2013

MULTIPLE-INSTANCE AND ONE-CLASS RULE-BASED ALGORITHMS

Dat Nguyen
Virginia Commonwealth University

MULTIPLE-INSTANCE AND ONE-CLASS RULE BASED ALGORITHMS

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

by

DAT TIEN NGUYEN

B.S., University of Science, HoChiMinh city, Vietnam, 1991

M.S., University of Science, HoChiMinh city, Vietnam, 1997

Director: KRZYSZTOF J. CIOS

PROFESSOR AND CHAIR

DEPARTMENT OF COMPUTER SCIENCE

Virginia Commonwealth University

Richmond, Virginia

May, 2013

Acknowledgement

I would like to express my sincere gratitude to my advisor Dr. Krzysztof J. Cios for his continuous help and comments during the entire period of my graduate studies. It would have been impossible to complete my dissertation without his guidance. I am grateful to all of my doctoral committee members, Drs. Hobson, Kecman, Arodz, Kurgan and Ventura, for their helpful comments and also to Dr. Cao Nguyen.

I would also like to thank all professors from whom I have learned a lot of valuable knowledge during my time as a Ph.D. student: Drs. Cios, Kecman, Arodz, Najarian and Hobson. I am grateful to all of them and am sure that this knowledge will help me in my future career.

Most of all, I would like to dedicate this thesis to my parents and my family. I could not have invested the valuable time needed to complete my dissertation without their continued support.

Table of Contents

Page

List of Tables v

List of Figures ix

Abstract xi

Introduction 1

Chapter

1 Inductive Machine Learning 3

1.1 Inductive rule learners 3

1.2 Classical rule learners 6

2 Multiple-Instance Learning Algorithm: mi-DS 11

2.1 State of the art 11

2.2 mi-DS algorithm 18

2.3 Experiments 28

2.3.1 Experiment I 31

2.3.2 Experiment II 36

2.4 Summary 43

3 One-class Learning Algorithm: OneClass-DS 44

3.1 State of the art 44

3.1.1 Density estimation methods 46

3.1.2 Boundary methods 46

3.1.3 Reconstruction methods 48

3.1.4 Rule-based methods 49

3.2 OneClass-DS algorithm 50

3.3 Experiments 59

3.3.1 Experiments I 60

3.3.2 Experiments II 70

3.3.3 Experiments III 73

3.3.4 Experiments IV 77

3.4 Summary 83

4 Conclusions 84

Bibliography 86

Appendix

A Measures Used For Evaluating Goodness of Generated Models 96

A.1 General 96

A.2 Statistical 98

Wilcoxon Signed-Ranks Test 98

Friedman Test 101

B mi-DS and OneClass-DS Algorithms Implementation Details 105

List of Tables

Page

Table 2.1 Similarity matrix constructed in phase II. M_{ij} is the number of the rules covering the bags b_i and b_j; $(n+1)^{th}$ row and $(n+1)^{th}$ column are used during the subsequent prediction. 20

Table 2.2 Five training data bags. 23

Table 2.3 *POSITIVE* and *NEGATIVE* tables. 24

Table 2.4 *POSITIVE* and *NEGATIVE* tables (after remove inconsistent instances)....25

Table 2.5 *RULEPOS* and *RULENEG* tables; * indicates any value of a feature. 26

Table 2.6 Similarity matrix *M* obtained in phase II. 27

Table 2.7 Test bag *B* with 5 instances in it. 27

Table 2.8 Similarity matrix *M* with scores for the test bag *B*, with *R*=2 and *C*=2. 28

Table 2.9 Description of the data sets used in comparisons. 31

Table 2.10 Accuracy measures of mi-DS and 9 comparison MIL algorithms. 10-FCV is used to calculate accuracy (%). The best results are shown in bold with rank shown below in brackets. 33

Table 2.11 MCC measures of mi-DS and 9 comparison MIL algorithms. 10-FCV is used to calculate MCC. The best results are shown in bold with rank shown below in brackets.. 34

Table 2.12 Wilcoxon and Friedman test results for comparing 9 MIL algorithms on 13 data sets. 35

Table 2.13 Comparison with 9 MIL (in term of accuracy) algorithms on Text and Corel Images data; the best results for each data set are shown in bold. 39

Table 2.14 Comparison with 9 MIL algorithms (in terms of MCC) on Text and Corel Images data; the best results for each data set are shown in bold. 40

Table 2.15 Wilcoxon and Friedman test results for comparing 9 MIL algorithms on Text and Corel Images data. 41

Table 3.1 Training data instances. 54

Table 3.2 Summed-up values for features shown in Table 3.1. 55

Table 3.3 Instances in one-class training data set covered by (F1 =1). 55

Table 3.4 Summed-up values for features shown Table 3.3. 56

Table 3.5 Training data instances that remain after deleting the instances covered by the first rule. 56

Table 3.6 Summed-up values for features shown in Table 3.5. 57

Table 3.7 Instances in Table 3.5 covered by (F1 =1). 57

Table 3.8 Summed-up values of features (for instances in Table 3.7). 57

Table 3.9 Training data instances after deleting instances covered by the first two rules. 58

Table 3.10 Data sets used in experiments. 61

Table 3.11 Results of OneClass-DS algorithm on 11 data sets. 62

Table 3.12 Comparison of OneClass-DS with five other algorithms in terms of accuracy. 64
Table 3.13 Comparison of OneClass-DS with five other algorithms in terms of the IPR and FAR measures. 65
Table 3.14 Wilcoxon and Friedman tests results for comparing OneClass-DS with five algorithms on 11 data sets. 69
Table 3.15 Comparison of OneClass-DS with OneClass SVM, OCC-Gauss and OCC-EM. 71
Table 3.16 Wilcoxon and Friedman tests results for comparing OneClass-DS with 3 algorithms on 9 data sets. 72
Table 3.17 Data sets with missing values. 73
Table 3.18 OneClass-DS results on data with missing values. 74
Table 3.19 Comparison of OneClass-DS with REP-Tree, Decision Stump and Random Tree on four missing value data sets. 76
Table 3.20 Letter and digits data sets. 78
Table 3.21 Results on the Pen digits data set. 79
Table 3.22 Results on the Optical digits data set. 80
Table A.1 Misclassification matrix. 96
Table A.2 Most often used measure calculated from the confusion matrix. 97
Table A.3 Results, in terms of accuracy, of algorithms A and B on 13 data sets. 99
Table A.4 Part of Wilcoxon test table. 101

Table A.5 Results of 6 algorithms on 11 data sets. 102

Table A.6 Ranking of the 6 algorithms on 11 data sets. 103

Table A.7 Part of the chi-squared (χ2) distribution table 104

List of Figures

Page

Figure 1.1 Pseudocode of *DataSqueezer* algorithm. 8

Figure 2.1 Pseudocode of *mi-DS* algorithm. 18

Figure 2.2 Pseudocode, *PredictTestingBag*, for labeling test bag *B*. 22

Figure 2.3 Dinosaurs from Corel's data: before preprocessing (left column) and after preprocessing (right column). 37

Figure 2.4 Images in Corel's data sets after preprocessing. 37

Figure 2.5 Accuracy of mi-DS on data with missing values. 42

Figure 3.1 Training and test scenarios 51

Figure 3.2 Pseudocode of *OneClass-DS* algorithm. 53

Figure 3.3 Influence of changing the four parameters values on the Target rate, Precision, IPR and FAR, on the Heart data set. 63

Figure 3.4 Possible outcomes of a test in terms of Precision & Target rate (y-axis) vs. *MaxAttribute* (x-axis). 67

Figure 3.5 Possible outcomes of a test in terms of Precision& Target rate (y-axis) vs. *MaxAttribute* (x-axis). 75

Figure 3.6 Precision of letter A with different values of *MaxAttribute* and *MinCoverTarget*. 81

Figure 3.7 Target Rate of letter A with different values of *MaxAttribute* and *MinCoverTarget*. 81

Figure 3.8 Relation between Precision and Target Rate for different values of the *MaxAttribute* (Letter A). 82

Figure B.1 Data Mining Tools web page. 105

Figure B.2 WEKA packages' web page. 106

Figure B.3 Menu Package manger in WEKA. 107

Figure B.4 Package Manager window… 107

Figure B.5 Choose the package to install. 108

Figure B.6 Installing chosen package in WEKA. 109

Figure B.7 Open data set in WEKA. 109

Figure B.8 Choose data set in WEKA. 110

Figure B.9 Musk 1 data set. 110

Figure B.10 Select mi-DS and setting values of parameters. 111

Figure B.11 Results using 10-FCV. 112

Figure B.12 Results using 90% of instances for training and 10% for testing. 112

Abstract

MULTIPLE-INSTANCE AND ONE-CLASS RULE-BASED ALGORITHMS

By Dat Tien Nguyen, Ph.D.

A dissertation submitted in partial fulfillment of the requirements for the degree of Degree of Doctor of Philosophy in Engineering (Computer Science track) at Virginia Commonwealth University.

Virginia Commonwealth University, 2013

Major Director: Krzysztof J. Cios
Professor and Chair, Department of Computer Science

In this work we developed rule-based algorithms for multiple-instance learning and one-class learning problems, namely, the mi-DS and OneClass-DS algorithms. Multiple-Instance Learning (MIL) is a variation of classical supervised learning where there is a need to classify bags (collection) of instances instead of single instances. The bag is labeled positive if at least one of its instances is positive, otherwise it is negative. One-class learning problem is also known as outlier or novelty detection problem. One-class classifiers are trained on data describing only one class and are used in situations where data from other classes are not available, and also for highly unbalanced data sets.

Extensive comparisons and statistical testing of the two algorithms show that they generate models that perform on par with other state-of-the-art algorithms.

Introduction

Supervised machine learning is an important branch of artificial intelligent where instances are labeled as positive or negative by a teacher during the training task. However, in practice it is not always possible for the teacher to provide labels for all training instances. In classical supervised learning, there are two main drawbacks in the labeling task. First, the teacher does not know the label of an individual instance and second, the teacher has only instances for a single class (positive or negative) of labels.

The first problem is modeled as multiple instance learning (MIL) and is becoming increasingly important within machine learning. Unlike traditional supervised learning in which each individual instance is labeled, in multiple instance learning model a bag of instances is labeled as to whether any single instance is positive. This model was encountered by Dietterich et al. (1997) in the task of the drug activity prediction problem where each instance is a possible configuration for a molecule of interest and each bag contains all low-energy configurations for the molecule. To date, there are many in interesting applications in use of MIL scenario in addition to Dietterich's work. For examples, the learning simple description of a person from a series of images containing that person, the stock selection from highest return stocks and so on.

The problem in the second drawback is the one-class classification which is to formulate a target set of instances in a training dataset and to detect new instances similar to the training set. As such, the difference with traditional classification is that in one-class classification only instances of one class (target) are available. All other instances are the outliers.

In this thesis, we hypothesize that the above two problems can be solved using traditional supervised learning methods such as support vector machine, neural networks, Bayesian networks and/or classical rule-based learners. The motivation of the present study is to investigate and solve the issues raised by introducing multiple instance learning and one class problems into rule-based algorithms, in particular DataSqueezer (Kurgan et al., 2006) .

The thesis is organized as follows. Chapter 1 introduces and defines Inductive Machine Learning methods. Chapter 2 presents the novel Multiple-Instance Learning Algorithm: mi-DS. Chapter 3 details the novel OneClass-DS learning algorithm in solving One-class classification. The last Chapter is dedicated to Conclusions.

CHAPTER 1 Inductive Machine Learning

1.1 Inductive rule learners

Over last few years ML attracted considerable attention due to the demand for reliable and useful data mining techniques in the information technology, medical, decision making, and gaming industries, to name but a few. Machine learning (ML) is frequently used to solve classification problems, perform intelligent data analysis, and in development of diagnostic, decision support, and knowledge-based systems (Langley and Simon, 1995; Paliouras et al., 2001). ML methods are divided into two major groups, supervised and unsupervised (like clustering). Here we deal only with supervised methods.

ML methods are frequently used in the knowledge discovery process because of their many desired characteristics. We define knowledge discovery as a nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns from large collections of data (Fayyad et al., 1996). In view of this definition ML is one of the key tools used to perform data mining tasks (Fayyad et al., 1996) (Cios and Kurgan, 2004). ML is often defined as the ability of a computer program to improve its own performance at some task based on past experience, and as such is a very attractive vehicle for discovering patterns in the data and classification.

There is a class of ML algorithms called rule induction systems (or rule learners) and they are the subject of the work presented here. A rule induction system takes as input a set of training examples, and produces a set of production rules in the form *IF conditions THEN actions*. Rule induction is distinct from decision trees. It is easy to write a set of production rules, each specifying a path from the root to a leaf node, from a generated decision tree. However, a set of *IF…THEN…* rules generated by a rule learner cannot be converted into a decision tree (it forms only a graph). Rule learners exhibit a number of desirable properties:

– They generate rules that are easy for people to understand, which is a desirable property since they are more comprehensible than decision trees (Chisholm and Tapedalli, 2002). People often learn from the hypotheses generated by a rule learner, provided the hypothesis is in a human-comprehensible form. In this case, experts can participate in the learning process by critiquing or improving the learned hypotheses.

– Rule learners often outperform decision trees in some domains (Pagallo and Haussler, 1990).

– The output of a rule leaner can easily be translated into a first-order logic representation, or embedded within the knowledge-based systems (Cohen, 1995; Cios and Kurgan, 2004).

– Certain types of prior knowledge can be easily incorporated into rule learners (Pazzani and Kibler, 1992).

– The rules can be modified and analyzed because of their modularity, i.e., a single rule can be understood without reference to other rules (as is the case with decision trees), which is important when a decision maker needs to understand and validate the generated rules, for example in medical application (Holsheimer and Siebes, 1994).

Supervised rule learners can be sub-divided into three scenarios, depending on the type of training data available. In the first scenario, the “classical” one, each instance in training data set belongs to only one of the classes. For instance, we may have data describing patients with diseases 1, 2 and 3; thus training data pairs are (patient1, disease1), (patient 2, disease3), etc.

In the second scenario, called Multiple Instance Learning (MIL), the training data are very different (Dietterich et al., 1997). Namely, instead of each patient having only one diagnosis he/she can have many diagnoses over a period of time. These diagnoses put together form a “bag” of diagnoses for this patient. Now, the task is to come up with a model to be used later for predicting whether a new patient has a certain disease (among several other diseases). We developed a new algorithm for this scenario and it is discussed in Chapter 2.

In the third scenario, known as one-class learning or outlier detection, we have data representing only one class, say of some unique disease (Cohen et al, 2004). Our task is to come up with a model to be used for predicting whether a new patient has or not this particular disease. In Chapter 3 we describe the algorithm developed for this learning scenario.

1.2 Classical rule learners

There are dozens of classical inductive rule learner algorithms that can be grouped into decision tree algorithms such as C4.5 (Quinlan, 1993), CART (Breiman et al., 1984), ID3 (Quinlan, 1986),… and the "true" rule learners such as RIPPER (Cohen, 1995), DLG (Webb and Agar, 1992), IREP (Furnkranz and Widmer, 1994)…

Below we describe one classical rule learner, namely DataSqueezer (Kurgan et al., 2006), which induces a set of production rules from a training data set as it constitutes a basis for the developed algorithms: for MIL problems and the other for one-class learning problems. The main advantages of DataSqueezer are its log-linear complexity and robustness to missing values, which commonly are present in big data. Below we review in some detail the DataSqueezer algorithm because it is used as a component of the developed new algorithms for more difficult learning tasks of one-class and MIL classification problems.

Let us denote training data set by D. It consists of s instances and k features. D includes two sets: positive set – D_P and negative set D_N. Both these set must satisfy conditions: $D_P \cup D_N = D$, $D_P \cap D_N = \varnothing$, $D_P \neq \varnothing$, $D_P \neq \varnothing$. From the training data set D, we distribute all instances into table *POS* that contains only positive instances and table *NEG* that contains only negative instances. DataSqueezer algorithm has two steps. First, it calls function DataReduction with table *POS* and then table *NEG* to generalize information stored in these two tables. In the second step, it creates *RULES* which contains rules for positive instances in data set. Pseudocode of DataSqueezer is shown in Figure 1.1. Vectors and matrices (tables) are denoted by capital letters, while their elements are denoted by lower-case letters (Kurgan et al., 2006).

```
DataSqueezer
Input: POS, NEG, k , s
Step 1.
          $G_{POS}$ =  DataReduction(POS,k);
          $G_{NEG}$= DataReduction(NEG,k);
Step 2.
2.1       Initialize RULES=[ ] ; i=1;
2.2       Create LIST =  List of all columns in $G_{POS}$
2.3       Within every $G_{POS}$ column that is on LIST, for every non missing value a from
          selected column j compute sum, $S_{aj}$, of values of $gpos_i[k+1]$ for every row I, in
          which a appears and multiply $S_{aj}$, by the number of values the feature j has.
2.4       Select maximal $S_{aj}$, remove j from LIST, add "j=a" selector to $rules_i$
2.5       If  $rule_i$ does not described any rows in $G_{NEG}$
2.6       then     Remove all rows described by $rule_i$ from $G_{POS}$, i=i+1;
2.7                If $G_{POS}$ is not empty  go to 2.2  else terminate
2.8       else  go to 2.3
Output: RULES

DataReduction // data reduction procedure for D=POS or D=NEG
Input: D, k
      Initialize G = [ ]; i = 1; tmp = $d_1$; $g_1$ = $d_1$; $g_1[k+1]$ = 1;
      for j=1 to $N_D$ //for positive/negative set: $N_D$ is $N_{POS}$ or $N_{NEG}$
           for kk=1 to k  // for all features
               if ($d_j[kk]$<>tmp[kk] or $d_j[kk]$='*') // '*' : missing "do not care" value
               then tmp[kk]='*';
           if (number of non-missing values in tmp>=2) then $g_i$ = tmp; $g_i[k+1]$++;
                          else  i++; $g_i$ = $d_j$; $g_i[k+1]$ = 1; tmp = $d_j$;
Output: G
```

Figure 1.1 Pseudocode of *DataSqueezer* algorithm

where

> *POS*: Table of positive instances
> N_{POS}: number of positive instances
> *NEG*: Table of negative instances
> N_{NEG}: number of positive instances
> G_{POS}: Table store results of *POS* after DataReduction
> G_{NEG}: Table store results of *NEG* after DataReduction
> S_{aj} : number of times value *a* in column *j* occurred
> *k*: number of features in data set
> *s*: number of instances
> *RULES*: rule for positive instances
> d_i: ith row of table *D* (similarly for g_i, $gpos_i$, $rules_i$)
> $d_i[j]$: jth column of ith row in table *D* (similarly for $g_i[j]$)
> *tmp[i]*: ith column of row *tmp*

In step 1, the learner performs data reduction to generalize information stored in the original data (Kurgan et al., 2006). Its algorithm performs data reduction to generalize information stored in the original data via use of the prototypical concept learning, which is based on the Find algorithm of Mitchell (Mitchell, 1997). It is performed for both positive and negative data and results in generation of the tables. The reduction procedure is also related to the least generalization, as used by the DLG learner (Webb and Agar, 1992). The main difference is that the least generalization is applied multiple times for the entire positive set through a beam search procedure, while DataSqueezer performs it once in a linear fashion by generalizing consecutive examples. Also, the DLG learner does not generalize the negative set.

In step 2, the learner generates rules by performing greedy hill-climbing search on the reduced data: A rule is generated by applying the search procedure starting with an empty rule, and adding selectors until the termination criterion fires. The rule, while being generated, consists of selectors generated using data set, and is checked against the data set.

If the rule covers any data in the data set, a new selector is added to the rules making it more specific, and thus able to better distinguish between positive and negative data. Next, the examples covered by the generated rule are removed, and the process is repeated. For more details the reader is referred to (Kurgan et al., 2006).

In Chapters 2 and 3 we describe the developed algorithms for MIL and one-class learning problems, respectively.

CHAPTER 2 Multiple-Instance Learning Algorithm: mi-DS

2.1 State of the Art

The multiple-instance learning (MIL) problem was introduced by (Dietterich et al., 1997). It is concerned with classifying bags of instances instead of single instances. A bag is labeled positive if at least one instance within it is positive and negative if all instances are negative. The goal of MIL algorithms is to correctly label new test bags. Standard supervised learning classifiers do not work on the MIL problems because although they can learn all correct negative bag instances, they cannot distinguish between the true- and false-positive instances in the positive bags. Several MIL scenarios exist:

- a positive bag can include one or more different true positive instances
- the number of true positive instances in a positive bag may be greater or smaller than the number of false positive instances
- the true positive instances may have similar or quite different feature values.

To illustrate MIL consider the locksmith problem which a locksmith must determine if a keychain is useful. The keychain is assumed positive (useful) if at least one of the keys opens the door, otherwise it is negative (useless). Dietterich et al. formalized MIL for the classification of aromatic molecules for drug design. The goal was to classify each molecule, represented by a bag of possible conformations, according to whether or not it was musky. MIL methods have been developed and used in applications such as image retrieval and annotation (Qi and Han, 2007; Chiang and Cheng, 2009; Zhang et al., 2008), failure prediction (Murray et al., 2005), and bioinformatics for protein-ligand docking (Mizianti et al., 2010). Key MIL algorithms are briefly reviewed below.

Many approaches for solving MIL problems are based on probabilistic models. One of them, the Diverse Density (MI-DD) algorithm (Maron and Lozano-Perez, 1998) attempts to find a concept point in the feature space that is close to at least one instance from every positive bag but far away from instances in the negative bags. The optimal concept point has maximum diversity density ("a measure of how many different positive bags have instances near that point, and how far the negative instances are from that point", (Maron and Lozano-Perez, 1998)). Thus, the concept point describes a region of the instance space that is dense in terms of instances from the positive bags. Another algorithm, the Expectation Maximization and Diverse Density (MI-EMDD) (Zhang and Goldman, 2001) extends the MI-DD method and forms a generic framework that can be used to convert a MIL problem into a single-instance setting by using the Expectation Maximization (EM) algorithm. The label of a bag is determined by the instance with the highest likelihood of being positive among all instances in that bag. The authors use a set of hidden variables that are estimated using the EM approach to find out which instance determines the label of a given bag. Starting with an initial guess of the concept point h, obtained by checking points from the positive bags, the MI-EMDD iteratively performs the following two steps. 1) E-step: the current hypothesis h is used to select one instance from each bag which is the most likely to be responsible for the label given to the bag. 2) M-step: a new instance h' is estimated to maximize the diverse density of the hypothesis *h* by using a gradient search. The two steps are repeated until the algorithm converges (when the diverse density of the hypothesis is $h' < h$). Also within the framework of probabilistic approach a linear logistic regression algorithm was long used in standard single-instance supervised machine learning to fit a linear model to the log-odds of the class probabilities. At classification time a new instance is assigned to the class which corresponding linear function value achieves maximum among all the classes. However, for the MIL problems the standard logistic regression model cannot be directly used thus an indirect estimate of the logistic model (MI-LR) was proposed by (Xu and Frank, 2004). They extended the standard instance-based logistic regression model to a bag-level model under the assumptions indicating how the instance-level class probabilities were combined to form the bag-level probability so that the actual class label for each instance was not required.

A different approach is based on support vector machine (SVM) classifiers. SVM is a supervised learning algorithm (by Vapnik, 1995) and it was used for solving MIL problems as follows. In the standard single-instance SVM the training data is provided as a set of instance-label pairs (b_i, y_i), where b_i is an instance and y_i is its label, which are then non-linearly mapped to a higher-dimensional space F. The SVM algorithm aims to find the maximum margin hyperplane in *F* that linearly separates two classes. Sequential Minimal Optimization (MI-SMO) method (Platt, 1998) extended the standard SVM for MIL problems by using a bag-level multi-instance kernel function. First, the bag-level kernel is defined and the bag-label pairs (B_i, Y_i) can then be used instead of instance-label pairs (b_i, y_i). (Andrews et al., 2002) introduced another SVM-based approach, the MI-SVM algorithm. The main idea was to transform the multi-instance data setting into a single-instance setting by properly assigning the unobserved class label to each individual instance in the positive bags. Then the standard single-instance SVM learning scheme was used for assigning the labels. The goal was to find the maximum-margin multi-instance separating hyperplane in which all instances in each negative bag are located on one side of the hyperplane and at least one positive instance from all positive bags is located on the other side of the hyperplane.

Another way to solve MIL problem is based on using distance measures. MI-OptimalBall method (Auer and Ortner, 2004) was introduced as a weak learner implemented within the boosting framework. The main idea was to find an optimal ball in the feature space such that all negative bags were outside of this ball. In other words, the surface of the ball separates positive and negative instances/concepts. The center of the optimal ball is an instance from a positive bag and the radius of the ball is determined based on the training data. During classification, if all instances in a test bag lie outside of the optimal ball then the bag is classified as negative, otherwise as positive. The authors of Citation-KNN (Wang and Zucker, 2000) used the K-nearest neighbor (KNN) algorithm to compute the shortest distance between any two instances of each bag. Specifically, they used Hausdorff metric at the bag-level to calculate the distance:

$$Dist(A,B) = \min_{1 \le i \le n,\ 1 \le j \le m} (Dist(a_i,b_j)) = \min_{a \in A} \min_{b \in B} ||a-b||$$

where: A *and* B denote bags;

n and m are the total number of instances in bags A and B, respectively;

a_i, b_j are instances in each bag.

The new bags are labeled using the KNN algorithm. However, in the MIL scenario, the majority label of the *k* nearest neighbors of an unlabeled bag is not always the true label of that bag. This is because the majority voting scheme may not work in the presence of false positive instances in the positive bags. This weakness was overcome by adding a citation approach which considered not only the bags as the nearest neighbors (known as references) of a given bag *B*, but also the bags that count B as their neighbors (known as citations). Hence, the Citation-KNN predicts the label of a bag based on the labels of both, the references and citers, of that bag. The performance of Citation-KNN was on par with MIL algorithms such as Diverse Density (Maron and Lozano-Perez, 1998) and Axis Parallel Rectangle (Dietterich et al., 1997). Unfortunately, Citation-KNN cannot be used to predict the labels of all individual instances. The Multiple-Instance Nearest Neighbor with Distribution Learner (MI-NND) method (Xu and Li, 2007) assumes that each bag contains enough instances and that all dimensions of the data are equally relevant to classification. Under these assumptions a distribution is derived for each dimension of each bag and the obtained distributions are used directly for classification, instead of doing it on the original data. MI-NND operates in two steps. The first step formulates distribution for each bag based on the training data by deriving a Gaussian model for each dimension of each bag. These Gaussian distributions are then used to represent the original data. The second step finds the nearest neighbors for a test bag where the test bag is also represented by the Gaussian distributions for each dimension. Next, the testing and training distributions are compared using Kullback-Leibler distance and the category is decided based on the closest match (classification process is just the same as in the standard KNN algorithm).

More recently, several other MIL algorithms were proposed. One, the Multiple-Instance Learning via Embedded instance Selection (MILES) (Chen et al., 2006) converted MIL problem to standard supervised learning by mapping each bag into a feature space defined by the instances in the training bags using instance similarity measure and the 1-norm SVM (Zhu et al., 2003) to solve it. Other two methods were also based on the SVM but used deterministic annealing for identifying all labels (AL-SVM) and for identifying the witness (AW-SVM) (Gehler and Chapelle, 2007). Based on the extended Random Forest algorithm (Breiman, 2001), the mi-Forest defined labels of all instances in every positive bag as random variables and used deterministic annealing procedure to find the true labels (Leistner et al., 2010).

The above methods were developed using either the probabilistic EM approaches, or nearest-neighbor approaches, or regression, or by extending the SVM classifier. Here, we propose a new learner, mi-DS, based on our rule classifier called DataSqueezer (Kurgan et al., 2006). mi-DS uses rules generated by DataSqueezer as a metric to measure the distance between two bags in the training data. Our main goal was to develop a method that builds predictive models that perform better or on par with the existing MIL methods. Importantly, the proposed approach can be used as a generic framework to transform other rule-based algorithms for solving MIL problems.

2.2 mi-DS algorithm

Pseudocode of the mi-DS is shown in Figure 2.1

```
mi-DS
Input: training bags {$b_i$}, number of features k, total number of instances s

        ScanInRuleTable
        Input: Training bag $b_k$, rule table RULE, matrix M
            FOR (every instance i  in bag $b_k$)
                FOR (each rule r in RULE)
                    IF (instance i  covered by rule r)
                        FOR (each bag $b_j$ IN covered bags of the rule r) AND (k<>j)
                            then  M[i,j]++

        SimilarityMatrix
        Input: List of training bags {$b_i$}
        Initialize the matrix M with n+1 rows and n+1 columns
        FOR (k=1 to n)
            ScanInRuleTable($b_i$, RULEPOS, M)
            ScanInRuleTable($b_i$, RULENEG, M)
        Output: Similarity matrix M

Step 1
        Create POSITIVE table that contains all instances of all positive bags
        Create NEGATIVE table that contains all negative instances of negative bags
        Remove inconsistent instances from POSITIVE table and update s.
Step 2
        RULEPOS ← DataSqueezer (POSITIVE, NEGATIVE, k, s)
        RULENEG ← DataSqueezer (NEGATIVE, POSITIVE, k, s)
        M ← SimilarityMatrix ( {$b_i$}, RULEPOS, RULENEG )
Output: RULEPOS, RULENEG, similarity matrix M
```

Figure 2.1 Pseudocode of *mi-DS* algorithm.

mi-DS algorithm consists of two major steps that are described in detail below.

In Step 1 tables for positive and negative instances are created. Namely, we create *POSITIVE* table that contains all instances of all positive bags and all of them (every instance in this table) are labeled as positive. We also create *NEGATIVE* table that contains all instances of all negative bags and all of them (every instance in this table) are labeled as negative. We then convert all bags into two tables containing positive instances and negative instances, respectively, like in a classical binary classification problem. Inconsistent instances (identical instances that appear in both negative and positive bags) are then removed from the *POSITIVE* table.

In Step 2 mi-DS calls DataSqueezer to generate the Positive (*RULEPOS*) and Negative (*RULENEG*) rule tables. The *RULEPOS* table thus contains rules for "positive" instances that, in fact, cover both the true positive and false positive instances because of the MIL definition of the positive bag. *RULENEG* table, however, covers only true negative instances. After performing data reduction, a rule is generated by incrementally adding features by checking the *POSITIVE* table against the *NEGATIVE* table for *RULEPOS*, and vice versa for the *RULENEG*. A feature with the highest summed up value, which is computed using the number of occurrence of that particular feature multiplied by the number of distinct values of the feature, is selected and added incrementally to a rule as a selector until the rule does not describe any rows in the *NEGATIVE* table. Similarly for the *NEGATIVE* table. Next, the rows described by the generated rule are removed from the

POSITIVE table and the process repeats. During this phase of rule generation the rules are collected along with the information about which bags are covered by the rules.

Two thresholds are used to address the bias-variance dilemma. First, a pruning threshold is used to control the rule generation process to keep the rulesfrom becoming too specific. This was done by not allowing a rule to add more selectors than the number specified by the threshold's value. Second, a generalization threshold is used to allow for generation of rules that cover a small number of negative examples.

After the *RULEPOS* and RULENEG tables are generated we build a similarity matrix *M* that measures the similarity between two bags, as explained in Table 2.1. The similarity matrix has *n+1* rows and *n+1* columns, where *n* is the number of bags in the training data set. The value M_{ij} stored in this matrix represents the total number of positive and negative rules such that the bags b_i and b_j are covered by the rules. In other words, the bag b_i refers M_{ij} times to the bag b_j or the bag b_j is cited M_{ij} times by the bag b_i. $(n+1)^{th}$ row and the $(n+1)^{th}$ column are used to store the numbers of References and Citations, which are used later to make a decision about a test bag label.

Table 2.1 Similarity matrix constructed in step 2. M_{ij} is the number of the rules covering the bags b_i and b_j; $(n+1)^{th}$ row and $(n+1)^{th}$ column are used during the subsequent prediction.

Row/Col	*1*	*2*	*3*	...	*n*	*n+1*
1	*0*	M_{12}	M_{13}	...	M_{1n}	
2	M_{21}	*0*	M_{23}	...	M_{2n}	
....	...	...	...	*0*	...	
n	M_{n1}	M_{n2}	M_{n3}	...	*0*	
n+1						

A measure was created to quantify similarity between the test bag and the training bags and decide a label for the new bag. This was achieved by modification of the measures described in (Wang and Zucker, 2000), so that it can be used within the rule-based models. Specifically, instead of using the metric distances between the bags, the number, u_{ij}, of rules that cover all the bags is used. For a given test bag, first fill the $(n+1)^{th}$ row and the $(n+1)^{th}$ column in the similarity matrix, in the same way as in phase II but only for the last row and the last column of the similarity matrix (see Table 2.1). To make a decision about class of a test bag both the number of References *(R)* and the number of Citations *(C)*must be used. To do so, first count, in row *(n+1)*, the bags that have *R* maximum values. The purpose is to find bags (both positive and negative) that the test bag most often referred to. Similarly, next scan all rows of the similarity matrix and count the *C* maximum values with the restriction that one of them must come from column *(n+1)*, the rule that covers the Test bag. The purpose is to find bags which the test bag *B* most often cited.

$$P= R_p + C_p$$

$$N= R_n + C_n$$

where R_p (R_n) is the number of positive (negative) bags in the *R* selected bags and C_p (C_n) is the number of positive (negative) bags in the *C* selected bags. The test bag is predicted as positive if $P > N$, otherwise it is negative. When $P = N$ there are different ways of labeling unknown bags. A simple algorithm was developed for this purpose. The pseudocode for the just described procedure is given in Figure 2.2.

```
PredictTestingBag
Input: test bag B, RULEPOS, RULENEG, matrix M)
    CalculateSimilarityofTestingBag(B, RULEPOS,RULENEG, matrix M)
    IF (P > N)                    then    B is positive
    ELSE IF (P < N)               then    B is negative
    ELSE IF ($R_p > R_n$)         then    B is positive
    ELSE IF ($R_p < R_n$)         then    B is negative
    ELSE IF ($C_p > C_n$)         then    B is positive
    ELSE IF ($C_p < C_n$)         then    B is negative
    ELSE IF ( # of positive bags > #number of negative bags in {$b_1,b_2,..,b_n$})
                        THEN  B is positive
    ELSE                B is negative
Output: Predicted label of testing bag B.

CalculateSimilarityofTestingBag
Input: Testing bag B, Rule tables RULEPOS and RULENEG, matrix M.
    Set $(n+1)^{th}$ row and $(n+1)^{th}$ column in the similarity matrix M to 0
    ScanInRuleTable(B,RULEPOS)
    ScanInRuleTable(B,RULENEG)
    Build list REF of bags which include the top  R nearest references to B
    Build list CITER of bags which include the top C nearest citers of B
    Calculate P, N, $R_p$, $R_n$, $C_p$ and $C_n$  from REF and CITER
Output: P,N,$R_p$,  $R_n$, $C_p$, $C_n$
```

Figure 2.2 Pseudocode, *PredictTestingBag*, for labeling test bag *B*.

Computational complexity of the mi-DS is analyzed next. In Step 1, the mi-DS requires $O(N_{pos}N_{neg})$ computations, where N_{pos} is the number of instances in the positive bags and N_{neg} is the number of instances in the negative bags. The complexities of Step 2, (see Kurgan et al., 2006) are $O(R_{pos}KN_{pos}logN_{pos})$ for generating the *RULEPOS* table and $O(R_{neg}KN_{neg}logN_{neg})$ for generating the *RULENEG* table, where R_{pos} and R_{neg} are the total numbers of rules in the *RULEPOS* and *RULENEG*, respectively, N is the total number of instances, and K is the number of features. The construction of the similarity matrix takes $O(NR_{total}K)$, where R_{total} is the total number of rules. Therefore, the computational complexity of the algorithm is approximately $O(R_{total}\,KN\,\log N)$.

Data shown in Table 2.2 are used to illustrate the working of the mi-DS algorithm.

Table 2.2 Five training data bags.

Bag #	Features			Class label
	Shape	Color	Width	
1	Rect	Green	200	+
	Circle	Blue	400	
	Circle	Green	300	
2	Circle	Green	300	-
	Rect	Blue	200	
	Triangle	Blue	200	
3	Rect	Green	200	+
	Circle	Red	300	
	Triangle	Blue	200	
4	Rect	Blue	200	-
	Triangle	Blue	200	
5	Rect	Green	200	+
	Circle	Blue	300	
	Rect	Blue	200	

Step 1

Create *POSITIVE* and *NEGATIVE* tables from the given training bags. Results are shown in Table 2.3.

Table 2.3 *POSITIVE* and *NEGATIVE* tables.

POSITIVE table

Features			Number of times occurred	Bag #
Shape	Color	Width		
Rect	Green	200	1	1
Circle	Blue	400	1	1
Circle	**Green**	**300**	1	1
Rect	Green	200	1	3
Circle	Red	300	1	3
Triangle	**Blue**	**200**	1	3
Rect	Green	200	1	5
Circle	Blue	300	1	5
Rect	**Blue**	**200**	1	5

NEGATIVE table

Features			Number of times occurred	Bag #
Shape	Color	Width		
Circle	**Green**	**300**	1	2
Rect	Blue	200	1	2
Triangle	Blue	200	1	2
Rect	**Blue**	**200**	1	4
Triangle	**Blue**	**200**	1	4

Inconsistent instances are shown in bold in Table 2.3 (i.e., (Circle, Green, 300), (Triangle, Blue, 200) and (Rect, Blue, 200)) are removed from the *POSITIVE* table but are kept in the *NEGATIVE* table. Results are shown in Table 2.4. The bag number and the count of the number of occurrences of each instance are also shown in Table 2.4.

Table 2.4 *POSITIVE* and *NEGATIVE* tables (after remove inconsistent instances).

POSITIVE table

Features			Number of times occurred	Bag #
Shape	Color	Width		
Rect	Green	200	1	1
Circle	Blue	400	1	1
Rect	Green	200	1	3
Circle	Red	300	1	3
Rect	Green	200	1	5
Circle	Blue	300	1	5

NEGATIVE table

Features			Number of times occurred	Bag #
Shape	Color	Width		
Circle	**Green**	**300**	1	2
Rect	Blue	200	1	2
Triangle	Blue	200	1	2
Rect	**Blue**	**200**	1	4
Triangle	**Blue**	**200**	1	4

Step 2

In this step, *RULEPOS* and RULENEG tables are generated, see Table 2.5; the table also shows bag numbers covered by these rules.

Table 2.5 *RULEPOS* and *RULENEG* tables; * indicates any value of a feature.

RULEPOS

Features			Number of times occurred	Bags
Shape	Color	Width		
Rect	Green	*	3	{1,3,5}
Circle	Blue	*	2	{1,5}
Circle	Red	*	1	{3}

RULENEG

Features			Number of times occurred	Bags
Shape	Color	Width		
*	Blue	200	4	{2,4}
Circle	Green	*	1	{2}

Now we construct the similarity matrix (as seen in Table 2.6) as follows. For example, bag 1 in Table 2.2 has three instances. Instance 1 (Rect,Green,200) is covered by rule (Rect,Green,*) in *RULEPOS* table; because this rule also covers bags 3 and 5 thus increase *M[1,3]* and *M[1,5]*. Instance 2 (Circle,Blue,400) is covered by rule (Circle,Blue,*) in *RULEPOS*; because it also covers bag 5 *M[1,5]* is increased. Instance 3 (Circle,Green,300) is covered by rule (Circle,Green,*) in *RULENEG*; because it also covers bag 2 we increase *M[1,2]* is increased. This results in the first row in Table 2.6 being (0,1,1,0,2).

Table 2.6 Similarity matrix *M*.

Bag# (label)	1(+)	2(-)	3(+)	4(-)	5(+)	Test bag
1(+)	0	1	1	0	2	
2(-)	0	0	0	2	0	
3(+)	1	1	0	1	2	
4(-)	0	2	0	0	0	
5(+)	2	1	1	1	0	
Test bag						

Next, classification of a test bag *B* is performed shown in Table 2.7. First, every instance in the bag is scanned to see whether it is covered by any rule from either *RULEPOS* or *RULENEG* tables. If an instance in bag *B* is covered by a rule then the values of *M[n+1,j]* and *M[j,n+1]* are incremented for every bag *j* in the bag lists (shown in the last column of Table 2.4) for both *RULEPOS* and *RULENEG* tables.

Table 2.7 Test bag *B* with 5 instances in it.

Shape	Color	Width
Rect	Green	300
Circle	Green	100
Rect	Blue	200
Circle	Blue	100
Rect	Green	100

For example, the instance (Rect, Green, 300) in bag *B* is covered by rule (Rect, Green, *) from the *RULEPOS* table, so increase *M[n+1,j]* and *M[j,n+1]* is increased by 1 with bag numbers being *j* = 1,3,5. The last row and the last column of the similarity matrix are updated as shown in Table 2.8. Finally, the algorithm shown in Figure 2.2 is used with *R*=2 and *C*=2, and the resulting R_{set} and C_{set} values are:

R_{set} = {bag1(+), bag5(+)}

C_{set} = {bag1(+), bag2(-), bag3(+), bag4(-), bag5(+)}.

Label of the test bag *B* is then predicted based on the labels of bags in R_{set} and C_{set}. Since R_p=2, R_n=0, C_p=3, C_n=2, $P=R_p+C_p$=5, and $N=R_n+C_n$=2, the test bag *B* is predicted as positive.

Table 2.8 Similarity matrix *M* with scores for the test bag *B*, with *R*=2 and *C*=2.

Rule covering bag# (label)	*1(+)*	*2(-)*	*3(+)*	*4(-)*	*5(+)*	Test bag *B*
1(+)	0	1	1	0	2	3
2(-)	0	0	0	2	0	2
3(+)	1	1	0	1	2	2
4(-)	0	2	0	0	0	1
5(+)	2	1	1	1	0	3
Test bag *B*	3	2	2	1	3	0

2.3 Experiments

The mi-DS's performance is evaluated by comparing its performance with several MIL algorithms using accuracy. For that purpose a diverse set of 26 benchmark data sets are used: Musk1 and Musk2 (Dietterich et al., 1997), three data sets concerning mutagenicity (Srinivasan et al., 1994), three data sets concerning content-based image retrieval (Andrews et al., 2002), one data set for protein identification problem (Wang et al. 2004), two data sets to predict whether a train is eastbound or westbound (Michalski and

Larson, 1977), two artificial data sets and three data sets for text categorization (Ray and Craven, 2005), and on Corel Corp. preprocessed image data. The data sets are summarized in Table 2.9. They are characterized by different number of features ranging from 3 to 231, number of instances ranging from 213 to 118,417, and different ratios of positive to negative instances, ranging from 0.08 to 3.3.

The Musk data sets, from UC Irvine repository (Asuncion and Newman, 2007) are often used in evaluation of MIL algorithms: the task is to predict whether a given molecule emits a musky odor, where each bag describes one molecule (Dietterich et al., 1997). The mutagenesis data concern analysis of drug activity with the goal of predicting whether a given drug molecule is mutagenic or non-mutagenic: the molecules are represented by atoms, atomic bonds, and chains (Srinivasan et al., 1994). The next three data sets concern identification of the intended target objects in images. They include bags representing photographs of animals from a Corel data set (Andrews et al., 2002). An image is represented by a set of segments (pixel regions) that are characterized by color, texture, and shape descriptors. A bag represents an image and instances in a bag represent individual segments of that image. We use image data of elephants, foxes, and tigers; a positive bag is based on an image that contains a given animal. Data set Trx was used to solve protein identification problem (Wang et al. 2004). The data sets EastWest and WestEast are used to predict whether a train is eastbound or westbound (Michalski and Larson, 1977). Two artificial data sets are also used, Artificial 1 includes 200 bags with 100 positive and 100 negative bags; each bag has 20 instances with 12 features uniformly assigned as 0 or 1

(Chevaleyre and Zucker, 2001). The concept is described by the first three features, (1,0,1), namely, if there exists in a bag at least one instance with the format of (1,0,1) with the remaining features taking on any value, then the bag is a positive. The second artificial data set includes 100 positive and 100 negative bags, each with 20 instances. Each instance has 2 features which were drawn randomly from a [0,100] x [0,100] in R^2. An instance is labeled as positive if the features fell in the square 5x5 in the middle of the domain, namely, in [48,52] x [48,52]. The bag is labeled as positive if at least one instance fells within this region and as negative otherwise (Maron and Lozano-Perez, 1998). Ray and Craven introduced text categorization data. Given a name of a protein and a full-text article from a biomedical journal, the task is to determine whether this protein-article pair can be annotated with a particular Gene Ontology (GO) term. For MIL setting each article is represented as a bag. An instance in a bag refers to a paragraph in the corresponding article. Each paragraph is represented as a set of word occurrence frequencies plus data about the nature of the protein-GO relationship. The assumption is that if there is at least one instance (paragraph) that is related to the protein-GO relationship, the bag is positive, otherwise it negative. There are three data sets that correspond to three categories of the GO terms, namely, the Component, Function, and Process.

Table 2.9 Description of the data sets used in comparisons.

Experiment #	Data sets	# features	# of bags/ Positive bags/ Negative bags	# of instances/ Positive instances/ Negative instances
I	Musk1	167	92/47/45	476/207/269
	Musk2	167	102/39/63	6598/1017/5581
	Atoms	11	188/125/63	1618/1073/545
	Bonds	17	188/125/63	3995/2955/1040
	Chains	25	188/125/63	5349/4116/1233
	Elephant	231	200/100/100	1391/762/629
	Fox	231	200/100/100	1320/647/673
	Tiger	231	200/100/100	1220/676/544
	Trx	8	193/25/168	26611/3341/23270
	East-West	24	20/10/10	213/81/132
	West-East	24	20/10/10	213/81/132
	Artificial 1	13	200/100/100	4000/2000/2000
	Artificial 2	3	200/100/100	4000/2000/2000
II	Corel Corp. image data sets	9	250/50/200	variable
	Component	200	3130/423/2707	36894/9104/27790
	Function	200	5242/443/4799	55536/5543/49993
	Process	200	11718/757/10961	118417/9272/109145

2.3.1 Experiment I

The mi-DS algorithm with MI-DD (Maron and Lozano-Perez, 1998), MI-EMDD (Zhang and Goldman, 2001), MDD (Dietterich et al., 1997), MI-LR (Xu and Frank, 2004), MI-SVM (Andrews et al., 2002), MI-SMO (Platt, 1998), MI-OptimalBall (Auer and Ortner, 2004), Citation-KNN (Wang and Zucker, 2000) and MI-NND (Xu and Li, 2007) on the first 13 data sets are compared. Weka (Witten and Frank, 2005) is used for implementations of those algorithms. All experiments are performed on a 2.4 GHz CPU with 3GB RAM and each algorithm is parameterized to maximize its accuracy using 10-fold cross validation.

Table 2.10 shows performance of these algorithms in terms of accuracy. It is worth noting that discretization was performed only on 8 training data sets (out of 13). Results of each method on each data set are ranked according to accuracy. It is easy to observe that no method is universally best. The average rank, over all data sets, of mi-DS is 6.4, which is the second best after MI-DD with 6.2. The third best method, MI-SMO, has average accuracy rank of 7.3. Interestingly, mi-DS performed best 4 times on the 13 data sets while other methods were best at most once. The Citation-KNN algorithm on the Elephants, Tigers and Fox data sets, as well as MI-SVM on the three mutagenesis data sets, generate only positive labels, which explains their low accuracies.

Table 2.11 shows performance comparison in terms of the MCC (Matthews Correlation Coefficient); see Appendix A for its definition. It shows that mi-DS performed best on 7 out of 13 data sets, and the second best in 2 out of the remaining 6. Average rank of mi-DS is the smallest (value of 2.8), while the next best is MI-SMO (value of 4.3).

Table 2.10 Accuracy measures of mi-DS and 9 comparison MIL algorithms. 10-FCV is used to calculate accuracy (%). The best results are shown in bold with rank shown below in brackets.

#	Alg / Data set	Musk 1	Musk 2	Ele-phan t	Fox	Tiger	Atoms	Bonds	Chains	Trx	East West	West East	Artifi-cial 1	Artifi-cial 2	Avg. Rank
1	mi DS	86.7 [5]	77 [8]	**80 [1]**	74.7 [6]	79.5 [3]	79.5 [5]	64.5 [2]	64.3 [9]	89 [2]	64.2 [5]	60 [3]	**100 [1]**	**68.8 [1]**	3.9
2	Citation KNN	**90.4 [1]**	84.6 [3]	73.2 [4]	75.4 [4]	74.1 [7]	50 [10]	50 [8]	50 [10]	87.6 [4]	45 [10]	50 [4]	60.5 [3]	67 [2]	5.4
3	MI DD	84.7 [6]	80.4 [6]	72.9 [5]	76 [3]	79.9 [2]	81.5 [3]	59.4 [4]	72.2 [4]	**90 [1]**	64.5 [4]	36 [8]	76.1 [2]	43.2 [10]	4.5
4	MDD	77.6 [8]	73 [9]	71.7 [6]	72.5 [8]	77.1 [5]	78.4 [7]	**64.8 [1]**	67.7 [6]	87.1 [5]	56.5 [8]	49 [5]	51.3 [10]	51.8 [4]	6.3
5	MI EMDD	83.6 [7]	**85.6 [1]**	68.9 [8]	73.4 [7]	71.5 [8]	75.3 [8]	59.7 [3]	71.4 [5]	87.9 [3]	64 [6]	38 [7]	56 [6]	51.8 [4]	5.6
6	MI NND	75.1 [9]	72.8 [10]	45.6 [10]	31.2 [10]	41.2 [10]	74.8 [9]	58.5 [5]	66.5 [7]	87 [6]	57 [7]	**74 [1]**	55.2 [8]	45.9 [8]	7.7
7	MI-SMO	87 [4]	84.9 [2]	71 [7]	**82.4 [1]**	**85.2 [1]**	**82.4 [1]**	55.2 [7]	**80.3 [1]**	86.1 [8]	74 [2]	**74 [1]**	54.3 [9]	49.1 [7]	3.9
8	MI-SVM	89.2 [2]	83.9 [4]	66.5 [9]	66.5 [9]	66.5 [9]	81.6 [2]	49.4 [9]	75.7 [2]	87 [7]	52 [9]	22 [10]	55.5 [7]	50 [6]	6.5
9	MI-LR	73.5 [10]	78.5 [7]	74 [3]	76.6 [2]	78.3 [4]	78.9 [6]	57.4 [6]	75.3 [3]	85.3 [9]	67 [3]	35.5 [9]	57.8 [5]	52.7 [3]	5.4
10	MI-Optimal Ball	87.5 [3]	83.2 [5]	75.3 [2]	75.3 [5]	75.3 [6]	79.6 [4]	48.7 [10]	66.3 [8]	84.9 [10]	**79 [1]**	45.5 [6]	58.7 [4]	43.6 [9]	5.6

Table 2.11 MCC measures of mi-DS and 9 comparison MIL algorithms. 10-FCV is used to calculate MCC. The best results are shown in bold with rank shown below in brackets.

#	Alg / Data set	Musk 1	Musk 2	Ele-phant	Fox	Tiger	Atoms	Bonds	Chains	Trx	East West	West East	Artifi-cial 1	Artifi-cial 2	Avg. Rank
1	mi DS	0.877 [2]	0.521 [7]	**0.802 [1]**	**0.731 [1]**	**0.731 [1]**	**0.58 [1]**	0.589 [2]	**0.561 [1]**	0.204 [8]	-0.14 [6]	0.201 [4]	**1 [1]**	**0.402 [1]**	2.8
2	Citation KNN	**0.878 [1]**	0.645 [3]	0 [10]	0 [9]	0 [10]	0.434 [3]	0.317 [6]	0.331 [7]	-0.204 [10]	0 [4]	0.011 [10]	0.214 [3]	0.381 [2]	6.0
3	MI DD	0.858 [3]	0.356 [10]	0.653 [2]	0.315 [3]	0.531 [3]	0.332 [6]	0.413 [5]	0.445 [3]	0.301 [5]	-0.2 [7]	**0.527 [1]**	0.531 [2]	-0.138 [8]	4.5
4	MDD	0.59 [5]	0.478 [8]	0.607 [4]	0.411 [2]	0.492 [5]	0.237 [8]	0.278 [8]	0.426 [4]	0.25 [7]	0.218 [2]	0.187 [5]	0.026 [10]	0.039 [4]	5.5
5	MI EMDD	0.795 [4]	**0.807 [1]**	0.477 [8]	0.218 [5]	0.455 [8]	0.375 [5]	0.306 [7]	0.389 [5]	0 [9]	-0.502 [9]	0.108 [8]	0.12 [7]	0.039 [4]	6.2
6	MI NND	0.503 [10]	0.369 [9]	0.593 [5]	0.09 [8]	0.526 [4]	-0.109 [10]	-0.069 [10]	-0.02 [10]	0.408 [2]	0.104 [3]	0.108 [8]	0.103 [9]	-0.144 [9]	7.5
7	MI-SMO	0.506 [9]	0.658 [2]	0.61 [3]	0.203 [6]	0.636 [2]	0.29 [7]	**0.673 [1]**	0.494 [2]	0.4 [3]	**0.4 [1]**	0.187 [5]	0.111 [8]	-0.041 [7]	4.3
8	MI-SVM	0.584 [6]	0.56 [4]	0.559 [6]	-0.003 [10]	0.477 [6]	0 [9]	0 [9]	0 [9]	0.333 [4]	-0.436 [8]	0.187 [5]	0.14 [6]	0 [6]	6.8
9	MI-LR	0.559 [8]	0.54 [6]	0.559 [6]	0.194 [7]	0.456 [7]	0.508 [2]	0.538 [3]	0.379 [6]	0.301 [5]	0 [4]	0.291 [2]	0.16 [5]	0.058 [3]	4.9
10	MI-Optimal Ball	0.572 [7]	0.548 [5]	0.469 [9]	0.258 [4]	0.365 [9]	0.38 [4]	0.432 [4]	0.324 [8]	**0.436 [1]**	-0.524 [10]	0.204 [3]	0.192 [4]	-0.157 [10]	6.0

The rank provides information about relative performance of the algorithms. However, to test whether mi-DS performs significantly different than other algorithms two statistical tests are used, the Wilcoxon signed-ranked test (Demsar, 2006) and the Friedman test (Friedman, 1940; Demsar, 2006). Both are described in detail, including toy examples how to use them in Appendix A. The results are summarized below.

Wilcoxon test is used to investigate statistical significance of the differences between pairs of algorithms, one always being mi-DS and the other, in turn, each of the remaining nine MIL algorithms. Friedman test is used to compare many algorithms at the same time. The tests are performed using Table 2.10 (accuracies) and Table 2.11 (MCC). The results are shown in Table 2.12.

Table 2.12 Wilcoxon and Friedman test results for comparing 9 MIL algorithms on 13 data sets.

	Accuracy $\alpha = 0.05$, $N = 13$	**MCC** $\alpha = 0.05$, $N = 13$
Wilcoxon Test (See Appendix A for details)	The null-hypothesis that a given pair of algorithms performs equally well is rejected when ***T* ≤17 (in 5 cases).** In other words, mi-DS performs better than 5 algorithms.	The null-hypothesis that a given pair of algorithms performs equally well is rejected when ***T* ≤17 (in 8 cases).** In other words, mi-DS performs better than 8 algorithms.
Friedman test (See Appendix A for details)	The null hypothesis that all algorithms perform at the same level is rejected because the calculated *Fr* value (18.10) is bigger than the critical value (16.92). In other words, mi-DS is performing significantly different than other algorithms.	The null hypothesis that all algorithms perform at the same level is rejected because the calculated *Fr* value (22.87) is bigger than the critical value (16.92). In other words, mi-DS is performing significantly different than other algorithms.

Pair of compared algorithms	R+	R–	T=min(R+, R-)
mi-DS vs. Citation-KNN	79	12	**12**
mi-DS vs. MI-DD	59	32	32
mi-DS vs. MDD	84	7	**7**
mi-DS vs. MI-EMDD	75	16	**16**
mi-DS vs. MI-NND	81	10	**10**
mi-DS vs. MI-SMO	43	48	43
mi-DS vs. MI-SVM	76	15	**15**
mi-DS vs. MI-LR	70	21	21
mi-DS vs. MI-Optimal Ball	63	28	28

Pair of compared algorithms	R+	R–	T=min(R+, R-)
mi-DS vs. Citation-KNN	83	8	**8**
mi-DS vs. MI-DD	78	13	**13**
mi-DS vs. MDD	77	14	**14**
mi-DS vs. MI-EMDD	83	8	**8**
mi-DS vs. MI-NND	82	9	**9**
mi-DS vs. MI-SMO	64	27	27
mi-DS vs. MI-SVM	86	5	**5**
mi-DS vs. MI-LR	75	16	**16**
mi-DS vs. MI-Optimal Ball	83	8	**8**

From Table 2.12 it can be concluded that in terms of Wilcoxon test, using accuracy as the performance measure, mi-DS performs significantly better than Citation-KNN, MDD, MI-EMDD, MI-NND and MI-SVM. It performs on par with the MI-DD, MI-SMO, MI-LR and MI-OptimalBall algorithms.

When comparing MCC as the performance metric, Wilcoxon test tells us that mi-DS performs better than other 8 algorithms except MI-SMO.

While Friedman tests does not determine which algorithm is better or worse, it can show significant difference between MIL algorithms' performances. That is to say the mi-DS is significantly different from the other 9 algorithms, when analyzing performance in terms of both the accuracy and the MCC.

While the Wilcoxon and Friendman tests perform different comparisons, the same conclusion can be drawn, mi-DS performs better than or on par with the other MIL algorithms.

2.3.2 Experiment II

In the following experiments, the MIL algorithms are used on the three text data sets (see Table 2.9). Note that the positive and negative class distributions in these data sets are highly unbalanced. However, to balance the classes during training the same number of positive and negative bags are used, namely, 359 for Component, 385 for Function, and 620 for Process data sets. The remaining bags are used for testing, namely, 64 positive and 2348 negative bags for Component, 58 positive and 4414 negative for Function, and 137 positive and 10341 negative for Process.

The mi-DS algorithm is also tested on preprocessed Corel Corp. image data sets. After preprocessing, images are considered as bags, each containing from 3 to 13 instances of the same object (e.g., dinosaur), and are described by 9 features extracted from the images by (Chen and Wang, 2004). Images belong to 10 categories, with 100 images per category, and are stored in the JPEG format; sizes of the images are 384x256 or 256x384. Example dinosaurs' images, both unprocessed and processed, are shown in Figure 2.3.

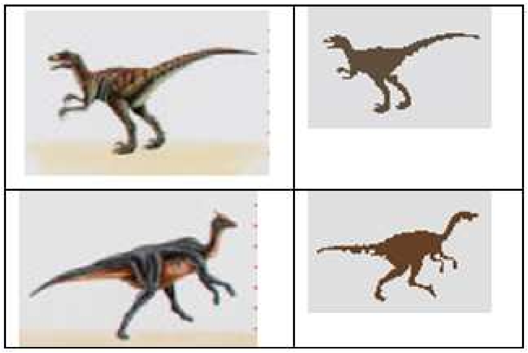


Figure 2.3 Dinosaurs images from Corel's data: before preprocessing (left column) and after preprocessing (right column).

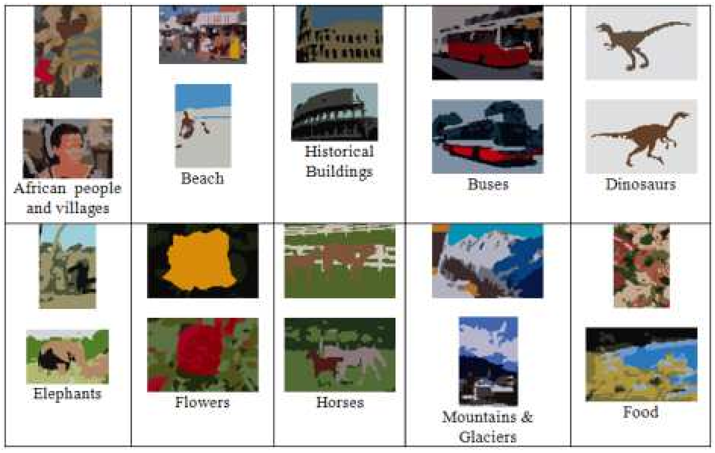


Figure 2.4 Images in Corel's data sets after preprocessing.

The training and testing data are created as follows: Choose randomly 5 categories and then select randomly 50 images from each category this constitutes the training data set, namely, total of 250 bags for training and the remaining 250 bags for testing. Next, we take 50 images of one category and mark it Positive, while the rest (200) is marked Negative. This is done for both training and testing. Table 2.5 summarizes performance of 10 algorithms for one case in which Africa, Beach, Building, Bus, and Dinosaur, were chosen.

Because of the high unbalance between the positive and negative bags we assess the performance of the algorithms in terms of both accuracy and MCC measures.

The performance of mi-DS and other algorithms in terms of accuracy on these data sets (text and image) are shown in Table 2.13. Text categorization experiments show that mi-DS performed on par with MDD, MI-SMO and MI-Optimal Ball algorithms. Average rank (last row in Table 2.13) shows that mi-DS performed better than the other 6 algorithms: Citation-KNN, MDD, MI-EMDD, MI-NND, MI-SMO and MI-SVM.

The performance of mi-DS and other algorithms in terms of the MCC (see details in Appendix A) are shown in Table 2.14.

Text categorization experiments show that mi-DS performed best, while on image data sets it achieved only average rank. The average rank (last row in Table 2.13) shows that mi-DS performed better than the same 6 algorithms as when using accuracy (Citation-KNN, MDD, MI-EMDD, MI-NND, MI-SMO and MI-SVM).

Table 2.13 Comparison with 9 MIL (in term of accuracy) algorithms on Text and Corel Images data; the best results for each data set are shown in bold.

Data set	mi-DS	Citation-KNN	MI-DD	MDD	MI-EMDD	MI-NND	MI-SMO	MI-SVM	MI-LR	MI-Optimal Ball
Component Train: 359 Pos, 359 Neg bags Test: 64 Pos, 2348 Neg bags	84.1 [3]	63.8 [9]	74.8 [7]	**91 [1]**	61.4 [10]	77.5 [6]	82.3 [5]	67.1 [8]	86.2 [2]	84.1 [3]
Function Train: 385 Pos, 385 Neg bags Test: 58 Pos, 4414 Neg bags	**97.1 [1]**	71.9 [8]	92.2 [5]	95.6 [3]	78.1 [7]	93.8 [4]	91.4 [6]	62.5 [10]	68.2 [9]	**97.1 [1]**
Process Train: 620 Pos, 620 Neg bags Test: 137 Pos, 10341 Neg bags	87.3 [5]	80.1 [9]	91.4 [3]	**96.8 [1]**	86.2 [7]	85.7 [8]	91.3 [4]	78.6 [10]	93.8 [2]	87.3 [5]
Africa Test and Train has 50 Positive bags, 200 Negative bags	87.6 [5]	85.2 [6]	**92.8 [1]**	84 [7]	90.8 [2]	78.4 [10]	83.6 [8]	80* [9]	90.4 [3]	88.4 [4]
Beach Test and Train has 50 Positive bags, 200 Negative bags	82.8 [7]	88 [2]	86 [4]	84 [6]	**88.8 [1]**	32.8 [10]	80* [8]	80* [8]	86 [4]	88 [2]
Building Test and Train has 50 Positive bags, 200 Negative bags	84 [5]	85.6 [3]	**91.2 [1]**	84.4 [4]	87.6 [2]	43.6 [10]	80* [8]	80* [8]	82 [7]	82.4 [6]
Bus Test and Train has 50 Positive bags, 200 Negative bags	92.4 [3]	91.2 [5]	93.2 [2]	80.8 [9]	89.2 [7]	89.2 [7]	**94.4 [1]**	80* [10]	92.4 [3]	90 [6]
Dinosaur Test and Train has 50 Positive bags, 200 Negative bags	98 [6]	99.6 [3]	100 [1]	80* [8]	**100 [1]**	58.8 [10]	88.4 [7]	80* [8]	99.6 [3]	99.2 [5]
Average Rank	4.4	5.6	3.0	4.9	4.6	8.1	5.9	8.9	4.1	4.0

* Algorithm assigns all bags into one class only.

Table 2.14 Comparison with 9 MIL algorithms (in terms of MCC) on Text and Corel Images data; the best results for each data set are shown in bold.

Data set	mi-DS	Citation-KNN	MI-DD	MDD	MI-EMDD	MI-NND	MI-SMO	MI-SVM	MI-LR	MI-Optimal Ball
Component	**0.3 [1]**	0.136 [6]	0.131 [7]	0.09 [10]	0.118 [8]	0.11 [9]	0.15 [4]	0.182 [2]	0.166 [3]	0.138 [5]
Function	**0.608 [1]**	0.108 [5]	0.165 [2]	0.08 [9]	0.119 [4]	0.09 [6]	0.127 [3]	0.08 [9]	0.09 [6]	0.09 [6]
Process	**0.517 [1]**	0.129 [7]	0.179 [4]	0.11 [10]	0.125 [8]	0.122 [9]	0.177 [5]	0.185 [2]	0.167 [6]	0.185 [2]
Africa	0.127 [9]	0.573 [6]	**0.764 [1]**	0.4 [7]	0.693 [2]	0.273 [8]	0.648 [4]	0 [10]	0.677 [3]	0.601 [5]
Beach	0.037 [8]	0.535 [3]	0.497 [5]	0.402 [6]	**0.615 [1]**	-0.071 [10]	0.064 [7]	0 [9]	0.51 [4]	0.585 [2]
Building	0.067 [8]	0.502 [3]	**0.706 [1]**	0.421 [4]	0.567 [2]	-0.197 [10]	0.387 [5]	0 [9]	0.375 [6]	0.361 [7]
Bus	**0.987 [1]**	0.652 [6]	0.778 [3]	0.18 [10]	0.631 [7]	0.437 [8]	0.872 [2]	0.22 [9]	0.75 [4]	0.665 [5]
Dinosaur	0.953 [5]	0.951 [6]	**1 [1]**	0 [10]	**1 [1]**	0.164 [9]	0.735 [8]	0.78 [7]	0.988 [3]	0.975 [4]
Average Rank	4.3	5.3	3	8.3	4.1	8.6	4.8	7.1	4.4	4.5

The Wilcoxon and Friedman tests are performed using Table 2.13 (accuracies) and Table 2.14 (MCC). The results are shown in Table 2.15.

Table 2.15 Wilcoxon and Friedman test results for comparing 9 MIL algorithms on text and Corel images data.

	Accuracy $\alpha = 0.05$, $N = 8$	**MCC** $\alpha = 0.05$, $N = 8$
Wilcoxon Test (See Appendix A for details)	The null-hypothesis that a given pair of algorithms performs equally well is rejected when ***T* ≤4 (in 2 cases).** In other words, mi-DS performs better than 2 algorithms.	The null-hypothesis that a given pair of algorithms performs equally well is rejected when ***T* ≤4 (in 2 cases).** In other words, mi-DS performs better than 2 algorithms.

Accuracy:

Pair of compared algorithms	R+	R–	T=min(R+, R-)
mi-DS vs. Citation-KNN	26	10	10
mi-DS vs. MI-DD	13	23	13
mi-DS vs. MDD	22	14	14
mi-DS vs. MI-EMDD	19.5	16.5	16.5
mi-DS vs. MI-NND	36	0	**0**
mi-DS vs. MI-SMO	29	7	7
mi-DS vs. MI-SVM	36	0	**0**
mi-DS vs. MI-LR	11.5	24.5	11.5
mi-DS vs. MI-Optimal Ball	16	20	16

MCC:

Pair of compared algorithms	R+	R–	T=min(R+, R-)
mi-DS vs. Citation-KNN	18	18	18
mi-DS vs. MI-DD	14	22	14
mi-DS vs. MDD	27	9	9
mi-DS vs. MI-EMDD	14	22	14
mi-DS vs. MI-NND	34	2	**2**
mi-DS vs. MI-SMO	22	14	14
mi-DS vs. MI-SVM	36	0	**0**
mi-DS vs. MI-LR	17	19	17
mi-DS vs. MI-Optimal Ball	18	18	18

	Accuracy	**MCC**
Friedman test (See Appendix A for details)	The null hypothesis that all algorithms perform at the same level is rejected because the calculated *Fr* value (26.60) is bigger than the critical value (16.92). In other words, mi-DS is performing significantly different than other algorithms.	The null hypothesis that all algorithms perform at the same level is rejected because the calculated *Fr* value (27.95) is bigger than the critical value (16.92). In other words, mi-DS is performing significantly different than other algorithms.

From Table 2.15 we conclude that in terms of Wilcoxon test, using accuracy as the performance measure, mi-DS is significantly better than MI-NND and MI-SVM. It performs on par with the Citation-KNN, MI-DD, MDD, MI-EMDD, MI-SMO, MI-LR and MI-OptimalBall algorithms.

When using MCC as the performance measure, Wilcoxon test tells us that mi-DS performs better than MI-NND and MI-SVM. The mi-DS performs on par with other 7 algorithms.

While the Wicloxon and Friendman tests perform different comparisons, the same conclusion can be drawn, mi-DS performs better than or on par with the other MIL algorithms.

An important feature of the mi-DS algorithm, in contrast to other MIL algorithms, is that it works well on data with missing values, which are treated as “do not care” values. To test this scenario we introduced missing values into Musk 1, Atoms, Bonds, Elephants, Fox and Artificial 1 data sets. It was done by randomly deleting 5%, 10%, 15%, 20%, 25% and 30% of the feature values. mi-DS was then run using 10FCV and the results are shown graphically in Figure 2.5. They indicate that mi-DS performs quite well up to 10% of missing values. This result could not be compared with other MIL algorithms because no published results were available at the time of this writing.

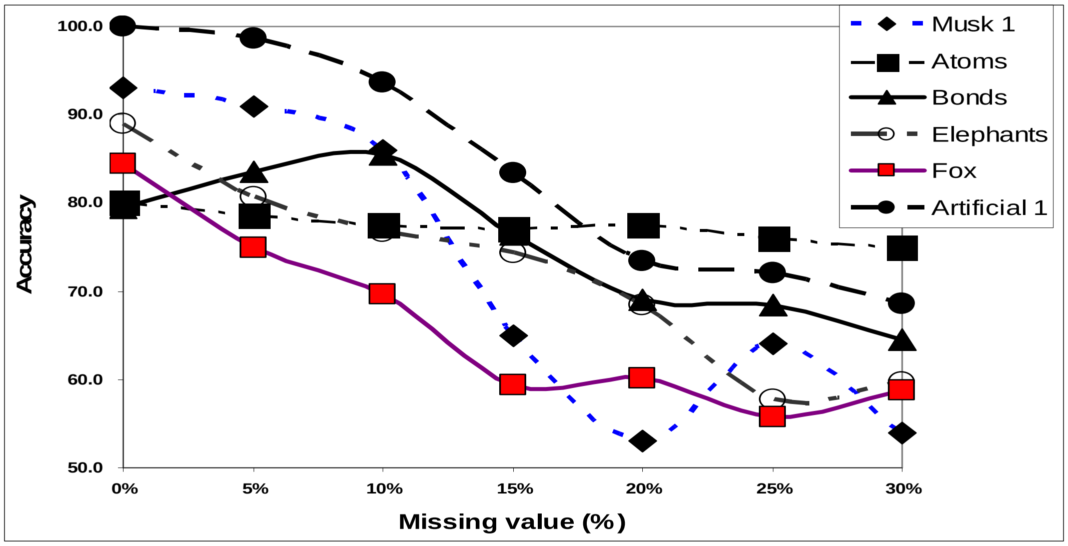


Figure 2.5 Accuracy of mi-DS on data with missing values.

2.4 Summary

A new rule-based MIL algorithm is created, called mi-DS, and compared with 9 state-of-the-art MIL algorithms on 26 diverse data sets that ranged from numerical data, to text, to image data. The results indicated that although there was no single generally best-performing algorithm, however on all data sets the mi-DS performed very well and had very desirable characteristics that distinguished it from the rest.

First, on average, it showed very good predictive accuracy on most data sets, as measured by both accuracy and MCC criteria. In particular, it exhibited good performance on challenging image and textual data. Second, the differences in performance between mi-DS and the other algorithms were statistically significant for the six of them . Third, mi-DS also performed quite well on data with missing values.

Important to note is that the approach taken in the mi-DS algorithm can be used as a generic framework for converting other rule-based algorithms so they can be used to solve MIL problems. This can be done in phase I of mi-DS, as the rule generation process can be done by any rule learner, while the construction of the similarity matrix in phase II, and the prediction procedure used in phase III would remain the same.

CHAPTER 3 One-class Learning Algorithm: OneClass-DS

3.1 State of the Art

In classical learning problems, training data are available for all classes. In such cases the learning algorithm can use all this information to discriminate between classes. However, there are many problems where only a single class of instances is known at the training time. At prediction time, new instances, with unknown class labels, can either belong to the target class (learned during training) or to some other class that was not seen during training. In this scenario, two different predictions are possible: Target, meaning an instance belongs to the class learned during training or Unknown, where the instance does not belong to the previously learned Target class. This type of a learning problem is known as one-class classification. One-class classification problem is also known as an outlier/novelty detection problem because the learning algorithm differentiates between target data (normal) and the rest (abnormal) using information about distribution of training data.

There is a wide variety of application domains for one-class algorithms such as a strange traffic patterns in a computer network (that can be caused by a hacked computer which sends sensitive data to an unauthorized destination), abnormal patterns in patient medical records (that could be symptoms of a new disease), outliers in credit card transaction data (that could indicate credit card theft), an unusual change in satellite images of the enemy area (that could indicate enemy troop movements), and many more.

In many other cases it is advantageous to convert a binary classify problem to one-class problem to obtain better results. For example, Nosocomial infections (Nis) is one of the major causes of increased mortality among hospitalized patients. The goal is to identify patients with one or more Nis on the basis of clinical data and data collected during a survey. It is a two-class classification problem (one has Nis or not) but there is a significant imbalance in data stored in database, namely, between the positive (infected) cases (11%) and the negative (non-infected) cases (89%). A shown by Cohen et al (Cohen et al, 2004), solving this problem as a binary classification problem results in sensitivity of 50.6%. However, if this is converted into one-class classification problem, where “Target” means non-infected patients and solve it as one-class classification it increases the sensitivity to 92.6% (Cohen et al, 2004).

Several methods have been proposed to solve the one-class classification problem. The most popular approaches can be divided into four categories: the density estimation, boundary, reconstruction methods, and rule-based methods. For each of these approaches different algorithms can be constructed. Each of these one-class classification methods differs in its ability to cope with or exploit, different characteristics of the data. The most salient characteristics when considering these problems are the scaling of features in the data, grouping of objects into clusters, convexity of data distribution and their position based on a set of prototypes as defined with reconstruction error. An overview of the categories of classification methods is presented in the next sections.

3.1.1 Density estimation methods

The most straightforward method is to estimate density of the training data (Tarassenko et al., 1995) and then use some threshold to encompass the data. When data are sufficiently large and a density model, such as Parzen, is used this approach works quite well. The drawback, however, is that it requires a large number of instances to overcome the challenge of dimensionality (Duda and Hart, 1973). If the dimensionality of data and complexity of the density model are restricted, then a large bias may be introduced, resulting in a model that does not fit the data well. Finding the right model to describe the target data distribution is a typical bias-variance dilemma. Using the density approach one needs to assume a distribution type such as Gaussian (Bishop, 1995; Ullman, 1978), or mixture of Gaussians (Duda and Hart, 1973; Bishop, 1995), or Parzen (Parzen, 1962; Kraaijveld and Duin, 1991). Hempstalk et al. (Hempstalk et al., 2007) developed an algorithm which builds a density function from a chosen distribution and then combines this function with a class probability to form an adjusted estimate of the density function of the target class, which is then used for constructing a decision tree.

3.1.2 Boundary methods

In boundary methods a closed boundary around the target data set is defined first and then optimized. Those methods rely heavily on distances between instances and are sensitive to feature scaling (Vapnik, 1998). Although the volume of data is not always minimized, most methods have a strong bias towards a minimal-volume solution. The size of the volume is depends on the model. The advantage of boundary methods is that the number of instances required for training is smaller than the number required in density methods. The difficulty is shifted into defining appropriate distance measures. One such method is the k-center method that covers the data with k small balls with equal radii (Ypma and Duin, 1998). The ball centers, μ_k, are placed on training instances such that the maximum of all minimum distances between training instances and the centers is minimized. In fitting the model the following error is minimized:

$$\varepsilon_{k-centr} = \max_i \left(\min_k \|x_i - \mu_k\|^2 \right)$$

The method uses a greedy search strategy starting with random initialization. The radius is determined by the maximum distance to the instances that the corresponding ball captures. This method is sensitive to outliers, possibly present in the data, but works well when data have good (compact) clustering structure. The user, however, needs to specify a priori both the number of balls, k, and the maximum number of tries (the number of runs with random initialization), which are a weaknesses. Another method is the nearest-neighbor method, NN-d, which is designed from local density estimation by the nearest neighbor classifier (Duda and Hart, 1973). It avoids explicit density estimation and uses only distances to the first nearest neighbors. It is similar to the methods of (Knorr et al., 2000) and (Breunig et al., 2000) used for outlier detection in large databases. In the nearest neighbor density estimation a cell, often a hypersphere in a d dimensional space, is centered around the training instance z. The volume of this cell is grown until it captures k instances of training data. Support vector machine (SVM) with RBF kernel was also used for anomaly detection (Ratsch et al., 2002). For each new instance, the method determines if it falls within the learned region: if it does then the instance is considered as Target, otherwise as an outlier. A similar approach, called Robust SVM, was used for intrusion detection (Hu et al., 2002).

3.1.3 Reconstruction methods

Reconstruction methods use prior knowledge about the data and make assumptions about the data-generating process to build a classifier fitting the data. One typical assumption is that a compact representation of the target data can be obtained to decrease noise influence (Bishop, 1995; Carpenter et al., 1991). The reconstruction methods assume that outlier instances do not satisfy assumptions about the target class distribution. During testing, a new instance may have low or high noise component and thus a corresponding low or high reconstruction error calculated as a distance to the target data set. Users need to choose appropriate thresholds when using these methods. Other reconstruction methods require a priori and/or predetermined suitable parameter values in order to create a successful classifier. For example, in learning vector quantization (Carpenter et al., 1991) and k-means clustering (Bishop, 1995) it is the number of clusters. In self-organizing feature maps (Kohonen, 1995) it is the dimensionality of the manifold, the number of prototypes per manifold and the learning rate. In PCA (Bishop, 1995) it is the mean and basis vectors for each of the subspaces, and the noise variance outside the subspaces. In diabolo networks (Hertz et al., 1991; Baldi and Hornik, 1989) and auto-encoder networks

(Japkowicz et al., 1995) it is the number of layers and neurons, learning rates, and stopping criteria.

3.1.4 Rule-based methods

Rule based techniques generate rules that capture target behavior of a system (Skalak and Rissland, 1990; Salvador and Chan, 2005). Instances that are not covered by any of the rules are considered outliers. Different techniques generate rules in different ways. Classification techniques such as IREP and RIPPER (Cohen, 1995) learn rules from noisy data by accommodating outliers present in the training data. An outlier class is artificially generated so the classifier (RIPPER) can learn the boundaries between the two classes. (Fan et al., 2001) introduced a supervised outlier detection method to detect network intrusions. (John, 1995) adapted the C4.5 algorithm to detect outliers in the data. A similar approach was used by (Abe et al., 2006) where the authors utilized a Query by Bagging method.

Another approach is to use association rule mining to generate rules, which requires a user to provide a minimum support threshold (Barbará et al., 2001; Tandon and Chan, 2007). The advantage of this approach is that it utilizes the fact that outliers occur very rarely in the data, and are dealt with by careful choosing of the support threshold to ensure that outliers are not taken into account in the process of rule generation. To ensure that rules correspond to strong patterns only rules with low support value are pruned. An application of this technique for intrusion detection was used in the ADAM (Audit Data Analysis and Mining) system (Barbara et al., 2001), and by (Otey et al., 2003) for intrusion detection embedded on the Network Interface Card (NIC). LERAD (Mahoney and Chan, 2002) method generates association mining rules from data of the form P(notW | U), which is a conditional probability of one subset of attributes taking a particular set of values (denoted by notW) given that a disjoint subset of attributes takes on a particular set of values (denoted by U). In order to deal with high number of rules that can be generated, they used sampling and randomization techniques. A similar approach was used for credit card fraud detection by (Brause et al., 1999) and for fraud detection in spacecraft house-keeping data by (Yairi et al. 2001). (Zengyou et al., 2004) proposed an outlier detection algorithm for categorical data sets by making the observation that an outlying transaction occurs in fewer frequent itemsets as compared to a normal transaction. They also proposed a measure called Frequent Pattern Outlier Factor (FPOF) that ranked the transactions based on the number of frequent itemsets they occur in.

3.2 OneClass-DS algorithm

In a binary-classification problem there are positive and negative instances (representing two classes) available for training. When rules for the positive (or negative) class are generated, they are found based on comparisons of positive instances with the negative (or positive) instances. During testing an instance is assigned to either the positive or negative class. This simple scheme cannot be used for solving one class classification problem since instances representing only (one) Target class are available. Moreover, Target instances may contain noise so they can be true targets or false outliers

(FO), which is shown in the left rectangle in Figure 3.1. Similarly, the Outlier instances (known only during testing) can be true outliers or false targets, shown in the right rectangle in Figure 3.1.

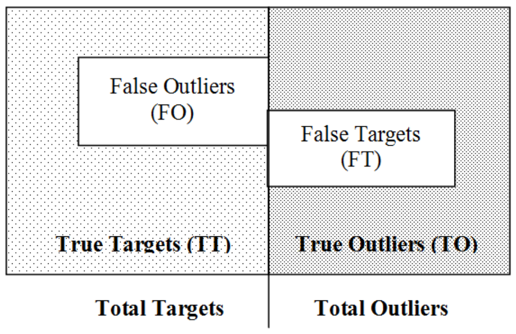


Figure 3.1 Training and test scenarios

It is required by many inductive learning algorithms, such as decision trees, that the data are discretized before they are used. OneClass-DS algorithm also requires discretization as a pre-processing step.

The OneClass-DS algorithm generates rules by performing greedy hill-climbing search in a manner similar to the DataSqueezer algorithm (Kurgan et al., 2004), thus the name OneClass-DS. However, DataSqueezer works only on binary (and multi-class) classification problems and generates rules by comparing the positive (one class) instances against the negative (other class) instances. Thus, to design an algorithm able to generate rules from one-class data only, the OneClass-DS uses four user-specified heuristic parameters to guide the rule generation process: *Threshold*, *MinCoverTager*, *MinAttribute* and *MaxAttribute*. The generated rules are accepted only if the total number of selectors in a rule is in the range between *MinAttribute* and *MaxAttibute*. The rules are in the format IF (Feature x = value a) and (Feature y = value b)… THEN Target instance. All selectors in a rule must satisfy two conditions: they must cover minimum number of target instances (*MinCoverTarget*) and have the summed-up value equal to or larger than the *Threshold*. The summed-value of a selector is calculated by multiplying the number of values a feature takes on by the number of times it appears in the entire target training data set. The pseudocode of the OneClass-DS algorithm is shown in Figure 3.2

```
OneClass-DS
Input: TargetTable, MinAttribute,MaxAttribute, MinCoverTarget, Threshold
Rules ← [ ]
While TargetTable ≠ empty
    Create list L of selectors, s, each selector satisfying two conditions:
                                        its summed-up-value ≥Threshold and
                                        its cover ≥ MinCoverTarget
    If (list L= empty) break;
    rule r ← []
    Select s from L which has the maximum summed-up-value and add it to rule r, and
    delete s from L
    While (total number of selectors in rule r < MinAttribute) and (list L not empty)
        Choose the next selector, s, and add it to rule r : r ← r U {s}
        Delete s from L
        If (total number of selectors in rule r = MaxAttribute) break
    End While
    Delete all instances in TargetTable covered by rule r
    Rules ← Rules U { r}
End While
Output: Rules (for Target class)
```

Figure 3.2 Pseudocode of *OneClass-DS* algorithm.

The process of generating rules is repeated several times with different settings of the four parameters. To choose the best model from the generated ones, the Target rate (sensitivity) measure is used:

$$\text{Target Rate} = \frac{\text{True Targets (TT)}}{\text{Total Targets (TT + FO)}} = \text{Sensitivity}$$

Target Rate is used in the following way. Although Target data is supposed to consist only of true targets, in reality it often contains false outliers, as shown in Figure 3.1. To identify the FO we use 10-FCV in the following way. The model is built from 9/10 of the target data but when testing it on the 1/10 of the data, some of its instances may not be recognized as (true) targets, and thus they are identified as false outliers, FO. At the end of the 10-FCV process all the instances that were recognized as FO are subtracted from the Target data. The model that has the largest Target Rate is chosen as the best one.

Choosing higher values of parameters results in creation of “strong rules” and such rules would predict new test target instances with higher accuracy. Conversely, using lower values of the parameters would result in generating “weaker” rules.

Computational complexity of OneClass-DS algorithm is approximately O(*RKNlogN*), where *R* is the number of rules in *Rules*, *N* is the number of instances, and *K* is the number of attributes.

Data shown in Table 3.1 are used to illustrate the working of the OneClass-DS algorithm.

Table 3.1 Training data instances.

F1	F2	F3	F4	Class
1	1	1	1	T
1	2	1	2	T
1	4	2	2	T
1	4	4	1	T
4	5	5	4	T

Suppose we choose the following values of the parameters: *MinCoverTarget*=25%, *MaxAttribute*=2, *MinAttribute*=2, *Threshold*=0. The summed-up values (v_{ij}) are calculated in Table 3.2, where *i* refers to the value of the feature, *j* refers the feature number, and (v_{ij}) = (the number of times that value appears)*(the total number of different values) for each feature, are calculated as depicted in Table 3.2

Table 3.2 Summed-up values for features shown in Table 3.1.

Feature	Total number of values	Summed-up values
F1	2 values {1,4}	v_{11}=4x2, v_{41}=1x2
F2	4 values {1,2,4,5}	v_{12}=1x4, v_{22}=1x4,v_{42}=2x4,v_{52}=1x4
F3	4 values {1,2,4,5}	v_{13}=2x4, v_{23}=1x4,v_{43}=1x4,v_{53}=1x4
F4	3 values {1,2,4}	v_{14}=2x3, v_{24}=2x3, v_{44}=1x3

We notice that F1, F2, and F3 have the same maximal summed-up values, namely, $v_{11} = v_{42} = v_{13} = 8$.

We choose the first feature, F1, and add selector "F1=1" to start creating the first rule, which results in:

IF F1=1

Table 3.3 shows instances covered (all but one) by (F1=1). We continue calculating summed-up values of the remaining features (except for F1), as shown in Table 3.4, in order to choose the next selector for the first rule.

Table 3.3 Instances in one-class training data set covered by (F1 =1).

F1	F2	F3	F4	Class
1	1	1	1	T
1	2	1	2	T
1	4	2	2	T
1	4	4	1	T

Table 3.4 Summed-up values for features shown in Table 3.3.

Feature	Total number of values	Summed-up values
F2	3 values {1,2,4}	v_{12}=1x4 v_{22}=1x4,v_{42}=2x4
F3	2 values {1,2,4}	v_{13}=2x4, v_{23}=1x4,v_{43}=1x4
F4	2 values {1,2}	v_{14}=2x3, v_{24}=2x3

Now features F2 and F3 have the same max summed-up values, namely, $v_{42} = v_{13} = 8$ so we choose F2 with value 4 as the next selector and add it to the first rule:

IF F1=1 AND F2=4

After choosing the second selector the process stops because *MaxAttribute*=2. Thus, the final version of the first rule is:

IF F1=1 AND F2=4 THEN CLASS = T (it covers 40% of Target data)

To generate the remaining rules we delete all instances covered by the first rule; the reduced data set is shown in Table 3.5.

Table 3.5 Training data instances that remain after deleting the instances covered by the first rule.

F1	F2	F3	F4	Class
1	1	1	1	T
1	2	1	2	T
4	5	5	4	T

We again calculate the summed-up values (v_{ij}) for each feature, as shown in Table 3.6.

Table 3.6 Summed-up values for features shown in Table 3.5.

Feature	Total number of values	Summed-up values
F1	2 values {1,4}	v_{11}=2x2, v_{41}=1x2
F2	3 values {1,2,5}	v_{12}=1x3, v_{22}=1x3,v_{52}=1x3
F3	2 values {1,5}	v_{13}=2x2, v_{53}=1x2
F4	3 values {1,2,4}	v_{14}=1x3, v_{24}=1x3, v_{44}=1x3

We notice that F1 and F3 have the same maximum summed-up values, namely, $v_{11} = v_{13} = 4$. Thus, we choose the first feature, F1, and add selector "F1=1" to start creating the second rule:

IF F1 = 1

Table 3.7 shows instances covered (all but one) by (F1=1). We continue calculating summed-up values of the remaining features (except for F1), as shown in Table 3.8, in order to choose the next selector for the second rule.

Table 3.7 Instances in Table 3.5 covered by (F1 =1).

F1	F2	F3	F4	Class
1	1	1	1	T
1	2	1	2	T

Table 3.8 Summed-up values of features (for instances shown in Table 3.7).

Feature	Total number of values	Summed-up values
F2	2 values {1,2}	v_{12}=1x2, v_{22}=1x2
F3	1 values {1}	v_{13}=2x1
F4	2 values {1,2}	v_{14}=1x2, v_{24}=1x2

Features F2, F3 and F4 have the same max summed-up values, namely, $v_{12} = v_{22} = v_{13} = v_{14} = v_{24} = 2$. In this case, F3=1 is the best selector because this selector covers all instances in Table 3.7. Thus, we choose F3 with value 1 as the next selector and add it to the second rule:

IF F1=1 AND F3=1

After adding the second selector the process stops because *MaxAttribute*=2. Thus, the final version of second rule is:

IF F1=1 AND F3=1 THEN CLASS = T (it covers 40% of Target data)

After deleting all instances (in Table 3.5) covered by the second rule we are left with data shown in Table 3.9.

Table 3.9 Training data instances after deleting instances covered by the first two rules.

F1	F2	F3	F4	Class
4	5	5	4	T

Since there is only one remaining instance, and *MinCoverTarget*=25% of target instances, the OneClass-DS algorithm stops generating rules.

Thus, only these two rules were generated from the target training instances:

IF F 1 = 1 AND F2 = 4 THEN CLASS = T

(covers instances 3 and 4 , or 40% of target instances)

IF F1 = 1 AND F3 = 1 THEN CLASS = T

(covers instances 1 and 2 , or 40% of target instances)

Notice that by choosing the specified initial parameters it allowed for mot covering one instance.

3.3 Experiments

The OneClass-DS algorithm was implemented as a function for the WEKA software to allow for standard testing of all the experiments described below.

To compare OneClass-DS with other algorithms false alarm rate (FAR) and imposter pass rate (IPR) measures were used. They are defined (Hempstalk et al., 2007) as:

$$\text{FAR} = \frac{\text{False Outliers(FO)}}{\text{True Outliers(TO)} + \text{False Outliers(FO)}}$$

$$\text{IPR} = \frac{\text{False Targets (FT)}}{\text{True Targets(TT)} + \text{False Targets(FT)}}$$

The FAR specifies the number of true Target instances incorrectly identified as outliers (false negatives). The IPR specifies the number of outlier instances that are wrongly classified (false positives) as belonging to the Target class. Notice that a higher FAR corresponds to a lower IPR, and vice versa. Additionally common measures of Precision and Accuracy are calculated:

$$\text{Precision} = \frac{\text{True Targets (TT)}}{\text{True Targets(TT)} + \text{False Targets(FT)}}$$

$$\text{Accuracy} = \frac{\text{True Targets(TT)} + \text{True Outliers(TO)}}{\text{Total instances in testing data set (TT} + \text{FO} + \text{TO} + \text{FT)}}$$

as well as the Area Under Curve (AUC) of the receiver operating characteristics (ROC) graph, which is calculated by WEKA.

In all four types of experiments described below, the results are reported using a modified 10-fold cross-validation procedure. Namely, after the data are divided into ten parts, all instances not belonging to Target class are deleted from the nine parts used for training (representing the Target class). For example, if the data instances represent three categories (One, Two and Three), if the current Target is category One, all instances belonging to categories Two and Three are deleted from the 9 training sets. Next, models for categories Two and Three are generated, and finally the average of the three runs is reported as the final result.

The experiments described below consist of four parts. First, we run and compare OneClass-DS with the other five algorithms in WEKA environment. Second, OneClass-DS is compared with OneClass SVM, OCC-Gauss and OCC-EM algorithms but using only the published (Hemstalk et al., 2007) results for the latter three algorithms. Third, experiments using OneClass-DS, REP-Tree, Decision Stump and Random Tree algorithms on missing value data sets are performed. Fourth, OneClass-DS is used to solve a multi-class problem by converting them into one-class problems on three large data set.

3.3.1 Experiments I

In these experiments we use 11 data sets from the UCI repository (http://www.ics.uci.edu/~mlearn/MLRepository.html). Four data sets have nominal features. The number of instances ranges from 151 to 12,960, the number of features from 5 to 23, and the number of classes from 2 to 5. Details are shown in Table 3.10.

Table 3.10 Data sets used in experiments.

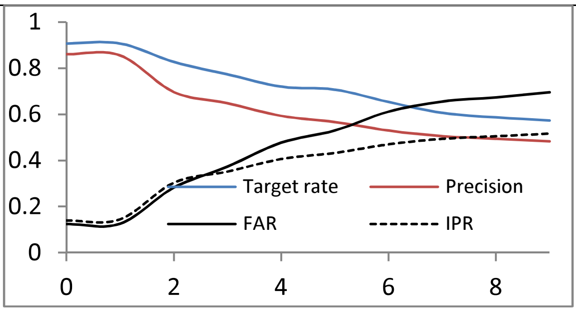
Name	Description	# of instances	# of features	# of classes
Car	Car Evaluation-Nominal	1728	7	4
Mrm	Mushroom -Nominal	8124	23	2
Nur	Nursery-Nominal	12960	9	5
Bre	Breast Cancer	699	10	2
Sca	Balance Scale Weight	625	5	3
Bup	BUPA livers disorder	345	7	2
Hea	Heart Data set	270	14	2
Pim	Pima Indians Diabetes Database	768	9	2
Spc	SPECT heart data-Nominal	267	23	2
Cmc	Contraceptive Method Choice	1473	10	3
Tae	Teaching Assistant Evaluation	151	6	3

Evaluating performance of a one-class classifier on a data set with N classes is performed by treating each class as the Target class, with all other classes treated as one “outlier” class. The results are shown in Table 3.11

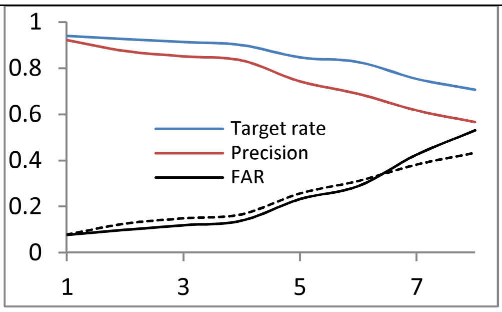
Table 3.11 Results of OneClass-DS algorithm on 11 data sets.

	Class name	# of targets	# of outliers	Time to build model (sec)	# of rules	Target rate	Accu-racy	Precision	FAR	IPR	AUC
Car	unacc	1210	518	0.02	11	0.97	0.91	0.91	0.09	0.09	0.876
	acc	384	1344	0.01	11	0.80	0.55	0.30	0.11	0.70	0.639
	vgood	65	1663	0.01	1	0.02	0.95	0.05	0.04	0.95	0.502
	good	69	1659	0.01	2	0.42	0.88	0.15	0.03	0.85	0.661
	Average Weight					**0.87**	**0.82**	**0.60**	**0.05**	**0.40**	**0.670**
Mrm	Class p	3916	4208	0.22	31	0.89	0.91	0.91	0.10	0.09	0.908
	Class e	4208	3916	0.34	32	0.96	0.95	0.95	0.05	0.05	0.954
	Average Weight					**0.93**	**0.93**	**0.93**	**0.07**	**0.07**	**0.931**
Nur	recom	2	12958	No test (too small)							
	prior	4266	8694	0.48	139	0.98	0.60	0.45	0.02	0.55	0.696
	not_recom	4320	8640	0.34	54	0.96	0.32	0.33	0.82	0.67	0.484
	very_recom	328	12632	0.12	27	0.89	0.80	0.10	0.00	0.90	0.841
	spec_prior	4044	8916	0.45	124	0.99	0.66	0.48	0.01	0.52	0.750
	Average Weight					**0.98**	**0.59**	**0.38**	**0.02**	**0.62**	**0.693**
Bre	Class 2	458	241	0.04	26	0.95	0.93	0.95	0.10	0.05	0.784
	Class 4	241	458	0.02	7	0.98	0.98	0.98	0.01	0.02	0.935
	Average Weight					**0.96**	**0.96**	**0.96**	**0.04**	**0.04**	**0.860**
Sca	Class B	49	576	0.1	13	0.67	0.66	0.14	0.04	0.86	0.666
	Class R	288	337	0.01	29	0.88	0.69	0.61	0.16	0.39	0.702
	Class L	288	337	0.01	28	0.91	0.69	0.61	0.14	0.39	0.708
	Average Weight					**0.88**	**0.68**	**0.51**	**0.10**	**0.49**	**0.692**
Bup	Class 1	145	200	0.03	1	0.94	0.46	0.43	0.28	0.57	0.526
	Class 2	200	145	0.05	1	0.95	0.59	0.59	0.40	0.41	0.527
	Average Weight					**0.94**	**0.53**	**0.52**	**0.33**	**0.48**	**0.527**
Hea	Class 1	150	120	0.05	4	0.71	0.69	0.73	0.35	0.27	0.691
	Class 2	120	150	0.05	4	0.67	0.68	0.63	0.28	0.37	0.680
	Average Weight					**0.69**	**0.69**	**0.69**	**0.31**	**0.31**	**0.686**
Pim	Class 0	500	268	0.4	7	0.99	0.66	0.66	0.30	0.34	0.499
	Class 1	268	500	0.39	7	0.97	0.35	0.35	0.54	0.65	0.508
	Average Weight					**0.99**	**0.50**	**0.50**	**0.43**	**0.50**	**0.504**
Spc	Class 0	55	212	0.1	2	0.87	0.45	0.26	0.09	0.74	0.609
	Class 1	212	55	0.1	4	0.96	0.76	0.79	1.00	0.21	0.481
	Average Weight					**0.94**	**0.61**	**0.57**	**0.17**	**0.43**	**0.545**
Cmc	Class 1	629	844	0.02	9	0.99	0.43	0.43	0.44	0.57	0.500
	Class 2	333	1140	0.01	10	0.99	0.25	0.23	0.10	0.77	0.510
	Class 3	511	962	0.02	8	0.99	0.37	0.35	0.13	0.65	0.513
	Average Weight					**0.99**	**0.35**	**0.34**	**0.15**	**0.66**	**0.508**
Tae	Class 1	49	102	0.17	7	0.94	0.42	0.35	0.15	0.65	0.553
	Class 2	50	101	0.01	7	0.96	0.43	0.36	0.11	0.64	0.564
	Class 3	52	99	0.01	7	0.96	0.42	0.37	0.13	0.63	0.546
	Average Weight					**0.95**	**0.42**	**0.36**	**0.13**	**0.64**	**0.554**

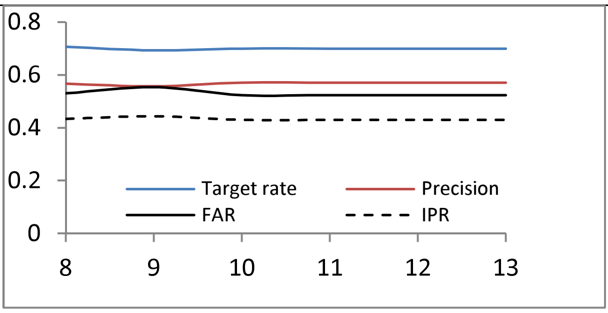
The four parameters in the OneClass-DS algorithm, namely, the *Threshold*, *MaxAttribute*, *MinAttribute*, and *MinCoverTarget* are varied to achieve the best target rate. Figure 3.3 shows the impact of varying one parameter while the other three parameters remain fixed; for the Heart data set. The purpose is to show that it is possible to obtain a better model by modifying these parameters during training.



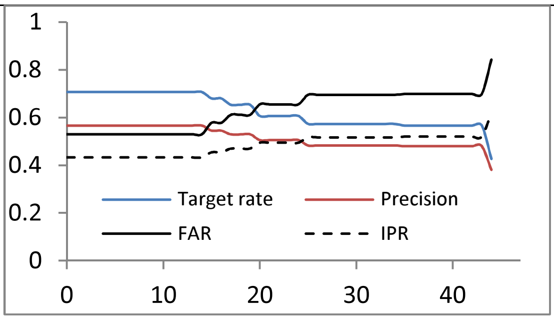
MinCoverTarget
With *Min Attribute*=8 *MaxAttribute*=8
Threshold=5



MinAttribute
With *MaxAttribute*=8 *MinCoverTarget*=5
Threshold=5



MaxAttribute
With *MinAttribute*=8 *MinCoverTarget*=5
Threshold=5



Threshold
With *MinAttribute*=8 *MaxAttribute* =8
MinCoverTarget=5

Figure 3.3 Influence of changing the four parameters values on the Target rate, Precision, IPR and FAR, on the Heart data set.

To compare OneClass-DS with other algorithms WEKA's OneClass Classifier package was used. It is based on the OCC algorithm of (Hemstalk et al., 2007) and it builds density function from a chosen distribution (can be Gauss or EM) and then combines it with a class a priori probability to form the adjusted estimate of the density function of the Target class. In their approach the model is learned from the Target data and the artificially created uniformly distributed data that constitutes the Outlier class. In this way they could use a two-class classifier for solving a one-class classification problem.

The comparison of results is shown in Tables 3.12 and 3.13, while Figure 3.4 illustrates the results in a graphical form.

Table 3.12 Comparison of OneClass-DS with five other algorithms in terms of accuracy.

Data Set	REP-Tree	J48	Random Tree	Decision Stump	Random Forest	OneClass-DS
Car	0.89	**0.92**	0.76	**0.92**	0.8	0.82
Mrm	**0.97**	0.89	**0.97**	0.75	0.9	0.93
Nur	0.61	0.6	**0.68**	0.67	0.66	0.59
Bre	0.5	0.5	0.63	0.89	0.8	**0.96**
Sca	0.47	0.45	0.42	0.63	0.45	**0.68**
Bup	0.51	0.51	0.5	0.52	0.51	**0.53**
Hea	0.61	0.54	0.65	0.63	0.68	**0.69**
Pim	0.52	0.51	0.52	0.51	**0.53**	0.5
Spc	**0.71**	0.7	0.65	**0.71**	**0.71**	0.61
Cmc	0.44	0.42	0.41	**0.47**	0.44	0.35
Tae	0.41	0.42	0.42	0.39	**0.44**	0.42

Table 3.13 Comparison of OneClass-DS with five other algorithms in terms of the IPR and FAR measures.

Data set	IPR					
	REP-Tree	J48	Random Tree	Decision Stump	Random Forest	OneClass-DS
Car	0.28	**0.20**	0.49	**0.20**	0.43	0.40
Mrm	0.02	0.04	0.01	0.30	**0.00**	0.07
Nur	0.62	0.62	0.62	**0.58**	**0.58**	0.62
Bre	0.50	0.50	0.42	0.10	0.25	**0.04**
Sca	0.62	0.63	0.64	0.53	0.63	**0.49**
Bup	0.49	0.49	0.50	0.49	0.50	**0.48**
Hea	0.41	0.47	0.39	0.41	0.37	**0.31**
Pim	0.49	0.50	0.49	0.49	**0.48**	0.50
Spc	**0.34**	0.36	0.40	**0.34**	0.35	0.43
Cmc	0.64	0.65	0.65	**0.63**	**0.63**	0.66
Tae	0.65	0.67	0.65	0.66	**0.64**	**0.64**
Data set	FAR					
	REP-Tree	J48	Random Tree	Decision Stump	Random Forest	OneClass - DS
Car	**0.03**	0.03	0.04	0.04	**0.03**	0.05
Mrm	**0.04**	0.17	**0.04**	0.14	0.17	0.07
Nur	0.05	0.06	**0.02**	0.05	0.04	**0.02**
Bre	0.53	0.50	0.10	0.11	0.12	**0.04**
Sca	0.16	0.18	0.18	0.13	0.16	**0.10**
Bup	0.44	0.45	0.49	0.41	0.47	**0.33**
Hea	0.33	0.46	0.21	0.23	**0.19**	0.31
Pim	0.40	0.45	0.42	0.46	**0.37**	0.43
Spc	0.18	**0.16**	0.20	0.19	**0.16**	0.17
Cmc	0.18	0.24	0.23	0.17	0.17	**0.15**
Tae	0.28	0.33	0.26	0.24	0.19	**0.13**

The graphs in Figure 3.4 show Precision and Target Rate (Sensitivity) versus *MaxAttribute*. In Figure 3.4 Precision (red line) and Target Rate (blue line) are shown for the OneClass-DS algorithm while varying values of the *MaxAttribute* (number of features to be used in a rule). The results of precision (red triangles) and true targets (blue triangles) for the best performing algorithm chosen among REP-Tree, J48, Random Tree, Decision Stump, or Random Forest algorithms are shown.

As can be noticed in Figure 3.4 in the case of the Cmc data (upper left graph), Random Forest gave the best result with precision of 0.27 (red triangle) and target rate of 0.91 (blue triangle). OneClass-DS achieved good balance between precision (0.24) and target rate (0.94) with the *MaxAttribute* fixed at 3. Note that *MaxAttribute* if set to 6 obtains a precision rate higher than Random Forest but lower target rate. Alternatively, *MaxAttribute* if set to 2 gives a better target rate but at the cost of lower precision.

Upper right graph (for Bre data set) shows that Decision Stump gave the best results with precision of 0.95 (red triangle) and target rate of 0.88 (blue triangle), for OneClass-DS model: we can choose many values of *MaxAttribute*, such as: *MaxAttribute*=7 (precision rate=0.86, target rate=0.89), *MaxAttribute*=4 (precision rate=0.77, target rate=0.97), *MaxAttribute*=5 (precision rate=0.82, target rate=0.93) , to balance target rate and precision rate. Similarly for Mrm data (lower left graph): choosing *MaxAttribute*=7, OneClass-DS model will give value of 0.89 in precision rate and 0.95 in target rate (comparing with the best of other 5 algorithms is Random Tree with 0.97 in precision rate and 0.95 in target rate).

In case of Spc data set (lower right graph), We can choose *MaxAttribute*=5 with precision rate=0.34, target rate=0.84 when comparing with REP Tree (precision=0.39, target rate=0.78)

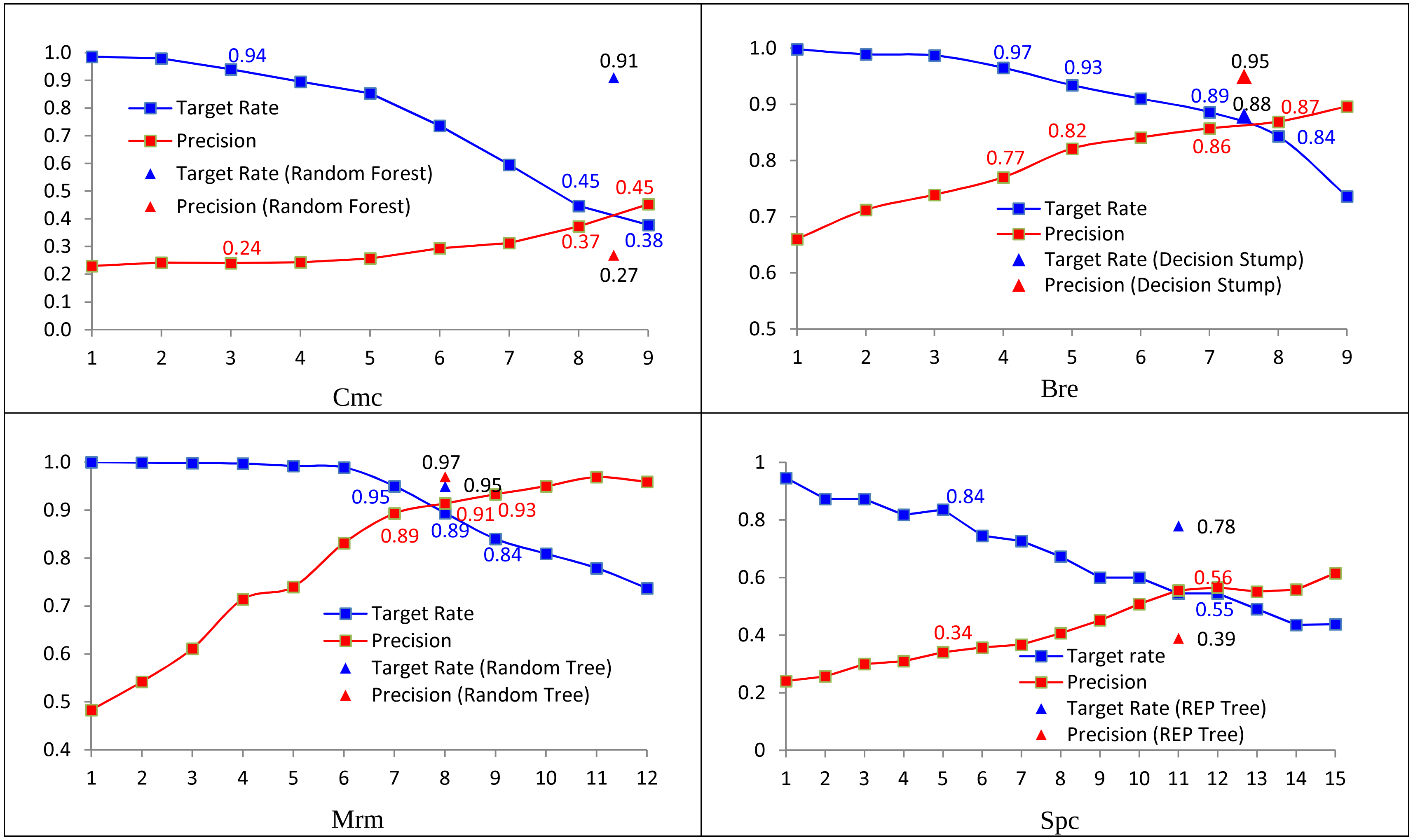


Figure 3.4 Possible outcomes of a test in terms of Precision &Target rate (y-axis) vs. *MaxAttribute* (x-axis).

The Wilcoxon and Friedman tests are performed on the results in terms of accuracies (Table 3.12) and IPR and FAR measures (Table 3.13). The results are shown in Table 3.14. From Table 3.14 we notice that in terms of Wilcoxon test, using accuracy and IPR, OneClass-DS performs on par with other algorithms. In terms of the FAR measure, however, OneClass-DS performs better than J48 and Decision Stump, and on par with the remaining algorithms.

Friedman tests show that OneClass-DS performs at the same level as the other algorithms in terms of all three measures: the accuracy, IPR, and FAR.

Table 3.14 Wilcoxon and Friedman tests results for comparing OneClass-DS with five algorithms on 11 data sets.

	Accuracy $\alpha = 0.05$, $N = 11$	**IPR** $\alpha = 0.05$, $N = 11$
Wilcoxon Test (See Appendix A for details)	The null-hypothesis that a given pair of algorithms performs equally well is rejected when $T \leq 11$ **(in 0 cases).** In other words, OneClass-DS performs equal with other algorithms.	The null-hypothesis that a given pair of algorithms performs equally well is rejected when $T \leq 11$ **(in 0 cases).** In other words, OneClass-DS performs equal with other algorithms.

Accuracy:

Pair of compared Algorithms	R+	R–	T=min(R+, R-)
OneClass-DS vs. REP-Tree	34	32	32
OneClass-DS vs. J48	26.5	39.5	26.5
OneClass-DS vs. Random Tree	30	36	30
OneClass-DS vs. Decision Stump	35.5	30.5	30.5
OneClass-DS vs. Random Forest	33	33	33

IPR:

Pair of compared algorithms	R+	R–	T=min(R+, R-)
OneClass-DS vs. REP-Tree	30.5	35.5	30.5
OneClass-DS vs. J48	27	39	27
OneClass-DS vs. Random Tree	19.5	46.5	19.5
OneClass-DS vs. Decision Stump	29	37	29
OneClass-DS vs. Random Forest	31	35	31

	Accuracy	**IPR**
Friedman test (See Appendix A for details)	The null hypothesis that all algorithms perform at the same level is accepted because the calculated *Fr* value (4.3) is smaller than the critical value (11.07). In other words, OneClass-DS is performing on par with other algorithms.	The null hypothesis that all algorithms perform at the same level is accepted because the calculated *Fr* value (8.25) is smaller than the critical value (11.07). In other words, OneClass -DS is performing on par with other algorithms.

	FAR $\alpha = 0.05$, $N = 11$
Wilcoxon Test	The null-hypothesis that a given pair of algorithms performs equally well is rejected when $T \leq 11$ **(in 2 cases).** In other words, OneClass -DS performs better than J48 and Decision Stump algorithms.
Friedman test	The null hypothesis that all algorithms perform at the same level is accepted because the calculated *Fr* value (10.58) is smaller than the critical value (11.07). In other words, OneClass -DS is performing on par with other algorithms.

FAR Wilcoxon Test:

Pair of compared Algorithms	R+	R–	T=min(R+, R-)
OneClass-DS vs. REP-Tree	13	53	13
OneClass-DS vs. J48	3	63	**3**
OneClass-DS vs. Random Tree	19.5	46.5	19.5
OneClass-DS vs. Decision Stump	11	55	**11**
OneClass-DS vs. Random Forest	20	46	20

3.3.2 Experiments II

We compared OneClass-DS with One-Class SVM and the OCC (One-Class Classification by Combining Density and Class Probability Estimation) with two different density functions, one based on Gaussian density (called OCC-Gauss) and other on EM density (OCC-EM). The results are shown in Table 3.15. Results of SVM and OCC algorithms are repeated based on (Hemstalk et al., 2007). In addition, we compared the time and average number of rules needed to construct a model using OneClass-DS, OneClass SVM, OCC-Gauss and OCC-EM algorithms. As expected, it shows that OneClass SVM was the fastest while OCC-EM was the slowest. The total number of rules generated by OneClass-DS was smaller than for OCC-Gauss and OCC-EM algorithms on all data sets but Ecoli and Glass.

The Wilcoxon and Friedman tests are performed using Table 3.15 (time to build model, IPR and FAR). The results are shown in Table 3.16.

Table 3.15 Comparison of OneClass-DS with OneClass SVM, OCC-Gauss and OCC-EM.

				FAR				**IPR**			
Data set	# of fea-tures	# of in-stances	# of classes	OneClass -DS	OneClass SVM	OCC-Gauss	OCC-EM	OneClass -DS	OneClass SVM	OCC-Gauss	OCC-EM
Diabetes	8	768	2	0.264	0.111	**0.098**	0.109	**0.482**	0.514	0.857	0.779
Ecoli	7	336	8	0.241	0.137	**0.129**	0.136	0.111	**0.068**	0.088	0.083
Glass	9	214	7	**0.064**	0.154	0.147	0.18	0.473	0.412	0.434	**0.331**
Heart-statlog	13	270	2	0.295	**0.122**	0.14	0.141	**0.46**	0.624	0.507	0.504
Ionosphere	34	351	2	0.211	**0.128**	0.15	0.169	**0.458**	0.738	0.732	0.697
Iris	4	150	3	**0.096**	0.12	0.125	0.137	0.13	**0.073**	0.076	0.077
Sonar	60	208	2	0.282	**0.12**	0.123	0.163	**0.455**	0.705	0.815	0.751
Vehicle	18	846	4	0.208	**0.103**	0.109	0.13	**0.342**	0.629	0.645	0.494
Waveform	21	5000	3	0.126	0.103	**0.075**	0.11	0.451	**0.307**	0.411	0.354

	Time it takes to build a model (sec)				**Average # of rules**			
	OneClass -DS	OneClass SVM	OCC-Gauss	OCC-EM	OneClass -DS	OneClass SVM	OCC-Gauss	OCC-EM
Diabetes	0.52	**0.37**	3.89	13.9	11	N/A	14	27
Ecoli	0.82	**0.13**	1.18	0.62	8	N/A	3	5
Glass	0.38	**0.09**	0.25	1.41	7	N/A	5	5
Heart-statlog	0.1	**0.06**	0.58	5.23	5	N/A	4	8
Ionosphere	9.9	**0.08**	1.65	30.64	5	N/A	3	5
Iris	0.1	**0.02**	0.13	0.26	4	N/A	7	14
Sonar	10.3	**0.13**	5.71	21.44	7	N/A	11	19
Vehicle	1.5	**0.68**	0.83	13.4	20	N/A	25	21
Waveform	77	**4.07**	47.5	106.17	12	N/A	27	71

Table 3.16 Wilcoxon and Friedman tests results for comparing OneClass-DS with three algorithms on 9 data sets.

	Time to build model $\alpha = 0.05$, $N = 9$	**IPR** $\alpha = 0.05$, $N = 9$
Wilcoxon Test (See Appendix A for details)	The null-hypothesis that a given pair of algorithms performs equally well is rejected when $T \leq 6$ **(in 2 cases).** In other words, OneClass-DS performs equal only with OCC-Gauss algorithm.	The null-hypothesis that a given pair of algorithms performs equally well is rejected when $T \leq 6$ **(in 0 cases).** In other words, OneClass -DS performs equal with other algorithms.
Friedman test (See Appendix A for details)	The null hypothesis that all algorithms perform at the same level is rejected because the calculated *Fr* value (21.13) is larger than the critical value (12.59).	The null hypothesis that all algorithms perform at the same level is accepted because the calculated *Fr* value (9.93) is smaller than the critical value (12.59). In other words, OneClass -DS is performing on par with other algorithms.

Time to build model:

Pair of compared Algorithms	R+	R–	T=min(R+, R-)
OneClass-DS vs. OneClass-SVM	45	0	**0**
OneClass-DS vs. OCC-Gauss	31	14	14
OneClass-DS vs. OCC-EM	2	43	**2**

IPR:

Pair of compared algorithms	R+	R–	T=min (R+, R-)
OneClass-DS vs. OneClass-SVM	14	31	14
OneClass-DS vs. OCC-Gauss	11	34	11
OneClass-DS vs. OCC-EM	13	32	13

	FAR $\alpha = 0.05$, $N = 9$
Wilcoxon Test	The null-hypothesis that a given pair of algorithms performs equally well is rejected when $T \leq 6$ **(in 1 cases).** In other words, OneClass -DS performs equal with OneClass-SVM and OCC-EM algorithms.
Friedman test	The null hypothesis that all algorithms perform at the same level is accepted because the calculated *Fr* value (3.8) is smaller than the critical value (12.59). In other words, OneClass -DS is performing on par with other algorithms.

Pair of compared Algorithms	R+	R–	T=min(R+, R-)
OneClass-DS vs. OneClass-SVM	39	7	7
OneClass-DS vs. OCC-Gauss	40	5	**5**
OneClass-DS vs. OCC-EM	37	8	8

From Table 3.16 we notice that in terms of Wilcoxon test, using the IPR measure, OneClass-DS performs on par with other algorithms. Using the FAR measure, however, OneClass-DS performs on par with the OneClass SVM and OCC-EM.

Wilcoxon test shows that when time to build is used as input the OneClass-DS performs on par only with OCC-Gauss.

Friedman tests show that OneClass-DS performs on par with the other three algorithms, both when in terms of IPR and FAR measures, but as well in terms of time to build model measure.

3.3.3 Experiments III

OneClass-DS was next used on four data sets, from the UCI data repository, with missing values. The data are described in Table 3.17. The number of instances ranges from 57 to 32,561 and number of features from 6 to 17. Note that these data sets are very different in terms of the number of missing values. The OneClass-DS results are shown in Table 3.18.

Table 3.17 Data sets with missing values.

Data set	Description	# of instances	# of missing values	# of features	# of classes
Vot	US Congressional Voting	435	104	17	2
Adt	Data set Adult From UCI	32561	1843	15	2
Lab	Labor negotiations in Canada	57	48	17	2
Mam	Mammographic Mass	961	76	6	2

Table 3.18 OneClass-DS results on data with missing values.

Data set	Classes	# of targets	# of outliers	Time to build model (sec)	# of rules	Target rate	Accu-racy	Preci-sion	FAR	IPR	AUC
Vot	Democrat	267	168	0.02	39	0.80	0.78	0.84	0.30	0.16	0.78
	Republican	168	267	0.01	20	0.85	0.92	0.95	0.09	0.05	0.91
	Average			0.02	30	0.82	0.85	0.89	0.19	0.11	0.84
Adt	Less50K	24720	7841	1253	158	0.83	0.68	0.77	0.72	0.23	0.52
	Over50K	7841	24720	1225	61	0.72	0.72	0.45	0.11	0.55	0.72
	Average			1239	110	0.77	0.70	0.61	0.41	0.39	0.62
Lab	Bad	20	37	0.01	6	0.70	0.74	0.61	0.18	0.39	0.73
	Good	37	20	0.01	2	0.54	0.70	1.00	0.46	0.01	0.77
	Average			0.01	4	0.62	0.72	0.80	0.32	0.20	0.75
Mam	0	516	445	0.03	4	0.59	0.72	0.83	0.35	0.17	0.73
	1	445	516	0.02	4	0.49	0.74	0.90	0.31	0.10	0.72
	Average			0.03	4	0.54	0.73	0.86	0.33	0.14	0.73

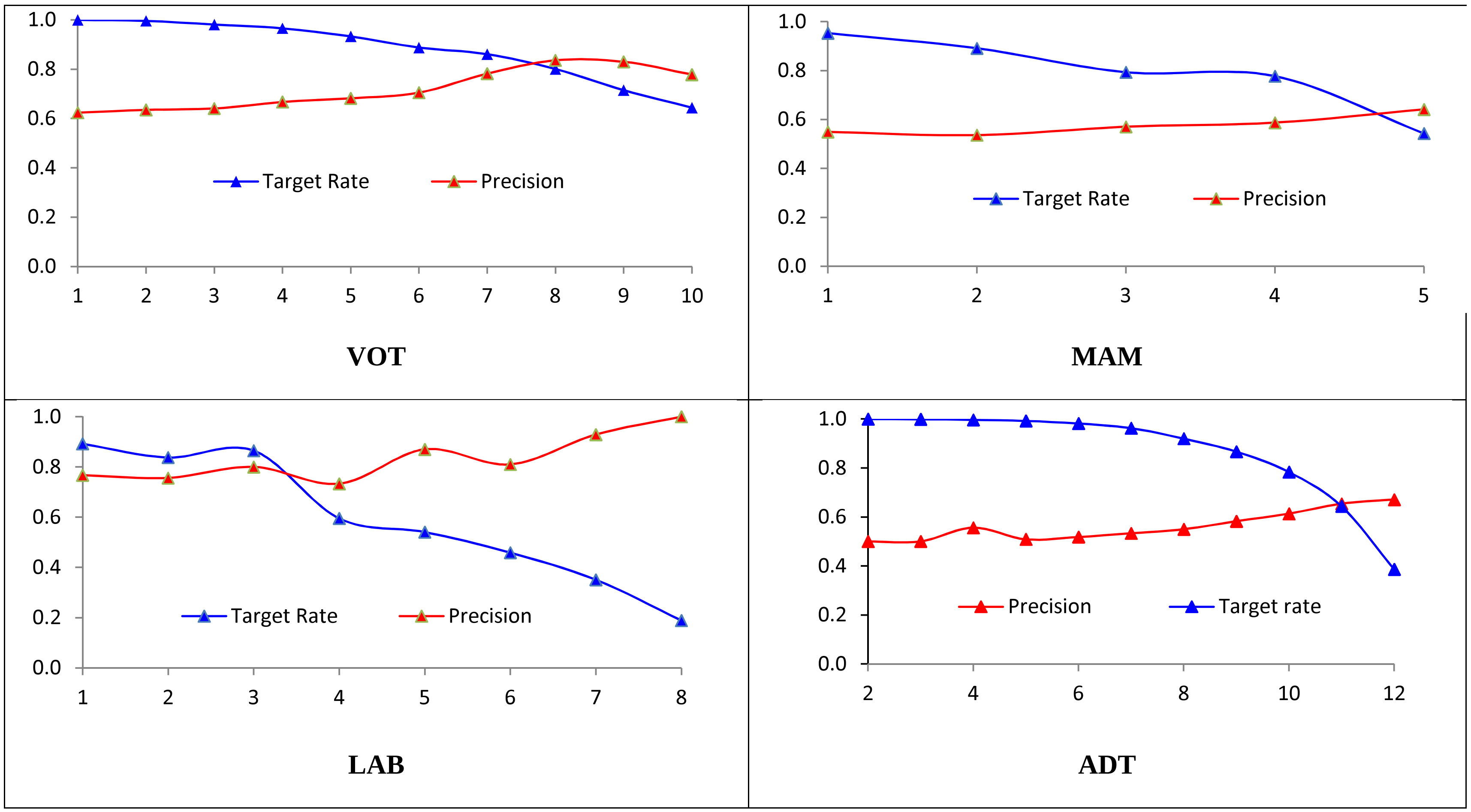


Figure 3.5 Possible outcomes of a test in terms of Precision&Target rate (y-axis) vs. *MaxAttribute* (x-axis).

Figure 3.5 shows the results in terms of precision and target rate versus *MaxAttribute* (in the same way as presented in Figure 3.4). To choose the best model for each data set, we ran 10-FCV with different parameter values.

In Figure 3.5, Upper left graph (using the VOT data set) shows the impact on the target rate and precision rate while varying value of the MaxAttribute parameter and keeping the remaining three parameters fixed. This graph shows the acceptable model at MaxAttribute= 8 because at this point the Target rate curve and Precision curve intersect.

Similarly for MAM data (upper right graph) choosing MaxAtrribute=5 results in a good model; for LAB data (lower left graph) with MaxAttribute=3, and for ADT data (lower right graph) MaxAttribute=11 seem to be good choices.

OneClass-DS is also compared with three other algorithms (REP-Tree, Decision Stump and Random Tree) when running it on four missing value data sets. The results of are shown in Table 3.19.

Table 3.19 Comparison of OneClass-DS with REP-Tree, Decision Stump and Random Tree on four missing value data sets.

	FAR				IPR				Accuracy			
Data set	OneClass -DS	REP-Tree	Decision Stump	Random Tree	OneClass -DS	REP-Tree	Decision Stump	Random Tree	OneClass -DS	REP-Tree	Decision Stump	Random Tree
Vote	0.19	**0.17**	0.21	0.54	0.11	0.26	**0.10**	0.38	**0.85**	0.77	**0.85**	0.55
Adt	**0.41**	0.77	0.70	0.75	0.39	0.23	**0.19**	0.24	**0.70**	0.50	0.58	0.51
Lab	**0.32**	0.36	0.34	0.33	**0.20**	0.35	0.43	0.47	**0.72**	0.64	0.64	0.63
Mam	**0.33**	0.54	0.38	0.53	**0.14**	0.45	0.47	0.45	**0.73**	0.50	0.57	0.51

Because of the small number of missing value data sets only the Friedman test ($\alpha = 0.05$, $N = 4$) is used and the result is that the null hypothesis: that all algorithms perform at the same level, both in terms of the FAR and accuracy measures, is rejected. The calculated *Fr* values (5.7 for FAR and 4.5 for accuracy) are larger than the critical value of 2.569. However, notice that OneClass-DS gives better results than the three in terms of accuracy and FAR measures.

In terms of the IPR measure all algorithms perform at the same level (the null hypothesis is accepted because the calculated Fr value (0.15) is smaller than the critical value (2.569)). In other words, OneClass -DS performs on par with the three algorithms.

3.3.4 Experiments IV

Three data sets consisting of letters and digits (from the UCI data repository) were used (Table 3.20). The Letters data set consists of 26 capital letters specified as black-and-white rectangular images. The images are based on 20 different fonts, and each letter for each font was randomly distorted to produce a file of 20,000 unique instances. Then, each instance was converted into 16 numerical attributes (such as statistical moments and edge counts) and scaled to fit into a range of integers from 0 to 15. The Optical digits data set was created by the NIST programs to extract normalized bitmaps of handwritten digits. 32x32 bitmaps are divided into non-overlapping blocks of the size 4x4, and number of pixels is counted in each block to generate an input matrix of 8x8, where each element is represented as integer in the range from 0 to 15, in order to reduce dimensionality of the data and to provide invariance to small distortions. The Pen digits data set was built by collecting instances from 44 writers using a pressure sensitive tablet for coding digits, while people wrote 250 digits at random inside boxes of 500 by 500 pixels.

In these experiments OneClass-DS is used for solving multi-class highly unbalanced data sets. To address the problem of dealing with highly unbalanced data sets they are transform into one-class problems in the following way. Each class is treated in turn not as a Target class but as an Outlier class because it has very small number of instances, while the rest of the classes are treated as the Target class. The rules are thus generated for the Outlier classes. Thus, during testing if the rule does not cover an instance than the instance is recognized as belonging to the Target class.

Table 3.20 Letter and digits data sets.

Data set	# of instances	# of features	# of classes
Pen digits	10,992	16	10
Optical digits	5,620	64	10
Letter	20,000	16	26

The CAIM algorithm is used (Kurgan et al., 2004) to discretize continuous features before running the OneClass-DS algorithm. Tables 3.20 and 3.21 show the results of the OneClass-DS algorithm.

Table 3.21 Results on the Pen digits data set.

Class name	# of targets	# of outliers	# of rules	Time to Build Model (sec)	Target rate	Accu-racy	Preci-sion	FAR	IPR	AUC
0	1143	9849	36	3.8	0.70	0.96	0.89	0.03	0.11	0.85
1	1143	9849	40	3.7	0.60	0.93	0.70	0.05	0.30	0.78
2	1144	9848	38	4.4	0.69	0.95	0.82	0.04	0.18	0.83
3	1055	9937	28	3.4	0.76	0.96	0.86	0.03	0.14	0.87
4	1144	9848	50	4.1	0.67	0.95	0.78	0.04	0.22	0.83
5	1055	9937	43	3.6	0.58	0.93	0.64	0.04	0.36	0.77
6	1056	9936	36	4.0	0.71	0.96	0.83	0.03	0.17	0.85
7	1142	9850	35	5.3	0.72	0.96	0.89	0.03	0.11	0.86
8	1055	9937	42	4.4	0.41	0.93	0.72	0.06	0.28	0.70
9	1055	9937	44	4.0	0.58	0.91	0.53	0.05	0.47	0.76
Average			39	4.1	0.64	0.94	0.76	0.04	0.24	0.81

Result in Table 3.21 show that when converting the original 10-class problem into 10 one-class problems, OneClass-DS gives acceptable results in terms of accuracy and FAR measures for all 10 classes. This is not the case, however, in terms of the target rate, precision and IPR measures.

In Table 3.22 OneClass-DS results are not very good, and much worse than on the Pen digits data set. Partial explanation is that the Optical data set has structure much more complex than Pen data set (both smaller number of instances and much higher dimensionality).

Table 3.22 Results on the Optical digits data set.

Class name	# of targets	# of outliers	# of rules	Time To Build Model (sec)	Target rate	Accu-racy	Preci-sion	FAR	IPR	AUC
0	554	5066	23	1.1	0.64	0.96	0.90	0.04	0.10	0.82
1	571	5049	15	1.0	0.71	0.94	0.68	0.03	0.32	0.84
2	557	5063	23	1.6	0.65	0.92	0.60	0.04	0.40	0.80
3	572	5048	22	1.1	0.45	0.89	0.45	0.06	0.55	0.78
4	568	5052	21	1.0	0.70	0.90	0.52	0.03	0.48	0.82
5	558	5062	23	1.1	0.60	0.94	0.73	0.04	0.27	0.79
6	558	5062	31	1.6	0.54	0.94	0.77	0.05	0.23	0.76
7	566	5054	22	1.0	0.65	0.93	0.63	0.04	0.37	0.81
8	554	5066	20	1.0	0.63	0.89	0.47	0.04	0.53	0.78
9	562	5058	18	2.5	0.64	0.91	0.53	0.04	0.47	0.79
Average			22	1.3	0.62	0.92	0.63	0.04	0.37	0.80

The prediction results shown in Tables 3.20 and 3.21 were shown after optimizing the model in a way that is illustrated below. The following three graphs are provided to illustrate the process of fine-tuning models during training for letter A only even though there are 26 classes (letters A through Z). The relation between Precision and Target rates are shown in Figures 3.6, 3.7 and 3.8. Depending on *MinCoverTarget* value in Figure 3.6, the precision increases from over 0.2 to about 0.6.

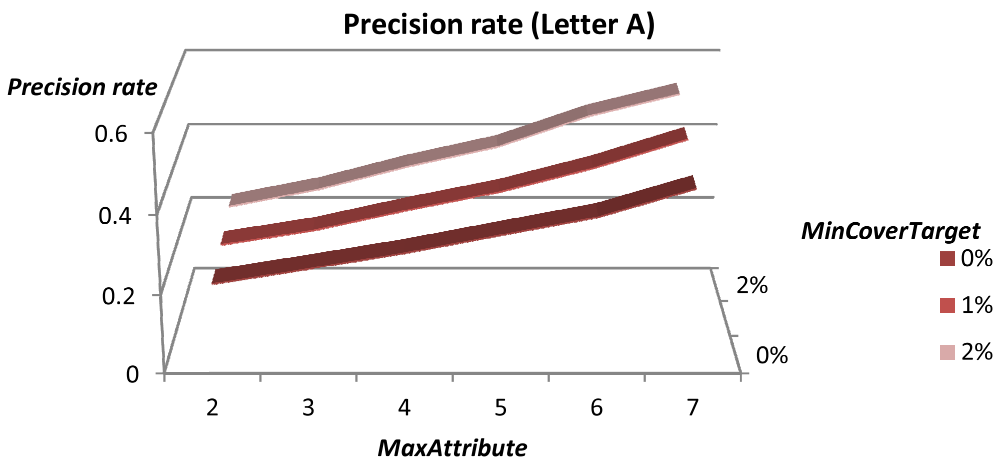


Figure 3.6 Precision of letter A with different values of *MaxAttribute* and *MinCoverTarget*.

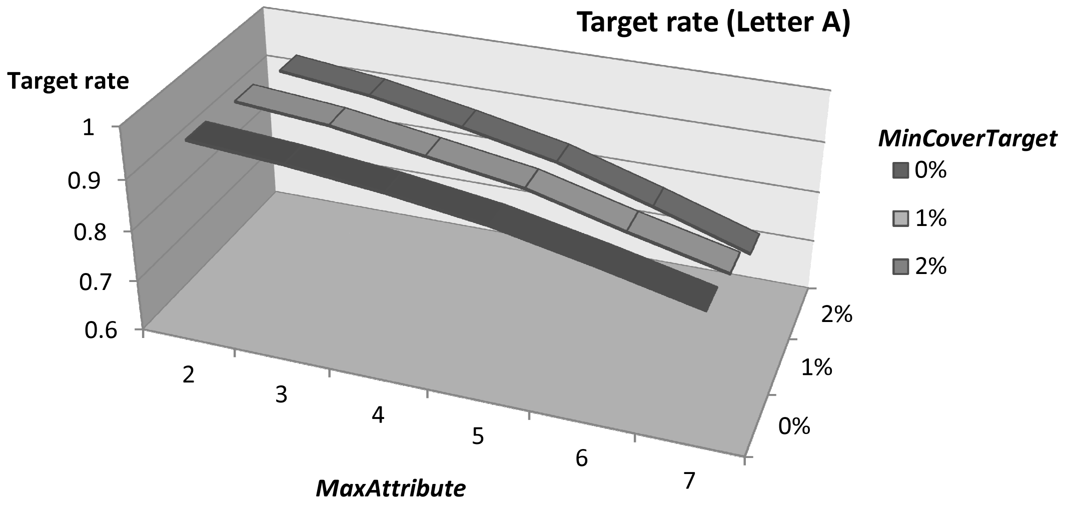


Figure 3.7 Target Rate of letter A with different values of *MaxAttribute* and *MinCoverTarget*.

Depending on *MinCoverTarget* value in Figure 3.7, the target rate decreases from over 0.9 down to about 0.7.

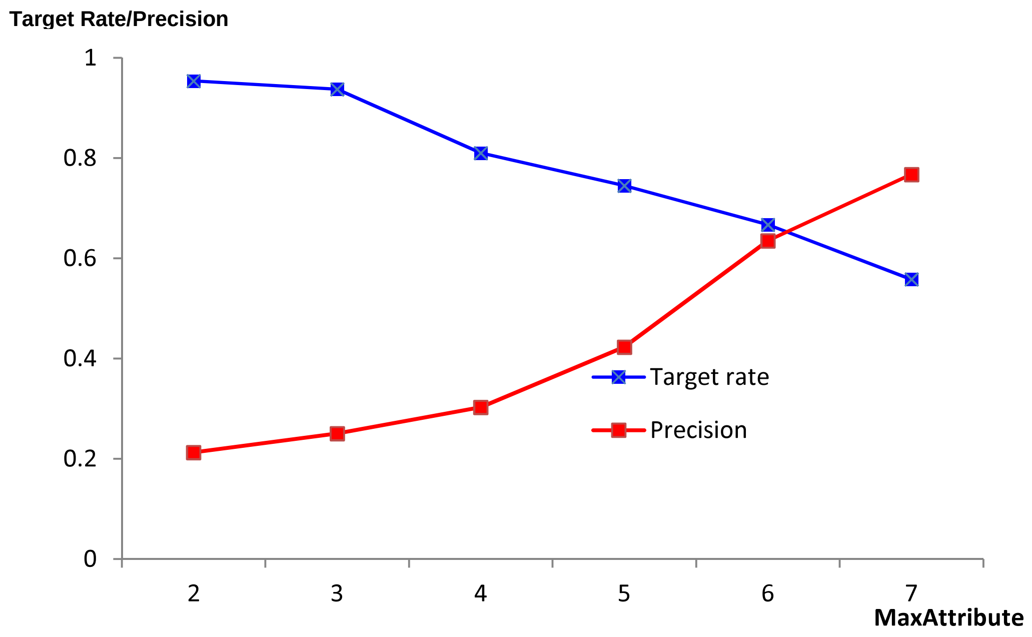


Figure 3.8 Relation between Precision and Target Rate for different values of the *MaxAttribute* (Letter A).

Based on Figures 3.6 and 3.7, if the *MinCoverTarget*=1% then setting *MaxAttribute*=6 balances Precision and Target rate.

3.4 Summary

Rule-based OneClass-DS algorithm is introduced and compared to eight one- class algorithms on 27 data sets. The data sets ranged from numerical, to nominal, to images. The experiments showed that OneClass-DS performs on par with other algorithms in terms of three measures: the accuracy, IPR (Impostor Pass Rate) and FAR (False Alarm Rate).

However the OneClass SVM algorithm, in contrast to OneClass-DS, cannot work on data with missing values. OneClass-DS also gave better results than REP-Tree, Decision Stump and Random Forest in terms of accuracy and FAR measures when tested on missing value data sets.

In addition, OneClass-DS also performed well when used for solving multi-class problems, which were converted into one-class problems. OneClass-DS can be easily tuned for good performance, measured by the Target rate and Precision, by appropriate changing of its parameters.

CHAPTER 4 Conclusions

The major contribution of this work is the development of two rule-based algorithms, mi-DS and OneClass-DS, for solving challenging machine learning problems.

The mi-DS algorithm was designed for solving multiple-instance learning classification problem where data exist for several classes. Instead of single instances one has to deal with bags, or collections, of many instances in one bag.

The OneClass-DS was designed for solving problems when data exist only for one-class; in other words data for other classes are not available. It was shown that the OneClass-DS algorithm also worked well on highly unbalanced data sets when one-class model was generated for a majority class while ignoring the much smaller minority class.

Both algorithms combined rule-based classification with greedy search algorithm based on density of features. The algorithms' performance was compared with many other algorithms on dozens of data sets and the results showed that both original algorithms performed better, or on par, with the many algorithms used in comparison.

In particular, when mi-DS was compared with nine state-of-the-art MIL algorithms on 27 diverse data sets, which ranged from numerical data, to text, to image data, the results indicated that although there was no single generally best-performing algorithm on all data sets, the mi-DS performed very well and was shown to be, on average, very efficient. It also showed good predictive accuracy on most data sets, as measured by both accuracy and MCC criteria. It also exhibited good performance on challenging image and textual data. In addition, mi-DS performed quite well on data with missing values.

Particularly noteworthy is that the approach taken in the mi-DS algorithm can be used as a generic framework for converting other rule-based algorithms so that they can be used to solve MIL problems. This can be done in Step 2 of mi-DS, as the rule generation process can be done by any rule learner, while the construction of the similarity matrix in Step 2, and the prediction procedure used in PredictTestingBag would remain the same.

When OneClass-DS algorithm was compared with eight one-class algorithms on 27 data sets (numerical, nominal, images) it performed on par with other algorithms in terms of accuracy, IPR (Impostor Pass Rate) and FAR (False Alarm Rate) measures. Note that a very efficient OneClass SVM algorithm, in contrast to OneClass-DS, does not work on data with missing values. OneClass-DS also gave better results than REP-Tree, Decision Stump and Random Forest, in terms of accuracy and FAR measures, when tested on data with missing values.

Bibliography

Bibliography

N. Abe, B. Zadrozny, and J. Langford, “Outlier detection by active learning,” in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006, pp. 504-509.

S. Andrews, I. Tsochantaridis, and T. Hofmann, “Support vector machines for multiple-instance learning,” in *Advances in Neural Information Processing Systems 15*, 2002, pp. 577–584.

A. Asuncion and D.J Newman. (2007) UCI Machine Learning Repository. University of California, School of Information and Computer Science, Irvine, CA. [Online]. Available: http://www.ics.uci.edu/~mlearn/MLRepository.html.

P. Auer and R.Ortner, “A boosting approach to multiple instance learning,” in *Proceeding 15th European Conference on Machine Learning*, 2004, pp. 63-74.

P. Baldi and K. Hornik, “Neural networks and principal component analysis: learning from examples without local minima,” *Neural Networks*, vol. 2, pp. 53–58, 1989.

D. Barbará, J. Couto, S. Jajodia and N. Wu, “Adam: a testbed for exploring the use of data mining in intrusion detection, “ in *Proceedings of the ACM SIGMOD 2001 International Conference on Management of Data*, 2001, pp. 15-24.

D. Barbará, N. Wu, and S. Jajodia, “Detecting novel network intrusions using Bayes estimators,” in *Proceeding of the First SIAM Conference on Data Mining*, 2001.

D. Barbará, Y. Li, J.-L. Lin, S. Jajodia, and J. Couto, “Boostrapping a data mining instrusion detection system,” in *Proceeding of ACM Symposium on Applied Computing*, 2003, pp. 421-425.

C. Bishop, *Neural Networks for Pattern Recognition*. Oxford: Oxford University Press, 1995.

H. Blockeeel, D. Page, and A. Srinivasan, “Multi-instance tree learning,” in *Proceeding of 22nd International Conference Machine Learning* , Bonn, Germany, 2005, pp. 57-64.

A. Blum and A. Kalai, “A note on learning from multi-instance examples,” *Machine Learning*, vol. 30, no. 1, pp. 23-29, 1998.

R.W. Brause, T. Langsdorf, and M. Hepp, “Neural data mining for credit card fraud detection,” in *Proceedings of IEEE International Conference on Tools with Artificial Intelligence*, 1999, pp. 103-106.

L. Breiman, “Random forests, “ *Machine Learning*, vol. 45, pp. 5-32, 2001.

L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*. London, U.K.: Chapman & Hall, 1984.

M. Breunig, H.P. Kriegel, R. Ng, and J. Sander, “LOF: indentifying density-based local outliers”, in *Proceedings of the ACM SIGMOD 2000 International Conference on Management of Data*, 2000.

G. Carpenter, S. Grossberg, and D. Rosen, “ART 2-A: an adaptive resonance algorithm for rapid category learning and recognition,” *Neural Networks*, vol. 4, no. 4, pp. 493–504, 1991.

Y. Chen and J. Wang, “Image Categorization by learning and reasoning with regions,” *Journal of Machine Learning Research* , pp. 913-939, 2004.

Y. Chen, J. Bi , and J. Wang, “Miles: Multiple-instance learning via embedded instance selection,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 12, pp. 1931–1947, 2006.

Y. Chevaleyre and J.D. Zucker, “Solving multi-instance and multi-part learning problems with decision trees and rule sets. Application to the mutagenesis problem,” *Lecture Notes in Artificial Intelligence*, Springer, Berlin, pp. 204-214, 2001.

Y. Chevaleyre and J.D. Zucker, “A framework for learning rules from multiple instance data,” in *Proceeding 12th European Conf. on Machine Learning*, 2001, pp. 49-60.

J.Y. Chiang and S.-R. Cheng, “Multiple-instance content-based image retrieval employing isometric embedded similarity measure,” *Pattern Recognition*, pp. 158-166, 2009.

M. Chisholm and P. Tadepalli, “Learning decision rules by randomized iterative local search,” in *Proceeding 2002 International Conference Machine Learning*, 2002, pp. 75–82.

K. J. Cios and L. A.Kurgan, “CLIP4: Hybrid inductive machine learning algorithm that generates inequality rules,” in *Inform. Sci., S. K. Pal and A. Ghosh, Eds.*, vol. 163, pp. 37–83, 2004.

K. J. Cios, W. Pedrycz, R. W. Swiniarski, and L. A. Kurgan, *Data Mining A knowledge Discovery Approach*. Springer, 2007.

W. Cohen, “Fast effective rule induction,” in *Proceeding 12th International Conference Machine Learning*, Lake Tahoe, CA, 1995, pp. 115–123.

J. Demsar, “Statistical Comparisons of Classifiers over Multiple Data Sets,” *Journal of Machine Learning Research*, Vol. 7, pp. 1–30, 2006.

T.G. Dietterich, R.H. Lathrop, and T. Lozano-Perez, “Solving the multi-instance problem with Axis-Paralell Rectangles,” *Artificial Intelligence Journal*, vol. 89, no. 1/2, pp. 31–71,1997.

L.Dong, “A comparision of multi-instance learning algorithms,” Thesis for degree master of science . *University of Waikato,* 2006.

R. Duda and P. Hart, *Pattern Classification and Scene Analysis*. New York: John Wiley & Sons, 1973.

W. Fan, M. Miller, S. J. Stolfo, W. Lee, and P. K. Chan, “Using artificial anomalies to detect unknown and known network intrusions,” in *Proceedings of the 2001 IEEE International Conference on Data Mining*, 2001, pp. 123-130.

U. M. Fayyad, G. Piatesky-Shapiro, P. Smyth, and R. Uthurusamy, “Advances in Knowledge Discovery and Data Mining, “ in *AAAI*, 1996.

M. Friedman, "A comparison of alternative tests of significance for the problem of m rankings," *The Annals of Mathematical Statistics*, Vol. 11, No. 1, pp. 86–92, 1940.

G. Fung, M. Dundar, B. Krishnapuram, and R. B. Rao, “Multiple instance learning for computer aided diagnosis, “ in *Neural Information Processing System*, 2007.

J. Furnkranz and G. Widmer, “Incremental reduced error pruning,” in *Machine Learning: Proc. 11th Annu. Conf.*, New Brunswick, NJ, 1994, pp. 70–77.

T. Gärtner, P. A. Flach, A. Kowalczyk, and A. J. Smola, “Multi-instance kernels,” in *Proc. 19th International Conference on Machine Learning.*, 2002, pp. 179–186.

P. Gehler and O Chapelle, “Deterministic annealing for multiple-instance learning, “ in *Proceeding of AISTATS*, 2007.

K. Hempstalk, E. Frank, and I. H. Witten, “One-class Classification by Combining Density and Class Probability Estimation,” *Department of Computer Science, University of Waikato*, Hamilton, NZ, 2007.

J. Hertz, A. Krogh, and R. Palmer, *Introduction to the theory of neural computation*. Addison Wesley, 1991.

M. Holsheimer and A. P. Siebes, “Data mining: The search for knowledge in databases,” Tech. Rep., CS-R9406, 1994.

W. Hu, Y.Liao, and V. R. Vemuri, “Robust anomaly detection using support vector machines,” in *Proceedings of the International Conference on Machine Learning*, 2002.

N. Japkowicz, C. Myers, and M. Gluck, “A novelty detection approach to classification,” in *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, 1995, pp. 518–523.

G.H. John, “Robust decision trees: Removing outliers from databases,” in *Proceeding of Knowledge Discovery and Data Mining*, 1995, pp. 174-179.

S.S. Keerthi, S.K. Shevade, C. Bhattacharyya, and K.R.K. Murthy, “Improvements to Platt's SMO algorithm for SVM classifier design,” *Neural Computation*, vol.13, no.3, pp. 637-649, 2001.

E. Knorr, R. Ng, and V. Tucakov, “Distance-based outliers: algorithms and applications,” *VLDB journal*, vol. 8, no. 3, pp. 237–253, 2000.

T. Kohonen, *Self-organizing maps*. Heidelberg : Springer-Verlag, 1995.

M. Kraaijveld and R. Duin, “A criterion for the smoothing parameter for parzen-estimators of probability density functions,“ Technical report . Delft University of Technology, 1991.

L.A. Kurgan, K.J. Cios, and S. Dick, “Highly scalable and robust rule learner: Performance evaluation and comparison,” *IEEE Systems, Man, and Cybernetics - Part B: Cybernetics*, Vol. .36, No. 1, pp.32-53, 2006.

L.A. Kurgan and K.J. Cios, “CAIM discretization algorithm,” *IEEE Trans. on Knowledge and Data Engineering,* Vol.16, No. 2, pp. 145-153, 2004.

P. Langley and H. Simon, “Applications of machine learning and rule induction,” *Communications ACM*, Vol. 38, No. 11, pp. 55–64, 1995.

C. Leistner, A. Saffari, and H. Bischof, “MIForests: multiple-instance learning with randomized trees,” in *Proceeding of 11th European Conference on Computer,* 2010.

P.M. Long and L. Tan, “PAC-learning axis aligned rectangles with respect to product distributions from multiple-instance examples,” in *Proceedings of Conference on Computational Learning Theory*, 1996.

M.V. Mahoney and P. K. Chan, “Learning non-stationary models of normal network track for detecting novel attacks,” in *Proceedings of the 8th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002, pp. 376-385.

O. L. Mangasarian and E. W. Wild, “Multiple instance classification via successive linear programming,” *Journal of Optimization Theory and Applications*, pp. 555-568, 2008.

O. Maron and T. Lozano-Perez, “A framework for multi-instance learning, “ *In Advances in Neural Information Processing Systems.* Cambridge, MA: MIT Press, 1998

T. M. Mitchell, *Machine Learning*. New York: McGraw-Hill, 1997.

M. Mizianty, L.A. Kurgan, and M. Ogiela, “Discretization as the enabling technique for the naive bayes and semi-naive bayes based classification,” *Knowledge Engineering Review*, vol. 25(4), pp. 421-449, 2010.

J.F. Murray, G.F. Hughes, and K. Kreutz-Delgado, “Machine learning methods for predicting failures in hard drives: A multiple-instance application,” *Journal of Machine Learning Research*, vol. 6, pp. 783 – 816, 2005.

D. T. Nguyen, C. D. Nguyen, R. Hobson., L. A. Kurgan, and K. J. Cios., “mi-DS: Multiple-instance learning algorithm,” *IEEE Systems, Man, And Cybernetics—Part B: Cybernetics*, Vol. 43, No. 1, pp. 143-154, 2013.

M. E. Otey, S. Parthasarathy, A. Ghoting, G. Li, S. Narravula and D. Panda, “Towards NIC-based intrusion detection,” in *Proceedings of the 9th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2003, pp. 723-728.

G. Pagallo and D. Haussler, “Boolean feature discovery in empirical learning,” *Machine Learning*, Vol. 5, No. 1, pp. 71–99, 1990.

G. Paliouras, V. Karkaletsis, and C. D. Spyropoulos, *Machine Learning and Its Application*. New York: Springer, 2001.

E. Parzen, “On estimation of a probability density function and mode,“ *Annals of Mathenatical Statistics*, Vol. 33, pp.1065–1076, 1962.

M. Pazzani and D. Kibler, “The utility of knowledge in inductive learning,“ *Machine Learning*, Vol. 9, No. 1, pp. 57–94, 1992.

T.A. Pham and A.N.Jain, “Customizing scoring functions for docking, “*Journal of Computer-Aided Molecular Design*, Vol. 22, No. 5, pp. 269-286, 2008.

J. Platt. 1998. Machines using Sequential Minimal Optimization. Advances in Kernel Methods - Support Vector Learning.

F. Provost and P. Domingos, “Tree induction for probability-based ranking,” *Machine Learning*, Vol. 52, No.3, pp. 199–215, 2003.

X. Qi and Y. Han, “Incorporating multiple SVMs for automatic image annotation,” Pattern Recognition, Vol. 40, No. 2, pp. 728-741, 2007.

J. R. Quinlan, *C4.5 Programs for Machine Learning*. New York:Morgan-Kaufmann, 1993.

J.R. Quinlan, “Induction of decision trees,” *Mach. Learn.*, vol. 1, pp. 81–106, 1986.

G. Ratsch, S. Mika, B. Schokopf, and K.-R. Muller, “Constructing boosting algorithms from svms: an application to one-class clasification,” *Pattern Analysis and Machine Intelligence*, Vol. 24, pp. 1184-1199, 2002.

S. Salvador and P. Chan, “Learning states and rules for detecting anomalies in time series,” *Applied Intelligence*, Vol. 23, No. 3, pp. 241-255, 2005.

B. Scholkopf, R. Williamson, A. Smola, and J. Shawe-Taylor, “SV estimation of a distribution’s support,” *in NIPS’99*, 1999.

B. Scholkopf, R. Williamson, A. Smola, J. Shawe-Taylor, and J. Platt, “Support vector method for novelty detection, “ in *Advances in Neural Information Processing Systems 12*, 2000, pp. 582–588.

D. B. Skalak, and E. L. Rissland, “Inductive learning in a mixed paradigm setting,” in *Proceedings of National Conference of American Association for Artificial Intelligence*, 1990, pp. 840-847.

A. Srinivasan, S. Muggleton, R.D. King, and M.J.E. Sternberg, “Mutagenesis: ILP experiments in a non-determinate biological domain, “ *in Proceeding of ILP*, 1994, pp. 217–232.

G. Tandon and P. K.Chan, “Weighting versus pruning in rule validation for detecting network and host anomalies” in *Proceedings of the 13th ACM SIGKDD international conference on Knowlegde discovery and data mining*, 2007, pp. 697-706.

L. Tarassenko, P. Hayton, and M. Brady, ” Novelty detection for the identification of masses in mammograms,” in *Proc. of the Fourth International IEE Conference on Artificial Neural Networks*, vol. 409, 1995, pp. 442–447.

N. Ullman, *Elementary statistics, an applied approach*. Wiley and Sons, 1978.

V. Vapnik, *Statistical Learning Theory*. Wiley, 1998.

J. Wang and J. Zucker, ” Solving the Multi-instance Problem: A lazy learning approach,” in *Proc. 17th Int'l Conf. on Machine Learning*, 2000, pp. 1119-1125.

H. Y. Wang, Q. Yang, and H. Zha, “Adaptive p-posterior mixture-model kernels for multiple instance learning,” in *Proceeding 25th Int'l Conf. on Machine Learning*, 2008, pp. 1136–1143.

G. I. Webb and J. Agar, “Inducing diagnostic rules for glomerular disease with the DLG machine learning algorithm,” *Artif. Intell. Med.*, Vol. 4, pp. 3–14, 1992.

I.H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*. San Francisco: Morgan Kaufmann, 2005.

X. Xu and B. Li, “Multiple Class Multiple-instance learning for image categorization,” Lecture Notes in Computer Science, pp. 155-165, 2007.

X. Xu and E. Frank, “Logistic regression and boosting for labeled bags of instances,” in *Proceedings of the Pacific Asia Conference on Knowledge Discovery and Data Mining*, 2004, pp. 779-806.

T. Yairi, Y. Kato, and K. Hori, “Fault detection by mining association rules from house-keeping data,” in *Proceedings of International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 2001.

A. Ypma and R. Duin,” Support objects for domain approximation, “ in *Proceedings of the International Conference on Artificial Neural Networks 1998,* Skovde, Sweden, 1998.

H. Zengyou, X. Xiaofei , J. Z. Huang, and S. Deng, “A frequent pattern discovery method for outlier detection, “ Springer, pp. 726-732.2004

Y. Zhang, A.C. Surendran, J.C. Platt, and M. Narasimhan, “Learning from multi-topic web documents for contextual advertisement,” in *Proceedings of the 2008 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2008, pp. 1051-1059.

Q. Zhang and S.A. Goldman, “EM-DD: An improved Multi-Instance Learning Technique,” in *Neural Information Processing System*, vol. 14, 2001.

Z. H. Zhou and J. M. Xu, “On the relation between multi-instance learning and semi-supervised learning,” in *Proceeding of 24th International Conference on Machine Learning*, 2007, pp.1167–1174.

Z. H. Zhou, Y. Y. Sun, and Y. F. Li, “Multi-instance learning by treating instances as non-i.i.d. samples, “ in *Proceedings of International Conference on Machine Learning*, 2009.

J. Zhu, S.Rosset, T.Hastie, and R. Tibshirani, “1-norm Support Vector Machines, “ in *Neural Information Processing System*, 2003, pp. 16-24.

Appendices

Appendix A

Measures Used for Evaluating Goodness of Generated Models

A.1 General

Below we define the measures that are used in the dissertation for evaluating the generated models and for comparing them with models generated by other algorithms.

If the input and the corresponding output for each training data point is known then one can construct a confusion matrix (also known as misclassification matrix, or contingency table). It is defined in Table A.1 for a binary classification problem. A multiclass classification problem can always be decomposed into several binary classification problems (one class versus the rest), so the method is general. Notice that a classifier almost always makes errors; they can be false negatives, or false positives, or both.

Table A.1 Misclassification matrix.

		Classifier-predicted classification		
		Positive	Negative	
True (gold standard) classification	Total # Positive (P) cases (P = TP)	True Positives (TP)	False Negatives (FN) ERROR	P = TP + FN
	Total # Negative (N) cases (N = TN)	False Positives (FP) ERROR	True Negatives (TN)	N = FP + TN

Suppose that Positive (P) cases indicate sick people and Negative (N) cases normal/healthy people. True Positives (TP) are then sick people correctly predicted by a classifier as being sick; False Negatives (FN) are sick people incorrectly predicted as healthy (error); True Negatives (TN) are healthy people correctly predicted by a classifier as healthy; False Positives (FP) are healthy people incorrectly predicted as sick (error). From the misclassification matrix we can calculate several types of measures, some of which are shown in Table A.2.

Table A.2 Most often used measure calculated from the confusion matrix.

Measure	Formula	Meaning
Sensitivity (also known as Recall, or True Positive Rate (TPR)	$\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$	Sensitivity is the test's ability to correctly predict TP cases. It is a probability of a positive test given that the patient is sick. Sensitivity of 1 means that the model recognizes all sick people as sick.
Specificity	$\frac{\mathrm{TN}}{\mathrm{FP}+\mathrm{TN}}$	Specificity is the ability of the test to correctly identify TN cases. It is a probability of a negative test given that the patient is healthy. Specificity of 1 means the test recognizes all negatives/healthy people as healthy.
False Positive Rate (FPR) (also known as False-alarm rate)	1 – specificity $=\frac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TN}}$	FPR defines how many incorrect positive cases occur among all negative cases.
Precision	$\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}$	Proportion of TP to all positive cases (both true positives and false positives)
Accuracy	$\frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{TP}+\mathrm{FN}+\mathrm{TN}+\mathrm{FP}}$	proportion of true results (both true positives and true negatives) in the population
MCC (Matthews correlation coefficient)	$\frac{\mathrm{TP}*\mathrm{TN}-\mathrm{FP}*\mathrm{FN}}{\sqrt{(\mathrm{TP}+\mathrm{FP})(\mathrm{TP}+\mathrm{FN})(\mathrm{TN}+\mathrm{FP})(\mathrm{TN}+\mathrm{FN})}}$	A correlation coefficient between the observed and predicted binary classifications; it is between −1 and +1. +1 represents a perfect prediction, 0 no better than random prediction, and −1 indicates total disagreement between prediction and observation.

Two robust non-parametric tests for statistical comparison of predictive power of classifiers are the Wilcoxon signed-ranks test and the Friedman test; the latter is used for comparison of more than two classifiers over multiple data sets. Both tests do not make any assumption about the underlying distribution of the data.

A.2 Statistical

Wilcoxon Signed-Ranks Test

The Wilcoxon signed-ranks test (Wilcoxon, 1945) is a non-parametric test that ranks the difference in performances of two classifiers on a data set, which ignores the signs and compares the ranks for the positive and the negative differences.

Let d_i be the difference between the performance scores of two classifiers on N data sets. The differences are ranked according to their absolute values; average ranks are assigned in case of ties. $R+$ is the sum of ranks for the data sets on which the first algorithm outperformed the second, while $R-$ is the sum of ranks for the opposite (second outperformed the first). Ranks of $d_i = 0$ are split evenly among the two sums; if there is an odd number of $d_i = 0$, one is simply ignored. The sums are defined as follows:

$$R+= \sum_{d_i>0} rank(d_i) + \frac{1}{2} \sum_{d_i=0} rank(d_i)$$

$$R-= \sum_{d_i<0} rank(d_i) + \frac{1}{2} \sum_{d_i=0} rank(d_i)$$

Let *T* indicate the smaller of the two sums:

T = min(*R*+, *R*-)

Then, we look up in a table a critical value for *T* at a certain confidence level and compare our T with this value. If it is smaller than the null-hypothesis is accepted, otherwise it is rejected.

We illustrate the calculations by means of the example shown in Table A.3.

Table A.3 Results, in terms of accuracy, of algorithms A and B on 13 data sets.

Data set name	Algorithm A (2)	Algorithm B (3)	Difference (4)= (2) – (3)	Absolute difference (5)=|column 4|	Rank (6)
1	86.68	90.37	-3.69	3.69	4
2	77	84.61	-7.61	7.61	7
3	80.02	73.19	6.83	6.83	6
4	74.74	75.36	-0.62	0.62	1
5	79.5	74.06	5.44	5.44	5
6	79.5	50	29.5	29.5	12
7	64.5	50	14.5	14.5	10
8	64.28	50	14.28	14.28	9
9	89	87.63	1.37	1.37	2
10	64.17	45	19.17	19.17	11
11	60	50	10	10	8
12	100.00	60.45	39.55	39.55	13
13	68.77	66.95	1.82	1.82	3

Column 1: data set name

Column 2: 10-FCV accuracy of algorithm A

Column 3: 10-FCV accuracy of algorithm B

Column 4: difference between the two accuracies

Column 5: absolute difference

Column 6: rank (from 1 to 13) based on values in column 5

Null-hypothesis is H_0: algorithms A and B perform at the same level (in accuracy).

Decision Rule: Reject H_0 if T<=critical value at α= 0.05.

Now, using column 6, we calculate the sum of ranks for which column 4 values were positive (red color), for the case when algorithm A performed better than algorithm B:

Sum R+ = 6+5+12+10+9+2+11+8+13+3 = 79

Similarly we calculate the sum of ranks when algorithm B performed better than A (blue color):

Sum R- = 4+7+1 = 12

Obviously

Sum R_0 = 0

Thus, the sums are

$R+$ = (SumR+) + 0.5*Sum R_0 = 79

$R-$ = (SumR-) + 0.5*Sum R_0 = 12

From the two we calculate:

T = min ($R+$, $R-$) = 12

Assuming confidence level $\alpha = 0.05$, with the total number of data sets $N = 13$, we find in the table of critical values for the Wilcoxon test (part of which is shown in Table A.4) that the critical value is 17.

Table A.4 Part of Wilcoxon's test table.

N	**0.05**	0.02	0.01
8	4	2	0
9	6	3	2
10	8	5	3
11	11	7	5
12	14	10	7
13	**17**	13	10
14	21	16	13

Using the value 17, and the calculated T=12, we see that $T \leq 17$ so we reject the null-hypothesis. In other words, the two algorithms do not perform at the same level.

Friedman Test

The Friedman test is used for comparing performance of two or more algorithms over several data sets. The input is a table of results. They are ranked across the rows and the mean rank for each column is calculated. We use the results (in terms of accuracy) shown in Table A.5 to illustrate.

Table A.5 Results of six algorithms on 11 data sets.

Data Set Name	Algorithm 1	Algorithm 2	Algorithm 3	Algorithm 4	Algorithm 5	Algorithm 6
1	0.89	0.92	0.76	0.92	0.8	0.82
2	0.97	0.89	0.97	0.75	0.9	0.93
3	0.61	0.6	0.68	0.67	0.66	0.59
4	0.5	0.5	0.63	0.89	0.8	0.96
5	0.47	0.45	0.42	0.63	0.45	0.68
6	0.51	0.51	0.5	0.52	0.51	0.53
7	0.61	0.54	0.65	0.63	0.68	0.69
8	0.52	0.51	0.52	0.51	0.53	0.5
9	0.71	0.7	0.65	0.71	0.71	0.61
10	0.44	0.42	0.41	0.47	0.44	0.35
11	0.41	0.42	0.42	0.39	0.44	0.42

Null hypothesis is H_0: All algorithms perform at about the same level in accuracy

Decision Rule: Reject H_0 if Fr >=critical value at $\alpha = 0.05$.

Calculations

The differences between the sum of the ranks is evaluated by calculating the Friedman statistic (Fr) using this formula:

$$Fr = \frac{12}{\mathrm{nk(k+1)}} \sum \mathrm{R}_\mathrm{j}^2 - 3\mathrm{n(k+1)}$$

where:

k is # of columns (also called “treatments”)
n is # of rows (also called “blocks”)
R_j is sum of the ranks in column j.

If there is no significant difference between the sum of the ranks for each column, then Fr is small, but if at least one column shows significant difference then Fr is larger.

From Table A.5 we find ranks for each algorithm for each data set and they are shown in Table A.6.

Table A.6 Ranking of the six algorithms on 11 data sets.

Data set Name	Algorithm 1	Algorithm 2	Algorithm 3	Algorithm 4	Algorithm 5	Algorithm 6
1	4.0	5.5	1.0	5.5	2.0	3.0
2	5.5	2.0	5.5	1.0	3.0	4.0
3	3.0	2.0	6.0	5.0	4.0	1.0
4	1.5	1.5	3.0	5.0	4.0	6.0
5	4.0	2.5	1.0	5.0	2.5	6.0
6	3.0	3.0	1.0	5.0	3.0	6.0
7	2.0	1.0	4.0	3.0	5.0	6.0
8	4.5	2.5	4.5	2.5	6.0	1.0
9	5.0	3.0	2.0	5.0	5.0	1.0
10	4.5	3.0	2.0	6.0	4.5	1.0
11	2.0	4.0	4.0	1.0	6.0	4.0
Sum of ranks	39	30	34	44	45	39
Sum of $(\text{ranks})^2$	1521	900	1156	1936	2025	1521

# of algorithms (k)	6
# of data sets (n)	11
Sigma R_j^2	9059
12/(nk(k+1))	0.026
3n(k+1)	231
Fr value	4.31

Assuming the confidence level of 0.05, we look up the critical value in the chi-squared ($\chi 2$) distribution table with (k-1) degrees of freedom (part of which is shown in Table A.7), and it is 11.07.

Total number of algorithms, k=6, so degree of freedom (df) is 5.

Table A.7 Part of the chi-squared (χ2) distribution table.

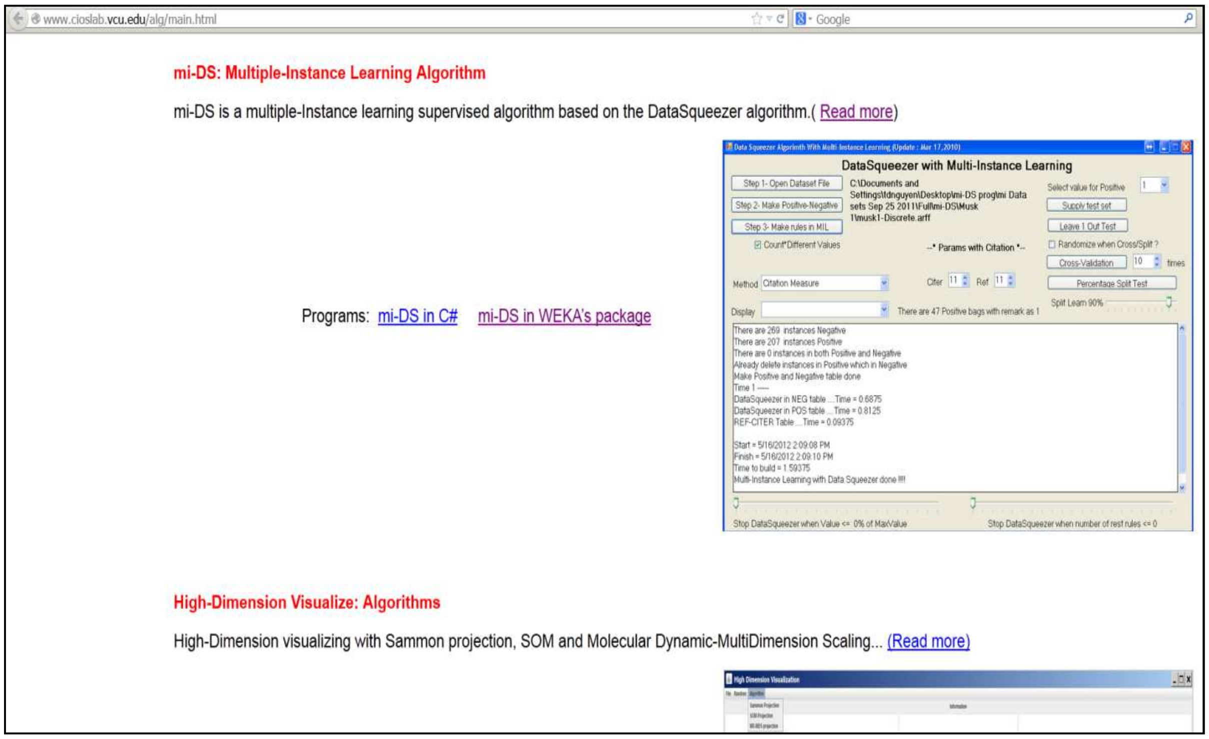
df	0.25	0.1	**0.05**	0.025	0.01	0.005
3	4.11	6.25	7.81	9.35	11.34	12.84
4	5.39	7.78	9.49	11.14	13.28	14.86
5	6.63	9.24	**11.07**	12.83	15.09	16.75
6	7.84	10.64	12.59	14.45	16.81	18.55

Because Fr = 4.31 < critical value (11.07) we accept H_0 and conclude that that all algorithms perform at the same level, i.e., none of them is significantly better than the rest.

Appendix B

mi-DS and OneClass Algorithms Implementation Details

We implemented OneClass-DS and mi-DS algorithms as packages in popular WEKA software so they can be downloaded and embedded into WEKA for easy use and testing. To download and install them go to www.cioslab.vcu.edu/alg/main.html (Figure B.1 shows a screen shot).



mi-DS: Multiple-Instance Learning Algorithm

mi-DS is a multiple-Instance learning supervised algorithm based on the DataSqueezer algorithm.(Read more)

Programs: mi-DS in C# mi-DS in WEKA's package

Data Squeezer Algprimth With Multi-Instance Learning (Update : Mar 17,2010)

DataSqueezer with Multi-Instance Learning

C:\Documents and Settings\tdnguyen\Desktop\mi-DS prog\mi Data sets Sep 25 2011\Full\mi-DS\Musk 1\musk1-Discrete.arff

Select value for Positive: 1

[✓] Count*Different Values

☐ Randomize when Cross/Split ?

Cross-Validation 10 times

--* Params with Citation *--

Method: Citation Measure

Citer: 11 Ref: 11

Split Learn 90%

There are 47 Positive bags with remark as 1

```
There are 269  instances Negative
There are 207  instances Positive
There are 0 instances in both Positive and Negative
Already delete instances in Positive which in Negative
Make Positive and Negative table done
Time 1 -----
DataSqueezer in NEG table ....Time = 0.6875
DataSqueezer in POS table ....Time = 0.8125
REF-CITER Table ....Time = 0.09375

Start = 5/16/2012 2:09:08 PM
Finish = 5/16/2012 2:09:10 PM
Time to build = 1.59375
Multi-Instance Learning with Data Squeezer done !!!!
```

Stop DataSqueezer when Value <= 0% of MaxValue

Stop DataSqueezer when number of rest rules <= 0

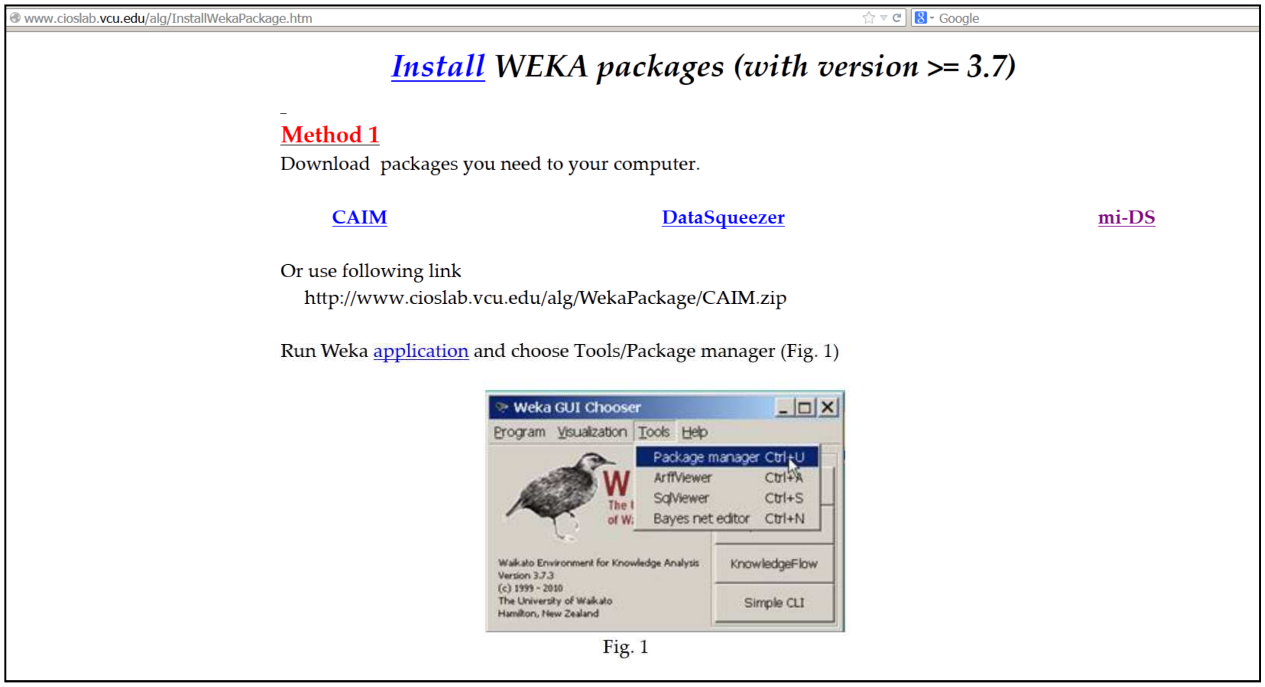
Step 1- Open Dataset File
Step 2- Make Positive-Negative
Step 3- Make rules in MIL
Supply test set
Leave 1 Out Test
Percentage Split Test
Display

High-Dimension Visualize: Algorithms

High-Dimension visualizing with Sammon projection, SOM and Molecular Dynamic-MultiDimension Scaling... (Read more)

Figure B.1 Data Mining Tools web page.

For example, if one chooses to embed the mi-DS algorithm, then a new web page appears as shown in a screen shot in Figure B.2.



www.cioslab.vcu.edu/alg/InstallWekaPackage.htm

Install WEKA packages (with version >= 3.7)

Method 1

Download packages you need to your computer.

CAIM DataSqueezer mi-DS

Or use following link

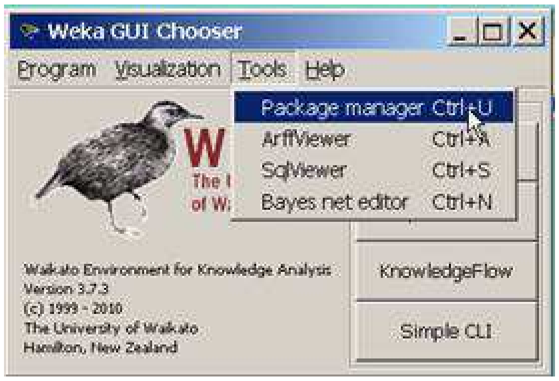
http://www.cioslab.vcu.edu/alg/WekaPackage/CAIM.zip

Run Weka application and choose Tools/Package manager (Fig. 1)

Fig. 1

Figure B.2 WEKA packages's web page.

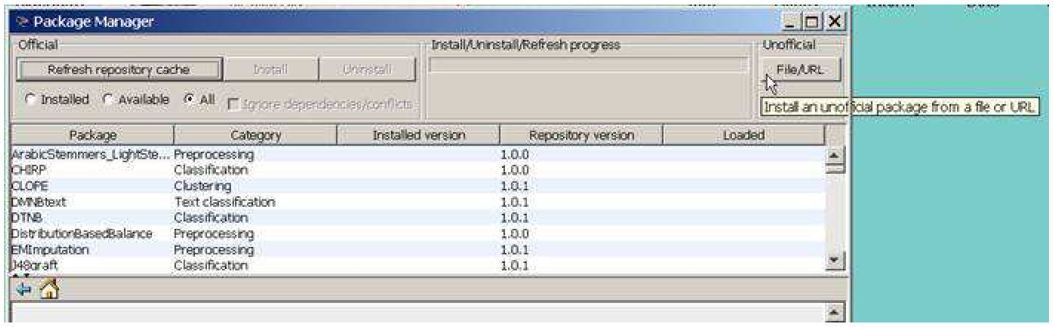
The user now can choose the required package (i.e., mi-DS) to download and save it on her/his computer. To install the downloaded package (algorithm), run the WEKA (the assumption is that the user has the WEKA already installed on his/her computer) on your computer and choose the "Tools: Package manager" in the main window, as illustrated in Figure B.3.



Waikato Environment for Knowledge Analysis
Version 3.7.3
(c) 1999 - 2010
The University of Waikato
Hamilton, New Zealand

Figure B.3 Menu Package manger in WEKA.

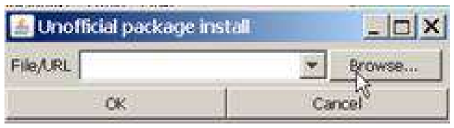
A new window will appear that lists all current packages. Then click button "File/URL" at the upper right hand corner (see Figure B.4).


Package Manager
Official
Refresh repository cache
Install
Uninstall
Installed
Available
All
Ignore dependencies/conflicts
Install/Uninstall/Refresh progress
Unofficial
File/URL
Install an unofficial package from a file or URL

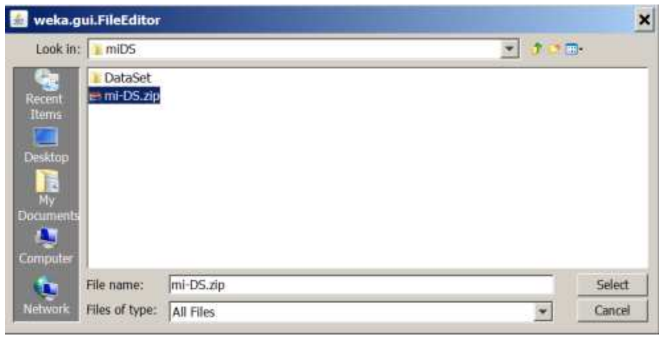
Package	Category	Installed version	Repository version	Loaded
ArabicStemmers_LightSte...	Preprocessing		1.0.0	
CHIRP	Classification		1.0.0	
CLOPE	Clustering		1.0.1	
DMNBtext	Text classification		1.0.1	
DTNB	Classification		1.0.1	
DistributionBasedBalance	Preprocessing		1.0.0	
EMImputation	Preprocessing		1.0.1	
J48graft	Classification		1.0.1	

Figure B.4 Package Manager window.

And follow the three steps (shown in Figure B.5) to choose the downloaded and saved package.



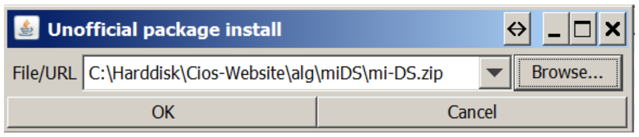
Step 1: Click button “Browse”.



File name: mi-DS.zip

Files of type: All Files

Step 2: Search and choose downloaded package and click “Select”.

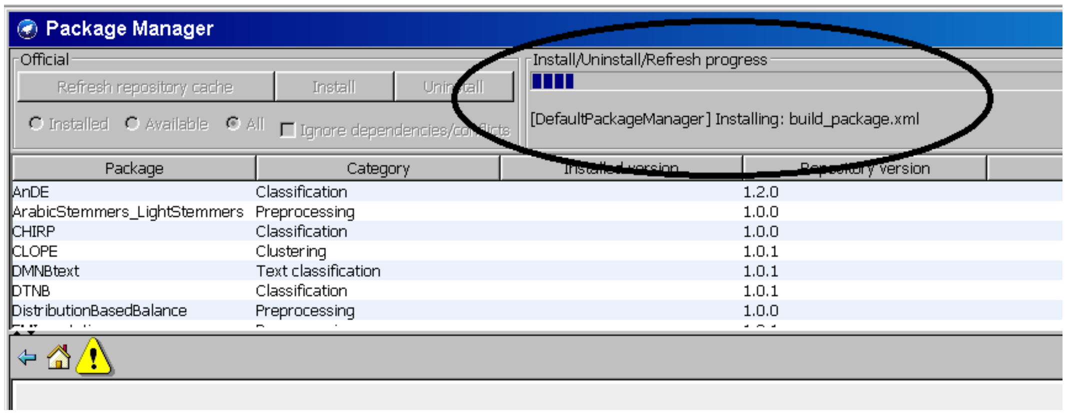


File/URL C:\Harddisk\Cios-Website\alg\miDS\mi-DS.zip

Step 3: Click button “OK”

Figure B.5 Choose the package to install.

WEKA will install the chosen package (mi-DS in our case), as shown in Figure B.6.

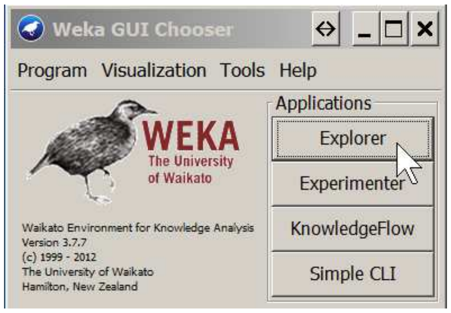


[DefaultPackageManager] Installing: build_package.xml

Package	Category	Installed version	Repository version
AnDE	Classification		1.2.0
ArabicStemmers_LightStemmers	Preprocessing		1.0.0
CHIRP	Classification		1.0.0
CLOPE	Clustering		1.0.1
DMNBtext	Text classification		1.0.1
DTNB	Classification		1.0.1
DistributionBasedBalance	Preprocessing		1.0.0

Figure B.6 Installing chosen package in WEKA.

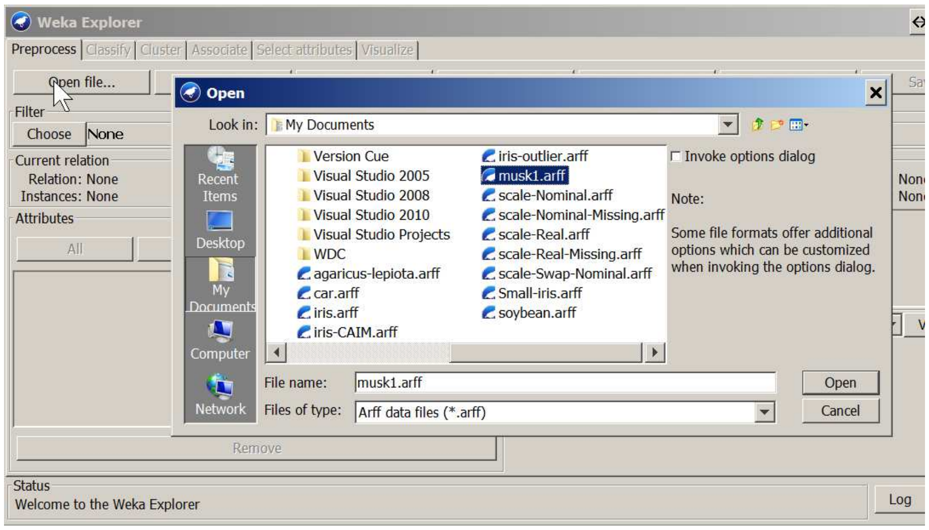
Open a data set in WEKA by clicking button “Explorer” in the main window, as shown in Figure B.7.



Waikato Environment for Knowledge Analysis
Version 3.7.7
(c) 1999 - 2012
The University of Waikato
Hamilton, New Zealand

Figure B.7 Open data set in WEKA.

Now we need to choose a data set to be analyzed, see Figure B.8, which shows we choose a data set named MUSK1 (benchmark for MIL testing).



Current relation
Relation: None
Instances: None

Look in: My Documents

Version Cue	iris-outlier.arff
Visual Studio 2005	musk1.arff
Visual Studio 2008	scale-Nominal.arff
Visual Studio 2010	scale-Nominal-Missing.arff
Visual Studio Projects	scale-Real.arff
WDC	scale-Real-Missing.arff
agaricus-lepiota.arff	scale-Swap-Nominal.arff
car.arff	Small-iris.arff
iris.arff	soybean.arff
iris-CAIM.arff	

☐ Invoke options dialog

Note:

Some file formats offer additional options which can be customized when invoking the options dialog.

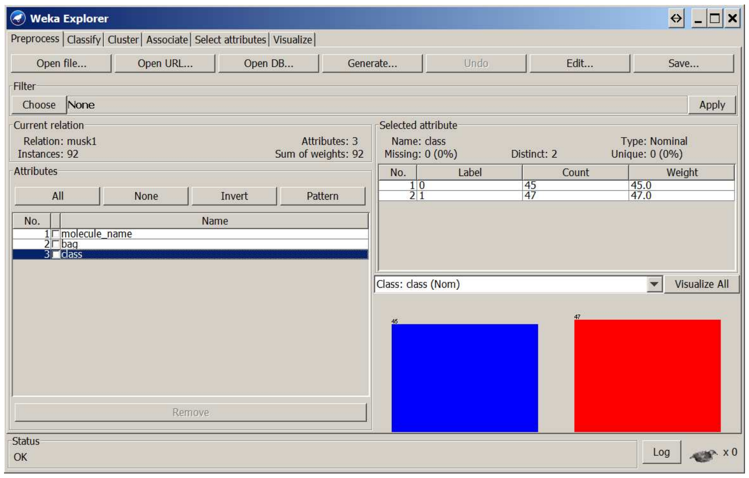
File name: musk1.arff

Files of type: Arff data files (*.arff)

Status
Welcome to the Weka Explorer

Figure B.8 Choose data set in WEKA.

The selected data set will be opened and loaded into WEKA application, as shown in Figure B.9.



Filter: None

Current relation
Relation: musk1
Instances: 92
Attributes: 3
Sum of weights: 92

Attributes

No.	Name
1	molecule_name
2	bag
3	class

Selected attribute
Name: class
Missing: 0 (0%)
Distinct: 2
Type: Nominal
Unique: 0 (0%)

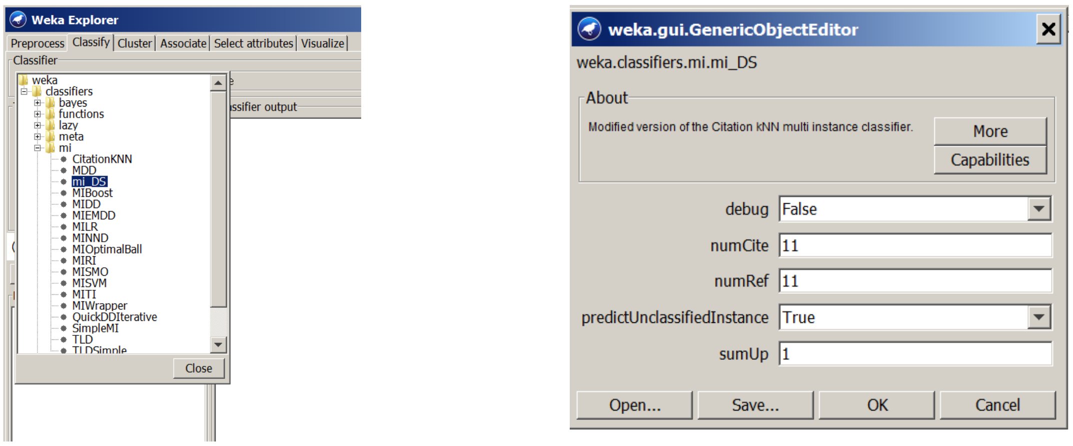
No.	Label	Count	Weight
1	0	45	45.0
2	1	47	47.0

Class: class (Nom)

Status
OK

Figure B.9 Musk 1 data set .

Now, the user can choose the mi-DS algorithm and set values for its parameters, as shown in Figure B.10, or use the default values (they are displayed in the image on the right hand side and appear there automatically).



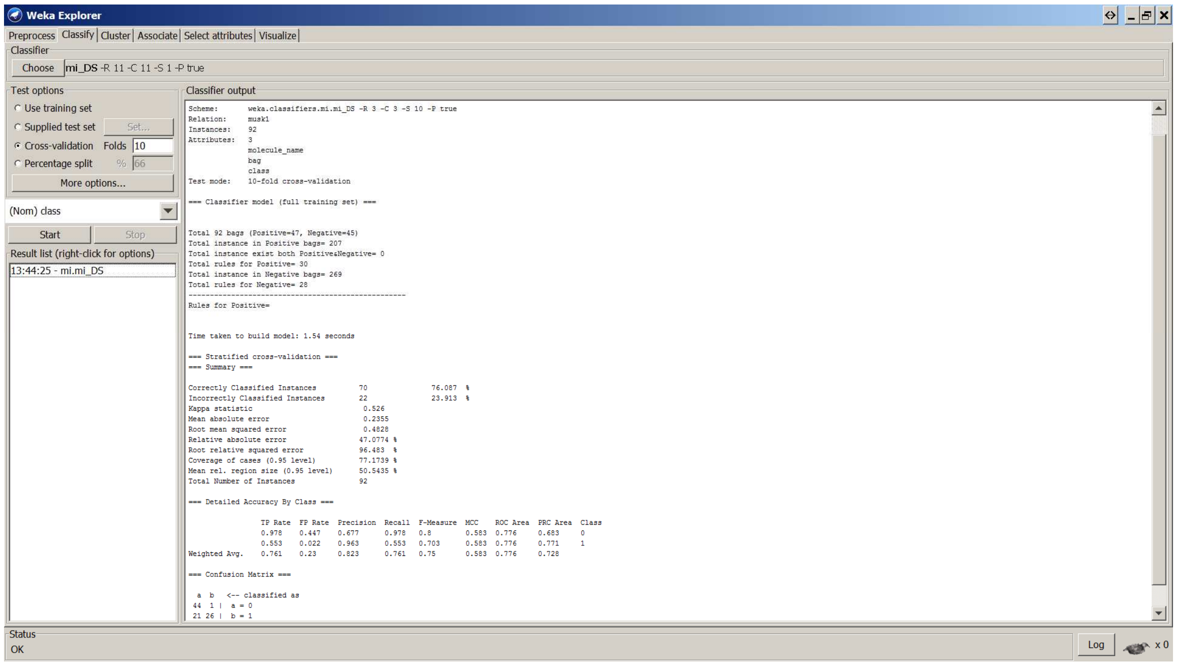
weka.classifiers.mi.mi_DS

Modified version of the Citation kNN multi instance classifier.

debug False
numCite 11
numRef 11
predictUnclassifiedInstance True
sumUp 1

Figure B.10 Select mi-DS and setting values of parameters.

mi-DS will generate the rules that are shown in Figures B.11 using 10-FCV and in Figure B.12 using splitting technique (with 90% for training and 10% for testing).



Classifier: mi_DS -R 11 -C 11 -S 1 -P true

Test options: Cross-validation Folds 10

(Nom) class

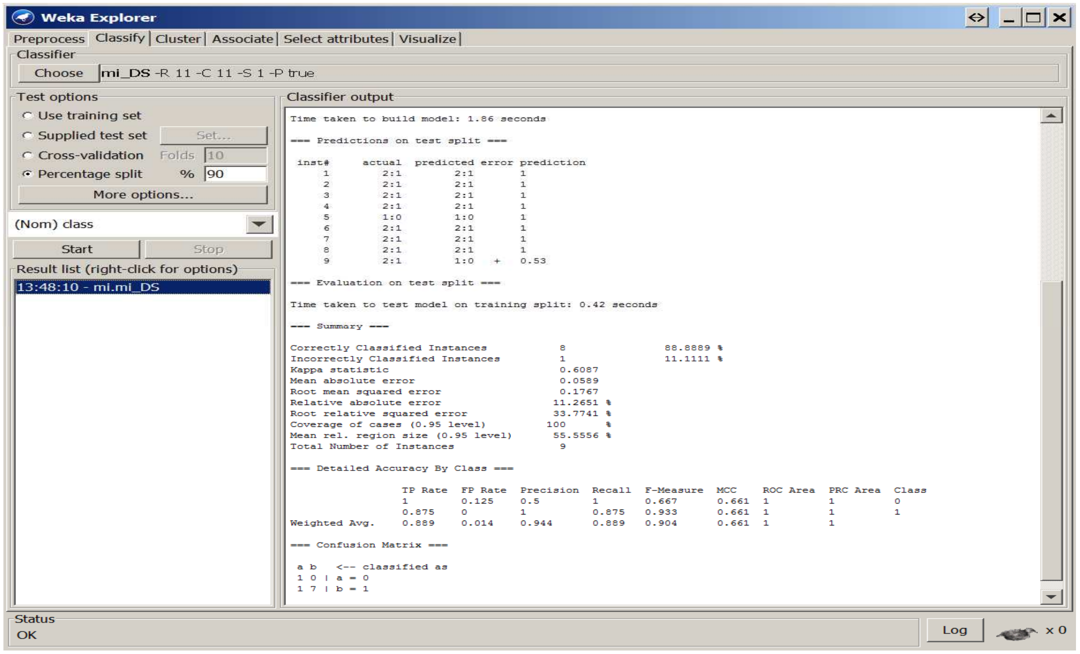
Result list (right-click for options): 13:44:25 - mi.mi_DS

Classifier output

```
Scheme:       weka.classifiers.mi.mi_DS -R 3 -C 3 -S 10 -P true
Relation:     musk1
Instances:    92
Attributes:   3
              molecule_name
              bag
              class
Test mode:    10-fold cross-validation

=== Classifier model (full training set) ===


Total 92 bags (Positive=47, Negative=45)
Total instance in Positive bags= 207
Total instance exist both Positive&Negative= 0
Total rules for Positive= 30
Total instance in Negative bags= 269
Total rules for Negative= 28
---------------------------------------------------
Rules for Positive=


Time taken to build model: 1.54 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances          70               76.087  %
Incorrectly Classified Instances        22               23.913  %
Kappa statistic                          0.526
Mean absolute error                      0.2355
Root mean squared error                  0.4828
Relative absolute error                 47.0774 %
Root relative squared error             96.483  %
Coverage of cases (0.95 level)          77.1739 %
Mean rel. region size (0.95 level)      50.5435 %
Total Number of Instances               92

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC    ROC Area  PRC Area  Class
                 0.978    0.447    0.677      0.978    0.8        0.583  0.776     0.683     0
                 0.553    0.022    0.963      0.553    0.703      0.583  0.776     0.771     1
Weighted Avg.    0.761    0.23     0.823      0.761    0.75       0.583  0.776     0.728

=== Confusion Matrix ===

  a  b   <-- classified as
 44  1 |  a = 0
 21 26 |  b = 1
```

Status: OK

Figure B.11 Results using 10-FCV.



Classifier: mi_DS -R 11 -C 11 -S 1 -P true

Test options: Percentage split % 90

(Nom) class

Result list (right-click for options): 13:48:10 - mi.mi_DS

Classifier output

```
Time taken to build model: 1.86 seconds

=== Predictions on test split ===

 inst#     actual  predicted error prediction
     1        2:1        2:1       1
     2        2:1        2:1       1
     3        2:1        2:1       1
     4        2:1        2:1       1
     5        1:0        1:0       1
     6        2:1        2:1       1
     7        2:1        2:1       1
     8        2:1        2:1       1
     9        2:1        1:0   +   0.53

=== Evaluation on test split ===

Time taken to test model on training split: 0.42 seconds

=== Summary ===

Correctly Classified Instances           8               88.8889 %
Incorrectly Classified Instances         1               11.1111 %
Kappa statistic                          0.6087
Mean absolute error                      0.0589
Root mean squared error                  0.1767
Relative absolute error                 11.2651 %
Root relative squared error             33.7741 %
Coverage of cases (0.95 level)         100      %
Mean rel. region size (0.95 level)      55.5556 %
Total Number of Instances                9

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC     ROC Area  PRC Area  Class
                 1        0.125    0.5        1        0.667      0.661   1         1         0
                 0.875    0        1          0.875    0.933      0.661   1         1         1
Weighted Avg.    0.889    0.014    0.944      0.889    0.904      0.661   1         1

=== Confusion Matrix ===

 a b   <-- classified as
 1 0 | a = 0
 1 7 | b = 1
```

Status: OK

Figure B.12 Results using 90% of instances for training and 10% for testing.

Similarly, other packages from our website can be embedded into WEKA and used.

VITA

Dat T. Nguyen was born in Saigon, Vietnam. He received his Bachelor and Master degrees from Computer Science Department, School of Science, National University of Ho Chi Minh city, Vietnam, in 1991 and 1997, respectively. In 2009 he joined Department of Computer Science at VCU as a Graduate Research Assistant. His research interests include data mining, machine learning, high-dimension data visualization, and pattern recognition.

List of Relevant Publications:

Journal papers

[1] D.T. Nguyen, C. Nguyen, R. Hobson, L.A. Kurgan, and K.J. Cios. mi-DS: Multiple-Instance Learning Algorithm. IEEE Systems, Man, and Cybernetics -Part B: Cybernetics, pp. 143-154, Vol. 43, No. 1, Feb 2013.

[2] D.T. Nguyen and K.J. Cios. O*neClass-DS Algorithm*, submitted to a journal.

Conference paper

[3] D.T. Nguyen, W. Dzwinel and K.J. Cios. *Visualization of Highly-Dimensional Data in 3D Space*, ISDA 2011 Conference, Còrdoba, Spain, pp. 225-230