Multi-Instance Support Vector Machine with Bag Representatives

Gabriella Melki[a], Alberto Cano[a], Sebastián Ventura[b,c]

[a]*Department of Computer Science, Virginia Commonwealth University, USA*
[b]*Department of Computer Science and Numerical Analysis, University of Cordoba, Spain*
[c]*Department of Computer Science, King Abdulaziz University, Saudi Arabia Kingdom*

Abstract

Multiple-instance (MI) learning is a generalization of *supervised learning*, where samples are represented by labeled bags, each containing sets of instances. The individual labels of the samples within a bag are unknown, and labels are assigned based on a multi-instance assumption. One of the major complexities associated with this type of learning is the ambiguous relationship between a bag's label and the instances it contains. This paper proposes a novel support vector machine (SVM) formulation and algorithm with a bag-representative selector for MI learning that trains a model based on bag-level information, rather than operating on the instance-level. SVM models are similar to other machine learning methods, but research has shown that they usually outperform them in terms of computational efficiency, scalability, and robustness against outliers, which is why they are the base learner for this proposal. The experimental study compares and evaluates the performance of this proposal against 10 state-of-the-art multi-instance methods over 15 MI datasets, and the results are validated through multiple non-parametric statistical analysis. The results indicate that bag-based learners outperform the instance-based learners, as well as this proposal's overall superior performance.

Keywords: Multi-instance binary classification, support vector machines, multi-instance bag-based classification.

I have to recognize this is not the first MIL application ever...

---

1. Introduction

In traditional binary classification, learning algorithms attempt to correctly separate two disjoint sets of samples by finding patterns that exists between the input feature space and their class label. It is a subset of *supervised learning*, where sample class labels are known and used to train a model that accurately predicts unknown sample class labels.

Multi-instance learning (or *multiple-instance* learning) is a generalization of *supervised learning* that has been recently been gaining interest because of its applicability to many real-world problems such as text categorization [3], image classificaion [26, 45], student performance prediction [44], human action recognition [43], and drug activity prediction [16, 34].

The difference between MI learning and traditional learning is the nature of the data samples. In the traditional setting, each sample is represented by a single feature vector that has its own class label. In multi-instance learning, a sample is considered to be a *bag* that contains multiple examples. Each bag is associated with a single label. The individual labels of the samples within a bag, referred to as *instances*, are unknown, and labels are assigned based on a multi-instance assumption, or hypothesis. Introduced by Diettrich et.al. [16], the standard MI assumption states that a bag is labeled as positive if and only if it contains at least one positive instance. More recently, other MI hypotheses and frameworks have been proposed [19] to encompass a wider range of applications with multi-instance data.

One of the major complexities associated with multi-instance learning is the ambiguity of the relationship between a bag label and the instances within the bag [10]. This stems from the standard MI assumption, where the underlying distribution among the instances within positive bags is unknown, but at least one instance has a positive label. There have been different attempts to overcome this complexity. One approach was to "flatten" the MI datasets, meaning samples contained in positive bags each adopt a positive label and classical supervised learning techniques can be used [38]. This approach assumes that positive bags contain a significant number of positive samples, which may not be the case. This can cause the classifier to mislabel negative samples within the bag decreasing the power of the resulting MI model.

this is not right. It's just the opposite

To overcome this, a different MI approach was proposed, where subsets of samples are selected from the positive bags to contribute to training the classifier [34, 45]. One drawback of this type of approach is that the resulting training datasets become imbalanced towards the positive samples by significantly increasing the number of negative samples. For instance, some methods, such as APR, DD based methods [12, 13, 16, 33, 45], assume that all instances within a postivie bag are also labeled positive. This is also implicit in the initialization step of MISVM [3]. The performance of these methods deteriorates when more samples are selected as subsets than needed, limiting their applicability [11]. Our proposal aims to deal with this drawback by minimizing class imbalance by optimally selecting bag representatives from both classes using Support Vector Machines.

Support Vector Machines (SVMs) represent a set of supervised, linear and nonlinear, classification and regression methods that have theoretical foundations on Vapnik-Chervonenkis (VC) theory [28, 39]. SVM models are similar to other machine learning algorithms and techniques, but research has shown that they usually outperform them in terms of computational efficiency, scalability, and robustness against outliers [30, 32, 35], which makes them a useful and powerful data mining tool for various real-world applications.

I have already read this phrase!

I like this part of the intro

To address the limitations presented by MI algorithms, this paper proposes a novel support vector machine formulation with a bag-representative selector for MIL called Multiple-Instance Representative Support Vector Machine (MIRSVM). SVMs are known to perform well when data is limited, therefore combining them with a bag-representative selector aims to remedy class imbalance caused by limited positive samples, without assuming their distributions. The algorithm selects bag-representatives iteratively according to the standard MI assumption, ensuring known information, such as the negative bag labels, are fully utilized during the training process. The optimal separating hyperplane between bags is then found with respect to the bag-representatives using a Gaussian kernel and a quadratic programming solver. The optimal set of representatives is found when they have not changed from one iteration to the next. The algorithm does not assume any distribution of the instances and is not affected by the number of samples 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.

- 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 key feature of MIRSVM is its ability to identify instances (support vectors) within positive and negative bags that highly impact the model. The dual formulation is kernelized with a Gaussian radial basis function, which calculates the distances between bag representatives.

Is not this too much detail for the intro?

- Devising a unique bag-representative selector method that makes no presumptions about the underlying distributions of the instances within each bag, while maintaining the SMI assumption. This approach eliminates the issue of class imbalance caused by techniques such as flattening or subsetting positive instances from each bag. A unique Gaussian RBF kernel is first used in conjunction with the SMI assumption to select representative instances by calculating the distances between the each bag and the current set of representatives. The bags' output vectors are then calculated, and instance index of the maximal output value is selected as representative. This method exploits existing knowledge about negative bags' distributions, favoring the negative, while learning about the positive class, while converging to the optimal set of representatives, which stops the training process.

This work is organized as follows. First, the notation used throughout the paper is provided, the MIL problem is formalized, and recent state-of-the-art multiple-instance learning methods as well as traditional support vector machines are described and reviewed in Section 2. The primal L1-SVM is formulated with respect to optimizing over bags, the dual formulation is then derived, along with the conditions for optimality, and the proposed algorithm, MIRSVM, is formulated and described in Section 3. Section 4 presents the experimental environment, including results from various metrics and non-parametric statistical analysis over 15 benchmark MI datasets compared with 10 state-of-the-art algorithms. Finally, Section 5 presents the conclusions of this contribution.

2. Background

This section defines the notation that will be used throughout the paper, and reviews related works on multi-instance learning and support vector machines.

2.1. Notation

Let $\mathcal{D}$ be a training dataset of n bags. Let $\boldsymbol{B} \in \mathbb{Z}_+^n$ be a vector containing each bags identifier. 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, n bags, and a total of m samples, having a domain of $\boldsymbol{X} \in \mathbb{R}^{m \times d}$. Each bag contains a number of instances, sometimes of different size and usually non-overlapping, such as $\boldsymbol{X}_I = \{\boldsymbol{x}_1^{(I)}, \ldots, \boldsymbol{x}_{p_I}^{(I)}\}$, where p_I is the number of instances in bag I. Using the training dataset $\mathcal{D} = \{(\boldsymbol{x}^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}^{(n)}, y^{(n)})\}$, the goal is to learn a model $f : \boldsymbol{X} \times \boldsymbol{Y}$, that assigns a label, $\boldsymbol{Y}_I \in \{-1, 1\}$, for each input bag $\boldsymbol{X}_I$. The model will then be used to predict the labels of new, unlabeled, and unseen input bags. Table 1 provides a summary of the notation used in this paper.

2.2. Multiple-Instance Learning

In traditional binary classification or pattern recognition problems, the goal is to learn a model that maps input samples to labels, $f : \mathbb{R}^{m \times d} \to \boldsymbol{Y}^m \in \{-1, +1\}$. Multiple instance learning generalizes this framework and makes weaker assumptions about sample labeling. In the MIL case, samples are called *bags* and each bag contains one or more input instances. Each bag is assigned a label, unlike traditional classification problems, where individual samples are assigned

Table 1: Summary of Notation

Definition	Notation
Total Number of Samples	m
Number of Bags	n
Number of Input Attributes	d
Bag IDs	$\boldsymbol{B}_I \in \mathbb{Z}^n,\ \forall I = \{1, \ldots, n\}$
Input Space	$\boldsymbol{X} \in \mathbb{R}^{m \times d}$
Input Bag	$\boldsymbol{X}_I = \left\{\boldsymbol{x}_1^{(I)}, \ldots, \boldsymbol{x}_{p_I}^{(I)}\right\},\ \forall I = \{1, \ldots, n\}$
Bag Size	$p_I = \lvert\boldsymbol{X}_I\rvert,\ \forall I \in \{1, \ldots, n\}$
Input Instance from Bag I	$\boldsymbol{x}_1^{(I)} = (x_1^{(I)}, \ldots, x_d^{(I)}) \in \boldsymbol{X}_I$
Bag Labels	$\boldsymbol{Y} \in \{-1, 1\}^n$
Full Multi-Instance Training Dataset	$\mathcal{D} = \{(\boldsymbol{x}_1^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}_d^{(n)}, y^{(n)})\}$
Full Single-Instance Training Dataset	$\mathcal{S} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\} \in \mathbb{R}^{m \times d}$

a label. Input instances, $\{\boldsymbol{x}_1,\ \boldsymbol{x}_2,\ \ldots,\ \boldsymbol{x}_m\}$, are grouped into bags with unique identifiers, $\boldsymbol{X} = \{\boldsymbol{X}_1,\ \boldsymbol{X}_2,\ \ldots,\ \boldsymbol{X}_n \,|\, \boldsymbol{X}_I = \{\boldsymbol{x}_i \,|\, \forall i \in \boldsymbol{B}_I\},\ \forall I = 1, \ldots, n\}$ and assigned a label, $\boldsymbol{Y}_I$. An example representation of an MI dataset is shown in Figure 1.

In MIL, the goal is to train a classifier that predicts the label of an unseen bag, $f(\boldsymbol{X}_{n+1}) \rightarrow \boldsymbol{Y}_{n+1}$ [2]. In order to build a classifier without any knowledge of the individual training instance labels, Dietterich et. al. proposed the *standard MI* (SMI) hypothesis [16] based on the domain of drug-activity prediction, shown in Equation 1, which states that a bag is labeled positive if at least one of the instances in the bag is positive, and is labeled negative otherwise.

$$\boldsymbol{Y}_I = \begin{cases} +1 & \text{if } \exists\, y_i = +1,\ i \in I \\ -1 & \text{otherwise.} \end{cases} \tag{1}$$

this is not correct. There are problems where this hypothesis is valid too

This implies that individual instance labels y_i exist, but are not known during training. Equation 1 can also be rewritten as equation 2 for simplicity.

Making such assumptions is very dangerous. Reviewers can feel upset reading them!

$$\boldsymbol{Y}_I = \text{argmax}_{i \in I}\ \ y_i \tag{2}$$

Since the SMI is not guaranteed to hold on domains other than drug-activity prediction, Foulds and Frank [19] reviewed the uses of alternative MI assumptions in the context of different domains. Amores [2] then presented a review describing the taxonomy of multi-instance classification, and 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 all 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.

This is confusing too. I can't get the difference from this assertion and embedded methods

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

I am not sure about this assertion. Instance-based method build models using instances, but these method don't have to make assumptions about positive/negative bags. Is this what you mean?

BagID	F1	F2	F3	…	FN	Class
	0.1	0.8	2.5	…	0.8	
1	0.2	2.0	5.5	…	3.0	POS
	0.1	?	4.5	…	0.1	
	1.5	4.0	0.8	…	0.1	
	0.8	0.4	2.9	…	1.1	
2	2.3	0.2	4.0	…	5.5	NEG
	6.7	5.0	?	…	0.5	
	0.1	4.0	8.7	…	3.3	

Figure 1: Example of multi-instance data representation.

Rectangle (APR) [16], 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. This method follows the max-rule shown in Equation 2. Another similar method is the Diverse Density (DD) [33] 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 [45], Zhang et. al. propose a similar framework that iteratively maximizes the DD measure. In [4], Auer and Ortner 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 in [4], optimal balls per positive bags are computed. A major challenge affecting these methods is that the distributions of the positive and negative bags affects their performance. Methods such as APR [16] are not designed to handle types of distributions where instances are grouped into compact and distinct clusters [11]. Methods based on the DD metric [12, 13, 33, 45] assume the positive instances form a cluster, which may not be the case. In MILIS [20], the distribution of negative bags is modeled with Gaussian kernels, which can prove difficult when the quantity of data is limited.

Some methods in literature [38, 41], such as Simple-MI [17], transform the multi-instance dataset to a traditional instance-level dataset. Simple-MI represents each bag with the mean vector of the instances within the bag. This approach was evaluated by Bunescu and Mooney in [9] and proposed mapping each bag to a max-min vector, a concatenation of the features with the highest and lowest values. 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.

Blockeel et. al. introduced the Multi-Instance Tree Inducer (MITI) [7], based on the standard MI assumption, which uses decision trees and a pure-positive leaf heuristic to solve the MI problem. A pure-positive leaf node is defined as a node that only contains instances from positive bags. The aim of this approach is to strongly 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 then enhanced this approach in [6], creating the method Multi-Instance Rule Induction (MIRI). When a positive leaf is encountered by the MITI algorithm, all positive bags associated with the leaf node are removed from the training data, and the path from root to the node associated with this leaf is turned into an if-then rule. The algorithm is then restarted and the tree is regrown using the reduced data. This eliminates any possibility of a suboptimal split because the tree is discarded and regrown.

Why are you including this here?

Is not it better making a SVM-MI section? There are more references

Including APR [33] in related work is a bit outdated ...

Andrews et. al. proposed a mixed-integer quadratic program that solves the MI problem using a support vector machine, MISVM, that can be solved heuristically [3]. Rather than maximizing the margin of separability between instances of different classes, it maximizes the margin between bags. 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. The main disadvantage of this approach is that it creates an imbalanced class problem that favors the negative class, resulting in a biased classifier.

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. 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.

2.3. *Support Vector Machines*

Most classical machine learning techniques require knowledge of the data distribution in order to learn accurate models. This is a serious restriction because, in most cases, the distribution of the data is unknown. Another disadvantage of these methods stems from high dimensional, sparse datasets, which are very common in real-world applications. Small sample size also poses problems of model reliability, especially when coupled with high dimensional feature spaces. SVMs represent learning techniques that have been introduced under the *structural risk minimization* (SRM) framework and VC theory [29]. Rather than optimizing over L1 or L2 norms and classification error, SVMs perform SRM [40], minimizing the expected probability of classification error, resulting in a generalized model without making assumptions about the data distribution [14].

SVMs are a particularly useful for learning linear predictors in high dimensional feature spaces, which is a computationally complex learning problem. In the context of classification, this problem is approached by searching for the optimal *maximal margin* of separability between classes. A training set $\mathcal{D}$ is *linearly separable* if a halfspace exists, $(\boldsymbol{w}, b)$, such that $y_i = \text{sign}\left(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b\right), \forall i \in \{1, \ldots, m\}$, where $\boldsymbol{w} \in \mathbb{R}^d$ is a d-dimensional weight vector, and $b \in \mathbb{R}$ is a bias term. All halfspaces, defined as $d(\boldsymbol{x}_i) = \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b$, satisfying $y_i\left(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b\right) > 0,\ \forall i \in \{1, \ldots, m\}$, have no error. This hard constraint is associated with the *Hard-Margin SVM*. No feasible solution exists for the Hard-Margin SVM problem if the dataset is non-linearly separable, which is the case for most datasets. To overcome this, Cortes and Vapnik [14] introduced the *Soft-Margin L1-SVM*,

$$\min_{\boldsymbol{w}, b, \boldsymbol{\xi}} \frac{1}{2}||\boldsymbol{w}||^2 + \frac{C}{m}\sum_{i=1}^{m} \xi_i, \tag{3}$$

$$\text{s.t. } y_i(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle + b) \geq 1 - \xi_i,\ \forall i \in \{1, \ldots, m\} \tag{3a}$$

$$\xi_i \geq 0,\ \forall i \in \{1, \ldots, m\} \tag{3b}$$

where $C \in \mathbb{R}$ is the penalty parameter that controls the trade-off between margin maximization and classification error minimization, penalizing large norms and errors. The slack variable $\boldsymbol{\xi}$ allows for optimizing over the samples' errors. The resulting hyperplane is called *soft-margin hyperplane*.

Although the Soft-SVM learns an optimal hyperplane, if the training data set is not linearly separable, the classifier learned may not have a good generalization capability [36]. Generalization and linear separability can be enhanced by mapping the original input space to a higher dimensional dot-product space by using a kernel function shown in Equation 4.

$$\mathcal{K}\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) = \langle \phi\left(\boldsymbol{x}_i\right), \phi\left(\boldsymbol{x}_j\right) \rangle, \tag{4}$$

where $\phi\left(\cdot\right)$ represents a function mapping from the original feature space to a higher dimensional space. This kernel mapping is particularly helpful when solving the dual SVM optimization problem shown in Equation 5. Rather than calculating the inner product of two mapped vectors, their corresponding *scalar* kernel value can be used.

$$\max_{\boldsymbol{\alpha}} \; \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \mathcal{K}\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) - \sum_{i=1}^{m} \alpha_i \tag{5}$$

$$\text{s.t.} \; \sum_{i=1}^{m} \alpha_i y_i = 0, \tag{5a}$$

$$0 \leq \alpha_i \leq \frac{C}{m}, \; \forall i \in \{1, \ldots, m\}. \tag{5b}$$

A traditional and widely used method of solving the L1-SVM problem is the *Sequential Minimal Optimization* (SMO) technique [37]. It is an iterative procedure that divides the SVM dual problem into a series of sub-problems, which are then solved analytically by finding the optimal $\boldsymbol{\alpha}$ values that satisfy the Karush-Kuhn-Tucker (KKT) conditions. 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, as well as the SVM hyperparameters [39]. *Iterative Single Data Algorithm* (ISDA) [28, 30] is a more recent and efficient approach for solving the L2-SVM problem, shown to be faster than the SMO algorithm and equal in terms of accuracy [31, 32]. It iteratively updates the objective function by working on one data point at a time, using coordinate descent to find the optimal objective function value. Other methods for solving the SVM problem include *Quadratic Programming* solvers, such as the interior point method. These types of algorithms find the true optimal objective function value at the trade-off of having a relatively slower run-time.

In the following section our contribution, MIRSVM, will be introduced, which modifies the traditional SVM optimization problem to accommodate multi-instance datasets, optimizes over the representatives chosen per each bag, and is solved using a quadratic programming solver.

2.3 VM in MI problems. 5

3. Multiple-Instance Representative SVM

The SVM problem has been extended to be applied to multi-instance data by treating each bag as a sample. A representative is iteratively chosen from each bag based on selection criterion described below, and a new hyperplane is formed according to the representatives until they converge, i.e. stop changing per iteration. 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. This is evident during the representative selection process by taking the maximum output value's sign within each bag based on the current hyperplane using the following rule, $s_I = \text{argmax}_{i \in B_I}(\langle \boldsymbol{w},\, \boldsymbol{x}_i \rangle + b),\, \forall I \in B$. In other words, the most positive sample is chosen from each positive bag and the least negative sample is chosen from each negative bag, pushing the decision boundary "*closer*" towards the positive bags. Equation 6 presents the primal MIRSVM optimization problem,

$$\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \frac{1}{2}||\boldsymbol{w}||^2 + \frac{C}{n}\sum_{I=1}^{n} \xi_I, \tag{6}$$

$$\text{s.t. } y_I(\langle \boldsymbol{w}, \boldsymbol{x}_{s_I} \rangle + b) \geq 1 - \xi_I,\, \forall I \in B \tag{6a}$$

$$\xi_I \geq 0,\, \forall I \in B \tag{6b}$$

where $\boldsymbol{S}_I$ is a vector of the bag representatives' indices and $\boldsymbol{x}_{s_I}$ is the sample representative of bag I. Note the variables in MIRSVMs formulation are the similar to those of the traditional SVM, except they are now representing each bag as a sample. Solving the optimization problem given in Equation 6 using a quadratic programming solver is a computationally expensive task due to the number of constraints, which scales by the number of bags m, as well as the calculation of the inner product between two d-dimensional vectors in constraint 6a. The proposed solution for these problems was deriving the dual of Equations 6.

The dual can be found by first forming the primal Lagrangian given by Equation 7, where $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are the non-negative Lagrange multipliers.

$$\mathcal{L}\left(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}\right) = \frac{1}{2}\sum_{j=1}^{d} w_j^2 + \frac{C}{n}\sum_{I=1}^{n}\xi_I - \sum_{I=1}^{n}\beta_I\xi_I - \sum_{I=1}^{n}\alpha_I\left(y_I\left(\sum_{j=1}^{d} w_j \boldsymbol{x}_{s_I j} + b\right) - 1 + \xi_I\right) \tag{7}$$

The following Karush-Kuhn-Tucker (KKT) [8] conditions must be satisfied:

$$\frac{\partial \mathcal{L}}{\partial w_j} = 0,\, \forall j \in \{1, \ldots, d\}$$

$$\frac{\partial \mathcal{L}}{\partial b} = 0$$

$$\frac{\partial \mathcal{L}}{\partial \xi_I} = 0,\, \forall I \in \boldsymbol{B}$$

$$\alpha_I\left(y_I(\langle \boldsymbol{w}, \boldsymbol{x}_{s_I} \rangle + b) - 1 + \xi_I\right) = 0,\, \forall I \in \boldsymbol{B}$$

$$\beta_I\xi_I = 0, \forall I \in \boldsymbol{B}$$

$$\alpha_I \geq 0,\, \beta_I \geq 0,\, \xi_I \geq 0,\, \forall I \in \boldsymbol{B}$$

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

$$\frac{\partial \mathcal{L}}{\partial w_j} : w_j = \sum_{I=1}^{n} \alpha_I \, y_I \, \boldsymbol{x}_{s_I \, j}, \, \forall j \in \{1, \ldots, d\} \tag{8}$$

$$\frac{\partial \mathcal{L}}{\partial b} : \sum_{I=1}^{n} \alpha_I y_I = 0, \tag{9}$$

$$\frac{\partial \mathcal{L}}{\partial \xi_I} : \alpha_I + \beta_I = \frac{C}{n}, \, \forall I \in \boldsymbol{B} \tag{10}$$

By substituting Equations 8, 9, and 10 into the Lagrangian in 7, the following dual problem, $q(\boldsymbol{\alpha}, \boldsymbol{\beta})$, is obtained:

$$q(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \inf_{\boldsymbol{w},b,\boldsymbol{\xi}} \frac{1}{2} \sum_{I=1}^{n} \sum_{K=1}^{n} \sum_{j=1}^{d} \alpha_I \alpha_K y_I y_K x_{s_I j} x_{s_K j} + \sum_{I=1}^{n} \xi_I \left(\alpha_I + \beta_I\right) - \sum_{I=1}^{n} \xi_I \left(\alpha_I + \beta_I\right)$$
$$- \sum_{I=1}^{n} \sum_{K=1}^{n} \sum_{j=1}^{d} \alpha_I \alpha_K y_I y_K x_{s_I j} x_{s_K j} - \sum_{I=1}^{n} \alpha_I y_I b + \sum_{I=1}^{n} \alpha_I$$

At optimality, $\sum_{I=1}^{n} \alpha_I y_I = 0$, so the term with b can be removed. The terms with ξ can also be cancelled out because they negate each other. The resulting function is now with respect to the dual variables, so the infemum can be dropped. The dual MIRSVM formulation then becomes,

$$\max_{\boldsymbol{\alpha},\boldsymbol{\beta}} \sum_{I=1}^{n} \alpha_I - \frac{1}{2} \sum_{I=1}^{n} \sum_{K=1}^{n} \sum_{j=1}^{d} \alpha_I \alpha_K y_I y_K x_{s_I j} x_{s_K j} \tag{11}$$

$$\text{s.t.} \sum_{I=1}^{n} \alpha_I y_I = 0 \tag{11a}$$

$$\alpha_I + \beta_I = \frac{C}{n}, \, \forall I \in B \tag{11b}$$

$$\alpha_I \geq 0, \, \forall I \in B \tag{11c}$$

$$\beta_I \geq 0, \, \forall I \in B \tag{11d}$$

where s_I is computed for each bag, as shown in Equation 12.

$$s_I = \underset{i \in B_I}{\text{argmax}} \Big(\sum_{K=1}^{n} \sum_{j=1}^{d} \alpha_K y_K x_{s_K j} x_{ij} + b \Big), \, \forall I \in \boldsymbol{B} \tag{12}$$

The implicit constraints 11b through 11d imply three possible cases for the α_I values,

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

2. If $0 \leq \alpha_I \leq C/n$, 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 that is *unbounded*. If $0 \leq \xi_I < 1$, then the instance is correctly classified, otherwise it is misclassified.

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

$$\max_{\boldsymbol{\alpha}} \sum_{I=1}^{n} \alpha_I - \frac{1}{2} \sum_{I=1}^{n} \sum_{K=1}^{n} \sum_{j=1}^{d} \alpha_I \alpha_K y_I y_K \mathcal{K}\left(\boldsymbol{x}_{s_I}, \boldsymbol{x}_{s_K}\right) \tag{13}$$

$$\text{s.t.} \sum_{I=1}^{n} \alpha_I y_I = 0 \tag{13a}$$

$$0 \leq \alpha_I \leq \frac{C}{n}, \ \forall I \in \boldsymbol{B}, \tag{13b}$$

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. A Gaussian kernel is used in the MIRSVM algorithm, given by Equation 14, where σ is the shape parameter.

$$\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{-\frac{||x_i - x_j||^2}{2\sigma^2}} \tag{14}$$

To evaluate the output vector, $\boldsymbol{o}$, of a bag using the kernel, the following equation [28] is used,

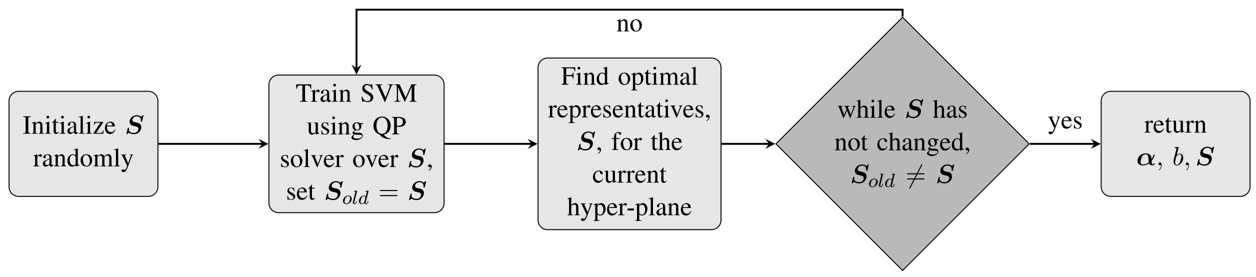
$$\boldsymbol{o} = \mathcal{K}(\boldsymbol{X}_I, \boldsymbol{X}_S) * (\boldsymbol{\alpha} \cdot \boldsymbol{Y}_S) + b \tag{15}$$

where $\boldsymbol{X}_I$ are the instances of bag I, $\boldsymbol{X}_s$ are the optimal bag representatives, $\boldsymbol{Y}_s$ are the representative bag labels. The bias term b is calculated as shown in Equation 16, where $\boldsymbol{sv}$ is the vector of support vector indices and n_{sv} is the number of support vectors [28].

$$b = \frac{1}{n_{sv}} \sum_{\boldsymbol{sv}} \boldsymbol{Y}_{\boldsymbol{sv}} - \mathcal{K}(\boldsymbol{X}_{\boldsymbol{sv}}, \boldsymbol{X}_{\boldsymbol{sv}}) * (\boldsymbol{\alpha}_{\boldsymbol{sv}} \cdot \boldsymbol{Y}_{\boldsymbol{sv}}) \tag{16}$$

Algorithm 1 and Figure 2 show the procedure for training the multi-instance representative SVM classifier, along with obtaining the optimal representatives from each bag. Algorithm 2 lists the method for testing the model on unseen data. During training, the representatives, $\boldsymbol{S}$, are first initialized by randomly selecting an instance from each bag. A hyper-plane is then trained using the representative instances, and new optimal representatives are found with respect to the current hyper-plane, by using the rule given in Equation 12. 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, i.e. when $\boldsymbol{S}_{old} = \boldsymbol{S}$. During the testing procedure, each bag produces an output vector based on the hyper-plane 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.

Figure 2: MIRSVM Flow Diagram


Initialize $\boldsymbol{S}$ randomly
Train SVM using QP solver over $\boldsymbol{S}$, set $\boldsymbol{S}_{old} = \boldsymbol{S}$
Find optimal representatives, $\boldsymbol{S}$, for the current hyper-plane
while $\boldsymbol{S}$ has not changed, $\boldsymbol{S}_{old} \neq \boldsymbol{S}$
no
yes
return $\boldsymbol{\alpha}$, b, $\boldsymbol{S}$

This figure represents a summary of the steps performed by the MIRSVM algorithm. The representatives are first randomly initialized and continuously updated according to the current hyper-plane, which is found using a quadratic programming (QP) solver. Upon completion, return the model variables and the optimal bag-representatives.

Algorithm 1 Multi-Instance Representative SVM (MIRSVM)

flush right?

Input: Training dataset $\mathcal{D}$, Bag IDs $\boldsymbol{B}$, SVM Parameters C and σ
Output: SVM model parameters $\boldsymbol{\alpha}$ and b, Bag Representative IDs $\boldsymbol{S}$

```
 1: $\boldsymbol{S}_{old} \leftarrow -\infty$
 2: for $I \in \boldsymbol{B}$ do
 3:     $\boldsymbol{S}_I \leftarrow$ rand $(|\boldsymbol{B}_I|, 1, 1)$          Assign each bag a random instance
 4: end for
 5: $\boldsymbol{X}_S \leftarrow \boldsymbol{X}(\boldsymbol{S})$, $\boldsymbol{Y}_S \leftarrow \boldsymbol{Y}(\boldsymbol{S})$, $\boldsymbol{B}_S \leftarrow \boldsymbol{B}(\boldsymbol{S})$          Initialize representative dataset
 6: while $\boldsymbol{S} \neq \boldsymbol{S}_{old}$ do
 7:     $\boldsymbol{S}_{old} \leftarrow \boldsymbol{S}$
 8:     $\boldsymbol{G} \leftarrow (\boldsymbol{Y}_S \times \boldsymbol{Y}_S) \cdot \mathcal{K}(\boldsymbol{X}_S, \boldsymbol{X}_S, \sigma)$          Build Graham matrix
 9:     $\boldsymbol{\alpha} \leftarrow$ quadprog $(\boldsymbol{G}, -\boldsymbol{1}^n, \boldsymbol{Y}_S, \boldsymbol{0}^n, \boldsymbol{0}^n, \boldsymbol{C}^n)$          Solve QP Problem
10:     $\boldsymbol{sv} \leftarrow$ find $(0 < \boldsymbol{\alpha} \leq C)$          Support vector indices
11:     $n_{sv} \leftarrow$ count $(0 < \boldsymbol{\alpha} \leq C)$          Number of support vectors
12:     $b \leftarrow \frac{1}{n_{sv}} \sum_{i=1}^{n_{sv}} (\boldsymbol{Y_{sv}} - \boldsymbol{G_{sv}} * (\boldsymbol{\alpha_{sv}} \cdot \boldsymbol{Y_{sv}}))$
13:     for $I \in \boldsymbol{B}$ do
14:         $\boldsymbol{G}_I \leftarrow (\boldsymbol{Y}_I \times \boldsymbol{Y}_S) \cdot \mathcal{K}(\boldsymbol{X}_I, \boldsymbol{X}_S, \sigma)$
15:         $\boldsymbol{S}_I \leftarrow \text{argmax}_{i \in I} (\boldsymbol{G}_I * \boldsymbol{\alpha} + b)$          Select representatives for current hyperplane
16:     end for
17:     $\boldsymbol{X}_S \leftarrow \boldsymbol{X}(\boldsymbol{S})$, $\boldsymbol{Y}_S \leftarrow \boldsymbol{Y}(\boldsymbol{S})$, $\boldsymbol{B}_S \leftarrow \boldsymbol{B}(\boldsymbol{S})$          Re-set representative dataset
18: end while
19: return $\boldsymbol{\alpha}$, $b$, $\boldsymbol{S}$
```

Algorithm 2 MIRSVM Prediction

Input: Testing dataset $\mathcal{D}$, Bag IDs $\boldsymbol{B}$, Representatives $\boldsymbol{S}$, Classifier $\boldsymbol{\alpha}$ and b, SVM Parameter σ
Output: Classifier output $\boldsymbol{O}$

```
1: for $I \in \boldsymbol{B}$ do
2:     $\boldsymbol{G}_I \leftarrow (\boldsymbol{Y}_I \times \boldsymbol{Y}_S) \cdot \mathcal{K}(\boldsymbol{X}_I, \boldsymbol{X}_S, \sigma)$
3:     $\boldsymbol{O}_I \leftarrow$ max $(\boldsymbol{G}_I * \boldsymbol{\alpha} + b)$
4: end for
5: return $\boldsymbol{O}$
```

Table 2: Dataset Information

Dataset	Attributes	Positive Bags	Negative Bags	Total	Instances
Suramin	20	7	4	11	2378
EastWest	24	10	10	20	213
WestEast	24	10	10	20	213
Musk1	166	47	45	92	476
Musk2	166	39	63	102	6598
Webmining	5863	17	58	75	2212
TRX	8	25	193	168	26611
Mutagenesis-Atoms	10	125	63	188	1618
Mutagenesis-Bonds	16	125	63	188	3995
Mutagenesis-Chains	24	125	63	188	5349
Tiger	230	100	100	200	1391
Elephant	230	100	100	200	1220
Fox	230	100	100	200	1320
Component	22	423	2707	3130	36894
Function	202	443	4799	5242	55536

4. Experiments

This section presents the experimental setup and comparison of our contribution, as well as ten other state-of-the-art methods on 15 different benchmark datasets. First, the experimental setup and state-of-the-art methods are described. The results for each metric, as well as the statistical analysis are then presented. Finally, the performance of the algorithms is discussed and analyzed.

4.1. Experimental Setup

Table 2 presents a summary of the 15 datasets used throughout the experiments, where the number of attributes (dimensionality), bags, and total number of instances are shown. The datasets were obtained from the Weka [25] and KEEL [1] dataset repositories.

The experimental environment was designed to test the difference in performance of the proposed method against 10 state-of-the-art algorithms, contrasting instance-level methods and bag-level methods. Instance-level methods include *MIOptimalBall*, *MIBoost*, *MISVM*, *MIDD*, and *MIWrapper*. Bag-space methods include *MISMO*, *SimpleMI*, and *TLC*. The ensemble-based bag-space methods, *Bagging* and *Stacking* were also used. The base algorithms used 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.

Experiments were performed using k-fold cross validation, with $k = 10$, in order to evaluate the models' performances and tune hyper-parameters. The data is separated fairly into 10 equally sized sections where, at every iteration of the cross-validation loop, a section is held out as the test set, while the remainder of the data is used for training. This procedure ensures the model is not optimistically biased towards the full dataset. The tuning of the model during cross-validation includes finding the best penalty parameter, C, as well as the best shape parameter for the Gaussian radial basis function (RBF) kernel, σ. The best hyper-parameters were chosen from the following

I think explaining 10-fold CV is not necessary

6×6 possible combination runs, shown in Equations (17a) and (17b), referred to as (17).

$$C \in \{0.1, 1, 10, 100, 1000, 10000\} \tag{17a}$$

$$\sigma \in \{0.1, 0.5, 1, 2, 5, 10\} \tag{17b}$$

4.2. Results & Statistical Analysis

The classification performance was measured using five metrics: Accuracy, Precision, Recall, Cohen's Kappa Rate, and Area under ROC curve (AUC). The accuracy metric can be misleading when classes are imbalanced, as is the case with the *component* and *function* datasets, which have six and ten times as many negative instances than positive, respectively. Cohen's Kappa Rate and the AUC measures are used as complementary measures in order to evaluate the algorithms comprehensively [5]. Cohen's kappa, shown in Equation 18d, evaluates classifier merit according to the class distribution and ranges between -1 (full disagreement), 0 (random classification), and 1 (full agreement). The AUC metric highlights the trade-off between the true positive rate, or recall, and the false positive rate, as shown in Equation 18e. The ~~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 18. The results are shown in Tables 3, 5, 6, 8, and 10.

$$\text{Accuracy} \qquad \frac{TP + TN}{n} \tag{18a}$$

$$\text{Precision} \qquad \frac{TP}{TP + FP} \tag{18b}$$

$$\text{Recall} \qquad \frac{TP}{TP + FN} \tag{18c}$$

$$\text{Cohen's Kappa Rate} \qquad \frac{n - \frac{(TP+FN)*(TP+FP)}{n}}{1 - \frac{(TP+FN)*(TP+FP)}{n}} \tag{18d}$$

$$\text{Area Under ROC Curve} \qquad \frac{1 + \text{recall} - \frac{FP}{FP+TN}}{2} \tag{18e}$$

In order to analyze the performances of the multiple models, non-parametric statistical tests are used to validate the experimental results obtained [15, 21]. The Iman-Davenport non-parametric test is run to investigate whether significant differences exist among the performance of the algorithms [22] by ranking the algorithms over the datasets used, using the Friedman test. The ranks are presented in the last row of the results tables, and the lowest (best) rank value is typeset in bold. Table 12 contains the ranks and meta-rank of all methods, which helps determine and highlight the best performing algorithms across all datasets and metrics.

After the Iman-Davenport test indicates significant differences, the Bonferroni-Dunn post-hoc test [18] is then used to find where they occur between algorithms by assuming the classifiers performances are different by at least some critical value [23]. Below each result table, a diagram highlighting the critical distance (in gray) between each algorithm is shown.

The Wilcoxon, Nemenyi, and Holm [24, 27, 42] tests were then run for each of the metrics to compute multiple pairwise comparisons among the proposed algorithm and the state-of-the-art methods, investigating whether statistical differences exist among pairs of algorithms. Tables 4, 5, 7, 9, and 11 show the sum of ranks R^+ and R^- of the Wilcoxon rank-sum test, and the p-values for the three tests, showing statistical confidence rather than using a fixed α value.

Table 3: Accuracy

Datasets	MIRSVM	MIBoost	MIOptimalBall	MIDD	MIWrapper	MISMO	MISVM	SimpleMI	TLC	Bagging	Stacking
suramin	0.6000	0.5000	0.7250	0.4250	0.5000	0.7250	0.5000	0.2308	0.6923	0.6750	**0.7564**
eastWest	**0.8000**	0.5000	0.7250	0.6125	0.5000	0.7125	0.6375	0.5000	0.5000	0.6375	0.4500
westEast	0.6500	0.5000	0.3750	0.4500	0.5000	**0.7375**	0.4625	0.5000	0.5000	0.6875	0.6375
musk1	**0.9022**	0.5109	0.7717	0.8804	0.5109	0.7826	0.8043	0.5109	0.8587	0.8804	0.8587
musk2	**0.8146**	0.6139	0.7723	0.7228	0.6139	0.7030	0.7129	0.6139	0.6238	0.7129	0.6733
webmining	**0.8500**	0.8142	0.7699	0.8142	0.8142	0.8407	0.6903	0.8142	0.8142	0.7876	0.8053
trx	0.8825	0.8705	**0.9016**	0.8808	0.8705	0.8705	0.8705	0.8705	0.8756	0.8964	0.8860
mutagenesis-atoms	0.7714	0.6649	0.6436	0.7074	0.6649	0.6915	0.6649	0.6649	0.7766	**0.8032**	0.7606
mutagenesis-bonds	0.8252	0.6649	0.6915	0.7713	0.6649	0.7979	0.6649	0.6649	0.8351	**0.8830**	0.8564
mutagenesis-chains	0.8411	0.6649	0.6702	0.7766	0.6649	0.8351	0.6649	0.6649	0.8404	**0.8457**	0.8351
tiger	**0.7750**	0.5000	0.5000	0.7100	0.5000	0.7200	0.7550	0.5000	0.6650	0.7700	0.7250
elephant	0.8300	0.5000	0.5000	0.7900	0.5000	0.8100	0.8000	0.5000	0.8000	**0.8500**	0.8250
fox	**0.6550**	0.5000	0.5000	0.5800	0.5000	0.5250	0.5900	0.5000	0.6450	0.6200	0.6500
component	0.9366	0.8649	0.8696	0.8780	0.8649	0.8968	0.8703	0.8649	0.9358	**0.9371**	0.9355
function	0.9523	0.9155	0.9138	0.9193	0.9155	0.9376	0.9195	0.9155	0.9649	**0.9655**	0.9647
Average	**0.8057**	0.6390	0.6886	0.7279	0.6390	0.7724	0.7072	0.6210	0.7552	0.7968	0.7746
Ranks	**2.4000**	8.8000	7.2667	6.0333	8.8000	4.8000	7.0667	9.0000	4.7333	2.7667	4.3333

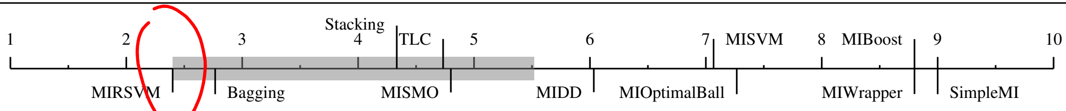


Figure 3: Bonferroni-Dunn test for Accuracy

Why multiple test for demonstrating the same thing?

Table 4: Wilcoxon, Nemenyi, and Holm tests for Accuracy

MIRSVM vs.	Wilcoxon R^+	Wilcoxon R^-	Wilcoxon p-value	Nemenyi p-value	Holm p-value
MIBoost	120.0	0.00	0.0001	0.0000	0.0056
MIOptimalBall	112.0	8.00	0.0015	5.9E-5	0.0071
MIDD	120.0	0.00	0.0001	0.0027	0.0100
MIWrapper	120.0	0.00	0.0001	0.0000	0.0063
MISMO	95.00	25.0	0.0424	0.0475	0.0125
MISVM	120.0	0.00	0.0001	1.2E-4	0.0083
SimpleMI	120.0	0.00	0.0001	0.0000	0.0050
TLC	94.00	26.0	0.0554	0.0540	0.0167
Bagging	60.00	60.0	1.0000	0.7621	0.0500
Stacking	88.00	32.0	0.1170	0.1104	0.0250

Post Hoc (Friedman) comparison for $\alpha = 0.05$

3 test are for one comparison!

Why are you doing a multiple comparison and a single comparison

???

This function is for ALBERTO!

4.3. *Accuracy*

Table 3 shows the accuracy results of the 11 algorithms over 15 multi-instance datasets, along with their average and rank. The results indicate that the bag-based and ensemble learners perform better than the instance-based and wrapper methods. Specifically, MIRSVM achieves the best accuracy over 6 of the 15 datasets with a competitive average against the Bagging, Stacking, and TLC algorithms.

Figure 3 and Table 4 show the results for the statistical analysis on the accuracy results. The algorithms with ranking higher than 5.51 (MIRSVM rank + Bonferroni-Dunn critical value), to the right of the grey bar in Figure 3, perform statistically worse than MIRSVM. Table 4 shows the p-values of the Wilcoxon, Nemenyi, and Holm tests, as well as the sum of ranks obtained by the Wilcoxon rank-sum, R^+ and R^-. The results from these tests are complement each other. Nemenyi's procedure indicates that MIRSVM performs significantly better than algorithms with p-value ≤ 0.005. Wilcoxon's and Holm's procedures determine statistical significance of the performance of the algorithms with p-values < 0.01 and ≤ 0.0125, respectively.

Table 5: Precision

Datasets	MIRSVM	MIBoost	MIOptimalBall	MIDD	MIWrapper	MISMO	MISVM	SimpleMI	TLC	Bagging	Stacking
suramin	0.6385	**1.0000**	**1.0000**	0.2857	**1.0000**	**1.0000**	0.5000	0.0000	0.6667	0.7333	0.7116
eastWest	0.7143	0.5000	**0.8750**	0.5882	0.5000	0.7429	0.8667	0.5000	0.5000	0.6667	0.4444
westEast	0.6300	0.5000	0.2727	0.4600	0.5000	**0.6939**	0.3846	0.5000	0.5000	0.6364	0.6038
musk1	0.8519	**1.0000**	0.9286	0.9048	**1.0000**	0.8049	0.8857	**1.0000**	0.8478	0.9250	0.8478
musk2	0.7167	0.6139	**0.7826**	0.7576	0.6139	0.7424	0.7538	0.6139	0.7400	0.7797	0.7164
webmining	0.7500	0.8142	0.8173	0.8142	0.8142	0.8936	**1.0000**	0.8142	0.8817	0.8469	0.8500
trx	0.6500	0.8705	**0.9306**	0.9191	0.8705	0.8705	0.8705	0.8705	0.9138	0.8936	0.9011
mutagenesis-atoms	0.7872	**1.0000**	0.4630	0.6111	**1.0000**	0.5439	**1.0000**	**1.0000**	0.7059	0.7321	0.6667
mutagenesis-bonds	0.8468	**1.0000**	0.5385	0.7500	**1.0000**	0.6812	**1.0000**	**1.0000**	0.7857	0.8596	0.8333
mutagenesis-chains	0.8571	**1.0000**	0.5091	0.7059	**1.0000**	0.7759	**1.0000**	**1.0000**	0.7705	0.7742	0.7581
tiger	0.7365	0.5000	0.5000	0.6944	0.5000	0.7444	0.7802	0.5000	0.6514	**0.7935**	0.7320
elephant	0.8576	0.5000	0.5000	0.7959	0.5000	0.8444	0.7679	0.5000	0.8000	**0.8804**	0.8283
fox	0.6040	0.5000	0.5000	0.5833	0.5000	0.5287	0.6216	0.5000	**0.6747**	0.6304	0.6705
component	**0.9866**	0.8649	0.8778	0.8902	0.8649	0.8958	0.8696	0.8649	0.9462	0.9431	0.9449
function	0.8459	0.9155	0.9202	0.9317	0.9155	0.9376	0.9197	0.9155	**0.9729**	0.9720	0.9726
Average	0.7649	0.7719	0.6944	0.7128	0.7719	0.7800	**0.8147**	0.7053	0.7572	0.8045	0.7654
Ranks	6.0667	6.7333	6.7333	6.7667	6.7333	5.3667	5.1000	7.3000	5.5000	**3.9333**	5.7667

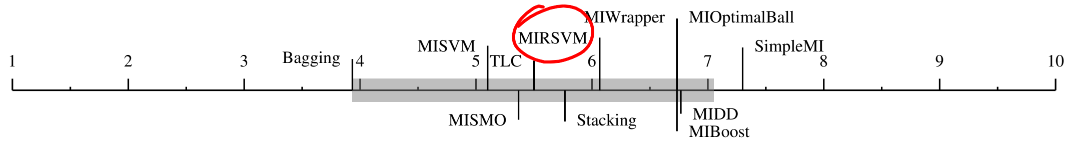


Figure 4: Bonferroni-Dunn test for Precision

Figure 5: Wilcoxon, Nemenyi, and Holm tests for Precision

MIRSVM vs.	Wilcoxon R^+	Wilcoxon R^-	Wilcoxon p-value	Nemenyi p-value	Holm p-value
MIBoost	56.0	64.0	≥ 0.2	0.0208	0.0071
MIOptimalBall	79.0	41.0	≥ 0.2	0.0208	0.0063
MIDD	87.0	33.0	0.1354	0.0193	0.0056
MIWrapper	56.0	64.0	≥ 0.2	0.0208	0.0083
MISMO	57.0	63.0	≥ 0.2	≥ 0.2	0.0250
MISVM	35.0	85.0	≥ 0.2	≥ 0.2	0.0500
SimpleMI	71.0	49.0	≥ 0.2	0.0054	0.0050
TLC	69.0	51.0	≥ 0.2	0.1958	0.0167
Bagging	29.0	91.0	≥ 0.2	0.1301	0.0125
Stacking	65.0	55.0	≥ 0.2	0.0782	0.0100

Post Hoc (Friedman) comparison for $\alpha = 0.05$

4.4. Precision

Table 5 shows the precision results obtained by each algorithm. Figure 4 shows that there are no statistically significant differences between the precision results obtained by all algorithms, except SimpleMI. The p-values in Table 5 reflect the results of the Bonferroni-Dunn procedure. The Wilcoxon, Nemenyi, and Holm tests indicate that statistical significance exists for algorithms with p-values < 0.01, ≤ 0.005, and ≤ 0.005 respectively.

Table 6: Recall

Datasets	MIRSVM	MIBoost	MIOptimalBall	MIDD	MIWrapper	MISMO	MISVM	SimpleMI	TLC	Bagging	Stacking
suramin	**1.0000**	0.0000	0.4500	0.1000	0.0000	0.4500	0.5000	0.0000	0.6667	0.5500	0.9125
eastWest	**1.0000**	0.7000	0.5250	0.7500	0.7000	0.6500	0.3250	**1.0000**	0.5000	0.5500	0.4000
westEast	0.9000	0.9000	0.1500	0.5750	0.9000	0.8500	0.1250	**1.0000**	0.5000	0.8750	0.8000
musk1	**0.9787**	0.0000	0.5778	0.8444	0.0000	0.7333	0.6889	0.0000	0.8667	0.8222	0.8667
musk2	0.9250	**1.0000**	0.8710	0.8065	**1.0000**	0.7903	0.7903	**1.0000**	0.5968	0.7419	0.7742
webmining	0.2857	**1.0000**	0.9239	**1.0000**	**1.0000**	0.9130	0.6196	**1.0000**	0.8913	0.9022	0.9239
trx	0.4833	**1.0000**	0.9583	0.9464	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.9464	**1.0000**	0.9762
mutagenesis-atoms	**0.8880**	0.0000	0.3968	0.3492	0.0000	0.4921	0.0000	0.0000	0.5714	0.6508	0.5714
mutagenesis-bonds	**0.8960**	0.0000	0.5556	0.4762	0.0000	0.7460	0.0000	0.0000	0.6984	0.7778	0.7143
mutagenesis-chains	**0.9120**	0.0000	0.4444	0.5714	0.0000	0.7143	0.0000	0.0000	0.7460	0.7619	0.7460
tiger	0.8700	0.5000	**1.0000**	0.7500	0.5000	0.6700	0.7100	**1.0000**	0.7100	0.7300	0.7100
elephant	0.9100	0.6000	**1.0000**	0.7800	0.6000	0.7600	0.8600	**1.0000**	0.8000	0.8100	0.8200
fox	0.9000	0.7000	**1.0000**	0.5600	0.7000	0.4600	0.4600	**1.0000**	0.5600	0.5800	0.5900
component	0.5839	**1.0000**	0.9867	0.9797	**1.0000**	0.9967	**1.0000**	**1.0000**	0.9815	0.9867	0.9826
function	0.5327	**1.0000**	0.9919	0.9840	**1.0000**	0.9983	0.9994	**1.0000**	0.9892	0.9908	0.9894
Average	**0.8044**	0.5600	0.7221	0.6982	0.5600	0.7483	0.5385	0.6667	0.7350	0.7820	0.7851
Ranks	**4.4333**	6.2667	5.8000	6.7667	6.2667	6.2667	7.4333	4.5333	7.0333	5.3333	5.8667

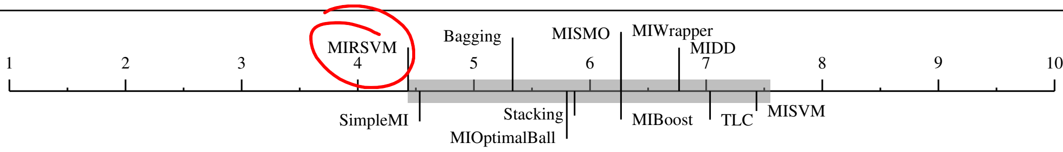


Figure 6: Bonferroni-Dunn test for Recall

Table 7: Wilcoxon, Nemenyi, and Holm tests for Recall

MIRSVM vs.	Wilcoxon R^+	Wilcoxon R^-	Wilcoxon p-value	Nemenyi p-value	Holm p-value
MIBoost	74.0	31.0	0.1937	0.1301	0.0071
MIOptimalBall	71.0	49.0	≥ 0.2	≥ 0.2	0.0167
MIDD	74.0	46.0	≥ 0.2	0.0540	0.0063
MIWrapper	74.0	31.0	0.1937	0.1301	0.0083
MISMO	70.0	50.0	≥ 0.2	0.1301	0.0100
MISVM	91.0	29.0	0.0833	0.0132	0.0050
SimpleMI	60.0	45.0	≥ 0.2	≥ 0.2	0.0500
TLC	70.0	50.0	≥ 0.2	0.0318	0.0056
Bagging	68.0	52.0	≥ 0.2	≥ 0.2	0.0250
Stacking	69.0	51.0	≥ 0.2	≥ 0.2	0.0125

Post Hoc (Friedman) comparison for $\alpha = 0.05$

4.5. Recall

Table 6 shows the recall results obtained by the algorithms, and Figure 6 and the Wilcoxon and Nemenyi tests in Table 7 show that there are no statistically significant differences between them. However, the stricter Holm test indicates that MIRSVM performs statistically better than MISVM, with p-value ≥ 0.005.

Table 8: Cohen's Kappa Rate

Datasets	MIRSVM	MIBoost	MIOptimalBall	MIDD	MIWrapper	MISMO	MISVM	SimpleMI	TLC	Bagging	Stacking
suramin	**0.6320**	0.0000	0.4500	-0.1500	0.0000	0.4500	0.0000	-0.5854	0.3810	0.3500	0.5121
eastWest	**0.6000**	0.0000	0.4500	0.2250	0.0000	0.4250	0.2750	0.0000	0.0000	0.2750	-0.1000
westEast	0.3500	0.0000	-0.2500	-0.1000	0.0000	**0.4750**	-0.0750	0.0000	0.0000	0.3750	0.2750
musk1	**0.8036**	0.0000	0.5396	0.7604	0.0000	0.5642	0.6067	0.0000	0.7174	0.7602	0.7174
musk2	**0.6263**	0.0000	0.5031	0.4039	0.0000	0.3613	0.3856	0.0000	0.2492	0.4029	0.2940
webmining	0.3468	0.0000	0.0246	0.0000	0.0000	**0.4535**	0.3771	0.0000	0.3744	0.2112	0.2458
trx	0.4542	0.0000	**0.5228**	0.4224	0.0000	0.0000	0.0000	0.0000	0.3858	0.3032	0.3364
mutagenesis-atoms	0.5395	0.0000	0.1709	0.2654	0.0000	0.2909	0.0000	0.0000	0.4738	**0.5458**	0.4431
mutagenesis-bonds	0.5699	0.0000	0.3131	0.4356	0.0000	0.5569	0.0000	0.0000	0.6195	**0.7310**	0.6659
mutagenesis-chains	0.6303	0.0000	0.2359	0.4738	0.0000	0.6225	0.0000	0.0000	0.6391	**0.6525**	0.6285
tiger	**0.5500**	0.0000	0.0000	0.4200	0.0000	0.4400	0.5100	0.0000	0.3300	0.5400	0.4500
elephant	**0.7000**	0.0000	0.0000	0.5800	0.0000	0.6200	0.6000	0.0000	0.6000	**0.7000**	0.6500
fox	**0.3100**	0.0000	0.0000	0.1600	0.0000	0.0500	0.1800	0.0000	0.2900	0.2400	0.3000
component	0.6644	0.0000	0.1613	0.2836	0.0000	0.3656	0.0675	0.0000	**0.6945**	0.6924	0.6906
function	0.6292	0.0000	0.0966	0.2801	0.0000	0.4083	0.0933	0.0000	0.7529	**0.7534**	0.7507
Average	**0.5604**	0.0000	0.2145	0.2973	0.0000	0.4055	0.2013	-0.0390	0.4338	0.5022	0.4573
Ranks	**2.2333**	9.2333	6.5333	6.1667	9.2333	4.7667	6.7667	9.4333	4.4000	3.0000	4.2333

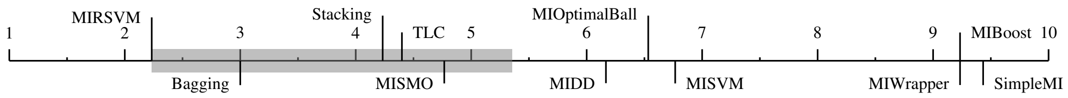


Figure 7: Bonferroni-Dunn test for Cohen's Kappa rate

Table 9: Wilcoxon, Nemenyi, and Holm tests for Cohen's Kappa rate

MIRSVM vs.	Wilcoxon R^+	Wilcoxon R^-	Wilcoxon p-value	Nemenyi p-value	Holm p-value
MIBoost	120.0	0.00	0.0001	0.0000	0.0056
MIOptimalBall	119.0	1.00	0.0001	3.8E-4	0.0083
MIDD	120.0	0.00	0.0001	0.0012	0.0100
MIWrapper	120.0	0.00	0.0001	0.0000	0.0063
MISMO	110.0	10.0	0.0026	0.0365	0.0125
MISVM	119.0	1.00	0.0001	1.8E-4	0.0071
SimpleMI	120.0	0.00	0.0001	0.0000	0.0050
TLC	97.00	23.0	0.0353	0.0736	0.0167
Bagging	73.00	32.0	0.2166	0.5267	0.0500
Stacking	97.00	23.0	0.0353	0.0987	0.0250

Post Hoc (Friedman) comparison for $\alpha = 0.05$

4.6. Kappa

Table 8 shows Cohen's Kappa Rate obtained by the algorithms. These results support the accuracy results in the sense that the instance-based and wrapper methods perform worse than bag-based and ensemble learners. Specifically, MIRSVM achieves the best kappa rate over 7 of the 15 datasets with a competitive average against the Bagging, Stacking, SMO, and TLC algorithms. MIRSVM's kappa values all fall within the range (0-1], indicating that its merit as a classifier agrees with the class distribution and is not random. Note that SimpleMI, MIOptimalBall, MIDD, MISVM, and Stacking contain some negative kappa values, indicating performance worse than the default-hypothesis. MIBoost and MIWrapper are shown to randomly classify all 15 datasets.

Figure 7 and Table 9 show the results of the statistical analysis on the Cohen's Kappa Rate results. Nemenyi's procedure indicates that MIRSVM performs significantly better than algorithms with p-value ≤ 0.005. Wilcoxon's and Holm's procedures determine statistical significance of the performance of the algorithms with p-values < 0.01 and ≤ 0.0125, respectively.

Table 10: AUC

Datasets	MIRSVM	MIBoost	MIOptimalBall	MIDD	MIWrapper	MISMO	MISVM	SimpleMI	TLC	Bagging	Stacking
suramin	0.5000	0.5000	**0.7250**	0.4250	0.5000	**0.7250**	0.5000	0.2143	0.6905	0.6750	0.6811
eastWest	**0.8000**	0.5000	0.7250	0.6125	0.5000	0.7125	0.6375	0.5000	0.5000	0.6375	0.4500
westEast	0.6500	0.5000	0.3750	0.4500	0.5000	**0.7375**	0.4625	0.5000	0.5000	0.6875	0.6375
musk1	**0.9005**	0.5000	0.7676	0.8797	0.5000	0.7816	0.8019	0.5000	0.8589	0.8792	0.8589
musk2	**0.8351**	0.5000	0.7432	0.6981	0.5000	0.6772	0.6900	0.5000	0.6317	0.7043	0.6435
webmining	0.6320	0.5000	0.5096	0.5000	0.5000	0.7184	**0.8098**	0.5000	0.6837	0.5939	0.6048
trx	0.7243	0.5000	**0.7392**	0.6932	0.5000	0.5000	0.5000	0.5000	0.6732	0.6000	0.6281
mutagenesis-atoms	0.7106	0.5000	0.5824	0.6186	0.5000	0.6420	0.5000	0.5000	0.7257	**0.7654**	0.7137
mutagenesis-bonds	0.7856	0.5000	0.6578	0.6981	0.5000	0.7850	0.5000	0.5000	0.8012	**0.8569**	0.8211
mutagenesis-chains	**0.8252**	0.5000	0.6142	0.7257	0.5000	0.8051	0.5000	0.5000	0.8170	0.8250	0.8130
tiger	**0.7750**	0.5000	0.5000	0.7100	0.5000	0.7200	0.7550	0.5000	0.6650	0.7700	0.7250
elephant	0.8200	0.5000	0.5000	0.7900	0.5000	0.8100	0.8000	0.5000	0.8000	**0.8500**	0.8250
fox	**0.6550**	0.5000	0.5000	0.5800	0.5000	0.5250	0.5900	0.5000	0.6450	0.6200	0.6500
component	0.7855	0.5000	0.5536	0.6033	0.5000	0.6272	0.5201	0.5000	**0.8123**	0.8030	0.8081
function	0.7563	0.5000	0.5298	0.6015	0.5000	0.6391	0.5268	0.5000	**0.8456**	0.8408	0.8434
Average	**0.7437**	0.5000	0.6015	0.6390	0.5000	0.6937	0.6062	0.4810	0.7100	0.7406	0.7135
Ranks	**2.7667**	9.2000	6.4000	6.2333	9.2000	4.7000	6.6667	9.4333	4.2000	3.1000	4.1000

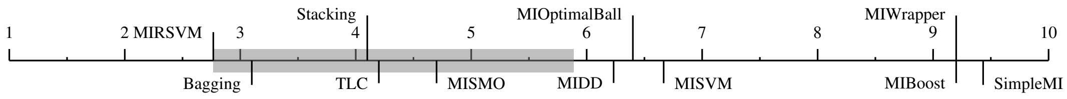


Figure 8: Bonferroni-Dunn test for AUC

Table 11: Wilcoxon, Nemenyi, and Holm tests for AUC

MIRSVM vs.	Wilcoxon R^+	Wilcoxon R^-	Wilcoxon p-value	Nemenyi p-value	Holm p-value
MIBoost	105.0	0.00	0.0001	0.0000	0.0056
MIOptimalBall	109.0	11.0	0.0034	0.0027	0.0083
MIDD	120.0	0.00	0.0001	0.0042	0.0100
MIWrapper	105.0	0.00	0.0001	0.0000	0.0063
MISMO	91.00	29.0	0.0753	0.1104	0.0125
MISVM	98.00	7.00	0.0022	0.0013	0.0071
SimpleMI	120.0	0.00	0.0001	0.0000	0.0050
TLC	75.00	45.0	0.4212	0.2366	0.0167
Bagging	60.00	60.0	1.0000	0.7831	0.0500
Stacking	78.00	42.0	0.3095	0.2709	0.0250

Post Hoc (Friedman) comparison for $\alpha = 0.05$

4.7. AUC

Table 10 shows AUC results obtained by the algorithms. These results complement the accuracy and kappa rate, emphasizing the better performance of bag-based methods. MIRSVM achieves the best AUC score on 6 of the 15 datasets, while MIBoost, SimpleMI, and MIWrapper obtain the worst results. Their AUC score indicates random predictor behavior, having values ≤ 0.5. Bag-level methods all obtain scores between 0.7 and 0.75 indicating a high true positive rate and a low false positive rate.

Figure 8 and Table 11 show that MIRSVM performs significantly better than 6 out of the 10 competing algorithms. Nemenyi's procedure rejects the numm-hypothesis for algorithms with p-value ≤ 0.005. Wilcoxon's and Holm's procedures determine statistical significance of the performance of the algorithms with p-values < 0.01 and ≤ 0.0125, respectively.

Table 13: Run Time (seconds)

Datasets	MIRSVM	MIBoost	MIOptimalBall	MIDD	MIWrapper	MISMO	MISVM	SimpleMI	TLC	Bagging	Stacking
suramin	**0.1**	8.8	30.5	7922.0	9.5	52.3	333.9	7.3	35.5	80.5	1085.0
eastWest	**0.1**	5.5	9.4	217.1	6.3	14.8	21.4	5.8	15.4	14.2	15.3
westEast	**0.1**	6.5	7.8	79.7	6.5	14.7	99.5	6.0	16.6	11.1	10.8
musk1	**0.4**	13.4	32.1	3542.6	20.6	89.7	198.4	11.1	93.0	474.3	759.5
musk2	**2.3**	97.3	782.9	126016.8	208.3	1799.4	26093.5	16.1	1772.2	14817.3	16759.0
webmining	300.6	45745.4	60474.8	47601.4	68736.7	51923.6	105622.3	2685.9	86272.6	667636.3	**10.8**
trx	61.8	17.6	682.3	339110.5	19.3	8670.3	134622.1	**7.4**	2229.3	17887.3	592948.9
mutagenesis-atoms	9.8	8.8	99.2	2623.0	8.0	55.0	53.5	**6.4**	44.0	182.8	153.9
mutagenesis-bonds	**8.3**	10.2	310.2	17538.7	12.3	457.4	2794.8	8.4	131.1	755.5	853.1
mutagenesis-chains	19.3	12.0	525.0	48982.7	14.9	2451.9	6637.4	**7.2**	224.4	1449.6	1619.0
tiger	29.5	44.5	157.8	23220.5	56.2	208.0	608.8	**16.3**	183.0	1276.7	11927.9
elephant	47.7	45.5	243.9	56456.3	69.7	232.1	1114.3	**20.8**	212.1	1030.7	1462.2
fox	81.0	44.3	206.1	27773.8	66.0	369.6	891.5	**23.5**	243.3	1332.5	1729.1
component	231.7	572.5	228209.6	96263.9	1096.9	629366.4	37224.6	**144.0**	9861.5	74860.9	79149.8
function	740.3	935.5	768458.0	350124.7	1887.5	1052225.3	565026.4	**232.8**	12128.2	138742.0	185918.5
Average	**102.2**	3171.2	70682.0	76498.2	4814.6	116528.7	58756.2	213.3	7564.1	61370.1	59626.8
Rank	2.2	2.8	6.3	10.1	3.9	7.5	8.9	**1.6**	6.3	8.1	8.4

Table 12: Rank Results

Ranks	MIRSVM	MIBoost	MIOptimalBall	MIDD	MIWrapper	MISMO	MISVM	SimpleMI	TLC	Bagging	Stacking
accuracy	**2.4000**	8.8000	7.2667	6.0333	8.8000	4.8000	7.0667	9.0000	4.7333	2.7667	4.3333
precision	6.0667	6.7333	6.7333	6.7667	6.7333	5.3667	5.1000	7.3000	5.5000	**3.9333**	5.7667
recall	**4.4333**	6.2667	5.8000	6.7667	6.2667	6.2667	7.4333	4.5333	7.0333	5.3333	5.8667
kappa	**2.2333**	9.2333	6.5333	6.1667	9.2333	4.7667	6.7667	9.4333	4.4000	3.0000	4.2333
auc	**2.7667**	9.2000	6.4000	6.2333	9.2000	4.7000	6.6667	9.4333	4.2000	3.1000	4.1000
run-time	2.2000	2.7667	6.2667	10.1333	3.9000	7.5333	8.8667	**1.6000**	6.2667	8.0667	8.4000
Average	**3.3500**	7.1667	6.5000	7.0167	7.3556	5.5722	6.9833	6.8833	5.3556	4.3667	5.4500
Rank	**2.0000**	7.7500	6.5833	8.0000	7.9167	5.3333	7.6667	7.8333	5.2500	3.0000	4.6667

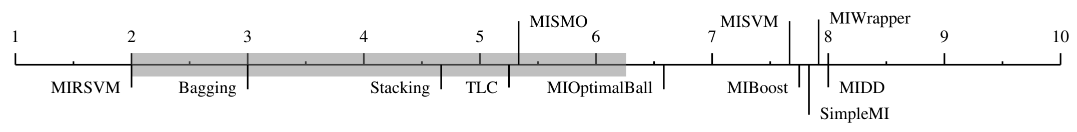


Figure 9: Bonferroni-Dunn test for Ranks

4.8. Overall Comparison

Table 13 shows the run time results, in seconds, for each algorithm. MIRSVM performs has the fastest run time and is ranked second. 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 state-of-the-art algorithms. 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 efficient as the bag-level methods.

Table 12 shows the ranks achieved by each of the metrics along with the average and meta-ranks. MIRSVM has the best meta-rank and the Bagging ensemble method has the next best. The meta-ranks also highlight the better performance of bag-level methods over instance-level and wrapper methods, emphasizing the importance of training at the bag-level. Not only does

MIRSVM use bag-level information during classification, but it also optimizes over the instances within the bag, which helps determine which instances contribute the most information about the bags label. SimpleMI, MIWrapper, MIBoost, MISVM, MIOptimalBall, and MDD have the worst performance compared to MIRSVM and Bagging. 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.

5. Conclusion

This paper proposed a novel formulation and algorithm for multiple-instance support vector machines, which optimizes over bags and bag-representatives. First, the primal formulation was posed and the dual was then derived and solved using a quadratic programming solver within the algorithm, MIRSVM. This formulation was designed to utilize bag-information and find an optimal separating hyperplane between bags, rather than instances, using the standard multi-instance assumption 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 MIRSVM are its ability to identify instances within positive and negative bags, i.e. support vectors, that highly impact the model as well as eliminating uncertainties and issues caused by techniques that flatten or subset positive instances from bags.

The experimental study shows the better performance of MIRSVM compared with traditional MI learners, as well as more recent bag-level and ensemble methods. The results were then validated using statistical analysis with non-parametric tests which indicate that MIRSVM is performs statistically better, while keeping competitive run-times. They also highlight the advantages of using bag-based and ensemble learners such as Bagging, Stacking, and MISMO, while showing the instance-based learners performed poorly in comparison or were deemed as random classifiers.

References

[1] J. Alcalá-Fdez, A. Fernández, J. Luengo, J. Derrac, S. García, L. Sánchez, F. Herrera, Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework, analysis framework, Journal of Multiple-Valued Logic and Soft Computing (2011) 255–287.

[2] J. Amores, Multiple instance classication: Review, taxonomy and comparative study, Artificial Intelligence (2013) 81–105.

[3] S. Andrews, I. Tsochantaridis, T. Hofmann, Support vector machines for multiple-instance learning, Neural Information Processing System (2002) 561–568.

[4] P. Auer, R. Ortner, A boosting approach to multiple instance learning, European Conference on Machine Learning 3201 (2004).

[5] A. Ben-David, About the relationship between roc curves and cohens kappa, Engineering Applications of Articial Intelligence 21 (2008) 874–882.

[6] L. Bjerring, 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, Springer, 2011.

[7] H. Blockeel, D. Page, A. Srinivasan, Multi-instance tree learning, in: Proceedings of the International Conference on Machine Learning, ACM, 2005, pp. 57–64.

[8] S. Boyd, L. Vanderberghe, Convex Optimization, Cambridge University Press, 2004.

[9] R. Bunescu, R. Mooney, Multiple instance learning for sparse positive bags, International Conf. on Machine Learning (2007) 105–112.

[10] M. Carbonneau, E. Granger, A. Raymond, G. Gagnon, Single- vs. multiple-instance classification, Pattern Recognition 48 (2015) 2831–2838.

[11] M. Carbonneau, E. Granger, A. Raymond, G. Gagnon, Robust multiple-instance learning ensembles using random subspace instance selection, Pattern Recognition 58 (2016) 83–99.

[12] Y. Chen, J. Bi, J. Wang, Miles: Multiple-instance learning via embedded instance selection, IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE 28 (2006) 1931–1947.

[13] Y. Chen, J. Wang, Image categorization by learning and reasoning with regions, Journal of Machine Learning Research (2004) 913–939.

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

[15] J. Derrac, S. García, D. Molina, F. Herrera, A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms, Swarm and Evolutionary Computation 1 (2011) 3–18.

[16] T. Dietterich, R. Lathrop, T. Lozano-Perez, Solving the multiple instance problem with axis-parallel rectangles, Artificial Intelligence 89 (1997) 31–71.

[17] L. Dong, A comparison of multi-instance learning algorithms, Master of Science (MSc) Thesis (2006).

[18] O. Dunn, Multiple comparisons among means, J. Amer. Stat. Assoc. 56 (Mar. 1961) 52–64.

[19] J. Foulds, E. Frank, A review of multi-instance learning assumptions, The Knowledge Engineering Review 25-1 (2010) 1–24.

[20] Z. Fu, A. Robles-Kelly, J. Zhou, Milis: Multiple instance learning with instance selection, IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE 33 (2011) 958–977.

[21] S. García, A. Fernández, J. Luengo, F. Herrera, Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power, Information Sciences 180 (2010) 2044–2064.

[22] S. García, F. Herrera, An extension on statistical comparisons of classifiers over multiple data sets for all pairwise comparisons, Journal of Machine Learning Research 9 (2008) 2677–2694.

[23] S. García, D. Molina, M. Lozano, F. Herrera, A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour - a case study on the CEC2005 Special Session on Real Parameter Optimization, Heuristics 15 (2008) 617–644.

[24] J. Gibbons, S. Chakraborti, Nonparametric Statistical Inference, 5th ed., Chapman & Hall/CRC Press, 2011.

[25] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemannr, I. Witten, The weka data mining software: An update, SIGKDD Explorations (2009) 10–18.

[26] G. Herman, G. Ye, J. Xu, B. Zhang, Region-based image categorization with reduced feature set., 10th IEEE Workshop on Multimedia Signal Processing (2008) 586–591.

[27] M. Hollander, D. Wolfe, Nonparametric Statistical Methods, John Wiley & Sons, Inc., 1999.

[28] T. Huang, V. Kecman, I. Kopriva, Kernel Based Algorithms for Mining Huge Data Sets, Supervised, Semi- supervised, and Unsupervised Learning, Springer-Verlag, 2006.

[29] V. Kecman, Learning and Soft Computing: Support Vector Machines, Neural Networks, and Fuzzy Logic Models, MIT Press, 2001.

[30] V. Kecman, Iterative k data algorithm for solving both the least squares svm and the system of linear equations, Proceedings of the IEEE SoutheastCon (2015).

[31] V. Kecman, T. Huang, M. Vogt, Iterative single data algorithm for training kernel machines from huge data sets: Theory and performance, Studies in Computational Intelligence (2005).

[32] V. Kecman, L. Zigic, Algorithms for direct l2 support vector machines, The IEEE International Symposium on INnovations in Intelligent SysTems and Applications (2014).

[33] O. M. T. Lozano-Perez, A framework for multiple-instance learning, Neural Information Processing Systems 3201 (1998) 570–576.

[34] O. Maron, T. Lozano-Perez, A framework for multiple-instance learning, Neural Information Processing System (1997) 570–576.

[35] G. Melki, V. Kecman, Speeding up online training of l1 support vector machines, IEEE SoutheastConf (2016).

[36] J. Nocedal, S. Wright, Numerical Optimization, Springer, 2006.

[37] J. Platt, Sequantial minimal optimization: A fast algorithm for training support vector machines, Technical Report: MSR-TR-98-14 (1998).

[38] S. Ray, M. Craven, Supervised versus multiple instance learning: An empirical comparison, Proceeding of ICML (2005) 697–704.

[39] B. Schoelkopf, A. Smola, Learning with Kernels; Support Vector Machines, Regularization, Optimization, and Beyond, MIT Press, 2002.

[40] S. Shalev-Shwartz, S. Ben-David, Understanding Machine Learning: From Theory to Algorithms, Cambridge University Press, 2014.

[41] P. Viola, J. Platt, C. Zhang, Multiple instance boosting for sparse positive bags, 2005.

[42] F. Wilcoxon, Individual comparisons by ranking methods, Biometr. Bull. 1 (Dec. 1945) 80–83.

[43] Y. Yi, M. Lin, Human action recognition with graph-based multiple-instance learning, Pattern Recognition 53 (2016) 148–162.

[44] A. Zafra, S. Ventura, Multi-instance genetic programming for predicting student performance in web based educational environments, Applied Soft Computing 12 (2012) 2693–2706.

[45] Q. Zhang, S. Goldman, Em-dd: An improved multiple-instance learning technique, In Proceeding of NIPS 15 (2002) 1073–1080.