A JAVA IMPLEMENTATION OF THE BAYESIAN DATA REDUCTION ALGORITHM

A Project

Presented to the

Faculty of

California State University,

San Bernardino

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

---

by

Daniel Lee Patterson

December 2007

A JAVA IMPLEMENTATION OF THE BAYESIAN DATA REDUCTION ALGORITHM

A Project

Presented to the

Faculty of

California State University,

San Bernardino

---

by

Daniel Lee Patterson

December 2007

Approved by:

David Turner, Advisor, Computer Science	Date
Arturo Concepcion	
Keith Evan Schubert	

© 2007 Daniel Lee Patterson

ABSTRACT

This project concerns a Java implementation of the Bayesian Data Reduction Algorithm (BDRA). The algorithm was developed by Dr. Robert Lynch of the Naval Undersea Warfare Center and Dr. Peter Willett of the University of Connecticut. The minimal setup of the algorithm and its good prediction rate showed it had the potential to be developed into a commercial product. The Office of Technology Transfer (OTTC) at California State University funded this project. with the computer science department to develop a desktop application of the BDRA.

The main goal of the project was to develop an accurate working implementation of the algorithm. We aimed to make the implementation efficient in both space and time and also allow for it to be easily extended to be run on a distributed environment. This first iterations only interface is the command line, but later iterations will develop a graphical user interface or Web front end, depending on the needs of clients.

The original code for the algorithm was developed in MATLAB by Dr. Lynch and Dr. Peter Willett. We implemented the BDRA in Java using an object oriented approach. Dr. Lynch used data sets from the University of California, Irvine Machine Learning database for his research. We compared fifteen of these data sets with our implementation. Our results matched Dr. Lynch's results on all but two of these data sets.

ACKNOWLEDGEMENTS

I would like to acknowledge the Office of Technology Transfer for their support of this project. I would also like to acknowledge Dr. Turner, Dr. Concepcion, and Dr. Schubert for their guidance.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

1.1 Introduction

Chapter one presents an overview of the context, motivation and significance of the work documented in this project. The purpose of the project is discussed, followed by a history of the problem and the significance of the work.

1.2 Purpose

The purpose of this project was to build a working implementation of the Bayesian Data Reduction Algorithm (BDRA). This algorithm was developed by Dr. Robert Lynch while working at the Naval Undersea Warfare Center as his PhD dissertation. This method uses Bayesian statistics to select the most relevant pieces of information in the data for the purpose of classification of data.

Dr. Lynch and his advisor Dr. Peter Willet, developed code in MATLAB to implement the algorithm. Because of the algorithm's good running time and accurate classification results it showed the potential for commercialization. I and several other members of the Department of Computer Science at California State University, San Bernardino were asked to develop a desktop application that would be marketable to institutions in the private sector. The system that uses the BDRA is called the Automatic Data Reduction System (ADRS). This document outlines the main algorithm,

its variations and its design and architecture as implemented using Java.

The project was supported by the Office of Technology Transfer under two grants. The first was for $25,000 under OTTC project #60657. This phase of the project ran from April 1, 2006 until July 31, 2006. The focus of this stage was to develop a working implementation of the BDRA. The second was OTTC project #20089 which funded the project an additional $18,000. This second phase ran from from March 1, 2007 until July 31, 2007. The main focus of the second stage was to develop the ADRS into a commercial system.

1.3 Context of the Problem

The basic idea behind classification lends itself to commonsense. We use our previous experiences to judge what decision to make when faced with a new situation. Decisions are ubiquitous in every field of study. A surgeon will base his/her decision to operate based on previous experiences that closely resemble the present case. A loan officer will decide to grant a loan based on metrics determined by many previous applicants. Using numerical methods to help automate these types of decisions is the goal of classification.

Statistical classification is widely used throughout many disciplines. Many methods have been used to classify data, each with their own strengths and weaknesses: artificial neural networks, vector machines, and nearest neighbor algorithms to name a few. The Bayesian Data Reduction Algorithm (BDRA) falls in the category of Bayesian classifiers.

1.4 History

Statistical classification can roughly be separated into three approaches: statistical, machine learning, and neural network [3]. These boundaries are inexact and often principles from one area may be used in another.

One of the earliest examples of statistical classification is Fisher's linear discriminants. More modern approaches use joint probabilities of features to provide classification rules. Statistical classification is usually described by an underlying probability model.

Decision trees, inductive logic procedures and genetic algorithms fall under the category of machine learning. The goal of such approaches is to use logical operations which learn from previous examples. This technique is called training and is common in the other categories of classification as well.

Artificial neural networks (ANN) are designed to represent interactions between neurons in the brain. They consist of layers of nodes, each producing a non-linear function that can feed into another node. This node can be farther in the connection or may feed back into a previous node. Like other classification techniques, neural nets train on a subset of the data. This data is usually preprocessed so that only a subset of features are used. A common technique for this preprocessing is primary component analysis.

Some of the most important issues with statistical classification are accuracy, speed, comprehensibility, and time to learn. It makes sense that we would like our classifications to be as accurate as possible. But a slight decline in accuracy might be traded off for noticeable gains in the other mentioned areas. Accuracy is usually

represented by the number of correct classifications made by the classifier.

Since real world data sets can be quite large and computationally demanding, inefficient algorithms could take very long times to complete. Since the ultimate goal is to have the classification model be used by humans, it is better to have results and patterns that are easily understood by people.

The time it takes an algorithm to learn is also important. Neural network, which requires some overhead and preprocessing, often take more time to complete. Nearest neighbor algorithms take no time to train since they classify simply by looking for the closest match in data, but may not be as accurate in difficult cases.

1.5 Review of the Literature

Data can have values that are either categorical (nominal) or continuous (ordinal). BDRA only operates on categorical data. Any continuously valued attributes are discretized before the algorithm begins. There are several approaches to discretization. Liu and Setiono [4] discuss using discretization to help select relevent features using the Chi-Squared distribution. BDRA uses equal frequency intervals where equal numbers of observations are placed in each quantization level. Another approach uses equal width intervals.

A common strategy for classification algorithms is to use only a subset of features for classification. Searching the entire feature set is only feasible for small problems. A large set of features leads to the "curse of dimensionality" [2]. The search for the optimal subset of features is NP-hard [5]. Much research has been done to find optimal or near optimal solutions by only searching a small area of the solution space.

Such techniques incorporate two main features - a search engine and a evaluation function [1].

The most basic search techniques are forward and backward sequential search. The backward search starts with an empty set of features and begins adding a single feature to the subset. After finding the best single feature, it then adds one more from the remaining features. At each stage it adds the feature that best improves classification. Similarly to the backward search, the forward search starts with all features and removes each singly and commits each change.

An algorithm is greedy in the sense that if at a stage in the process it has inserted feature f_n it will never look at solutions without f_n, although the optimal solution may not contain f_n. Greedy approaches often settle on local minimum. Aside from shortening the computational time, such feature reduction can improve classification accuracy [5].

At each iteration of the search the present subset of features must be compared to previous and future subsets. Several works are cited as references to Dr. Lynch's work as background to the development to the BDRA. The BDRA is based on the combined Bayes test which can be referenced in [7].

The BDRA is unique in that not only does it remove features, but it can transform the quantization of the features that remain. The BDRA only deals with discrete data, and the number of levels that the data has been quantized to are referred to as bins. The BDRA removes thresholds between these bins to improve the classification rate.

1.6 Document Organization

Chapter two outlines the software requirements for ADRS which incorporates BDRA as its classification engine. Chapter three discusses the design of the software. Chapter four discusses the testing of the algorithm and compares the results to the previous implementation of the BDRA developed by Dr. Lynch. Chapter five gives directions for future work.

2. REQUIREMENTS

2.1 Introduction

Chapter two consists of the software requirements specifications following the IEEE standard 830-1998. It provides a detailed view of the functions the Automatic Data Reduction Algorithm will provide to the end user.

2.2 Purpose

The purpose of the requirements section is to provide a clear picture about the specifications and functions of the Automatic Data Reduction System (ADRS). The intended audience is the Department of Computer Science at California State University, San Bernardino and any students continuing the project.

The purpose of ADRS is to create a working engine for the Bayesian Data Reduction Algorithm (BDRA) that can be extended for commercial use. The BDRA trains on historical classified data and produces a classifier which can be used to make predictions on new unclassified data. This type of classification has uses throughout many fields such as medical research and financial markets.

2.3 Scope

This is the first iteration of ADRS and is an independent system. ADRS can be installed and operated on a desktop computer, or the user can connect to the ADRS server machine at California State University, San Bernardino (CSUSB). The interfaces for these two modes are similar.

The main goal of this iteration is to produce a working engine for the ADRS based on the Bayesian Data Reduction Algorithm. Its only interface will be the command line. It is designed to be flexible and extendible. It will be easily adaptable so that later iterations can be enhanced to include a GUI or Web interface. It could also be extended into a distributed system.

2.4 Definitions, Acronyms and Abbreviations

ADRS Automatic Data Reduction System.

BDRA Bayesian Data Reduction Algorithm

Forward Search A search that begins with no feature information and adds feature information in at each iteration.

Backward Search A search that begins with all of the feature information and removes feature information at each iteration.

Unclassified Data Data in which the class is unknown and must be predicted.

Classified Data Data which the class is know. This can be used for training.

Feature Reduction The process that removes the thresholds of features to reduce the number of bins.

Training Data The subset of classified data used to train ADRS

Test Data The subset of classified data used by ADRS to test the results of the training

Quantization The process of taking continuously valued data and separating it into discrete categories

Classifier The set of rules used to take an unclassified observation an make a prediction as to its class.

PDF Portable Document Format.

2.5 Overview

The SRS is organized in such a way to provide an outline for the functional and specification requirements for the ADRS. The remaining two parts provides a clear view of the product functionalities and report outputs.

2.6 Overall Description

ADRS uses the Bayesian Data Reduction Algorithm to classify data. The BDRA uses historical, classified data to train a classifier. The classifier is used to make predictions for new unclassified data. An example would be a medical researcher who is attempting to diagnose patients with schizophrenia. After taking data on several known cases, they would use the ADRS to analyze the data. ADRS will find patterns

in the data that distinguish between those with the condition and those without. The researcher can use these results to help classify new patients whose condition is unknown.

ADRS has three modes of operation: evaluation, training, and classify. ADRS is designed for users who have a set of classified data they wish to analyze. The analysis will help them make predictions for new unclassified observations they will encounter in their field of study. In this scenario all three modes will be explained.

The user begins in evaluation mode which will allow them to adjust the various parameters of the BDRA to optimize classification success. The adjustment of these parameters will lead to different error rates. Different data sets have different characteristics so the default settings may not provide the best results.

After the user finds parameters that are best suited for this data set, they select to run the ADRS in training mode. The parameters found in evaluation mode are used as inputs and a classifier is produced that can be used to make predictions for new unclassified observations.

Once the user has new data that needs to be classified, they will operate in classify mode. The user will input the unclassified data. The classifier produced in training mode will be used to predict the class for each data type.

2.7 Product Perspective

ADRS is an independent system. It provides two methods for the user to interact with the system - through a Secure Shell (SSH) connection to a server hosted at CSUSB, or on an individual machine where the ADRS has been installed. Figure 2.1

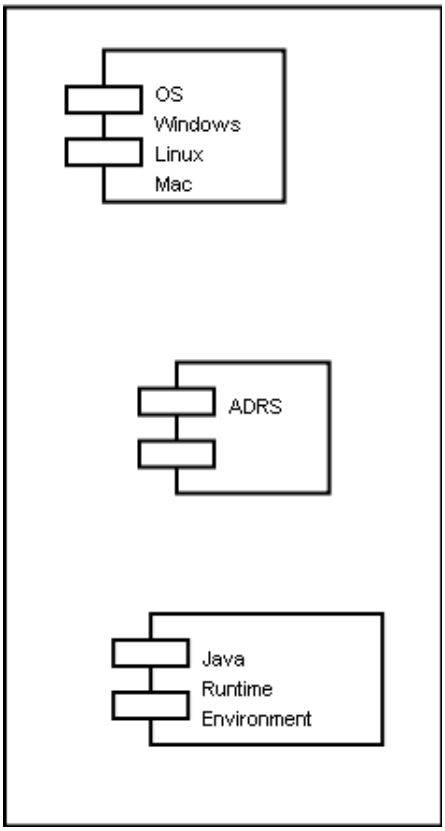

Fig. 2.1: Stand Alone Deployment Diagram.

shows the deployment diagram stand alone system. Figure 2.2 shows the deployment diagram server system.

2.7.1 System Interfaces

The system will have two available interfaces. If the user is connecting to the server, the system interface will be a secure shell connection, after a connection has been established with the server, then the user operates through the servers command line. If the user has the ADRS installed on their personal desktop then they will interface with the system through the command line.

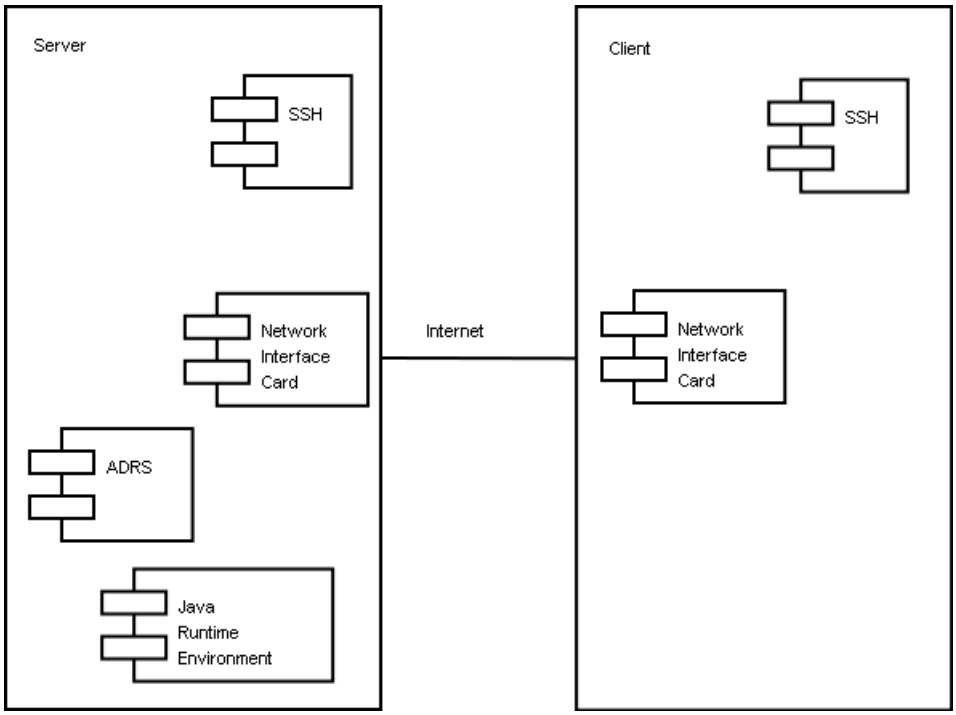

Fig. 2.2: Server Deployment Diagram.

2.7.2 User Interfaces

The user has three steps to using ADRS. The user must first preprocess the data to format it to be read by the system. This requires using an external text editor to edit the data. The user then uses the command lines to analyze the data in one of the systems three modes. The user then views or prints the pdf reports.

2.7.3 Hardware Interfaces

If the user is connecting to the server machine a network interface card (NIC) is needed.

2.7.4 Software Interfaces

Because the ADRS is written in Java, it will run on Windows, Mac OS and Linux operating systems. In each case the Java runtime environment version 1.5 or higher must be installed. If the user wishes to use the server machine, then they will need to install an SSH client such as PutTy or openSSH. To review and print reports, the user's machine must have a PDF viewer installed such as Adobe Acrobat Reader or Foxit Reader.

2.7.5 Communication Interfaces

The stand alone system requires no communication interface. If the user wishes to run the ADRS on the server machine at CSUSB then an Internet connection is needed.

2.7.6 Memory

A minimum of 512 MB RAM is needed.

2.7.7 Operations

If the ADRS is installed on the user's desktop machine, then it will be continuously available. Connections to the server machine must be done individually. Although two users could run the separate instances of the ADRS in separate threads there would be a serious performance decline. The use of the ADRS server machine must be used with a schedule so that two users do not access the ADRS at the same time.

2.8 Product Functions

The use case diagram in Figure 2.3 shows the user and tasks which they can perform. The three modes of operation are selected by the user based on command line inputs. Each of the three modes provides a different output in PDF format. The user can view these reports or choose to print them. The output of training mode also includes a serialized Java object called a classifier.

The three modes of operation have an order in which they are intended to be used. Figure 2.4 shows a flow chart of the three modes and the inputs and outputs

Evaluation mode The user inputs various parameters that can affect the classification error and looks at the resulting error. They continue to make adjustments until they feel confident they have found the optimal settings. The output of each run is a PDF report showing statistics of that run.

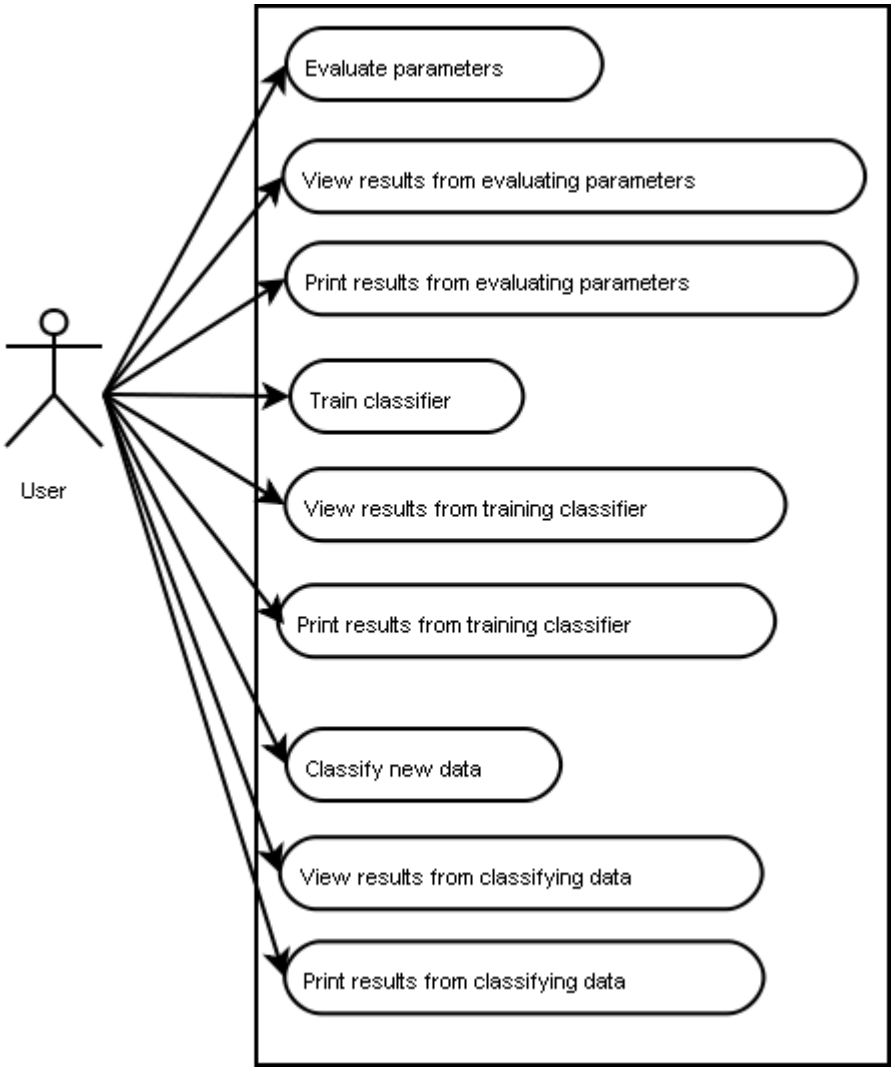

Fig. 2.3: Use Case Diagram for the Automatic Data Reduction System.

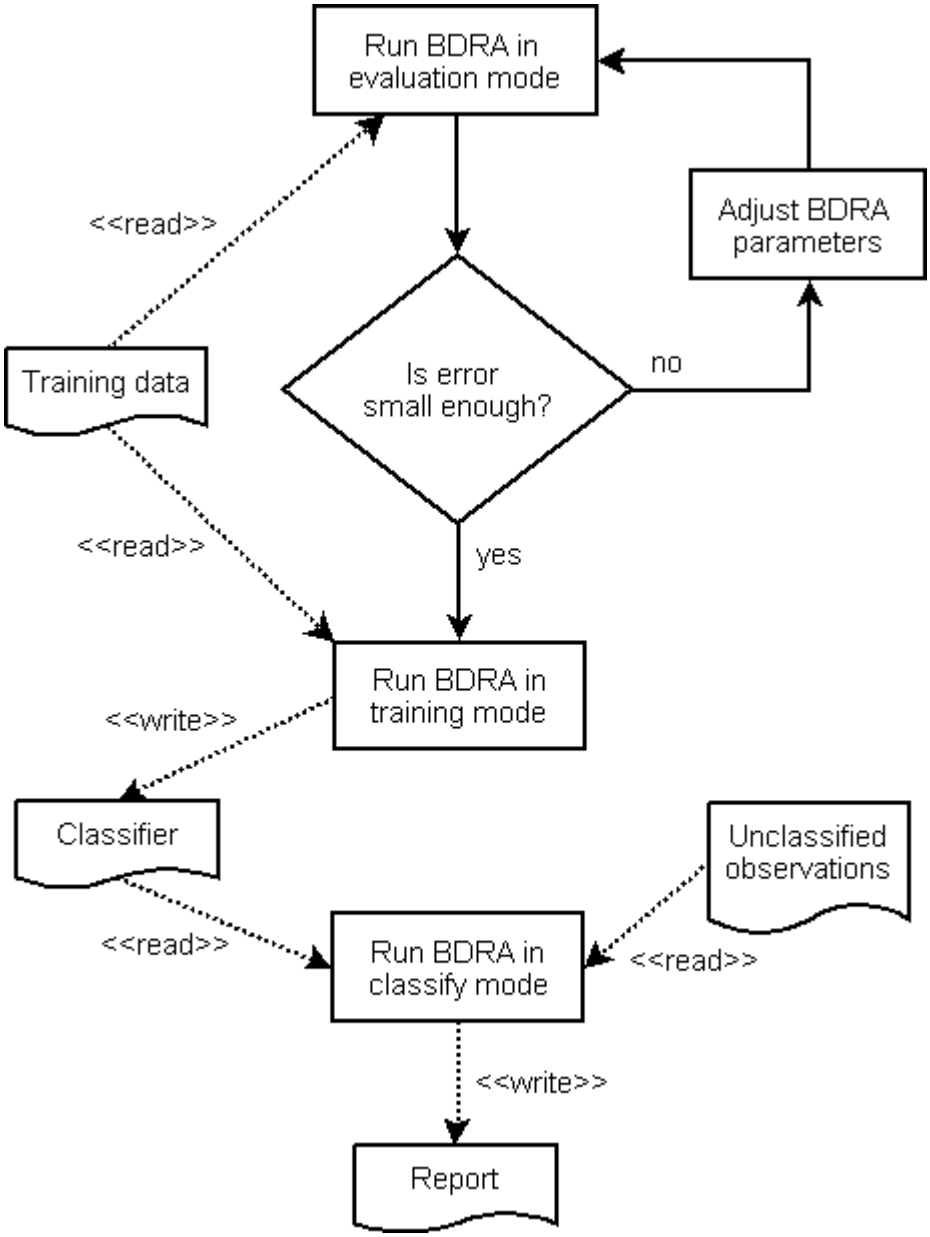

Fig. 2.4: Intended Order of the Three User Modes

Training mode The user inputs the optimal settings they found in evaluation mode, except now the entire data set is used to produce a classifier. The output is a PDF report showing the training error and a serialized classifier object that can be opened in classify unclassified observation mode.

Classify mode The user inputs a data file with unclassified observations and the name of a classifier that was produced in training mode. The output is a list of predicted classes for each observation in the data.

2.8.1 User Characteristics

The ADRS user has a basic knowledge of statistics and the algorithm. They need to understand how the different parameters can adjust the performance of the system.

2.9 Specific Requirements

This section explains in detail the user inputs for each mode of operation. It discusses the valid range of inputs and the outputs produced by each mode.

2.9.1 Command Line Arguments

The three modes of operations are distinguished by their command line arguments. The "mode" argument takes the parameters training, classify, evaluation to enter thee three modes. The full list of arguments and their meaning is given in Appendix A.

The simplest form of the BDRA program invocation is illustrated by the following example.

```
java bdra.Main iris.txt
```

In this case the program reads a set of classified observations from file iris.txt, and runs with full defaults. At the end of the run, it creates file named report.pdf containing a summary of results. To override a default, parameters are placed before the input file name. For example, to have the program run in evaluation mode and output a report called pdfname, the command is as follows.

```
java bdra.Main  --mode evaluation --report-name
iris default iris.bdra
```

2.10 Classified Data File Format

The file contains character data, so that users can edit the file manually with a text editor. Data The file is organized into lines that are terminated by line feed, or carriage return/line feed. The data items in a line are delimited with commas. Figure 2.5 shows the line by line contents of the file.

The first line is the missing data symbol is a string that is used to indicate a feature value that is missing. The second line is an integer n that represents the number of features. The n lines starting at line 3 describe the features. Each of these lines contain the information shown in Table 2.1.

The m subsequent lines contain the observations describe the features. Each of these lines contain n feature values, followed by a category symbol. Table 2.2 illus-

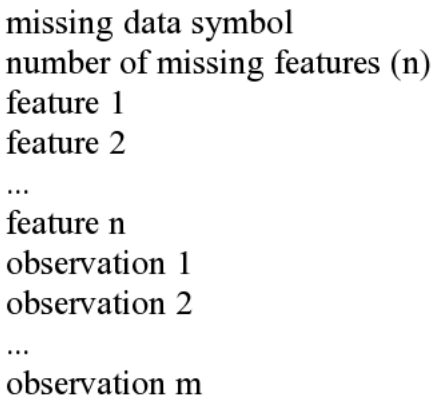

Fig. 2.5: Format of Data File.

Tab. 2.1: Format for Feature Information

Column	Description
1	"continuous" or "categorical"
2	feature label
3	feature description

Tab. 2.2: Format for Feature Values

Column	Description
1	feature value 1
2	feature value 1
3	feature value 1
	...
n	feature value n
n + 1	category symbol

trates this.

2.11 Modes of Operation

This section will explain in detail the inputs and outputs of each of the three modes of operations

2.11.1 Evaluation Mode

This mode allows the user to adjust the various settings of the ADRS to search for the optimal classifier. The following inputs are used in evaluation mode:

Continuous bins The number of bins affects the error rate and the performance. A higher number of bins causes the algorithm to run slower, but will not necessarily provide better classification results. If not specified the default is three bins.

Search direction The user can choose forward or backward. Backward search should provide better results but requires more time to process the data. Forward is

faster and may be a better choice for very large data sets. If not specified the default value is forward.

Search technique The user can choose between floating and sequential. Sequential is faster than floating. Floating should theoretically provide better results. If not supplied the by the user the default is sequential.

Data set The user must specify the training data. This must always be included.

Missing value technique The choices are mean field and pseudo bin. For data sets with many missing values mean field should provide better results, but is substantially slower. The default is pseudo bin.

Report name The of the PDF report that will be generated by the ADRS. If not specified the default is report.pdf.

Window size If the user inputs a window size of one, then the algorithm will run as many times as there are observations. Each time it will withhold a different observation for testing. The results are aggregated in the report. If the number is higher than one, the algorithm sets aside that number of observations for testing and runs a single time. If the window size is more than the number of observations an error message will bet reported to the user. The output of evaluation mode is a PDF report summarizing the statistics of the run.

A sample input using all inputs is given below:

```
java bdra.Main --mode evaluation
--window-size 1 --search-technique floating
```

```
--search-direction forward
--missing-value-approach pseudo-bin --report-name
iris iris.bdra
```

2.11.2 Training Mode

Training mode is almost identical to evaluation mode. All of the inputs for evaluation mode are used except for window size. The output of training mode is a serialized Java object called a classifier. This will be saved on disk to be used as an input in classify mode. A PDF report is also generated in training mode. It is similar to the report from evaluation mode.

2.11.3 Classify Mode

Classify mode takes data from unclassified observations and predicts their class membership. The Classifier produced in training mode is used as an input. The set of inputs is as follows:

Classifier name the name of the serialized java object produce in training mode. This must exist on disk or a exception will occur.

Unclassified data file name the name of the file that contains the unclassified data. This must be in comma separated value format with each line being a single observation.

2.12 Performance Requirements

Although design decisions were made to make the algorithm as efficient as possible, performance is not the main goal of this iteration. Some large data sets take days to process, but the set up for the process is minimal.

2.13 Reliability

ADRS is expected to be available continuously.

2.14 Maintainability

ADRS needs no routine maintenance. The user should remove old reports and data as needed.

2.15 Portability

Because ADRS is written in Java, it can run on Windows, OS X, Linux and Sun systems.

3. DESIGN

3.1 Introduction

ADRS is computationally intensive algorithm. Many decisions were made to make the algorithm as efficient as possible, but flexible enough so that it could be extended in future iterations.

The design of ADRS will be discussed in the roughly same order that the program is executed. Data is first read in from disk. ADRS only works on discrete data so the continuous values are discretized and then converted into symbols. The algorithm then processes the set of symbols using a technique called feature reduction. The resulting set of symbols can be used to classify new data.

The program starts in the class Main. Figure 3.1 shows the class diagram of Main. Here the command line arguments are processed and one of the three modes of operations will run. The overall design of ADRS will be discussed through its evaluation mode. This is where the user will spend most of their time. The other modes of operations can be thought of subsets of evaluation mode. Training mode produces a classifier. It is essentially the same as evaluation mode except the data is not partitioned into training and testing sets. Classify mode uses the classifier produced in training mode to classify new data.

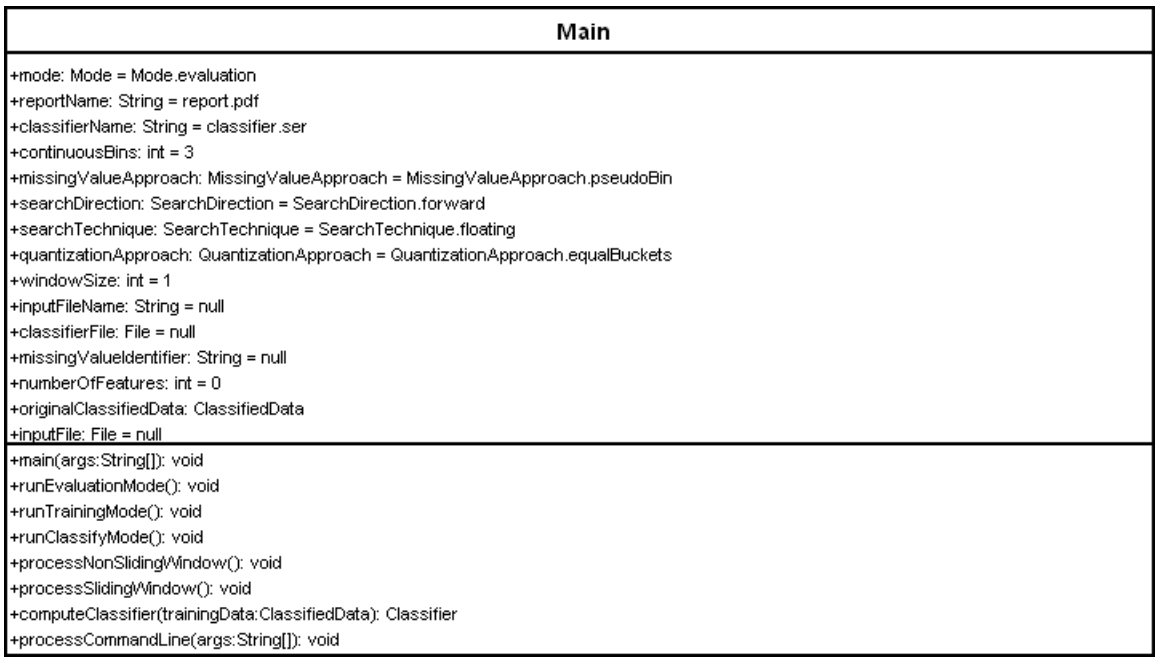

Fig. 3.1: Class Diagram of Main.

3.2 Reading the Data File

The algorithm spends most of its time applying transformations to the model and calculating the training error. The set up of the algorithm consists of a single read of the data file from disk. This text file is parsed and an object called observations is created. It will be stored in memory until all iterations of the algorithm have completed.

The raw data is kept in comma separated value format. The first few lines of the file provide some Meta data to make reading the data more efficient. The first line denotes the missing value identifier. If there are no missing values this can be any symbol because it will not be applied. The next is an integer denoting the number of features for each observation. If there are four features, then the next four lines state

```
?
6
continuous,mcv,mean corpuscular volume
continuous,alkphos,alkaline phosphotase
continuous,sgpt,alamine aminotransferase
continuous,sgot,aspartate aminotransferase
continuous,gammagt,gamma-glutamyl transpeptidase
continuous,drinks,number of half-pint equivalents of alcoholic beverages
85,92,45,27,31,0,1
85,64,59,32,23,0,2
86,54,33,16,54,0,2
91,78,34,24,36,0,2
87,70,12,28,10,0,2
.
.
.
```

Fig. 3.2: Data Format for Classified Data.

information about each of these features. These lines have three following pieces of information in them, each separated by a comma: categorical or continuous, a label, and a description.

Data can be continuous or categorical. Continuous data is numerically valued and therefore has an ordering. ADRS only deals with discrete data so continuously valued data will be quantized into discrete categories. Categorical value can take on a finite set of values and there is no applicable ordering to such data.

The label is something that can be used as a header in a table if needed, while as the description can be longer and used in a more detailed report. The lines following the Meta Data in the file represent the actual data. The values are comma separated and the last value in each line is the class the observation belongs to. The beginning of the raw data file for liver disorders is shown in Figure 3.2:

To read in the data file a instance of the class ClassifiedObservationFile is created.

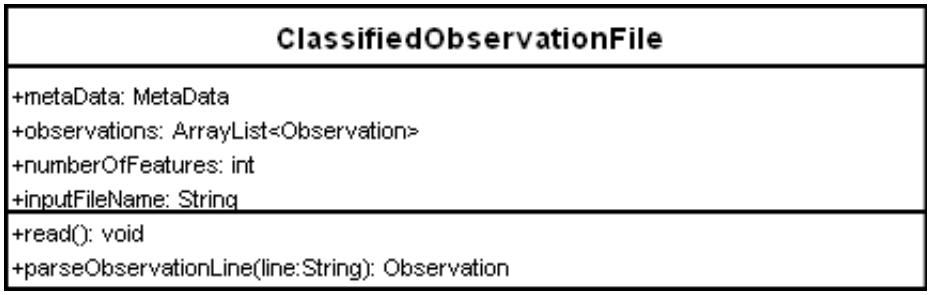

Fig. 3.3: Class Diagram for ClassifiedObservationFile.

Figure 3.3 shows the class diagram for this class. A path to the file is needed at the time of its creation. This class contains a read method that will parse the data from the file. After the read method completes several classes will be created. Figure 3.4 shows the class diagram for the data structures important to this part of the program.

3.2.1 Meta Data

The read method begins by creating a object called MetaData. MetaData is organized by feature. It contains an ArrayList of FeatureMetaData. Figure 3.5 shows the class diagram of MetaData and Figure 3.6 shows the class diagram for FeatureMetaData. Each instance of the FeatureMetaData class keeps tracks of the Meta data for each feature. The features are indexed by their position in the MetaData ArrayList. The first feature would be in position 0, the second in position 1, and so on. Some of this Meta data comes from the beginning of the data file as seen in 3.2, and some from the data itself. For features that are categorical, we must know how many different values the data takes on. As the data is read in we keep track of distinct values and then save them in the FeatureMetaData for that particular feature. We must also know the number of classes for the data file. As each observation is read we store the

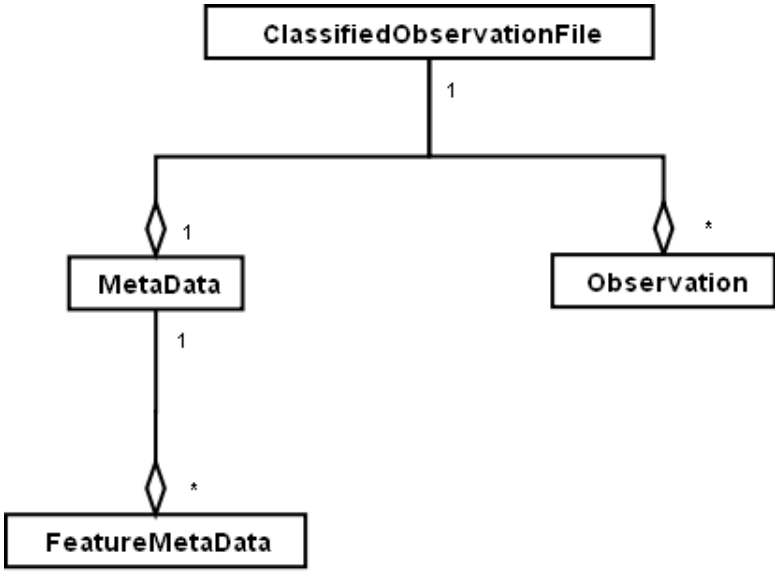

Fig. 3.4: UML for Data Classes.

distinct classes and store them in an array in MetaData.

3.2.2 Observations

After the lines of the data file containing Meta data have been processed, the read method begins processing the actual data. Each line in the file represents a set of measurements and the class they belong to. Each of these is referred to as an observation. Each measurement in the observation is referred to as a feature value or value. In the case of iris classification, four measurements are taken: sepal width, sepal length, petal width, and petal length. The last value in the line is the class of the observation. An example of an observation from the iris data set is as follows.

```
5.1,3.5,1.4,0.2,Iris-setosa
```

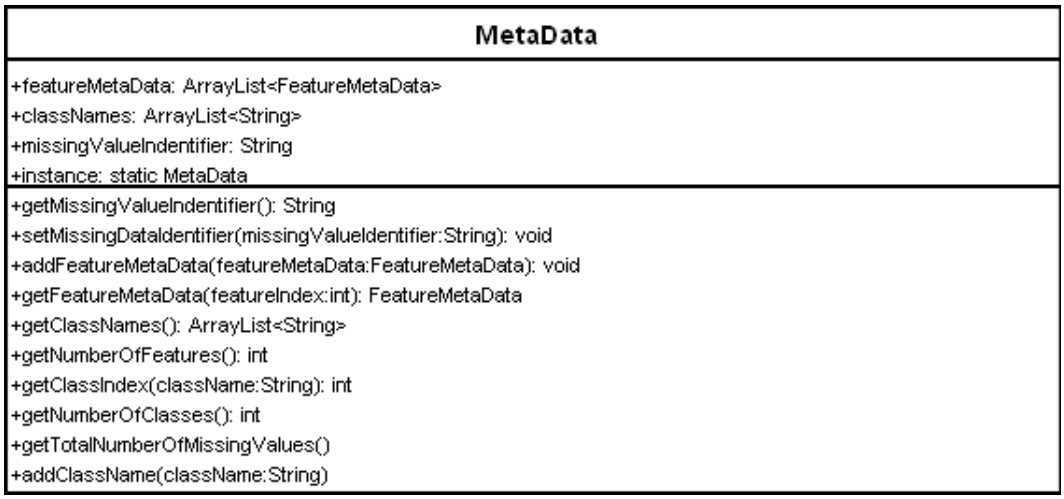

Fig. 3.5: Class Diagram for MetaData.

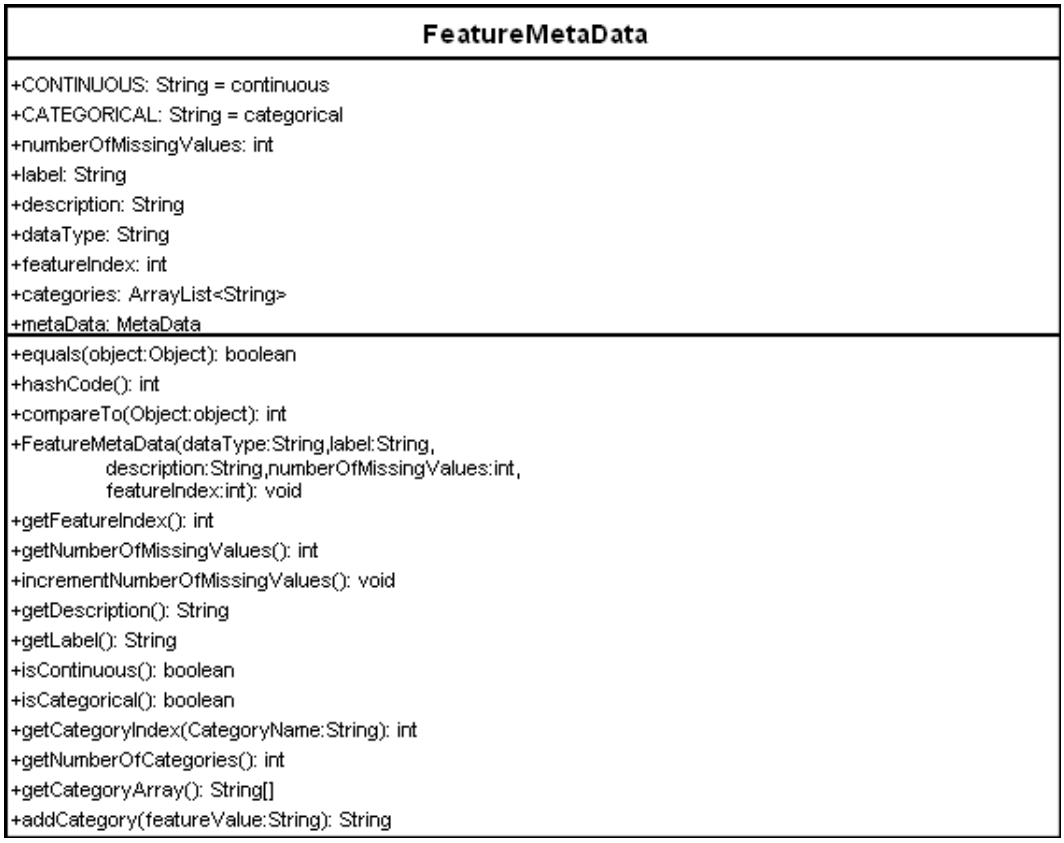

Fig. 3.6: Class Diagram for FeatureMetaData.

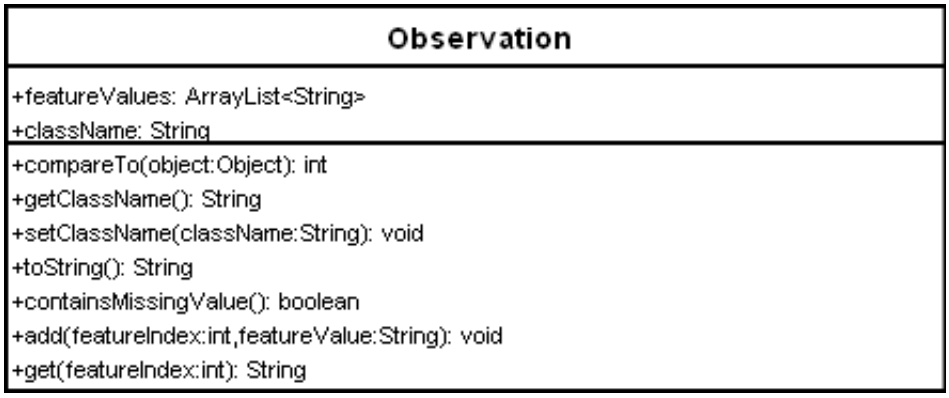

Fig. 3.7: Class Diagram for Observation.

The method parseObservation is called for each line of data in the file. This method returns an object called Observation for each line in the data file. Although each line in the data file is a vector of numerical and string data, we store it as an ArrayList of String. It is during quantization that numerical valued data will be converted to Doubles. The method parseObservation also calls the method incrementNumberOfMissingValues in the FeatureMetaData class when it encounters a missing value. The number of missing values will be useful to the user and is included in the reports generated. Inside ClassifiedObservationsFile is an ArrayList called observations. This will contain all of the observations created by parseObservations. This data structure will be used over and over again at the beginning of each run of the algorithm. Observations will be partitioned into training and test data at the beginning of each run.

3.3 Partitioning the Classified Data

A single run of the algorithm does not produce meaningful results. It is a statistical technique that requires random observations to be held out for testing. The algorithm must be run many times and the results are aggregated. Each time the algorithm is run, different observations are held out as test data. The user may choose to hold out a single observation or many. The observations used for testing are randomly selected, so the training observations must be quantized at the beginning of each trial.

If the user chooses to set a side a single value, the algorithm will run once for each observation set aside. If the user selects a certain percentage to set aside, the algorithm will run once. If the user selects the latter, they should run the algorithm many times to compare results.

The observations are put into two separate objects called trainingData and testData. The training data is quantized and the test data will be used at the end of the run to test the classifier.

The data is partitioned in the class Main. One of two methods is called depending on the user inputs. ProcessSlidingWindow is called if the user chooses a set aside value of one. ProcessNonSlidingWindow is called if the user chooses a set aside value higher than one. Regardless of the mode the user decides to use, the observations set aside for training will be quantized and symbols will be created.

3.4 Quantization

Continuous features must be quantized into discrete categories prior to running the algorithm. This must be done prior to each run of the algorithm since different

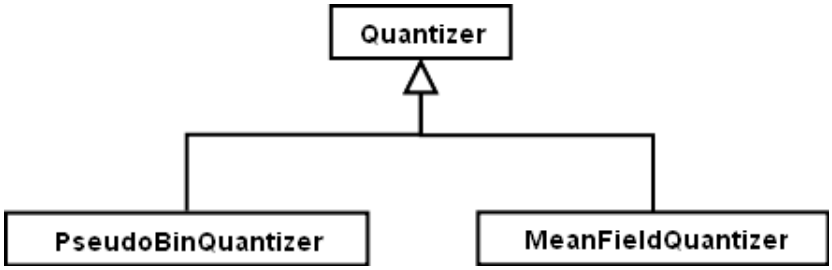

Fig. 3.8: Class Inheritance for Quantizer.

Quantizer
+rawTrainingData: classifiedData +observatrionGroups: ObservationGroups +backwardSearchModel: Model
+createSymbols(): void +quantizeContinuousFeatures(): void +quantizeContinuousFeatureEqualBucket(): void +qunatizeContinuousFeatureRangeBased(): void

Fig. 3.9: Class Diagram for Quantizer.

observations will be set aside as test data, thus changing the overall nature of the data. The class Quantizer sorts the continuous data and determines thresholds that will be used to divide the data into percentiles or "bins". The Quantizer also creates an instance of the class ObservationGroups. This class consists of objects called symbols and symbol counts.

The class Quantizer sorts the data and creates symbols. Quantizer has two subclasses: MeanFieldQuantizer and PseudoBinQuantizer. These two subclasses differ in the way they handle missing values which affects the way the symbols are created. Figure 3.8 Show the inheritance structure for the Quantizer classes. Figure 3.9, 3.10, and 3.11 show the class diagrams for each of the three classes.

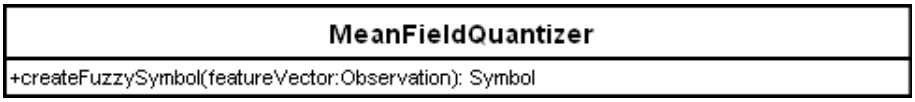

Fig. 3.10: Class Diagram for MeanFieldQuantizer.

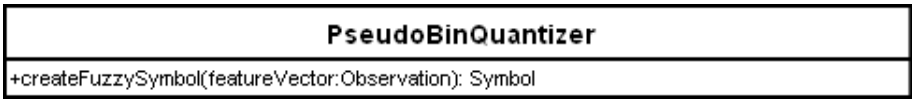

Fig. 3.11: Class Diagram for PseudoBinQuantizer.

The PseudoBinQuantizer takes missing values and assigns them all to a bin called the pseudo bin. This bin is a level higher than the number chosen by the user to quantize. If the user chose three bins to quantize, all missing values are assigned a fourth bin.

The MeanFieldQuantizer assigns missing values a value of null when the symbols are created. This will be used to "spread out" the assignment of bins in the mean field algorithm. This will be explained in detail in the mean field algorithm section.

3.4.1 Creating Symbols

After each feature's thresholds are determined by the Quantizer, each observation is converted into a symbol. It is the symbols that are operated on by the algorithm, not the raw data. The method createSymbol is called on each observation. This method takes each observation and converts it into a symbol. Each feature value is converted to an integer that represents its bin. If the observation has four features, then the symbol consists of four integers, each representing a discrete level of the original data.

Creating Symbols for Continuous Features

For continuous features, the user determines the number of bins. If there are four features and we use three bins to quantize, then 3^4 or 81 symbols are possible. The quantizer sorts the data for a single feature and determines thresholds so that the data can be separated into percentiles.

If the user chooses to quantize using four bins, then the continuous data is sorted and then divided into four roughly equal groups. The lowest 25 % would be assigned a symbol of 0, the next 25 % a symbol of 1 and so on. The percentiles will not always be evenly distributed. Figure 3.12 shows twenty features quantized into three bins. Since 20 divided by 3 is 6 in integer division, the 6^{th} and 12^{th} observation's values will be chosen as thresholds. As you can see in both cases, ties occurred with other observations. In this case seven observations we're placed in the first bin instead of six. Even though ties can occur, in most cases the distribution is fairly even.

The values of the thresholds are saved in a class called the Model. The Model is where the information is kept as to how the data is partitioned to create the symbols used by the algorithm. The Model will be discussed in detail in the next section. The thresholds saved in the model will be used by the Quantizer to convert the raw data into symbols. Figure 3.13 shows the model and thresholds used by a random run of the iris data set.

Creating Symbols from Categorical Features

Categorical features naturally fall into bins. Each distinct category is given an integer that it maps to. For example, if the feature represents values that are either true or

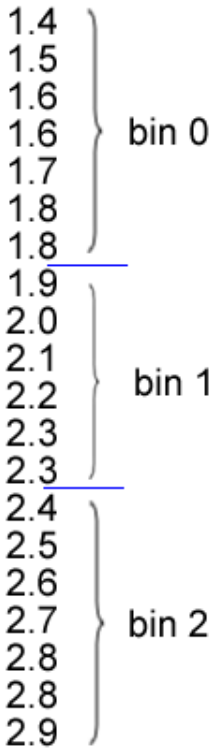

Fig. 3.12: Quantizing Continuous Features into Bins

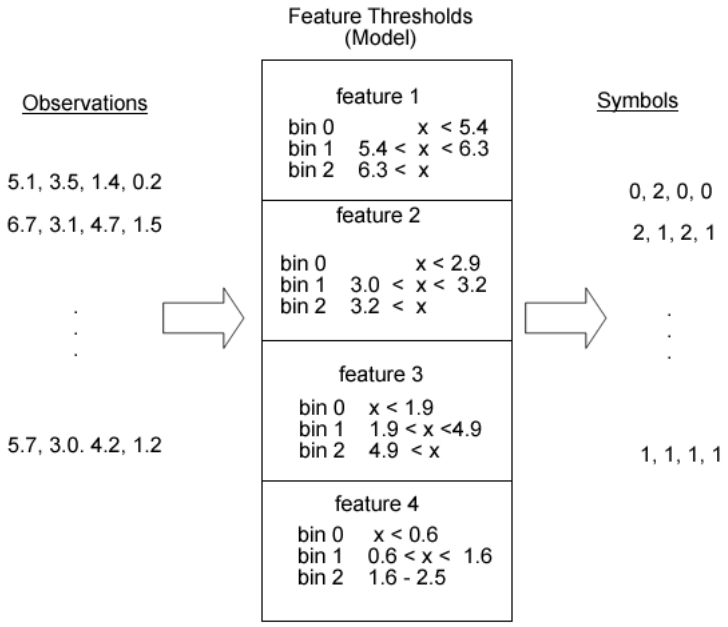

Fig. 3.13: Using Thresholds to Create Symbols

false, then a true would become a 0 and a false would become a 1. Since there is no natural ordering to categories the number they map to is irrelevant as long as each distinct category has its own unique number.

3.4.2 Observation Groups

Since more than one observations may produce the same symbol, we condense these observations into an class called ObservationGroup. The object ObservationGroup contains the symbol which all of the observations have in common and a symbol count, which is an array of how many observations belong to each class. The symbol counts are simply an array where a position in the array represents a certain class. Figure 3.14

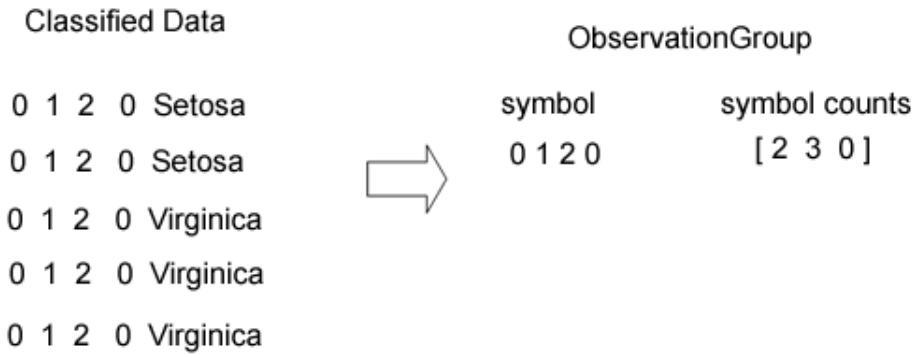

Fig. 3.14: Using Duplicate Symbols to Create a ObservationGroup

The observation groups represent a histogram of the original data and the distribution of the symbols for each class. At this point we could begin to classify data. The algorithm will take these observation groups and combine symbols so that the distinction between classes becomes more evident

3.5 Model

The model is the most important data structure in the algorithm. It is what we apply transformations to during feature reduction. It also serves as a function that maps original symbols to their new symbols based on these transformations. The best model found in evaluation mode is used to build the classifier to be used in classify mode.

The model is organized by feature. It is based on a number of ArrayLists. At the topmost level the class Model contains an ArrayList of the class Feature. A Feature may be continuous or categorical. This affects the way that transformations are

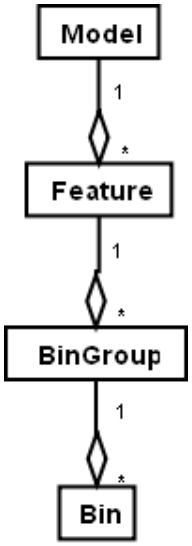

Fig. 3.15: Class Aggregation for Model

applied to the feature. Each feature contains one or more objects called BinGroups. BinGroups is an object that contains an ArrayList of Bin. Figure 3.15 shows the UML for the class Model.

Each feature has a number of bins. If the feature is categorical, each bin holds the name of the category that is mapped to that bin. Each bin has an integer attribute called level. This level is the bin number it represents. This will be used to keep track of the bins as the move from one BinGroup to another. If the feature is continuous the bin contains the threshold that represents the highest value that could be placed in that bin. These thresholds are used in the classifier to map unclassified observations into the appropriate symbols so their class can be predicted.

When the model is created Each bin is placed in a container called a BinGroup. When the model is created each BinGroup contains a single Bin. If the user is doing a backward search then the algorithm progresses by combining Bins into the same

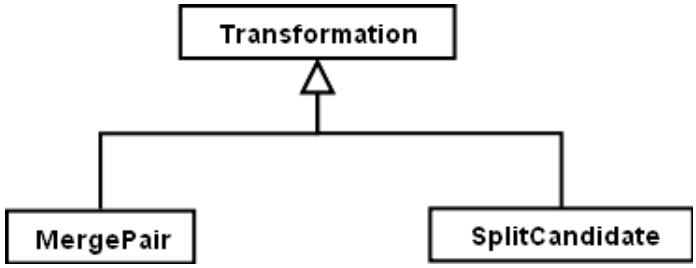

Fig. 3.16: Class Inheritance for Transformation

BinGroups. This is called a merge. If the user is doing a forward search, all the bins in a feature are merged into a single BinGroup. The bins are then separated out using a what is referred to as a split. Both are considered transformations.

3.5.1 Transformations

The original quantization levels chosen by the user will be adjusted through a series of steps called feature reduction. There are two directions to adjusting the feature quantization level, adding a threshold which we refer to as a split or taking a threshold away which refer to as a merge. The class called Transformation has two subclasses called Merge and Split to represent these two processes. Figure 3.16

Merges

A merge removes a threshold from the model. An example of the would be taking Figure 3.13 and removing the lowest threshold of feature one to create Figure 3.22. This has the lowest threshold of the first feature remove. This reduces the number of bins from three to two. After we merge two levels the symbols are recalculated.

The way this is represented in the ADRS is by assigning a bin a new bingroup.

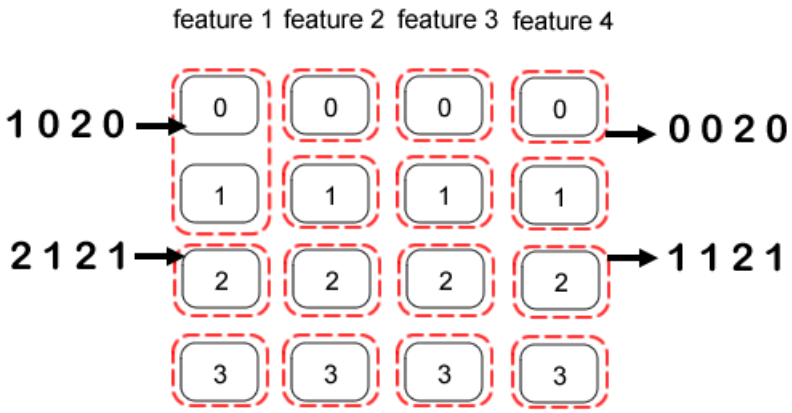

Fig. 3.17: Merging Two Continuous Bins

In Figure 3.18 the dashed lines represent the index of the bingroup in the Feature's ArrayList. Example A shows the initial model with each bin in its own bin group. Example B shows the second bin of the first feature was placed in the first bingroup and the second bingroup was removed which causes a shift in the higher bins. The symbols are transformed by taking all of the levels in a certain bingroup and assigning it to that Bin Groups ArrayList index. Figure 3.17 shows how symbols are remapped to new symbols base on the model. If all of the bins in a feature have been placed in the same bingroup then the feature has be eliminated. This means that that measurement has no bearing on determining the class of the observation.

Splits

If the user is doing a forward search, the model is in a fully merged state. This means that all features have a single bingroup that contains all of the bins. Bins are removed one at a time in an attempt to improve classification. Splits take a bingroup that has

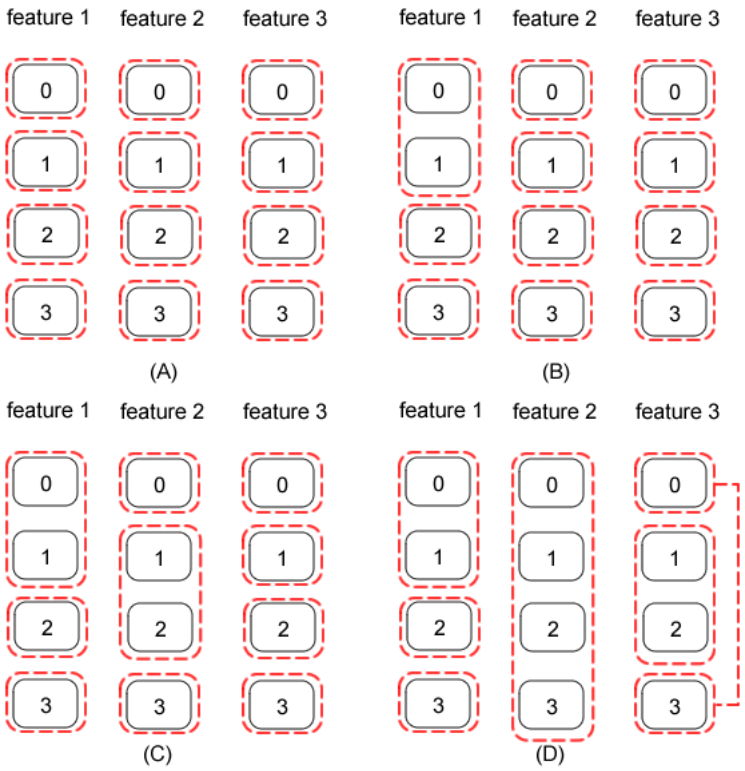

Fig. 3.18: Merging Levels in the Model

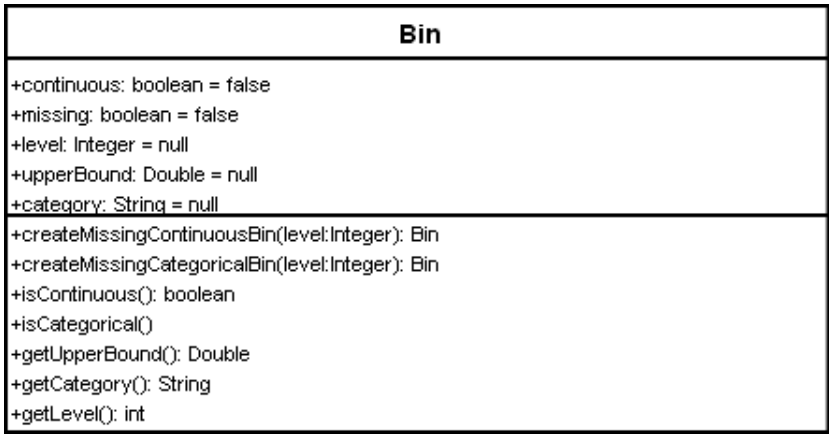

Fig. 3.19: Class Diagram for Bin.

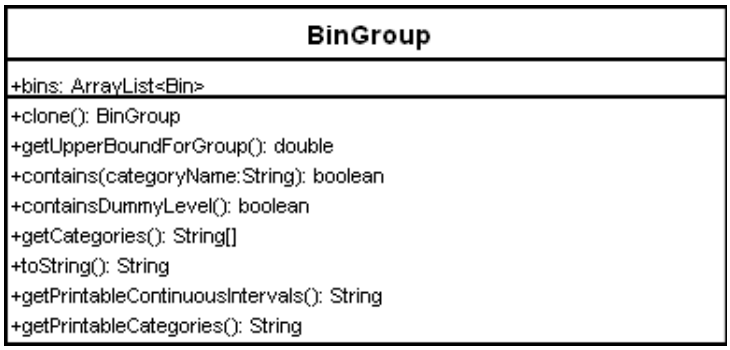

Fig. 3.20: Class Diagram for BinGroup.

more than one Bin in it and removes it into a new previously empty bingroup.

3.5.2 Transforming Continuous and Categorical Features

Continuous Data can only be merged and split adjacently. This is due to the natural ordering of numerical data. Categorical Bins can merge with non adjacent Bins. If a four bins represent the colors red, white, blue and green then they can merge in six different ways. Four continuous bins could only be merged three ways.

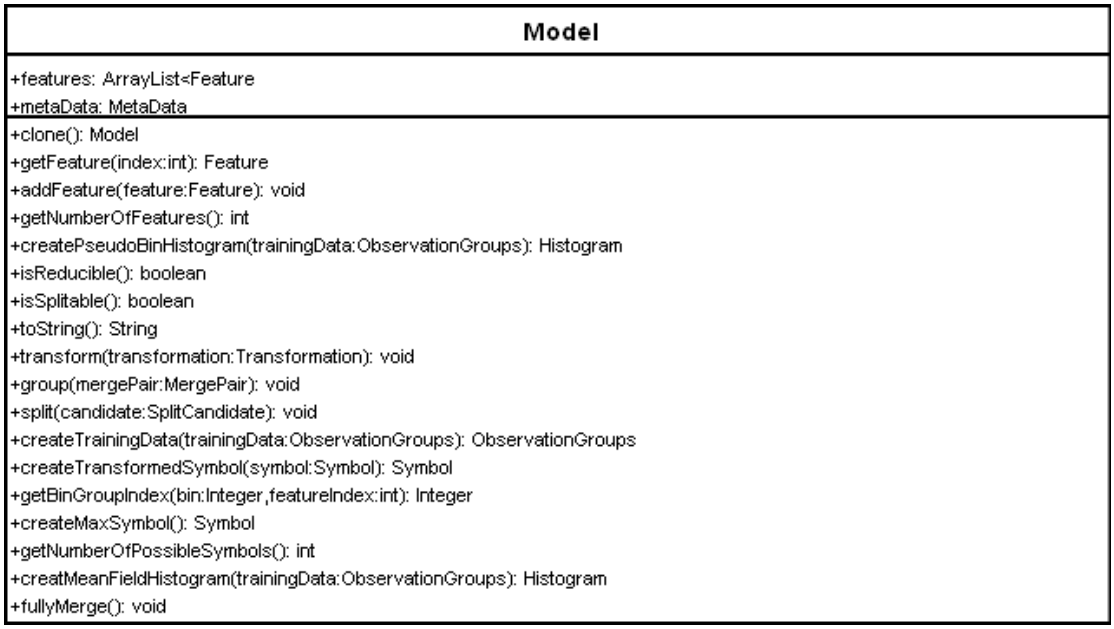

Fig. 3.21: Class Diagram for Model.

3.6 Algorithm

ADRS spends most of its time training on classified data. The algorithm needs the object ObservationGroups and Model to begin. The main loop of the algorithm will be discussed using the backward sequential search. The other approaches are similar and will discussed in the upcoming sections.

1. Begin with the initial model. Create a histogram of the symbols and compute the training error.

2. Merge the lowest two bins of feature one. These two lowest values of feature one are now indistinguishable. Recalculate the symbols, create a new histogram, and calculate the training error.

3. Starting again with the initial model, merge the next lowest bins of feature one.

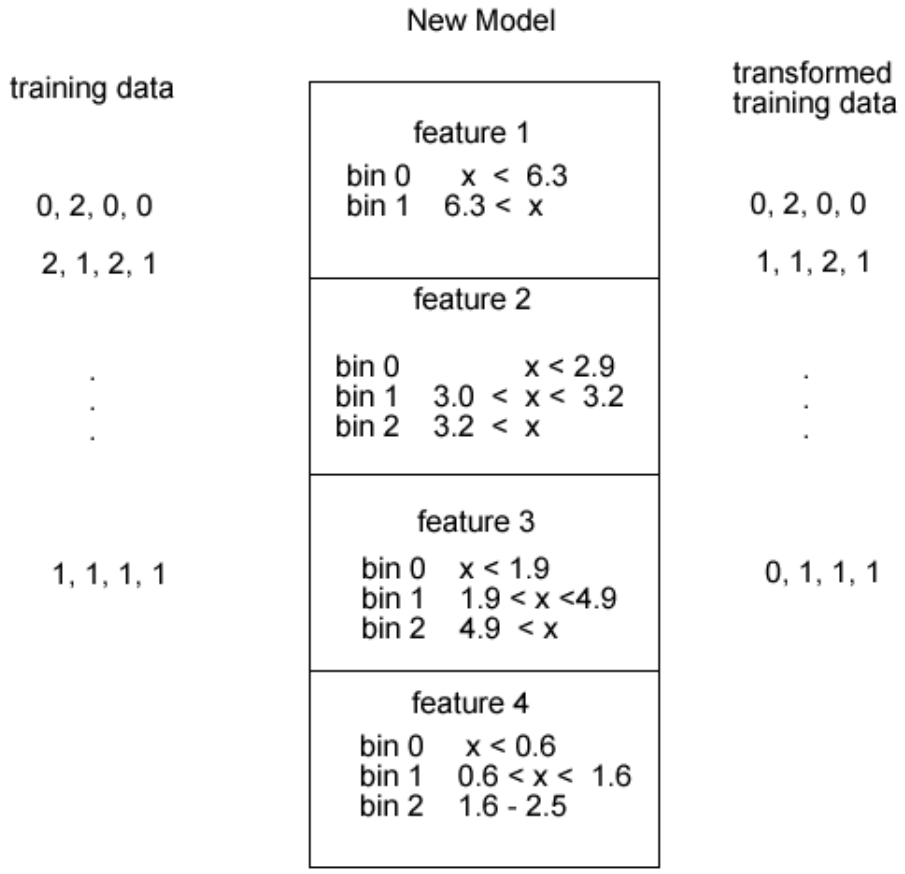

Fig. 3.22: Using the Model to Transform a Symbol

Continue this for all possible merges for this feature and the next features. Each time recalculate the symbols, create a histogram, and compute the error.

4. Take the lowest error of steps 1 - 3 and replace the initial model with the new, reduced model.

5. Repeat steps 1 - 3 now starting with the reduced model. Continue until the error no longer decreases or the model cannot be merged anymore.

The algorithm is greedy since it accepts the best solution at each step unconditionally. Three search directions are available for the user: forward, backward and floating.

3.6.1 Histograms

An important data structure to the algorithm is the class Histogram. The histogram is very similar to ObservationGroups. It contains a number of symbols that map to symbol counts. The object ObservationGroups serve as a set of inputs to the model. Each time we apply a transformation to the model, we recalculate, or remap, the original symbols and create a new histogram. The histogram is then used to calculate the training error.

The class Histogram represents the symbol distributions for each class. Figure 3.23 shows the initial histogram for iris data in a random run. Out of the 135 observations used for training, 24 unique symbols were created. At this point we can use the histogram to classify data. We can see that the fifth symbol is exclusively represented by the class Setosa. If a new unclassified observation also maps to symbol five, then we can be confident it is a Setosa. Conversely, symbol seven has the same number

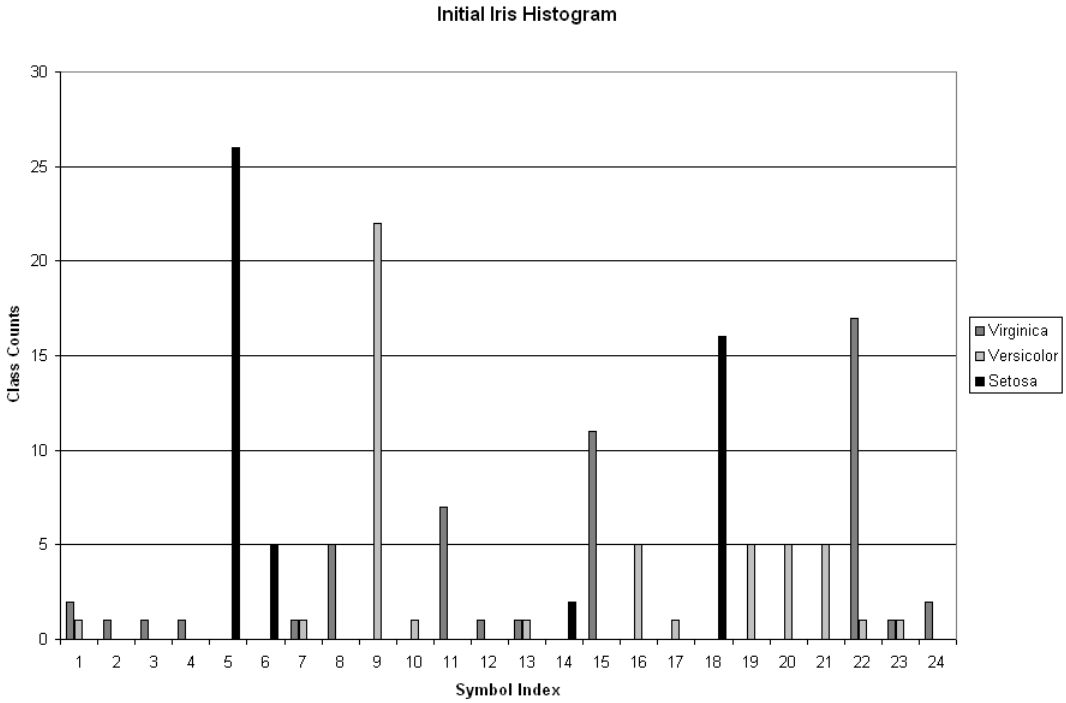

Fig. 3.23: The Initial Histogram for Iris Data

of virginica and versicolor. Based on this distribution we can not make a conclusive decision.

Table 3.1 shows a table representation of the the histogram created from the iris data after the algorithm completes. By comparing the two histograms, it is easy to see the difference in class distribution. The reduced histogram has an error of 10.5% associated with it while the unreduced histogram has an error of 28% associated with it.

Tab. 3.1: Reduced Histogram for Iris Data.

	(0,0)	(0,1)	(1,0)	(1,1)
Setosa	45	0	0	0
Versicolor	0	0	40	5
Virginica	0	0	4	41

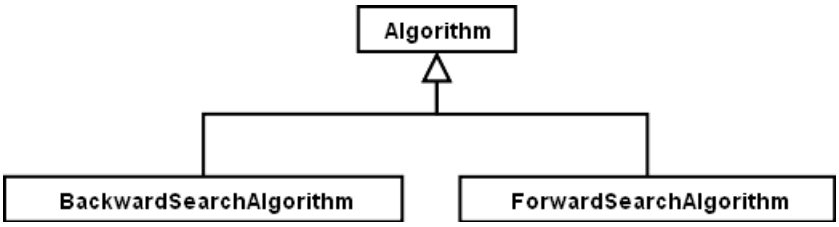

Fig. 3.24: Algorithm and its Subclasses.

3.6.2 Search Direction

The class Algorithm has two subclasses: ForwardAlgorithm and BackwardAlgorithm. Figure 3.24 shows the class hierarchy. Figure 3.25 shows the class diagram for the class Algorithm.

Backward Search

The backward search begins with the the model created at end of quantization. It then begins a loop to do feature reduction. The first step of the loop merges the two lowest bins of the first feature and recalculates the histogram and the error. It then merges the next higher bins together. Once it has attempted this for each possible MergePair on the first feature, it moves to the next feature. It continues until it has tried all possible merges. It then chooses the best merge based on the

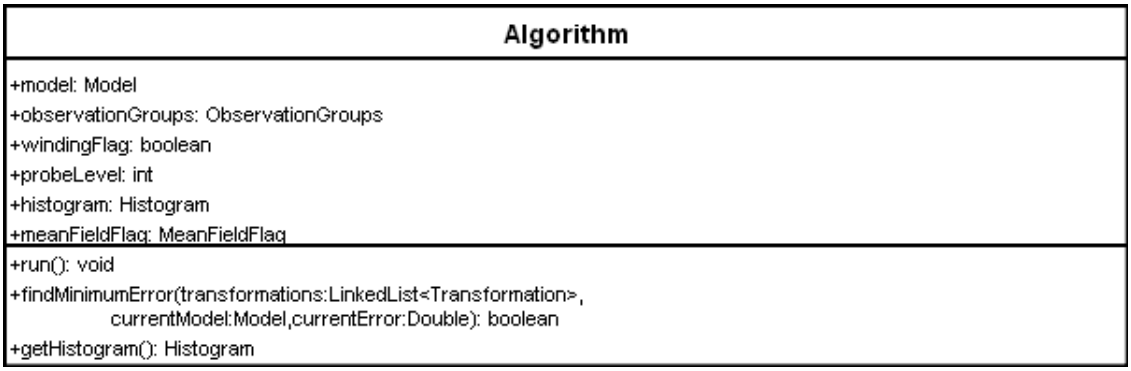

Fig. 3.25: Class Diagram of Algorithm.

error calculation. This merge is then committed and we start over again by merging the two lowest bins on feature one. The algorithm is greedy since once a merge is committed we never undo it to see if the error is better.

To perform this loop we clone the model at each step. We apply the transformation to the cloned model and then we use the model as a function to remap the symbols and create a histogram. The model has a function createHistogram which takes the original observation groups and then recalculates the symbols to create the histogram. The algorithm stops once the error no longer decreases or the model cannot be reduced any further. Figure 3.26 show the class diagram for BackwardSearchAlgorithm.

```
MergePair(featureIndex, binGroupIndexA, binGroupIndexB)
```

Figure 3.18 shows a possible beginning to the backward search where Figure A shows the initial model, Figure B shows that the two lowest bins in feature one were merges at the first iteration. Figure C shows that bins 1 and 2 where merged at the second iteration. Figure D shows a possible final model where feature 2 has be eliminated.

Fig. 3.26: Class Diagram of BackwardSearchAlgorithm.

Fig. 3.27: Class Diagram of ForwardSearchAlgorithm.

Forward Search

The forward search begins with all of the bins in each feature fully merged into on BinGroup. It then follows the same loop as the forward search except at each iteration it removes a Bin from the merged BinGroup. Figure 3.27 show the class diagram for ForwardSearchAlgorithm.

```
SplitCandidate(int featureIndex, int binGroupIndex,
int binLevel)
```

3.6.3 Search Techniques

There are two search techniques, sequential and floating. Sequential only considers moving in one direction. If the user is doing a forward search it only considers splits. If the user is doing a backward search it only considers merges. The floating search considers both.

Sequential

In the class Feature there are several functions that are used to create list of transformations used for searching. This is stored as a LinkedList of Transformation. Depending on the direction of the search one of the following functions would be called: getAllCategoricalSplitCandidates, getAllPossibleTransformations, or getAllPossibleMergePairs.

Floating

Floating creates a linked list of all possible merges and splits and chooses the best one at each iteration

3.6.4 Training Error Calculation

The training error is based on the distribution of symbols for each class in the histogram. The conditional probability of error is given in Equation 3.1. For the full derivation of the training error calculation, see [7]. Since a single observation is tested at a time Equation 3.1 reduces to Equation 3.2. This is the equation used to perform the testing error calculation. $x_{k,i}$ is the number of occurrences of the ith symbol in the training data for class k. N_k is the Total number of training data for class k. M is the total number of symbols.

$$P(e|X) = \sum_{k=1}^{C} \sum_{k=1} P(H_k) \Psi_{\{z_k \leq z_l, forall k \neq l\}} f(y|x_k, H_k) \tag{3.1}$$

$$f(y = i|x_k, H_k) = \frac{x_{k,i} + 1}{N_k + M} \tag{3.2}$$

3.6.5 Testing Error Calculation

After the algorithm completes and we have a reduced model, We test the observations that were set aside for testing to see if we are able to predict the correct class for each observation. The testing error is calculated by taking the number of incorrect predictions divided by the total number of predictions.

To predict the class, we take each observation and create its symbol based on the merged model. We then look up the symbol in the histogram based on the training data. We then predict the class based on the most prominent class of of the symbol in the histogram. For example if the reduced histogram were the one given in Table 3.1, and the symbol to predict was (0,0), we would predict it is a Setosa.

3.7 Missing Value Approaches

There are two approaches to dealing with missing values, the pseudo bin approach and the mean field algorithm. The pseudo bin approach is the default and completes much faster. The mean field algorithm requires a substantial amount of extra computations but will usually give results slightly closer the optimal. It is up to the user to decide if the trade off in increased computation time is worth the increased classification performance.

3.7.1 Pseudo Bin Approach

The pseudo bin approach places all missing values for a feature into the same bin. This bin, which we refer to as the pseudo bin, is separate from the other bins with known values. Our convention is to label this bin with the integer one higher than the

number of bins with known values. If the user chooses five bins to quantize continuous features, then the pseudo bin would be a separate sixth bin. If the feature has four categories, and missing values would be put in a fifth bin.

After the pseudo is created during the quantization stage the algorithm runs as normal. The pseudo bin is allowed to merge with any of the known bins. It is reported to the user if the pseudo bin merged with another bin or stayed by itself.

Because all of the missing values are placed in the same bin, the results depend on the true nature of the data. With credit scores this result works well since many questions left blank are because the respondent does not want to fill in the answer with something that could hurt their score. If the missing answers are evenly distributed among answers, then the results will not be as accurate.

3.7.2 Mean Field Algorithm

The mean field algorithm takes a much different approach to dealing with missing values. Instead of lumping the values into one bin, we spread the weight among all of the possible symbols it can take on.

For example, consider the case where the given data has three features that are quantized using two bins. The following symbol has the third value missing which we will denote with an x. Since this represents a single observation, we place the value of one in the appropriate place in the symbol count. For this example we will assume we have two classes, A and B, where A is the first position in the symbol count array and class B is the second position. This observation is of class B. The the symbol and its symbol count would be as follows:

(1,0,x) [0 1]

In the pseudo bin approach this would be represented as:

(1,0,2) [0 1]

For the mean field algorithm, we create all the possible symbols that the missing value could represent. Since the last value could be either a zero or a one, we create both symbols and give each half of the symbols weight. Two symbols are created and each would count as half an observation

(1,0,0) [0 0.5]

(1,0,1) [0 0.5]

Table 3.2 shows a sample data set of ten observations, some of which have missing values. The double line after the fifth observations denotes that the first five observations are of class A and the observations six through ten are of class B. Although the classes are important for the error calculation, for the mean field adjustments they are not needed.

After all of the symbols are given weights, we make adjustments based on the number of other symbols that match them. For the full development of the mean field algorithm calculation, see [6]. The algorithm is begins by assigning weights. Equation 3.3 shows how these weights are determined. $\pi_{i,j}$ is just the reciprocal of the total number of symbols that can be represented by the missing values. Equation 3.4 redistributes the weights based on how many other occurrences are found. Equation 3.5 states that we repeat the calculation in Equation 3.4 until the values stabilize based on some user defined threshold. Equation 3.6 states that the weights found in

Tab. 3.2: Initial Mean Field Values

	(0,0,0)	(0,0,1)	(0,1,0)	(0,1,1)	(1,0,0)	(1,0,1)	(1,1,0)	(1,1,1)
(0,0,0)	1	0	0	0	0	0	0	0
(0,0,1)	0	1	0	0	0	0	0	0
(0,1,0)	0	0	1	0	0	0	0	0
(0,0,x)	0.5	0.5	0	0	0	0	0	0
(0,1,1)	0	0	0	1	0	0	0	0
(0,0,1)	1	0	0	0	0	0	0	0
(0,1,1)	0	0	1	0	0	0	0	0
(0,1,1)	0	0	1	0	0	0	0	0
(1,0,1)	0	0	0	0	1	0	0	0
(x,0,x)	0.25	0.25	0	0.25	0.25	0	0	0

the last iteration are used to populate the histogram.

$$\beta_{i,j}^{(1)} = \left\{ \begin{array}{ll} 0 & j \notin w_i \\ \\ \pi_{i,j} & j \in w_i \end{array} \right\} \tag{3.3}$$

$$\beta_{i,j}^{(n+1)} = \begin{array}{ll} 0 & j \notin w_i \\ \frac{\left(1+\sum_{l=1}^{N} \beta_{(n)}^{(lj)}\right)\pi_{i,j}}{\sum_{w_i}\left(\left(1+\sum_{l}^{N=1}\beta_{l,j}^{(n)}\right)\pi_{i,j}\right)} & j \in w_i \end{array} \tag{3.4}$$

$$\sum_{l=1}^{N}\sum_{j=1}^{M}\left(\beta_{l,j}^{(n+1)} - \beta_{l,j}^{(n)}\right)^2 > (Tolerance) \tag{3.5}$$

$$x_j = \sum_{i=1}^{N} \beta_{l,j}^{(n)} \tag{3.6}$$

These recalibrations are done in a loop until the changes in the weights are too minimal to continue. This is calculated in part by calculating the overall difference between the two tables as a whole. The first two iterations of the mean field algorithm for Table 3.2 are shown in Table 3.3, and Table 3.4. Each time we make a transformation to the model, we use the mean field algorithm to adjust the weights. These weights are used to create the histogram which is used for the testing error. Note that in the pseudo bin approach the class counts are whole numbers. Using the mean field algorithm allows decimal values for class counts.

3.8 Training Mode

Training mode produces a serializable Java object called a Classifier. This Classifier class is shown in Figure 3.28. Training mode does not set aside any data for testing, it uses the entire set of classified data to produce the classifier. This classifier is saved on disk to be used to classify unclassified data.

Tab. 3.3: Values After First Adjustment

	(0,0,0)	(0,0,1)	(0,1,0)	(0,1,1)	(1,0,0)	(1,0,1)	(1,1,0)	(1,1,1)
(0,0,0)	1	0	0	0	0	0	0	0
(0,0,1)	0	1	0	0	0	0	0	0
(0,1,0)	0	0	1	0	0	0	0	0
(0,0,x)	0.41	0.59	0	0	0	0	0	0
(0,1,1)	0	0	0	1	0	0	0	0
(0,0,1)	1	0	0	0	0	0	0	0
(0,1,1)	0	0	1	0	0	0	0	0
(0,1,1)	0	0	1	0	0	0	0	0
(1,0,1)	0	0	0	0	1	0	0	0
(x,0,x)	0.28	0.39	0	0.11	0.22	0	0	0

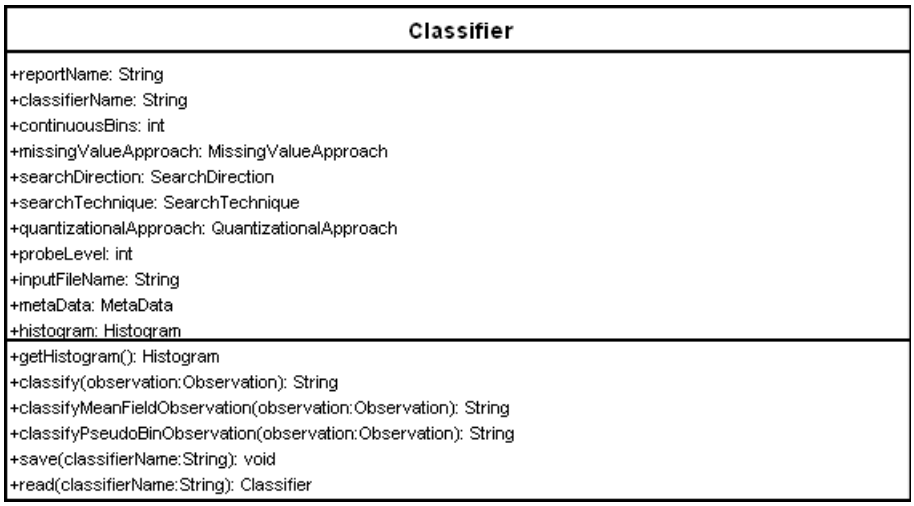

Fig. 3.28: Class Diagram for Classifier.

Tab. 3.4: Values After Second Adjustment

	(0,0,0)	(0,0,1)	(0,1,0)	(0,1,1)	(1,0,0)	(1,0,1)	(1,1,0)	(1,1,1)
(0,0,0)	1	0	0	0	0	0	0	0
(0,0,1)	0	1	0	0	0	0	0	0
(0,1,0)	0	0	1	0	0	0	0	0
(0,0,x)	0.40	0.60	0	0	0	0	0	0
(0,1,1)	0	0	0	1	0	0	0	0
(0,0,1)	1	0	0	0	0	0	0	0
(0,1,1)	0	0	1	0	0	0	0	0
(0,1,1)	0	0	1	0	0	0	0	0
(1,0,1)	0	0	0	0	1	0	0	0
(x,0,x)	0.27	0.40	0	0.11	0.22	0	0	0

3.9 Classify Mode

Classify mode takes two arguments as inputs: a Classifier and a data set of unclassified data. It then uses the Classifier to predict the classes of the unclassified data. This is done in the same manner as the testing error in evaluation mode. A PDF report is given of the the predicted class for each observation.

4. TESTING

4.1 Introduction

Several different strategies were used to test the correctness of ADRS. Since the algorithm makes thousands of calculations for each of its runs, it is impractical to verify each computation by hand. For the two major computations involved, the training error calculation and the mean field algorithm, unit tests were developed. For other stages of the implementation, we chose certain stages of the algorithm and logged the outputs. These were verified by hand wherever possible.

During the development of ADRS, close contact was kept with Dr. Lynch. We used many of the data sets Dr. Lynch used in his published paper on the BDRA. These results were shared with him and he approved our results.

4.1.1 Comparison of Published Results

To verify the correctness of our implementation of the BDRA, we compared results from our implementation to the results Dr. Lynch published that were calculated in MATLAB. The data sets used came from University of California, Irvine's Machine Learning Database. This data set is available online at the following URL:

`http://mlearn.ics.uci.edu/databases/`

Table 4.2 gives the details of the data sets used in common with Dr. Lynch's research. Table **??** shows the comparison of the classification error results with Dr. Lynch's implementation of the BDRA. In some cases we were able to improve on his results. In two of the cases our results were significantly above his. We have not yet found the reason for this discrepancy.

4.1.2 Quantizer Test

To test the quantizer, unit tests were developed. Figure 3.12 shows a one of the unit tests for quantizing continuous features.

Another way we tested the quantizer was to use training mode. Since training mode does not set aside any values, we outputted the intial model that was created. The model keeps track of the thresholds for the continuous features and the categories for the categorical features. We then used a spread sheet to csort the continuous features and determine the thresholds by hand. These results were compared to the output of the program to verify its correctness.

4.1.3 Symbol Test

After quantization the observations must be turned into symbols and the observation groups are created. Again these were tested by checking the output of the program to hand calculated results. Figure **??** showed how symbols are created using the model. This calculation was hand checked by looking at the model created the observation and the resulting symbol. We then checked to see if the observation group was created correctly by verifying the symbol counts.

Tab. 4.1: University of California, Irvine Data Set Details

Data Set	Observations	Features	Classes	Continuous Classes
Annealing	798	38	5	6
Balance Scale	625	4	3	0
Credit	690	15	2	6
Ecoli	336	7	8	7
Glass	214	9	6	9
Hepatitis	155	19	2	6
Ionosphere	351	34	2	32
Iris	150	4	3	4
Lenses	24	4	3	0
Liver	345	6	2	6
Lung Cancer	32	56	3	0
Diabetes	768	8	2	8
Tic Tac Toe	958	9	2	0
Wine	178	13	3	13
Yeast	1484	8	10	8

Tab. 4.2: Comparison of ADRS and Published BDRA Results

Data Set	ADRS	Mat
Annealing	0.10	1.7
Balance Scale	0.15	0.27
Credit	0.14	0.14
Ecoli	0.20	0.30
Glass	0.33	0.35
Hepatitis	0.13	0.26
Ionosphere	0.07	0.24
Iris	0.05	0.06
Lenses	0.05	0.04
Liver	0.26	0.40
Lung Cancer	0.53	0.15
Diabetes	0.19	0.29
Tic Tac Toe	0.03	0.05
Wine	0.09	0.08
Yeast	0.46	0.42

Tab. 4.3: Unit Test for Training Error

	(0,0)	(0,1)	(0,2)	(1,0)	(1,1)	(1,2)
A	6	2	1	1	0	0
B	0	6	2	0	1	1
C	1	0	1	0	6	2

4.1.4 Training Error Unit Test

The training error is used to decide which transformation we use to perform feature reduction at each stage of the algorithm. This calculation is made hundreds of times during the course of a single run of the algorithm. To verify the correctness we used the histogram shown in Table 4.3 as a unit test. The associated training error for this histogram is 39.6%. We applied the algorithm to this histrogram and also checked the error for the associated transformations.

4.1.5 Mean Field Algorithm Unit Test

The mean field algorithm is also computationally intensive. To verify its correctness we developed a unit test. Tables 3.2, 3.3, 3.4, which were used as examples in chapter three, show the unit test used verify the mean field algorithm.

5. FUTURE DIRECTIONS

5.1 Introduction

The are many possible directions to extend the work completed here. ADRS is meant to be a commercial product and there is much left to do in terms of a user interface. Although commercialization was the main focus, many theoretical questions are left to be answered as well.

5.1.1 Commercialization

ADRS was developed as a commercial system. Its user interface was not developed because we did not want to spend time guessing what clients may want. The next phase of the commercial implementation is to get companies to share data with us and to develop an interface to suit their needs.

Interface

During the first phase of this project a GUI was created using Java Swing. Our first iteration of the algorithm did not allow for search direction other than backward. The data structures of the algorithm were redesigned to allow for more flexibility. Since we did not have any clients to discuss design with, the GUI was abandoned and left until the commercialization of the project was further developed.

To make ADRS more accessible a more enhanced interface needs to be created. Two different interfaces could be developed. A java swing GUI could be used for a stand alone desktop implementation of ADRS. If we choose not to distribute ADRS as desktop application, we could create a web front end for users to use the server housed at CSUSB.

Distributed System

For extremely large data sets ADRS could run for a number of days. A distributed system to complete the training would be extremely helpful in these situations. The Computer Science Department at California University, San Bernardino has been developing a distributed system over the last few years called Spider. This uses Java threads an distributes them among computers in the distributed system. Adapting ADRS for the Spider system would greatly enhance performance.

5.1.2 Theoretical

The BDRA has many opportunities for theoretical research. Some of these could improve the classification of the algorithm and consequently improve the commercial product.

Branch and Bound

Since the algorithm uses a greedy approach it is not optimal. Depending on the nature of the data, the search could stop at a local minimum. Implementing a branch and bound search would find an optimal solution without having to search the entire solution space. A branch and bound algorithm needs an evaluative function to see if

a certain path in the search tree could possibly contain a solution. There has not yet been a chance to see if this type of search is possible for the BDRA.

Mean Field Classifier

The mean field algorithm works well with data sets with missing values. Another approach to using the algorithm is to think of the class label as a missing value. The mean field algorithm would treat this as an extra feature and make adjustments to its weights as the other features. Unclassified observations could be added to the classified observations and the classes for the unclassified observations would be predicted by the dominant weight.

APPENDIX A

COMMAND LINE PARAMETERS

Command Line Parameter Default Values

Parameter	Possible Values	Default Value
mode	evaluation,training, classify	evaluation
report-name	$< string >$	report.pdf
classifier-name	$< string >$	classifier.ser
continuous-bins	$< integer >$	3
missing-value-approach	pseudo-bin, mean-field	mean-field
search-direction	forward, backward	forward
search-technique	floating, sequential	floating
window-size	$< integer >$	1

Command Line Parameter Meanings

Parameter	Meaning
mode	The mode of operation
report-name	Sets the name of the mode-dependent report. The extension ".pdf" is appended to the $< string >$.
classifier-name	Sets the name of the classifier file. The extension ".ser" is appended the $< string >$ This option is only valid in training mode (when the file is written) and classify mode (when the file is read).
continuous-bins	Specifies the number of equally-sized bins into which continuous feature values are placed. This option is only valid in evaluation mode and training mode.

Command Line Parameter Meanings Continued

Parameter	Meaning
missing-value-approach	Specifies the approach to use to handle missing values. This option is only valid in evaluation and training modes. In classify mode, the approach is determined by the approach used in classifier mode, which is stored in the classifier file.
search-direction	Specifies the search direction (or starting point) for the BDRA search. Forward means start the search from a fully reduced model, and backward means start from a non-reduced model. This option is only valid in evaluation and training modes.

Command Line Parameter Meanings Continued

Parameter	Meaning
search-technique	Specifies whether the BDRA search is floating or sequential. This option is only valid in evaluation and training modes.
window-size	The number of observations to set aside as test data. If a value greater than one is specified, then the program randomly selects that many observations to set aside as test data, and runs the BDRA algorithm on the remaining training data. If 1 is specified, the program will set aside a single observation and run the BDRA algorithm; however, it will do this for each observation, and report results that are averages across all the runs. This option is only valid in evaluation mode.

REFERENCES

[1] M. Kechadi B. Huang. A fast feature selection model for online handwriting symbol recognition. *Proceedings of the 5th International Conference on Machine Learning and Applications*, 2006.

[2] R. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961.

[3] C.C. Taylor D. Michie, D.J. Spiegelhalter. *Machine Learning, Neural and Statistical Classification*. Addison-Wesley Publishing Company, 1994.

[4] R. Setiono H. Lui. Feature selection via discretization. *IEEE Transactions on Knowledge and Data Engineering*, 2004.

[5] F. Durugollu A. Amira M. Tahir, A. Bouridane. Feature selection using tabu search for improving classification rate of prostate needle biopsies. *Proceedings of the International Conference on Pattern Recognition*, 2004.

[6] P. Willett R. Lynch. Adaptive classification by maximizing separability with respect to the unlabeled data. AeroSense, 2003.

[7] P. Willett R. Lynch. Bayesian clasification and feature reduction using uniform dirichlet priors. *IEEE Transactions on Systems, Man, and Cybernetics*, 2003.