NOVEL SUPPORT VECTOR MACHINES FOR DIVERSE LEARNING PARADIGMS

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

by

GABRIELLA ANGELA MELKI

Ph.D. Candidate

Director: Alberto Cano,

Assistant Professor, Department of Computer Science,

Virginia Commonwealth University

Director: Sebastian Ventura,

Professor, Department of Computer Science & Numerical Analysis,

University of Córdoba

Virginia Commonwealth University

Richmond, Virginia

September 2018

CHAPTER 1

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.

1.1 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.

1.1.1 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 $I \in \{1, \ldots, n\}$.

1.1.2 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, $f(\mathcal{B}_{n+1}) \rightarrow Y_{n+1}$ [2].

In order to build a classifier without any knowledge of the individual training instance labels, Dietterich et. al. [12] proposed the *standard MI* (SMI) hypothesis, shown in Equation (1.1), 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.

$$Y_I = \begin{cases} +1 & \text{if } \exists\, y_i = +1,\, \forall i \in I \\ -1 & \text{otherwise.} \end{cases} \tag{1.1}$$

This implies that individual instance labels y_i exist, but are not known (for positive bags) during training. Equation (1.1) can also be rewritten as Equation (1.2) for simplicity:

$$Y_I = \text{argmax}_{\forall i \in I}\ \ y_i. \tag{1.2}$$

In addition to the SMI assumption, alternative MI assumptions have been proposed to date [16]. A recent review [2] 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) [12], 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) [23] 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 [28] propose a similar framework that iteratively maximizes the DD measure. Auer and Ortner [4] 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 [4] 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 [9, 10, 11] assume the positive instances form a cluster, which may not be the case. Alternatively, Fu et al.[18] 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 [13], 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 [25] 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. [7] 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 [5] 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 [20], MIWrapper [17], and Two-Level Classifier (TLC) [26].

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 [14].

Andrews et. al. [3] 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. [3] 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. [19], who defined a bag-level kernel. A bag-level kernel determines the similarity between two bags in a higher dimensional space. Blaschko et. al. [6] 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 [2]. 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.

1.2 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 (1.3) presents the primal MIRSVM optimization problem:

$$\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \frac{1}{2}||\boldsymbol{w}||^2 + \frac{C}{n}\sum_I \xi_I, \tag{1.3}$$

$$\text{s.t. } Y_I(\langle \boldsymbol{w}, \boldsymbol{x}_{s_I} \rangle + b) \geq 1 - \xi_I, \; \forall I \in \{1, \ldots, n\} \tag{1.3a}$$

$$\xi_I \geq 0, \; \forall I \in \{1, \ldots, n\}, \tag{1.3b}$$

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 (1.3) 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 (1.3a). The proposed solution for these problems was deriving and solving the dual of the optimization problem given by Equation (1.3).

The dual can be formed by taking the Lagrangian of (1.3), given by Equation (1.4), where $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are the non-negative Lagrange multipliers.

$$\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2}\sum_{j=1}^{d} w_j^2 + \frac{C}{n}\sum_I \xi_I - \sum_I \beta_I \xi_I - \sum_I \alpha_I \left(Y_I \left(\sum_{j=1}^{d} w_j x_{s_I j} + b \right) - 1 + \xi_I \right) \tag{1.4}$$

At optimality [8], $\bigtriangledown_{w,b,\xi}\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = 0$ and the following conditions are met:

$$\frac{\partial \mathcal{L}}{\partial w_j} : w_j = \sum_I \alpha_I \, Y_I \, x_{s_I \, j}, \, \forall j \in \{1, \ldots, d\} \tag{1.5}$$

$$\frac{\partial \mathcal{L}}{\partial b} : \sum_I \alpha_I Y_I = 0, \tag{1.6}$$

$$\frac{\partial \mathcal{L}}{\partial \xi_I} : \alpha_I + \beta_I = \frac{C}{n}, \, \forall I \in \{1, \ldots, n\} \tag{1.7}$$

By substituting Equations (1.5), (1.6), and (1.7) into the Lagrangian in (1.4), the dual MIRSVM formulation becomes the following:

$$\max_{\boldsymbol{\alpha},\boldsymbol{\beta}} \sum_I \alpha_I - \frac{1}{2} \sum_I \sum_{K \in I} \sum_{j=1}^{d} \alpha_I \alpha_K Y_I Y_K x_{s_I j} x_{s_K j} \tag{1.8}$$

$$\text{s.t.} \sum_I \alpha_I Y_I = 0 \tag{1.8a}$$

$$\alpha_I + \beta_I = \frac{C}{n}, \, \forall I \in \{1, \ldots, n\} \tag{1.8b}$$

$$\alpha_I \geq 0, \, \forall I \in \{1, \ldots, n\} \tag{1.8c}$$

$$\beta_I \geq 0, \, \forall I \in \{1, \ldots, n\}, \tag{1.8d}$$

where s_I is computed for each bag, as shown in Equation (1.9):

$$s_I = \underset{i \in I}{\text{argmax}} (\sum_{K \in I} \sum_{j=1}^{d} \alpha_K Y_K x_{s_K j} x_{ij} + b), \, \forall I \in \{1, \ldots, n\}. \tag{1.9}$$

The implicit constraints (1.8b) through (1.8d) 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:

$$\max_{\boldsymbol{\alpha}} \sum_I \alpha_I - \frac{1}{2} \sum_I \sum_{K \in I} \alpha_I \alpha_K Y_I Y_K \mathcal{K}\left(\boldsymbol{x}_{s_I}, \boldsymbol{x}_{s_K}\right) \tag{1.10}$$

$$\text{s.t.} \sum_I \alpha_I Y_I = 0 \tag{1.10a}$$

$$0 \leq \alpha_I \leq \frac{C}{n}, \forall I \in \{1, \ldots, n\}. \tag{1.10b}$$

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

$$\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||^2}{2\sigma^2}} \tag{1.11}$$

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.

$$\boldsymbol{o}_I = \mathcal{K}(\mathcal{B}_I, \boldsymbol{X}_S) * (\boldsymbol{\alpha} \cdot \boldsymbol{Y}_S) + b \tag{1.12}$$

The bias term b is calculated as shown in Equation (1.13), where $\boldsymbol{sv}$ is the vector of support vector indices and n_{sv} is the number of support vectors [22].

$$b = \frac{1}{n_{\boldsymbol{sv}}} \sum_{\boldsymbol{sv}} Y_{\boldsymbol{sv}} - \mathcal{K}(\boldsymbol{X}_{\boldsymbol{sv}}, \boldsymbol{X}_{\boldsymbol{sv}}) * (\boldsymbol{\alpha}_{\boldsymbol{sv}} \cdot Y_{\boldsymbol{sv}}) \tag{1.13}$$

Algorithm 1.1 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 (1.9).

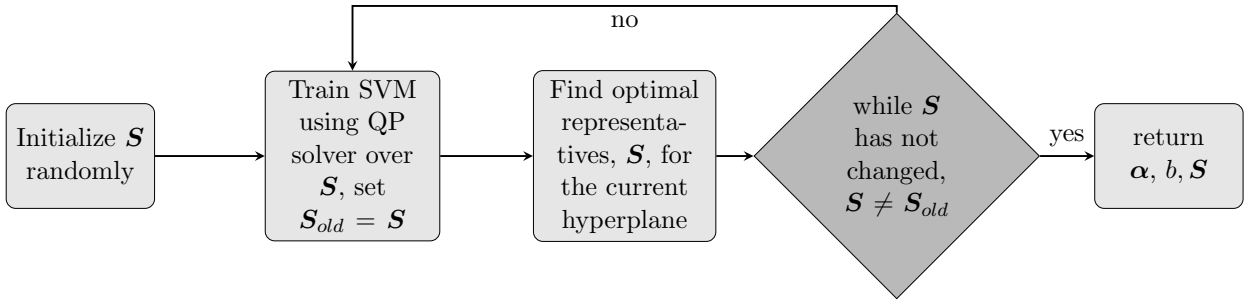

Fig. 1.1.: A summary of the steps performed by MIRSVM. The representatives are first randomly initialized and continuously updated according to the current hyperplane, which is found using a quadratic programming solver. Upon completion, the model is returned along with the optimal bag-representatives.

Algorithm 1.1 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 $I \in \{1, \ldots, n\}$ **do**
  $\boldsymbol{S}_I \leftarrow$ rand $(|\mathcal{B}_I|, 1, 1)$ ▷ Assign each bag a random instance
end for
while $\boldsymbol{S} \neq \boldsymbol{S}_{old}$ **do**
  $\boldsymbol{S}_{old} \leftarrow \boldsymbol{S}$
  $\boldsymbol{X}_S \leftarrow \boldsymbol{X}(\boldsymbol{S})$, $\boldsymbol{Y}_S \leftarrow \boldsymbol{Y}(\boldsymbol{S})$ ▷ Initialize the representative dataset
  $\boldsymbol{G} \leftarrow (\boldsymbol{Y}_S \times \boldsymbol{Y}_S) \cdot \mathcal{K}(\boldsymbol{X}_S, \boldsymbol{X}_S, \sigma)$ ▷ Build Gram matrix
  $\boldsymbol{\alpha} \leftarrow$ quadprog $(\boldsymbol{G}, -\mathbf{1}^n, \boldsymbol{Y}_S, \mathbf{0}^n, \mathbf{0}^n, \boldsymbol{C}^n)$ ▷ Solve QP Problem
  $\boldsymbol{sv} \leftarrow$ find $(0 < \boldsymbol{\alpha} \leq C)$ ▷ Get the support vector indices
  $n_{\boldsymbol{sv}} \leftarrow$ count $(0 < \boldsymbol{\alpha} \leq C)$ ▷ Get the number of support vectors
  $b \leftarrow \frac{1}{n_{\boldsymbol{sv}}} \sum_{i=1}^{n_{\boldsymbol{sv}}} (\boldsymbol{Y}_{\boldsymbol{sv}} - \boldsymbol{G}_{\boldsymbol{sv}} * (\boldsymbol{\alpha}_{\boldsymbol{sv}} \cdot \boldsymbol{Y}_{\boldsymbol{sv}}))$ ▷ Calculate the bias term
  for $I \in \{1, \ldots, n\}$ **do**
    $\boldsymbol{G}_I \leftarrow (Y_I \times \boldsymbol{Y}_S) \cdot \mathcal{K}(\mathcal{B}_I, \boldsymbol{X}_S, \sigma)$
    $\boldsymbol{S}_I \leftarrow \text{argmax}_{i \in I} (\boldsymbol{G}_I * \boldsymbol{\alpha} + b)$ ▷ 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 1.2. 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.

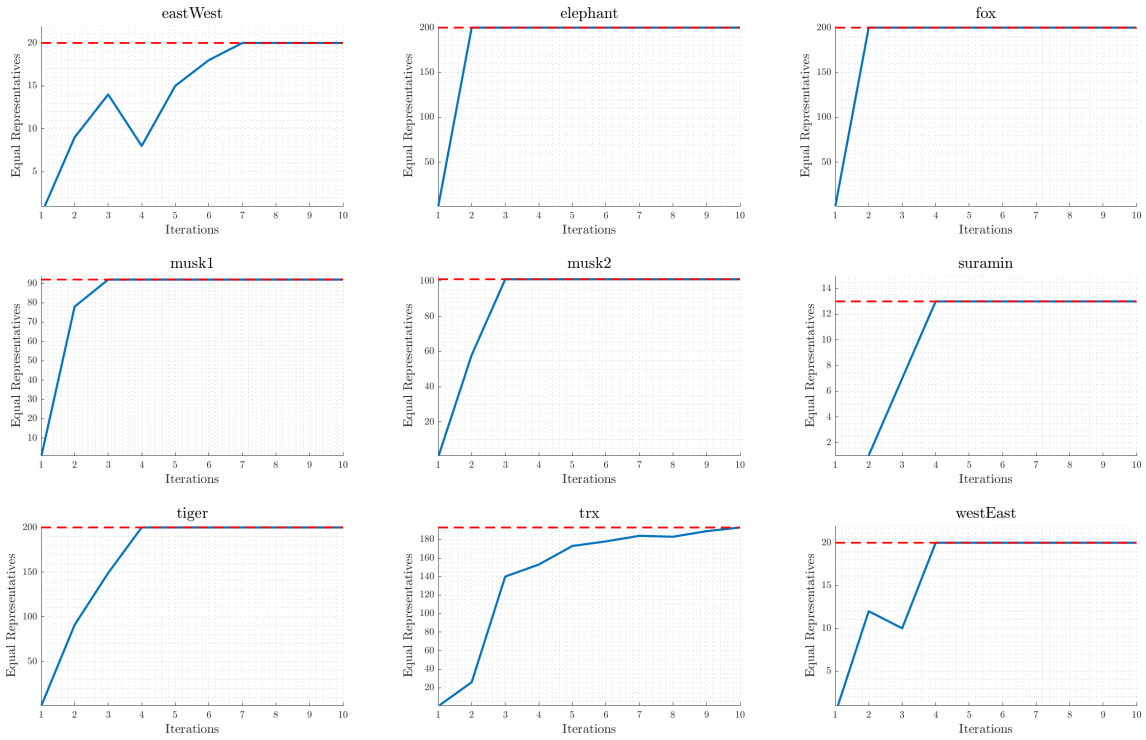

Fig. 1.2.: Bag representative convergence plots on 9 datasets. The blue line shows the number of bag representatives that are equal from one iteration to the next. The red dashed line represents the total number of bags.

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

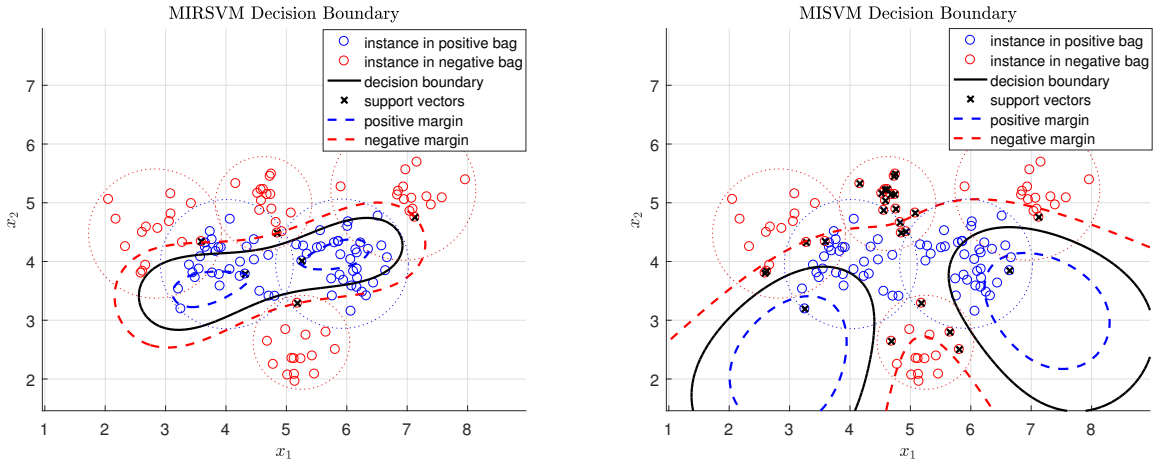

Fig. 1.3.: Difference between MIRSVM and MISVM on a random 2-dimensional toy dataset. Note the differing number of support vectors produced by the two methods. MIRSVM has 6, one for each bag, and MISVM has 29. Also note the smoother representation of the data distribution given by MIRSVM's decision boundary, 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.

the freedom to select as many negative support vectors as it needs and restricts the support vectors chosen from positive bags to be one. Figure 1.3 shows the decision boundaries produced by MIRSVM and MISVM to highlight the differences in their solutions. As Figure 1.3 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.

1.3 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 1.1 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 1.1.: 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] [20] and KEEL[2] [1] 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] [27], 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 (1.14a) and (1.14b), referred to as (1.14).

$$C \in \{0.1, 1, 10, 100, 1000, 10000\} \tag{1.14a}$$

$$\sigma \in \{0.1, 0.5, 1, 2, 5, 10\} \tag{1.14b}$$

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.

1.4 Results & Statistical Analysis

The classification performance was measured using five metrics: Accuracy (1.15a), Precision (1.15b), Recall (1.15c), Cohen's kappa rate (1.15d), and Area under ROC curve (AUC) (1.15e). 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 (1.15d), 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 (1.15e). 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 (1.15) 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 1.2, 1.4, 1.6, 1.8, 1.10, and 1.12.

$$\text{Accuracy} \qquad \frac{TP + TN}{n'} \tag{1.15a}$$

$$\text{Precision} \qquad \frac{TP}{TP + FP} \tag{1.15b}$$

$$\text{Recall} \qquad \frac{TP}{TP + FN} \tag{1.15c}$$

$$\text{Cohen's Kappa Rate} \qquad \frac{n' - \frac{(TP + FN) * (TP + FP)}{n'}}{1 - \frac{(TP + FN) * (TP + FP)}{n'}} \tag{1.15d}$$

$$\text{Area Under ROC Curve} \qquad \frac{1 + \frac{TP}{TP + FN} - \frac{FP}{FP + TN}}{2} \tag{1.15e}$$

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 (1.15) are presented in the last row of the results tables, and the lowest (best) rank value is typeset in bold. Table 1.13 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 [15] 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 1.4, 1.5, 1.6, 1.7, 1.8, 1.9 show the results of the Bonferroni-Dunn post-hoc procedure over the metrics in (1.15), as well as the meta-rank results in Table 1.13. The Holm (multiple) and Wilcoxon (pairwise) rank-sum post-hoc tests [21] 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 1.3, 1.5, 1.7, 1.9, and 1.11 show the p-values for the Holm test for $\alpha = 0.05$, and the rank-sums and adjusted p-values for the Wilcoxon test.

1.4.1 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 1.4 and Table 1.3 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 1.4, perform statistically worse than MIRSVM. Table 1.3 shows the p-values of the Holm and Wilcoxon tests and their results com-

Table 1.2.: 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

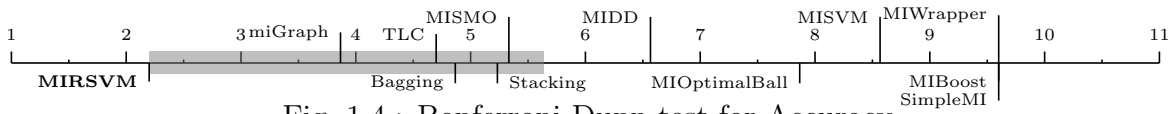

Fig. 1.4.: Bonferroni-Dunn test for Accuracy

Table 1.3.: 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.

1.4.2 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 1.4.: 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

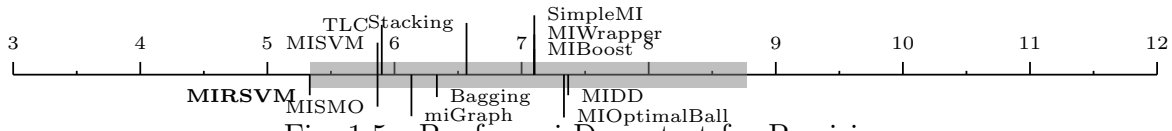

Fig. 1.5.: Bonferroni-Dunn test for Precision

Table 1.5.: 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

Figure 1.5 and 1.6 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

Table 1.6.: 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

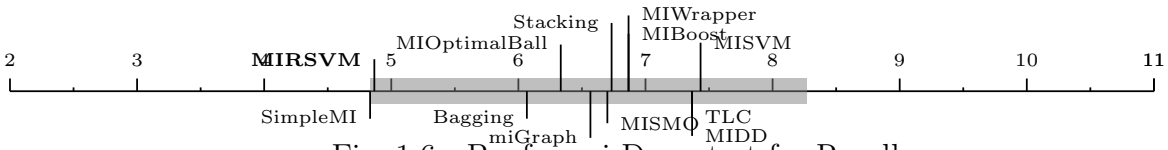

Fig. 1.6.: Bonferroni-Dunn test for Recall

Table 1.7.: 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

behavior, whereas MIRSVM demonstrates proper balance of the precision-recall trade-off.

1.4.3 Cohen's Kappa Rate

Table 1.8 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,

Table 1.8.: 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

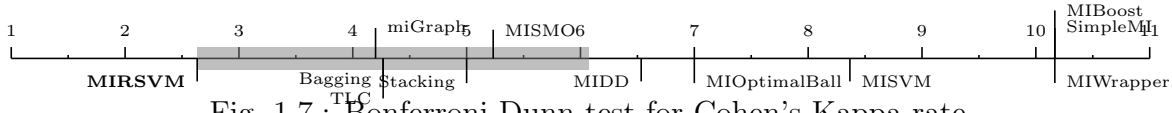

Fig. 1.7.: Bonferroni-Dunn test for Cohen's Kappa rate

Table 1.9.: 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

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 1.7 and Table 1.9 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.

1.4.4 Area Under ROC Curve

Table 1.10 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, 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 1.8 and Table 1.11 show that MIRSVM performs significantly better

Table 1.10.: 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. 1.8.: Bonferroni-Dunn test for AUC

Table 1.11.: 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

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.

1.4.5 Overall Comparison

Table 1.12 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 1.2.

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 1.13 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 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

Table 1.12.: 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 1.13.: 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

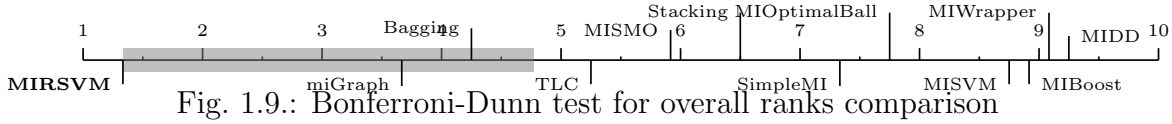

Fig. 1.9.: Bonferroni-Dunn test for overall ranks comparison

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.

1.5 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 [24].