©Gabriella Angela Melki, September 2018

All Rights Reserved.

NOVEL SUPPORT VECTOR MACHINES FOR DIVERSE LEARNING PARADIGMS

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

by

GABRIELLA ANGELA MELKI

Ph.D. Candidate

Director: Alberto Cano,

Assistant Professor, Department of Computer Science,

Virginia Commonwealth University

Director: Sebastian Ventura,

Professor, Department of Computer Science & Numerical Analysis,

University of Córdoba

Virginia Commonwealth University

Richmond, Virginia

September 2018

Abstract

NOVEL SUPPORT VECTOR MACHINES FOR DIVERSE LEARNING PARADIGMS

This dissertation introduces novel support vector machines (SVM) for the traditional and non-traditional learning paradigms: *Online Learning*, *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 random chains 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 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.

Add OLLAWV and DS Application

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 both paradigms. The experimental study and analysis confirms that the proposals achieve better performances and more scalable solutions than the methods compared, making them competitive in their respected fields.

TABLE OF CONTENTS

Chapter Page

Abstract i

Table of Contents ii

List of Algorithms iii

List of Tables iv

List of Figures vi

1 Introduction 9

1.1 Contributions of the Proposal 12

2 Background 15

2.1 Support Vector Machine Classification 15
2.2 Support Vector Regression 18
2.3 Methods for Solving the SVM Problem 20

3 Multi-Target SVR using Maximum Correlation Chains 23

3.1 Multi-Target Regression Background 23
3.1.1 Notation 24
3.1.2 Multi-Target Regression 24
3.2 Three Novel SVMs for Multi-Target Regression 26
3.3 Experimental Environment 30
3.4 Results & Statistical Analysis 33
3.4.1 Average Correlation Coefficient 33
3.4.2 Mean Square Error 36
3.4.3 Average Root Mean Square Error 36
3.4.4 Average Relative Root Mean Square Error 37
3.4.5 Run Time 39
3.4.6 Discussion 39
3.5 Conclusions 41

4 Multi-Instance SVM using Bag-Representatives 43

4.1 Multi-Instance Classification Background 43
4.1.1 Notation 43

4.1.2 Multi-Instance Classification 44
4.2 Novel SVM for Multi-Instance Classification 47
4.3 Experimental Environment 53
4.4 Results & Statistical Analysis 55
4.4.1 Accuracy 57
4.4.2 Precision & Recall 59
4.4.3 Cohen's Kappa Rate 61
4.4.4 Area Under ROC Curve 62
4.4.5 Overall Comparison 63
4.5 Conclusions 65

5 Novel OnLine SVM using Worst-Violators 67

5.1 Online Learning Background 67
5.1.1 Notation 67
5.1.2 Regularized Loss Minimization 68
5.1.3 Stochastic Gradient Descent 68
Gradient Descent 68
Subgradients 68
Stochastic Gradient Descent 68
5.1.4 Solving the SVM Problem with Stochastic Gradient Descent 68
5.2 Novel Online Learning Algorithm using Worst-Violators 68
5.3 Experimental Environment 73
5.4 Results & Statistical Analysis 75
5.5 Conclusions 75

6 Conclusions 76

7 Future Work 77

References 79

Vita 88

LIST OF ALGORITHMS

Algorithm Page

3.1 MT Support Vector Regression (SVR) 26

3.2 Build Chained Model 27

3.3 MT SVR with Random-Chains (SVRRC) 28

3.4 MT SVR with Max-Correlation Chain (SVRCC) 29

4.1 Multi-Instance Representative SVM (MIRSVM) 51

5.1 OnLine Learning Algorithm using Worst-Violators (OLLAWV) 74

LIST OF TABLES

Table Page

3.1 Multi-Target (MT) Regression datasets 31

3.2 Average Correlation Coefficient (aCC) for MT regressors 34

3.3 Wilcoxon, Nemenyi, and Holm tests for aCC 34

3.4 Mean Square Error (MSE) for MT regressors 35

3.5 Wilcoxon, Nemenyi, and Holm tests for MSE 35

3.6 Average Root Mean Square Error (aRMSE) for MT regressors 37

3.7 Wilcoxon, Nemenyi, and Holm tests for aRMSE 37

3.8 Average Relative Root Mean Square Error (aRRMSE) for MT regressors 38

3.9 Wilcoxon, Nemenyi, and Holm tests for aRRMSE 38

3.10 Run Time (seconds) for MT regressors 40

3.11 Wilcoxon, Nemenyi, and Holm tests for Run Time 40

4.1 Multi-Instance (MI) Classification datasets 54

4.2 Accuracy for MI classifiers 58

4.3 Holm and Wilcoxon tests for Accuracy 58

4.4 Precision for MI classifiers 59

4.5 Holm and Wilcoxon tests for Precision 59

4.6 Recall for MI classifiers 60

4.7 Holm and Wilcoxon tests for Recall 60

4.8 Cohen's Kappa Rate for MI classifiers 61

4.9 Holm and Wilcoxon tests for Cohen's Kappa rate 61

4.10 AUC for MI classifiers 62

4.11 Holm and Wilcoxon tests for AUC 62

4.12 Run Time (seconds) for MI classifiers 64

4.13 Overall ranks comparison for MI classifiers 64

5.1 Datasets used for OLLAWV Comparison 75

LIST OF FIGURES

Figure Page

2.1 A 2-dimensional example of different possible separating hyperplanes that correctly classify all the toy data points. 16

2.2 An illustration of the soft margin SVM solution on an example 2-dimensional non-linearly separable dataset. 16

2.3 Vapnik's ϵ-insensitivity loss function. 19

2.4 Support vector regression example solution. 19

3.1 SVR Flow Diagram. Firstly, the SVR method divides the MT dataset into m ST datasets, $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_m$. It then independently trains models, $h_1, h_2, \ldots, h_m$, for each ST dataset. 26

3.2 SVRRC Flow Diagram on a dataset with 3 targets. SVRRC first builds the 6 random chains of the target's indices (3 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. 28

3.3 SVRCC Flow Diagram on a sample dataset with 3 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. . . 29

3.4 Bonferroni-Dunn test for aCC 34

3.5 Bonferroni-Dunn test for MSE 35

3.6 Bonferroni-Dunn test for aRMSE 37

3.7 Bonferroni-Dunn test for aRRMSE 38

3.8 Bonferroni-Dunn test for Run Time 40

4.1 A summary of the steps performed by MIRSVM. The representatives are first randomly initialized and continuously updated according to the current hyperplane, which is found using a quadratic programming solver. Upon completion, the model is returned along with the optimal bag-representatives. 51

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. 52

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. 53

4.4 Bonferroni-Dunn test for Accuracy 58

4.5 Bonferroni-Dunn test for Precision 59

4.6 Bonferroni-Dunn test for Recall 60

4.7 Bonferroni-Dunn test for Cohen's Kappa rate 61

4.8 Bonferroni-Dunn test for AUC 62

4.9 Bonferroni-Dunn test for overall ranks comparison 64

5.1 A case of classifying 2-dimensional normally distributed data with different covariance matrices, (left) for 200 and (right) 2000 data points. The theoretical separation boundary (denoted as the Bayes Separation Boundary) is quadratic and is shown as the dashed black curve. The other two separation boundaries shown are the ones obtained by OLLAWV and SMO (implemented within LIBSVM), respectively. In this particular case (left), the difference between the OLLAWV boundary and the SMO boundary is hardly visible. The case presented on the right shows that, with an increase of training samples, the OLLAWV and SMO boundaries converge to the theoretical Bayesian solution. 72

5.2 A summary of the steps performed by OLLAWV. The model parameters ($\boldsymbol{\alpha}$, b, $\boldsymbol{S}$) and the algorithm variables ($\boldsymbol{o}$, t, wv, and yo) are first initialized. The worst-violator with respect to the current hyperplane is then found and the model parameters are then updated. Once no more violating samples are found, the model is returned. 74

CHAPTER 1

INTRODUCTION

In traditional classification and regression problems, learning algorithms attempt to correctly predict unknown samples by finding patterns between training samples and their outputs. Identifying these patterns is a non-trivial task due to many factors such as the high dimensionality of the data, as well as the 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 traditional methods have been extended to accommodate various types of data paradigms. Examples include *Multiple Target* (MT) learning and *Multiple Instance* (MI) learning. These emerging paradigms require algorithms to be robust and accommodate non-traditional data representations.

Multi-target learning is a challenging task that consists of creating predictive models for problems with multiple outputs. Specifically, MT learning is an approach to transfer learning that improves generalization by using domain information present in training data of multiple related targets as an inductive bias. This is done by learning in parallel, and what is learned for each target can be used to help other targets be learned better [7, 20, 69]. MT learning includes *multi-target regression* (MTR), which addresses the prediction of continuous targets, *multi-label classification* [83] which focuses on binary targets, and *multi-dimensional classification* which describes the prediction of discrete targets [13].

Multi-target prediction has the capacity to generate models representing a wide variety of real-world applications, ranging from natural language processing [43] to bioinformat-

ics [54]. A characteristic of the MT datasets used in these applications is that they are generated by a single system, indicating that the nature of the outputs captured has some structure. Even though modeling the multi-variate nature and possible complex relationships between the target variables is challenging, they are more accurately represented by a multi-target model [20, 32].

Several methods have been proposed for solving such multi-target tasks and can be categorized into two groups. The first being *problem transformation* 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 which adapt existing traditional algorithms to predict all the target variables simultaneously [13]. Using *problem transformation* algorithms for a domain of t target variables, t predictive models must be constructed, each predicting a single-target variable. Prediction for an unseen sample would be obtained by running each of the t single-target models and concatenating their results. Conversely, when using *algorithm adaptation* algorithms for the same domain of t target variables, a single model would need to be constructed which would output all t predictions.

It is known that *algorithm adaptation* methods outperform *problem transformation* methods. 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 [7, 20]. Single-target techniques, on the other hand, eliminate any possibility of learning from the possible relationships between the target variables because a single, independent model is trained for each target separately. Another advantage of MT techniques is model interpretability [8]. A single multi-target model is highly more interpretable than a series of single-target models. Not only is a single MT model more interpretable, but it could also be considerably more computationally efficient to train, rather than training multiple single-target models individually [33].

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 [40], human action recognition [82], and drug activity prediction [27]. The difference between MIL and traditional learning is the nature of the data. In the multi-instance 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. [27], the standard MI assumption states that a bag is labeled positive if and only if it contains at least one positive instance. Other hypotheses and frameworks have been proposed by Foulds and Frank [34] to encompass a wider range of applications with MI data, but for the scope of this thesis, we will focus on the standard MI assumption.

One of the major complexities associated with MIL is the ambiguity of the relationship between a bag label and the instances within the bag. This 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 [62]. 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 [55]. 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 [19].

Our MTR and MIL proposals aim to deal with these drawbacks that exist in both paradigms using support vector machines. Support vector machines (SVM), proposed by Cortes and Vapnik [26], 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 [61], a method for solving the L1-SVM's Quadratic Programming (QP) task.

In this thesis, various approaches are devised for solving machine learning problems for various types of learning paradigms using traditional support vector machine solvers. Specifically, novel algorithms are proposed for solving multi-target regression (a subset of multi-target learning) and multi-instance classification problems.

1.1 Contributions of the Proposal

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 proposal aims to investigate the performance changes when building a regression model using two distinct *algorithm adaptation* 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 *algorithm adaptation* approach, inspired by the chaining classification method, Ensemble of Random Chains Corrected (ERCC) [66], to investigate the effects and advantages of exploiting correlations among target variables during model training, in the context of regression problems. The main issues to be investigated with this approach are the *randomness* of the created chains because they might not capture of correlations between the targets, as well as the time taken to build all the regressors in the ensemble.

- 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.

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) [14, 26, 45, 63, 64]. 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 [64, 75]. This is the basis for developing the maximal margin classifier [75]. Based on the work of Aizerman et. al. [1], Boser et. al. [14] generalized the linear algorithm to the non-linear case. Then, Cortes and Vapnik [26] 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 et. al. [76] 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 linear *Hard Margin* classifier, and the nonlinear *Soft Margin SVM*. Next, the concept of Support Vector Regression (SVR) is introduced. Afterwards, popular methods for solving the SVM problem are presented and their advantages and problems are discussed.

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.

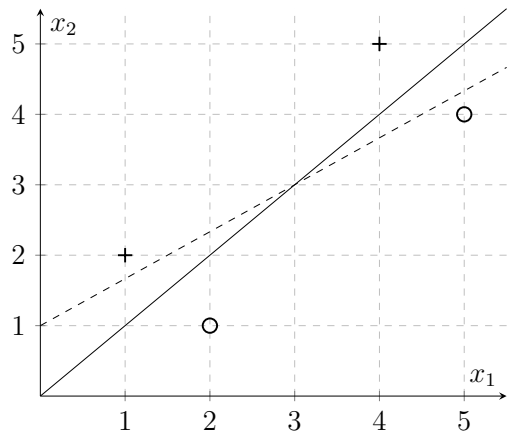

Fig. 2.1. A 2-dimensional example of different possible separating hyperplanes that correctly classify all the toy data points.

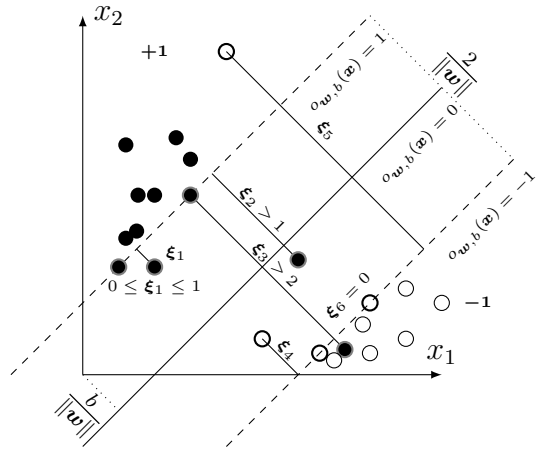

Fig. 2.2. An illustration of the soft margin SVM solution on an example 2-dimensional non-linearly separable dataset.

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,

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

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 problem is defined as follows,

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

where the penalty parameter $C \in \mathbb{R}$ controls the trade-off between margin maximization and classification error minimization, penalizing large norms and errors.

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 [1], as shown in Equation 2.6:

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

where $\phi(\cdot)$ represents a mapping function from the original feature space to a higher dimen-

sional 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 [30, 74], 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 $\mathcal{S} = \big\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n) \in \mathbb{R}^d \times \mathbb{R}\big\}$, 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

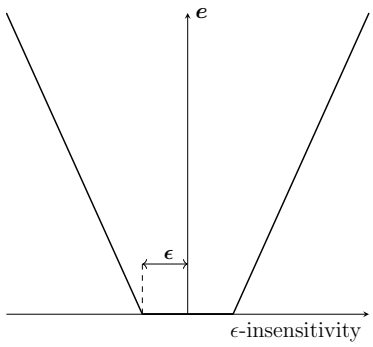

Fig. 2.3. Vapnik's ϵ-insensitivity loss function.

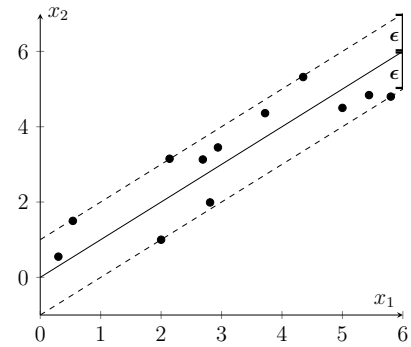

Fig. 2.4. Support vector regression example solution.

in Figure 2.4. If the predicted value is within the tube, no loss is incurred [45]. 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:

$$\begin{aligned} \min_{(\boldsymbol{w},b,\boldsymbol{\xi},\boldsymbol{\xi}^*)} \quad & \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) \\ \text{s.t.} \quad & y_i - \boldsymbol{w}\cdot\boldsymbol{x}_i - b \le \xi_i + \epsilon,\ \forall i = \{1,\ldots,n\} \\ & \boldsymbol{w}\cdot\boldsymbol{x}_i + b - y_i \le \xi_i^* + \epsilon,\ \forall i = \{1,\ldots,n\} \\ & \xi_i, \xi_i^* \ge 0,\ \forall i = \{1,\ldots,n\}. \end{aligned} \tag{2.10}$$

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 [63]. It is as follows:

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

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

2.3 Methods for Solving the SVM Problem

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

The first attempts at speeding up the training time and decreasing algorithm memory requirements were aimed at decomposing the underlying SVMs quadratic programming problem. First, Boser et al. [14] implemented Vapnik's *chunking* method. *Sequential Minimal Optimization* by Platt [61] and its improvement by Keerthie et al. [49] are an alternative approach to decomposing the QP problem, and are implemented in popular, widely used software package LIBSVM [22]. SMO is an iterative procedure that divides the SVM dual problem into a series of sub-problems, which are solved analytically by finding the optimal $\boldsymbol{\alpha}$ values that satisfy the Karush-Kuhn-Tucker conditions [17]. Although SMO is guaranteed to converge, heuristics are used to choose $\boldsymbol{\alpha}$ 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 [63].

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 [9] and minimum enclosing balls such as *Core Vector Machines* (CVM) [71]. Tsang et al. [70] then 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 [68], 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) [86] is an efficient approach for solving the SVM problem, shown to

be faster than SMO and equal in terms of accuracy [48]. NNISDA is an iterative algorithm that finds a solution to the L2-SVM using coordinate descent, inspired by *Iterative Single Data Algorithm* (ISDA) [42] which was originally introduced by Kecman et al. [46].

Recently, several authors have proposed the use of a standard stochastic gradient descent (SGD) approach for SVMs to optimize large-scale learning problems [39, 50, 63, 64, 65]. Kivinen et al.[51] and Bousquet and Bottou [16] showed that stochastic algorithms can be both the fastest, and have the best generalization performances. Shalev-Shwartz and Ben-David [64] have also demonstrated that the basic SGD algorithm is very effective when data are sparse, taking less than linear [$O(d)$] time and space per iteration to optimize a system with d parameters. It can greatly surpass the performance of more sophisticated batch methods on large data sets. The previously mentioned approaches are extended variants of a classic kernel perceptron algorithm [25].

Notable representatives of this method of learning include the *Naïve Online R Minimization Algorithm* (NORMA) by Kivinen et al. [51] and the *Primal Estimated Sub-Gradient SOlver for SVM* (PEGASOS) by Shalev-Shwartz et al. [65]. 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 [51]. 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) [44] is a unification, simplification, and expansion of the somewhat similar approaches presented in [39, 50, 63, 64, 65] and [47, 57, 59]. 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 [25] 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 [44], but for the scope of this proposal 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 disadvantage of kernelized online algorithms is that the training time for each update increases superlinearly with the number of samples.

CHAPTER 3

MULTI-TARGET SVR USING MAXIMUM CORRELATION CHAINS

This contribution 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) [66]. 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.

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 $\boldsymbol{Y} \in \mathbb{R}^{n \times m}$.

3.1.2 Multi-Target Regression

In the multi-target learning paradigm, the *problem transformation* approach involves 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 is considered as a baseline for measuring model performance [66]. Many *problem transformation* methods have been proposed for solve multi-target problems, however, the main issue with this type of approach is that the relationships between the targets is lost once independent models are built for each target. Examples of *problem transformation* approaches include Linear Target Combinations for MT Regression [73], and Multi-Objective Random Forests (MORF) [52].

The RC, MTS, MTSC, ERC, and ERCC methods are introduced by Spyromitros et. al. in [66]. The idea behind these algorithms was to investigate whether advances in multi-label learning can be successfully used in a multi-target regression setting and shed light on modeling target dependencies. These methods involve two stages of learning, the first being building ST models. 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.

The two stages of training in MTS involve firstly, 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 ERC method is somewhat similar to the MTS method. In the training of a Regression Chain (RC) model, a random chain, or 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 [67]. If the default, ordered chain is $C = \{\boldsymbol{Y}_1, \boldsymbol{Y}_2, \ldots, \boldsymbol{Y}_m\}$, the first model $h_1 : \boldsymbol{X} \rightarrow \mathbb{R}$ 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. So 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 in 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. Also, prediction error in the earlier stages of the models will be propagated as the rest of the models are trained, which is why the Ensemble of Regressor Chains was proposed in [66]. Instead of a single chain, k 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.

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 [66], 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 Regression Chains Corrected (RCC) and Multi-Target Stacking Corrected (MTSC).

The 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 algorithms were tested and compared using Bagging of 100 regression trees as their base regression algorithm with ERC and ERCC ensemble size of

10, and 10-fold cross-validation. 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. These methods can be found and used through the open-source Java library, Mulan [72]; to replicate the results found in [66].

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

$$\begin{tikzcd}
 & \mathcal{D}_1 : [\boldsymbol{X}][\boldsymbol{Y}_1] \arrow[r] & h_1 : \mathcal{D}_1 \to \hat{\boldsymbol{Y}}_{\boldsymbol{1}} \\
 & \mathcal{D}_2 : [\boldsymbol{X}][\boldsymbol{Y}_2] \arrow[r] & h_2 : \mathcal{D}_2 \to \hat{\boldsymbol{Y}}_{\boldsymbol{2}} \\
\mathcal{D} : [\boldsymbol{X}][\boldsymbol{Y}] \arrow[uur, bend left] \arrow[ur, bend left] \arrow[dr, bend right] & \vdots & \\
 & \mathcal{D}_m : [\boldsymbol{X}][\boldsymbol{Y}_m] \arrow[r] & h_m : \mathcal{D}_m \to \hat{\boldsymbol{Y}}_{\boldsymbol{m}}
\end{tikzcd}$$

Fig. 3.1. SVR Flow Diagram. Firstly, the SVR method divides the MT dataset into m ST datasets, $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_m$. It then independently trains models, $h_1, h_2, \ldots, h_m$, for each ST dataset.

Algorithm 3.1 MT Support Vector Regression (SVR)

Input: Training dataset $\mathcal{D}$
Output: ST models $h_j, j = 1, \ldots, m$

```
for j = 1 to m do
    D_j = {X, Y_j}                          ▷ Get ST data
    h_j : X → ℝ                             ▷ Build ST model for the j^th target
end for
return h_j, j = 1, ..., m
```

Algorithm 3.2 Build Chained Model

Input: Training dataset $\mathcal{D}$, random chain $\boldsymbol{C}$
Output: A chained model $h_j, j = \{1, \ldots, m\}, c \leq 10$

```
1: D_1 = {X, Y_{C_1}}                          ▷ Initialize first dataset
2: for j = 1 to m do                           ▷ For each target in chain C
3:     h_j : D_j → ℝ                           ▷ Train model on appended dataset
4:     if j < m then
5:         D_{j+1} = {D_j, Y_{C_j}}            ▷ Append new target in chain to dataset
6:     end if
7: end for
8: return h_j, j = 1, ..., m
```

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 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 SVRCC 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

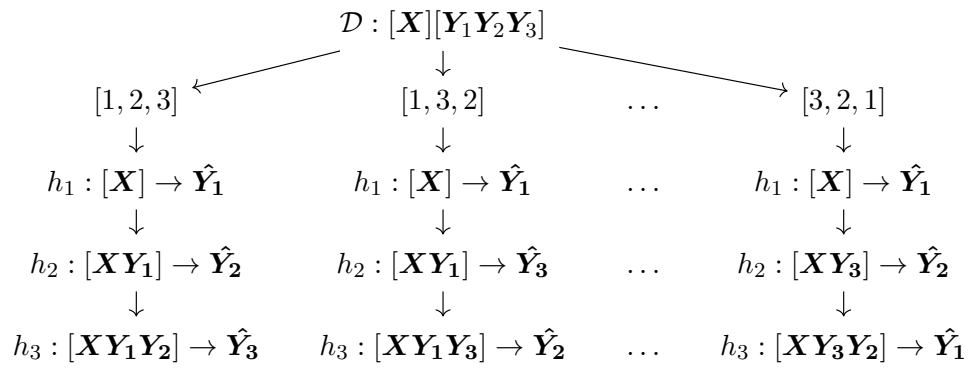

Fig. 3.2. SVRRC Flow Diagram on a dataset with 3 targets. SVRRC first builds the 6 random chains of the target's indices (3 examples are shown). It then constructs a chained model by proceeding recursively over the chain, building a model, and appending the current target to the input space to predict the next target in the chain.

Algorithm 3.3 MT SVR with Random-Chains (SVRRC)

Input: Training dataset $\mathcal{D}$, c random chains $\mathcal{C}$
Output: An ensemble of chained models $h_{\mathcal{C}}$

```
for each C ∈ 𝒞 do                                  ▷ For each random chain
    h_C = build chained model(𝒟, C)                ▷ build a chained model for chain C
end for
return h_𝒞
```

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 $(m!)\times m$ 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

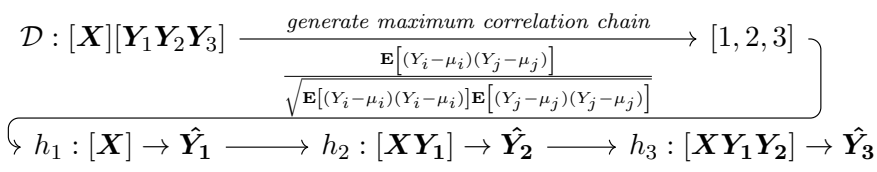

Fig. 3.3. SVRCC Flow Diagram on a sample dataset with 3 targets. SVRCC first finds the direction of maximum correlation among the targets and uses that order as the only chain. It then constructs the chained model as done in SVRRC.

Algorithm 3.4 MT 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: $\boldsymbol{C} = \text{sort}\,(\boldsymbol{C}, \textbf{decreasing})$ ▷ Sort sums in decreasing order
4: $h_{\boldsymbol{C}} = \text{build chained model}(\mathcal{D}, \boldsymbol{C})$ ▷ build a chained model for max correlation chain $\boldsymbol{C}$
5: **return** $h_{\boldsymbol{C}}$

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.

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 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, $\mathbf{\Sigma}$, is first calculated as shown in Equation 3.1:

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

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:

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

It will describe 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 [72], as well as the UCI Machine Learning Repository [5]. Information on the 24 datasets used is summarized in Table 3.1, where the number of samples, attributes (dimensionality), and targets are shown.

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 [66]. 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] [72], which was built on top of Weka[2] [38]. LIBSVM's Epsilon-SVR [22] implementation was used as the base SVR model. The parameters experimented with for

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

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

Table 3.1. Multi-Target (MT) Regression datasets

Dataset	Samples (n)	Attributes (d)	Targets (m)
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

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).

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

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

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

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 [66]. This includes the following. The ST base-line model used was Bagging [18] of 100 regression trees [80]. 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 ensemble size of 100 trees was used for MORF, and the rest of its parameters were set as recommended by [53].

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 3.2, 3.4, 3.6, 3.8, and 3.10 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 [31] 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 [79] tests were run for each of the result metrics to compute multiple pairwise comparisons among the algorithms used in the experimental study. Tables 3.3, 3.5, 3.7, 3.9, and 3.11 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 3.2 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

Table 3.2. 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**

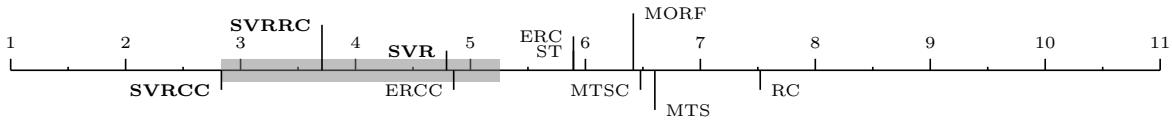

Fig. 3.4. Bonferroni-Dunn test for aCC

Table 3.3. 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	$4.0E^{-4}$	$3.2E^{-1}$	$5.0E^{-2}$

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 3.3 provides complementary analysis of the results. 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 significantly 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 3.4. 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**

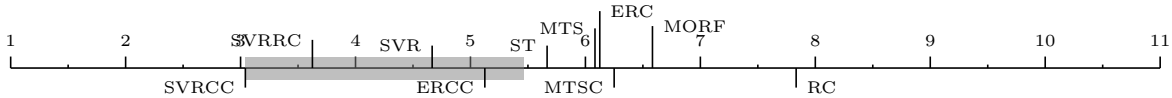

Fig. 3.5. Bonferroni-Dunn test for MSE

Table 3.5. 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 3.4 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 3.5, 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 3.6 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 3.7, 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 3.6. 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**

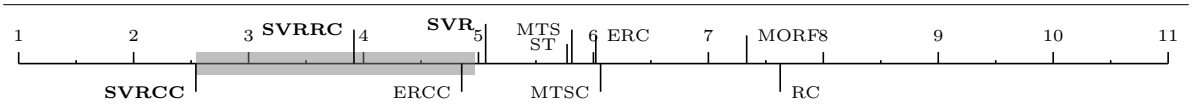

Fig. 3.6. Bonferroni-Dunn test for aRMSE

Table 3.7. 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	$2.0E^{-5}$	$1.2E^{-1}$	$5.0E^{-2}$

3.4.4 Average Relative Root Mean Square Error

Table 3.8 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.

Table 3.8. 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**

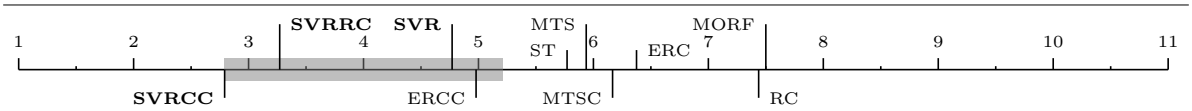

Fig. 3.7. Bonferroni-Dunn test for aRRMSE

Table 3.9. 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	$3.8E^{-2}$	$5.8E^{-1}$	$5.0E^{-2}$

According to the Wilcoxon test, shown in Table 3.9, 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 3.10 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 3.11, 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, 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.

Table 3.10. 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

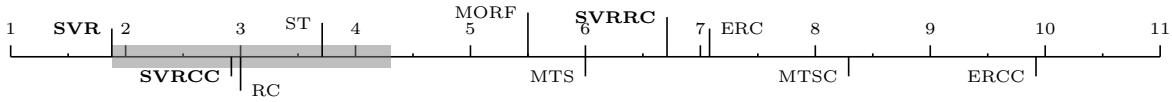

Fig. 3.8. Bonferroni-Dunn test for Run Time

Table 3.11. 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	$8.3E^{-3}$

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 rela-

tionship 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 [58].

CHAPTER 4

MULTI-INSTANCE SVM USING BAG-REPRESENTATIVES

This contribution proposes a novel support vector machine multiple-instance formulation and presents an algorithm with a bag-representative selector that trains the SVM based on bag-level information, named MIRSVM. The contribution 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. The experimental study evaluates and compares the performance of this proposal against 11 popular 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 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 the paper, and reviews related works on multi-instance learning and multi-instance support vector machines.

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

The difference between MIL and traditional learning is the nature of the data. In the traditional binary classification setting, the goal is to learn a model that maps input samples to labels, $f : \mathbb{R}^{m \times d} \rightarrow Y^m \in \{-1, +1\}$. 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. 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, $f(\mathcal{B}_{n+1}) \rightarrow Y_{n+1}$ [3].

In order to build a classifier without any knowledge of the individual training instance labels, Dietterich et. al. [27] 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 [34]. A recent review [3] 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. One traditional method in this category is the Axis-Parallel Rectangle (APR) [27], 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 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 method is the Diverse Density (DD) [55] framework which 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 [84] propose a similar framework that iteratively maximizes the DD measure. Auer and Ortner [6] 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 [6] 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 [19, 23, 24] assume the positive instances form a cluster, which may not be the case. Alternatively, Fu et al.[36] models the distribution of negative bags with Gaussian kernels, which can prove difficult when the quantity of data is limited.

Some methods in literature, such as Simple-MI [28], transform the MI dataset to a traditional instance-level dataset. Simple-MI represents each bag with the mean vector of the instances within it. The major disadvantage of these types of approaches is that they assume the distribution the instances in positive bags is positive, when it may not be.

An extension of traditional single-instance k-nearest neighbors method (KNN) was pro-

posed by Wang and Zucker [77] 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.

Blockeel et al. [12] 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 [10] 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. Other popular methods include MIBoost [38], MIWrapper [35], and Two-Level Classifier (TLC) [78].

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 [29].

Andrews et. al. [4] 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. [4] 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.

One of the first bag-level approaches to the multi-instance SVM problem was proposed by Gärtner et. al. [37], who defined a bag-level kernel. A bag-level kernel determines the similarity between two bags in a higher dimensional space. Blaschko et. al. [11] 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.

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 [3]. 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 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:

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

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

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

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.

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

At optimality [17], $\bigtriangledown_{w,b,\xi}\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = 0$ and the following conditions are met:

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

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

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

By substituting Equations (4.5), (4.6), and (4.7) into the Lagrangian in (4.4), the dual MIRSVM formulation becomes the following:

$$\max_{\boldsymbol{\alpha},\boldsymbol{\beta}} \sum_I \alpha_I - \frac{1}{2} \sum_I \sum_{K \in I} \sum_{j=1}^{d} \alpha_I \alpha_K Y_I Y_K x_{s_I j} x_{s_K j} \tag{4.8}$$

$$\text{s.t.} \sum_I \alpha_I Y_I = 0 \tag{4.8a}$$

$$\alpha_I + \beta_I = \frac{C}{n}, \, \forall I \in \{1, \ldots, n\} \tag{4.8b}$$

$$\alpha_I \geq 0, \, \forall I \in \{1, \ldots, n\} \tag{4.8c}$$

$$\beta_I \geq 0, \, \forall I \in \{1, \ldots, n\}, \tag{4.8d}$$

where s_I is computed for each bag, as shown in Equation (4.9):

$$s_I = \underset{i \in I}{\operatorname{argmax}} (\sum_{K \in I} \sum_{j=1}^{d} \alpha_K Y_K x_{s_K j} x_{ij} + b), \, \forall I \in \{1, \ldots, n\}. \tag{4.9}$$

The implicit constraints (4.8b) through (4.8d) imply three possible cases for α_I values:

1. If $\alpha_I = 0$, then $\beta_I = C/n$ and $\xi_I = 0$, implying the instance is correctly classified and outside the margin.

2. If $0 < \alpha_I < C/n$, then $\beta_I > 0$ and $\xi_I = 0$, indicating that the instance sits on the margin boundary, i.e. is an *unbounded support vector*.

3. If $\alpha_I = C/n$, then $\beta_I = 0$ and there is no restriction for $\xi_I \geq 0$. 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 otherwise.

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

$$\max_{\boldsymbol{\alpha}} \sum_I \alpha_I - \frac{1}{2} \sum_I \sum_{K \in I} \alpha_I \alpha_K Y_I Y_K \mathcal{K}\left(\boldsymbol{x}_{s_I}, \boldsymbol{x}_{s_K}\right) \tag{4.10}$$

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

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

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 (4.11), where σ is the Gaussian shape parameter.

$$\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||^2}{2\sigma^2}} \tag{4.11}$$

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.

$$\boldsymbol{o}_I = \mathcal{K}(\mathcal{B}_I, \boldsymbol{X}_S) * (\boldsymbol{\alpha} \cdot \boldsymbol{Y}_S) + b \tag{4.12}$$

The bias term b is calculated as shown in Equation (4.13), where $\boldsymbol{sv}$ is the vector of support vector indices and n_{sv} is the number of support vectors [42].

$$b = \frac{1}{n_{\boldsymbol{sv}}} \sum_{\boldsymbol{sv}} Y_{\boldsymbol{sv}} - \mathcal{K}(\boldsymbol{X}_{\boldsymbol{sv}}, \boldsymbol{X}_{\boldsymbol{sv}}) * (\boldsymbol{\alpha}_{\boldsymbol{sv}} \cdot Y_{\boldsymbol{sv}}) \tag{4.13}$$

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 (4.9).

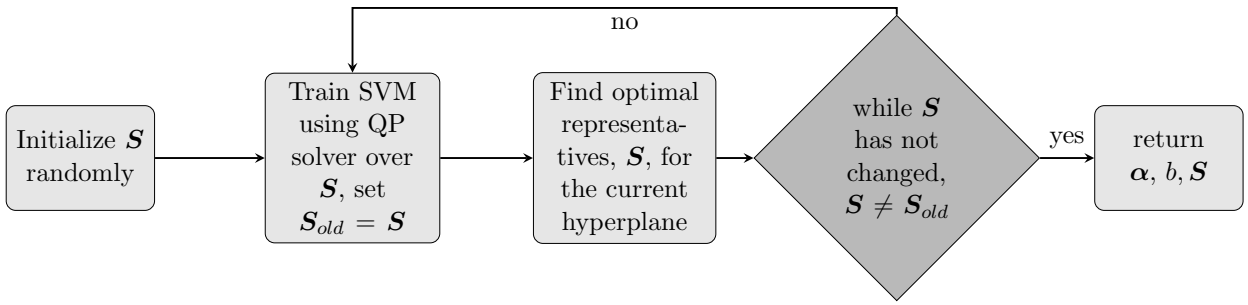

Fig. 4.1. A summary of the steps performed by MIRSVM. The representatives are first randomly initialized and continuously updated according to the current hyperplane, which is found using a quadratic programming solver. Upon completion, the model is returned along with the optimal bag-representatives.

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 $\boldsymbol{S}$

```
for I ∈ {1, ..., n} do
    S_I ← rand(|B_I|, 1, 1)                          ▷ Assign each bag a random instance
end for
while S ≠ S_old do
    S_old ← S
    X_S ← X(S), Y_S ← Y(S)                           ▷ Initialize the representative dataset
    G ← (Y_S × Y_S) · K(X_S, X_S, σ)                 ▷ Build Gram matrix
    α ← quadprog(G, −1^n, Y_S, 0^n, 0^n, C^n)        ▷ Solve QP Problem
    sv ← find(0 < α ≤ C)                             ▷ Get the support vector indices
    n_sv ← count(0 < α ≤ C)                          ▷ Get the number of support vectors
    b ← (1/n_sv) Σ_{i=1}^{n_sv} (Y_sv − G_sv * (α_sv · Y_sv))   ▷ Calculate the bias term
    for I ∈ {1, ..., n} do
        G_I ← (Y_I × Y_S) · K(B_I, X_S, σ)
        S_I ← argmax_{i∈I} (G_I * α + b)             ▷ Select optimal bag-representatives
    end for
end while
```

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.

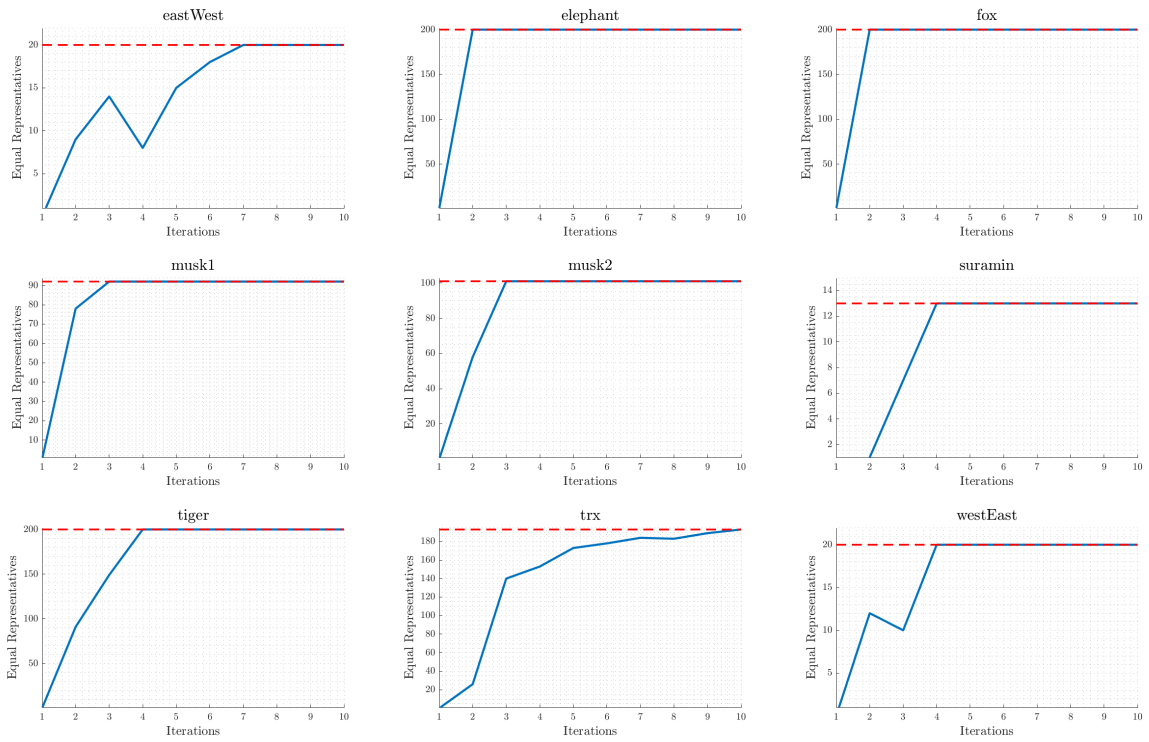

Fig. 4.2. Bag representative convergence plots on 9 datasets. The blue line shows the number of bag representatives that are equal from one iteration to the next. The red dashed line represents the total number of bags.

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 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

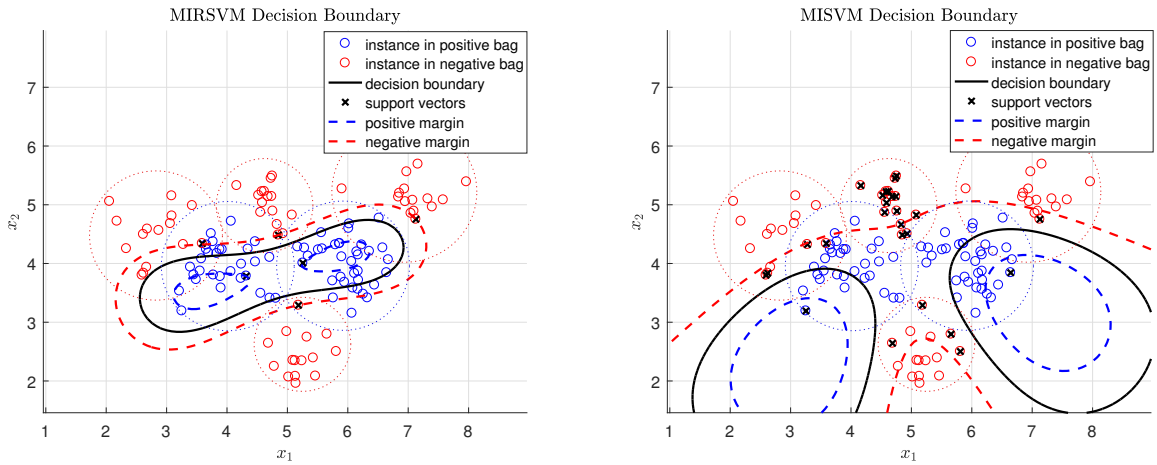

Fig. 4.3. Difference between MIRSVM and MISVM on a random 2-dimensional toy dataset. Note the differing number of support vectors produced by the two methods. MIRSVM has 6, one for each bag, and MISVM has 29. Also note the smoother representation of the data distribution given by MIRSVM's decision boundary, 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.

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.

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 5.1 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

Table 4.1. 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

obtained from the Weka[1] [38] and KEEL[2] [2] 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. 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] [81], 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

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

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

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

which was made available by Zhou et al.[4] and tested in MATLAB.

We have compared the models trained on the different hyper-parameters 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 hyper-parameters were chosen from the following 6×6 possible combination runs, shown in Equations (4.14a) and (4.14b), referred to as (4.14).

$$C \in \{0.1, 1, 10, 100, 1000, 10000\} \tag{4.14a}$$

$$\sigma \in \{0.1, 0.5, 1, 2, 5, 10\} \tag{4.14b}$$

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 (4.15a), Precision (4.15b), Recall (4.15c), Cohen's kappa rate (4.15d), and Area under ROC curve (AUC) (4.15e). 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 (4.15d), 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

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

the trade-off between the true positive rate, or recall, and the false positive rate, as shown in Equation (4.15e). 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 (4.15) 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.

The results for the following are shown in Tables 4.2, 4.4, 4.6, 4.8, 4.10, and 4.12.

$$\text{Accuracy} \qquad \frac{TP + TN}{n'} \tag{4.15a}$$

$$\text{Precision} \qquad \frac{TP}{TP + FP} \tag{4.15b}$$

$$\text{Recall} \qquad \frac{TP}{TP + FN} \tag{4.15c}$$

$$\text{Cohen's Kappa Rate} \qquad \frac{n' - \dfrac{(TP + FN) * (TP + FP)}{n'}}{1 - \dfrac{(TP + FN) * (TP + FP)}{n'}} \tag{4.15d}$$

$$\text{Area Under ROC Curve} \qquad \frac{1 + \dfrac{TP}{TP + FN} - \dfrac{FP}{FP + TN}}{2} \tag{4.15e}$$

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 (4.15) are presented in the last row of the results tables, and the lowest (best) rank value is typeset in bold. Table 4.13 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 [31] 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 (4.15), as well as the meta-rank results in Table 4.13. The Holm (multiple) and Wilcoxon (pairwise) rank-sum post-hoc tests [41] 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 4.3, 4.5, 4.7, 4.9, and 4.11 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, 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 3.3 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 grey bar in Figure 4.4, perform statistically worse than MIRSVM. Table 3.3 shows the p-values of the Holm and Wilcoxon tests and their results com-

Table 4.2. 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	0.5625	**0.9649**	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

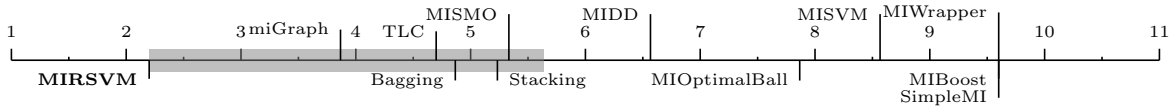

Fig. 4.4. Bonferroni-Dunn test for Accuracy

Table 4.3. 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

plement 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 miGraph, 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 4.4. 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

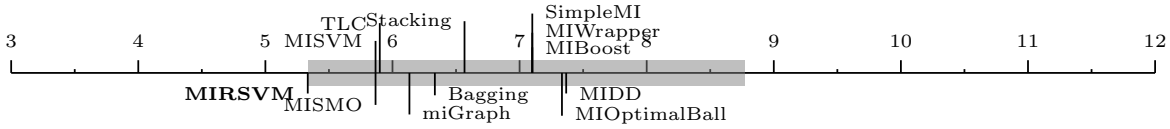

Fig. 4.5. Bonferroni-Dunn test for Precision

Table 4.5. 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 4.6. 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

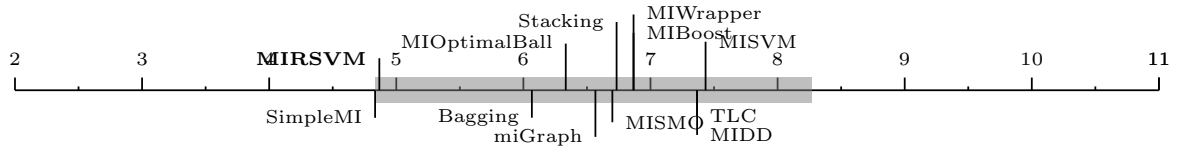

Fig. 4.6. Bonferroni-Dunn test for Recall

Table 4.7. 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.

Table 4.8. 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	0.5900	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

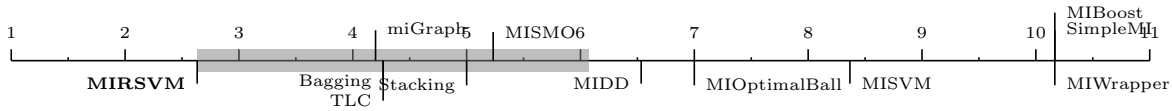

Fig. 4.7. Bonferroni-Dunn test for Cohen's Kappa rate

Table 4.9. 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

4.4.3 Cohen's Kappa Rate

Table 4.8 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 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 4.9 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.

4.4.4 Area Under ROC Curve

Table 4.10 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 MIWrapper obtain the worst results. Their AUC score indicates random predictor behavior,

Table 4.10. 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

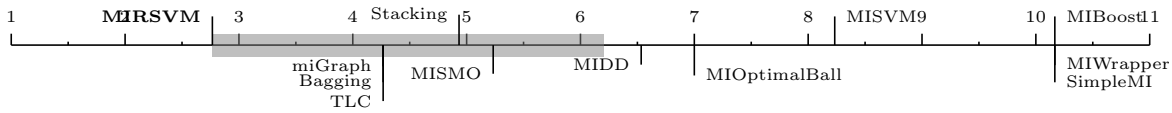

Fig. 4.8. Bonferroni-Dunn test for AUC

Table 4.11. 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

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 4.11 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 4.12 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 has been shown to be random and not as effective as the bag-level methods.

Table 4.13 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

Table 4.12. 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 4.13. 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

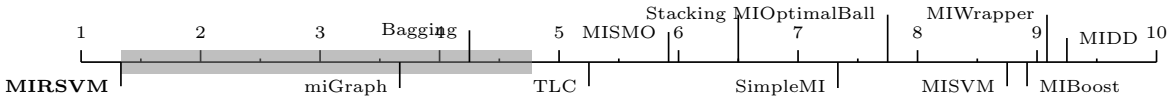

Fig. 4.9. Bonferroni-Dunn test for overall ranks comparison

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

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 [56].

CHAPTER 5

NOVEL ONLINE SVM USING WORST-VIOLATORS

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

5.1 Online Learning Background

The scope of this paper is concerned with developing a unique learning algorithm for such large data problems without the use of parallelization techniques and distributed systems.

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, such that $\boldsymbol{Y} \in \{-1, 1\}^n$. In the non-binary (more

than two classes) classification cases, $\boldsymbol{Y} \in \mathbb{Z}^n$. Let $\boldsymbol{X} \in \mathcal{D}$ be a matrix consisting of n samples that are d-dimensional, $\boldsymbol{X} \in \mathbb{R}^{n \times d}$.

5.1.2 Regularized Loss Minimization

5.1.3 Stochastic Gradient Descent

Gradient Descent

Subgradients

Stochastic Gradient Descent

5.1.4 Solving the SVM Problem with Stochastic Gradient Descent

5.2 Novel 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 [44] 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 5.1 shows the method for selecting the worst-violator, where $yo \in \mathbb{R}$ is the error value, $wv \in \{1, \ldots, n\}$ is the error value's index, $\boldsymbol{o} \in \mathbb{R}^n$ is the decision function output, and $\neg$ is the 'not' symbol. For the L1-SVM, an error value will always be negative which is why the minimum function is used (i.e. the most negative output value or incorrectly classified sample). The worst violating sample becomes the model's support vector because its weight is updated and non-zero. Therefore, the number of iterations of OLLAWV is equal to the number of support vectors that the resulting model has. This is an interesting property of OLLAWV; if the number of iterations is set beforehand, one is implicitly setting a bound on the number of support vectors.

$$[yo, wv] = \min \{y_{wv} \cdot o_{wv}\}, \forall wv \in \{\neg \boldsymbol{S}\} \tag{5.1}$$

Algorithm 5.1 lists OLLAWV's pseudocode and Figure 5.2 illustrates the steps taken by OLLAWV. First, the model parameters ($\boldsymbol{\alpha}$, b, $\boldsymbol{S}$) and the algorithm variables ($\boldsymbol{o}$, iteration counter (t), initial worst-violator index wv and it's error yo) are first initialized. The worst-violator with respect to the current hyperplane is then found and the model parameters are updated. Once no more violating samples are found or the maximum number of iterations is reached, the model is returned.

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

$$\min_{\boldsymbol{w} \in \mathcal{H}_o \times \mathbb{R}} R \;=\; \frac{1}{2}||\boldsymbol{w}||^2 + C \sum_{i=1}^{n} \max\left\{0, 1 - y_i o_{(w)}(\boldsymbol{x}_i)\right\}, \tag{5.2}$$

where $o_{(w)}(\boldsymbol{x}_i) = \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle$ is a linear predictor, with the bias term excluded for simplicity. Note that the loss function used in Equation 5.2 is non-differentiable, but it has a subgradient due to a knick-point at $yo = 1$. The loss function's gradient after the knick-point equals zero, which leads to a sparse model. Hence, when the value of $yo \geq 1$ the loss is zero, and

for $yo < 1$ the loss increases linearly. The subgradient of the above cost function is given by:

$$\frac{\partial R}{\partial \boldsymbol{w}} = \begin{cases} \boldsymbol{w} - C\sum_{i=1}^{n} y_i \boldsymbol{x_i} & y_i o_i < 1 \\ \boldsymbol{w} & \text{otherwise.} \end{cases} \tag{5.3}$$

In the stochastic case, the calculation of the gradient needed for the weight update, is *pattern based*, not *epoch based* as in batch gradient descent. It has been shown [45] 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 5.3 becomes:

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

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

where $\eta > 0 \in \mathbb{R}$ is the learning rate. According to the Representer Theorem, a vector $\boldsymbol{\alpha} \in \mathbb{R}^n$ exists such that $\boldsymbol{w} = \sum_{i=1}^{n} \alpha_i \phi(\boldsymbol{x}_i)$ is an optimal solution to Equation 5.2, where $\phi(\cdot)$ is a mapping from feature space to Hilbert space [64]. On the basis of the Representer Theorem, Equation 5.2 can be optimized with respect to $\boldsymbol{\alpha}$ instead of $\boldsymbol{w}$. By expressing $\boldsymbol{w}$ this way and mapping input sample $\boldsymbol{x_i}$ to $\phi(\boldsymbol{x_i})$, the kernelized SGD update becomes:

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

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

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

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

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

$$\forall i : \alpha_i \leftarrow \alpha_i + \eta(Cy_i - \alpha_i) \tag{5.4}$$

If the bias term b is included in Equation 5.2, it's stochastic update is as follows:

$$\forall i : b \leftarrow b + \eta \frac{Cy_i}{n} \tag{5.5}$$

In this experimental study, $\eta = 2/\sqrt{t}$ is used, where t is the current iteration; however, other learning rates such as $\eta = 1/t$ can also be used. Let $\Lambda = \eta C y_i$ and $\mathrm{P} = \eta \alpha_i$ be the update parameters for OLLAWV, and the $\boldsymbol{\alpha}$ update can be expressed as: $\alpha_i \leftarrow \alpha_i + (\Lambda - \mathrm{P})$. Note that Λ is the update resulting from the loss function and P is derived from the regularizer term in Equation 5.2. In the case of OLLAWV, $\mathrm{P} = 0$ because samples are never updated more than once and their initial α value is always 0. It is important to note that in OLLAWV's case, Λ never equals 0 because the samples being updated are worst-violators, meaning they are misclassified or incur some loss. The values of the decision function (output vector $\boldsymbol{o} \in \mathbb{R}^n$ in Algorithm 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. From Equation 2.7, the output vector update becomes the following:

$$\boldsymbol{o} \leftarrow \boldsymbol{o} + \Lambda * \mathcal{K}\left(\boldsymbol{x}_{\neg \boldsymbol{S}}, \boldsymbol{x}_{wv}, \gamma\right) + B \tag{5.6}$$

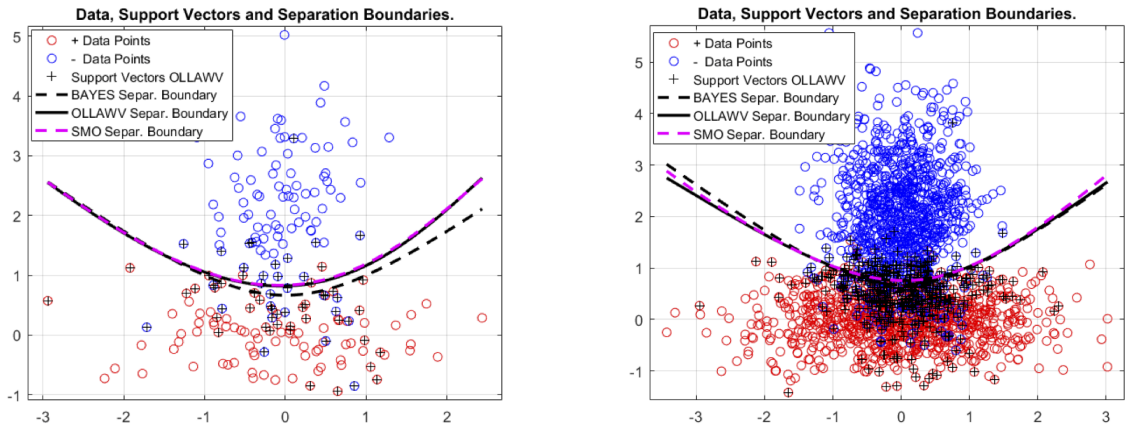

Fig. 5.1. A case of classifying 2-dimensional normally distributed data with different covariance matrices, (left) for 200 and (right) 2000 data points. The theoretical separation boundary (denoted as the Bayes Separation Boundary) is quadratic and is shown as the dashed black curve. The other two separation boundaries shown are the ones obtained by OLLAWV and SMO (implemented within LIBSVM), respectively. In this particular case (left), the difference between the OLLAWV boundary and the SMO boundary is hardly visible. The case presented on the right shows that, with an increase of training samples, the OLLAWV and SMO boundaries converge to the theoretical Bayesian solution.

where $\mathcal{K}(\cdot)$ is the Gaussian radial basis function (RBF) kernel, $\gamma \in \mathbb{R}$ is it's parameter, and $B = (\Lambda * \beta)/n$ denotes the bias update. Only non-support vector output values are calculated per iteration, as denoted by $\boldsymbol{x}_{\neg \boldsymbol{S}}$ in the kernel function, because samples are never selected to be updated more than once. Because the output values scale with the C value, the stopping criteria for OLLAWV is also set to scale with C, rather than the classic formulation $y_i o_i \geq 1$. If the value of C is very large, $y_i o_i$ will never be greater than 1 and the algorithm will never terminate. Therefore, the stopping criteria is set to be $y_i o_i \geq M$, where $M \in \mathbb{R}$ is a scaled value of C. For the B calculation in Equation 5.6, $\beta \in \{0, 1\}$ indicates whether the bias term is to be used. If b is not a part of the model, it should be omitted from Equations 2.7 and 5.6 by setting $\beta = 0$, otherwise $\beta = 1$.

OLLAWV is a stochastic gradient method (SGM) that has a convex cost function. Its learning rate coefficient can decrease linearly or semi-linearly during the learning stage.

Hence, OLLAWV shares the complexity characteristics of SGM methods. Primarily, it can achieve linear convergence, making it a particularly convenient and practical method for solving very large machine learning problems. OLLAWV also works over a cost function without local minima, always leading towards the global minimum, even though it stops the learning process as soon as all samples are outside the prescribed margin. Figure 5.1 shows the decision boundaries achieved by OLLAWV versus SMO (implemented within LIBSVM) and Bayes for toy datasets.

5.3 Experimental Environment

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

Table 5.1 presents a summary of the 23 datasets used throughout the experiments, where the number of attributes (dimensionality), classes, and samples are shown. The datasets used and the results obtained are divided into three groups: *small*, *medium* and *large*. The datasets were obtained from the UCI Machine Learning repository[1] [5], and the LibCVM[2] [71]

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

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

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

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

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

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

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

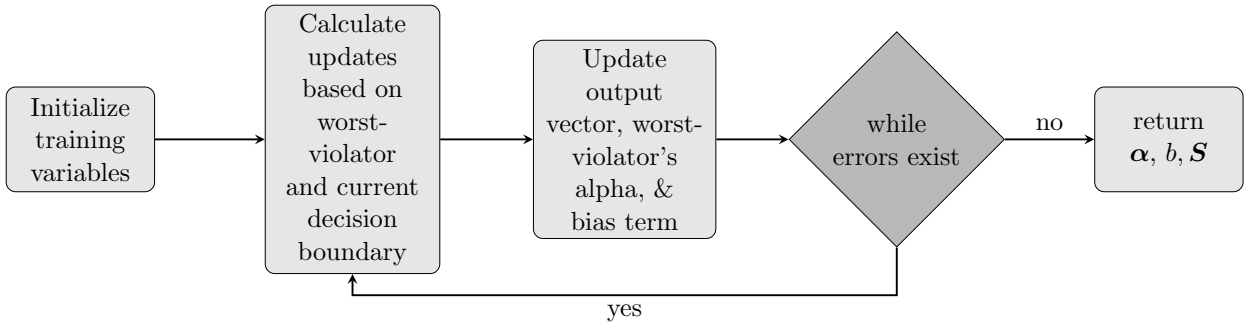

Fig. 5.2. A summary of the steps performed by OLLAWV. The model parameters ($\boldsymbol{\alpha}$, b, $\boldsymbol{S}$) and the algorithm variables ($\boldsymbol{o}$, t, wv, and yo) are first initialized. The worst-violator with respect to the current hyperplane is then found and the model parameters are then updated. Once no more violating samples are found, the model is returned.

Table 5.1. Datasets used for OLLAWV Comparison

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

and LIBSVM[3] [22] sites.

5.4 Results & Statistical Analysis

5.5 Conclusions

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

CHAPTER 6

CONCLUSIONS

This thesis introduced four novel SVM algorithms for learning from the non-standard and diverse learning paradigms: multi-target regression and multi-instance 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 is a problem transformation method, 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, 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 multiple-instance bag-level formulation and algorithm, dubbed MIRSVM, with a bag-representative selector, are proposed. The algorithm trains the SVM 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 SVM multi-instance algorithms, as well as other algorithms of different classes.

CHAPTER 7

FUTURE WORK

The contributions introduced in this work highlight the performance quality of SVMs for solving difficult learning problems. The SVM solvers used were LIBSVM's SMO implementation and MATLAB's standard QP solver. Even though the proposals outperformed existing popular algorithms in each learning paradigm, these two popular and standard solvers have some disadvantages. Although SMO provides an exact solution to the SVM QP problem, its performance is highly dependent on the SVM hyperparameters and MATLAB's QP solver is notoriously dependent on the size and complexity of the data. To address these problems, the future work of this dissertation first aims to deal with these limitations without sacrificing accuracy for scalability with respect to large-scale problems, while maintaining applicability to small and medium sized datasets.

Stochastic gradient descent (SGD) methods are popular tools used to optimize large-scale learning problems because of their generalization performance, simplicity, and scalability. Recently, SGD algorithms have been shown to have considerable performance and generalization capabilities in the context of large-scale learning [15], and have been used to solve the SVM problem, such as *NORMA* [51] and *PEGASOS* [65, 85]. 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 [60]. 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 [21].

To address the limitations presented by current popular SVM solvers, the future work of this dissertation involves the following:

- Devising a unique iterative procedure for solving the L1-SVM problem, as well as a novel method for identifying support vectors. Rather than randomly iterating over the data samples, the proposal will aim to reduce training time by selecting and updating the samples that are most incorrectly classified with respect to the current decision hyperplane.

- Designing a novel stopping criteria by utilizing this support vector 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.

- Applying, experimenting, and analyzing the performance of the newly proposed method to a popular new online paradigm, learning from streaming data, where data is not persistent, rather it is transient over time. This challenging setting will test the new contribution's performance on not only a dynamic data representation, but also on how efficiently computational resources are being used when learning is constrained over time.

REFERENCES

[1] 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.

[2] 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.

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

[4] 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.

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

[6] 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.

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

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

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

[10] 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.

[11] 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.

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

[13] 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.

[14] 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.

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

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

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

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

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

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

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

[22] 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.

[23] 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.

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

[25] 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.

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

[27] 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.

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

[29] 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.

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

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

[32] 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.

[33] 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.

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

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

[36] 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.

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

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

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

[40] 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.

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

[42] 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.

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

[44] 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.

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

[46] 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.

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

[48] 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.

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

[50] 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.

[51] 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.

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

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

[54] 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.

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

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

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

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

[59] Gabriella Angela Melki. *Fast online training of L1 support vector machines*. Virginia Commonwealth University, 2016.

[60] 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.

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

[62] 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.

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

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

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

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

[67] 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.

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

[69] 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.

[70] 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.

[71] 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.

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

[73] 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.

[74] 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.

[75] 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.

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

[77] 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.

[78] X. Wang et al. "A multi-view two-level classification method for generalized multi-instance problems". In: *Proceeding of the IEEE International Conference on Big Data*. 2014, pp. 104–111.

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

[80] 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.

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

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

[83] 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.

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

[85] 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.

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

VITA

Gabriella Melki received her BSc. in Computer Science at 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 various 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