The American University in Cairo

School of Sciences and Engineering

ADR-Miner - An Ant-Based Data Reduction Algorithm for Classification

A Thesis Submitted to

Department of Computer Science and Engineering

In partial fulfilment of the requirements for

the degree of Master of Science

By

Ismail M. Anwar

Under the Supervision of

Dr. Ashraf M. Abdelbar

May, 2015

The American University in Cairo

ADR-Miner - An Ant-Based Data Reduction Algorithm for Classification

A Thesis Submitted by

Ismail Mohamed Anwar Abdel Salam

To the Master of Science in Computer Science Program

March / 2015

In partial fulfillment of the requirements for

The degree of Master of Science

Has been approved by

Thesis Committee Supervisor/Chair _____

Affiliation _____

Thesis Committee Reader/Examiner _____

Affiliation _____

Thesis Committee Reader/Examiner _____

Affiliation _____

Thesis Committee Reader/External Examiner _____

Affiliation _____

_____	_____	_____	_____
Dept. Chair/Director	Date	Dean	Date

Acknowledgements

I would like to start by thanking Dr. Ashraf Abdelbar for his guidance and patience, for without them this work would not have been possible. I would also like to thank my family and friends for their unyielding support throughout the entire process. To all those who helped me complete this thesis, I sincerely thank you and will forever cherish your support.

Abstract

Classification is a central problem in the fields of data mining and machine learning. Using a training set of labeled instances, the task is to build a model (classifier) that can be used to predict the class of new unlabeled instances. Data preparation is crucial to the data mining process, and its focus is to improve the fitness of the training data for the learning algorithms to produce more effective classifiers. Two widely applied data preparation methods are feature selection and instance selection, which fall under the umbrella of data reduction. For my research I propose ADR-Miner, a novel data reduction algorithm that utilizes ant colony optimization (ACO). ADR-Miner is designed to perform instance selection to improve the predictive effectiveness of the constructed classification models. Two versions of ADR-Miner are developed: a base version that uses a single classification algorithm during both training and testing, and an extended version which uses separate classification algorithms for each phase. The base version of the ADR-Miner algorithm is evaluated against 20 data sets using three classification algorithms, and the results are compared to a benchmark data reduction algorithm. The non-parametric Wilcoxon signed-ranks test will is employed to gauge the statistical significance of the results obtained. The extended version of ADR-Miner is evaluated against 37 data sets using pairings from

five classification algorithms and these results are benchmarked against the performance of the classification algorithms but without reduction applied as pre-processing.

Keywords: Ant Colony Optimization (ACO), Data Mining, Classification, Data Reduction.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Data mining is the process of extracting insightful knowledge from large quantities of data either in an automated or a semi-automated fashion [19], [34]. Data mining techniques have been traditionally divided into four categories: classification, clustering, regression and association rule mining. Classification techniques analyze a given data set and attempt to learn the relationship between the input attributes in the data set and a categorical class label. This relationship is then used to build a model that can be used to predict the class label of unforeseen instances. Since the classes for the instances used to build the model are known beforehand, classification is considered to be a form of supervised learning. Clustering is in direct contrast with classification in that it analyzes data sets that have no known class associated with the instances in those data sets. The aim of clustering is to uncover hidden relationships between instances in the data sets being analyzed. In essence, a clustering algorithm will attempt to group the instances in the data set being analyzed into naturally occurring classes, where the instances assigned to a group are "similar" to one another and "dissimilar"

to members in other groups. Depending on the clustering algorithm, cluster membership may vary from mutual exclusivity to supporting membership to multiple clusters at the same time. Similarity or dissimilarity in clustering is defined by a distance function - a function that returns the distance between two instances in a data set. As the class labels are not known beforehand, clustering is considered to be a form of unsupervised learning. Regression (and other numerical techniques) are used to build models capable of predicting unseen values in data series that are continuous in nature. Finally, association rule mining analyzes data sets in search of frequently occurring patterns within its instances and attempts to encode these findings in the form of rules. The rules can be used to predict the occurrence of related items when one or more items that are frequently associated with them are detected.

As the work presented in this dissertation is related to classification, let us revisit it and formally define classification. Given a set of labeled instances, the aim of a classification algorithm is to learn the relationship between the input attributes and the label and encode this information as a model. The label associated with each instance defines the class to which the instance belongs, and has to be categorical in nature. If the label is continuous in fashion, then the data mining task used to build models for this label is usually identified as belonging to the regression family of algorithms. The model, as mentioned earlier is used to identify the class of instances that have not been seen by the classification algorithm before. The data set used to *learn* the model is usually known as the training set. Many techniques have been developed over time to perform classification, and these include

decision trees, classification rules, probabilistic graphical models, instance-based learning, neural networks and support vector machines [5], [8], [25], [34].

With the increasing availability of affordable computational power, abundant cheap storage, and the fact that more and more data are starting their life in native digital form, more and more pressure is put on classification algorithms to extract effective and useful models. Real world data sets are frequently rife with noisy, erroneous, outlier, miss-labeled and irrelevant data. These can be harmful to the classification algorithm and, in turn, may affect the predictive power of the produced classification models. In an attempt to remedy these maladies, the data presented to the algorithm for the purposes of training is put through a phase that either removes instances, attributes or both before the actual learning process. This pre-processing phase is known as data reduction. Not only does the data reduction process aim to improve the predictive effectiveness of the produced classifiers by removing the attributes and instances that can be misleading to the learning algorithm, it also decreases the size of the training set presented to the algorithm by keeping only the most representative instances. The benefits of having a reduced set to work with is two-fold: learning using a smaller volume of data is faster, and the maintenance overhead is diminished.

Ant colony optimization (ACO) [7], [15], [16] is a search meta-heuristic inspired by the behavior of real ants as they forage for food. By observing the ants, it was noted the ants are capable of converging on the shortest path between their nest and the closest food source without any means of central coordination. This *emergent* behavior displayed by the ants is used

to develop a meta-heuristic that utilizes the same underlying mechanisms to tackle combinatorial optimization problems. Since its introduction, ACO has shown a successful track record with combinatorial optimization problems in multiple fields of application, and has since been extended to handle other types of problems, including data mining problems [38].

In this dissertation I introduce ADR-Miner: an ant-inspired algorithm designed to do data reduction. ADR-Miner adapts ACO to perform data reduction via instance selection with the sole purpose of improving classifier effectiveness. The objective of ADR-Miner is to process a given data set and arrive at the minimum set of instances to use that produces the model with the highest classifier effectiveness. The ADR-Miner algorithm is introduced here in two versions: a base version and an extended one. The base version of ADR-Miner will use a single classification algorithm during both its training and testing phases. The extended version on the other hand will use two separate classification algorithms, one per each phase of the algorithm.

The base version of ADR-Miner will be evaluated using three well known classification algorithms and against 20 data sets from the well-known UCI Machine-Learning repository. The results obtained will be benchmarked against the performance of another data reduction algorithm: the Iterative Case Filtering (ICF) algorithm [11]. Statistical significance in the results will be established using the non-parametric Wilcoxon Signed-Rank test. The extended version of ADR-Miner will be evaluated using all the possible pairings between five classification algorithms and against 37 data sets from the UCI repository. The results obtained are then benchmarked against the performance of the five classification algorithms without data reduction.

The rest of this dissertation is structured as follows. Chapter 2 provides a brief introduction to ACO, Data Reduction and Classification. Chapter 3 provides a review the previous work done with ACO and Data Reduction. Chapter 4 presents an in depth explanation of ADR-Miner and its components. Chapter 5 covers the technical details of implementing the ADR-Miner algorithm. Chapter 6 covers the experimental approach used to evaluate ADR-Miner, with the results obtained being interpreted in Chapter 7. Finally, I round things off in Chapter 8 with my conclusions and suggested avenues of further research.

Chapter 2

Background

In this chapter, we go through the concepts of ACO, classification and data reduction. These form the core concepts whose intersection is the main impetus behind the development of ADR-Miner.

2.1 Ant Colony Optimization

Ant colony optimization (ACO) is a search meta-heuristic that is designed to tackle combinatorial optimization problems. First described by Dorigo et al in 1992 [4], ACO is inspired by the behavior observed in multiple species of ants as they forage for food. As the ants forage for food, the scouts would first wander around the opening of the nest looking for a viable food source. As the scouts locate one, they would carry food back to the nest. Over time, it is observed that the ants out searching for food would converge on the shortest path between the nest and the *closest* food source. This behavior is not explicitly wired into the psyche of the individual ants. Instead, the ants only follow a simple set of rules and the resultant property of the ants following

these rules is that the colony displays the capacity of finding the shortest path between it and the closest food source. An individual ant would first start out at the nest, and wander randomly around it favoring paths that have the scent of food. As soon as an ant locates a source of food, it would carry a portion of the food back to the nest. While the ant travels back and forth between the nest and food source, it deposits a chemical, known as a pheromone, along its trail. New ants setting out from the nest would then probabilistically favor paths that contain both scents of pheromone and food on them versus those which do not. As time passes, the pheromone evaporates. Longer paths to the food would have a larger percentage of the pheromone deposited on them evaporate versus shorter paths due to the amount of time an ant takes to traverse this course. As more and more ants find the shortest of the paths favorable, more ants would take this path, all the while depositing pheromone on it further reinforcing its selection in the future. Eventually, all the ants that set out to search and obtain food adopt the shortest path due its highest concentration of pheromone and abandon all the other paths. This emergent behavior of being able to converge on the shortest path between the nest and the food source is what drove Dorigo et al to adapt it into a search meta-heuristic that can be used to solve combinatorial optimization problems.

Over the next few subsections, well will cover how the behavior observed in real life ants was adapted into the ACO search meta-heuristic and some of the most common ACO algorithms in chronological order of development.

2.1.1 The Ant Colony Optimization Meta-heuristic

Before describing the ACO meta-heuristic, let us first deconstruct it into a set of constituent components:

- A construction graph: This is a representation of the search space that the ants would course in search of a solution. The construction graph is basically a translation of the problem being solved into a form that can be used by the ants. Composed of a set of vertices and edges connecting them, the construction graph's vertices represent components that when combined with other vertices in the graph would form a valid solution to the problem being solved. For example, if we are trying to solve the Traveling Salesman Problem (TSP), a given instance of the problem would be translated into a construction graph where the nodes or cities in the instances are directly translated to vertices and the roads connecting them are translated in the edges connecting them. An ant could then start at a city, and construct a valid solution by continuously selecting the next city/vertex to visit till it returns to its starting point. The vertices in a construction graph are also known as decision components, since the ants have to decide to either include or exclude it by choosing whether to visit it or not.

- A fitness function: This is a function that is used to gauge how well a proposed solution performs in the context of the problem being solved. This is problem specific, and would vary depending on the problem being tackled, whether it is a minimization or maximization type problem. The fitness function is used by the meta-heuristic to determine the fitness of the solutions presented by the ants, and is employed

to determine the best solution encountered throughout the run of the meta-heuristic (and thus return it as the solution to the problem being solved) and is sometimes employed during the pheromone update procedure as elaborated further below. Using TSP as an example, a valid fitness function would be one over the length of the route proposed.

- A colony of ants: This is a collection of agents that share no means of direct communication with each other and that follow a simple set of rules. These rules dictate the behavior of these agents as they course the construction graph in search of valid solutions.

- A transition probability function: This a function that calculates the probability for a given decision component on the construction graph for selection from another node on the same graph. This is extensively used in the *Construct_Solution()* sub-procedure in the meta-heuristic as will be explained shortly.

- A pheromone update scheme: This describes how the pheromone trails are updated by the ants as they traverse the construction graph, and how the pheromone evaporates over time.

The overall meta-heuristic shared by most ACO algorithm can be glimpsed in Algorithm 1. The meta-heuristic begins by initializing its parameters and the colony. Depending on the implementation of the meta-heuristic, this can include setting the total number of iterations to be performed, the number of ants in the colony, the number of iterations after which the meta-heuristic is considered to have converged among other possible parameters. The meta-heuristic then enters its main loop (lines 5-9), that only exits when a pre-

defined set of stopping criteria are satisfied. Again, these stopping criteria will depend on the particular implementation of the meta-heuristic, and can range from simple criteria such as exhausting a preset number of iterations to more complex schemes such as detecting stagnation after a number of restarts. Within the main loop, the meta-heuristic goes through three steps: having the ants in the colony construct solutions, applying local search and updating the pheromone trails.

The first step within the main loop is for each ant within the colony to construct a solution (line 6). This is further expanded in the *Construct_Solutions()* sub-procedure (lines 12-22). To construct a solution, an ant would first start at a node within the construction graph. The beginning node code be a random one or a specific one depending on the problem being solved. From there, the ant would continuously check all feasible neighbors, select the next component from the feasible neighbors and traverse to selected component. Once reaching a node with no feasible neighbors, then the ant has basically traversed the construction graph and built a solution. At this point it returns the route it has taken as the solution proposed by that particular ant. Since the selection of components is probabilistic, before a ant selects a node, it will first calculate the probabilities for each node in the feasible neighborhood using the transition probability function and then stochastically select one with a bias towards those with a higher probability. How the ants calculate the probability for each node in their feasible neighbors and then proceeds to select one is governed by the implementation's transition probability function and therefore will differ from one implementation to the other. The transition probability function is a function of two

values: heuristic information available locally to the ants (η) and the amount of pheromone available at each node (τ). Going with our example of TSP, one possible value of η could be 1 divided by the distance from each node to the next, giving favor of selection towards node that are closer by.

The second step within the main loop of the meta-heuristic is the application of local search. Here, an attempt is made to improve the fitness of the solutions returned by the ants via the application of a local search algorithm. The algorithm would search in the immediate vicinity of the solution for one with a higher fitness. If such a solution is found, then it would replace the solution presented by the ant, otherwise no change is made. This is an optional step, and depending on implementation, it may or may not be applied.

The third and final step within the meta-heuristic's main loop is the process of updating the pheromone trails. This is done over two phases. First, the ants within the colony would deposit pheromone on the routes they have selected throughout the construction graph. Which ants get to deposit pheromones and how much pheromone is deposited depends on the particular implementation of the meta-heuristic. Generally speaking, the amount of pheromone deposited is usually tied to the fitness of the solutions presented by ants, whether in terms of defining which ants get to deposit pheromones on their routes, or the amount of pheromone deposited per trail. To gauge the fitness of a solution, the meta-heuristic uses the fitness function that is deployed along with the implementation and would therefore differ depending on the type of the problem being solved. Second, the pheromone trails are allowed to evaporate, and as with the process of depositing pheromones is governed by the implementation. Both the deposit and evaporate mechanics

Algorithm 1 General ACO Meta-heuristic

```
1: procedure ACO_Metaheuristic()
2:     set parameters
3:     initialize colony
4:     initialize pheromone trails to τ_θ
5:     repeat
6:         Construct_Solutions()
7:         Apply_Local_Search()                    ▷ This step is optional.
8:         Update_Pheromone_Trails()
9:     until stopping criteria are satisfied
10:    return best solution encountered
11: end procedure
12: procedure Construct_Solutions()
13:    for each ant in colony do
14:        x_ant ← x_ant ∪ Initial_Component
15:        while valid nodes still exist for ant in construction graph do
16:            𝒩 ← get feasible neighbors for ant
17:            𝒫 ← Get probabilities for each node in 𝒩
18:            d ← Select_Component_Probabilistically(𝒩, 𝒫)
19:            x_ant ← x_ant ∪ d
20:        end while
21:    end for
22: end procedure
```

of the meta-heuristic form part of the pheromone update scheme of the meta-heuristic.

Once the main loop exits, the meta-heuristic returns the best solution encountered. This is the general meta-heuristic that is adhered to by most implementations, and the implementations discussed next are some such examples. For a deeper treatment of the meta-heuristic, the reader is referred to [17].

2.1.2 ACO Algorithms

The number of ACO algorithms has been growing steadily since their introduction in the early 1990's, with extensions and applications emerging for various optimization and data mining problems. In the following subsections, we cover the following algorithms: Ant System, the first and quintessential ACO algorithm on which meta-heuristic is based; and $MAX - MIN$ Ant System, an extension on which ADR-Miner is based; Ant Colony System; AS-Rank and Antabu. For a broader coverage of the ACO algorithms, the reader is referred to [15]–[17], [32].

Ant System

The first of the ACO algorithms, Ant System was developed in 1992 [4]. Ant System follows the basic structure of the meta-heuristic described earlier and is in fact what lead to its development. In Ant System, the transition probability function to move from node i to node j at time t is defined by

the following equation:

$$p_{ij}^{k}(t) = \begin{cases} \frac{\tau_{ij}^{\alpha}(t)\eta_{ij}^{\beta}(t)}{\sum_{u \in \mathcal{N}_i^k(t)} \tau_{ij}^{\alpha}(t)\eta_{ij}^{\beta}(t)} & if\ j \in \mathcal{N}_i^k(t) \\ 0 & if\ j \notin \mathcal{N}_i^k(t) \end{cases} \tag{2.1}$$

where

- τ_{ij} represents the amount of pheromone deposited on the trail linking node i to node j
- η_{ij} represents the a priori effectiveness of moving from node i to node j
- α and β represent bias towards pheromone and heuristic information respectively
- $\mathcal{N}$ represents the feasible neighborhood for node i at time t

An ant at node i would first get its feasible neighborhood $\mathcal{N}$. The feasible simply represents the nodes the ant can visit next without violating any problem specific constraints. For each node j in the neighborhood, the ant would calculate the probability p_{ij} and the use Roulette Wheel Selection to pick a node j from the neighborhood to move to.

The pheromone evaporation and update procedure for Ant System are performed using the following formulas:

$$\tau_{ij}(t) \leftarrow (1-\rho)\tau_{ij} \tag{2.2}$$

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \triangle\tau_{ij}(t) \tag{2.3}$$

with

$$\triangle\tau_{ij}(t) = \sum_{k=1}^{n_k} \triangle\tau_{ij}^{k}(t) \tag{2.4}$$

where

$$\triangle\tau_{ij}(t) = \begin{cases} \frac{Q}{f(x^k(t))} & if\ link(i,j) \in x^k(t) \\ 0 & otherwise \end{cases} \tag{2.5}$$

Here, ρ represents an evaporation factor and ranges between 0 and 1. After evaporation has been applied, the ant solutions are inspected and for each link in the graph:

- Get all ants that have link(i, j) as part of their solution $(x^k(t))$.
- For each ant in the cohort that include link(i, j) in their solutions, deposit pheromone in proportion to the quality of the solutions attained (Equation 2.5). Q is a constant that is set by the user, and Equation 2.5 assumes a minimization problem. For a maximization problem, $Qf(x^k(t))$ is used instead.

Ant System was first employed by Dorigo et al to solve the Traveling Salesman and the Quadratic Assignment problems, and the algorithm has shown to be successful. The algorithm was later extended, for example by allowing a number of elite ants to deposit an extra amount of pheromones, and was the basis of the ACO meta-heuristic as we know it.

$MAX - MIN$ Ant System

The $MAX - MIN$ Ant System is an extension of the Ant System algorithm, and was designed to tackle a specific issue: stagnation. During experimentation with Ant System, it was observed the ants would sometimes converge early on a path in the construction graph that might not be optimal and get stuck on a local minima. To remedy this situation, the original Ant System was altered in the following manner:

- Only one ant is allowed to deposit pheromone: The current iteration best or the global best.
- The pheromone levels on the links in the graph are bounded to be between τ_{min} and τ_{max}.
- Upon initialization, pheromone levels are set to τ_{max}. This is in contrast with the Ant System algorithm where the pheromone levels where initialized to zero or a small random figure.
- If and when the algorithm reaches stagnation, the pheromone levels are reset to τ_{max}.

There are a number of ways to set the values for τ_{min} and τ_{max}. One way is to set the values for the bounds as static values at the beginning of the algorithm run. Another way, would be to use the following equations based on the best estimation available of the optimal solution:

$$\tau_{max}(t) = (\frac{1}{1-\rho})\frac{1}{f(\hat{x}(t))} \tag{2.6}$$

$$\tau_{min}(t) = \frac{\tau_{max}(t)(1-\sqrt{\hat{p}}n_G)}{(n_G/2-1)\sqrt{\hat{p}}n_G} \tag{2.7}$$

where

- $f(\hat{x}(t))$: the fitness of the best solution encountered so far. This could be the iteration best or the global best.
- $\hat{p}$: the probability of finding the best solution. This is a user set parameter and $\hat{p}$ must be less than 1 so that τ_{min} could take on a value greater than zero.
- n_G: the total number of nodes in the graph.

Note that since τ_{max} depends on the best solution found so far, this makes it time dependent. This results in both bounds for τ being dynamic and adjust throughout the run of the algorithm.

Ant Colony System

Ant colony system (ACS) is an extension of Ant System that was developed by Dorigo and Gambardella in an attempt to improve performance and achieve a better balance between exploration and exploitation. ACS differs from Ant System on three aspects: the transition probability function, pheromone maintenance and the introduction of the use of candidate lists.

ACS uses Equation 2.8 as its transition probability function. r_0 is a user paremeter set between 0 and 1, while r is a random number generated such that $r \in U(0,1)$. Using this scheme, an ant will occasionally favor making a greedy decision to select the node with the highest pheromone and heuristic values within its feasible neighborhood. At times when r is greater than r_0, then ant would select its next step based on Equation 2.1, but with α preset to 1.

$$P(i,j) = \begin{cases} arg\,max_{u \in \mathcal{N}_j^k(t)}\{\tau_{iu}(t)\eta_{iu}^{\beta}(t)\} & if\ r \leq r_0 \\ J & otherwise \end{cases} \tag{2.8}$$

The pheromone trails in ACS are only updated by the best ant, and as in MMAS, this could either be the iteration best or the global best. To hasten evaporation, and promote exploration within the colony, a little amount of pheromone is removed every time an ant visits a node within the graph. Also, as in MMAS, the pheromone levels are bounded between a minimum and a

maximum level to stave off early convergence on local minima.

Candidate lists are preferred nodes to be visited next given an ants position, and influence how an ant transitions from one node to the next within the graph. Given a choice to select a node, an ant would first inspect the candidate list for choices and make a selection there, before looking into the rest of the feasible neighborhood. Members in the candidate list are usually sorted from highest utility to least, and a candidate list is always less than feasible neighborhood in terms of size.

AS-Rank

AS-Rank is a relatively straight forward extension to Ant System, with modifications introduced to how pheromones are updated, using elitist ants and only allowing the best ant to deposit an extra amount of pheromone. The pheromone update rule in AS-Rank is modified into the following:

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + n_e \triangle \hat{\tau}_{ij}(t) + \triangle \tau^r_{ij}(t) \tag{2.9}$$

where

$$\triangle \hat{\tau}_{ij}(t) = \frac{Q}{f(\hat{x}(t))} \tag{2.10}$$

$$\triangle \tau^r_{ij}(t) = \sum_{\sigma=1}^{n_e} \triangle \tau^\sigma_{ij}(t) \tag{2.11}$$

$$\triangle \tau^\sigma_{ij}(t) = \begin{cases} \frac{(n_e-\sigma)Q}{f(x^\sigma(t))} & if\,(i,j) \in x^\sigma(t) \\ 0 & otherwise \end{cases} \tag{2.12}$$

Here, σ represents rank is assigned from 1 to n to the ants from the fittest to the least fit. n_e represents the number of elite ants used. By using rank, the elite ants get to deposit pheromone in proportion to their fitness.

Antabu

Kaji et al have introduced a version of AS that use Tabu search during the local search step, as well as factor in the fitness of the current path, best solution and worst solution encoutnered so for into the process of updateing the pheromone trails. The pheromone update function now becomes:

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + (\frac{\rho}{f(x^k(t))})(\frac{f(x^-(t)) - f(x^k(t))}{f(\hat{x}(t))}) \tag{2.13}$$

where x^k, $x^-(t)$ and $\hat{x}(t)$ stand for the fitness of the current ant's solution, the best solution encountered so far and the worst solution encountered so far respectively.

2.1.3 Convergence

Is the ACO meta-heuristic guaranteed to arrive at a solution? The question of convergence is usually the first theoretical question that asserts itself whenever a new meta-heuristic is introduced. When speaking of convergence in relation to a stochastic meta-heuristic designed to tackle combinatorial optimization, we usually consider two types of convergence: value convergence and solution convergence. Value convergence is concerned with quantifying the probability that the meta-heuristic will arrive at an optimal solution at least once throughout its lifetime. Solution convergence is concerned with quantifying the probability that the meta-heuristic will keep arriving at the

same optimal solution over and over again. As the ACO meta-heuristic is very broad in its definition, it renders itself difficult for this type of analysis. Instead of trying to arrive at a convergence proof for the overall meta-heuristic, researchers have instead opted to find proofs for specific models based on the ACO meta-heuristic, and we discuss one such proof over the next few paragraphs. As for the aspect of *speed* of convergence, no current measures provide an estimation for the various ACO algorithms, and one would have to resort to empirical evaluation and extensive testing to arrive at such measures.

Among the first proofs of convergence provided for an ACO algorithms is the one provided by Stützle and Dorigo for the $MIN - MAX$ Ant System (MMAS) [14]. The proof is developed under the following assumptions on the MMAS algorithm:

- $\tau_{min} < f(\pi^*)$, where π^* represents the optimal solution.
- $\beta = 0$, i.e. there is no dependence on heuristic information while constructing the solutions.
- Only the global best is allowed to deposit pheromone on its trail.
- The amount of pheromone deposited is proporional to the quality of the solution, in this case π^*.

With these stipulations in place, Stützle and Dorigo were able to prove the following:

- The probability of finding the optimal solution at least once is $P^* \geq 1-\epsilon$ and that this probability approaches one given a large enough number

of iterations and a relatively small non-zero error rate ϵ

$$\lim_{t \to \infty} P^*(t) = 1 \tag{2.14}$$

- Once an optimal solution is found, the algorithm will only need a limited number of steps before the trail associated with the optimal solution will have pheromone levels that are higher than any other possible trail. From that point forward, the probability of any ant of finding this optimal solution will be $P^* \geq 1 - \hat{e}(\tau_{min}, \tau_{max})$.

By providing a schedule that slowly decreases τ_{min} to 0, all ants in the colony will build the optimal solution τ^* once it is found. This in essence provides a solution convergence proof that complements the value convergence proof just presented. With slight adjustments, these proofs hold when we reintroduce the reliance on heuristic information while building solutions as well providing similar proofs to the Ant Colony System (ACS) algorithm. This, of course, is a brief discussion of these proofs. For a more fuller discussion that includes how these proofs were arrived at, the reader is directed to [14], [31]

2.2 Data Reduction

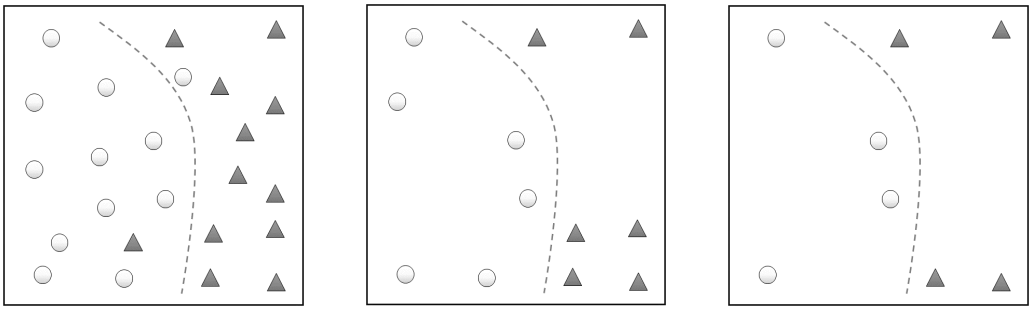
As mentioned earlier, data reduction is a vital preprocessing task for machine learning and data mining schemes, and its significance lies in that it removes noisy, outlier and other data from the training data set that can be detrimental or misleading to the algorithm learning a model. In addition to improving accuracy, it also reduces the size of the training set before it is presented to the machine learning algorithm. The use of a reduced training set results in shorter training times for *eager-learning* classification algorithm,

such as induction decision trees, classification rules and probabilistic models. For *lazy-learning* algorithms, such as nearest neighbor(s), the reduced data set decreases the time needed for arriving at the class of a new instance in question. In addition, a smaller data set would require less resources for storage and maintenance.

Data reduction employs a number of techniques, including instance selection, feature selection, replacing the data set with a representative model and encoding the data set to arrive at a "compressed" representation of the data set amongst others[9]. Since an in depth treatment of the various data reduction techniques is beyond the scope of this proposal, I will choose to focus instead on both feature and instance selection, the latter of which has a specific significance to my proposal.

Feature selection is the process of selecting a subset of the attributes associated with the data for usage and ignoring the rest. The objective here is to arrive at the minimum set of attributes that can be used without upsetting the class distribution in the original data set, or be close to it as possible. Attributes that considered for exclusion are usually those which are redundant or whose values are random in relation to the data classes and as such will not contribute any statistically significant information to the machine learning or data mining model being built. Using a reduced attribute set to build such models also has the added benefit that the models are more comprehensible to human operators versus employing the entire set of attributes.

Instance selection is analogous to attribute selection, but instead of finding the minimum set of attributes it attempts to find the minimum set of



(a) Original set. (b) Interim set. (c) Final set.

Figure 2.1: An example of instance selection performed on a data set with two classes.

instances within the data set that can be used. The focus with instance selection is to rid the data set of outlier (which are usually detrimental to the quality of the model being built), superfluous and noisy instances. Instance selection can either start with an empty set and continuously add instances from the data set till it satisfies the objective of finding the minimum representative set, or start with the full set of instances in the data set and whittle it down to the most representative instances. Another technique that can be deployed is to replace a cluster of instances with a representative (sometimes also known as a prototype instance), which may or may not be a natural instance from the original data set. An example of a how an instance selection might proceed to achieve data reduction can be seen in Figure 2.1. Research in instance selection stretches as far back as the 1970's with the advent of Wilson editing, and a survey of such algorithms is presented in the next chapter.

2.3 Classification

Classification is a data mining and machine learning field that focuses on building models that are capable of predicting the class of a given case. In particular, classifier models attempt to predict the categorical label or class of the case being studied. Given a set of labeled instances, a classification algorithm will try to capture the mapping between the attributes of the instances within the set and their respective class. Once this mapping has been captured it is used to build a model, which can take the form of a set of rules; a decision tree or a statistical model among others, which in turn is then used to predict the class of unforeseen instances. The set of instances used to build the model is usually known as a training set. Since the training set is labeled, i.e. the class of each instance is known beforehand, classification is considered to be a form of supervised learning. This is in contrast to other data mining schemes that build models from instances where no known class is present a priori, such as clustering. The effectiveness of a classifier is evaluated using a data set different from the training set, known as the testing test, and the performance of the classifier can be gauged using a number of metrics. The most common metric used for gauging classifier performance is accuracy, and it is simply the number of instances from the testing set that were classified correctly out of the total number of instances in the testing set. Over the next few subsections, we will examine three of the most common classification algorithms. For a broader coverage of classifiers, the reader is referred to [9], [19], [34].

2.3.1 k-Nearest Neighbor

First developed in the 1950's, the k-Nearest Neighbor classifier is considered to be of the earliest classification algorithms. Based on learning by comparison, a k-Nearest Neighbor Classifier will attempt to classify a new instance based on the nearest representative(s) in a set of maintained instances. Each instance of the maintained set will have n attributes and a class label. Given a new instance to classify, the classifier will go over its set of maintained instances, also known as the training set, and find the k-nearest neighbors. The proximity between instances is defined by a distance function which measures the distance between two instances by aggregating the differences between each of the n attributes for the two instances being compared. Once located, the k-nearest neighbors then vote on the class of the new instance by counting the number of instances per class. The class with the majority count then gets assigned to the new instance.

$$dist(\mathcal{X}_1, \mathcal{X}_2) = \sqrt{\sum_{i=1}^{n} (x_{1i} - x_{2i})^2} \tag{2.15}$$

An example of a distance function that is commonly used with the k-Nearest Neighbor classifier is the Euclidean distance, seen in Equation 2.15. Handling numerical attributes is straight forward, as the difference for each positional pair of numerical attributes is measured, squared, aggregated and the square root of the sum is returned. Before applying the distance measure, numerical attributes are usually normalized so as not to allow the numerical range of the attribute to skew the measure. For categorical attributes, the difference is considered to be one if their values do not match, and zero otherwise. The rest of the equation would apply without modification.

In the original version of the classifier, the distance measure is applied to the attributes of the instance without weighing. This made the classifier susceptible to noisy or irrelevant attributes. To remedy this, extensions were introduced that allowed the weighing of the attributes within the distance function, as well as the cleansing of noisy data.

2.3.2 C4.5

Developed by J. Ross Quinlan [5] as an extension to an earlier algorithm, the C4.5 is a greedy tree induction algorithm that recursively divides the training set into smaller and smaller sections, where each section is dominated by a particular class. Moving down the tree, each node leads to its children based on value ranges of one particular attribute. The general workings of a typical decision tree induction algorithm can be seen in Algorithm 2.

The algorithm starts by creating the root node, which starts with the entire training set of tuples and a list of attributes. It then checks to see if all the tuples belong to the same class or if the attributes list is empty. If that is the case then the node is assigned the majority class within the tuples and is returned. Next, we get the splitting criteria through the use of an attribute selection function. The attribute selection function inspects a collection of tuples, along with a list of attributes, and decides on the best attribute to split on along with the value ranges to use per branch. We then update the attribute list based on the attribute that was selected for the splitting. For each of the branches included in the splitting criteria we either add a leaf node containing the majority class of the current node if the branch being considered has no tuples or add an entire sub-tree based on the remaining

Algorithm 2 Decision Tree Induction Algorithm

```
1: procedure GetTree(D, attributes_list)
2:     N ← new node
3:     if attributes_list = φ or class(∀x ∈ D) = C then
4:         N.Class ← GetMajorityClass(D)
5:         return N
6:     end if
7:     split_criterion ← Attribute_Selection(D, attributes_list)
8:     update attributes_list
9:     for j ∈ split_criterion do
10:        D_j ← Split(D, j)
11:        if D_j is empty then
12:            leaf ← GetMajorityClass(D)
13:            N.Children ← N.Children ∪ leaf
14:        else
15:            N.Children ← N.Children ∪ GetTree(D_j, attributes_list)
16:        end if
17:    end for
18:    return N
19: end procedure
```

attributes and tuples associated with the branch being considered.

As mentioned earlier, the attribute selection function decides on the next attribute to split on. The first function used by Quinlan is known as Information Gain, and is defined as:

$$Gain(A) = Info(D) - Info_A(D) \tag{2.16}$$

where

$$Info(D) = -\sum_{i=1}^{m} p_i log_2(p_i) \tag{2.17}$$

$$Info_A(D) \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j) \tag{2.18}$$

Equation 2.16 measures the gain in purity if an attribute A is used for the split. Purity in this sense is attempting to have a set where all the tuples share the same class, or have this set as homogeneous as possible. Equation 2.16 measures this gain by seeing how entropy in the original set D is reduced by splitting on attribute A. Equation 2.17 attempts to measure the amount of information required to classify a tuple in D encoded as bits (thus the use of log_2 in the function). This information is measured by averaging the probabilities of an arbitrary tuple belonging to a class C_i. Once the entropy associated with the original tuple set D, we then test the amount of entropy that would removed by an attribute A using Equation 2.18. This function does the same as the previous one but on a smaller scale: on the scale of the tuple set that would result if we split on a value of the attribute at hand. To split using Information Gain, we simply select the attribute that returns the maximum gain. (To see Information Gain in action, the reader is referred to [9])

Using information gain had an issue that it was biased towards selecting the attributes that had a large number of unique values such as a row identifier which would not have value in classification. To remedy this, Quinlan introduced Gain Ratio, a successor of Information Gain and is the basis of the C4.5 extension.

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo(A)} \tag{2.19}$$

where

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times log_2(\frac{|D_j|}{|D|}) \tag{2.20}$$

Note that Equation 2.20 takes into account the number of tuples in the resultant set post split. By doing so, Gain Ratio favors those splits that result in larger sets versus those which result in smaller sets, thus alleviating the bias towards attributes with a large number of unique values.

2.3.3 Naive Bayes

The Naive Bayes classifier is a statistical classifer based on Bayes' Theorem. Although a relatively simple classifier, the Naive Bayes classifier has shown to be competitive with some of the more sophisticated classifiers such as decision tree induction and some forms of neural network classifiers. Over the next few paragraphs, we will look into the basics of Bayes Theorem and how it was used to form the classifier.

Bayes Theorem

Bayes Theorem was developed in the 18th century by one Thomas Bayes, a pioneer in the works of probability and decision theory. Let $\mathcal{X}$ be a tuple of

data described by n attributes. In the context of the theorem, $\mathcal{X}$ would be considered "evidence" of a fact. Let $\mathcal{H}$ be a hypothesis. In the context of classification, such a hypothesis could be that $\mathcal{X}$ belongs to a specific class $\mathcal{C}$. In order to arrive at a classification, our aim would be to find out $\mathcal{P}(\mathcal{H}|\mathcal{X})$, i.e. that probability of $\mathcal{X}$ belonging to a specific class $\mathcal{C}$ given the values of attributes in $\mathcal{X}$.

$\mathcal{P}(\mathcal{H}|\mathcal{X})$ is known as a posteriori probability of $\mathcal{H}$ being conditioned on $\mathcal{X}$. Suppose for example that $\mathcal{X}$ represents a loan application with two attributes: applicant employment status and applicant yearly income. Now suppose that the values of these attributes were *"Employed"* and *"*$\geq 100,000$*"*. For our hypothesis, suppose that it would be whether the loan application is rejected or approved. $\mathcal{P}(\mathcal{H}|\mathcal{X})$ in this case would be whether a loan application is approved or rejected knowing that the applicant's employment status is *"Employed"* and that his yearly income is *"*$\geq 100,000$*"*.

$\mathcal{P}(\mathcal{H})$ or $\mathcal{P}(\mathcal{X})$ are known as a priori probability since their measurement is unconditional on the presence of other facts. This probability can be easily calculated using basic information that is supplied from the data set being analyzed.

Since our focus here is classificaiton, our aim is to calculate $\mathcal{P}(\mathcal{H}|\mathcal{X})$ for the various classes present with the $\mathcal{X}$ at hand. Bayes' Theorem allows us to estimate this probability using Equation 2.21.

$$\mathcal{P}(\mathcal{H}|\mathcal{X}) = \frac{\mathcal{P}(\mathcal{X}|\mathcal{H})\mathcal{P}(\mathcal{H})}{\mathcal{P}(\mathcal{X})} \tag{2.21}$$

Classification Algorithm

Now that we have seen Bayes' Theorem, let us see how it used to classify tuples or instances.

1. Let $\mathcal{T}$ be a set of training data. Each tuple $\mathcal{X}$ in the training data is made up of n attributes.

2. Let there be m classes in $\mathcal{T}$, $\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_m$. The aim here would be to get the maximize $\mathcal{P}(\mathcal{C}_i|\mathcal{X})$. This can be calculated using Equation 2.21

3. Since the denominator in the Bayes equation ($\mathcal{P}(\mathcal{X})$) is constant, we only need to worry about maximizing the numerator, i.e. $\mathcal{P}(\mathcal{X}|\mathcal{C}_i)\mathcal{P}(\mathcal{C}_i)$.

4. Using the assumption of conditional independence, we can estimate $\mathcal{P}(\mathcal{X}|\mathcal{C}_i)$ with
$$\mathcal{P}(\mathcal{X}|\mathcal{C}_i) = \prod_{k=1}^{n} \mathcal{P}(x_k|\mathcal{C}_i) \tag{2.22}$$
where k represents the positional index of the attributes in $\mathcal{X}$. If the attribute being handled is categorical, then the estimation of probability is a simple matter of counting occurrences and dividing by the total number of instances in $\mathcal{C}_i$. If the attribute is numerical however, then we can safely assume that it follows a Gaussian distribution within the data and the process of calculating its probability would involve calculating the mean and variance for that attribute and then using the Gaussian distribution function to estimate the probability.

5. After calculating the probabilities for a given $\mathcal{X}$ to belong to each of the classes, we return the highest probability as the class label.

Chapter 3

Related Work

In this chapter, we will look into previous work done with ACO and Data Reduction that is relevant to the research proposed in this article.

3.1 Ant Colony Optimization

Classically applied to combinatorial optimization problems, ACO has been extended to tackle a number of other classes of problems, including ordering problems, assignment problems, subset problems and grouping problems. Since the research introduced in this thesis is related to classification, I will focus on some of the applications of ACO in that field and would direct the reader to [17] for a survey of ACO's application in the aforementioned ones.

Ant-Miner [13] is the first ant-based classification algorithm, which aims to build a list of classification rules from a training data set that can successfully be used to classify instances in an unforeseen data set. A classification rule takes the form of:

IF $<$ *term1* AND *term2* AND ... $>$ THEN $<$ *class* $>$

where term refers to an assignment of a value to one attribute from the data set. Before the ants can start building these rules, the search space must first be converted into a construction graph. The nodes in the construction graph in Ant-Miner are all the possible terms that can be extracted from the data set, i.e all the possible value assignments that can be performed to the attributes in the data set. These nodes are fully connected to each other. To construct a solution, an ant will select one term from the set of nodes pertaining to one particular attribute before moving on to the next. Once the ant has selected a term per each attribute in the data set, then it has completed building the antecedent part of the classification rule. To get the consequent, the ant will get the majority vote for class on all the instances that this rule applies to. After the rule is constructed, it will go through a pruning phase that considers the impact of removing any of the terms in the antecedent on the rule's quality. If a term's absence from the antecedent does not negatively impact the rule's quality, then it is removed. The ants in Ant-Miner iteratively try to arrive at new rules, and only the best rule per iteration is kept. A rule's quality is judged based on its classification effectiveness over the instances that it covers. Once a rule is accepted, the instances covered by that rule are removed from the data set and the ants will try to arrive at new rules from the remaining instances till the number of uncovered instances falls under a prespecified threshold. As with any other ACO-based algorithm, the ants are biased to selecting one node over the other based on the heuristic information and pheromone levels associated with that node. The heuristic information used here is information entropy [5]. Information entropy attempts to measure the cost of describing the

information provided by a given attribute assignment or term. Terms with a high level of entropy, thus costing more to describe, are disliked and the ants would be biased against selecting them. As for pheromone updates, only the nodes associated with the last added rule have their pheromones incremented, while all other nodes have their pheromone levels decreased based on a predetermined evaporation schedule.

Since its introduction, a lot of extensions have been proposed for Ant-Miner. These extensions either aimed to address limitations in the original algorithm or improved the quality and precision of the rules that were produced by it, and include the following examples. The cAnt-Miner [26], [28] extension allows Ant-Miner to handle attributes with continuous values, a capability that was missing from the original design. The use of multiple pheromone types [33], [39], [41], one per each class in the data set being processed, resulted in improvements in the quality of the rules produced as well as their simplicity. Freitas and Chan [20] have introduced an improved rule pruning procedure that resulted in shorter rules, while improving the computational time used while handing large data sets. Liu et al.[12] proposed an improved function for calculating the heuristic value of the terms in the construction graph, thus improving the quality of the rules produced by the algorithm.

Boryczka and Kozak [29], [35] were able to use ACO to build classifiers based on decision trees. The authors use the splitting rule from CART (Classification and Regression Trees) to build a heuristic function which they use in their new algorithm, dubbed ACDT. This function assigns higher heuristic value to attributes that result in splits with the highest degree of

homogeneity in the descendant nodes. ACDT was compared with CART and Ant-Miner and has shown to produce favorable results.

Salama et al. [42], [43], [45] have adapted the ACO meta-heuristic to build Bayesian Network based classifier. The ABC-Miner algorithm attempts to build a Bayesian Augmented Naive-Bayes (BAN) classifier for the entire data set. Empirical results show that ABC-Miner performed competitively to state of the art algorithms that produced a similar classifier. Another approach for building for building Bayesian Network based classifiers is to build one network per class in the data set, and was also explored by the authors of ABC-Miner. This approach was tested against the original ABC-Miner algorithm, as well as start of the art algorithms, and has shown to produce superior classification results in most cases.

More recently, ACO has been applied to the field of neural network based classifiers. Liu et al. [22] introduced a hybrid ACO algorithm that uses BP to optimize the weights on links between neurons in a feed forward neural network. The ACO_R algorithm [27] (which performs continuous optimization) was adapted by Socha and Blum to train neural networks as classifiers [18], [24]. Finally, Abdelbar and Salama introduced ANN-Miner [44], which uses ACO to build multi-layered feed forward neural networks for use as classifiers.

3.2 Data Reduction

One of the earliest algorithms for data reduction is Wilson editing [1], also known as Editing Nearest Neighbor. This algorithm attempts reduction by going through the instances and removing those that are incorrectly classified

by their nearest neighbors, typically considering the three closest neighbors. This has the effect of removing noisy instances that lie within the body of homogeneous zones of a particular class, as well as smoothing the boundaries between zones of different classes. Two known extensions of Wilson editing are Repeated Edited Nearest Neighbor and All k-NN [2]. In RENN, Wilson editing is performed iteratively until no more instances can be removed from the data set. All k-NN is similar to RNN, but increased k (the number of nearest neighbors to consider) with each iteration. Both extensions provided higher accuracies than Wilson editing when paired with instance based classifiers.

The IB family of algorithms are a group of incremental lazy learners introduced by Aha et al in 1991 [3]. Two of these algorithms performed reduction by means of instance selection, and therefore are of interest as they fall in-line with this proposal's main context. With IB2, a new case is added to the set of maintained cases by the classifier if and only if it cannot be correctly classified by the set already maintained. At the end of the algorithm, the set that is used by the classifier then becomes our reduced set. This makes IB2 susceptible to noisy instances, as they will always be misclassified and thus added to the set of maintained cases by the classifier. In an attempt to remedy the problem of noisy instances, IB3 enforces a policy that removes instances from the maintained set if they contribute negatively to the classification. This is done by keeping track of how well instances in the maintained set classify instances in the training set. If that classification has a statistically significant negative record, then the instance that produced that record is removed from the set of maintained instances. IB3 has shown

to have a higher accuracy than IB2 when tested in application and since both IB2 and IB3 algorithms are incremental in nature, they are highly efficient when compared to other reduction algorithms.

Wilson et al introduced the DROP family of reduction algorithms in [10]. DROP 1 performs reduction by considering whether removing an instance would result in a misclassification of its neighbors. If that is not the case, the instance is removed. The algorithms starts with the entire training set, and for each instance in that set, it gets its nearest neighbors. The current instance is added to each of the neighbors associates list, those instances where the current instance can be found as a neighbor. It then considers the following question using each instances and its associates: if the current instance is removed from the neighborhood of each of its associates, would that result in a misclassification, or would the associates be classified correctly without the influence of the current instance? If the removal of the instance at hand results in a misclassification, then it is removed, otherwise it is kept. The algorithm does this iteratively and the data set that remains after this whittling is then the reduced data set. DROP 1 attempts to remove noisy instances, since they will always cause a misclassification amongst their neighbors. DROP 1 also has a bias to remove instances that are in the center of class "zones" as opposed to ones near the boundary, since these instances are superfluous and the membership of any newly encountered instances to the class zone can easily be inferred from the boundary instances. However, in the process of removing instances, DROP 1 may resolve to remove a lot of instances around a noisy instance(s) that lies within the center of a homogeneous region of a class before the noisy instances themselves are

removed. DROP2 was designed with a measure that addresses this issue. DROP 2 differs from DROP 1 in that during iterations, it will consider the effect of leaving a considered instance on the misclassification of deleted instances in pervious iterations as well as those who remain. DROP 2 also adds an ordering to the process of removing instances. At the beginning of each iteration it would sort the instances still being considered for reduction by distance from their respective nearest enemy: an instance of a different class. Instances are then considered for removal starting with those furthest away from their enemies. DROP 3 adds preprocessing stage whereby Wilson editing is performed on the data set before the algorithm proceeds. Since at times this has shown be too aggressive, and in effect filtering out the entire data set, DROP 4 adds a constraint to the preprocessing phase: An instance promoted for removal by Wilson editing is only removed if doing so would not cause a misclassification with its associates. Finally, DROP 5 differs from DROP 2 in that it considers instances for removal in the reverse of the order used in DROP 2. This means that instances that are closer to their enemies are considered before those further away. This has the effect of smoothing the class zone boundaries and removing central points more efficiently than DROP 2.

Iterative Case Filtering, or ICF, is an algorithm devised by Mellish et al [11] that achieves reduction over two phases. The first phase of the algorithm performs regular Wilson editing, with a neighborhood size of 3 typically. The second phase then builds two sets for each instance that still remains: a reachable set and a coverage set. The reachable set are those instances that include the current instance in their neighborhoods. The coverage set are

those instances that have the current instance as a neighbor. In essence, the reachable set are those instances that influence the classification of the current instance, and the coverage set are those whose classification is influenced by the current instance. After having built those sets, the algorithm then removes those instances who have a reachable set larger than that of their coverage set, for those instances are considered superfluous and their removal would not affect the general accuracy of a model built on the remaining instances. The ICF algorithm has proven to be an efficient reduction algorithm when tested against a wide panel of data sets from UCI Machine Learning repository.

Chapter 4

The ADR-Miner Algorithm

As alluded to earlier, the ADR-Miner adapts an ACO algorithm to perform data reduction with an emphasis on improving a classifier's predictive effectiveness. In particular, given a training set, a testing set and a classification algorithm, we want to see if we can achieve more accurate predictions if the classifier is constructed using a reduced set then tested, as opposed to being constructed using the raw training set.

Adapting the ACO algorithm to perform data reduction involves a number of steps:

1. Translating the problem into a search space that is traversable by the ants, also known as a construction graph.

2. Defining the overall meta-heuristic that will be used to direct the ants as they search the problem space.

3. Defining how an ant constructs a candidate solution (i.e., a reduced set) while traversing the construction graph.

4. Defining the mechanics of evaluating the quality of such solutions and updating the pheromone trails.

Over the next few sections, we will delve into the details associated with these steps and in effect describe how we can adapt ACO to perform data reduction via instance selection.

4.1 The Construction Graph

At the core of the ACO algorithm is the construction graph. This is a graph of decision components that when a subset is chosen from by an ant forms a solution to the problem being solved. Our main concern when tackling a problem is how to translate the related search into a graph of decision components with the stipulation that the graph contains subsets of the decision components that form valid solutions to the problem being solved. Take for example the traveling salesman problem, one of the first to be tackled by ACO. In the traveling salesman problem, we are given a set of cities and distances between them and asked to find the shortest route that visits all the cities exactly once and return to the origin city. Translating such a problem into a construction graph would first lead us to building a graph where the nodes represent the cities and vertices between the nodes would represent the distances among them. The nodes form decision components as at each node, an ant would have to "decide" which node to visit next whilst maintaining the process of building a valid solution: not to revisit any cities already visited. From this graph of cities and distances, an ant could start at one node and iteratively decide which city to visit next till it returns to origin. The graph in its current form can thus be considered a construction

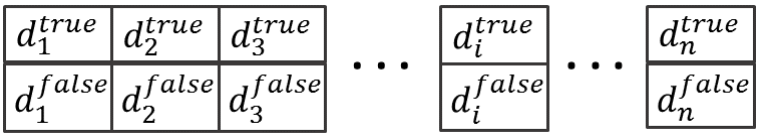


Figure 4.1: The ADR-Miner Construction Graph

graph as it could possibly contain one or more valid routes connecting all the cities. This graph was coined a "construction" graph since the ants traverse it one node after the next, iteratively deciding which node to visit next while "constructing" a solution.

In our case, we are attempting to perform data reduction via instance selection. Starting with a set of instances I, we want to arrive at a subset $R \subseteq I$ that produces the best possible classifier effectiveness. Essentially, we have to decide which instances from I are to be included in R, and that translates into each instance having two decision components within the graph: d_i^{true} whose selection would imply the inclusion of the i-th instance in I, and d_i^{false} whose selection would imply the exclusion of the i-th instance from I. Selection between d_i^{true} and d_i^{false} for the same value of i is mutually exclusive, and an ant constructing a solution cannot include both in its selection.

The decision components of the graph conceptually take the form of a two dimensional array, with a length of $|I|$ and a depth of 2, signifying the choice between inclusion and exclusion. A visual representation of the construction graph can be seen here 4.1. A valid candidate solution can be built from this construction graph by starting at the first node of the array, selecting either the inclusion or exclusion nodes from the second dimension and then moving onto the next node till the entire array is parsed.

4.2 The ADR-Miner Overall Algorithm

The overall procedure for ADR-Miner can be seen in Algorithm 3 We begin by initializing the pheromones on all the decision components to the value of 1 (line 4). This includes both the components for inclusion and exclusion for all instances in the data set to be reduced (i.e., the current `trainingset`). We then enter a $repeat - until$ loop (lines 5 to 21) that is terminated when either of the following criteria are reached: we exhaust `max_iterations` number of iterations, or the colony has converged on a solution and no visible improvement has been observed over `conv_iterations` number of iterations, where `max_iterations` and `conv_iterations` are external parameters.

Within each iteration t of this outer loop, each ant_a in the colony constructs a candidate solution R_a (line 7), that is, a reduced set of instances. After a candidate solution is produced, a classification model M_a is constructed using the reduced set R_a and an input classification algorithm g (line 8). The quality of model M_a is evaluated (line 9), and if it is higher than that achieved by other ants in the current iteration t, it supplants an iteration best solution R_{tbest} (lines 10 to 13).

After all the ants in the colony complete the building of their solutions, the best ant in the iteration is allowed to update the pheromone trails based on R_{tbest}. This complies with the pheromone update strategy of the $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System [6], on which this algorithm is based. The iteration best solution R_{tbest} will supplant the best-so-far solution R_{bsf} if it is better in quality (lines 16 to 20). This process is repeated until the main loop exits, at which point, the best-so-far solution, R_{bsf}, observed over the course of the algorithm is returned as the output reduced set.

Algorithm 3 Pseudo-code of ADR-Miner.

```
1: g ← classification_algorithm
2: I ← training_set
3: InitializePheromones()
4: repeat
5:     for a ← 1 to colony_size do
6:         R_a ← ant_a.CreateSolution(I)
7:         M_a ← ConstructModel(g, R_a)
8:         Q_a ← EvaluateModelQuality(M_a)
9:         if Q_a > Q_tbest then
10:            Q_tbest ← Q_a
11:            R_tbest ← R_a
12:        end if
13:    end for
14:    UpdatePheromones(R_tbest)
15:    if Q_tbest > Q_bsf then
16:        Q_bsf ← Q_best
17:        R_bsf ← R_tbest
18:        t ← t + 1
19:    end if
20: until t = max_iterations or Convergence(conv_iterations)
21: return R_bsf;
```

So far, the ADR-Miner algorithm uses a single classification algorithm g throughout its operation. This algorithm is used during the reduction phase to evaluate interim solution fitness (lines 7 - 8), and is implicitly used to build the final model using R_{bsf} for use in testing. Decoupling these two phases (reduction and testing), and allowing for different classification algorithms to be used during each phase, an extended version of the ADR-Miner algorithm can be seen in Algorithm 4. The first change that can be seen in Algorithm 4 is the explicit initialization of two distinct classification algorithms (g and h), and two data sets: one for training (I) and one for testing (T). The g classification algorithm, along with the training set I, are used during the training or reduction phase of the algorithm (lines 6 - 22) to arrive at a final reduced set R_{bsf}. The final reduced set, R_{bsf}, is then used with the h classification algorithm to build the final model M_{final} (line 23), which in turn can be used with the testing set T to evaluate the algorithm. These extensions allow us to experiment with the effects of using different algorithm during the training and testing phases of the algorithm and noting their impact on classifier effectiveness.

The parameters `max_iterations`, `colony_size`, and `conv_iterations` are set to 1000, 10, and 10 respectively.

4.3 Solution Creation

An overview of the solution construction process can be seen in Algorithm 5. Having mentioned it briefly in the previous section, each ant_a construct a solution by first starting with an empty structure T_a (which represents the ant trail) and incrementally appending decision components d_i^v from the con-

Algorithm 4 Extended ADR-Miner Algorithm

1: $g \leftarrow$ `classification_algorithm_1`
2: $h \leftarrow$ `classification_algorithm_2`
3: $I \leftarrow$ `training_set`;
4: $T \leftarrow$ `testing_set`;
5: $InitializePheromones()$
6: **repeat**
7: **for** $a \leftarrow 1$ to `colony_size` **do**
8: $R_a \leftarrow ant_a.CreateSolution(I)$
9: $M_a \leftarrow ConstructModel(g, R_a)$
10: $Q_a \leftarrow EvaluateModelQuality(M_a, I)$
11: **if** $Q_a > Q_{tbest}$ **then**
12: $Q_{tbest} \leftarrow Q_a$
13: $R_{tbest} \leftarrow R_a$
14: **end if**
15: **end for**
16: $UpdatePheromones(R_{tbest})$
17: **if** $Q_{tbest} > Q_{bsf}$ **then**
18: $Q_{bsf} \leftarrow Q_{best}$
19: $R_{bsf} \leftarrow R_{tbest}$
20: $t \leftarrow t + 1$
21: **end if**
22: **until** $t =$ `max_iterations` **or** $Convergence($`conv_iterations`$)$
23: $M_{final} \leftarrow ConstructModel(h, R_{bsf})$
24: **return** M_{final};

struction graph. In turn, the ant will consider the two decision components (d_i^{true} and d_i^{false}) for each instance i – with i ranging from 1 to $|I|$ – and select one from amongst them probabilistically using the following formula:

$$P(d_i^v) = \frac{\tau[d_i^v].\eta[d_i^v]}{\tau[d_i^{true}].\eta[d_i^{true}] + \tau[d_i^{false}].\eta[d_i^{false}]} \tag{4.1}$$

where τ represents the amount of pheromone, η represents the amount of heuristic information associated with decision component d_i^v, and v can either be true (inclusion) or false (exclusion). The heuristic value for d^{true} decision components is preset at 0.66, and for d^{false} is preset at 0.33, which gives a slight bias towards including instances. Decision components are selected in this fashion and appended to the ant's set until all instances have been processed, at which point the contents of the set represent the solution constructed by the ant.

Algorithm 5 Solution Construction

```
1: T_a ← φ                                              ▷ ant trail
2: R_a ← φ                                              ▷ reduced data set
3: for  doi ← 1 to |I|
4:     d_i^v ← SelectDecisionComponent();
5:     T_a ← T_a ∪ d_i^v
6:     if d_i^x = d_i^True then
7:         R_a ← R_a ∪ I_i
8:     end if
9: end for
10: return R_a
```

4.4 Quality Evaluation and Pheromone Update

Since the main aim of performing data reduction is to improve the predictive effectiveness of the classifier, the quality of a candidate solution produced by an ant is evaluated by computing the predictive accuracy of the constructed model M – using the reduced set – on the original training set. Accuracy is a simple and yet popular predictive performance measure, and is computed as:

$$accuracy = \frac{|correct|}{|\texttt{data\_set}|}, \tag{4.2}$$

where *correct* is the set of correctly classified instances (using the classifier model currently constructed) in the `data_set`. Depending on which version of the ADR-Miner algorithm we are using, and which phase we are in within that version, the `data_set` could either mean the training or testing set. The higher the predictive accuracy, the better the classifier is performing.

As shown in both Algorithm 3 (line 14) and Algorithm 4 (line 16), only the ant with the best solution R_{tbest} in the current iteration t is allowed to deposit pheromone on the trail connecting the decision components chosen by it. The trail T_{tbest} selected by the iteration-best ant will have its (decision components') pheromone values amplified by a factor equal to the quality of solution attained, as follows:

$$\tau[d_i^v] = \tau[d_i^v] + (\tau[d_i^v] \times Q_{tbest}) \quad \forall d_i^v \in T_{tbest} \tag{4.3}$$

To simulate pheromone evaporation, normalization is then applied on each pair of solution components associated with each connection c in the construction graph. This keeps the total pheromone amount on each pair

$\tau[d_i^{true}]$ and $\tau[d_i^{false}]$ equal to 1, as follows:

$$\tau[d_i^v] = \frac{\tau[d_i^v]}{\tau[d_i^{true}] + \tau[d_i^{false}]} \quad \forall i \in I \tag{4.4}$$

Chapter 5

Implementation

In this chapter, we will discuss some of the work that went into the implementation of the ADR-Miner algorithms and how we were able to integrate it with WEKA.

5.1 ADR-Miner

The ADR-Miner algorithms described in the previous chapter are implemented using C# and Microsoft .NET. It is implemented as a Microsoft Windows application that takes a set of configurations, a training data set and a testing test and produces a set of results as text file. In what follows, we will describe the data structures and classes used to materialize ADR-Miner as was elaborated in Chapter 4.

Construction Graph

Although described as a two-dimensional array in the previous chapter, the construction graph is implemented as a flattened list. For every instance

in the data set, the list will contain a pair of decision components: one for inclusion and one for exclusion. The pairs are added to the list in order, such that each adjacent set of nodes in a pair will correspond to the same instance. During solution construction, the ants consider a choice between the nodes in a pair before moving on and repeating till they have made a choice per each pair in the list. The solution construction sub-procedure will start by marking the first pair as valid, and the rest of the pairs as invalid. After the ant makes its choice from among the pair, the procedure will than mark those as invalid, mark the next pair as valid and maintain the rest as invalid. An ant traversing the graph will continue to select and add components to its own solution as long as there are valid pairs, and the ant will terminate its solution construction process when there are no more valid pairs to choose from. The construction graph and the sliding window of validity are illustrated in figure 5.1.

Ants, The Colony and Reduction

The classes that form the core of the ADR-Miner algorithms are shown in figure 5.2. First, we have the AntColony class: a class that is responsible from initializing the colony, initializing the construction graph and managing the run of the algorithm. This class is abstract though, and only the initialization processes have been implemented. This initialization logic is available in the class constructor and the Initialize method. The constructor sets the basic parameters of the colony size and max iterations allowed. The Initialize method initializes the construction graph and sets the initial pheromone levels on the decision components within the graph. The rest of the methods

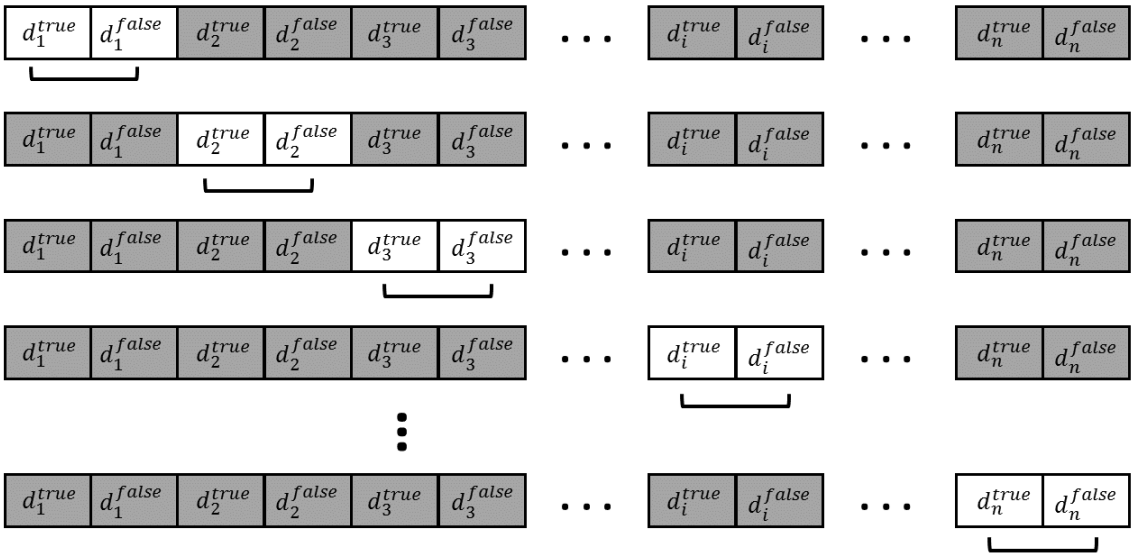


Figure 5.1: Materialization of the construction graph in ADR-Miner and the sliding window of validity during solution construction.

are materialized in the deriving class ACOMinerDR. AntColony was chosen to be an abstract class to allow for any future implementations that might choose to implement ACO models other than $MIN - MAX$ Ant System as the basis of ADR-Miner.

The ACOMinerDR is the main class that does the heavy lifting of implementing the ADR-Mienr logic. The Work method is the main method invoked by the logic responsible for running the various experiment batches. The Work method has a main loop that runs from zero till the max iteration count, and during each iteration it will allow all the ants in the colony to create solutions, update the global best ant and the update all the pheromone levels on the construction graph. As the ants construction their solution and the algorithm updates the current global best and pheromone levels, they will need a classifier to evaluate solution qualities. This classifier is provided by the WEKA framework, but more on that in the next section. You might have noticed that the class does not maintain a permanent collection of ants. That is because beyond constructing the solutions during the main loop in the Work method, the ants serve no other purpose. To save memory, the ants are only instantiated just in time, where they construct the solution and after which they are discarded. The only ants that we maintain references to are the iteration's current best and the global best.

Beyond the classes described above, there are a number of classes that are responsible for running the experiment batches and collating the results. These instantiate the ACOMinerDR class, supply it with a training data set, a classifier (or two depending on which version of ADR-Miner is being run) and invoke the Work method. Once the Work method is completed and we

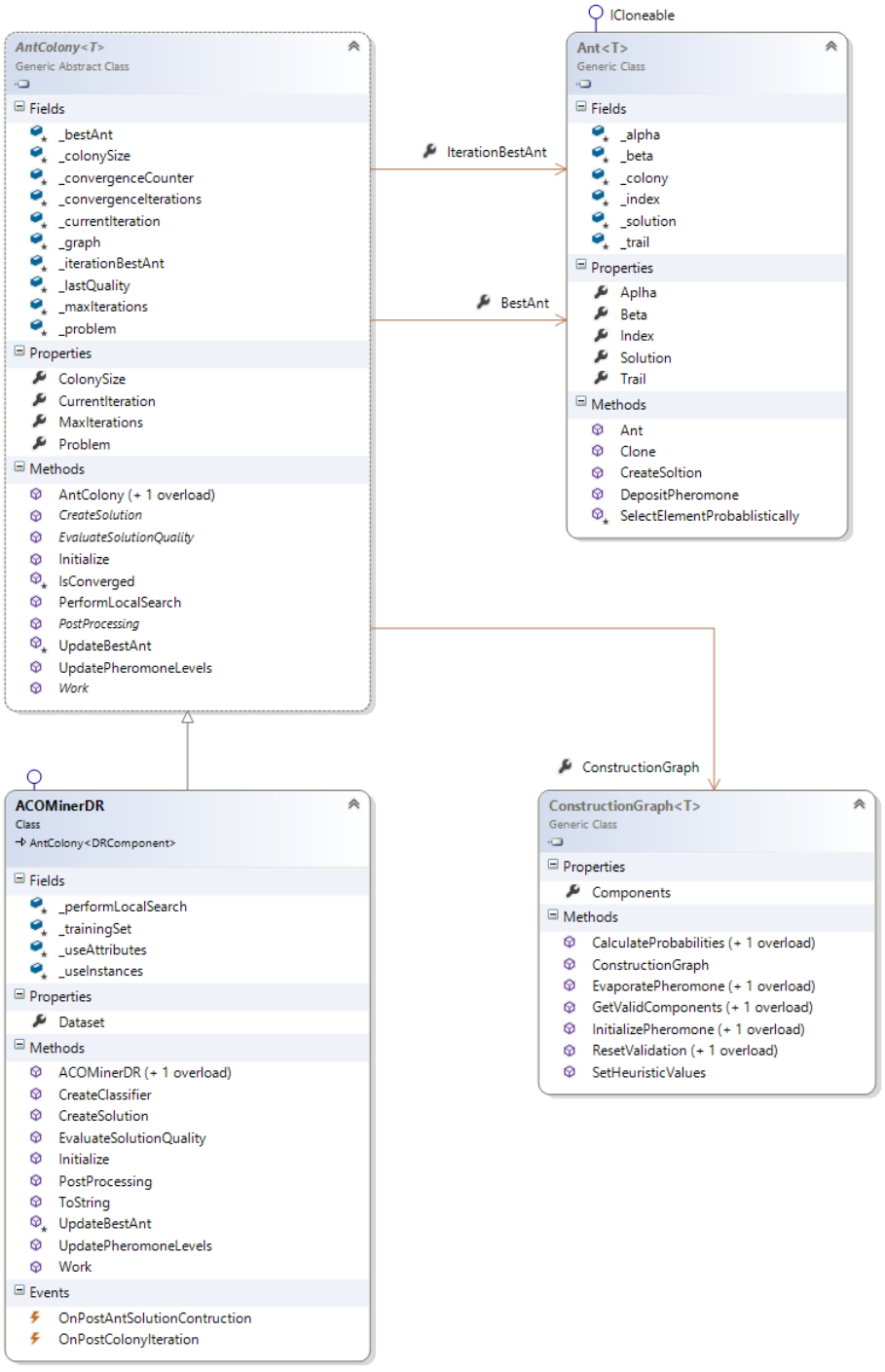


Figure 5.2: Class Diagram of ADRMinerDR, AntColony, Ant and ConstructionGraph.

have the indices of the instances to remove, these are then used to create a filter on the training data set to arrive at a reduced data set. The reduced data set is used to train the current classifier and its effectiveness is evaluated against a testing data set. The results of the evaluation are captured and stored as a text file.

5.2 WEKA Integration

The ADR-Miner algorithms are data reduction algorithms that were intended to improve the effectiveness of classifiers. In order to do so, ADR-Miner uses classifiers as fitness functions, and post-run as part of the testing procedure. Instead of reimplementing the classifiers from scratch, we instead opted to utilize the classifiers in the WEKA suite. WEKA, short for Waikato Environment for Knowledge Analysis, is an open-sourced collection of machine learning libraries written in Java and developed at the University of Waikato, New Zealand. With WEKA, one can perform a variety of data mining tasks, including classification, clustering, association rule mining and regression. As ADR-Miner is implemented using C# and Microsoft .NET technologies, we first have to expose the WEKA APIs to the .NET runtime before we are able to use them. In order to do so, we utilize the IKVM.NET project to provide us with a .NET library that exposes the WEKA APIs and thus can be used from the .NET runtime. The IKVM.NET project [36] is a .NET implementation for the Java Virtual Machine (JVM) and Java core libraries. In what follows are the steps taken to produce the .NET library:

- Download the latest IKVM binaries and source files archive. The archive can be found here: http://sourceforge.net/projects/ikvm/files/

- Extract the IKVM archive.
- Navigate to the main binaries. These can be found under <Extracted archive address>\bin.
- Execute the ikvmc.exe using the following parameters:

  ikvmc.exe -target:library <WEKA installation folder>\weka.jar

  This will produce a file called weka.dll, which is a .NET library equivalent of the weka.jar.

Disassembling the file produced by the aforementioned procedure, we can see that we get a number of namespaces under the root of "weka" that contain the logic for performing the various data mining tasks provided by the WEKA framework (see figure 5.3). Of particular interest is the "weka.classifiers" namespace which contains the implementations of the various classifiers that we are interested in.

The hierarchy of classes shown in figure 5.4 is built on top of the WEKA code, and it is the main facilitator that allows ADR-Miner access to the various classifiers provided by WEKA. First, an interface called "IClassifier" is introduced. This interface provides an abstraction to all the classifiers that ADR-Miner will interact with (be they from WEKA or not). The interface has three methods: Build, Initialize and Test. Initialize allows the implementing classifier to initialize itself using the parameters supplied. Build trains the implementing classifier using a provided data set and set of indexes that specify which instances to remove from the data set. Test assesses the classifier's accuracy using a provided testing data set. This interface is then implemented by a base class that provides most of the basic functionality

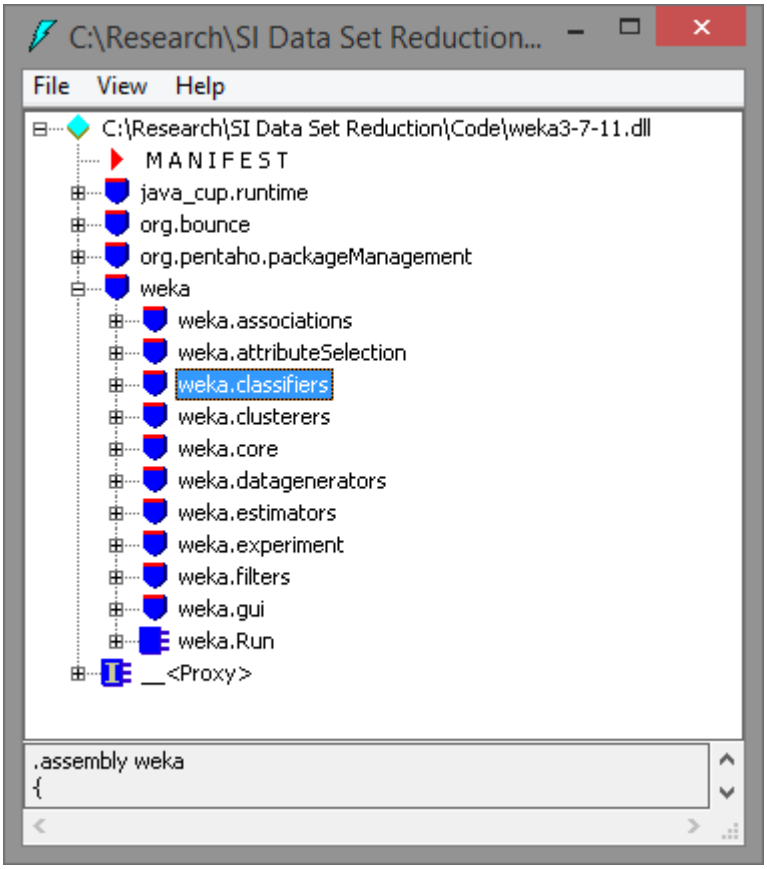


Figure 5.3: Disassembly of the weka.dll file. Performed by the Microsoft IL DASM tool.

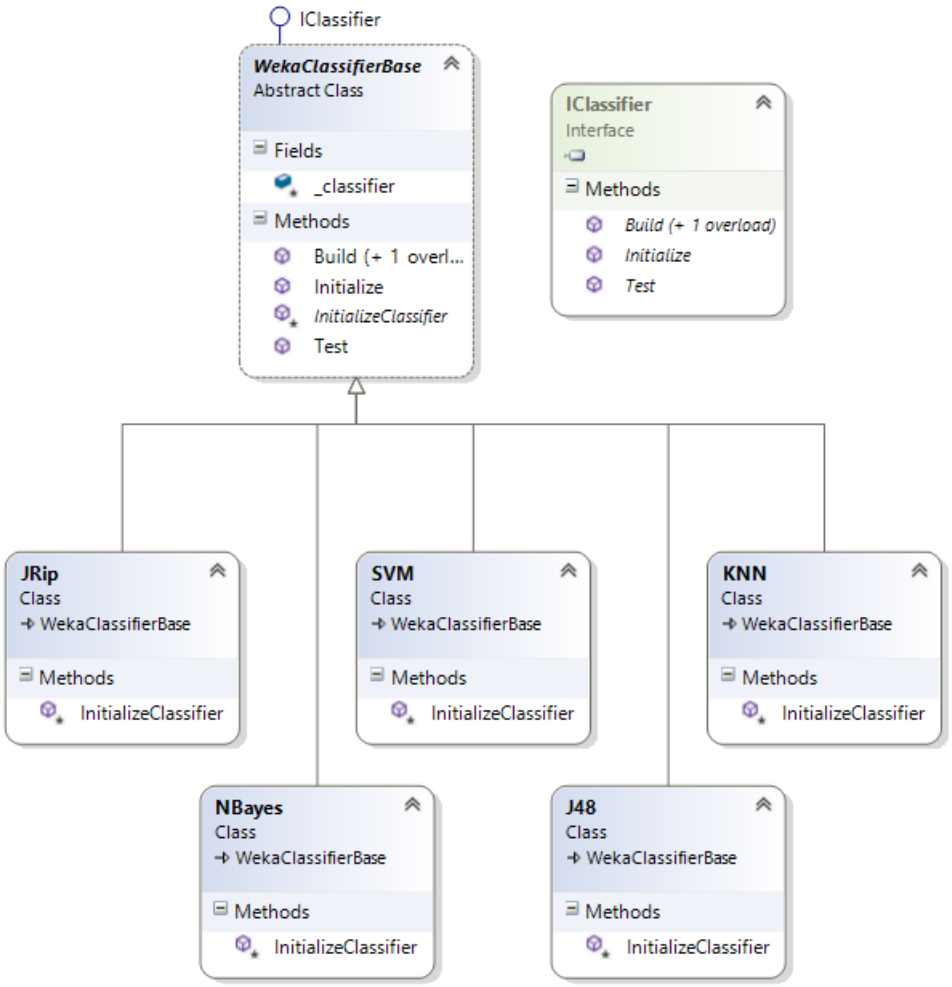


Figure 5.4: Class diagram of IClassifier, WekaClassifierBase and all derived classifiers.

for the Build and Test methods. Only the Initialize method is defined as abstract, to enforce deriving classes to implement the Initialize method in accordance to which classifier they represent. The WekaClassifierBase class is then extended into five classes, each representing a different type of classifier. Each extending class implements the Initialize method according to which classifier it represents using the property bag supplied as the method's sole parameter. This hierarchy not only cuts down on the amount of rework that would be required when working with the various classifiers that we use, but also allows us to dynamically expand ADR-Miner to support more classifiers, be they from the WEKA framework or not.

Chapter 6

Experimental Approach

In this chapter, we describe how we will examine the impact of using ADR-Miner on classifier effectiveness. We will begin by describing how we will examine the impact of using the first version of the ADR-Miner algorithm (Algorithm 3) on the effectiveness of produced classifiers, as well as how we will compare its performance to an established reducer algorithm as a benchmark. We will then go on to describe how we will explore the effect of using different classifiers with the extended version of the ADR-Miner algorith (Algorithm 4) and assess the influence such a scheme has on classifier effectiveness. The results of obtained from these experiments and their interpretation are presented in the next chapter.

6.1 Performance Evaluation and Benchmarking

We will begin our evaluation of the ADR-Miner algorithm using three well-known classification algorithms: k-nearest neighbor classifier, C4.5 decision tree induction algorithm, and the Ripper rule induction algorithm [19], [34]. In these experiments, we will use the WEKA [34] open source implementations of these algorithms (1-NN, J48 and JRip respectively). For each classification algorithm g, we will compare the predictive effectiveness of the classification models both before and after the application of the ADR-Miner algorithm as a reducer. As a benchmark, we will utilize the Iterative Case Filtering (ICF) algorithm [11] and compare our performance against its in terms of predictive accuracy as well as size reduction. For this phase of the evaluation, we will use 20 publicly available data sets from the the well-known University of California at Irvine (UCI) data set repository [23].

The experiments are to be carried out using the well-known *stratified* ten-fold cross validation procedure [19]. A ten-fold cross-validation procedure consists of dividing the data set into ten partitions of cases, wherein each partition has a similar number of cases and class distribution matching that of the original. For each partition, the classification algorithm is run using the remaining nine partitions as the training set and its performance is evaluated using the unseen (hold-out) partition. Since the ADR-Miner algorithm is stochastic in nature, it will be ran ten times using a different random seed to initialize the search each time. As the ICF algorithm is deterministic, it will be ran only once per classification algorithm for each iteration of the

cross-validation procedure.

The non-parametric Wilcoxon Signed-Rank test [21] is to be used to compare the predictive accuracy results of ADR-Miner to the results of the base algorithm, as well as the results of ICF, where the samples are the data sets and establish the statistical significance of the results if they are indeed so. The Wilcoxon Signed-Rank Test is performed as follows:

- Let N be the total number of data sets.
- Let d_i be the difference in performance between two algorithms on the i^{th} data set within N.
- Rank the differences based on absolute value starting from one. Case there is a tie, then all tied figures get an average value. For example, if both algorithms tie for the first and second spot, then they each recieve a rank of 1.5.
- Let R^+ be the sum of ranks where the second algorithm did better than the first.

$$R^+ = \sum_{d_i>0} rank(d_i) + \frac{1}{2}\sum_{d_i=0} rank(d_i) \tag{6.1}$$

- Let R^- be the sum of ranks where the second algorithm did worse than the first.

$$R^- = \sum_{d_i<0} rank(d_i) + \frac{1}{2}\sum_{d_i=0} rank(d_i) \tag{6.2}$$

- If there is an odd number of cases where $d_i = 0$ then one is ignored. The rest are split evenly between R^+ and R^-.
- Let $T = min(R^+, R^-)$, i.e. the minimum between the two sums.

- Most statistical references will have tabulations of the critical values of T, and whether or not the null-hypothesis is rejected can be inferred from these tabulations for the comparison of up to 25 data sets.

- For more than 25 data sets or samples, we calculate

$$z = \frac{T - \frac{1}{4}N(N+1)}{\sqrt{\frac{1}{24}N(N+1)(2N+1)}} \tag{6.3}$$

- The null hypothesis is rejected if z is less than -1.96 for a confidence level of 0.05.

The Wilcoxon Signed-Rank test was chosen over the paired T-test due to it being immune to the issues of needing the differences between the data sets under examination being comparable (also known as requiring commensurability); requiring a large sample size (at least around 30 data sets as the paired T-test has assumptions on the underlying distribution of the samples and these are only usually satisfied when the sample size is large enough) and being prone to being affected by the presence of outliers, issues that need to be resolved before using the paired T-test as a measure of statistical significance.

6.2 Effects of Using Different Classifiers During Reduction and Testing

For our second phase of testing, we will switch our attention to the effect of using different classifiers during the training and testing phases of the extended ADR-Miner algorithm (Algorithm 4) has on classifier effectiveness.

To do so, we are going to evaluate the extended version of ADR-Miner using pairings of the following classifiers for g and h: Support Vector Machines, Naive Bayes, C4.5 decision tree induction, k-Nearest Neighbor and Ripper. As with the previous section, we will use the WEKA open-source implementations of these classifiers, namely SMO, Naive-Bayes (NB), J48, 1-NN and JRip. This evaluation will be performed using 37 data sets from the UCI repository, and these data sets will prepared and used using the *stratified* ten-fold cross validation as before.

To aid in comparing the results obtained, we will rank both the base algorithms and all the pairings begotten from them. The average rank (be it of a base algorithm or a pairing) is to be calculated by first computing the rank of the entry on each data set individually. The individual rankings are then averaged across all the data sets to obtain the overall average rank. The lower the ranking, the better the algorithm or pairing is considered to be performing. This should provide a clearer picture of the performance of the pairings in terms of accuracy and how they compare to just using the base algorithms for both phases of the algorithm.

Chapter 7

Experimental Results

In this chapter, we will present the results obtained from both phases of testing described in the previous chapter. The results will be presented in their raw form, accompanied by insights observed and commentary on what the results imply.

7.1 Performance Evaluation and Benchmarking

Table 7.1 shows the average predictive accuracy (%) for each base classification algorithm g, ADR-Miner with the algorithm g (ADR-g) and ICF with the algorithm g (ICF-g). Size reduction (which represents the percentage of the data set that was removed during reduction) is shown between brackets below the provided accuracy results of both the ADR-Miner and the ICF data reduction algorithms. Both the highest predictive accuracies and highest reduction per algorithm g and data set are shown in boldface.

Table 7.1: Experimental Results - Predictive Accuracy % (Size Reduction %)

data set	1NN			JRip			J48		
	None	ADR	ICF	None	ADR	ICF	None	ADR	ICF
audiology	80.00	**80.83**	70.00	79.17	**84.17**	75.00	82.50	**85.83**	75.00
		(14.73)	**(81.12)**		(18.88)	**(81.12)**		(18.88)	**(81.12)**
breast-tissue	70.09	**72.18**	61.55	58.55	**63.45**	53.64	65.37	**66.36**	58.55
		(15.00)	**(84.29)**		(15.71)	**(84.29)**		(16.59)	**(84.29)**
car	61.81	63.33	**73.04**	87.66	**89.59**	78.36	**92.98**	92.87	83.10
		(16.49)	**(86.42)**		(13.58)	**(86.42)**		(19.01)	**(86.42)**
chess	84.53	86.04	**87.89**	99.00	**99.18**	97.77	**99.47**	99.43	98.02
		(18.58)	**(92.89)**		(7.11)	**(92.89)**		(25.88)	**(92.89)**
credit-a	81.02	81.58	**86.09**	85.51	**85.65**	85.07	**85.80**	85.07	84.78
		(16.36)	**(87.86)**		(12.14)	**(87.86)**		(20.08)	**(87.86)**
credit-g	71.50	**72.10**	71.90	**72.20**	70.40	69.10	69.60	**70.70**	70.20
		(16.20)	**(90.57)**		(9.43)	**(90.57)**		(18.02)	**(90.57)**
dermatology	94.53	**95.26**	92.09	88.01	**91.30**	79.80	**94.00**	93.99	89.93
		(22.89)	**(87.58)**		(12.42)	**(87.58)**		(23.86)	**(87.58)**
ecoli	81.31	81.72	**83.98**	81.86	**82.16**	77.66	**83.66**	82.78	83.39
		(16.9)	**(89.29)**		(10.71)	**(89.29)**		(18.35)	**(89.29)**
hepatitis	83.12	**84.46**	81.79	78.13	**85.79**	82.54	80.67	**80.71**	80.04
		(18.13)	**(93.41)**		(6.59)	**(93.41)**		(16.92)	**(93.41)**
horse	79.05	**79.46**	78.13	83.54	**85.03**	83.93	**84.75**	83.60	83.81
		(16.89)	**(95.62)**		(4.38)	**(95.62)**		(29.18)	**(95.62)**
ionosphere	87.38	**87.67**	83.63	**89.66**	89.39	85.34	**90.24**	88.52	89.07
		(17.39)	**(97.18)**		(2.82)	**(97.18)**		(22.18)	**(97.18)**
iris	95.33	**96.31**	95.33	92.00	**94.67**	89.33	**94.67**	94.00	89.33
		(29.98)	**(73.56)**		(26.44)	**(73.56)**		(30.51)	**(73.56)**
liver disorders	60.87	**62.62**	56.20	66.34	**68.08**	61.74	64.56	**64.88**	59.13
		(14.75)	**(91.95)**		(8.05)	**(91.95)**		(17.68)	**(91.95)**
lymphography	79.10	**83.76**	75.81	**79.95**	78.43	75.76	76.38	**77.90**	74.48
		(16.81)	**(81.83)**		(18.17)	**(81.83)**		(18.02)	**(81.83)**
s-heart	73.33	74.07	**79.26**	78.52	78.52	**79.26**	75.56	**76.30**	75.19
		(16.21)	**(88.89)**		(11.11)	**(88.89)**		(18.35)	**(88.89)**
soybean	88.62	**90.00**	85.52	83.10	**83.79**	74.48	83.11	**84.48**	75.17
		(20.18)	**(70.76)**		(11.11)	**(70.76)**		(18.57)	**(70.76)**
transfusion	61.55	62.22	**71.74**	**73.91**	72.82	72.95	73.78	71.82	**74.18**
		(16.25)	**(95.37)**		(4.63)	**(95.37)**		(20.59)	**(95.37)**
vertebral column 2	80.97	**81.32**	78.71	**82.58**	80.65	78.71	81.29	**82.26**	79.03
		(16.74)	**(92.80)**		(7.20)	**(92.80)**		(20.39)	**(92.80)**
vertebral column 3c	78.71	**79.68**	79.35	**80.32**	79.68	78.06	78.71	**80.65**	**80.65**
		(15.70)	**(93.19)**		(6.81)	**(93.19)**		(17.89)	**(93.19)**
voting	88.73	**88.99**	87.87	93.66	**95.55**	94.57	94.48	**94.88**	94.57
		(20.99)	**(95.96)**		(6.81)	**(95.96)**		(29.08)	**(95.96)**

The results in Table 7.1 indicate the following. In 1-NN instance-based learning, ADR-Miner achieved the highest predictive accuracies in 14 data sets, compared to 6 for ICF. For the JRip classification algorithm, ADR-Miner achieved the highest predictive accuracies in 13 data sets, compared to 1 for ICF and 6 for no reduction at all. For the J48 classification algorithm, ADR-Miner achieved a slight lead by achieving the highest predictive accuracies in 11 data sets (one a tie with ICF), followed closely by no reduction with 8 data sets and trailed by ICF with 2 data sets. This shows that the ADR-Miner has achieved the highest predictive accuracies overall when compared to no reduction and ICF per classification algorithm, and that this lead in accuracy comes with varying margins between the runner ups as was just described.

As for size reduction, the ICF algorithm produced smaller data set sizes in all the data set used in our experiments, compared to ADR-Miner. However, as indicated in the predictive accuracy results, this seems to have come at the expense of accuracy. It is evident that ICF seems to have removed training instances that were actually needed for learning the classification models, as it is shown that it has sacrificed accuracy for size (see results in Table 7.1).

Table 7.2 shows the results (p-values) of the Wilcoxon Signed-Rank test. In the table, p-values that are less than or equal to the conventional 0.05 threshold are shown in boldface.

As can be seen from Table 7.2, statistically significant improvements in classifier effectiveness have been attained with both the 1-NN and JRip algorithms when compared with the base algorithms without reduction, as we well as similarly significant results with JRip and J48 when compared against

ADR-Miner - Classifier	**ADR-1NN**	ADR-1NN	**ADR-JRip**	**ADR-JRip**	ADR-J48	**ADR-J48**
Reduction Algorithm	**None**	ICF	**None**	**ICF**	None	**ICF**
N	20	20	19	20	20	19
W^+	210.0	135.0	48.0	206.0	82.5	168.0
W^-	0.0	75.0	142.0	4.0	127.5	22.0
Mean Difference	76.99	18.63	19.19	29.28	16.22	25.58
W	0.00	75.00	48.00	4.00	82.50	22.00
Mean(W)	105	105	95	105	105	95
Standard Deviation(W)	26.79	26.79	24.85	26.79	26.79	24.85
Critical Value for W	60	60	53	60	60	53
z	-3.9199	-1.1200	-1.8914	-3.7706	-0.8400	-2.9377
p	**0.00050**	0.13136	**0.02938**	**0.00008**	0.20045	**0.00164**
Significant?	**Yes**	No	**Yes**	**Yes**	No	**Yes**

Table 7.2: Results of the Wilcoxon Signed-Rank Test. The ADR-Miner algorithm (paired with a classifier) is compared against the performance of a reducer.

ICF.

7.2 Effects of Using Different Classifiers During Reduction and Testing

Table 7.3 shows the predictive accuracy (%) of the classifiers without data reduction as a baseline. Figures 7.1 through 7.25 show the accuracy results of each pairing of classifiers post data reduction with the extended ADR-Miner algorithm. The figures are labeled as "g - h Results", where the first algorithm appearing in the figure caption represents g and the second representing h (the classifier used during training and testing respectively). An alternative view of the pairing results can be seen the Appendix.

Data Set	Accuracy
annealing	94.41
audiology	80.83
automobile	73.64
breast-p	69.16
breast-tissue	67.18
breast-w	95.79
car	63.33
chess	86.04
credit-a	80.58
credit-g	69.00
cylinder	68.19
dermatology	94.26
ecoli	80.72
glass	68.92
hay	63.08
heart-c	53.83
heart-h	62.43
hepatitis	82.46
horse	78.46
iris	95.33
liver-disorders	62.62
lymphography	83.76
monks	57.09
mushrooms	100.00
nursery	42.91
parkinsons	94.89
pop	70.00
s-heart	74.07
soybean	90.00
thyroid	93.96
transfusion	62.22
ttt	67.37
vehicle	67.73
vertebral-2c	80.32
vertebral-3c	79.68
voting	88.99
zoo	98.75

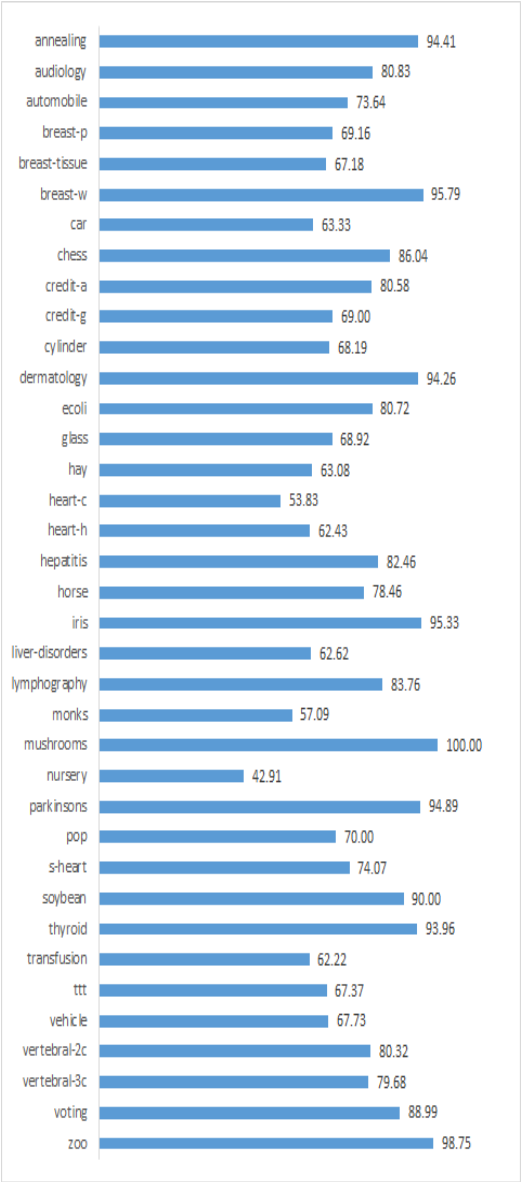


Figure 7.1: 1-Nearest Neighbor (1-NN) - 1-Nearest Neighbor (1-NN) Results

Data Set	Accuracy
annealing	77.73
audiology	85.00
automobile	57.52
breast-p	67.13
breast-tissue	66.18
breast-w	93.33
car	86.26
chess	87.86
credit-a	77.10
credit-g	74.90
cylinder	66.32
dermatology	97.81
ecoli	84.87
glass	48.57
hay	68.46
heart-c	56.77
heart-h	66.05
hepatitis	83.17
horse	78.53
iris	94.67
liver-disorders	54.74
lymphography	82.48
monks	61.09
mushrooms	95.38
nursery	90.40
parkinsons	71.68
pop	68.75
s-heart	84.44
soybean	88.28
thyroid	96.71
transfusion	70.26
ttt	69.16
vehicle	45.04
vertebral-2c	78.06
vertebral-3c	81.94
voting	85.94
zoo	93.75

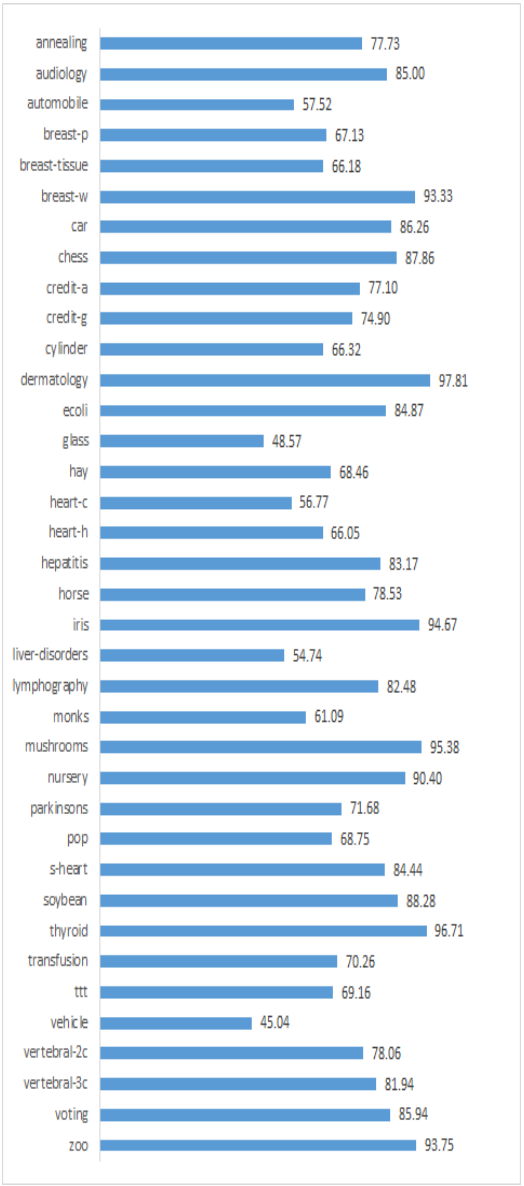


Figure 7.2: 1-Nearest Neighbor (1-NN) - Naïve Bayes (NB) Results

Data Set	Accuracy
annealing	93.93
audiology	78.33
automobile	71.71
breast-p	73.68
breast-tissue	58.82
breast-w	93.15
car	83.51
chess	99.06
credit-a	86.09
credit-g	73.60
cylinder	67.10
dermatology	87.95
ecoli	81.86
glass	67.00
hay	79.23
heart-c	55.48
heart-h	63.01
hepatitis	78.75
horse	86.44
iris	94.00
liver-disorders	69.58
lymphography	79.76
monks	57.09
mushrooms	99.96
nursery	96.45
parkinsons	82.11
pop	73.75
s-heart	77.78
soybean	83.79
thyroid	92.53
transfusion	73.99
ttt	97.37
vehicle	70.58
vertebral-2c	80.65
vertebral-3c	79.68
voting	92.89
zoo	97.50

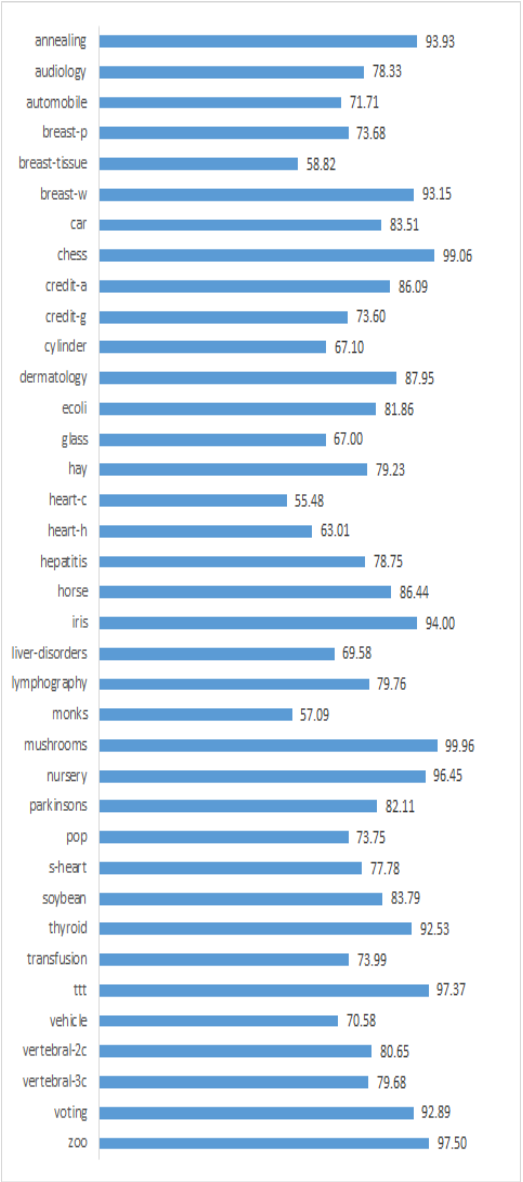


Figure 7.3: 1-Nearst Neighbor (1-NN) - Ripper (JRip) Results

Data Set	Accuracy
annealing	91.90
audiology	78.33
automobile	80.57
breast-p	73.21
breast-tissue	62.27
breast-w	95.79
car	92.05
chess	99.21
credit-a	85.51
credit-g	71.60
cylinder	73.79
dermatology	94.26
ecoli	83.36
glass	70.79
hay	63.08
heart-c	53.83
heart-h	67.06
hepatitis	79.33
horse	84.15
iris	92.67
liver-disorders	65.50
lymphography	77.10
monks	58.73
mushrooms	100.00
nursery	96.57
parkinsons	81.89
pop	75.00
s-heart	77.04
soybean	83.79
thyroid	91.60
transfusion	73.91
ttt	83.26
vehicle	69.39
vertebral-2c	81.94
vertebral-3c	81.29
voting	94.48
zoo	98.75

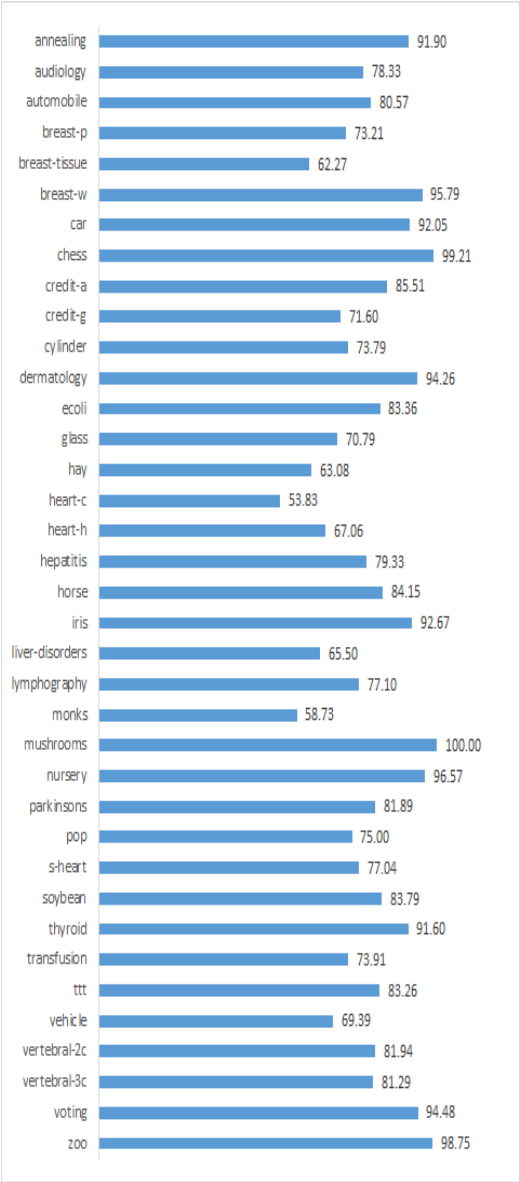


Figure 7.4: 1-Nearest Neighbor (1-NN) - C4.5 (J48) Results

Data Set	Accuracy
annealing	85.74
audiology	88.33
automobile	71.21
breast-p	76.29
breast-tissue	58.73
breast-w	97.54
car	92.81
chess	95.69
credit-a	84.78
credit-g	74.60
cylinder	72.11
dermatology	95.88
ecoli	83.06
glass	56.58
hay	78.46
heart-c	55.77
heart-h	66.76
hepatitis	83.83
horse	80.99
iris	96.00
liver-disorders	58.28
lymphography	83.10
monks	62.55
mushrooms	100.00
nursery	93.08
parkinsons	85.08
pop	70.00
s-heart	84.44
soybean	90.00
thyroid	87.03
transfusion	71.74
ttt	98.42
vehicle	74.94
vertebral-2c	73.55
vertebral-3c	80.65
voting	94.26
zoo	98.75

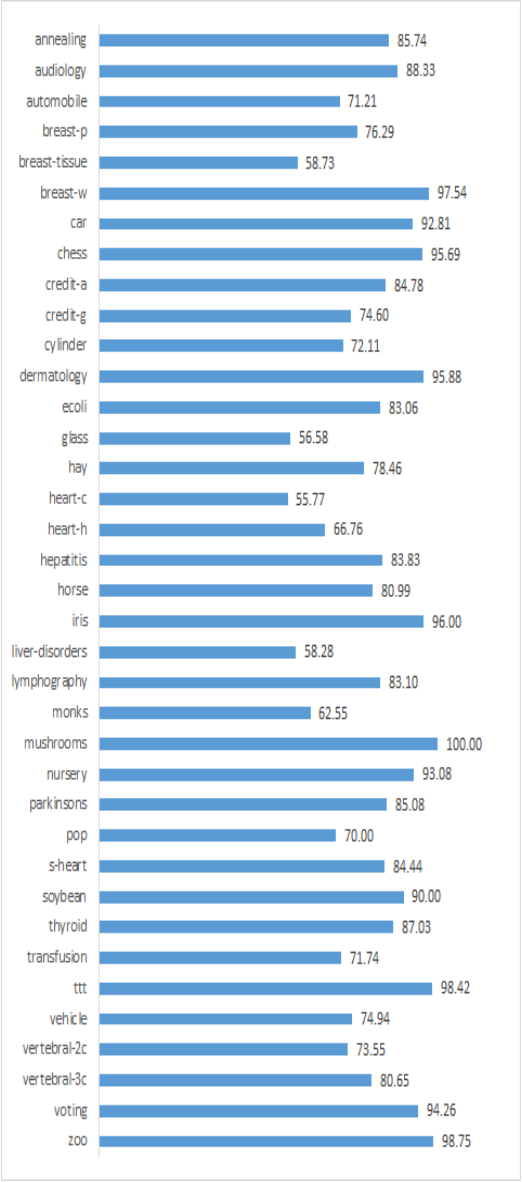


Figure 7.5: 1-Nearest Neighbor (1-NN) - Support Vector Machine (SMO) Results

Data Set	Accuracy
annealing	94.18
audiology	77.50
automobile	69.26
breast-p	67.71
breast-tissue	67.27
breast-w	95.79
car	62.98
chess	85.38
credit-a	81.30
credit-g	70.60
cylinder	67.26
dermatology	93.45
ecoli	80.44
glass	68.86
hay	62.31
heart-c	52.81
heart-h	48.57
hepatitis	81.88
horse	79.34
iris	96.00
liver-disorders	61.72
lymphography	77.81
monks	57.64
mushrooms	100.00
nursery	43.28
parkinsons	92.34
pop	71.25
s-heart	74.07
soybean	86.21
thyroid	93.96
transfusion	62.74
ttt	67.58
vehicle	69.26
vertebral-2c	77.74
vertebral-3c	80.32
voting	88.64
zoo	98.75

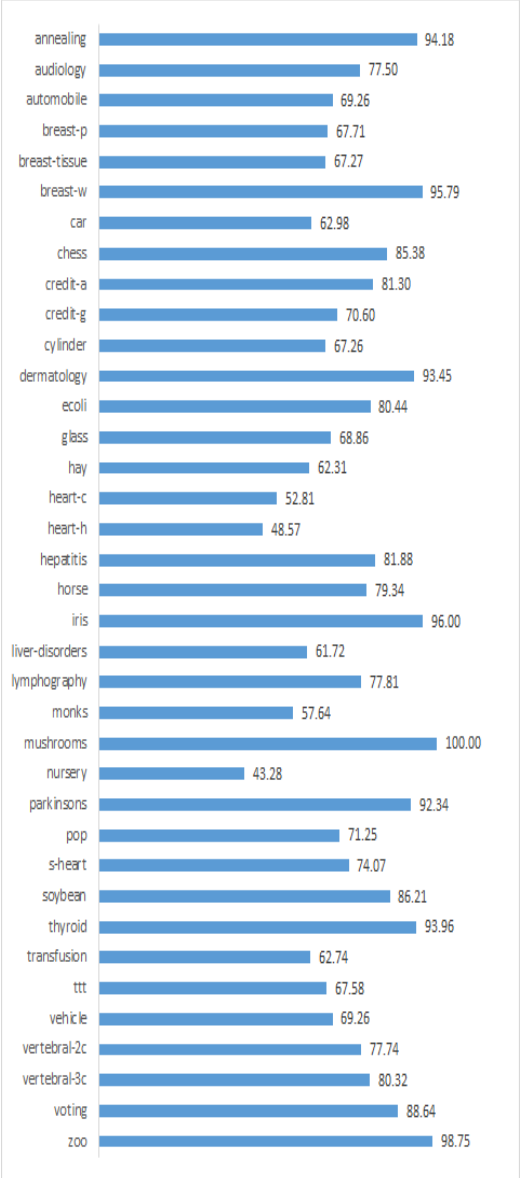


Figure 7.6: Naïve Bayes (NB) - 1-Nearest Neighbor (1-NN) Results

Data Set	Accuracy
annealing	88.71
audiology	85.00
automobile	61.93
breast-p	68.68
breast-tissue	65.45
breast-w	93.86
car	87.54
chess	88.96
credit-a	77.68
credit-g	74.80
cylinder	70.60
dermatology	97.26
ecoli	85.78
glass	59.37
hay	76.92
heart-c	56.44
heart-h	66.04
hepatitis	86.29
horse	79.96
iris	96.00
liver-disorders	64.02
lymphography	84.48
monks	63.45
mushrooms	96.02
nursery	91.00
parkinsons	74.29
pop	68.75
s-heart	83.70
soybean	88.62
thyroid	96.69
transfusion	71.49
ttt	72.63
vehicle	50.95
vertebral-2c	77.74
vertebral-3c	82.58
voting	85.98
zoo	96.25

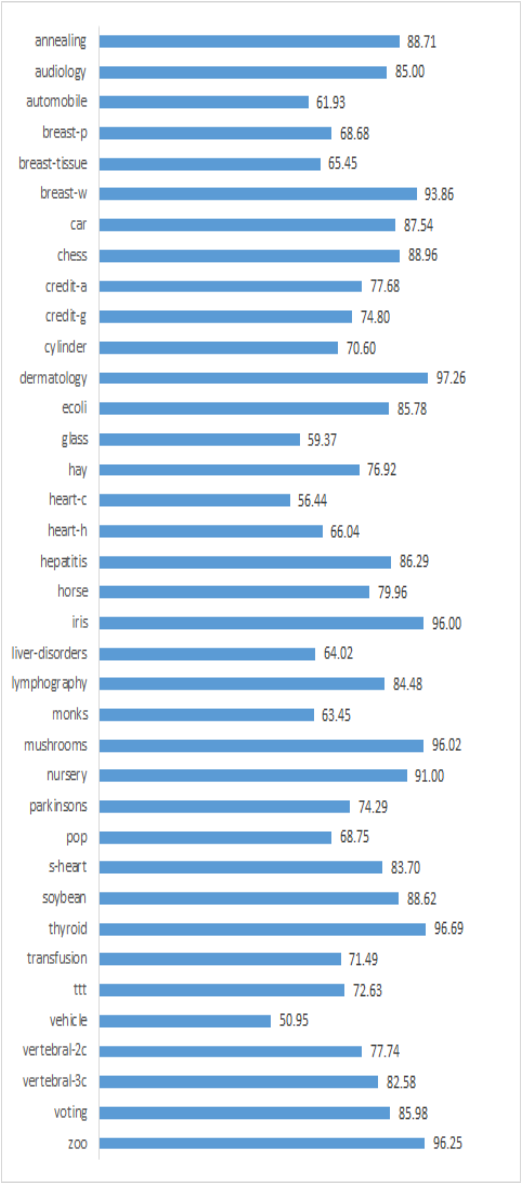


Figure 7.7: Naïve Bayes (NB) - Naïve Bayes (NB) Results

Data Set	Accuracy
annealing	93.83
audiology	79.17
automobile	71.24
breast-p	69.61
breast-tissue	62.18
breast-w	92.62
car	85.20
chess	99.09
credit-a	83.77
credit-g	73.80
cylinder	67.82
dermatology	88.57
ecoli	79.80
glass	65.14
hay	82.31
heart-c	54.47
heart-h	62.31
hepatitis	80.67
horse	84.11
iris	90.00
liver-disorders	67.82
lymphography	75.86
monks	62.18
mushrooms	99.96
nursery	96.40
parkinsons	84.61
pop	75.00
s-heart	77.04
soybean	81.38
thyroid	93.01
transfusion	72.89
ttt	97.79
vehicle	67.51
vertebral-2c	80.00
vertebral-3c	80.32
voting	93.68
zoo	91.25

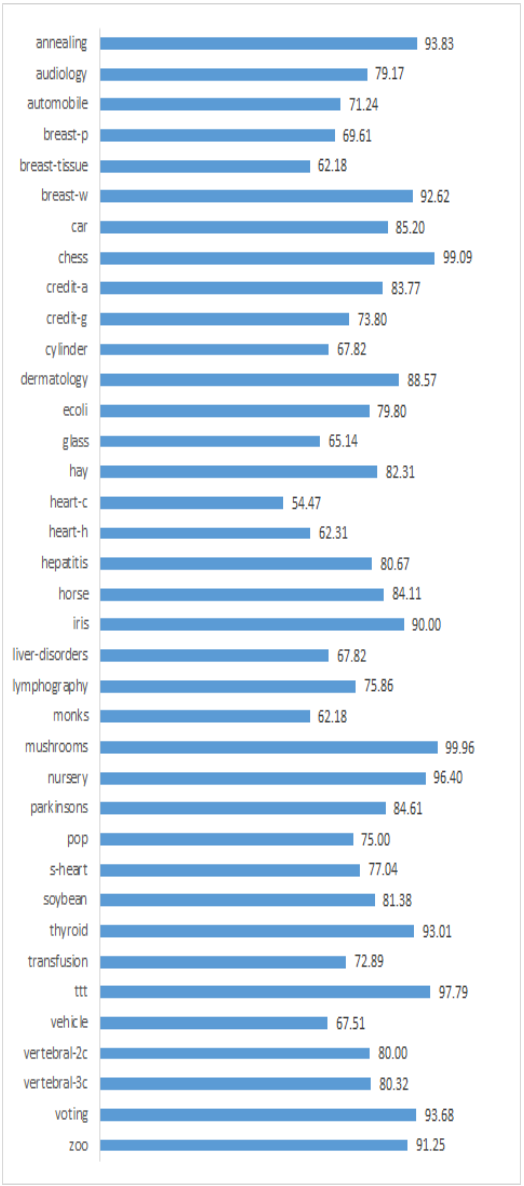


Figure 7.8: Naïve Bayes (NB) - Ripper (JRip) Results

Data Set	Accuracy
annealing	91.43
audiology	83.33
automobile	77.98
breast-p	69.13
breast-tissue	58.55
breast-w	92.44
car	91.81
chess	99.18
credit-a	85.51
credit-g	71.90
cylinder	73.79
dermatology	92.91
ecoli	81.60
glass	70.34
hay	65.38
heart-c	57.08
heart-h	66.42
hepatitis	79.38
horse	84.42
iris	91.33
liver-disorders	64.04
lymphography	77.14
monks	58.91
mushrooms	100.00
nursery	96.51
parkinsons	81.00
pop	75.00
s-heart	77.78
soybean	85.86
thyroid	93.44
transfusion	72.29
ttt	84.95
vehicle	72.11
vertebral-2c	81.29
vertebral-3c	79.35
voting	93.68
zoo	98.75

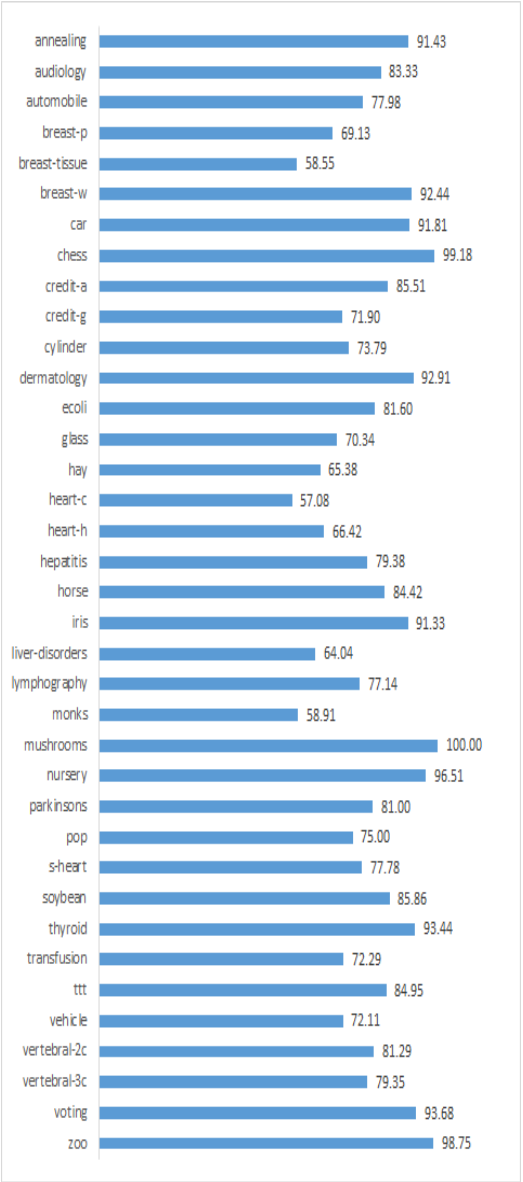


Figure 7.9: Naïve Bayes (NB) - C4.5 (J48) Results

Data Set	Accuracy
annealing	84.38
audiology	90.83
automobile	68.24
breast-p	76.29
breast-tissue	57.73
breast-w	97.37
car	93.10
chess	95.53
credit-a	84.78
credit-g	73.20
cylinder	74.34
dermatology	96.45
ecoli	81.86
glass	51.75
hay	77.69
heart-c	55.76
heart-h	67.10
hepatitis	82.54
horse	81.78
iris	96.67
liver-disorders	58.56
lymphography	79.81
monks	63.64
mushrooms	100.00
nursery	93.01
parkinsons	84.61
pop	71.25
s-heart	83.33
soybean	88.62
thyroid	87.01
transfusion	71.74
ttt	98.42
vehicle	74.23
vertebral-2c	71.29
vertebral-3c	76.45
voting	93.91
zoo	96.25

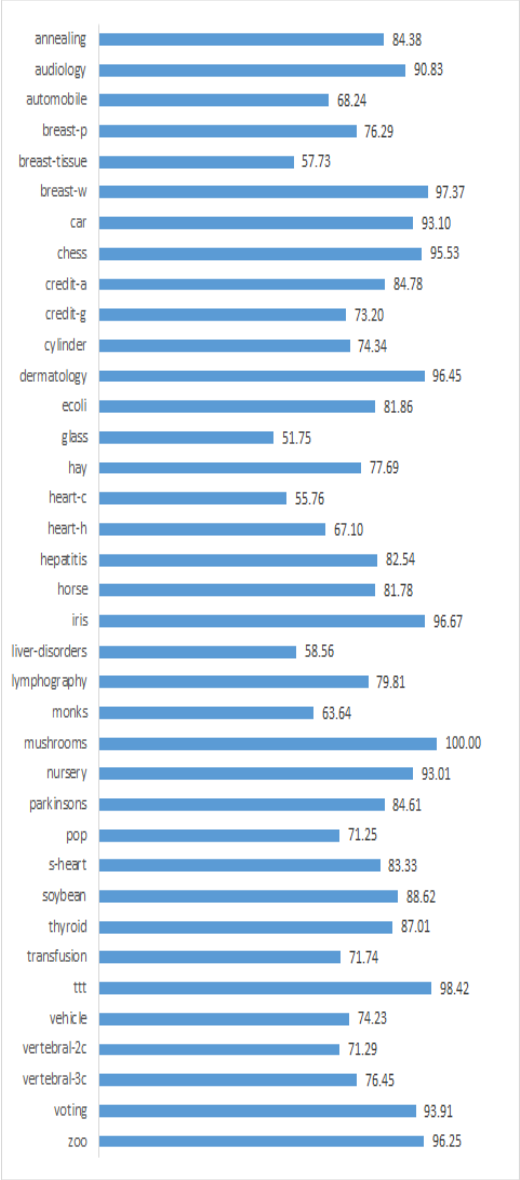


Figure 7.10: Naïve Bayes (NB) - Support Vector Machine (SMO) Results

Data Set	Accuracy
annealing	93.49
audiology	80.83
automobile	72.62
breast-p	67.16
breast-tissue	68.27
breast-w	94.91
car	63.27
chess	85.19
credit-a	80.72
credit-g	69.50
cylinder	68.94
dermatology	93.45
ecoli	79.79
glass	66.04
hay	58.46
heart-c	52.84
heart-h	46.51
hepatitis	80.58
horse	79.04
iris	95.33
liver-disorders	61.13
lymphography	79.05
monks	56.73
mushrooms	100.00
nursery	98.50
parkinsons	93.39
pop	71.25
s-heart	75.56
soybean	86.55
thyroid	95.78
transfusion	63.53
ttt	67.58
vehicle	68.08
vertebral-2c	77.10
vertebral-3c	78.39
voting	89.41
zoo	98.75

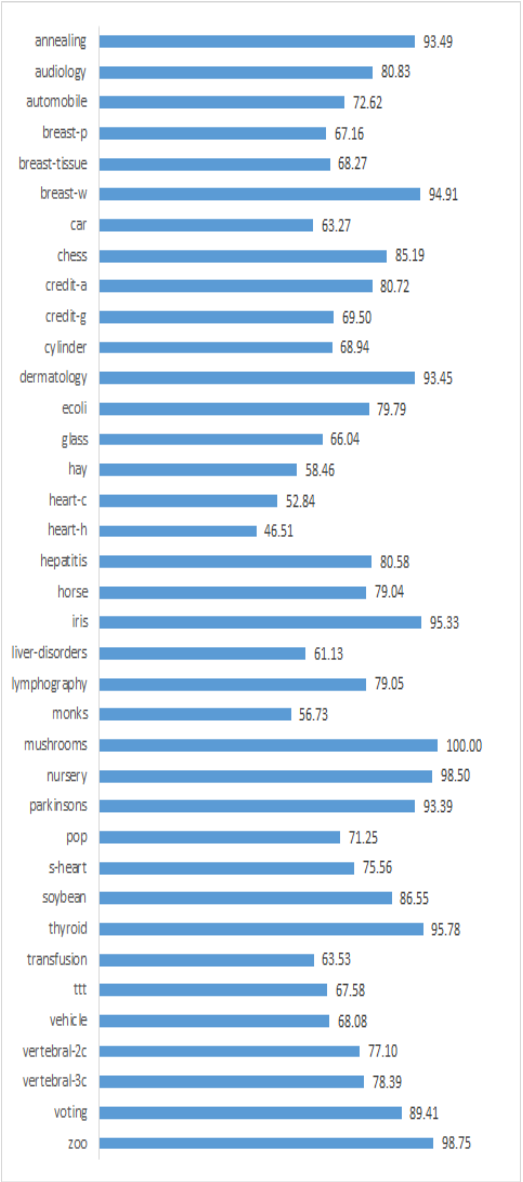


Figure 7.11: Ripper (JRip) - 1-Nearest Neighbor (1-NN) Results

Data Set	Accuracy
annealing	79.20
audiology	84.17
automobile	58.05
breast-p	66.13
breast-tissue	67.27
breast-w	93.51
car	85.38
chess	87.67
credit-a	76.67
credit-g	74.50
cylinder	67.64
dermatology	98.09
ecoli	84.27
glass	47.95
hay	69.23
heart-c	56.74
heart-h	63.30
hepatitis	82.58
horse	79.41
iris	96.00
liver-disorders	54.46
lymphography	82.48
monks	61.27
mushrooms	95.24
nursery	94.40
parkinsons	71.18
pop	68.75
s-heart	84.07
soybean	87.59
thyroid	96.71
transfusion	70.05
ttt	69.89
vehicle	44.69
vertebral-2c	78.39
vertebral-3c	82.90
voting	85.98
zoo	96.25

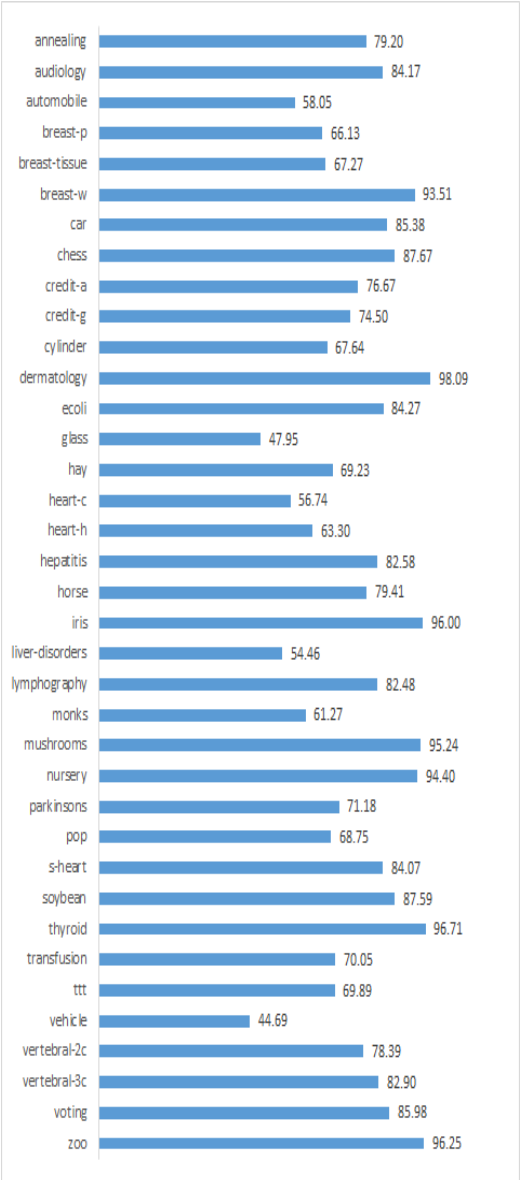


Figure 7.12: Ripper (JRip) - Naïve Bayes (NB) Results

Data Set	Accuracy
annealing	94.97
audiology	84.17
automobile	76.12
breast-p	69.21
breast-tissue	63.45
breast-w	94.20
car	89.59
chess	99.18
credit-a	85.65
credit-g	70.40
cylinder	73.25
dermatology	91.30
ecoli	82.16
glass	70.34
hay	74.62
heart-c	51.52
heart-h	64.74
hepatitis	85.79
horse	85.03
iris	94.67
liver-disorders	68.08
lymphography	78.43
monks	60.00
mushrooms	100.00
nursery	98.50
parkinsons	85.61
pop	72.50
s-heart	78.52
soybean	83.79
thyroid	90.69
transfusion	72.82
ttt	96.95
vehicle	71.15
vertebral-2c	80.65
vertebral-3c	79.68
voting	95.55
zoo	93.75

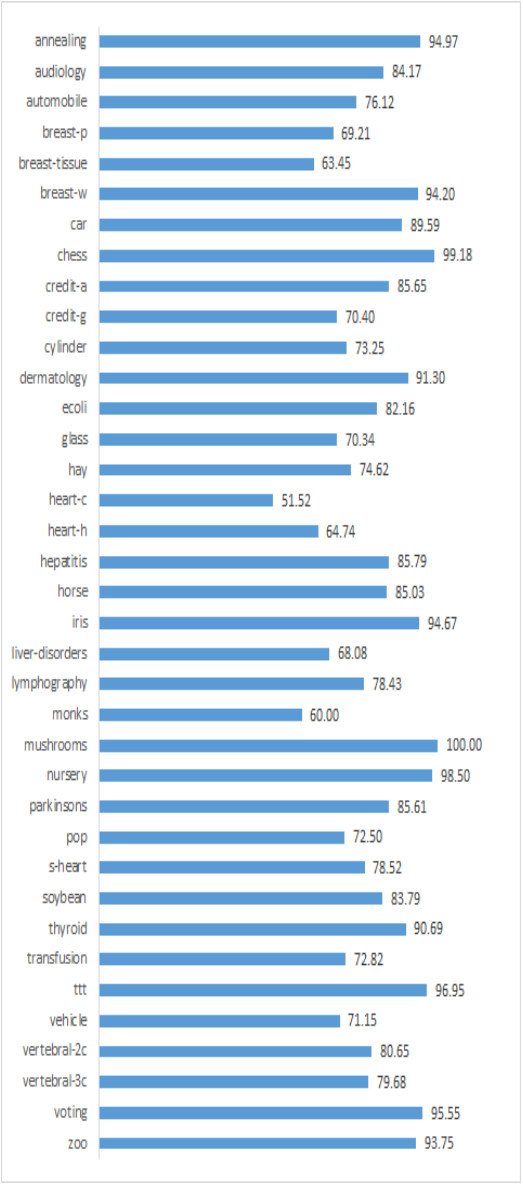


Figure 7.13: Ripper (JRip) - Ripper (JRip) Results

Data Set	Accuracy
annealing	90.65
audiology	82.50
automobile	77.05
breast-p	72.16
breast-tissue	62.36
breast-w	91.91
car	91.11
chess	99.37
credit-a	85.51
credit-g	70.60
cylinder	71.90
dermatology	93.97
ecoli	81.89
glass	68.41
hay	66.92
heart-c	51.51
heart-h	66.13
hepatitis	81.96
horse	84.74
iris	95.33
liver-disorders	63.99
lymphography	79.86
monks	59.64
mushrooms	100.00
nursery	97.30
parkinsons	83.11
pop	72.50
s-heart	75.19
soybean	83.10
thyroid	93.48
transfusion	72.41
ttt	82.74
vehicle	71.99
vertebral-2c	80.32
vertebral-3c	80.32
voting	94.25
zoo	97.50

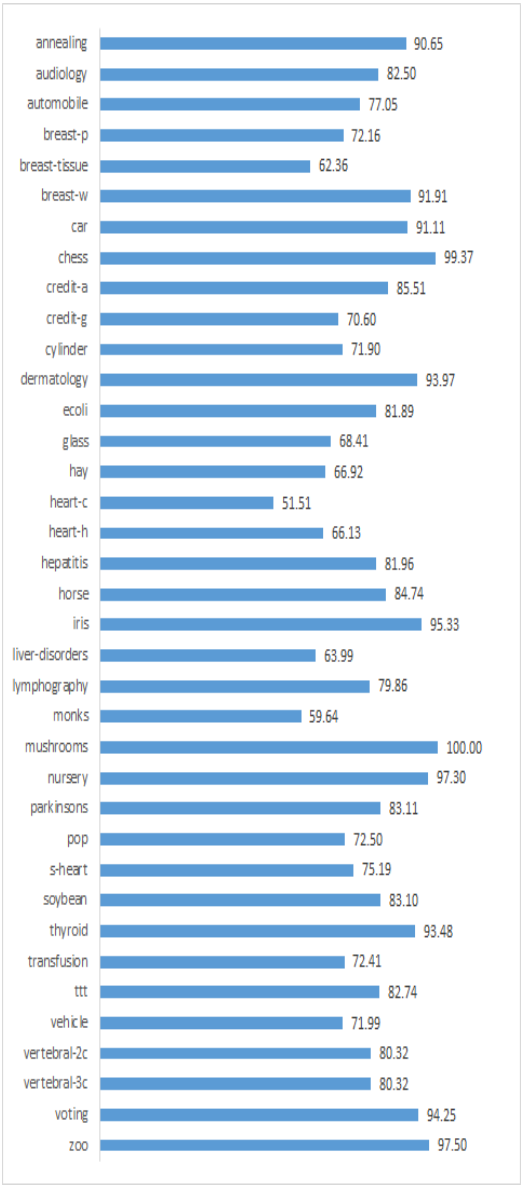


Figure 7.14: Ripper (JRip) - C4.5 (J48) Results

Data Set	Accuracy
annealing	85.97
audiology	90.00
automobile	66.69
breast-p	76.29
breast-tissue	58.82
breast-w	97.02
car	92.63
chess	95.22
credit-a	84.64
credit-g	73.50
cylinder	72.48
dermatology	97.55
ecoli	82.75
glass	56.01
hay	77.69
heart-c	56.78
heart-h	65.41
hepatitis	84.46
horse	83.30
iris	94.67
liver-disorders	57.70
lymphography	85.19
monks	63.45
mushrooms	100.00
nursery	97.30
parkinsons	87.18
pop	72.50
s-heart	84.81
soybean	89.31
thyroid	88.42
transfusion	71.94
ttt	98.42
vehicle	73.99
vertebral-2c	78.39
vertebral-3c	75.81
voting	93.63
zoo	96.25

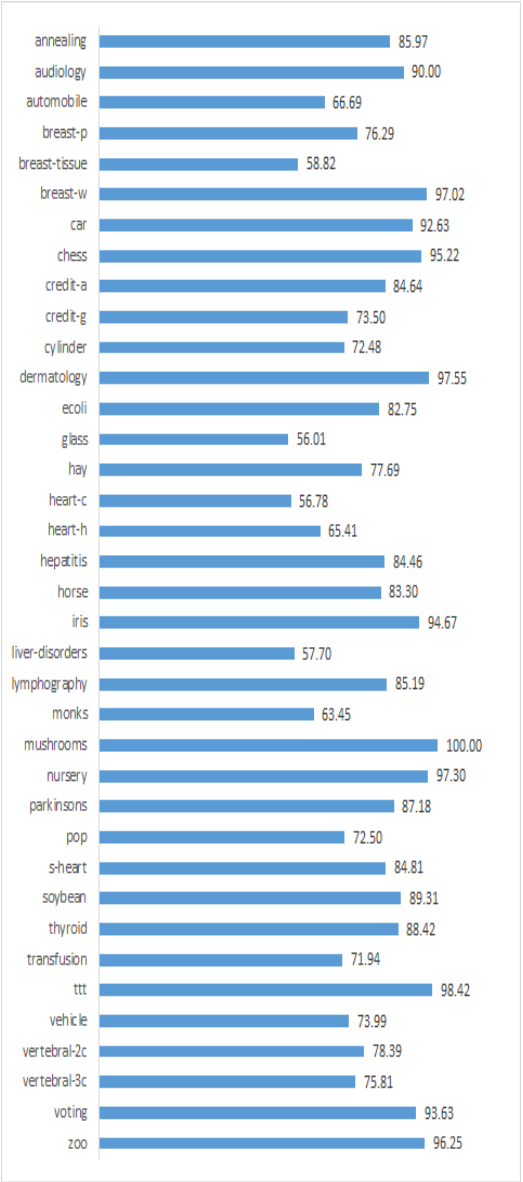


Figure 7.15: Ripper (JRip) - Support Vector Machine (SMO) Results

Table 7.4 shows the average rankings of the results from the 25 pairings, along with the base algorithms without reduction. The base algorithms here are displayed on their own and not in the $g - h$ format.

From the ranking results in Table 7.4, that using ADR-Miner with SMO-SMO for $g - h$ has produced the best results on average. This is followed by SMO without data reduction, ADR-Miner with 1-NN-SMO, ADR-Miner with J48-SMO and finally ADR-Miner with JRip-JRip for fifth place. We can observe from the top four results that using SMO as the classification algorithm to train the final model has generally produced superior results in terms of accuracy. This superiority comes at a cost though: incomprehensibility. SMO models are known for their effectivenesss, but the models produced by the algorithm are incomprehensible to human operators. The operators may opt to go with classifiers that generate models that are more user friendly than SMO, such as JRip or J48 [30], [37], [40].

We can also observe another fact from Table 7.4. Pairings for given algorithm while serving as h perform better than if that algorithm was used on its own without reduction. The only exception to that observation is J48 algorithm which performs well on its own before being paired with another while serving as h.

On the aspect of size reduction, we can see from Table 7.5 that the Naive-Bayes classifier has achieved the highest average size reduction, followed by SMO, J48, JRip and 1-NN. Note that for size reduction, we only consider the reduction during the training phase (i.e. the reduction achieved when the classifier is used as g), as the final model is built post reduction.

Tables 7.6a and 7.6b summarize the best pairings observed. Analyzing

Data Set	Accuracy
annealing	93.95
audiology	77.50
automobile	72.19
breast-p	67.16
breast-tissue	68.09
breast-w	96.14
car	63.22
chess	85.85
credit-a	81.16
credit-g	69.90
cylinder	68.75
dermatology	94.26
ecoli	82.79
glass	69.33
hay	62.31
heart-c	52.52
heart-h	51.42
hepatitis	83.17
horse	79.69
iris	94.67
liver-disorders	61.15
lymphography	79.10
monks	57.45
mushrooms	100.00
nursery	43.13
parkinsons	95.42
pop	68.75
s-heart	74.81
soybean	88.62
thyroid	94.87
transfusion	62.08
ttt	67.37
vehicle	70.09
vertebral-2c	76.77
vertebral-3c	78.71
voting	88.03
zoo	98.75

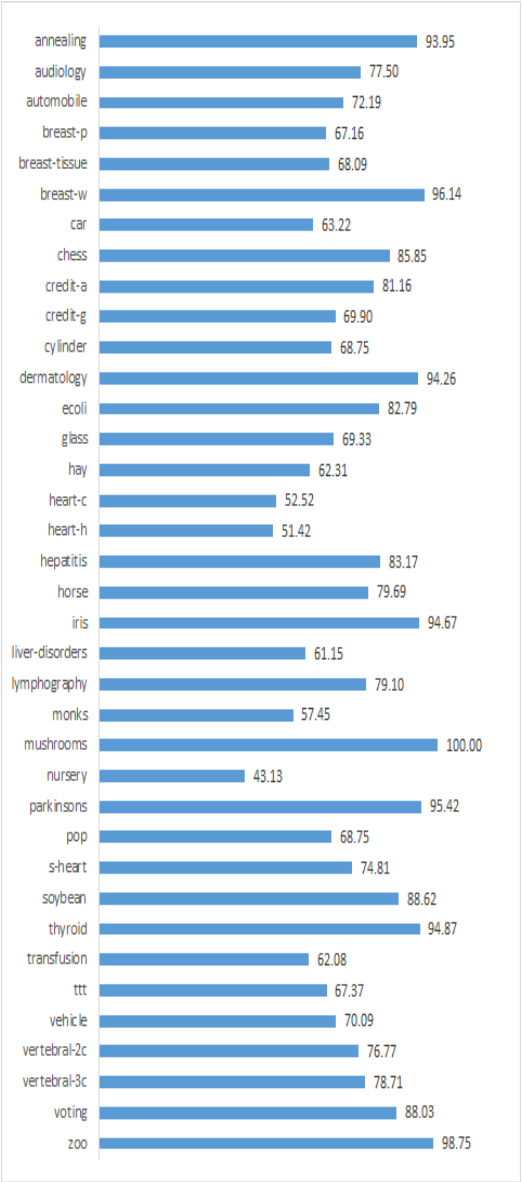


Figure 7.16: C4.5 (J48) - 1-Nearest Neighbor (1-NN) Results

Data Set	Accuracy
annealing	76.69
audiology	79.17
automobile	56.05
breast-p	64.63
breast-tissue	64.36
breast-w	93.33
car	86.43
chess	87.99
credit-a	77.83
credit-g	75.10
cylinder	66.51
dermatology	98.09
ecoli	85.18
glass	48.94
hay	71.54
heart-c	56.40
heart-h	65.37
hepatitis	83.21
horse	78.54
iris	96.00
liver-disorders	54.71
lymphography	81.14
monks	61.27
mushrooms	95.29
nursery	90.34
parkinsons	71.18
pop	72.50
s-heart	83.70
soybean	88.62
thyroid	97.64
transfusion	69.48
ttt	70.74
vehicle	45.04
vertebral-2c	78.71
vertebral-3c	82.26
voting	85.63
zoo	93.75

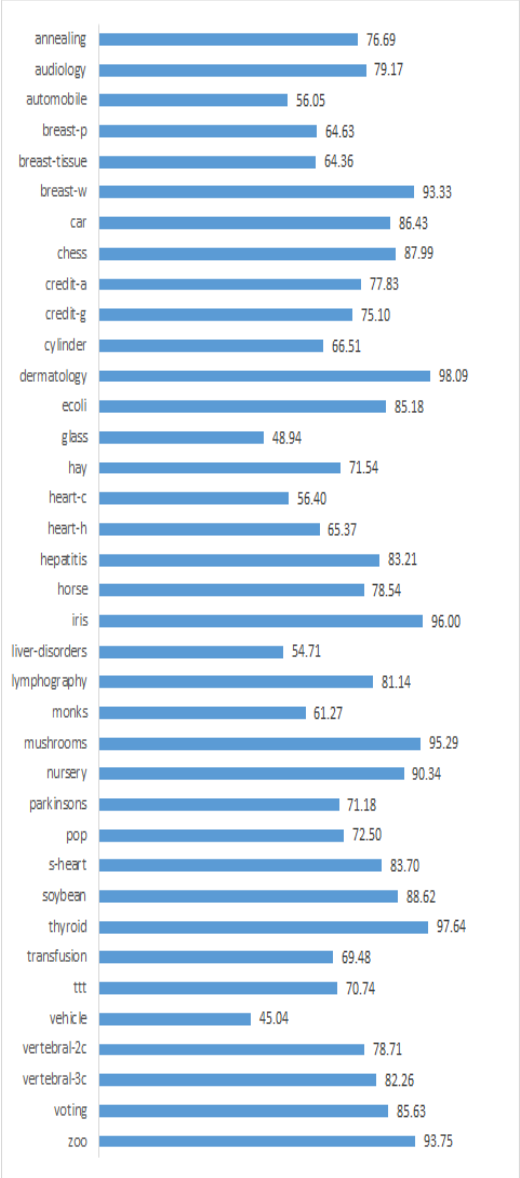


Figure 7.17: C4.5 (J48) - Naïve Bayes (NB) Results

Data Set	Accuracy
annealing	94.16
audiology	75.83
automobile	72.76
breast-p	73.24
breast-tissue	55.82
breast-w	93.32
car	84.97
chess	98.84
credit-a	85.22
credit-g	70.00
cylinder	69.87
dermatology	88.27
ecoli	80.98
glass	65.55
hay	75.38
heart-c	54.47
heart-h	64.07
hepatitis	81.25
horse	83.30
iris	92.00
liver-disorders	65.45
lymphography	79.14
monks	60.18
mushrooms	99.96
nursery	96.41
parkinsons	86.11
pop	70.00
s-heart	77.78
soybean	84.14
thyroid	92.10
transfusion	73.79
ttt	97.37
vehicle	69.86
vertebral-2c	78.71
vertebral-3c	78.71
voting	93.96
zoo	85.00

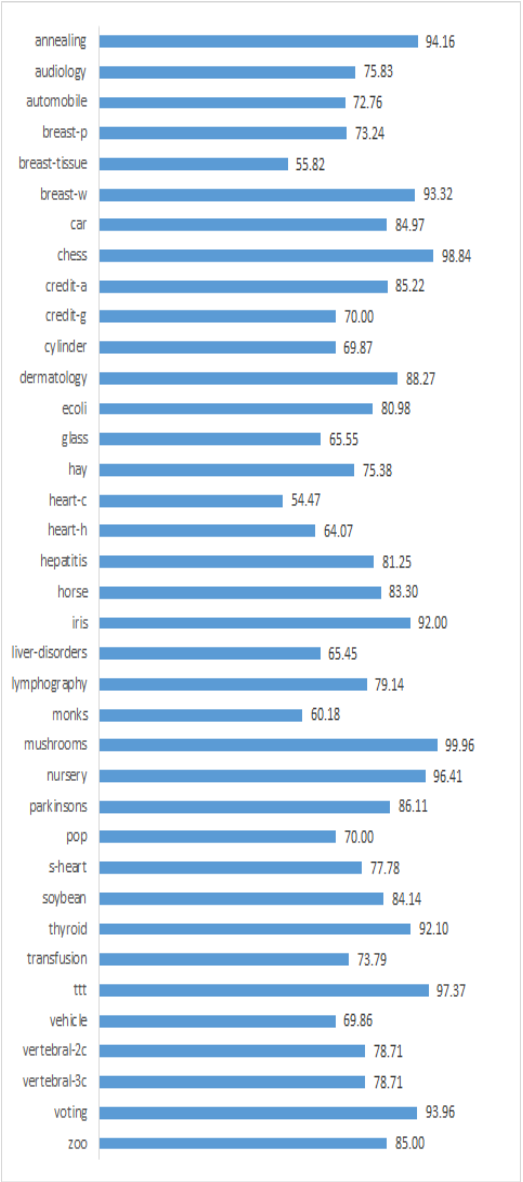


Figure 7.18: C4.5 (J48) - Ripper (JRip) Results

Data Set	Accuracy
annealing	94.98
audiology	85.83
automobile	84.33
breast-p	61.00
breast-tissue	66.36
breast-w	93.67
car	92.87
chess	99.43
credit-a	85.07
credit-g	70.70
cylinder	71.37
dermatology	93.99
ecoli	82.78
glass	68.82
hay	62.31
heart-c	51.44
heart-h	66.67
hepatitis	80.71
horse	83.60
iris	94.00
liver-disorders	64.88
lymphography	77.90
monks	54.55
mushrooms	100.00
nursery	96.78
parkinsons	86.16
pop	66.25
s-heart	76.30
soybean	84.48
thyroid	92.14
transfusion	71.82
ttt	85.79
vehicle	72.35
vertebral-2c	82.26
vertebral-3c	80.65
voting	94.88
zoo	96.25

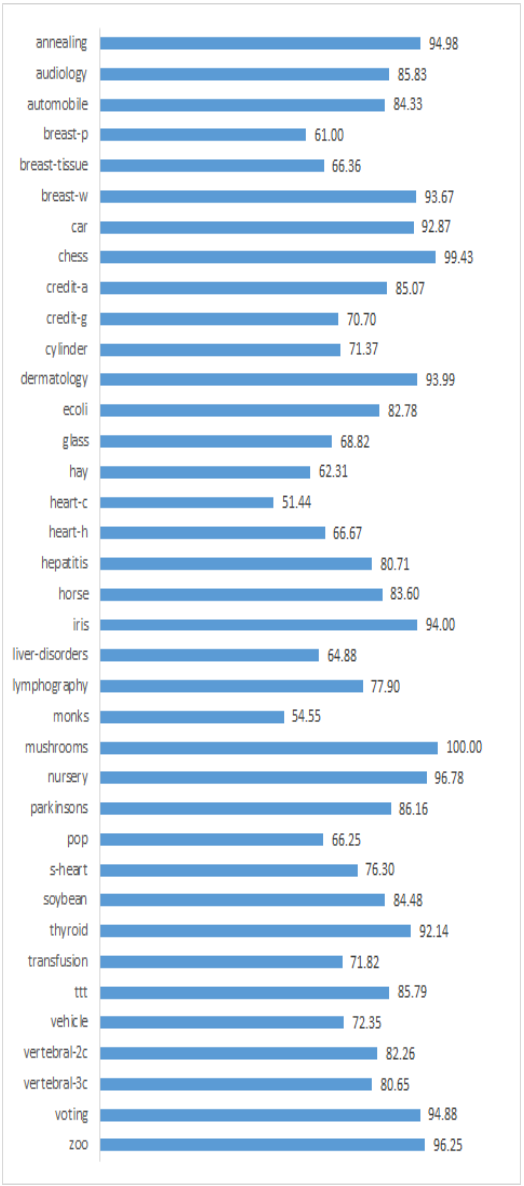


Figure 7.19: C4.5 (J48) - C4.5 (J48) Results

Data Set	Accuracy
annealing	85.98
audiology	89.17
automobile	65.31
breast-p	76.29
breast-tissue	55.82
breast-w	97.54
car	93.33
chess	95.22
credit-a	85.51
credit-g	73.90
cylinder	72.48
dermatology	97.27
ecoli	84.21
glass	55.15
hay	77.69
heart-c	56.75
heart-h	68.12
hepatitis	83.17
horse	80.73
iris	96.00
liver-disorders	57.98
lymphography	85.86
monks	63.64
mushrooms	100.00
nursery	93.05
parkinsons	86.13
pop	70.00
s-heart	85.19
soybean	90.00
thyroid	88.44
transfusion	71.74
ttt	98.42
vehicle	73.88
vertebral-2c	76.77
vertebral-3c	74.19
voting	93.82
zoo	98.75

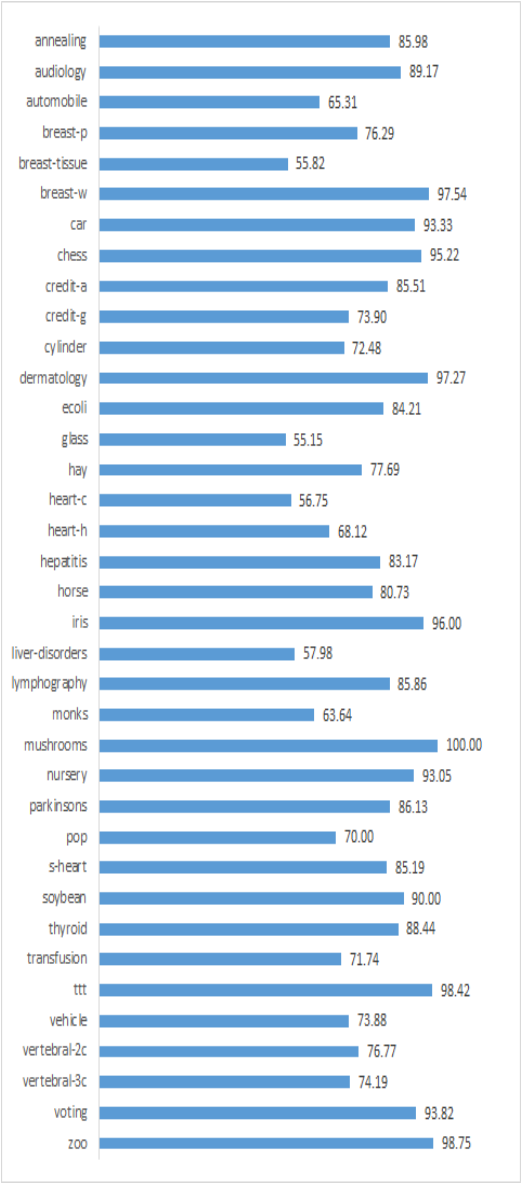


Figure 7.20: C4.5 (J48) - Support Vector Machine (Support) Results

Data Set	Accuracy
annealing	93.95
audiology	80.83
automobile	72.71
breast-p	67.68
breast-tissue	67.00
breast-w	95.26
car	63.27
chess	85.72
credit-a	78.84
credit-g	68.70
cylinder	67.82
dermatology	93.99
ecoli	82.18
glass	66.06
hay	60.00
heart-c	51.86
heart-h	48.53
hepatitis	84.50
horse	81.67
iris	96.00
liver-disorders	61.72
lymphography	79.14
monks	57.64
mushrooms	100.00
nursery	43.36
parkinsons	95.34
pop	70.00
s-heart	74.81
soybean	86.90
thyroid	95.78
transfusion	64.55
ttt	67.89
vehicle	67.72
vertebral-2c	80.97
vertebral-3c	80.32
voting	89.06
zoo	98.75

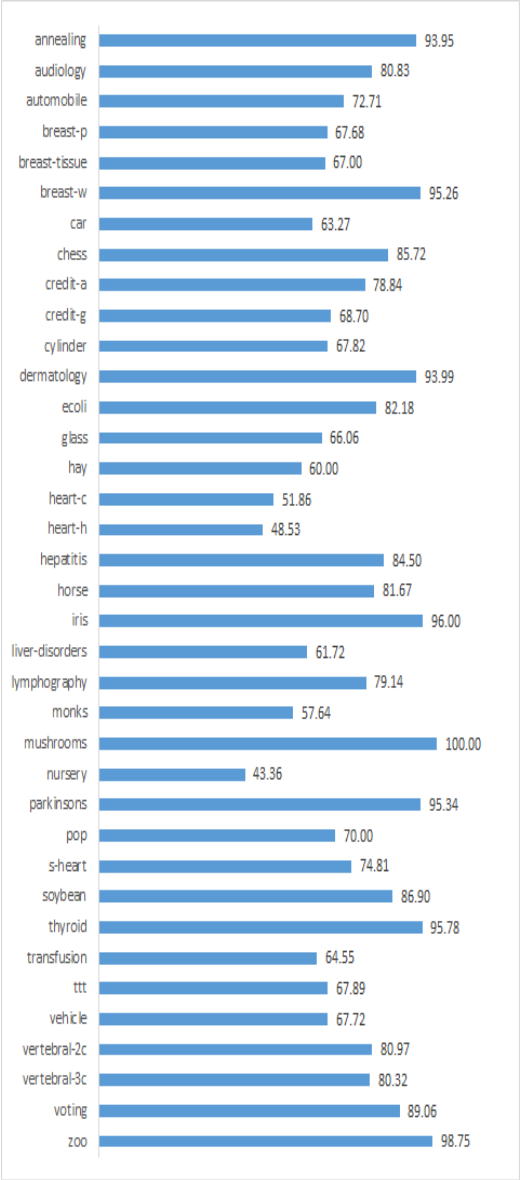


Figure 7.21: Support Vector Machine (SMO) - 1-Nearest Neighbor (1-NN) Results

Data Set	Accuracy
annealing	79.56
audiology	82.50
automobile	57.57
breast-p	66.13
breast-tissue	64.36
breast-w	92.80
car	86.61
chess	87.96
credit-a	76.96
credit-g	73.70
cylinder	69.30
dermatology	98.36
ecoli	85.80
glass	47.77
hay	73.08
heart-c	56.76
heart-h	65.69
hepatitis	83.21
horse	79.14
iris	96.00
liver-disorders	54.77
lymphography	81.90
monks	58.91
mushrooms	95.52
nursery	90.44
parkinsons	70.11
pop	71.25
s-heart	84.44
soybean	89.31
thyroid	97.16
transfusion	69.20
ttt	71.37
vehicle	45.14
vertebral-2c	78.39
vertebral-3c	82.58
voting	85.94
zoo	93.75

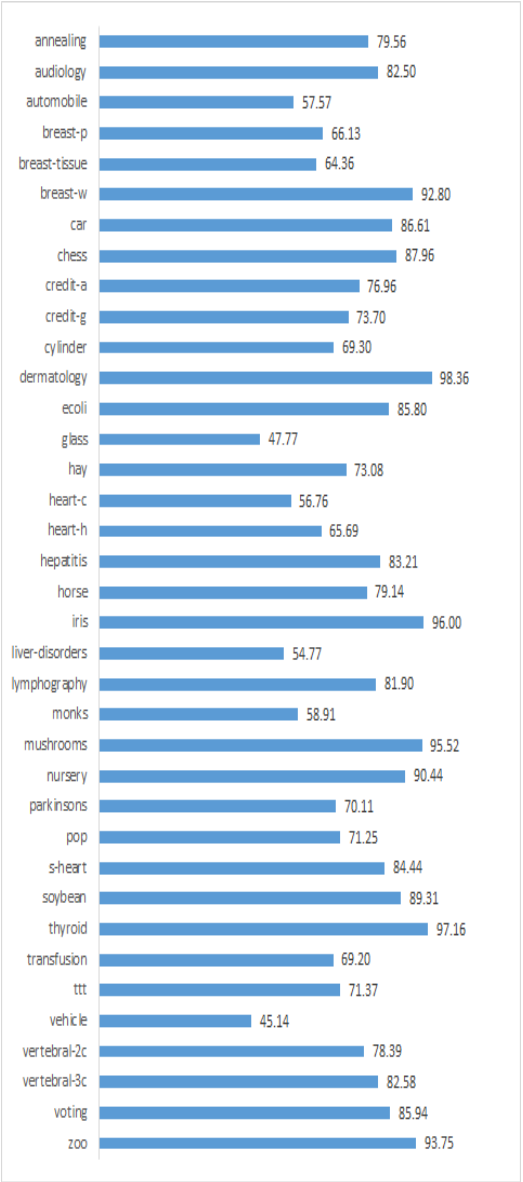


Figure 7.22: Support Vector Machine (SMO) - Naïve Bayes (NB) Results

Data Set	Accuracy
annealing	94.06
audiology	80.83
automobile	71.79
breast-p	71.24
breast-tissue	59.45
breast-w	93.32
car	83.92
chess	99.21
credit-a	86.96
credit-g	71.10
cylinder	65.60
dermatology	87.71
ecoli	77.11
glass	61.91
hay	81.54
heart-c	54.48
heart-h	65.78
hepatitis	82.58
horse	84.48
iris	92.67
liver-disorders	61.43
lymphography	73.05
monks	60.73
mushrooms	99.98
nursery	96.58
parkinsons	87.16
pop	73.75
s-heart	79.26
soybean	83.79
thyroid	92.06
transfusion	73.65
ttt	98.21
vehicle	69.98
vertebral-2c	82.58
vertebral-3c	77.42
voting	92.89
zoo	91.25

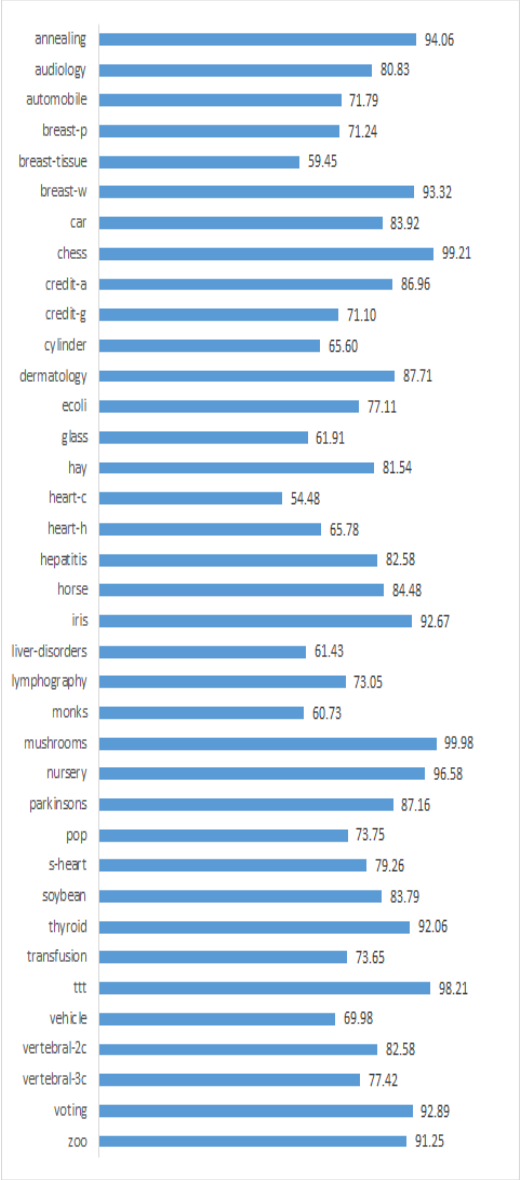


Figure 7.23: Support Vector Machine (SMO) - Ripper (JRip) Results

Data Set	Accuracy
annealing	92.02
audiology	84.17
automobile	76.14
breast-p	64.50
breast-tissue	62.18
breast-w	94.03
car	91.75
chess	99.21
credit-a	86.09
credit-g	70.60
cylinder	71.55
dermatology	93.98
ecoli	78.96
glass	72.18
hay	67.69
heart-c	50.85
heart-h	67.07
hepatitis	79.38
horse	84.17
iris	93.33
liver-disorders	64.71
lymphography	70.33
monks	58.73
mushrooms	100.00
nursery	96.36
parkinsons	84.08
pop	72.50
s-heart	78.89
soybean	83.10
thyroid	91.15
transfusion	72.04
ttt	82.32
vehicle	71.87
vertebral-2c	80.97
vertebral-3c	80.32
voting	93.77
zoo	97.50

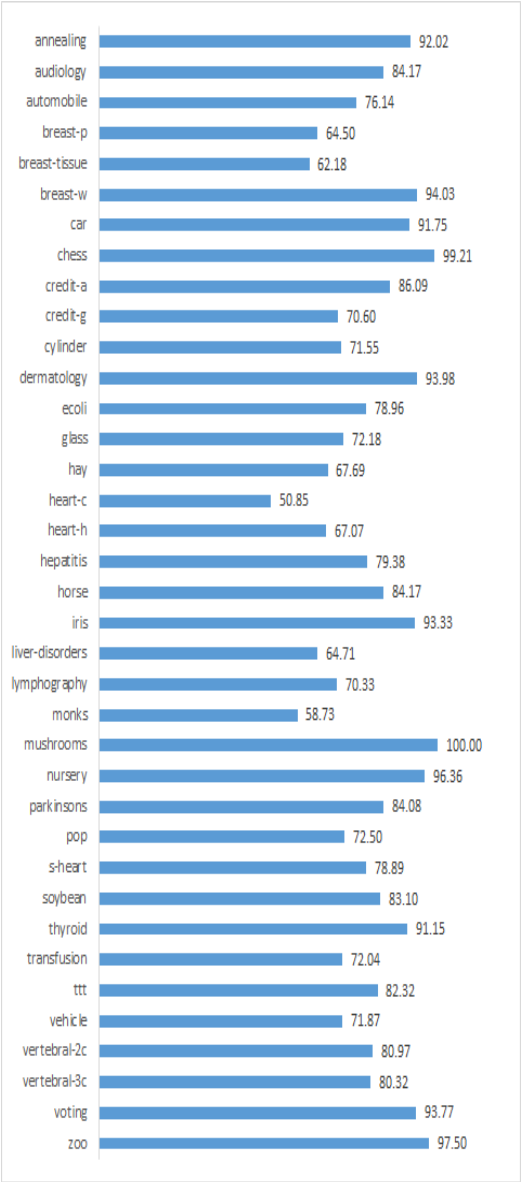


Figure 7.24: Support Vector Machine (SMO) - C4.5 (J48) Results

Data Set	Accuracy
annealing	89.86
audiology	90.83
automobile	72.64
breast-p	74.76
breast-tissue	60.64
breast-w	97.72
car	93.39
chess	96.70
credit-a	84.78
credit-g	74.10
cylinder	73.96
dermatology	97.27
ecoli	86.04
glass	62.27
hay	74.62
heart-c	56.75
heart-h	69.12
hepatitis	83.21
horse	79.56
iris	96.00
liver-disorders	58.56
lymphography	81.19
monks	64.00
mushrooms	100.00
nursery	93.33
parkinsons	87.68
pop	70.00
s-heart	82.59
soybean	88.62
thyroid	88.87
transfusion	72.31
ttt	98.42
vehicle	73.88
vertebral-2c	83.55
vertebral-3c	83.23
voting	95.46
zoo	98.75

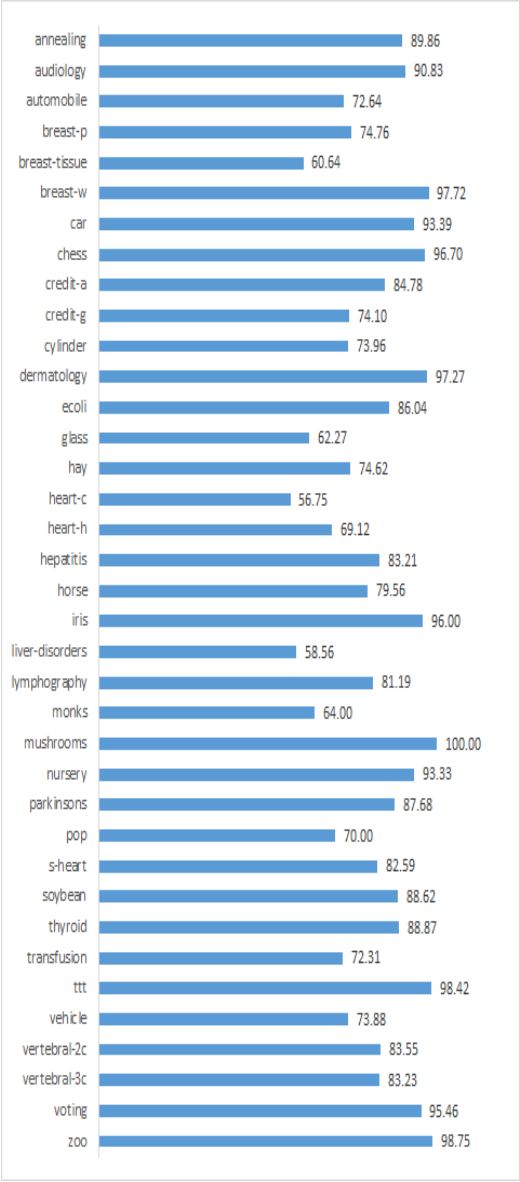


Figure 7.25: Support Vector Machine (SMO) - Support Vector Machine (SMO) Results

Table 7.3: Baseline Predictive Accuracy (%) Results for the Classification Algorithms Without Data Reduction

data set	1-NN	NB	JRip	J48	SMO
annealing	94.40	76.93	94.87	92.46	85.97
audiology	80.00	85.00	79.17	82.50	90.00
automobile	73.64	58.53	68.69	81.36	68.74
breast-p	67.18	64.58	75.79	71.08	76.29
breast-tissue	70.09	67.18	58.55	65.37	59.64
breast-w	95.61	93.50	94.20	94.91	97.90
car	61.81	85.91	87.66	92.98	93.51
chess	84.53	88.05	99.00	99.47	95.72
credit-a	81.02	77.10	85.51	85.80	84.93
credit-g	71.50	75.20	72.20	69.60	73.90
cylinder	68.75	66.70	64.29	74.50	73.22
dermatology	94.53	97.54	88.01	94.00	96.43
ecoli	81.31	85.47	81.86	83.66	83.35
Glass	68.90	49.52	66.54	68.50	58.46
hay	63.08	73.08	79.23	65.39	76.16
heart-c	52.52	56.42	55.15	51.21	56.43
heart-h	46.55	65.37	63.72	66.73	67.77
hepatitis	83.12	83.17	78.13	80.67	85.71
horse	79.05	77.96	83.54	84.75	81.51
iris	95.33	95.33	92.00	94.67	96.67
liver-disorders	60.87	55.89	66.34	64.56	58.56
lymphography	79.10	82.47	79.95	76.38	85.19
monks	56.73	62.73	60.36	61.46	63.64
mushrooms	100.00	95.77	100.00	100.00	100.00
nursery	36.75	90.37	96.93	97.18	93.12
parkinsons	95.42	70.13	88.76	88.16	87.16
pop	71.25	72.50	75.00	75.00	72.50
s-heart	73.33	84.82	78.52	75.56	84.45
soybean	96.70	79.94	94.10	96.59	92.60
thyroid	96.23	96.71	92.53	91.15	88.88
transfusion	61.55	70.07	73.91	73.78	71.75
ttt	67.37	70.63	98.00	85.58	98.42
vehicle	69.26	45.39	68.08	72.93	75.06
vertebral-2c	80.97	78.39	82.58	81.29	79.03
vertebral-3c	78.71	83.55	80.32	78.71	76.78
voting	88.73	85.94	93.66	94.48	92.97
zoo	98.75	93.75	95.00	97.50	98.75

Table 7.4: Average Rankings of Predictive Accuracy

Entry	Rank
SMO-SMO	9.58
SMO	10.66
1-NN-SMO	12.16
J48-SMO	12.19
JRip-JRip	12.36
J48	12.47
JRip-SMO	12.96
1-NN-J48	13.15
NB-SMO	13.91
JRip	14.04
NB-J48	14.38
J48-J48	14.38
JRip-J48	14.51
NB-NB	14.89
SMO-J48	15.66
1-NN-JRip	15.93
SMO-JRip	16.24
NB-JRip	17.31
J48-JRip	17.59
SMO-NB	17.66
SMO-1-NN	17.69
1-NN-1-NN	17.76
JRip-NB	18.05
J48-NB	18.34
NB	18.38
1-NN-NB	18.51
J48-1-NN	18.73
1-NN	18.84
NB-1-NN	19.23
JRip-1-NN	19.42

Table 7.5: Size Reduction (%) Results

data set	1-NN	NB	JRip	J48	SMO
annealing	19.93	20.57	21.13	21.79	19.18
audiology	14.73	16.6	15.16	18.88	16.28
automobile	14.53	18.54	17.94	18.05	18.1
breast-p	15.88	24.47	17.12	18.58	20.6
breast-tissue	15	19.54	17.75	16.59	23.98
breast-w	22.96	26.66	21.19	22.3	28.63
car	16.49	20.2	18.84	19.01	19.04
chess	18.58	20.8	19.58	25.88	19.99
credit-a	16.36	22.35	18.76	20.08	20.79
credit-g	16.2	19.06	19.98	18.02	19.31
cylinder	17.35	18.4	19.62	17.62	18.07
dermatology	22.89	27.23	21.07	23.86	26.32
ecoli	16.9	19.81	18.55	18.35	21.33
glass	13.97	19.02	16.37	16.16	17.58
hay	22.81	26.78	30.01	26.66	23.52
heart-c	12.73	17.42	18.34	16.21	17.45
heart-h	16.98	18.77	18.65	16.72	20.2
hepatitis	18.13	23.6	21.15	16.92	22.44
horse	16.89	20.29	19.66	29.18	21.22
iris	29.98	25.9	30.72	30.51	32.94
liver-disorders	14.75	20.64	18.5	17.68	17.55
lymphography	16.81	22.89	19.97	18.02	25.15
monks	16.48	32.98	17.59	17.93	19.04
mushrooms	33.5	33.37	33.64	33.09	19.46
nursery	17.28	19.87	23.5	19.32	19.67
parkinsons	28.27	27.41	20.74	21.25	26.1
pop	21.1	27.29	32.74	31.55	21.89
s-heart	16.21	21.36	19.3	18.35	22.06
soybean	20.18	20.57	16.75	18.57	21.88
thyroid	28.37	24.45	19.69	23.56	31.21
transfusion	16.25	25.38	21.02	20.59	21.88
ttt	16.2	34.17	21.08	19.19	18.92
vehicle	15.85	17.21	17.89	17.98	17
vertebral-2c	16.74	18.17	17.71	20.39	23.3
vertebral-3c	15.7	21.18	20.14	17.89	20.18
voting	20.99	24.33	25.32	29.08	28.42
zoo	31.49	30.25	21.78	25.47	33.51

Table 7.6: Best Performing Combinations

g	Best Performing h
1-NN	SMO
NB	J48
JRip	JRip
J48	SMO
SMO	SMO

(a) Best for g

h	Best Performing g
1-NN	SMO
NB	NB
JRip	JRip
J48	NB
SMO	SMO

(b) Best for h

for g (the classification algorithm used during training), we can see that SMO showed up as the best performing h for three algorithms: 1-NN, JRip and SMO itself. On the other hand, when analyzing for h (the classification algorithm used during testing), we see that SMO and NB tie for two algorithms each while JRip was the best performing when compared with itself.

Chapter 8

Conclusions and Future Work

In this chapter, we will summarize the conclusions that we have arrived at through our work building ADR-Miner and round things off with suggestions for possible avenues of future research.

8.1 Conclusions

In this dissertation, we introduced ADR-Miner: a data reduction algorithm that utilizes ant colony optimization to perform data reduction via instance selection. Ant colony optimization (ACO) is a search meta-heuristic that was inspired from the foraging behavior observed in ants and that was originally designed to tackle combinatorial optimization problems but has since been extended to tackle multiple other types of problems. Data reduction is the process of removing erroneous, outlier, irrelevant and noisy data prior to being presented to data mining and machine learning. The benefits of data reduction are two fold: remove any data that might be detrimental to the quality of the model being learned and reduce the amount of data to be

processed and maintained by the data mining or machine learning algorithm. One common method of data reduction is instance selection, where the aim of the reduction algorithm is to extract the minimum number of instances from the original data set that can be used to build a data mining model without sacrificing effectiveness. ADR-Miner adapts ACO to perform instance selection with aim of improving the effectiveness of classification algorithm models.

In order to adapt ACO to perform data reduction, we had to go through four steps: adapting the problem into a search space that can be traversed by the ants in ACO, defining the mechanics of the overall meta-heuristic that will be used to guide the ants as they traverse the search space, defining how the ants construct solutions, defining how solution quality is gauged and finally defining how pheromone trails are maintained and updated. In order to translate the problem into a traversable search space, a graph is constructed where each instance in the original set is represented by two in the graph: one node whose selection would imply the inclusion of that instance in the final set and another that implies its exclusion. Ants would traverse the graph selecting one of the two mentioned nodes per instance, till one node from the pair per instance has been selected for all instances in the original set. The instances that have their inclusion component selected will then make it in the reduced set. Each ant will do this, and only the ant with the best performing solution is allowed to drop pheromone on the components that make up its solution. As each ant contemplates a choice between two nodes for a given instance, it will consider two items of information: the amount of pheromone present on the node and the heuristic value associated

with picking that node. The ants iterate till they exhaust an iteration cap or the converge on a solution. Two versions of the ADR-Miner algorithm exist: one that uses the same classification algorithm for both training and testing, one that use separate classification algorithms for each phase of the algorithm.

The first version of the ADR-Miner algorithm was evaluated using three classification algorithms (k-Nearest Neighbor, Ripper and C4.5) with 20 data sets from the UCI Machine Learning repository, and its performance was benchmarked against Iterative Case Filtering (ICF), another data reduction algorithm. Results from this evaluation show that ADR-Miner has produced statistically significant better predictive accuracy at the conventional 0.05 threshold compared to the base classification algorithms when used with the k-Nearest Neighbor and Ripper algorithms. ADR-Miner also produced statistically significant improvements in predictive accuracy when compared to the ICF algorithm when paired with Ripper and C4.5 algorithms.

The extended version of the ADR-Miner algorithm (where two different classification algorithms are used during training and testing) was evaluated using five classification algorithms (Support Vector Machines, Naive-Bayes, k-Nearest Neighbor, Ripper and C4.5) and against 37 data sets from the UCI Machine Learning repository. Here, all possible pairings of the five classification algorithms were tested, and their performance is compared to the five classification algorithms without reduction. Results show that using pairings of classification algorithms in ADR-Miner has generally had a positive impact on the predictive accuracy when compared to using the base classifier algorithms without any reduction.

8.2 Future Work

The ADR-Miner algorithm in its current form can be considered an exploration into adapting ACO to perform instance selection in particular, and data reduction in general. Various improvements can be introduced to the base ADR-Miner algorithm that would present it as a more competent data reducer, and in the next few paragraphs we will discuss examples of such improvements.

One of the improvements that can be introduced to ADR-Miner is to use better inclusion or exclusion heuristics. The current heuristic used with ADR-Miner only gave instances a higher bias towards inclusion, and the pheromone feedback and fitness mechanisms were more relied on for exclusion. To allow for the heuristic side of the transition probability equation 2.1 to play a bigger role, one could use a different heuristic that would provide more apriori information towards the utility of keeping or removing a given instance. Building on the work presented in the "Related Work" chapter, one could suggest using one or more of the following heuristcs:

- Wilson Ratio: This is a ratio that is based on the Wilson Editing algorithm. First, for any given instance, one would retrieve the k nearest neighbors and collate their votes towards the classification of the instance at hand. The ratio then becomes the number of neighbors that correctly classify the instance studied divided by k. The lower the value of the ratio, the more undesirable the instance becomes and vice versa.

- Reach/Cover Ratio: This ratio is derived from the work of Brighton et al [11] on Iterative Case Filtering (ICF). For every instance investigated

we calculate two numbers: The number of instances that have the one investigated as their neighbor (Cover) and the number of instances where the investigated one lies within their neighborhood (Reach). The ratio is then calculated as Reach divided by Cover. The higher the value of the ratio, the more undesirable the instance becomes and vice versa.

- Drop Count: This measure is based on Wilson et al work on the DROP family of algorithms [10]. We start by considering the effect of removing a given instance from a neighborhood of k neighbors, where k is small and odd. We then count the number of instances that would be misclassified if the instance is removed. The higher the this number, the Drop count, the more desirable it is to keep this instance and vice versa.

The items on this list can be chained together to produce heuristics of a higher confidence towards inclusion or exclusion. Of course, the items presented on this list are for example and are by no means a comprehensive list of all possibilities for the development of better heuristics. One could easily envision new heuristics based on the work done in non-stochastic instance reduction.

From the results, one could see that although ADR-Miner proved to be superior in terms of accuracy, it was non-competitive in terms of size reduction. As such, one could improve the ADR-Miner algorithm in terms of size reduction by the tackling the problem as a multi-objctive one: one that seeks a compromise betwen optimizing for accuracy and optimizing for size reduction. As with any other multi-objective optimization problem, doing so with ADR-Miner will produce not just one possible reduction scheme but instead

a set of possible solutions - those lying on the Pareto front or as close a possible an estimation of it. This would provide a user a choice among solutions that are biased towards accuracy while sacrificing size reduction, those that favor size reduction over accuracy or those that achieve a balance between both. To achieve such an objective, one would shy away from aggregating the fitness functions during implementation (as that would only produce one solution lying on the front) and favor approaches based on dominance to get a better estimation of the Pareto front.

So far, ADR-Miner has been solely focused on instance selection and an obvious extension would be to add feature selection to the algorithm, making it a more well-rounded data mining preprocessing task. Feature selection is the process of removing features or attributes from the data set prior to processing with similar objectives as instance selection: improving the accuracy of the models produced and reducing the size of the data to be handled. As with instance selection, the reduction process will be based on both heuristic and stigmergic information that will be used by the ants in the colony. As an example of heuristic information, one could use the information gain ratio (Equations 2.16, 2.17 and 2.18) developed by Claude Shannon during her work on information theory [9]. Using the information ratio, the algorithm would be more biased to removing instances that relatively remove less entropy from the data set. The process of feature selection could be intertwined with instance selection in one of two schemes: as a phased approach, where one process would work on the data set for a phase of the entire run and then switch focus to the other, or as two independent subswarms that occasionally exchange information regarding reduction on

either front of instances and attributes.

If ADR-Miner is to be used in a commercial context, then it would have to deal with the fact that commercial applications deal with data sets that change over time. Running the ADR-Miner algorithm over and over again with every new change to the underlying data set would be computationally expensive and unpractical. This leads us to another possible area of further improvement to the algorithm: to change the algorithm to be dynamic, one that adapts to changes in the data set as they occur and updating the filtered data set accordingly. This would also need us to pair ADR-Miner with a dynamic classifier as it would make no sense if we only update a data mining preprocessing task to be dynamic only to end up retraining the classifier over and over again, and lose any gains in computational expense. To allow the ACO to cope with an ever changing search space, it would have to be redesigned to be biased more towards exploration versus exploitation with pheromone level updates that are triggered by changes in the underlying data set. This can be done by giving higher values to β, as well as adding a constant r_0 in the transition probability function that provides a higher bias towards the heuristic side of the equation. This, of course, is not the only means of adapting the ACO meta-heuristic to cope with dynamic search spaces, and other schemes can be found in [17]. Doing so, as well as keeping an archive of previous best solutions and their components, would allow the ADR-Miner to handle the dynamic nature of some commercial applications of instance selection.

The improvements suggested so far aim to improve the competence of ADR-Miner as a data reducer. These are by no means an exhaustive list,

and one could imagine that an interested researcher can come up with other means of improvements that did not occur to the author.

Appendix - Pairing Results (Alternate View)

Table 8.1: $g - h$ Pairing-based Predictive Accuracy (%) Results for ADR-Miner

	1-NN	1-NN	1-NN	1-NN	1-NN	J48	J48	J48	J48	J48	JRip	JRip	JRip	JRip	NB	NB	NB	NB	SMO	SMO	SMO	SMO	SMO	SMO
Data Set	**1-NN**	**J48**	**JRip**	**NB**	**SMO**	**1-NN**	**J48**	**JRip**	**NB**	**SMO**	**1-NN**	**J48**	**JRip**	**NB**	**SMO**	**1-NN**	**J48**	**JRip**	**NB**	**SMO**	**1-NN**	**J48**	**JRip**	**NB**	**SMO**
annealing	94.41	91.90	93.93	77.73	85.74	93.95	94.98	94.16	76.69	85.98	93.49	90.65	94.97	79.20	85.97	94.18	91.43	93.83	88.71	84.38	93.95	92.02	94.06	79.56	89.86
audiology	80.83	78.33	78.33	85.00	88.33	77.50	85.83	75.83	79.17	89.17	80.83	82.50	84.17	84.17	90.00	77.50	83.33	79.17	85.00	90.83	80.83	84.17	80.83	82.50	90.83
automobile	73.64	80.57	71.71	57.52	71.21	72.19	84.33	72.76	56.05	65.31	72.62	77.05	76.12	58.05	66.69	69.26	77.98	71.24	61.93	68.24	72.71	76.14	71.79	57.57	72.64
breast-p	69.16	73.21	73.68	67.13	76.29	67.16	61.00	73.24	64.63	76.29	67.16	72.16	69.21	66.13	76.29	67.71	69.13	69.61	68.68	76.29	67.68	64.50	71.24	66.13	74.76
breast-tissue	67.18	62.27	58.82	66.18	58.73	68.09	66.36	55.82	64.36	55.82	68.27	62.36	63.45	67.27	58.82	67.27	58.55	62.18	65.45	57.73	67.00	62.18	59.45	64.36	60.64
breast-w	95.79	95.79	93.15	93.33	97.54	96.14	93.67	93.32	93.33	97.54	94.91	91.91	94.20	93.51	97.02	95.79	92.44	92.62	93.86	97.37	95.26	94.03	93.32	92.80	97.72
car	63.33	92.05	83.51	86.26	92.81	63.22	92.87	84.97	86.43	93.33	63.27	91.11	89.59	85.38	92.63	62.98	91.81	85.20	87.54	93.10	63.27	91.75	83.92	86.61	93.39
chess	86.04	99.21	99.06	87.86	95.69	85.85	99.43	98.84	87.99	95.22	85.19	99.37	99.18	87.67	95.22	85.38	99.18	99.09	88.96	95.53	85.72	99.21	99.21	87.96	96.70
credit-a	80.58	85.51	86.09	77.10	84.78	81.16	85.07	85.22	77.83	85.51	80.72	85.51	85.65	76.67	84.64	81.30	85.51	83.77	77.68	84.78	78.84	86.09	86.96	76.96	84.78
credit-g	69.00	71.60	73.60	74.90	74.60	69.90	70.70	70.00	75.10	73.90	69.50	70.60	70.40	74.50	73.50	70.60	71.90	73.80	74.80	73.20	68.70	70.60	71.10	73.70	74.10
cylinder	68.19	73.79	67.10	66.32	72.11	68.75	71.37	69.87	66.51	72.48	68.94	71.90	73.25	67.64	72.48	67.26	73.79	67.82	70.60	74.34	67.82	71.55	65.60	69.30	73.96
dermatology	94.26	94.26	87.95	97.81	95.88	94.26	93.99	88.27	98.09	97.27	93.45	93.97	91.30	98.09	97.55	93.45	92.91	88.57	97.26	96.45	93.99	93.98	87.71	98.36	97.27
ecoli	80.72	83.36	81.86	84.87	83.06	82.79	82.78	80.98	85.18	84.21	79.79	81.89	82.16	84.27	82.75	80.44	81.60	79.80	85.78	81.86	82.18	78.96	77.11	85.80	86.04
glass	68.92	70.79	67.00	48.57	56.58	69.33	68.82	65.55	48.94	55.15	66.04	68.41	70.34	47.95	56.01	68.86	70.34	65.14	59.37	51.75	66.06	72.18	61.91	47.77	62.27
hay	63.08	63.08	79.23	68.46	78.46	62.31	62.31	75.38	71.54	77.69	58.46	66.92	74.62	69.23	77.69	62.31	65.38	82.31	76.92	77.69	60.00	67.69	81.54	73.08	74.62
heart-c	53.83	53.83	55.48	56.77	55.77	52.52	51.44	54.47	56.40	56.75	52.84	51.51	51.52	56.74	56.78	52.81	57.08	54.47	56.44	55.76	51.86	50.85	54.48	56.76	56.75
heart-h	62.43	67.06	63.01	66.05	66.76	51.42	66.67	64.07	65.37	68.12	46.51	66.13	64.74	63.30	65.41	48.57	66.42	62.31	66.04	67.10	48.53	67.07	65.78	65.69	69.12
hepatitis	82.46	79.33	78.75	83.17	83.83	83.17	80.71	81.25	83.21	83.17	80.58	81.96	85.79	82.58	84.46	81.88	79.38	80.67	86.29	82.54	84.50	79.38	82.58	83.21	83.21
horse	78.46	84.15	86.44	78.53	80.99	79.69	83.60	83.30	78.54	80.73	79.04	84.74	85.03	79.41	83.30	79.34	84.42	84.11	79.96	81.78	81.67	84.17	84.48	79.14	79.56
iris	95.33	92.67	94.00	94.67	96.00	94.67	94.00	92.00	96.00	96.00	95.33	95.33	94.67	96.00	94.67	96.00	91.33	90.00	96.00	96.67	96.00	93.33	92.67	96.00	96.00
liver-disorders	62.62	65.50	69.58	54.74	58.28	61.15	64.88	65.45	54.71	57.98	61.13	63.99	68.08	54.46	57.70	61.72	64.04	67.82	64.02	58.56	61.72	64.71	61.43	54.77	58.56
lymphography	83.76	77.10	79.76	82.48	83.10	79.10	77.90	79.14	81.14	85.86	79.05	79.86	78.43	82.48	85.19	77.81	77.14	75.86	84.48	79.81	79.14	70.33	73.05	81.90	81.19
monks	57.09	58.73	57.09	61.09	62.55	57.45	54.55	60.18	61.27	63.64	56.73	59.64	60.00	61.27	63.45	57.64	58.91	62.18	63.45	63.64	57.64	58.73	60.73	58.91	64.00
mushrooms	100.00	100.00	99.96	95.38	100.00	100.00	100.00	99.96	95.29	100.00	100.00	100.00	100.00	95.24	100.00	100.00	100.00	99.96	96.02	100.00	100.00	100.00	99.98	95.52	100.00
nursery	42.91	96.57	96.45	90.40	93.08	43.13	96.78	96.41	90.34	93.05	98.50	97.30	98.50	94.40	97.30	43.28	96.51	96.40	91.00	93.01	43.36	96.36	96.58	90.44	93.33
parkinsons	94.89	81.89	82.11	71.68	85.08	95.42	86.16	86.11	71.18	86.13	93.39	83.11	85.61	71.18	87.18	92.34	81.00	84.61	74.29	84.61	95.34	84.08	87.16	70.11	87.68
pop	70.00	75.00	73.75	68.75	70.00	68.75	66.25	70.00	72.50	70.00	71.25	72.50	72.50	68.75	72.50	71.25	75.00	75.00	68.75	71.25	70.00	72.50	73.75	71.25	70.00
s-heart	74.07	77.04	77.78	84.44	84.44	74.81	76.30	77.78	83.70	85.19	75.56	75.19	78.52	84.07	84.81	74.07	77.78	77.04	83.70	83.33	74.81	78.89	79.26	84.44	82.59
soybean	90.00	83.79	83.79	88.28	90.00	88.62	84.48	84.14	88.62	90.00	86.55	83.10	83.79	87.59	89.31	86.21	85.86	81.38	88.62	88.62	86.90	83.10	83.79	89.31	88.62
thyroid	93.96	91.60	92.53	96.71	87.03	94.87	92.14	92.10	97.64	88.44	95.78	93.48	90.69	96.71	88.42	93.96	93.44	93.01	96.69	87.01	95.78	91.15	92.06	97.16	88.87
transfusion	62.22	73.91	73.99	70.26	71.74	62.08	71.82	73.79	69.48	71.74	63.53	72.41	72.82	70.05	71.94	62.74	72.29	72.89	71.49	71.74	64.55	72.04	73.65	69.20	72.31
ttt	67.37	83.26	97.37	69.16	98.42	67.37	85.79	97.37	70.74	98.42	67.58	82.74	96.95	69.89	98.42	67.58	84.95	97.79	72.63	98.42	67.89	82.32	98.21	71.37	98.42
vehicle	67.73	69.39	70.58	45.04	74.94	70.09	72.35	69.86	45.04	73.88	68.08	71.99	71.15	44.69	73.99	69.26	72.11	67.51	50.95	74.23	67.72	71.87	69.98	45.14	73.88
vertebral-2c	80.32	81.94	80.65	78.06	73.55	76.77	82.26	78.71	78.71	76.77	77.10	80.32	80.65	78.39	77.74	81.29	80.00	77.74	71.29	80.97	80.97	82.58	78.39	83.55	
vertebral-3c	79.68	81.29	79.68	81.94	80.65	78.71	80.65	78.71	82.26	74.19	78.39	80.32	79.68	82.90	75.81	80.32	79.35	80.32	82.58	76.45	80.32	80.32	77.42	82.58	83.23
voting	88.99	94.48	92.89	85.94	94.26	88.03	94.88	93.96	85.63	93.82	89.41	94.25	95.55	85.98	93.63	88.64	93.68	93.68	85.98	93.91	89.06	93.77	92.89	85.94	95.46
zoo	98.75	98.75	97.50	93.75	98.75	98.75	96.25	85.00	93.75	98.75	98.75	97.50	93.75	96.25	96.25	98.75	98.75	91.25	96.25	96.25	98.75	97.50	91.25	93.75	98.75

Bibliography

[1] D. L. Wilson, "Asymptotic properties of nearest neighbor rules using edited data," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 2, no. 3, pp. 408–421, 1972.

[2] I. Tomek, "An experiment with the edited nearest-neighbor rule," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 6, no. 6, pp. 448–452, 1976.

[3] D. Aha, D. Kibler, and M. Albert, "Instance-based learning algorithms," *Machine Learning*, vol. 6, no. 1, pp. 37–66, 1991.

[4] M. Dorigo, "Optimization, learning and natural algorithms (in italian)," PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1992.

[5] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[6] T. Stützle and H. Hoos, "Max-min ant system and local search for the traveling salesman problem," *Evolutionary Computation, IEEE International Conference on*, pp. 309–314, 1997.

[7] M. Dorigo, G. M. Caro, and L. M. Gambardella, "Ant algorithms for discrete optimization," *Artificial Life*, vol. 5, no. 2, pp. 137–172, 1999.

[8] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge, UK: Cambridge University Press, 2000.

[9] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, 2nd. San Francisco, CA, USA: Morgan Kaufmann, 2000.

[10] D. Wilson and T. Martinez, "Reduction techniques for instance-based learning algorithms," *Machine Learning*, vol. 38, no. 3, pp. 257–286, 2000.

[11] H. Brighton and C. Mellish, "Advances in instance selection for instance-based learning algorithms," *Data Mining and Knowledge Discovery*, vol. 6, no. 2, pp. 153–172, 2002.

[12] B. Liu, H. Abbass, and B. McKay, "Density-based heuristic for rule discovery with ant-miner," *6th Australasia-Japan Joint Workshop on Intelligent and Evolutionary Systems*, pp. 180–184, 2002.

[13] R. S. Parpinelli, H. S. Lopes, and A. A. Freitas, "Data mining with an ant colony optimization algorithm," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 4, pp. 321–332, 2002.

[14] T. Stützle and M. Dorigo, "A short convergence proof for a class of ant colony optimization algorithms," *Evolutionary Computation, IEEE Transactions on*, vol. 6, no. 4, pp. 358–365, Aug. 2002.

[15] M. Dorigo and T. Stützle, "The ant colony optimization metaheuristic: algorithms, applications, and advances," in *Handbook of Metaheuristics*. New York City, NY, USA: Springer, 2003, ch. 9.

[16] M. Dorigo and T. Stützle, *Ant Colony Optimization*. Cambridge, MA, USA: MIT Press, 2004.

[17] A. P. Engelbrecht, *Fundamentals of Computational Swarm Intelligence*. Chichester, West Sussex, England, UK: Wiley, 2005.

[18] K. Socha and C. Blum, "Training feed-forward neural networks with ant colony optimization: an application to pattern classification," in *5th International Conference on Hybrid Intelligent Systems (HIS '05)*, Washington, DC, USA: IEEE Computer Society, 2005, pp. 233–238.

[19] P. N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*, 1st. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005.

[20] A. Chan and A. A. Freitas, "A new classification-rule pruning procedure for an ant colony algorithm," in *Artificial Evolution*, ser. Lecture Notes in Computer Science, vol. 3871, Berlin, Heidelberg: Springer, 2006, pp. 25–36.

[21] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine Learning Research*, vol. 1, no. 7, pp. 1–30, 2006.

[22] Y. P. Liu, M. G. Wu, and J. X. Qian, "Evolving neural networks using the hybrid of ant colony optimization and bp algorithms," in *3rd International Conference on Advances in Neural Networks (ISNN'06)*, Berlin, Heidelberg: Springer-Verlag, 2006, pp. 714–722.

[23] A. Asuncion and D. Newman. (2007). Uci machine learning repository, [Online]. Available: `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

[24] K. Socha and C. Blum, "An ant colony optimization algorithm for continuous optimization: application to feed-forward neural network training," *Neural Computing & Applications*, vol. 16, pp. 235–247, 2007.

[25] S. S. Haykin, *Neural Networks and Learning Machines*, 3rd. Upper Saddle River, New Jersey, USA: Prentice Hall, 2008.

[26] F. Otero, A. A. Freitas, and C. Johnson, "Cant-miner: an ant colony classification algorithm to cope with continuous attributes," in *Ant Colony Optimization and Swarm Intelligence (ANTS'08)*, ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2008, pp. 48–59.

[27] K. Socha and M. Dorigo, "Ant colony optimization for continuous domains," *European Journal of Operational Research*, vol. 185, pp. 1155–1173, 2008.

[28] F. Otero, A. A. Freitas, and C. Johnson, "Handling continuous attributes in ant colony classification algorithms," in *IEEE Symposium on Computational Intelligence in Data Mining (CIDM 2009)*, New York, NY, USA: IEEE Press, 2009, pp. 225–231.

[29] U. Boryczka and J. Kozak, "Ant colony decision trees – a new method for constructing decision trees based on ant colony optimization," in *Computational Collective Intelligence. Technologies and Applications*, vol. 6421, Berlin, Heidelberg: Springer, 2010, pp. 373–382.

[30] A. A. Freitas, D. Wieser, and R. Apweiler, "On the importance of comprehensible classification models for protein function prediction," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 7, no. 1, pp. 172–182, 2010.

[31] M. Gendreau and Y. Potvin, "Ant colony optimization: overview and recent advances," in *Handbook of Metaheuristics*, 2nd. New York City, NY, USA: Springer US, 2010, pp. 227–263.

[32] ——, *Handbook of Metaheuristics*, 2nd. New York City, NY, USA: Springer US, 2010.

[33] K. Salama and A. Abdelbar, "Extensions to the ant-miner classification rule discovery algorithm," in *7th International Conference on Swarm Intelligence (ANTS'10)*, ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2010, pp. 167–178.

[34] I. H. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd. San Francisco, CA, USA: Morgan Kaufmann, 2010.

[35] U. Boryczka and J. Kozak, "An adaptive discretization in the acdt algorithm for continuous attributes," in *Computational Collective Intelligence. Technologies and Applications*, vol. 6923, Berlin, Heidelberg: Springer, 2011, pp. 475–484.

[36] J. Frijters. (2011). Ikvm.net project, [Online]. Available: `http://www.ikvm.net/`.

[37] J. Huysmans, K. Dejaeger, C. Mues, J. Vanthienen, and B. Baesens, "An empirical evaluation of the comprehensibility of decision table,

tree and rule based predictive models," *Decision Support Systems*, vol. 51, no. 1, pp. 141–154, 2011.

[38] D. Martens, B. Baesens, and T. Fawcett, "Editorial survey: swarm intelligence for data mining," *Machine Learning*, vol. 82, no. 1, pp. 1–42, 2011.

[39] K. Salama, A. Abdelbar, and A. A. Freitas, "Multiple pheromone types and other extensions to the ant-miner classification rule discovery algorithm," *Swarm Intelligence*, vol. 5, no. 3-4, pp. 149–182, 2011.

[40] A. A. Freitas, "Comprehensible classification models: a position paper," *ACM SIGKDD Explorations*, vol. 15, no. 1, pp. 1–10, 2013.

[41] K. Salama, A. Abdelbar, F. Otero, and A. A. Freitas, "Utilizing multiple pheromones in an ant-based algorithm for continuous-attribute classification rule discovery," *Applied Soft Computing*, vol. 13, no. 1, pp. 667–675, 2013.

[42] K. Salama and A. A. Freitas, "Abc-miner+: constructing markov blanket classifiers with ant colony algorithms," *Memetic Computing*, vol. 6, no. 3, pp. 183–206, 2013.

[43] ——, "Learning bayesian network classifiers using ant colony optimization," *Swarm Intelligence*, vol. 7, no. 2-3, pp. 229–254, 2013.

[44] K. M. Salama and A. M. Abdelbar, "A novel ant colony algorithm for building neural network topologies," in *9th International Conference on Swarm Intelligence (ANTS'14)*, ser. Lecture Notes in Computer Science, vol. 8667, Berlin, Heidelberg: Springer, 2014, pp. 1–12.

[45] K. Salama and A. A. Freitas, "Ant colony algorithms for constructing bayesian multi-net classifiers," *Intelligent Data Analysis*, 2014.