46.66	81.51	84.61	63.73	66.54	49.00	31.46
RBFBlips	**98.74**	94.14	94.06	46.97	98.57	93.13	55.08	84.27	85.20	69.57	72.85	56.99	32.73
WaveformGenerator	75.91	74.48	**76.21**	70.62	70.19	75.41	45.82	63.76	70.28	70.29	67.59	60.39	32.62
STAGGERGeneratorF1	**100.0**	99.94	99.94	**100.0**	**100.0**	**100.0**	99.56	**100.0**	78.28	14.61	**100.0**	99.78	**100.0**
HyperplaneFaster02	**78.98**	65.55	70.72	54.50	68.55	75.28	57.77	70.23	67.68	73.02	73.51	62.46	75.17
RBFGradualRecurringv2	**96.23**	91.04	90.66	43.16	94.30	90.91	43.94	74.25	79.24	49.79	51.65	32.37	30.81
MixedGeneratorBT	95.99	98.21	**98.64**	83.87	95.33	98.21	66.24	86.56	86.32	81.81	82.39	78.31	97.96
MixedGeneratorBF	96.04	98.35	**98.72**	84.00	95.16	98.40	79.82	88.56	86.79	81.42	82.81	77.10	97.88
RandomRBFGeneratorC4A25	**98.82**	96.53	96.13	75.22	98.23	95.80	62.39	85.19	87.34	70.61	72.56	49.68	36.25
RandomRBFGeneratorC4A50	**99.65**	98.86	98.83	89.06	98.87	98.64	73.71	94.07	89.19	81.10	87.09	64.30	35.40
SineGeneratorF1BF	95.55	**99.50**	99.45	86.98	91.04	99.32	89.54	91.75	88.95	84.97	86.49	84.25	99.01
SineGeneratoF2BF	95.55	**99.49**	99.45	86.98	90.74	99.32	90.45	92.15	89.07	84.97	86.55	84.25	98.95
STAGGERGeneratorF1BF	**100.0**	99.94	99.94	**100.0**	**100.0**	**100.0**	99.56	**100.0**	78.28	14.61	**100.0**	99.78	**100.0**
STAGGERGeneratorF2BF	**100.0**	99.95	99.95	**100.0**	**100.0**	99.95	99.73	**100.0**	89.92	00.00	**100.0**	**100.0**	-96.47
HyperplaneFasterAN5	**78.58**	65.38	70.49	74.75	68.37	77.47	58.27	69.77	67.45	72.81	73.35	62.24	74.75
SEASuddenAN0	73.80	67.61	68.19	74.80	72.76	**76.39**	59.86	68.06	68.29	69.27	72.05	64.99	74.80
SEASuddenAN05	73.19	66.84	67.41	73.28	72.20	**74.92**	59.91	67.94	67.13	68.97	71.21	64.56	73.28
Average	**85.79**	83.37	84.36	69.36	83.28	82.59	61.35	77.61	74.91	60.48	73.83	63.12	56.37
Rank	**3.02**	5.54	4.58	7.75	4.85	4.58	11.38	6.60	8.25	8.92	6.94	11.15	7.44

Fig. 6.2.: Bonferroni-Dunn test for Cohen's Kappa rate

Table 6.7.: Training Time (seconds) for Data Stream Classifiers

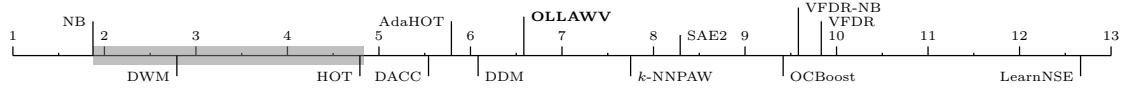| Dataset	OLLAWV	HOT	AdaHOT	NB	k-NNPAW	DDM	VFDR	VFDR-NB	SAE2	LearnNSE	DWM	DACC	OCBoost
CovType	0.0451	0.0765	0.0909	**0.0009**	0.0387	0.0577	0.0371	0.0445	0.1121	6.2806	0.0419	0.0357	0.0984
Census	0.0168	0.1040	0.1162	**0.0007**	0.0386	0.0345	0.0735	0.0458	0.0363	0.9368	0.0156	0.0175	0.0543
Shuttle	0.0103	0.0276	0.0298	**0.0004**	0.0386	0.0069	0.0108	0.0069	0.0217	0.2692	0.0104	0.0186	0.0581
RBFNoDrift	0.0329	0.0174	0.0247	**0.0002**	0.0383	0.0828	0.4267	0.3172	0.0491	3.4029	0.0104	0.0137	0.0402
LEDNoDrift	0.0697	0.0373	0.0649	**0.0003**	0.0389	0.0245	0.0097	0.0082	0.0694	5.4734	0.0175	0.0290	0.0547
HyperplaneSlow	0.0353	0.0094	0.0142	**0.0002**	0.0389	0.1405	0.1871	0.3006	0.0473	3.3734	0.0091	0.0142	0.0389
HyperplaneFaster	0.0263	0.0295	0.0416	**0.0002**	0.0384	0.1231	0.3016	0.3784	0.0424	3.2991	0.0098	0.0143	0.0376
RBFGradualRecurring	0.0456	0.0316	0.0369	**0.0003**	0.0389	0.0391	1.1284	0.9295	0.1004	11.7980	0.0335	0.0474	0.1070
RBFBlips	0.0396	0.0243	0.0299	**0.0003**	0.0384	0.0278	0.7587	0.8342	0.0948	11.9934	0.0342	0.0473	0.1050
WaveformGenerator	0.0394	0.0751	0.0965	**0.0006**	0.0390	0.1847	0.9670	0.9271	0.1715	17.9675	0.0511	0.0743	0.1318
STAGGERGeneratorF1	0.0021	0.0006	0.0007	**0.0001**	0.0389	0.0005	0.0013	0.0007	0.0018	0.5517	0.0003	0.0020	0.0113
HyperplaneFaster02	0.0268	0.0302	0.0465	**0.0002**	0.0388	0.0260	0.3192	0.3010	0.0456	3.2893	0.0095	0.0141	0.0372
RBFGradualRecurringv2	0.0545	0.0169	0.0203	**0.0003**	0.0383	0.0859	0.7622	0.7634	0.0552	12.0529	0.0367	0.0464	0.1053
MixedGeneratorBT	0.0205	0.0027	0.0026	**0.0001**	0.0336	0.0071	1.7640	1.5534	0.0086	0.9761	0.0024	0.0052	0.0204
MixedGeneratorBF	0.0299	0.0024	0.0024	**0.0001**	0.0334	0.0065	1.1035	1.0924	0.0093	0.8971	0.0024	0.0058	0.0209
RandomRBFGeneratorC4A25	0.0200	0.0441	0.0407	**0.0003**	0.0358	0.0730	1.7831	1.6927	0.1245	13.9876	0.0395	0.0719	0.1580
RandomRBFGeneratorC4A50	0.0157	0.0333	0.0337	**0.0007**	0.0350	0.0909	2.8802	2.5559	0.1917	27.8456	0.0752	0.1432	0.2974
SineGeneratoF1BF	0.0122	0.0056	0.0056	**0.0001**	0.0334	0.0073	1.9301	2.9169	0.0127	1.3768	0.0034	0.0071	0.0235
SineGeneratoF2BF	0.0121	0.0051	0.0053	**0.0001**	0.0314	0.0077	5.0598	5.9477	0.0117	1.3996	0.0027	0.0074	0.0236
STAGGERGeneratorF1BF	0.0021	0.0006	0.0006	**0.0001**	0.0323	0.0005	0.0006	0.0005	0.0015	0.6175	0.0003	0.0022	0.0143
STAGGERGeneratorF2BF	0.0022	0.0010	0.0009	**0.0001**	0.0340	0.0006	0.0005	0.0005	0.0017	0.6752	0.0003	0.0020	0.0139
HyperplaneFasterAN5	0.0264	0.0200	0.0200	0.0440	0.0269	0.0115	0.2298	0.2417	0.0312	2.5014	**0.0080**	0.0147	0.0740
SEASuddenAN0	0.0208	0.0051	0.0052	0.0268	0.0312	0.0099	0.0370	0.0373	0.0176	1.3545	**0.0032**	0.0059	0.0335
SEASuddenAN05	0.0284	0.0055	0.0055	0.0257	0.0309	0.0095	0.0321	0.0334	0.0167	1.3365	**0.0032**	0.0061	0.0326
Average	0.0264	0.0252	0.0307	**0.0043**	0.0359	0.0441	0.8252	0.8721	0.0531	5.5690	0.0175	0.0269	0.0663
Rank	6.5833	4.7917	5.7917	**1.8750**	7.7500	6.0833	9.8333	9.5833	8.2917	12.6667	2.7917	5.5417	9.4167

Fig. 6.3.: Bonferroni-Dunn test for Training Time

Table 6.8.: Testing Time (seconds) for Data Stream Classifiers

Dataset	OLLAWV	HOT	AdaHOT	NB	k-NNPAW	DDM	VFDR	VFDR-NB	SAE2	LearnNSE	DWM	DACC	OCBoost
CovType	0.0391	0.0256	0.0321	0.0089	1.3744	0.0041	**0.0010**	0.0304	0.0493	0.1896	0.0358	0.0170	0.0325
Census	0.0223	0.0082	0.0095	0.0033	2.2826	0.0014	**0.0012**	0.0082	0.0048	0.1052	0.0125	0.0076	0.0099
Shuttle	0.0072	0.0054	0.0060	0.0051	0.3265	0.0017	**0.0012**	0.0105	0.0092	0.1228	0.0096	0.0108	0.0176
RBFNoDrift	0.0372	0.0051	0.0065	0.0016	0.4792	**0.0014**	0.0196	0.0180	0.0168	0.0322	0.0084	0.0021	0.0102
LEDNoDrift	0.0927	0.0014	0.0021	0.0027	1.6070	0.0018	**0.0010**	0.0190	0.0302	0.0603	0.0148	0.0050	0.0105
HyperplaneSlow	0.0407	0.0029	0.0038	0.0016	0.5687	0.0020	**0.0004**	0.0048	0.0220	0.0814	0.0080	0.0037	0.0095
HyperplaneFaster	0.0273	0.0088	0.0113	0.0019	0.5691	**0.0016**	0.0020	0.0051	0.0164	0.0794	0.0085	0.0038	0.0083
RBFGradualRecurring	0.0436	0.0141	0.0154	0.0060	0.7791	**0.0040**	0.0067	0.0171	0.0485	0.2963	0.0301	0.0103	0.0448
RBFBlips	0.0370	0.0106	0.0127	0.0061	0.7407	**0.0032**	0.0039	0.0148	0.0464	0.5582	0.0312	0.0133	0.0457
WaveformGenerator	0.0604	0.0231	0.0291	0.0094	2.4650	0.0053	**0.0033**	0.0207	0.0671	0.3095	0.0454	0.0142	0.0400
STAGGERGeneratorF1	0.0018	**0.0002**	0.0003	**0.0002**	0.1834	**0.0002**	0.0004	0.0006	0.0004	0.2034	**0.0002**	0.0012	0.0005
HyperplaneFaster02	0.0272	0.0088	0.0118	0.0016	0.5671	**0.0012**	0.0019	0.0048	0.0174	0.0790	0.0081	0.0037	0.0085
RBFGradualRecurringv2	0.0551	0.0067	0.0078	0.0064	0.8919	0.0039	**0.0035**	0.0158	0.0259	0.0853	0.0322	0.0059	0.0452
MixedGeneratorBT	0.0202	0.0012	0.0012	0.0004	0.1348	**0.0003**	0.0008	0.0018	0.0029	0.0371	0.0018	0.0018	0.0028
MixedGeneratorBF	0.0330	0.0010	0.0011	0.0004	0.1372	**0.0003**	0.0017	0.0028	0.0032	0.0316	0.0019	0.0020	0.0030
RandomRBFGeneratorC4A25	0.0218	0.0153	0.0145	0.0072	0.6871	0.0030	**0.0027**	0.0158	0.0559	0.1909	0.0356	0.0135	0.0690
RandomRBFGeneratorC4A50	0.0204	0.0115	0.0117	0.0137	1.2234	0.0036	**0.0015**	0.0234	0.0848	1.0711	0.0687	0.0443	0.1373
SineGeneratorF1BF	0.0111	0.0025	0.0025	0.0006	0.1586	**0.0003**	**0.0003**	0.0016	0.0042	0.0938	0.0028	0.0027	0.0035
SineGeneratoF2BF	0.0111	0.0023	0.0025	0.0006	0.1500	**0.0003**	**0.0003**	0.0016	0.0041	0.0655	0.0022	0.0029	0.0036
STAGGERGeneratorF1BF	0.0018	**0.0002**	0.0003	**0.0002**	0.1150	**0.0002**	**0.0002**	0.0006	0.0004	0.1853	**0.0002**	0.0013	0.0007
STAGGERGeneratorF2BF	0.0019	0.0004	0.0004	0.0002	0.1232	0.0002	**0.0001**	0.0011	0.0004	0.2370	0.0002	0.0012	0.0011
HyperplaneFasterAN5	0.0290	0.0058	0.0059	0.0091	0.2602	**0.0009**	0.0016	0.0041	0.0129	0.0573	0.0068	0.0039	0.0153
SEASuddenAN0	0.0212	0.0009	0.0009	0.0044	0.1736	**0.0004**	0.0118	0.0129	0.0045	0.0326	0.0026	0.0021	0.0055
SEASuddenAN05	0.0325	0.0009	0.0009	0.0041	0.1673	**0.0004**	0.0106	0.0119	0.0044	0.0321	0.0025	0.0022	0.0053
Average	0.0290	0.0068	0.0079	0.0040	0.6736	**0.0017**	0.0032	0.0103	0.0222	0.1765	0.0154	0.0074	0.0221
Rank	10.375	4.7500	5.6250	3.3125	12.875	**1.6458**	2.8333	7.0417	8.9583	11.958	7.0417	5.7917	8.7917

Fig. 6.4.: Bonferroni-Dunn test for Testing Time

6.4.3 Training & Testing Time

Tables 6.7 and 6.8 show the training and testing times in seconds for each algorithm. The Naïve Bayes classifier has the fastest training time and is ranked first, while exhibiting fast testing times. This is to be expected since it is the simplest algorithm out of the 13 compared. However, its accuracy and Cohen's kappa rate results were far from competitive. The DWM and HOT algorithms were ranked next in terms of training time, however, DWM's accuracy and Cohen's kappa rate were also statistically significantly worse in comparison to the competing methods. Fast execution times are insignificant without complementing performance results, as in the case with the Naïve Bayes and VFDR.

Based on the previous metrics, the top performing algorithms were: OLLAWV, Ada-HOT, DDM, k-NNPAW, and HOT. In terms of training time, OLLAWV showed better scalability than AdaHOT, DDM, and k-NNPAW, despite having to rebuild the classifier with every received batch of data. Additionally, OLLAWV performs 5-fold cross-validation for hyperparameter selection, and builds multiple pairwise models for the multi-class streams; both of which as expensive procedures. Despite these traits, OLLAWV still obtains competitive training times. This is due to OLLAWV's early stopping capability.

In terms of testing time, OLLAWV performed notably better than k-NNPAW, which has the lowest ranking testing time. However, in comparison to the HOT, AdaHOT, DDM and DWM methods, OLLAWV's testing times are slower. This is yet another trait of building multiple pairwise models. The DDM method ranked first and had the fastest testing times due to its wrapper drift detection method over the HOT classifier. HOT as a single classifier was shown to have competitive testing times; therefore, in conjunction with a drift detector, its run time performance is expected to increase. However, despite the slower testing times exhibited by OLLAWV, it showed competitive performance concerning model training time.

124

6.5   Conclusions

This chapter proposed two novel support vector machines, OLLA-L2 and OLLAWV, for the batch data stream problem. The algorithms are online, i.e. stochastic, solvers which are naturally suited for operating in the data stream setting. The main aim of the study was to design a competitive base-line algorithm that satisfied the performance constraints posed by the data stream problem, without sacrificing classification accuracy. The interesting properties of OLLA-L2, in the context of data streams, are its efficient model update procedure, its unordered iterative nature, and its ability to produce sparse models. However, OLLA-L2 suffers from a lack of meaningful stopping criteria, which could lead to denser models. The key features of OLLAWV (OLLA-L2's successor) are its self-stopping condition and unique method of iterating over and selecting samples, as well as a distinct update procedure. These characteristics result in faster training times and sparser models. These algorithmic attributes are desirable for the data stream setting, where time and memory constraints play an important role in classifier efficacy. A preliminary experimental study was conducted to compare the performance of the two solutions, and the results showed the better performance of OLLAWV.

The experimental study showed the better performance of OLLAWV against 12 popular data stream classifiers. As a base-line algorithm, OLLAWV was shown through various metrics and statistical analysis to be very attractive in the batch data stream setting, sharing competitive run times with the top performing contemporary methods.

CHAPTER 7

CONCLUSIONS

This thesis introduced several novel SVM algorithms for learning from the following diverse machine learning paradigms: multi-target regression, multi-instance classification, traditional supervised classification, and data stream classification.

Three unique approaches for multi-target regression were proposed: the baseline problem transformation support vector regressor SVR, an ensemble of randomly generated chains using this base model SVRRC, and a maximally correlated chained model SVRCC. The results highlighted the better performance of SVR as a base model, however, because it generates independent regressors, the possible correlations amongst the targets are lost on the final model. SVRRC was designed to test whether taking these correlations into account would benefit the final learning model, and the results showed a performance increase. However, due to the random nature of SVRRC and its limit on the number of generated chains, capturing target correlations is not guaranteed. SVRCC was designed to remedy this issue using a maximum correlation chain and proved to capture target correlations accurately, providing the best results among the contributions, as well as against the methods compared.

A novel multi-instance bag-level formulation and algorithm, MIRSVM, with a bag-representative selector, are proposed. MIRSVM trains the model on bag-level information, iteratively selecting the best representative instance for each positive and negative bag, while finding the optimal separating hyperplane. This approach, unlike other existing ones, eliminates possible class imbalance issues by allowing both positive and negative bags to be represented. The experimental and statistical study showed that bag-level learners outperform instance-level learners and wrapper methods. MIRSVM outperformed current contemporary multi-instance SVMs, as well as other algorithms of different classes over several metrics.

After the previous experimental studies, it was evident that the existing popular SVM

126

solvers that were used suffered from several disadvantages when used in traditional and non-traditional settings. This prompted the design and implementation of a novel online, also known as stochastic, learning algorithm for solving the primal L1-SVM problem, dubbed OLLAWV. Unlike other online methods, OLLAWV eliminates the need for specifying the number of iterations, as well as the use of a regularization term. The proposed algorithm uses early stopping as its regularizer. OLLAWV also was designed to have a novel stopping criteria, a trait that most stochastic methods do not have. The experimental study, involving strict nested cross-validation, evaluated and compared the proposal with current popular SVM kernel methods that have been shown to outperform the traditional and widely used approaches for solving L1-SVMs, such as SMO and quadratic programming solvers. The results of the experimental study, along with complementary statistical analysis, showed the better performance of OLLAWV against the compared methods, along with 5 non-SVM contemporary methods. OLLAWV was shown to produce sparse models at very fast speeds, without sacrificing accuracy. This, along with the online nature of OLLAWV, prompted the investigation of its performance in the data stream setting.

The final contribution of this thesis involved implementing OLLAWV in a batch data stream classification setting due to its major success with traditional classification. However, before OLLAWV's implementation and experimentation was complete, its parent, OLLA-L2, was the first contender for a data stream algorithm due to its online nature, competitive performance against the popular SMO algorithm, as well as its ability to produce sparse models at very fast rates. After preliminary experiments between the two algorithms, OLLAWV was shown to outperform OLLA-L2 in terms of accuracy. The prequential experimental study analyzed the performance of OLLAWV against 12 popular data stream algorithms, most capable of adapting to drifts. The goal of the study was to assess OLLAWV's performance as a base-line model in the data stream (stationary or not) environment. The results highlighted OLLAWV's better performance against the methods compared. They also indicated that adaptive algorithms and methods with drift-detectors were superior to the rest.

CHAPTER 8

FUTURE WORK

Due to the diversity of the contributions of this thesis, there are a few paths that could be explored for future work.

One development would be updating the base learners in the multi-target regression and multi-instance classification contributions. The purpose of the design and implementation of OLLAWV was to remedy the flaws of current popular SVM solvers and provide a useful alternative. Therefore, a natural extension to this research would be to apply OLLAWV to the contributions in Chapters 3 and 4, in the place of the SVM solvers currently used. This, in turn, would lead to another opportunity, which would be to extend OLLAWV to the regression case, thus enabling its use in the multi-target regression setting. Having both the classification and regression OLLAWV solvers could also enable investigating their performance in the *multi-instance regression* or *multi-label classification* setting.

Another possible improvement is to extend OLLAWV to the online stream setting. The complexity in this task stems from one of OLLAWV's unique attributes: it operates by selecting and updating a worst violating sample per iteration. In an online setting, there is no definitive way of knowing which samples are/will be worst-violators. Due to this, OLLAWV (in its present condition) may not be able to be adapted to the online stream environment. OLLA-L2, on the other hand, would fit nicely in the pure online setting, since it proceeds sample by sample, with no sample order required. The main issue with OLLA-L2 is its lack of meaningful stopping criteria, possibly resulting in a dense model, which would deteriorate its run time and classification performance. However, one potential remedy could come from investigating combining OLLAWV and the positive attributes possessed by OLLA-L2, as well as implementing a novel decremental unlearning mechanism for the efficient handling non-stationary streams. These possibilities would further improve this family of online algorithms

under the stationary and non-stationary data stream setting.

Although OLLAWV shows considerable performance against widely used SVM solvers and other machine learning methods, it still has room for improvement. The biggest computational bottleneck in the steps taken by OLLAWV is the kernel vector calculation step. One remedy for this would be to develop a parallelized/distributed version of OLLAWV and analyze the amount of improvement achieved.

REFERENCES

[1] C. Aggarwal. "A survey of stream classification algorithms". In: *Data Classification: Algorithms and Applications*. CRC Press, 2014.

[2] M. A. Aizerman, E. A. Braverman, and L. Rozonoer. "Theoretical foundations of the potential function method in pattern recognition learning". In: *Automation and Remote Control*. 25. 1964, pp. 821–837.

[3] J. Alcalá-Fdez et al. "KEEL Data-mining software tool: data set repository, integration of algorithms and experimental analysis framework, analysis framework". In: *Journal of Multiple-Valued Logic and Soft Computing* 17 (2011), pp. 255–287.

[4] C. Alippi. *Intelligence for embedded systems*. Springer, 2014.

[5] C. Alippi, G. Boracchi, and M. Roveri. "Just in time classifiers: Managing the slow drift case". In: *Proceedings of the International Joint Conference on Neural Networks*. IEEE, 2009, pp. 114–120.

[6] J. Amores. "Multiple instance classification: review, taxonomy, and comparative study". In: *Artificial Intelligence* 201 (2013), pp. 81–105.

[7] S. Andrews, I. Tsochantaridis, and T. Hofmann. "Support Vector Machines for Multiple-Instance Learning". In: *Proceedings of the 15th International Conference on Neural Information Processing Systems*. 2002, pp. 577–584.

[8] A. Asuncion and D. Newman. *UCI machine learning repository*. 2007.

[9] P. Auer and R. Ortner. "A Boosting Approach to Multiple Instance Learning". In: *European Conference on Machine Learning*. Vol. 3201. Lecture Notes in Computer Science. 2004, pp. 63–74.

[10] B. Babenko, M. H. Yang, and S. Belongie. "Visual tracking with online multiple instance learning". In: *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2009, pp. 983–990.

[11] M. Baena-García et al. "Early drift detection method". In: *Proceedings of the 4th International Workshop on Knowledge Discovery from Data Streams*. 2006.

[12] J. Baxter. "A Bayesian/information theoretic model of learning to learn via multiple task sampling". In: *Machine Learning* 28 (1997), pp. 7–39.

[13] S. Ben-David and R. Schuller. "Exploiting task relatedness for multiple task learning". In: *Learning Theory and Kernel Machines*. Springer, 2003, pp. 567–580.

[14] K. P. Bennett and E. J. Bredensteiner. "Duality and geometry in SVM classifiers". In: *International Conference on Machine Learning*. 2000, pp. 57–64.

[15] A. Bifet and R. Gavalda. "Learning from time-changing data with adaptive windowing". In: *Proceedings of the SIAM international conference on data mining*. SIAM. 2007, pp. 443–448.

[16] A. Bifet et al. "Efficient data stream classification via probabilistic adaptive windows". In: *Proceedings of the 28th annual symposium on applied computing*. ACM. 2013, pp. 801–806.

[17] A. Bifet et al. "MOA: Massive Online Analysis". In: *Journal of Machine Learning Research* 11 (2010), pp. 1601–1604.

[18] A. Bifet et al. "New ensemble methods for evolving data streams". In: *Proceedings of the 15th SIGKDD international conference on knowledge discovery and data mining*. ACM. 2009, pp. 139–148.

[19] L. Bjerring and E. Frank. "Beyond trees: adopting MITI to learn rules and ensemble classifiers for multi-instance data". In: *Proceedings of the Australasian Joint Conference on Artificial Intelligence*. 2011, pp. 41–50.

[20]   M. Blaschko and T. Hofmann. "Conformal Multi-instance Kernels". In: *Proceedings of the 19th Conference on Advances in Neural Information Processing Systems*. 2006, pp. 1–6.

[21]   H. Blockeel, L. De Raedt, and J. Ramon. "Top-down induction of clustering trees". In: *Proceedings of the 15th International Conference of Machine Learning*. 1998, pp. 55–63.

[22]   H. Blockeel, D. Page, and A. Srinivasan. "Multi-instance tree learning". In: *Proceedings of the International Conference on Machine Learning*. 2005, pp. 57–64.

[23]   H. Borchani et al. "A survey on multi-output regression". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 5.5 (2015), pp. 216–233.

[24]   A. Bordes et al. "Fast kernel classifiers with online and active learning". In: *Journal of Machine Learning Research* 6 (2005), pp. 1579–1619.

[25]   A. Bordes et al. "Solving multiclass support vector machines with LaRank". In: *Proceedings of the 24th international conference on Machine learning*. ACM. 2007, pp. 89–96.

[26]   B. E. Boser, I. M. Guyon, and V. N. Vapnik. "A training algorithm for optimal margin classifiers". In: *Proceedings of the 5th Annual Workshop on Computational Learning Theory*. 1992, pp. 144–152.

[27]   L. Bottou. "Large-scale machine learning with stochastic gradient descent". In: *Proceedings of nineteenth International Conference on Computational Statistics*. 2010, pp. 177–186.

[28]   L. Bottou and Y. L. Cun. "Large scale online learning". In: *Advances in neural information processing systems*. 2004, pp. 217–224.

[29]   L. Bottou, F. E. Curtis, and J. Nocedal. "Optimization methods for large-scale machine learning". In: *Society for Industrial and Applied Mathematics Review* 60.2 (2018), pp. 223–311.

[30] L. Bottou and N. Murata. *Stochastic approximations and efficient learning*. 2002.

[31] L. Bottou et al. *Large-scale kernel machines*. MIT press, 2007.

[32] O. Bousquet and L. Bottou. "The tradeoffs of large scale learning". In: *Advances in Neural Information Processing Systems*. 2008, pp. 161–168.

[33] S. Boyd and L. Vanderberghe. *Convex Optimization*. Cambridge University Press, 2004.

[34] L. Breiman. "Bagging predictors". In: *Machine Learning* 24.2 (1996), pp. 123–140.

[35] L. Breiman. "Random forests". In: *Machine learning* 45.1 (2001), pp. 5–32.

[36] L. Breiman and J. H. Friedman. "Predicting multivariate responses in multiple linear regression". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 59.1 (1997), pp. 3–54.

[37] L. Breiman et al. *Classification and regression trees*. Chapman & Hall, 1984.

[38] R. J. Brown and J. V. Zidek. "Adaptive multivariate ridge regression". In: *The Annals of Statistics* (1980), pp. 64–74.

[39] M. Brudnak. "Vector-valued support vector regression". In: *International Joint Conference on Neural Networks*. IEEE. 2006, pp. 1562–1569.

[40] M. C. Burl et al. "Diamond Eye: A distributed architecture for image data mining". In: *Data Mining and Knowledge Discovery: Theory, Tools, and Technology*. Vol. 3695. International Society for Optics and Photonics. 1999, pp. 197–207.

[41] A. Cano and B. Krawczyk. "Learning classification rules with differential evolution for high-speed data stream mining on GPUs". In: *IEEE Congress on Evolutionary Computation*. 2018, pp. 197–204.

[42] A. Cano, A. Zafra, and S. Ventura. "Speeding up multiple instance learning classification rules on GPUs". In: *Knowledge and information systems* 44.1 (2015), pp. 127–145.

[43] M. A. Carbonneau et al. "Multiple instance learning: A survey of problem characteristics and applications". In: *Pattern Recognition* 77 (2018), pp. 329–353.

[44] M. A. Carbonneau et al. "Robust multiple-instance learning ensembles using random subspace instance selection". In: *Pattern Recognition* 58 (2016), pp. 83–99.

[45] R. Caruana. "Multitask learning". In: (1998), pp. 95–133.

[46] G. Cauwenberghs and T. Poggio. "Incremental and decremental support vector machine learning". In: *Advances in Neural Information Processing Systems*. 2001, pp. 409–415.

[47] C. C. Chang and C. J. Lin. "LIBSVM: A library for support vector machines". In: *ACM Transactions on Intelligent Systems and Technology* 2.3 (2011). Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`, pp. 1–27.

[48] O. Chapelle. "Training a Support Vector Machine in the Primal". In: *Neural Computing* 19.5 (2007), pp. 1155–1178.

[49] Y. Chen, J. Bi, and J. Wang. "MILES: multiple-instance learning via embedded instance selection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28.12 (2006), pp. 1931–1947.

[50] Y. Chen and J. Wang. "Image categorization by learning and reasoning with regions". In: *Journal of Machine Learning Research* 5 (2004), pp. 913–939.

[51] Pak-Ming Cheung and James T Kwok. "A regularization framework for multiple-instance learning". In: *Proceedings of the 23rd international conference on Machine learning*. ACM. 2006, pp. 193–200.

[52] J.-P. Chilès and P. Delfiner: *Geostatistics: Modeling Spatial Uncertainty*. Wiley Series in Probability and Statistics, 1999.

[53] R. Collobert and S. Bengio. "Links between perceptrons, MLPs and SVMs". In: *Proceedings of the twenty first International Conference on Machine Learning*. 2004, p. 23.

[54] C. Cortes and V. Vapnik. "Support-vector networks". In: *Machine Learning* 20.3 (1995), pp. 273–297.

[55] National Research Council. *Frontiers in massive data analysis*. National Academies Press, 2013.

[56] G. De'Ath. "Multivariate regression trees: a new technique for modeling species–environment relationships". In: *Ecology* 83.4 (2002), pp. 1105–1117.

[57] J. Derrac et al. "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms". In: *Swarm and Evolutionary Computation* 1.1 (2011), pp. 3–18.

[58] T. G. Dietterich, R. H. Lathrop, and T. Lozano-Perez. "Solving the multiple instance problem with axis-parallel rectangles". In: *Artificial Intelligence* 89 (1997), pp. 31–71.

[59] G. Ditzler et al. "Learning in nonstationary environments: A survey". In: *Computational Intelligence Magazine* 10.4 (2015), pp. 12–25.

[60] C. Domeniconi and D. Gunopulos. "Incremental support vector machine construction". In: *Proceedings of the IEEE International Conference on Data Mining*. IEEE. 2001, pp. 589–592.

[61] P. Domingos and G. Hulten. "Mining high-speed data streams". In: *Proceedings of the 6th SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2000, pp. 71–80.

[62] L. Dong. "A comparison of multi-instance learning algorithms". PhD thesis. The University of Waikato, 2006.

[63] G. Doran and S. Ray. "A theoretical and empirical analysis of support vector machine methods for multiple-instance classification". In: *Machine Learning* 97.1-2 (2014), pp. 79–102.

[64] H. Drucker et al. "Support vector regression machines". In: *Advances in neural information processing systems*. 1997, pp. 155–161.

[65] O. J. Dunn. "Multiple comparisons among means". In: *Journal of the American Statistical Association* 56.293 (1961), pp. 52–64.

[66] F. Eibe et al. "The WEKA workbench". In: *Online Appendix for Data Mining: Practical Machine Learning Tools and Techniques* 4 (2016).

[67] R. Elwell and R. Polikar. "Incremental learning of concept drift in nonstationary environments". In: *IEEE Transactions on Neural Networks* 22.10 (2011), pp. 1517–1531.

[68] T. Evgeniou, C.A. Micchelli, and M. Pontil. "Learning multiple tasks with kernel methods". In: *Journal of Machine Learning Research* 6 (2005), pp. 615–637.

[69] T. Evgeniou and M. Pontil. "Regularized multi–task learning". In: *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. 2004, pp. 109–117.

[70] J. Foulds and E. Frank. "A review of multi-instance learning assumptions". In: *The Knowledge Engineering Review* 25.1 (2010), pp. 1–24.

[71] E. T. Frank and X. Xu. *Applying propositional learning algorithms to multi-instance data*. Tech. rep. University of Waikato, Department of Computer Science, 2003.

[72] Y. Freund and R. E. Schapire. "Experiments with a new boosting algorithm". In: *Proceedings of the 13th International Conference on Machine Learning*. 1996, pp. 148–156.

[73] Y. Freund and R. E. Schapire. "Large margin classification using the perceptron algorithm". In: *Machine Learning* 37.3 (1999), pp. 277–296.

[74] T. T. Frie, N. Cristianini, and C. Campbell. "The kernel-adatron algorithm: a fast and simple learning procedure for support vector machines". In: *Proceedings of the 15th International Conference on Machine Learning*. 1998, pp. 188–196.

[75] Z. Fu, A. Robles-Kelly, and J. Zhou. "MILIS: multiple instance learning with instance selection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.5 (2011), pp. 958–977.

[76] G. Fung and O. L. Mangasarian. "Incremental support vector machine classification". In: *Proceedings of the International Conference on Data Mining*. SIAM. 2002, pp. 247–260.

[77] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy. "A survey of classification methods in data streams". In: *Data streams*. Springer, 2007, pp. 39–59.

[78] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy. "Mining data streams: a review". In: *ACM Sigmod Record* 34.2 (2005), pp. 18–26.

[79] J. Gama, R. Fernandes, and R. Rocha. "Decision trees for mining data streams". In: *Intelligent Data Analysis* 10.1 (2006), pp. 23–45.

[80] J. Gama and P Kosina. "Learning decision rules from data streams". In: *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*. Vol. 22. 1. 2011, pp. 1255–1261.

[81] J. Gama, R. Sebastião, and P. P. Rodrigues. "On evaluating stream learning algorithms". In: *Machine learning* 90.3 (2013), pp. 317–346.

[82] J. Gama et al. "A survey on concept drift adaptation". In: *ACM computing surveys (CSUR)* 46.4 (2014), p. 44.

[83] J. Gama et al. "Learning with drift detection". In: *Proceedings of the 7th Brazilian symposium on artificial intelligence, lecture notes in computer science.* Springer. 2004, pp. 286–295.

[84] T. Gärtner et al. "Multi-Instance Kernels". In: *Proceedings of the 19th International Conference on Machine Learning.* 2002, pp. 179–186.

[85] S. Godbole and S. Sarawagi. "Discriminative methods for multi-labeled classification". In: *Pacific-Asia conference on knowledge discovery and data mining.* Springer. 2004, pp. 22–30.

[86] H. M. Gomes and F. Enembreck. "Sae: Social adaptive ensemble classifier for data streams". In: *IEEE Symposium on Computational Intelligence and Data Mining.* IEEE. 2013, pp. 199–206.

[87] H. M. Gomes and F. Enembreck. "SAE2: advances on the social adaptive ensemble classifier for data streams". In: *Proceedings of the 29th Annual ACM Symposium on Applied Computing.* ACM. 2014, pp. 798–804.

[88] M. Hall et al. "The WEKA data mining software: an update". In: *ACM SIGKDD explorations newsletter* 11.1 (2009), pp. 10–18.

[89] T. Hastie et al. "The entire regularization path for the support vector machine". In: *Journal of Machine Learning Research* 5 (2004), pp. 1391–1415.

[90] R. Herbrich. *Learning kernel classifiers.* MIT Press, 2016.

[91] G. Herman et al. "Region-based image categorization with reduced feature set". In: *Proceedings of the 10th IEEE Workshop on Multimedia Signal Processing.* 2008, pp. 586–591.

[92] F. Herrera et al. *Multiple Instance Learning: Foundations and Algorithms.* Springer, 2016.

[93]   A. E. Hoerl and R. W. Kennard. "Ridge regression: Biased estimation for nonorthogonal problems". In: *Technometrics* 12.1 (1970), pp. 55–67.

[94]   M. Hollander and D.A. Wolfe. *Nonparametric statistical methods*. John Wiley & Sons, Inc., 1999.

[95]   C. J. Hsieh et al. "A dual coordinate descent method for large-scale linear SVM". In: *Proceedings of the 25th international conference on Machine learning*. ACM. 2008, pp. 408–415.

[96]   T. M. Huang, V. Kecman, and I. Kopriva. *Kernel based algorithms for mining huge data sets, supervised, semi-supervised, and unsupervised learning*. Springer-Verlag, 2006.

[97]   G. Hulten, L. Spencer, and P. Domingos. "Mining time-changing data streams". In: *Proceedings of the 7th International Conference on Knowledge Discovery and Data Mining*. ACM. 2001, pp. 97–106.

[98]   A. J. Izenman. "Reduced-rank regression for the multivariate linear model". In: *Journal of multivariate analysis* 5.2 (1975), pp. 248–264.

[99]   G. Jaber, A. Cornuéjols, and P. Tarroux. "A new on-line learning method for coping with recurring concepts: the ADACC system". In: *International Conference on Neural Information Processing*. Springer. 2013, pp. 595–604.

[100] M. Jeong and G. G. Lee. "Multi-domain spoken language understanding with transfer learning". In: *Speech Communication* 51.5 (2009), pp. 412–424.

[101] T. Joachims. "Advances in Kernel Methods - Support Vector Learning". In: ed. by B. Schölkopf, C. J. C. Burges, and A. J. Smola. MIT Press, 1999. Chap. Making Large-scale Support Vector Machine Learning Practical, pp. 169–184.

[102] T. Joachims. "Training linear SVMs in linear time". In: *Proceedings of the 12th SIGKDD international conference on knowledge discovery and data mining*. ACM. 2006, pp. 217–226.

[103]   G. H. John and P. Langley. "Estimating continuous distributions in Bayesian classifiers". In: *Proceedings of the 11th conference on uncertainty in artificial intelligence.* Morgan Kaufmann Publishers Inc. 1995, pp. 338–345.

[104]   H. Kargupta et al. "MobiMine: Monitoring the stock market from a PDA". In: *SIGKDD Explorations Newsletter* 3.2 (2002), pp. 37–46.

[105]   H. Kargupta et al. "VEDAS: A mobile and distributed data stream mining system for real-time vehicle monitoring". In: *Proceedings of the SIAM International Conference on Data Mining.* SIAM. 2004, pp. 300–311.

[106]   V. Kecman. "Fast online algorithm for nonlinear support vector machines and other alike models". In: *Optical Memory and Neural Networks* 25.4 (2016), pp. 203–218.

[107]   V. Kecman. *Learning and soft computing: support vector machines, neural networks, and fuzzy logic models.* MIT Press, 2001.

[108]   V. Kecman, T. M Huang, and M. Vogt. "Iterative single data algorithm for training kernel machines from huge data sets: theory and performance". In: *Studies in Computational Intelligence* 177 (2005), pp. 255–274.

[109]   V. Kecman and G. Melki. "Fast online algorithms for support vector machines". In: *Proceedings of the IEEE Southeast Conference.* 2016, pp. 1–6.

[110]   V. Kecman and L. Zigic. "Algorithms for direct L2 support vector machines". In: *Proceedings of the IEEE International Symposium on Innovations in Intelligent Systems and Applications.* 2014, pp. 419–424.

[111]   S. S. Keerthi et al. "Improvements to Platt's SMO algorithm for SVM classifier design". In: *Neural computation* 13.3 (2001), pp. 637–649.

[112]   J. Kivinen, A. J. Smola, and R. C. Williamson. "Large margin classification for moving targets". In: *International Conference on Algorithmic Learning Theory.* Vol. 2. 2002, pp. 113–127.

[113] J. Kivinen, A. J. Smola, and R. C. Williamson. "Online learning with kernels". In: *IEEE Transactions on Signal Processing* 52.8 (2004), pp. 2165–2176.

[114] R. Klinkenberg and T. Joachims. "Detecting Concept Drift with Support Vector Machines." In: *Proceedings of the 17th international conference on machine learning.* 2000, pp. 487–494.

[115] D. Kocev et al. "Ensembles of multi-objective decision trees". In: *European Conference on Machine Learning.* Springer. 2007, pp. 624–631.

[116] D. Kocev et al. "Tree ensembles for predicting structured outputs". In: *Pattern Recognition* 43 (3 2013), pp. 817–833.

[117] D. Kocev et al. "Using single-and multi-target regression trees and ensembles to model a compound index of vegetation condition". In: *Ecological Modelling* 220.8 (2009), pp. 1159–1168.

[118] J. Z. Kolter and M. A. Maloof. "Dynamic weighted majority: An ensemble method for drifting concepts". In: *Journal of Machine Learning Research* 8.Dec (2007), pp. 2755–2790.

[119] B. Krawczyk and A. Cano. "Online Ensemble Learning with Abstaining Classifiers for Drifting and Noisy Data Streams". In: *Applied Soft Computing* 68 (2018), pp. 677–692.

[120] B. Krawczyk and M. Woźniak. "One-class classifiers with incremental learning and forgetting for data streams with concept drift". In: *Soft Computing* 19.12 (2015), pp. 3387–3400.

[121] B. Krawczyk et al. "Ensemble learning for data stream analysis: A survey". In: *Information Fusion* 37 (2017), pp. 132–156.

[122] I. A. Lawal and S. A. Abdulkarim. "Adaptive SVM for data stream classification". In: *South African Computer Journal* 29.1 (2017), pp. 27–42.

[123] X. Li and W. Yu. "Data stream classification for structural health monitoring via on-line support vector machines". In: *Proceedings of the 1st international conference on big data computing service and applications*. IEEE. 2015, pp. 400–405.

[124] Q. Liu et al. "Multi-task learning for cross-platform siRNA efficacy prediction: an in-silico study". In: *BMC Bioinformatics* 11.1 (2010), pp. 181–196.

[125] O. L. Mangasarian and D. R. Musicant. "Successive overrelaxation for support vector machines". In: *IEEE Transactions on Neural Networks* 10.5 (1999), pp. 1032–1037.

[126] O. Maron and T. Lozano-Pérez. "A framework for multiple-instance learning". In: *Neural Information Processing Systems* 3201 (1998), pp. 570–576.

[127] G. Melki. *Fast online training of L1 support vector machines*. Virginia Commonwealth University, 2016.

[128] G. Melki, A. Cano, and S. Ventura. "MIRSVM: Multi-Instance Support Vector Machine with Bag Representatives". In: *Pattern Recognition* (2018).

[129] G. Melki and V. Kecman. "Speeding up online training of L1 Support Vector Machines". In: *SoutheastCon, 2016*. IEEE. 2016, pp. 1–6.

[130] G. Melki et al. "Multi-target support vector regression via correlation regressor chains". In: *Information Sciences* 415 (2017), pp. 53–69.

[131] G. Melki et al. "OLLAWV: OnLine Learning Algorithm using Worst-Violators". In: *Applied Soft Computing* 66 (2018), pp. 384–393.

[132] E. Osuna, R. Freund, and F. Girosi. "An improved training algorithm for support vector machines". In: *Proceedings of the IEEE Workshop on Neural Networks for Signal Processing*. IEEE. 1997, pp. 276–285.

[133] N. C. Oza and O. Russel. "Online bagging and boosting". In: *Proceedings of the 8th international workshop on articial intelligence and statistics*. Morgan Kaufmann, 2001, pp. 105–112.

[134] N. C. Oza and S. Russell. *Online ensemble learning*. University of California, Berkeley, 2001.

[135] C. Panagiotakopoulos and R. Tsampouka. "The stochastic gradient descent for the primal L1-SVM optimization revisited". In: *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases*. 2013, pp. 65–80.

[136] R. Pelossof et al. "Online coordinate boosting". In: *IEEE 12th International Conference on Computer Vision Workshops*. IEEE. 2009, pp. 1354–1361.

[137] B. Pfahringer, G. Holmes, and R. Kirkby. "New options for hoeffding trees". In: *Australasian Joint Conference on Artificial Intelligence*. Springer. 2007, pp. 90–99.

[138] J. Platt. "Sequential minimal optimization: A fast algorithm for training support vector machines". In: *Technical Report: MSR-TR-98-14* (1998).

[139] R. Polikar et al. "Learn++: An incremental learning algorithm for supervised neural networks". In: *IEEE Transactions on systems, man, and cybernetics* 31.4 (2001), pp. 497–508.

[140] Z. Qi et al. "Online multiple instance boosting for object detection". In: *Neurocomputing* 74.10 (2011), pp. 1769–1775.

[141] L. Ralaivola and F. dAlché Buc. "Incremental support vector machine learning: A local approach". In: *International Conference on Artificial Neural Networks*. Springer. 2001, pp. 322–330.

[142] S. Ray and M. Craven. "Supervised versus multiple instance learning: an empirical comparison". In: *Proceeding of the International Conference on Machine Learning*. 2005, pp. 697–704.

[143] J. Read et al. "Classifier chains for multi-label classification". In: *Machine learning* 85.3 (2011), p. 333.

[144]   H. Robbins and S. Monro. "A Stochastic Approximation Method". In: *The Annals of Mathematical Statistics* 22.3 (1951), pp. 400–407.

[145]   F. Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological Review* 65.6 (1958), pp. 386–408.

[146]   S. Ruping. "Incremental learning with support vector machines". In: *Proceedings of the IEEE International Conference on Data Mining*. IEEE. 2001, pp. 641–642.

[147]   R. E. Schapire and Y. Singer. "Improved boosting algorithms using confidence-rated predictions". In: *Machine learning* 37.3 (1999), pp. 297–336.

[148]   K. Scheinberg. "An efficient implementation of an active set method for SVMs". In: *Journal of Machine Learning Research* 7 (2006), pp. 2237–2257.

[149]   B. Schölkopf, R. Herbrich, and A. J. Smola. "A generalized represWenter theorem". In: *International conference on computational learning theory*. Springer. 2001, pp. 416–426.

[150]   B. Schölkopf and A. J. Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[151]   S. Shalev-Shwartz and S. Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[152]   S. Shalev-Shwartz and N. Srebro. "SVM optimization: inverse dependence on training set size". In: *Proceedings of the 25th International Conference on Machine learning*. ACM. 2008, pp. 928–935.

[153]   S. Shalev-Shwartz et al. "Pegasos: Primal estimated sub-gradient solver for svm". In: *Mathematical programming* 127.1 (2011), pp. 3–30.

[154]   J. Shawe-Taylor and S. Sun. "A review of optimization methodologies in support vector machines". In: *Neurocomputing* 74.17 (2011), pp. 3609–3618.

[155] A. Shilton et al. "Incremental training of support vector machines". In: *IEEE transactions on neural networks* 16.1 (2005), pp. 114–131.

[156] T. Similä and J. Tikka. "Input selection and shrinkage in multiresponse linear regression". In: *Computational Statistics & Data Analysis* 52.1 (2007), pp. 406–422.

[157] E. Spyromitros-Xioufis et al. "Multi-label classification methods for multi-target regression". In: *ArXiv e-prints* (2012).

[158] E. Spyromitros-Xioufis et al. "Multi-target regression via input space expansion: treating targets as inputs". In: *Machine Learning* 104.1 (2016), pp. 55–98.

[159] R. Strack. "Geometric approach to support vector machines learning for large datasets". PhD thesis. Virginia Commonwealth University, 2013.

[160] J. Struyf and S. Džeroski. "Constraint based induction of multi-objective regression trees". In: *International Workshop on Knowledge Discovery in Inductive Databases*. Springer. 2005, pp. 222–233.

[161] J. A. K. Suykens and J. Vandewalle. "Least squares support vector machine classifiers". In: *Neural processing letters* 9.3 (1999), pp. 293–300.

[162] N. A. Syed, H. Liu, and K. K. Sung. "Handling concept drifts in incremental learning with support vector machines". In: *Proceedings of the 5th SIGKDD international conference on knowledge discovery and data mining*. ACM. 1999, pp. 317–321.

[163] P. N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005. ISBN: 0321321367.

[164] S. Thrun. "Is learning the n-th thing any easier than learning the first?" In: *Advances in Neural Information Processing Systems*. 1996, pp. 640–646.

[165] D. Tomar and S. Agarwal. "Multiple Instance Learning Based on Twin Support Vector Machine". In: *Advances in Computer and Computational Sciences*. Springer, 2017, pp. 497–507.

[166] I. W. Tsang, A. Kocsor, and K. T. Kwok. "Simpler core vector machines with enclosing balls". In: *Proceedings of the twenty fourth International Conference on Machine Learning*. 2007, pp. 911–918.

[167] I. W. Tsang, J. T. Kwok, and P. M. Cheung. "Core vector machines: Fast SVM training on very large data sets". In: *Journal of Machine Learning Research* 6 (2005), pp. 363–392.

[168] I. Tsochantaridis et al. "Large margin methods for structured and interdependent output variables". In: *Journal of machine learning research* 6 (2005), pp. 1453–1484.

[169] G. Tsoumakas et al. "Mulan: A java library for multi-label learning". In: *Journal of Machine Learning Research* 12 (2011), pp. 2411–2414.

[170] G. Tsoumakas et al. "Multi-target regression via random linear target combinations". In: *Machine Learning and Knowledge Discovery in Databases* 8726 (2014), pp. 225–240.

[171] V. Vapnik. *The nature of statistical learning theory*. Springer-Verlag, 1995.

[172] V. Vapnik, S. E. Golowich, and A. J. Smola. "Support vector method for function approximation, regression estimation and signal processing". In: *Advances in neural information processing systems*. 1997, pp. 281–287.

[173] V. N. Vapnik and A. J. Chervonenkis. "On the uniform convergence of relative frequencies of events to their probabilities". In: *Theory of Probability and its Applications*. Vol. 16. Springer, 1971, pp. 264–280.

[174] E. Vazquez and E. Walter. "Multi-Output Suppport Vector Regression". In: *IFAC Proceedings Volumes* 36.16 (2003), pp. 1783–1788.

[175] J. S. Vitter. "Random sampling with a reservoir". In: *Transactions on Mathematical Software* 11.1 (1985), pp. 37–57.

[176] J. Wang and J. Zucker. "Solving the multiple-instance problem: a lazy learning approach." In: *Proceedings of the International Conference on Machine Learning.* 2000, pp. 1119–1126.

[177] G. I. Webb et al. "Characterizing concept drift". In: *Data Mining and Knowledge Discovery* 30.4 (2016), pp. 964–994.

[178] Nils Weidmann, Eibe Frank, and Bernhard Pfahringer. "A two-level learning method for generalized multi-instance problems". In: *Fourteenth European Conference on Machine Learning.* Springer, 2003, pp. 468–479.

[179] B. Widrow and M. E. Hoff. *Adaptive switching circuits.* Tech. rep. University of California, 1960.

[180] F. Wilcoxon. "Individual comparisons by ranking methods". In: *Biometrics bulletin* 1.6 (1945), pp. 80–83.

[181] S. Wright and J. Nocedal. *Numerical optimization.* New York: Springer-Veralg, 1999.

[182] Q. Wu et al. "ML-TREE: A tree-structure-based approach to multilabel learning". In: *IEEE Transactions on Neural Networks and Learning Systems* 26.3 (2015), pp. 430–443.

[183] Y. Xie et al. "Online multiple instance gradient feature selection for robust visual tracking". In: *Pattern Recognition Letters* 33.9 (2012), pp. 1075–1082.

[184] T. Xiong, Y. Bao, and Z. Hu. "Multiple-output support vector regression with a firefly algorithm for interval-valued stock price index forecasting". In: *Knowledge-Based Systems* 55 (2014), pp. 87–100.

[185] S. Xu et al. "Multi-output least-squares support vector regression machines". In: *Pattern Recognition* 34.9 (2013), pp. 1078–1084.

[186] X. Xu. "Statistical learning in multiple instance problems". PhD thesis. The University of Waikato, 2003.

[187] Y. Yi and M. Lin. "Human action recognition with graph-based multiple-instance learning". In: *Pattern Recognition* 53 (2016), pp. 148–162.

[188] A. Zafra, E. L. Gibaja, and S. Ventura. "Multiple instance learning with multiple objective genetic programming for web mining". In: *Applied Soft Computing* 11.1 (2011), pp. 93–102.

[189] A. Zafra, C. Romero, and S. Ventura. "Multiple instance learning for classifying students in learning management systems". In: *Expert Systems with Applications* 38.12 (2011), pp. 15020–15031.

[190] W. Zang et al. "Comparative study between incremental and ensemble learning on data streams: Case study". In: *Journal Of Big Data* 1.1 (2014), p. 5.

[191] C. Zhang, J. C. Platt, and P. A. Viola. "Multiple instance boosting for object detection". In: *Advances in neural information processing systems*. 2006, pp. 1417–1424.

[192] M. L. Zhang and Z. H. Zhou. "A review on multi-label learning algorithms". In: *IEEE Transactions on Knowledge and Data Engineering* 26.8 (2014), pp. 1819–1837.

[193] Q. Zhang and S. Goldman. "Em-DD: an improved multiple-instance learning technique". In: *Advances in Neural Information Processing Systems*. 2002, pp. 1073–1080.

[194] T. Zhang. "Solving large scale linear prediction problems using stochastic gradient descent algorithms". In: *Proceedings of the twenty first International Conference on Machine Learning*. ACM. 2004, p. 116.

[195] W. Zhang, X. Liu, and D. Shi. "Multi-output LS-SVR machine in extended feature space". In: *Proceedings of the International Conference on Computational Intelligence for Measurement Systems and Applications* (2012), pp. 130–134.

[196] Z. H. Zhou, Y. Y. Sun, and Y. F. Li. "Multi-instance learning by treating instances as non-iid samples". In: *Proceedings of the 26th annual international conference on machine learning*. ACM. 2009, pp. 1249–1256.

[197] Z. H. Zhou et al. "Big data opportunities and challenges: Discussions from data analytics perspectives [discussion forum]". In: *Computational Intelligence Magazine* 9.4 (2014), pp. 62–74.

[198] L. J. Zigic. "Direct L2 support vector machine". PhD thesis. Virginia Commonwealth University, 2016.

VITA

Gabriella Melki received her BSc. in Computer Science from the American University of Beirut in 2011 and her MSc. in Computer Science from Virginia Commonwealth University in 2016. As a full-time student in the dual Ph.D. program between Virginia Commonwealth University and the University of Córdoba in Spain, her research is focused on machine learning algorithms for large datasets and diverse data paradigms. Her more specialized field of interest is support vector machines.

Publications:

- **Melki G**, Kecman V, Ventura S, Cano A, "OLLAWV: OnLine Learning using Worst-Violators", In: *Applied Soft Computing* 66, (2018), pp. 384–393

- **Melki G**, Cano A, Ventura S, "MIRSVM: Multi-Instance Support Vector Machine with Bag Representatives", In: *Pattern Recognition* 75, (2018), pp. 228–214

- **Melki G**, Cano A, Kecman V, Ventura S, "Multi-Target Support Vector Regression Via Correlation Regressor Chains", In: *Information Sciences* 415, (2017), pp. 53–69

- **Melki G**, "Fast Online Training of L1 Support Vector Machines", Master's Thesis, *Virginia Commonwealth University*, (2016), pp. 1–64

- **Melki G**, Kecman V, "Speeding Up Online Training of L1 Support Vector Machines", In: *Proceedings of the IEEE SoutheastCon, 2016*, (2016), pp. 1–6

- Kecman V, **Melki G**, "Fast Online Algorithm for SVMs", In: *Proceedings of the IEEE SoutheastCon, 2016.* (2016), pp. 1–6

- Kecman V, Zigic L, **Melki G**, "Models and Algorithms for Support Vector Machines: Direct L2 SVM", *Seminar at Max Planck Institute for Intelligent Systems, Empirical Inference*, Tübigen, Germany, 2015