TREATMENT LEARNING: IMPLEMENTATION AND APPLICATION

by

YING HU

B.Eng., University of Electronic Science and Technology of China, 1996

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

(Department of Electrical and Computer Engineering)

We accept this thesis as conforming
to the required standard

. . . .

. . . .

. . . .

. . . .

THE UNIVERSITY OF BRITISH COLUMBIA

May 2003

Abstract

Data mining and machine learning focus on inducing previously unknown, potentially useful, and ultimately understandable information from data. In this master's thesis, we propose a new learning approach called treatment learning. Treatment learning aims at mining a small number of control variables in a large option space that can lead to better system behavior. It addresses two central issues in data mining: (1) the understandability of learnt theories; (2) how can the learnt theories benefit decision making.

We design and implement a novel mining algorithm and deliver two treatment learners that are freely downloadable from an online distribution. We describe the implementation details of both learners and compare them through algorithmic performance analysis.

We conduct extensive data experiments and case studies to demonstrate the effectiveness of using treatment learner to seek a small number of control variables that constrain the option space to a tight, near-optimal convergence.

We compare treatment learning with other learning schemes in the framework of feature subset selection for supervised classification. Our treatment learner selects smaller feature subsets than most other methods with minimal or no loss in classification accuracy. Treatment learner has been successfully applied to various research domains through a collaboration with other researchers. By presenting four examples, we show the general paradigms of using it for decision making.

Table of contents

List of Tables

List of Figures

Preface

This thesis describes the research proceeded during the pursuit of my Master's degree. The work presented in this thesis has resulted in the following publications:

- "Agents in a Wild World", a book chapter in "Formal Approaches to Agent-Based Systems" [MH02]
- "Model-based Tests of Truisms", in the proceedings of IEEE ASE 2002[MRoS$^+$02]
- "Condensing uncertainty via Incremental Treatment Learning", in "Annals of Software Engineering" [MCF$^+$02]
- "Reusing models for requirements engineering" [MH01b] and "Constraining discussions in requirements engineering" [MH01a], for the "First International Workshop on Model-based Requirements Engineering"

Also, currently under review, are the following two submissions:

- "Data Mining for Busy People", submitted to IEEE Computer
- "Just Enough Learning (of Association Rules)", submitted to the Journal of Data and Knowledge Engineering

The software package discussed in this thesis has been used by other researchers in the following papers:

- “Should NASA embrace Agile Processes?” [Sm02]
- “Metrics That Matter” [MSCM02]
- “Model-based Tests of Truisms” [MRoS$^+$02]
- “What Makes Finite-State Models more (or less) Testable” [DO02]

I would like to thank Dr.Tim Menzies, my supervisor for his vision, insight and guidance. Dr.Menzies helped me by means of encouragement and enthusiasm, without him, this thesis would not be possible. I also wish to thank my husband, for his constant support.

Chapter 1

Introduction

We are living in an information age where powerful database systems for data collecting and managing are in use in virtually all companies, accumulating data on operations, activities and performance. The need for automated extraction of useful information (e.g., trends and patterns) from huge amounts of data has led to the rapid development of knowledge discovery and data mining techniques. *Knowledge Discovery in Database* (KDD) is defined as the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data [UPSS96]. *Data Mining* is a step in the KDD process. It consists particular algorithms that, under some acceptable computational efficiency limitations, produce a particular enumeration of the required patterns. Although data mining gains its popularity only recently, the similar concept has been well researched in an artificial intelligence area called machine learning. *Machine Learning* concentrates on induction algorithms and on other algorithms that can be said to "learn". Because both data mining and machine learning aim at inducing previously unknown, and potentially useful, information from data, we use the two terms interchangeably.

Despite the explosion in the development of data mining, there remains two central issues:

- Understandability of the learnt theory.
- How the knowledge gained from data actually benefit decision making?

In this thesis, we introduce the concept of *treatment learning*. Treatment learning is a new learning approach that aims at identifying a small number of control variables in a large search space. In the rest of this chapter, we address the above issues respectively to bring in the distinguishing features of treatment learning.

1.1 Simplicity First Methodology

Theories learnt by different learners vary widely in term of their explanatory value. At one extreme, some learning algorithms output theories too intricate to be understood by human. For example, it is difficult to understand the prediction system of a neural network merely by studying the net topology and individual node weights. If a particular prediction is in some sense surprising to the end-user, it is harder to establish any rationale for the value generated. By comparison, decision tree learners are commonly considered to be easily understandable. By explicitly enumerating rules used by the prediction system, such learners can lead to insights about the data.

Improving both accuracy and simplicity is one of the goals of machine learning research. In the past, researchers have designed learning algorithms with a strong bias toward short rules. One such bias is Ockham's Razor, a principle proposed by William of Ockham in the fourteenth century. It states that "entities should not be multiplied unnecessarily", which is normally interpreted as "keep it simple". In machine learning, when we face two theories with the same predictions and the available data cannot distinguish between them, Ockham's Razor favors the simplest one.

There are always tradeoffs between simplicity and accuracy. Based on different emphasis, two methodologies exist in parallel:

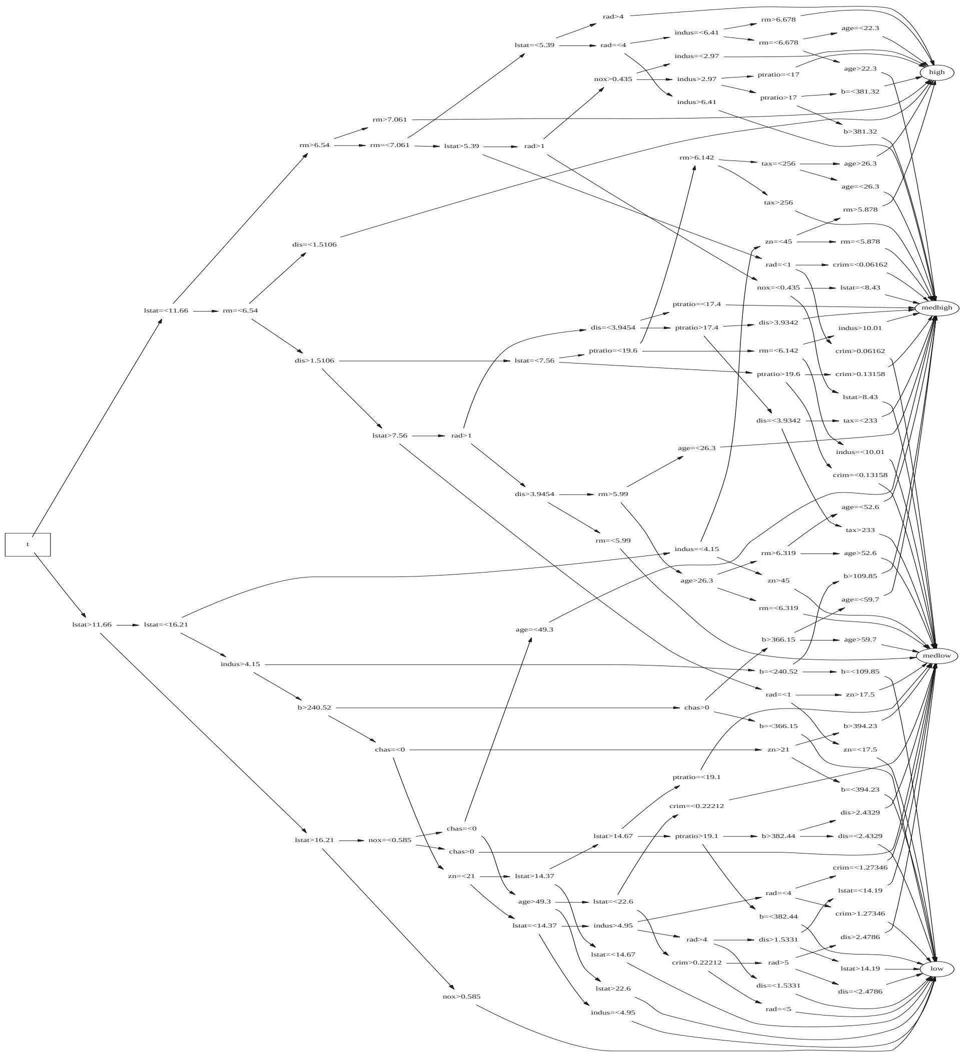
- Traditional methodology encourages learning algorithm to search through very large hypothesis space containing complex hypothesis.

- Alternatively, a "simplicity first" methodology directs learning algorithm to search through a relatively small space containing only simple hypothesis.

Although it is perfectly possible that simple hypothesis can be produced by using the traditional methodology, we emphasize that simplicity first approach offers attractive features. Firstly, systems designed using this methodology are guaranteed to produce theories that are near-optimal with respect to simplicity. Secondly, accuracy of the simple theory provides a baseline for more complex ones. In other words, increased complexity must be justified by increased accuracy.

Treatment learning adopts the simplicity first methodology. It produces minimal model of the target domain. If the theory found by treatment learner is unsatisfactory, then there does not exist a simple satisfactory theory. As a result, the output of treatment learner is small, simple, understandable and

Figure 1.1: A decision tree learnt from HOUSING data set from UCI data repository `http://www.ics.uci.edu/~cmerz/mldb.tar.Z`.



easy to interpret.

For example, figure 1.1 shows a decision tree learnt by C4.5 from the same HOUSING dataset. The dataset comes from the UC Irvine machine learning data repository [CEC98]. It contains 506 examples of *high, mediumHigh, mediumLow*, and *low* quality houses in the Boston area. The decision tree is a predicting system. Given a new housing record, the system can predict what quality this house might be classified. Although decision trees are one of the most explainable learners, the tree is still too complex for human to understand. If a person wants to find high quality houses in this area, giving him this tree doesn't provide intuitive guidelines for his house hunting process. The tree doesn't tell him what elements to watch for or what elements to avoid, which are the keys for human to make decisions. Treatment learning, on the other hand, gives this kind of information as we should see in the next section.

1.2 Classification vs. Class Comparison

In most domains, data are grouped into several comparable categories or classes. This resembles the natural way of human learning. Two different data mining tasks arise when understanding classes: classification and class comparison. Classification is one of the most well researched and widely used techniques. It analyzes the data to construct one or a set of models, and attempts to predict the behavior of new data examples. Models usually take the form of decision trees, rules, neural nets or Beyesian belief networks. Class comparison focuses on mining descriptions that distinguish a target class from its contrasting classes. Intuitively, both techniques discover differences between classes. If an attribute is highly relevant with respect to a given class that it can be used to classify novel instances of that class, then it is likely that the values of the attribute can distinguish the class from others. However, classification and class comparison put quite different emphasis on the interpretation of discovered theories by human domain experts. Rubinstein and Hastie [RH97] argue that the goal of automatically finding classes differences can be approached from 2 points of view:

1. discriminative: where the algorithm focuses on learning the class boundaries without regard to the underlying class densities. This way, it attempts to find differences that are useful for predictive classification with a high degree of accuracy.

2. Informative: where the algorithm learns the class densities, and attempts to find significant differences in the class descriptions, some of which may also be highly predictive but are not necessarily so.

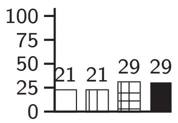
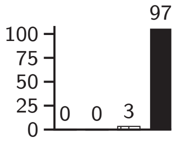
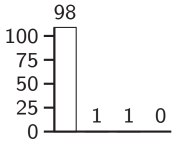
Most classifiers are discriminative miners. The learnt theory can be thought as a black box: once the box is ready, all we care is whether it could output a correct classification when giving a new example. Unlike classifiers, treatment learner takes the informative approach. It learns the class distribution, and seeks for solutions (e.g., treatments) that could most change the distribution in a user preferred way. Theory learnt from treatment learner is more like a white box: we actually look into it and want to understand why this would lead to certain direction. By emphasizing on the understandability of the theory, treatment learning generates insights into the target domain and inspires decision making. Take the HOUSING example, treatment learner outputs a theory shown in figure 1.2. The left most plot in that figure shows the quality distribution of houses in that area: among the 506 houses, 21% is of poor quality while 29% is very good. For house hunting policy within the area, treatment learner offers two strategies (Visualized in the middle and the right plot of figure 1.2 respectively):

$$rule\ 1 \begin{cases} IF: & 6.7 \leq RM \leq 9.8\ AND\ 12.6 \leq PTRATIO \leq 15.9 \\ THEN: & 97\%\ \text{of the found houses will be}\ \textit{high}\ \text{quality} \end{cases}$$

$$rule\ 2 \begin{cases} IF: & 0.6 \leq NOX \leq 1.9\ AND\ 17.16 \leq LSTAT \leq 39 \\ THEN: & 98\%\ \text{of the found houses will be}\ \textit{low}\ \text{quality} \end{cases}$$

Despite any domain specific details, the above two rules are easy to interpret and helpful for people seeking houses in that area.

Figure 1.2: Treatments learnt in the same domain

	baseline	`controllerH` (i.e. best action)	`monitorH` (i.e. diaster if..)
Treatment	nothing	$6.7 \leq RM < 9.8$ $\wedge 12.6 \leq PTRATION < 15.9$	$0.6 \leq NOX < 1.9$ $\wedge 17.16 \leq LSTAT < 39.0$
Results			
N	506	38	81

Attributes used in the treatments:

rm	=	number of rooms
ptratio	=	parent-teacher ratio at local schools
nox	=	nitric oxides concentration
lstat	=	living standard

1.3 Treatment Learning

In summary, treatment learning addresses the understandability and decision making issues of data mining by providing two attractive features:

- It takes a simplicity first methodology. As a result, it seeks to generate minimal theories that are small, simple, easily understandable from the target domain.
- It approaches learning in an informative way, emphasizing on the interpretation of the learnt theory by human experts to inspire decision making.

We present this novel learning approach, and advocate the use of treatment learning as an alternative to other, more elaborate learning algorithms. As shall be seen in the rest of the thesis, treatment learning has been successfully applied to numerous research domains such as software engineering, requirement optimization and attribute subset selection. Treatment learning contributes to the data mining community a new learning concept, a readily accessible learner and a wide applicability.

1.4 Organization

This thesis discusses four main topics:

1. Introduction of treatment learning in the context of machine learning and data mining.
2. A detailed description of treatment learning by providing algorithm implementation and performance comparison of two treatment learners.

3. A evaluation of treatment learning with respect to other state-of-the-art techniques in the framework of Feature Subset Selection (FSS) for supervised classification.
4. Application of treatment learning in various research domains.

The above topics are organized as follows:

In chapter 2, we present a literature review that serves as background of this thesis. Two groups of concepts and techniques are outlined: one is supervised classification in machine learning, the other is association rule mining in data mining. We also review some recent development in integration of classification and association rule mining. All of them are closely relevant to the topics discussed in this thesis, and represent the state-of-the-art in each of these areas.

In chapter 3, we first bring forward the concept of *narrow funnel effect*: an observation repeated in many researches, where most domain variables are controlled by a very small subset. We then introduce treatment learning as an ideal way to identify funnel variables: a lightweight learning approach that focuses on producing the *minimal* models to describe significant differences among groups of data. We go deep into the problem by presenting implementation details of a treatment learner TAR2. This is followed by two case studies illustrating the effectiveness of using treatment learner in practice for actionable decision making. Finally, we relate treatment learning to extensions of standard learning techniques and general change detecting algorithms to show their differences and the novelty of our approach.

In chapter 4, we examines the algorithmic performance of the learner described in the previous chapter. We point out its efficiency limitation by reporting runtime curves with respect to parameters such as data size and treatment size. After analyzing the search procedure that leads to the problem, we solve it by employing a series of strategies, including a random sampling algorithm. The improved learner TAR3 is evaluated through comparison experiments with TAR2 and a revised case study. The results show that TAR3 has made major improvement in efficiency: it can reach stable conclusions in linear time.

In chapter 5, we further explore treatment learning in the framework of Feature Subset Selection for supervised classification. Feature subset selection is the process of identifying and removing as much of the irrelevant and redundant information from data as possible prior to learning. We use treatment learner as feature subset selector on ten commonly used datasets and compare the result with six standard techniques. Experiments show that our approach is the best overall feature subset selection method. It finds the smallest feature subsets with minimal or no loss in classification accuracy.

The test of any technique cannot be how much the inventor likes it. Rather, it is how much *other people* wants to use it. In chapter 6, we describe how other researchers have used the software developed in this thesis. We present real world applications of treatment learning to demonstrate how it can be integrated into different research frameworks to assist decision making. We present studies in four domains:

1. Assessment of software development paradigm
2. Project quality analysis using software metrics
3. Study of software inspection policies
4. Testability analysis of Finite-State Models

Among them some are model-based while others are data-present. In either case, we give brief background and state the approach and goal of the study. Although each case is discussed from a domain-specific point of view, we emphasize the general applicability of treatment learning and the approach of modelling the problem such that we can make decisions by identifying minimal key factors in the domain.

In chapter 7, we conclude this by reviewing the main contributions of our research and pointing out future research issues.

Chapter 2

Literature Review

We present in this chapter a literature review which is closely related to the topics discussed in this thesis. We also provide basic definitions associated with data mining and supervised machine learning. They serve as a background to the thesis and will be frequently used throughout it. Some of the definitions will be repeated or emphasized again if necessary, other additional, non-frequently used definitions will be given when required.

Classification and association rule mining are two major topics in the review. Typical algorithms as well as other commonly used methods in each field are discussed. Association based classification methods have attracted much attention in recent years, showing an increasing interest in integrating the two approaches for real world application. Our goal is to provide a representative sample of the research in each of these areas.

2.1 Classification

2.1.1 Introduction

The problem of classification has been well studied and continued to be an important research topic in the fields of machine learning, pattern recognition, statistics over decades, and recently in the database and data mining communities. Classification is defined as the process of finding a set of models(or functions) that describe and distinguish data classes or concepts, for the purpose of being able to use the model to predict the class of objects whose class label is unknown [JH01]. The derived model may be represented in various forms, such as classification (IF-THEN) rules, decision trees, mathematical formulae, or neural networks.

Typically, a classification task consists of two steps:

- The learning phase: In this phase, a model is constructed by analyzing data examples described by attributes. Each data example is assumed to belong to a predefined class(e.g., edible or poisonous, play or don't play), they collectively form the training data set. This step is also known as *supervised learning*, metaphor being that the learning of the model is "supervised" explicitly by the class label of each training example.

- The testing phase: In this step, the accuracy of the learnt model is estimated on the test data set. If the accuracy is acceptable, the model is used to classify future data examples for which the class label is unknown.

Two different learning strategies are employed for classification: *eager learning* and *lazy learning.* Lazy learners [Aha97] (also called instance-based learning) defer processing of training examples until requests for classification of new data examples are received. They accomplish classification by combing their stored (e.g., training) data and discard the constructed answer as well as any intermediate results. In contrast, eager learners greedily compile their inputs into an intensional concept model (e.g., rule set, decision tree, or neural network), and in this process discard the inputs. They then use this induced model for classification.

Decision tree induction and neural networks are examples of eager learning method, nearest neighbor classifiers are typical learners using the lazy learning strategy. In the following section, we outline 2 representative algorithms: C4.5 and k-nearest neighbor. To show the diversity of approaches, other methods such as backpropagation, Naive Bayesian classifier and 1R are also discussed.

2.1.2 Decision Tree Induction

Ross Quinlan's work on ID3 [Qui86] and C4.5 [Qui92] is widely acknowledged to have made some of the most significant contributions to the development of classification. The idea originates from the concept learning systems(CLS) [HS66].

A decision tree is a tree structure, where each internal node denotes a test on an attribute, each branch descending from that node corresponds to one of the possible values for this attribute, and leaf nodes represent classes [Qui92]. An instance is classified by starting at the root node of the decision

outlook	*temp(^oF)*	*humidity*	*windy?*	*class*
sunny	*75*	*70*	*true*	*play*
sunny	*80*	*90*	*true*	*don't play*
sunny	*85*	*86*	*false*	*don't play*
sunny	*72*	*95*	*false*	*don't play*
sunny	*69*	*70*	*false*	*play*
overcast	*72*	*90*	*true*	*play*
overcast	*83*	*88*	*false*	*play*
overcast	*64*	*65*	*true*	*play*
overcast	*81*	*75*	*false*	*play*
rain	*71*	*96*	*true*	*don't play*
rain	*65*	*70*	*true*	*don't play*
rain	*75*	*80*	*false*	*play*
rain	*68*	*80*	*false*	*play*
rain	*70*	*96*	*false*	*play*

Table 2.1: A small training set.

tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute. This process is then repeated at the node on this branch and so on until leaf node is reached. Table 2.1 is an example training data, and its corresponding decision tree is shown in figure 2.1. That tree has 100% accuracy on the 14 training instances. Decision trees can easily be converted into a set of decision rules.

The basic algorithm for decision tree induction is a greedy algorithm that constructs decision trees in a top-down recursive divide-and-conquer manner. It involves determining the attribute which is most discriminatory and then splitting the training instances into groups, containing multi-class instances or single-class instances, categorized by this attribute. Next, the process is repeated to further partition each group until every subgroup contains data of one class only. The key to constructing a decision tree is how to choose an

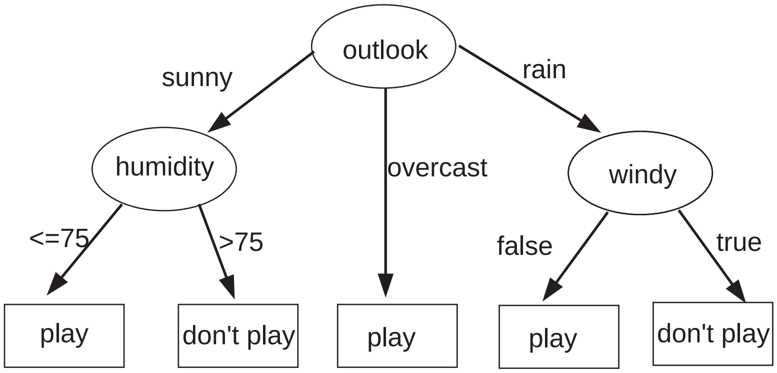


Figure 2.1: A decision tree corresponding to table 2.1.

appropriate attribute to divide the data, and subsequently other attribute values for subgroups.

ID3 [Qui86] uses an entropy-based measure known as *information gain* as a heuristic for selecting the attribute. The *information theory* underlying it states that: The information conveyed by a message depends on its probability and can be measured in bits as minus the logarithm to base 2 of that probability. Let S be a set containing s training examples, $C^i (i = 1..m)$ be one of m distinct classes. The *entropy* or expected information needed to classify a given sample is given by:

$$E(S) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

where $p_i = \frac{|S, C_i|}{|S|}$ is the probability that an arbitrary sample belongs to class C_i. Consider a similar measurement after S has been partitioned in accordance with the n distinct values of attribute A. The entropy based on the partitioning into subsets by A is the weighted sum over all subsets:

$$\begin{aligned} E(A) &= \sum_{j=1}^{n} \frac{|S_j|}{|S|} * E(S_j) \\ &= \sum_{j=1}^{n} \frac{|S_j|}{|S|} * (-\sum_{i=1}^{m} p_{ij} \log_2(p_{ij})) \end{aligned}$$

where $p_{ij} = \frac{|S_j, C_i|}{|S_j|}$ is the probability that a sample in S_j belongs to class C_i. The information gained by partitioning S in accordance with attribute A is:

$$Gain(A) = E(S) - E(A)$$

ID3 in its each iteration selects attribute to maximize this information gain criterion.

The gain measure is biased in that it tends to prefer attributes with many values. To avoid this, C4.5 uses *gain ratio* which considers the probability of each attribute value:

$$\begin{aligned} gain\ ratio(A) &= \frac{gain(A)}{split\ info(A)} \\ &= \frac{gain(A)}{-\sum_{i=1}^{n} \frac{|S_i|}{|S|} * \log_2(\frac{|S_i|}{|S|})} \end{aligned}$$

Experiments show that gain ratio is robust and typically gives a consistently better choice of test than the gain criterion [Qui88]. However, it is reported to have the tendency to favor unbalanced splits in which one subset is much smaller than the others [Min89]. Various other selection measures have been proposed, including Gini index of CART [BFOS84], distance-based measures [deM91] and the TARZAN tree query language (also a prolog prototype of the TAR2 system described in the later chapters) [MK01].

Decision Tree 1R

1R is a simple decision tree learner developed by Holte [Hol]. It reads in dataset and outputs 1-level decisions tree. The learning algorithm is straightforward:

- Divide the value range of continuous attribute into several disjoint intervals.
- Treat missing value as a legitimate value.
- For each attribute, construct a 1-level decision tree using that attribute by assigning class membership for each of its value. Class C is assigned to attribute A value V such that most examples having value V of attribute A belong to class C (i.e., $P(A.V|C)$ is maximum).
- Choose the decision tree that has the highest accuracy on the training set. This is the final output of 1R.

In summary, 1R ranks attributes according to the accuracy on the training set. The highest ranked attribute is selected to construct the 1-level decision tree.

1R was compared to C4 (the immediate ancestor of C4.5 [Qui86]) on 16 commonly used machine learning datasets. The experiment showed that 1R's 1-level trees, although much simpler, are only a few percentage points (3.1%) less accurate, on most of the datasets, than the decision trees produced by C4 [Hol]. The comparison offers two important implications:

1. 1R provides a benchmark accuracy for other, more sophisticated classifiers. More complex systems must justify their additional complexity with improved accuracy.

2. The comparison suggests that very simple learners can perform well in practice.

2.1.3 k-Nearest Neighbors

Nearest-neighbor methods are among the most popular for classification. They represent the earliest general(nonparametric) methods proposed for this problem and were heavily investigated in the fields of statistics and pattern recognition [CH67]. Recently renewed interest in them has emerged in the connectionist literature and machine learning. Despite their basic simplicity and the fact that many more sophisticated alternative techniques have been developed since their introduction, nearest-neighbor methods still remain among the most successful for many classification problems.

The nearest neighbor searching is: given a set of S of n points in a metric space X, the task is to preprocess these points so that, given any query point $q \in X$, the data point nearest to q can be reported quickly. The intuition behind the k-nearest neighbor classification is that the class label of a test instance is most likely to be the class prevailing the k nearest training examples of the test instance. Training examples are described by n-dimensional numeric attributes. Each sample represents a point in an n-dimensional space. When given an unknown sample, the k-nearest neighbor classifier searches the n-dimensional pattern space for the k training examples that are closest to the unknown sample. "Closeness" is defined in terms of Euclidean distances, where the Euclidean distance between two points,

$X = (x_1, x_2, ..., x_n)$ and $Y = (y_1, y_2, ..., y_n)$ is:

$$d(X,Y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

The unknown sample is assigned the most common class among its k nearest neighbors. Specifically, when $k = 1$, the unknown sample is assigned the class of the training sample that is closest to it in pattern space. Nearest neighbor classifiers do not require categorical class attributes, they could return a real-valued prediction for a given unknown example. Therefore, thy can also be used for regression, in which case, the classifier returns the average value of the numeric labels associated with the k nearest neighbors of the unknown example.

Recall that the eager learning classifiers such as C4.5 and backpropagation produce discriminating knowledge from training data before any individual decision is made, and maintain the knowledge unchanged unless new training instances are added. In contrast, nearest neighbor classifiers are lazy learners in the sense that they store all of the training examples and do not build a generalization model until a new sample needs to be classified. Since all the training is delayed to that time, lazy learners can incur expensive computational costs when the number of potential neighbors (i.e., stored training examples) is great. These costs become a serious issue in applications where many objects are to be classified in a very short time.

2.1.4 Other Classification Methods

In addition to decision tree and nearest neighbors, there are numerous other classification methods. In this section, we briefly describe neural nets and

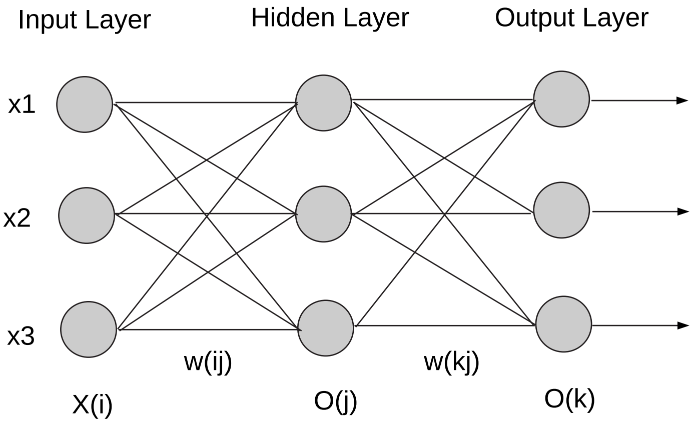


Figure 2.2: A sample multi-layer feed-forward neural network. Input (x_1, x_2, x_3) is fed to the input layer. Weighted connections exist between each layer, where w_{ij} denotes the weight from a unit j in one layer to a unit i in the previous layer.

bayesian networks.

Neural Networks

Neural networks [Ros62] were inspired by psychologists and neurobiologists who sought to develop and test computational analogues of neurons. Neural network is a set of connected input/output units where each connection has a weight associated with it. During the training phase, the network learns by adjusting the weights for each unit. After successful completion of training, the neural network architecture is frozen. When new instances traverse the network, they are multiplied by appropriate weights and the products are summed up. The output from one node serves as input to another node following the connection. This process repeats until the neural network generates an output value which determines the instance's class.

Different neural network models exist depending on how the units are connected: The network may be *feed forward* networks [Ros62] in which the output of one set of units is fed into another layer of units. Networks can be *recurrent* networks such as Boltzmann Machines [HS86] in which the output of a unit as well as being an input to other units is also an input to itself. Networks may either be fully connected (e.g., Hopfield networks [Hop82])or sparsely connected.

The most widely used neural network learning algorithm is backpropagation. It performs learning on a multi-layer feed-forward neural network such as the one shown in figure 2.2. Backpropagation learns by iteratively processing a set of training examples. In each iteration (called epoch), the mean squared error between the network's prediction and the actual class is calculated and propagated backwards from the output layer through each hidden layer down to the first hidden layer. The weights are updated, using gradient descent, to minimize the error. Although it is not guaranteed, in general the weights will eventually converge, and the learning process stops [Fau94].

Neural networks involve long training times. They require a number of parameters that are typically best determined empirically, such as the network topology. They have been criticized for their poor interpretability, since it is difficult for humans to interpret the semantics behind the learned weights [Hay99]. Advantages of neural networks, however, include their high tolerance to noisy data as well as their ability to classify pattern on which they have not been trained. Shavlik et. al., [SMT91] did extensive experiments comparing packpropagation with ID3. Their results show that backpropagation is more accurate than ID3 with small training set, when data contains numerical-valued features, and when examples are noisy or have more missing values. Recent efforts in speeding up the algorithms and extraction of rules from trained neural networks contribute towards the usefulness of neural networks for classification.

Naive Bayesian Classifier

Beyesian classifiers, originally presented by Duda and Hart [DH73] are statistical classifiers using Bayes' theorem to compute a probabilistic summary for each class. Beyes' theorem states that: the *posterior probability* of a hypothesis H conditioned on observation O is a function of the *prior probability* of the hypothesis H (i.e., the probability you would have assigned to the hypothesis before you made the observation). In mathematical formula, it is:

$$P(H|O) = \frac{P(O|H)P(H)}{P(O)}$$

the essence of the Bayesian approach is to provide a mathematical rule explaining how we should change our existing beliefs in the light of new evidence. In classification, we try to predict the class membership probability given the data examples.

Naive Bayesian classifiers assume that the values of the attributes are conditionally independent of one another given the class label of the example, that is, there are no dependencies among the attributes. Despite the fact that the assumption is only made to simplify the computation, Naive Bayesian classifier is reported to be comparable in performance with decision tree and neural network classifiers [Koh96].

When the class conditional independence condition holds true, Naive Bayesian classifier has the minimum error rate in comparison to all other classifiers. In practice however, dependencies can exist between variables. In this case, Bayesian belief networks which allow the representation of dependencies among subsets of attributes can be used instead. Bayesian classifiers are also useful in that they provide a theoretical justification for other classifiers that do not explicitly use Bayes theorem. For example, under certain assumptions, many neural network and curve-fitting algorithms output the maximum likelihood hypothesis, just as naive Bayesian classifier does [D.H96]
.

2.2 Association Rules

Association rule discovery [TA93] [AS94] differs fundamentally from classification rule discovery paradigms. While the classification concentrates on finding rules that are predictive of a single, predefined class label, association rule discovery has been motivated by finding rules that predict increased frequency of an attribute value, or collection of attribute values, without limitation on the values that may appear in the consequent of a rule. Association rule discovery can be distinguished by the aims of [Web00]:

1. discovering all rules that satisfy a given set of constraints,
2. an emphasis on processing large training sets, and
3. allowing any available condition to appear as either an antecedent or consequent.

As shown in the classification section, any machine learning system has its own bias. Since association rules do not filter through a machine learning system, it enables the user to identify the interesting rules rather than relying on a machine learner to determine the rules of interest. This makes association rule discovery a very valuable tool for discovering inter-relationships between variables in many different types of domains.

2.2.1 Background and Formal Definitions

Association rule discovery originates in basket (transactional) data analysis [TA93]. In transactional databases, the attributes are the names of merchandise such as *bread*, *milk* and *butter*. Such attributes are binary with the values of either 0 or 1. An attribute with the value 1 means that this merchandise was bought, otherwise it was not. An **item** is an attribute-value pair. Table 2.2 is a natural representation of the real transactions.

TID	*Transactions*
1	*A,B,C*
2	*B,C,E*
3	*B,C,D,E*
4	*B,D*

Table 2.2: A natural representation of transactions.

Definition: Support of an itemset Given a database D and an itemset X, the **support** of X in D is the percentage of transactions in D

containing X.

$$support(X) = \frac{|X|}{|D|} = P(X)$$

Definition: Large or frequent itemset Given a database D and a real number $\delta(0 \leq \delta \leq 1)$, and itemset X is defined as a **large** (or **frequent**) itemset if $support(X) \leq \delta$.

Definition: Association rules Given a database D, an **association rule** is an implication of the form $X \longrightarrow Y$, where X and Y are two itemsets in D and $X \cap Y = \emptyset$. The itemset X is the **antecedent** of the rule, and the itemset Y is the **consequent** of the rule.

Definition: Support and confidence of an association rule Given a database D and an association rule $X \rightarrow Y$, the **support** of the rule is the percentage of the transactions in D that contain both X and Y. The **confidence** of the rule is the percentage of the transactions in D that contain X, also contain Y.

$$\begin{aligned} support(X \rightarrow Y) &= \frac{|X \cup Y|}{|D|} = P(X \wedge Y) \\ confidence(X \rightarrow Y) &= \frac{|X \cup Y|}{|X|} = P(Y|X) \end{aligned}$$

For example, in table 2.2, the rule $B, C \rightarrow E$ has a support of 50%, and a confidence of 66.7%

Given a set of transactions D, the problem of mining association rules is to generate all association rules that have support and confidence greater than the user-specified minimum support (called *minsup*) and minimum confidence (called *minconf*) respectively [TA93].

This problem can be decomposed into two subproblems]:

1. Generate all large itemsets with respect to the support threshold *minsup*. A naive approach is to generate all itemsets and test them. In a data set containing n items per transaction, this would result in $2^n - 1$ itemsets (not including the empty set).
2. Use the large itemsets to generate rules whose confidence satisfy the threshold *minconf*. Note that, if $ABCD$ and AB are both large itemsets, we have $confidence(AB \rightarrow CD) = \frac{support(ABCD)}{support(AB)}$. Hence this problem is straightforward.

The efficiency issue of discovering all large itemsts has been extensively studied in both the database and data mining communities. There exists two widely used algorithms, the APRIORI algorithm [AS94] and the MAX-MINER algorithm [Bay98].

2.2.2 The APRIORI Algorithm

The basic idea in the APRIORI algorithm is to use the *Apriori property* of large itemset to narrow search space. The *Apriori property* states that: all non-empty subset of a large itemset is also large. In other words, if a k-itemset is not large, all its supersets are not large either. The APRIORI algorithm generates large itemsets in a level-wise manner (e.g., generates large k-itemset first, then large $(k + 1)$-itemset), following a two-step process:

- The join step: generating a collection of k-itemset candidates $\mathcal{C}_k$ by joining the large $(k - 1)$-itemsets $\mathcal{L}_{k-1}$.
- The prune step: going through the database and calculating the supports of the candidates to get the large k-itemsets $\mathcal{L}_k$. In this step, the

Apriori property is used to reduce the candidates before testing.

We use an example ([AS94], APRIORI-GEN function) to illustrate the main point in candidate generation. Suppose the collection $\mathcal{L}_3$ of all the large 3-itemsets in some database be $\{\{1,2,3\},\{1,2,4\},\{1,3,4\},\{1,3,5\},\{2,3,4\}\}$. After a joint step, the candidate collection $\mathcal{C}_4$ will be $\{\{1,2,3,4\},\{1,3,4,5\}\}$. In the prune step, the itemset $\{1,3,4,5\}$ will be removed from $\mathcal{C}_4$ because its subset $\{1,4,5\}$ is not in $\mathcal{L}_3$. As a result, only itemset $\{1,2,3,4\}$ is a candidate and needs support calculation.

The APRIORI algorithm achieves a good performance by reducing the size of candidate itemsets. However, in some situations where there exist long large itemsets or where a quite low support threshold is required, the APRIORI algorithm still suffers from heavy computational costs.

2.2.3 The MAX-MINER Algorithm

MAX-MINER Algorithm differs from APRIORI in both its output representation and prune strategy. Unlike APRIORI, the MAX-MINER algorithm doesn't explicitly output all large itemsets, it outputs only those large itemsets whose proper supersets are not large. Those large itemsets are called maximal large itemsets. They can be seen as a frontier boundary to separate the large itemsets from non-large ones. Because any subset of the maximal large itemsets is also large, this output implicitly and concisely represents all large itemsets.

There are two prune strategies used in MAX-MINER:

- superset-frequency pruning: if an itemset is large, then its subsets must also be large, hence there is no need to generate its subsets for support calculation.

- subset-frequency pruning: if an itemset is non-large, then its supersets must also be non-large and it's unnecessary to generate its supersets for support calculation.

Those prune strategies of MAX-MINER are implemented in set-enumeration trees [Rym92] by incorporating some heuristic. This allows MAX-MINER for looking ahead to quickly identify long frequent itemsets and short non-frequent itemsets so that both pruning can be used simultaneously. Compared to APRIORI, which only uses the second pruning, MAX-MINER has been shown to perform two or more orders of magnitude better on some data sets , especially when support threshold is low or the data set is high dimensional [Bay98].

Related work close to MAX-MINER include PINCER-SEARCH [LK98] and MAX-CLIQUE [ZPOL97].

2.3 Integration of Classification and Association Rule Mining

A range of different types of classification algorithms, including decision tree induction, nearest neighbor methods, error back propagation, Bayesian learning, have been discussed in section 2.1. Those algorithms arrive at a classification decision by making a sequence of micro decisions, where each decision is concerned with one attribute only. In this section, we study classification approaches based on association rule mining concept by describing two classifiers, the CBA classifier [BLM] and JEP classifier [LDR00]. We also briefly discuss a concept called contrast set mining, that mines a special case of associations in classified data.

2.3.1 The CBA Classifier

Classification Based on Associations (CBA) [BLM] is a successful classification method using the dependency of association rules. The basic idea of CBA is to discover a special type of association rule, called *class association rules* (CARs), satisfying the user-specified support and confience threshold requirements. The CBA classifier then selects the most interesting rules for classification.

In CBA, a CAR is an association rule whose consequence (or RHS:Right Hand Side) is restricted to the class label. The algorithm consists of two parts, a rule generater and a classifier builder.

1. Generating all CARs that satisfy user-specified support and confidence thresholds.
2. Evaluating all CARs and selecting the subset that gives the least number of errors.

In this step, CBA uses the following heuristic: Given two rules r_1 and r_2, the rule r_1 precedes r_2 if

1. the confidence of r_1 is greater than that of r_2; or
2. their confidence are the same, but the support of r_1 is greater than that of r_2; or

3. both the confidences and supports are the same, but r_1 is generated earlier than r_2.

CBA was reported more accurate than C4.5 as it outperforms C4.5 on 16 out of 26 datasets and decreases the average error rate [BLM].

A major disadvantage of CBA is that the number of discovered rules is usually very large. In their 26 experimented data sets, the average rule limit is 80,000. It is ironic that they claim standard classification has an understandability problem because those systems use domain independent biases and heuristics to generate a small set of rules. However, their huge rule set, although may be complete, is clearly overwhelming. Secondly, CBA relies on user-specified support and confidence thresholds to mine rules (by default, they set support=1%, confidence=50%). This could be tricky, since a high support dramatically degrades the classification accuracy while a low support results in long run times. Thirdly, CBA discretizes continuous attributes first, different discretization could lead to different collection of rules.

2.3.2 The JEP Classifier

The JEP classifier is based on the notion of *emerging patterns* (EPs). An EP is an itemset whose support increases significantly from one class of data to another. The ratio of the two supports is the *growth rate* of EP, i.e.,

$$growth\ rate(X) = \frac{support_{D_2}(X)}{support_{D_1}(X)}$$

where D_1, D_2 are two different classes and X is an itemset. The JEP classifier exploits the discriminating power of a special type of EPs called *jumping* *emerging patters (JEPs)*, whose support increases abruptly from zero in one class to non-zero in another–the growth rate being ∞ or 0.

Suppose we have a training set containing 3 classes, the JEP classifier works as follows:

- The training set $\mathcal{D}$ is partitioned into $\mathcal{D}_1$, $\mathcal{D}_2$, $\mathcal{D}_3$ 3 subsets, each has only 1 class.
- Because the JEP works in pairs, the 3 subsets are combined into 3 pairs, i.e., $\mathcal{D}_1/\mathcal{D}_2 \cup \mathcal{D}_3$; $\mathcal{D}_2/\mathcal{D}_1 \cup \mathcal{D}_3$; $\mathcal{D}_3/\mathcal{D}_2 \cup \mathcal{D}_1$. This process is called extracting *pair-wise features*. Then a border-based algorithm is used to identify the JEPs for each pair.
- JEPs with largest support (the *most expressive JEPs*) are collected by taking the union of the left bounds of the borders for each pair.
- When a test example is given, the classifier calculates the collective impacts in favor of 3 classes respectively, then the class with the largest collective impact is assigned to the test example.

The border-based representation of rules and the border manipulation algorithm are the distinct features of emerging patterns. Classifiers based on EPs are novel and fundamentally different from classic association rule mining. JEP classifier has been found very accurate: in 25 data sets where results of CBA and C4.5 are available, JEP outperforms both of them on 15 data sets; CBA wins on 5, C4.5 wins on 5. JEP classifiers also performs well on unbalanced data sets where the main class of interest is in minority. It scales up on data volume and dimensionality [LDR00].

Other classifiers based on emerging patterns is the CAEP classifier (classification by aggregating emerging patterns), which uses EPs with finite growth rates rather than JEPs. CAEP is considered complementary to JEPs, and is discussed in [DL99].

2.3.3 Contrast Set

Contrast set is a term proposed by Bay and Pazzarni. A contrast set is a conjunction of attribute-value pairs (similar to itemsets in association rule) defined on groups(classes). The task of mining contrast sets is to find all contrast sets whose support differs meaningfully across groups[BP01]. That is:

$$|P(X|y = c_i) - P(X|y = c_j)| > \delta; i \neq j$$

where X is the contrast set, y is the class variable, δ is a user defined threshold of minimum support difference. The discovery of contrast sets allows us to ask questions such as "what are the differences between people with Ph.D. and bachelor degrees?".

Bay and Pazzarni designed a complete mining algorithm called STUCCO to search for contrast sets. STUCCO operates through a combination of search and summarization. In the search stage, a set enumeration tree is constructed. STUCCO searches the tree in a level wise manner. For each set, STUCCO counts its support to determine whether it should be pruned or not. STUCCO also keeps careful track of a number of statistical tests made to check if a contrast set is significant[BP99]. In the summarization stage, STUCCO uses a filter algorithm to show user only a portion of the contrast sets discovered. The most general sets (those contain only single item) are shown first, then more complicated conjunctions. For example, it starts showing single sets: $set1 : sex = male$, $set2 : school = ECE$,

$set3 : GPA > 4$; it then shows: $set4 : sex = male \wedge school = ECE$; at last it shows: $set5 : sex = female \wedge school = ECE \wedge GPA > 4$. The conjunctions are only shown if their frequencies could not be predicted from the subsets using a log-linear model[BP01].

The concept of contrast set differs from both classification and association. It brings out the problem of detecting differences across groups or trends if the groups are temporal. STUCCO employs sophisticated statistical hypothesis testing in its search to find significant and insightful contrast sets. However, the "insightfulness" of a rule is extremely subjective. As a result, there is no apparent way to evaluate and benchmark patterns discovered by contrast set mining.

2.4 Summary

The reviews presented here concentrate on the fields of classification and association rule mining. We provided both basic definitions and algorithms for each task. The learning systems being described can be summarized into three groups:

- Classical algorithms that usually serve as benchmark for new methods emerged recently. For classification, such algorithms include C4.5 decision tree classifier, K-nearest neighbor lazy learner, backpropogation neural net and naive bayes classifier. For association rule mining, there are APRIORI algorithm and Max-Miner algorithm. These systems have been well studied in literature and widely applied in practice. They represent the state-of-the-art in each of their fields.

- Extensions to standard methods described above. This includes classifiers based on association rule mining concept. We presented two such classifiers, the CBA classifier and the JEP classifier. They are accurate classifiers constructed using new KDD patterns known as class based association and emerging patterns respectively.

- We also include two non-standard learning systems in our review: the 1R decision tree learner and the contrast set mining algorithm. Holte's 1R decision tree is a successful example of classifiers designed using the simplicity first methodology. Bay et.al. bring forward the concept of detecting differences cross contrasting classes. They stress two essential elements on which our research is based. More precisely, our treatment learning is a system that takes the simplicity first methodology to identify class differences.

The review has shown how the standard learners work in classification and association rule mining. In the following thesis, we will introduce our own approach to learning. We will compare it to some of related systems within this framework.

Chapter 3

Treatment Learning and The TAR2 Treatment Learner

3.1 The Narrow Funnel Effect

Improving both accuracy and simplicity is one of the goals of machine learning research. There are always tradeoffs between the two criteria, yet we have seen in history that the pursuit of high accuracy has attracted dominant attention. This research methodology encourages learning systems to search in very large hypothesis space containing, among other things, very complex hypotheses. However, when is just enough learning enough? Are complex hypotheses always necessary with respect to accuracy?

There are in literature some indications that many domains lack complex relationships. In other words, a small number of critical variables control the remaining others within a system. As a result, these domains can be "easy to learn" using lightweight approaches and can be adequately described by simple models. For example:

- In the experimental comparison of symbolic and neural learning algorithms, Shavlik et.al. investigated sensitivity of the algorithms to the number of features. They observe that randomly dropping half the features only slightly impairs performance of perceptron, ID3 and backpropagation. For some cases, performance even improves when a small number of features are dropped. They were surprised to notice the apparent redundancies in several domains and concluded that extra features could degrade inductive learning algorithms [SMT91]. Another result of their experiments is how well the simple perceptron algorithm (the simplest neural network) performs. Despite its inherent limitations, the accuracy of perceptron is hardly distinguishable on most datasets from the more complicated learning algorithms [SMT91].
- In decision tree classification, Holte reports the results of experiments measuring the performance of a very simple decision tree learner 1R on 16 datasets commonly used in machine learning research [**?**]. The 1-level decision trees produced by 1R are only a few percentage points less accurate than the more elaborated decision trees produced by C4 [Qui92]. He also examined whether or not the datasets used in his study have been particularly engineered to make induction easy. The investigation has shown that most of them are representatives of datasets in practice.
- In data engineering, Kohave and John studied a specific feature subset selection method. Their experiments show that, on 8 real world datasets, an average 81% features can be ignored. Further, ignoring

those features doesn't degrade the learner's classification accuracy, on the contrary, it results an average increase of 2.14% [KJ97] (see table 3.1).

dataset	before	after	retain	accuracy change
breast cancer	10	2.9	0.29%	+0.14%
cleve	13	2.6	0.2%	+5.89%
crx	15	2.9	0.19%	+4.49%
DNA	180	11	0.06%	+3.63%
horse-colic	22	2.8	0.13%	+1.63%
Pima	8	1	0.13%	+0.79%
sick-euthyroid	25	4	0.16%	+0.38%
soybean	35	12.7	0.36%	+0.15%
average	38.5	4.99	0.19%	+2.14%

Table 3.1: Feature subset selection results from Kohavi and John, [KJ97]

In summary, the above experiments are saying that within a very large space of attributes, there are a few key values that matter most. A similar effect has been seen outside the machine learning literature. For example, in an application of satisfiability algorithms to scheduling problems, Crawford and Baker [CB94] compared TABLEAU, a depth-first search backtracking algorithm, to ISAMP, a randomised sampling algorithm. Both algorithms assign a value to one variable, then infer some consequences with forward checking. After the checking, if a contradiction was detected, TABLEAU backtracks while ISAMP simply starts over and re-assigns other variables randomly (giving up after `MAX-TRIES` number of times). Otherwise, they continue looping till all variables are assigned. Table 3.2 shows the relative performance of the two algorithms on a suite of scheduling problems based on real-world parameters. Surprisingly, ISAMP took *less* time than

	TABLEAU:		ISAMP:		
	full search		partial, random search		
	% Success	Time (sec)	% Success	Time (sec)	Tries
A	90	255.4	100	10	7
B	100	104.8	100	13	15
C	70	79.2	100	11	13
D	100	90.6	100	21	45
E	80	66.3	100	19	52
F	100	81.7	100	68	252

Table 3.2: Average performance of TABLEAU vs ISAMP on 6 scheduling problems (A..F) with different levels of constraints and bottlenecks. From [CB94].

TABLEAU to reach *more* scheduling solutions using just a small number of `TRIES`. Crawford and Baker offer a speculation why ISAMP was so successful: the variables in scheduling problems can be grouped into *control* variables that define a solution and *dependent* variables whose values are derived from the control variables [CB94]. They further hypothesized that the solutions are not uniformly distributed throughout the search space. The depth-first search sometimes wonders into the deserts containing no solutions by making an early unlucky choice. On the other hand, randomized sampling effectively searches in a smaller space since it restarts on every contradiction [CB94]. Systems containing such features are not difficult to find solutions and a few key tests are sufficient to set the control variables.

We call this class of phenomena *narrow funnel effect*, the metaphor being that within the decision space, all pathways connecting inputs to desired goal run down the same narrow funnel [MENW99]. The core intuition in this term is that: what happens in the total space of a system can be controlled by a small critical region. Where the narrow funnel exists, the space of options within a large space reduces to just the range of a few variables within the narrow funnel. Machine learning in such domains could be very simple: an adequate theory needs only comment on assignments to the variables inside the funnel. By definition, any reasoning pathway to goals must pass through the funnel if it exists. Hence, one way to find the funnel is to find input variables that are associated with desired goals. Treatment learning is a machine learning method aims at seeking funnel variables. It is designed using the "simplicity first" methodology: treatment leaner searches through a relatively small space containing only simple hypothesis. Treatment learning is both a test and an application of narrow funnel effect. In domains containing narrow funnels, treatment learner will find them and generate very small, simple theories that are easier to understand. A unsatisfactory simple theory learnt by treatment learner suggests that the domain contains complex relations, hence other, more elaborate learning scheme should be tried.

3.2 Treatment Learning

In the context of data mining, treatment learning mines *minimal contrast set with weighted classes*. Conceptually, a treatment R_X is a conjunction of attribute-value pairs. Given a classified data set, treatment learner seeks a *treatment* R_X that returns a subset of the training set $D' \subseteq D$ with *higher frequency* of preferred classes and *lower frequency* of undesired classes than in D. Here, D' contains all examples that don't contradict the treatment; i.e. $D' = \{D \cap R_X\}$. In the following sections, we discuss treatment learning by provide the implementation details of a specific learner TAR2.

3.2.1 Problem Specification: Input/Output

TAR2 takes classified data sets such as the one shown in figure 3.1:

occupation	*age*	*city*	*gender*	*salary*
sales	*45*	*Calgary*	*male*	*medium*
engineer	*29*	*Toronto*	*male*	*medium*
cashier	*22*	*Victoria*	*female*	*low*
manager	*42*	*Vancouver*	*male*	*high*
...	...	...	...	...

Figure 3.1: A example data set.

The sample data set contains 4 attributes, among which 3 take categorical values, 1 (e.g. age) takes continuous values. The class label takes on categorical values from set $\{low, medium, high\}$.

Before showing the output form, we state the following assumptions and concepts:

Assumption There exists a partial ordering among the classes, where one class is considered superior than the others and referred to as the *best* class. Similarly, there exists a *worst* class which is the least desirable. A scoring function returns weights for each class, denoted by Score(class). The scoring function models the domain-specific view of the relative merits of the classes.

Definition Let $A_1, A_2, ...A_k$ be a set of k attributes. Each A_i takes on values from the set $\{V_{i1}, V_{i2}, ...V_{im}\}$(continuous values are discretized before process). A treatment R_X is a conjunction of attribute-value pairs that have different levels of confidence with respect to each class.

Definition The **confidence** of a treatment Rx with respect to a particular class C is the conditional probability of that class given the treatment is true.i.e.,

$$\begin{aligned} confidence(Rx\ w.r.t.\ C) &= P(C|Rx) \\ &= \frac{\text{\#of examples where C \& Rx are true}}{\text{\#of examples where Rx is true}} \end{aligned}$$

The general rule form of the output is:

$$\boxed{\begin{array}{lll} R & IF & Rx:\ Attr_1 = V_{a1} \wedge Attr_2 = V_{a2} \wedge ... \\ & THEN & class(C_i)\ \ : confidence(Rx\ w.r.t.\ C_i) \end{array}}$$

where R is a set of rules containing treatments that have significant higher confidence in best class and significant lower confidence in worst class compared to the raw data set. (Note that the best and the worst class is defined by the scoring function according to the user's preference.) The number of pairs appeared in the conditional of a treatment(i.e., in the IF statement) is called the *treatment size*. When applying the treatment, it constrains the original data set so that only a subset of examples in which the treatment holds is returned. This subset is referred to as *treated*. The rule set indicates an improvement in class distribution in the treated subset. For the example data set shown above, the output could be:

$$\boxed{\begin{array}{lll} R_1 & IF & occupation ='' manager'' \\ & THEN & \textit{salary="low":0} \\ & & \textit{salary="medium":20\%} \\ & & \textit{salary="high":80\%} \end{array}}$$

Rule R_1 returns a treatment of size 1: *occupation* $=''$ *manager*$''$. The change of class distribution caused by this treatment is visualized in figure 3.2. Rule R_1 is also called a *controller rule* as it favors the best class. Rules that favor worst class are called *monitor rules*, which point out things we want to avoid.

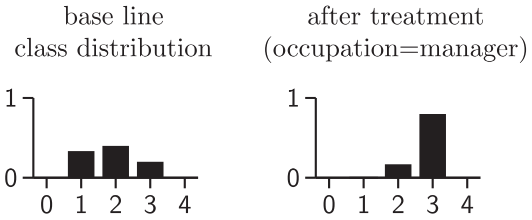


Figure 3.2: class distribution seen in the original salary data set and the subset after applying the treatment [occupation=manager]. Three bars correspond to 3 classes (”low”,”medium”,”high”) respectively. Height of each bar indicates the percentage of examples fallen into that class. The original data set contains 100 examples while the treated subset contains only 25.

Treatment Assessment

We use the notion of **lift** to numerically evaluate the merit of each treatment.

Definition The *worth* of a dataset D in terms of class distribution is defined as a weighted probability sum across all classes in the dataset:

$$worth(D) = \sum_{classes} Score(C_i) * P(C_i)$$

Definition The *lift* of a treatment Rx is the ratio of the *worth* of the treated subset to the *worth* of the baseline dataset D. i.e.,

$$lift(Rx) = \frac{worth(D \wedge Rx)}{worth(D)}$$

where $(D \wedge Rx)$ is the treated subset, i.e., in which Rx is true for all the examples.

Note that if $lift(Rx) > 1$ indicates an improvement of the class distribution. The goal of treatment learning is to find treatments that generate a large *lift*. The notion of *lift* distinguishes a treatment from decision rules. Take rule R_1 for example: treatment *occupation* = *"manager"* does not implies *salary* = *"high"*. It actually means: in the subset where the treatment *occupation* = *"manager"* is true , the percentage of examples whose *salary* = *"high"* is much higher than those in the original data set while the percentage of examples whose *salary* = *"low"* is significantly lower. In other words, treatments are constrains that change the original class distribution in the resulting subset. Class distribution favors the better classes after applying controller rules.

3.2.2 The Algorithm

Treatment learning involves a combination of search and attribute utility estimation. It produces item ranking which demonstrates the relative merit of individual attribute-value pair for changing the class distribution.

Discretization

TAR2 accepts both categorical and continuous attributes. Internally, it treats all the attributes uniformly. For a categorical attribute, all the possible values are mapped to a set of consecutive positive integers. For a continuous attribute, its value range is discretized into intervals, and the intervals are then mapped to consecutive positive integers. With these mappings, a data example is treated as a set of (attribute, integer-value) pairs (also called items) along with a class label.

TAR2 uses *Equal Width Interval Binning* to discretize continuous attributes [DKS95]. In this procedure, TAR2 first sorts the observed values of a continuous attribute, and then divides them into k equally sized bins, where k is a configurable parameter.

Confidence1 Measure

TAR2's core strategy is the *confidence1* measure. With ordered classes, TAR2 associates each class with a score(weight). The highest scoring class is the best class C_{best}, others are non-best classes $C_j, (j \neq best)$. The Scoring function could be customized to reflect user preference. Let:

- $a.r$: attribute a takes value r. (if a is a continuous attribute, r denotes one of its possible ranges.)
- $|a.r|$: the number of examples whose attribute a takes value r
- $|a.r, C_{best}|$: the number of examples whose attribute a takes value r and belong to class C_{best}.
- $|a.r, C_j|$: the number of examples whose attribute a takes value r and belong to class C_j, where $j \neq best$.
- $S(C_{best}), S(C_j)$: Score of class C_{best} and C_j respectively, returned by the scoring function.

The *confidence1* measure $\triangle_{a.r}$ for an attribute value pair $a.r$ is:

$$\triangle_{a.r} = \frac{\sum_j (S(C_{best}) - S(C_j))(|a.r, C_{best}| - |a.r, C_j|)}{|a.r|}$$

The attributes in our "salary" example has a confidence1 histogram shown in figure 3.3

$confidence1$ is a heuristic that measures the weighted difference of an item's confidence on non-best classes with respect to the best class. It differs significantly from the standard association rule $confidece$ definition:

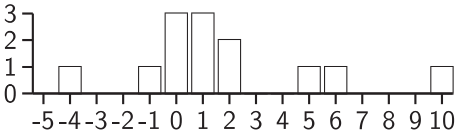


Figure 3.3: confidence1 histogram seen in "salary" data. Each bar denotes a particular $\triangle$ value. Height of each bar indicates how many attribute value pairs have that $\triangle$ value

1. it only studies an single item at a time (hence the name $confidence1$).
2. it focuses on the *confidence difference* of an item between classes than the item's $confidence$ value itself.
3. it weights the difference according to the comparative score of each class.

Reporting Treatments

After getting that $confidence1$ distribution, TAR2 explores subsets of items whose $confidence1$ value are above a certain threshold using a depth-first search. Each treatment is then evaluated by its *lift*. $lift > 1$ indicates an improvement. TAR2 only reports treatments with $lift(Rx)$ above a certain threshold. Display of treatments takes the visualization form, as shown in figure 3.2.

Cross Validation

Cross validation is a standard method for estimating generalization error based on "re-sampling". TAR2 comes with a N-way cross validation facility that allows user to divide the data into N subsets of approximately equal size, with each subset's class distribution as uniform as possible [Qui86]. TAR2 is then run N times, in each run, one subset is used as the test set and the other

N-1 subsets are put together to form a training set. TAR2 learns treatments from the training set and tests them on the test set. After the validation, N output files and one summary file are generated, recording treatments learnt from each run. In this procedure, each example appears exactly once in the test set. N is commonly set to 10.

Configuration File

TAR2 encourages user's interference of the learning process by providing a configuration file for data processing. In that file, there are several optional sections:

- NOW section: NOW specifies the current status of the data, i.e., only attributes satisfying NOW criteria will be read in and processed by TAR2. User could always use other tools to pre-process the dataset instead of configuring the NOW section.
- CHANGES section: CHANGES represents some desired zone within the data set that the user wishes to approach. Only attribute values specified in CHANGES would appear in the treatments.
- SCORE section: SCORE encodes user's preference of the classes. User can assign a specific score (weight) to a class. Without the specification, TAR2 scores the classes according to a default scoring function: $score(C) = 2^n$, where n is the order of the class C.

Those three sections restrict the data processing scope of the input dataset. A little language is designed to specify attribute ranges in NOW and CHANGES sections, for example:

```
Attribute1:true -- all possible values are acceptable
Attribute2:ignore -- none values are acceptable
Attribute3:a, b, c -- for categorical attribute, only values
           a, b, c are acceptable
Attribute4:[-;10), [20;30], [50;-) -- for continuous attribute,
           the acceptable ranges are: x<10 OR 20<=x<=30 OR x<=50
```

3.2.3 The TAR2 Software Package

Treatment learner TAR2 (current version 2.2) is coded in C and distributed under the GNU free software license. The TAR2 (and TAR3, described later) system is accessible online at: `http://www.ece.ubc.ca/twiki/bin/view/Softeng/TreatmentLearner`. The system is a complete package including the following contents:

- `/bin`: folder containing all executables
- `/doc`: folder containing the user manual and several related publications
- `/src`: folder containing all C source code
- `/samples`: folder containing sample data sets and corresponding output files.
- `readme.txt`: file containing general information of the software
- `COPYRITE.txt`: file containing the GPL-2 copy policy

We have actively maintained the online distribution of the TAR2/TAR3 system. The easy access from other researchers has encouraged a wide application of treatment learning on various domains (described in chapter 6).

3.3 Case Study 1: Risk Assessment

This section presents a case study tot demonstrate how treatment learning can benefit decision making when dealing with uncertainties.

3.3.1 Modelling

In the model-based requirements engineering (MBRE)[1], models are built, or borrowed (from previous projects), to assist in early life cycle decision making. Often at requirements time, much is unknown about a project.

			KC-1	
	ranges		now	changes
Scale drivers	prec = 0..5	precedentness	0, 1	
	flex = 0..5	development flexibility	1, 2, 3, 4	1
	resl = 0..5	architectural analysis or risk resolution	0, 1, 2	2
	team = 0..5	team cohesion	1, 2	2
	pmat = 0..5	process maturity	0, 1, 2, 3	3
Product attributes	rely = 0..4	required reliability	4	
	data = 1..4	database size	2	
	cplx = 0..5	product complexity	4, 5	
	ruse = 1..5	level of reuse	1, 2, 3	3
	docu = 0..4	documentation requirements	1, 2, 3	3
Platform attributes	time = 2..5	execution time constraints	?	
	stor = 2..5	main memory storage	2, 3, 4	2
	pvol = 1..4	platform volatility	1	
Personnel attributes	acap = 0..4	analyst capability	1, 2	2
	pcap = 0..4	programmer capability	2	
	pcon = 0..4	programmer continuity	1, 2	2
	aexp = 0..4	analyst experience	1, 2	
	pexp = 0..4	platform experience	2	
Project attributes	ltex = 0..4	experience with language and tools	1, 2, 3	3
	tool = 0..4	use of software tools	1, 2	
	site = 0..5	multi-site development	2	
	sced = 0..4	time before delivery	0, 1, 2	2

Table 3.3: COCOMO-II parameters. Scale drivers are listed first. The cost drivers are union of the product, platform, personnel, and project attributes. Last two columns show values known within one NASA software project.

[1] This case study is published on the First International Workshop on Model-based Requirements Engineering. `http://www.bfsng.com/mbre01/`

Table 3.3 shows a NASA software project KC-1 scored on the 22 parameters of the COCOMO-II software cost estimation model [ACDC$^+$98]. The Madachy extension of COCOMO-II model allows one to estimate the cost, effort and schedule when planning a new software development activity. Based on parameter values, the model generates *STAFF* and *WORRIES* values. *STAFF* is the number of staff required to get person months of work done in the recommended number of months: $STAFF = \frac{person\ months}{monthes}$. *WORRIES* is a numeric effort-risk index representing how concerned an experienced analyst might be about a particular software project. The column labelled *now* in table 3.3 shows the current applicable parameter settings of the KC-1 project. The analysts interviewed for this case study are uncertain about some aspects of this project in requirement time. Where somewhat uncertain, they used ranges; e.g. it was unclear if developers had ever seen this kind of application before so $prec = \{0, 1\}$. When totally uncertain, they just used a question mark; e.g. no knowledge about execution time constraints was available so $time = ? = \{2, 3, 4, 5\}$ where $\{2, 3, 4, 5\}$ is the complete range of possible values for *time*. The column labelled *changes* in table 3.3 shows eleven proposed changes to the current situation.

3.3.2 Simulation

We ran the model repeatedly with random selected parameter values from their possible ranges shown in table 3.3's *now* column. The model also takes another argument SLOC (Source Lines Of Code) as input. Since some uncertainty also existed in the size estimates, the SLOC was taken to be 75K, 100K, 125K. The model was run 30,000 times (10,000 times for each SLOC

Staff	Worries 0	7	14	21	28	35	Totals
200							
180							
160							
140							
120				1	1		2
100			1	4	1		6
80		1	9	5	1		16
60		14	15	3			32
40	5	29	5				39
20	2	3					5
Totals	7	56	30	13	3		100

Figure 3.4: Simulation outputs using inputs specified in KC-1 project.

Staff	Worries 0	7	14	21	28	35	Totals
200							
180							
160							
140							
120							
100							
80							
60		23					23
40	11	45					56
20	14	7					21
Totals	25	75					100

Figure 3.5: Re-simulation results after constraining the model using the treatment.

setting). Model computes and outputs the *STAFF* and *WORRIES* values to assess each combination of parameter settings. Figure 3.4 shows the results as a *percentile matrix*; i.e. it shows what percentage of the 30,000 runs falls into a particular range. The percentiles matrix is color-coded: the darker the cell, the larger the percentage of the runs falling in that cell. There is a large variance in the simulation results.

3.3.3 Results and Validation

TAR2 was used to identify the critical changes capable of reducing both *STAFF* level and *WORRIES* index about the project. We firstly configured the configuration file so that TAR2 only explored the proposed changes listed in table 3.3's "changes" column. After running on the simulation dataset, it gave a best treatment:

$$acap = 2 \wedge sced = 2 \wedge pmat = 1$$

- acap=2: using analysts with a middle-range of ability (fall between the 45th to 65th percentile);
- sced=2: ensuring that the project was at least in the upper-half of CMM1, but don't go to CMM process level 2.
- pmat=1: increasing the time to delivery to 100% of the time proposed by the project- i.e. no pressure for an early delivery;

To validate the treatment, we used re-simulation. Re-simulation is a validation method, by which the experimental results are feed back into the model. Compared to cross validation, it is robust in that the result is assessed through a outside device, eliminating the effect of training data (e.g., small data size or noisy data examples). In this case, the input values for *pmat,acap,sced* were set as above, and the rest of the inputs were left randomized as before. After generated another 30k data, the constrained simulation results are shown in figure 3.5. Compared to figure 3.4, it is evident that the proposed treatment has greatly *reduced* the variance in the model's behavior and *improved* the mean values (decreased both the *STAFF* level and *WORRIES* index). The three items found in the treatment are proved to be the critical changes that could benefit the KC-1 project among all the possible propositions.

3.4 Case Study 2: Requirement Optimization

This example illustrates the application of treatment learning on requirement optimization via an iterative learning cycle.

Planning for the optimal attainment of requirements is an important early life cycle activity. Such planning is difficult when dealing with competing requirements, limited resources, and the incompleteness of information available at requirements time. The pilot study discussed here is an evaluation of a promising piece of research-quality spacecraft technology. The purpose of the evaluation is to identify the risks that would arise in maturing this technology to flight readiness, and what mitigation could be identified to address those risks in a cost-effective manner.

3.4.1 The Requirement Interaction Model

For the pilot study, NASA experts built a real-world model developed in the Defect Detection and Prevention (DDP) framework [CFH01]. The model is a network connecting 32 requirements, 69 risks and 99 mitigations. Risks are quantitatively related to requirements, to indicate how much each risk, should it occur, impacts each requirements. Mitigations are quantitatively related to risks, to indicate how effectively each mitigation, should it be applied, reduces each risk. A set of mitigations achieves benefits, but incurs costs. The main purpose of the model is to facilitate the judicious selection of a set of mitigations, attaining requirements in a cost-effective manner. This kind of requirements analysis seeks to maximize benefits (i.e., our coverage of the requirements) while minimizing the costs of the risk mitigation actions. Optimizing in this manner is complicated by the interactions inside the model - a requirement may be impacted by multiple risks, a risk may impact multiple requirements, an action may mitigate multiple risks, and a risk may be mitigated by multiple actions.

3.4.2 The Iterative Learning Cycle

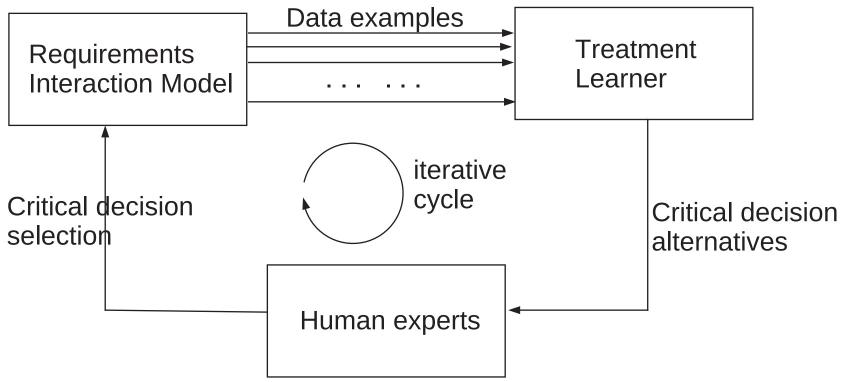

Requirements Interaction Model
Data examples
. . . . . .
Treatment Learner
iterative cycle
Critical decision selection
Critical decision alternatives
Human experts

Figure 3.6: The iterative cycle of Simulation/Summarization/Decision.

Our approach is to follow the iterative cycle of simulation, summarization and decision shown in figure 3.6. The requirements interaction model is used to grow dataset representing the space of options, treatment learner summarizes the data and gives critical decision alternatives (e.g., the control variables and their corresponding settings), the domain experts review the alternatives and make final decisions. This way, experts make more effective use of their skill and knowledge by focusing their attention on the relatively small number of most critical decision alternatives. Repeating this cycle leads the iterative approach to the optimal (or near optimal) decision within the options space.

Baseline Simulation

The model was initially executed by selecting risk mitigations at random. This generated 30,000 instances of combinations from the 99 risk mitigations actions. Each instance of the combinations was evaluated by the numerical cost and benefit values automatically computed based on domain data. The study needs to identify the optimal solutions that attain high benefit(approximately 250) while remaining a relative low cost limit(around \$600,000). The option space is huge: $2^{99} \approx 10^{30}$ sets of decisions are to be explored. Figure 3.7 shows the initial output of the cost-benefit distribution from the model. The wide spread dots indicate a large variance in the possible cost and benefit ranges.

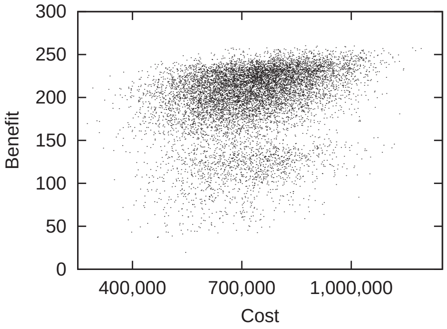


Figure 3.7: Initial result from executing the model of pilot domain.

Combining Cost and Benefit Values

TAR2 takes dataset containing one single discrete class attribute. We must combine the cost and benefit values into a single score before applying TAR2 on the simulation data. This domain-specific process was proceeded as follows:

- Partitioning cost value into 4 regions: below \$600,000 (most desirable region); \$600,000 to \$649,999; \$650,000 to \$699,999; at or above \$700,000 (least desirable region).

score	$< \$600K$	$[\$600K, \$650K)$	$[\$650K, \$700K)$	$> \$700K$
high 25%	16	14	11	7
mid high 25%	15	12	8	4
mid low 25%	13	9	5	2
low 25%	10	6	3	1

Table 3.4: Balanced score combination of cost and benefit values

- Partitioning benefit value by subdividing it into quartiles, i.e., putting the lowest 25% of the benefit figures into the lowest benefit range, the next 25% into the next, etc.
- Ranking the 16 possible pairings of cost and benefit according to a balanced scheme which yielded a combined score of "goodness". The scheme is shown in table 3.4.2.

Learning Iterations

We used TAR2 as a knowledge acquisition tool to summarize the simulation dataset. After ran it on the examples, a set of treatments was discovered and the best was selected by the domain experts. We then imposed the treatments on the model, i.e., some mitigations were to be performed and some were not; others were kept random. Simulating the constrained model again gave us another example set. The whole process was repeated, each run of TAR2 resulted in a new set of constraints, which were then imposed on the model before the next simulation. After five iterations, TAR2 found 30 out of 99 decisions (6 per run. 6 was the maximum size for which it successfully terminated) that significantly effected the cost/benefit distribution. Figure 3.8 shows the model output following the 5^{th} iteration. Compared to figure 3.7, the variation among the cost-benefit figures is relatively small. Since the model represents human experts' estimates, the computed cost-benefit figures should not be misinterpreted to have high precision. At the point where the figures are so tightly clustered, it is appropriate to stop.

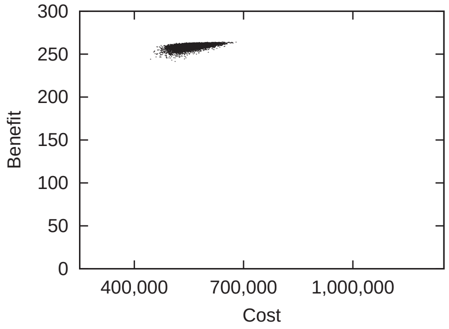


Figure 3.8: Result from executing the model of pilot domain when it was constrained by treatments after the 5^{th} iteration.

The entire series is shown in 3.9. The first percentile matrix (called **round 0**) summarizes figure 3.7. The **round 4** corresponds to the dot plotting in figure 3.8, in which a compact set of points concentrated at the upper end of the benefit range (around 250), and at a cost of approximately $6000. From **round 0** to **round 4**, the variance was reduced and the mean values improved.

3.4.3 Compared to Simulated Annealing

Parallel to treatment learning, a simulated annealing algorithm (SA) was also applied to the same requirement analysis task [FM02b]. Simulated Annealing is a commonly used search algorithm for optimization problem. It combines random selection and hill climbing to find global maxima. In particular, it does a random walk, choosing neighbors at random and deciding

round 0:

	Cost				
Benefit	400K	600K	800K	1,000K	Totals
250		6	15	5	26
200	1	22	27	4	54
150	1	6	5	1	13
100		3	3		6
50		1%			1
Totals	2	38	50	10	100

round 1:

	Cost				
Benefit	400K	600K	800K	1,000K	Totals
250	7	45	13		65
200	12	22	1		35
150					
100					
50					
Totals	19	67	14		100

round 2:

	Cost				
Benefit	400K	600K	800K	1,000K	Totals
250	9	8	7		24
200	18	58			76
150					
100					
50					
Totals	27	66	7		101

round 3:

	Cost				
Benefit	400K	600K	800K	1,000K	Totals
250	9	70	11		90
200	3	7			10
150					
100					
50					
Totals	12	77	11		100

round 4:

	Cost				
Benefit	400K	600K	800K	1,000K	Totals
250	1	81	17		99
200		1			1
150					
100					
50					
Totals	1	82	17		100

Figure 3.9: Percentile matrices showing four rounds of treatment learning for the pilot study.

at random whether to visit that neighbor. The randomness is a function of a "temperature" variable. When $T = \infty$, it chooses neighbors at random; in the limit as T approaches zero, it chooses only neighbors that improve the value. If the temperature is reduced slowly enough, this guarantees to find the global optimal result.

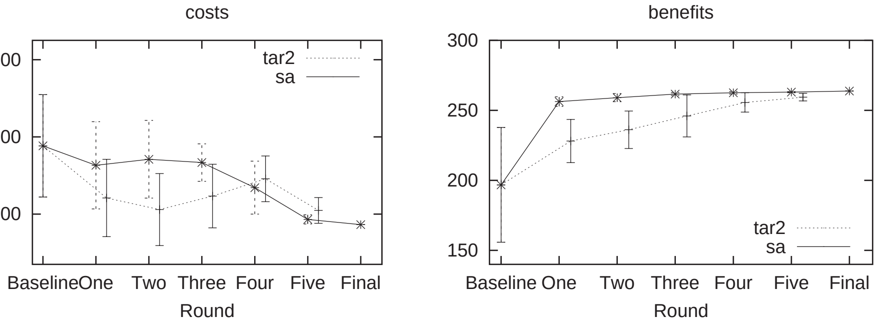


Figure 3.10: Comparison of TAR2 and simulated annealing.

Figure 3.10 compares TAR2 and simulated annealing. At each round X (shown on the x-axis), simulated annealing or TAR2 was used to extract key decisions from a log of runs of the model. A new log is generated, with the inputs constrained to the key decisions found between round zero and round X. Further rounds of learning continue until the observed changes on costs and benefits stabilizes. The comparisons show that:

- As seen in Figure 3.10, simulated annealing and TAR2 terminate in (nearly) the same cost-benefit zone.
- Simulated annealing did so using only 40% of the data needed by TAR2;
- However, while TAR2 proposed constraints on 33% of the mitigations, each SA solution specifies whether a mitigation should be taken or not

for all 99 mitigations. Hence there was no apparent way to ascertain which of them are the most critical decisions. This loses the main advantage of TAR2; i.e. no drastic reduction in the space of options.

3.4.4 Discussion

The iterative treatment learning on the pilot study has successfully arrived at a near-optimal attainment of requirements. By identifying only one-third of the mitigations (30 out of 99), we are able to significantly narrow the widely spread cost/benefit distribution.

This case study also demonstrated an incremental use of TAR2: At each iteration, users are presented with list of treatments that have most impact on a system. They select some of theses and the results are added to a growing set of constraints for a model simulator. This approach has two advantages: Firstly, it narrows down the solutions one step at a time, giving a clear statement on which attributes are *most* important; Secondly, the domain experts found this approach user-friendly, since it provided the opportunities for them to inject their knowledge into the process, and allowed them to focus on only a small number of the most critical alternatives.

3.5 Relation To Other Techniques

3.5.1 Extension to Standard Machine Learning

Treatment learning closely relates to both classification and association rule mining, yet significantly differs from them. Classification analyzes the data in order to construct one or a set of models, and attempts to predict the class membership of new data examples. Standard classifier algorithms such as C4.5 [Qui92] or CART [BFOS84] treat each class equally. Treatment learning, however, mines descriptions that distinguish a target class from its contrasting classes. It has a notion of *class weighting*. Such learners can filter their learnt theories to emphasize the location of the *good* classes or *bad* classes.

Some association rule learners such as MINWAL [CW98], explore *weighted learning* in which some attributes are given a higher priority weighting that others. This is a generalization of the association rule mining problem. In this case, the APRIORI property of the support measure no longer exists and can not be applied. [CW98] et.al., proposed two algorithms based on the support bounds. Such weighted learning can focus the learning onto issues that are of particular interest to some audience. Unlike association rule mining where data contains no pre-defined classes, treatment learning deals with multiple classes. The weights are associated with each class to represent the level of user preference of that particular class. One approach to directly use association rule mining algorithm to find treatments would be to mine frequent itemset for each class separately and then combine them in a post analysis. However, this is a poor idea as it won't push confidence into the search process thus lose the prune opportunity. Further, the resulting rules are difficult to interpret because the rule learner does not enforce consistent contrast [DB96] i.e., using the same attributes to separate classes.

Another difference between treatment learning and association rule mining lies in the prune strategies they employ. Most association rule learners use support based pruning to mine frequent itemsets first, and then use confidence to construct association rules. Instead, treatment learner uses confidence based pruning to shrink the search space. One problem with support based pruning is that most rules with high support are obvious and well-known. It is the rules of low-support that provide interesting new insights. The task of mining association rules without support requirement was recently considered in [KW01] and [EC00]. With only the confidence requirement available, [KW01] exploited a certain monotonicity of confidence, called the universal-existential upward closure. This property yields a level-wise candidate generation with a confidence-based pruning and was implemented in a disk-based environment. Different from them, Cohen et.al. [EC00] developed a family of algorithms for solving this problem, employing a combination of random sampling and hashing techniques. However, a major restriction in their work is that they only deal with pairs of columns. It is not clear whether their techniques could be extended to the identification of more complex rules.

3.5.2 Relation to Change Detecting Algorithms

Concurrent with our work, Bay and Pazzarni propose the concept of contrast set [BP99]. Our work's variant is to combine contrast sets with weighted classes with minimality. That is, treatments can be viewed as the smallest possible contrast sets that distinguish highly weighted classes from lowly weighted classes. Further, the $confidence1$ heuristic aims at maximizing:

$$|P(y = c_1|X) - P(y = c_2|X)| \text{ (y is the class label, X is a itemset)}$$

while contrast sets aims at maximizing:

$$|P(X|y = c_1) - P(X|y = c_2)|$$

Note that the two equations can be relate with Bayes Rule:

$$P(X|y) = \frac{P(y|X) * P(X)}{P(y)}$$

Thus we can always convert from one to the other. Although the forms can be made equivalent, the difference is that the X that optimizes/maximizes one equation is not necessarily the same as the X that is best for another.

Another promising change detecting method is the mining of *emerging patterns*(EP), introduced by Dong and Li [DL99]. EPs are associated with two datasets and are used to describe significant differences or trends between the two datasets. EPs have been used to construct powerful classifiers, which are more accurate than C4.5 and CBA [BLM] for many datasets [LDR00]. But there are several drawbacks with the EP approach. Firstly, their algorithm must mine the data multiple times for different base supports. Secondly, it is not clear if the method can be extended to handle more than two classes. Thirdly, there is a problem of displaying the large volume of results. For example, on the Mushroom data set they found 299811 borders, each representing about 2^{18} sets. This is far too many results to show to an end user. In fact, the result itself might be a source for further data mining in order to provide understandable knowledge.

3.6 Conclusion

We have pointed out the repeated observation of the narrow funnel effect. Although there is no conclusive proof of its existence, empirical evidences suggest that narrow funnels are common. In domains containing narrow funnels, a small number of variables are enough to control the others in the option space.

We propose treatment learning as both a test and an application of the narrow funnel effect. Treatment learning is a machine learning method for finding items associated with desired classes. It uses $confidence1$ measure to evaluate merit of individual items and output treatments that capture difference between classes.

We conducted two case studies to explore this approach. In model-based domains, our uncertainty about the domain or over the parts of the model usually results in a wide spread of output possibility. However, when models contain narrow funnels, there exist key decisions which can condense the possibility. In both studies, treatment learner has successfully identified funnel variables that reduced the variance and improved the mean of values within the output distribution.

Chapter 4

Algorithmic Evaluation and Improvement

This chapter examines the algorithmic performance of TAR2 and present an improved learner TAR3. Before introducing TAR3, we first offer some baseline measurements on TAR2. In summary, while TAR2 is practical for many datasets, there exists situations where its runtimes can grow exponentially. TAR3 fixes this problem. On the datasets where TAR2 is exponential, TAR3 runs in linear time.

4.1 Algorithm Performance of TAR2

We have experimented TAR2 on many domains, some of which come from the UCI machine learning data repository [CEC98], others come from real world application domains. Table 4.1 reports TAR2 runtimes(sec) on 11 data sets of different sizes. Experiments are conducted on a 333 MHz Windows machine with 200MB of ram. It shows TAR2 is suitable for handling small to medium sized data set. For example, the algorithm learnt treatments in 23 seconds from a dataset containing 250,000 examples: see the *reachness2* domain in table 4.1.

domain	#example	#continous	#discrete	#class	size(T)	time(sec)
iris	150	4	0	3	1	< 1
wine	178	13	0	3	2	< 1
car	1,728	0	6	4	2	< 1
autompg	398	6	1	4	2	1
housing	506	13	0	4	2	1
pageblocks	5,473	10	0	5	2	2
circuit	35,228	0	18	10	4	4
cocomo	30,000	0	23	4	1	2
pilot	30,000	0	99	9	5	86
reacheness	25,000	4	9	4	2	3
reacheness2	250,000	4	9	4	1	23

Table 4.1: Runtimes for TAR2 on different domains. First 6 data sets come from the UC Irvine machine learning data repository; “cocomo” comes from the COCOMO software cost estimation model [MH01c]; “pilot” comes from the NASA Jet Propulsion Laboratory [FM02a]; “reachness” and “reachness2” come from other source [MH02].

4.1.1 Runtime vs. Data Size

To examine the runtime with respect to data size, we generated data set of different sizes from the COCOMO risk estimation model [ACDC$^+$98]. By simulating the model, we generated data set of different size.

Figure 4.1 shows TAR2’s runtimes with respect to dataset size measured in megabytes. Three curves correspond to three different treatment size setting, treatment size equals 1,2,3 respectively. It can be seen that for a fixed treatment size, runtimes are linear on dataset size. However, bigger treatment size results in bigger slope, indicating a decrease in efficiency.

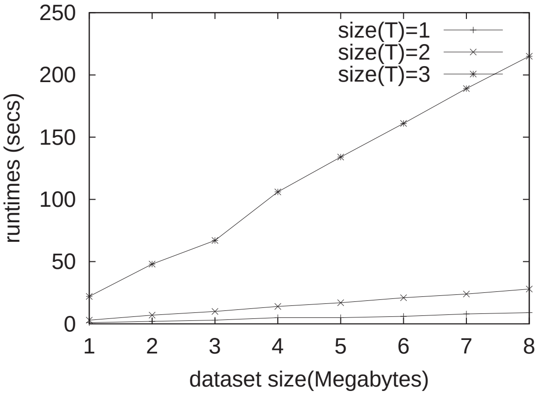


Figure 4.1: Runtime vs dataset size. Datasets are generated from COCOMO risk estimation model [ACDC^{+}98].

4.1.2 Runtime vs. Treatment Size

Figure 4.2 reports one study where the size of the data set was held constant(3MB), and the size of treatment(size(T)) was increased. The result is a line on the logarithm Y-axis coordinate, showing that TAR2's runtimes are exponential on treatment size.

Recall that treatments are generated by exploring subsets of pairs whose $confidence1$ value are greater than a threshold. For treatment size=r, if N pairs occurring above the threshold, TAR2 will explore $\binom{N}{r}$ such pairs. To find treatments of different sizes, there are total

$$\binom{N}{1} + \binom{N}{2} + \binom{N}{3} \ldots + \binom{N}{N-1} + \binom{N}{N} \approx 2^N$$

candidates to explore. We have used the *value exclusion* property of dataset to preliminarily shrink the search space. *value exclusion* ensures that items of the same attribute e.g. A1=a and A1=b can never be contained by the same instance. Hence, it is unnecessary to produce candidates with more

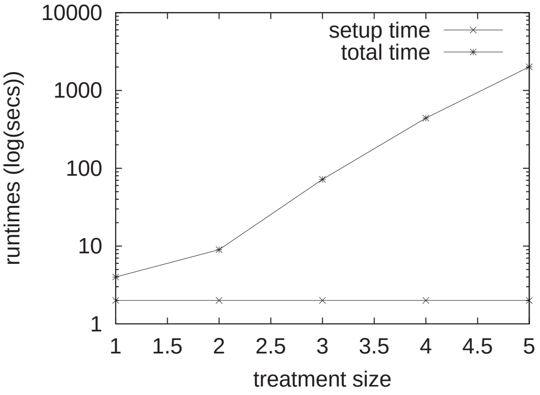


Figure 4.2: Runtime vs treatment size. Data set size is fixed to 3MB. Datasets are generated from COCOMO model. Note the Y-axis is the logarithm of the runtime.

than one value for the same attribute. Even so, the search is still intractable when N is large and will incur exponential runtime.

4.1.3 Runtime in Practice

The exponential impact of increasing treatment size demands small treatments in application. We argue that this is not necessarily a reason to reject TAR2. Firstly, if very large treatments are required, an iterative learning approach, such as described in the "pilot" case study in chapter 3, may suffice.

Secondly, one of the goals of treatment learning is to identify funnel variables. For domains containing narrow funnels, large treatments are not necessary. Unsatisfactory output from TAR2 could be a good indication that narrow funnels do not exist. In that case, more elaborate learning approach must be applied. Among the domains we have explored using TAR2, narrow funnels appear to be very common. Those domains exhibit the following

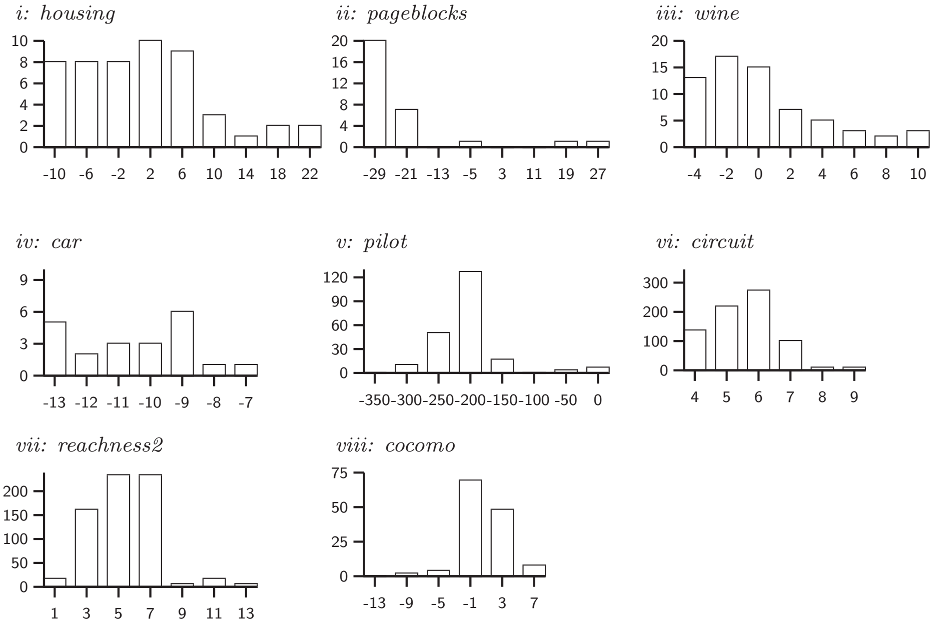


Figure 4.3: Confidence1 distributions seen in eight domains. Y-axis is the number of times a particular confidence1 was seen. (i)-(iv) come from datasets taken from the UC Irvine machine learning repository. (i)-(iv) were generated from other domains discussed in this thesis.

property: a small number of variables exert large influence on the overall behavior of the system. Figure 4.3 shows the confidence1 distributions seen in eight example datasets. We could observe a small right tail in all the $confidence1$ distributions. As a result, TAR2 was able to find effective treatments with $treatmentSize < 6$.

4.2 TAR3: The Improvement

Algorithmic analysis of TAR2 has shown that although it works well for many datasets, there exists situations where the runtime can grow exponentially. TAR3 is our solution to the problem. In summary, TAR3 uses a novel random sample method of the confidence distribution to find treatments. One drawback with such random sampling methods is that the resulting conclusions are unstable or fail to explore all the interesting parts of a theory. We show below that, for TAR3, this is not the case. In fact, despite the use of random sampling, TAR3 usually generates the same solutions as the non-random search of TAR2.

4.2.1 Random Sampling

In TAR2, $confidence1$ measure ($\triangle$) is a heuristic assessing the contribution each attribute-value pair makes toward changing the class distribution. If we think of $confidence1$ values as weights associated with attribute-value pairs, those with larger weights have higher probability to be selected into treatments than those with smaller weights. Let:

$$\begin{aligned} a_i &= \text{attribute-value pair in the data set} \\ \triangle_i &= \text{confidence1 value associated with } a_i \end{aligned}$$

What we need is to sample a_i from a multinomial distribution with discrete weights $\triangle_i$. This strategy is employed in TAR3 and is done as follows:

1. Place a_i in increasing order according to $\triangle_i$.
2. Compute the CDF(Cumulative Distribution Function) value of $\triangle_i$:

$$CDF(i) = \frac{\sum_{x=1}^{i} \triangle_x}{\sum_{x'=1}^{N} \triangle_{x'}} \quad \text{(N = the total number of } \triangle_i\text{)}$$

3. Sample a uniform value u in the interval [0,1].
4. The sample is the least a_i such that $u < CDF(i)$

The above process is repeated until we get a treatment Rx of a given size. As a side effect, random sampling also eliminates the necessity to specify the confidence1 threshold.

4.2.2 Treatment size

TAR2 requires treatment size be specified by the user as an input parameter. Each run of TAR2 returns treatments of that fixed size. In TAR3, treatment size is a uniform value sampled from the interval [1..*maxTreatmentSize*], where *maxTreatmentSize* is the maximum treatment size user interested. The upper bound of *maxTreatmentSize* is the total number of attributes in the data set. This allows us to obtain treatments of different sizes in one run. In practice, we found that *maxTreatmentSize* seldom exceeds half the number of attributes – a strong empirical evidence for narrow funnel effects.

4.2.3 $lift(Rx)$ evaluation

TAR2 evaluates a treatment candidate *Rx* by calculating $lift(Rx)$. Treatment *Rx* is qualified only if $lift(Rx) >$ certain threshold. We eventually abandoned this approach. Instead, treatment *Rx* is reported if and only if it is among the top N treatments found whose $lift(Rx) > 1$ ($lift(Rx) > 1$ ensures the treatment indeed makes improvement on class distribution), where N is the maximum number of treatments user wants.

4.2.4 $lift(Rx)$ penalization

For treatment *Rx*, *size*(*Rx*) has noticeable impact on *worth* of the treated data set (and hence $lift(Rx)$). Usually treatments of larger sizes tend to achieve higher $lift$ than those of smaller sizes. In the association rule mining community, Ke Wang et.al. [WZH00] found that the $confidence$ of a rule holds a property known as the *universal-existential upward closure.* To illustrate the property, consider the following association rules:

R1: Age.young → Salary.low
R2: Age.young, Gender.M → Salary.low
R3: Age.young, Gender.F → Salary.low

Suppose that R1 has confidence of 0.6, that is, 60% young people have low salary. Since the condition Gender.M and Gender.F are exhaustive and mutually exclusive, if one condition impacts confidence negatively, the other must impact confidence positively. Consequently, at least one of R2 or R3 has at least as much confidence as R1. Also, if none of R2 and R3 is confident, then R1 must also be non-confident. This property indicates that long rules tend to have high confidence than short rules.

In treatment learning, the evaluation criterion $lift(Rx)$ is a function of the treatment's confidence with respect to classes. It exhibits a similar property. Consider one extreme example where a dataset has total 5 attributes. A treatment of size 5 might select only one example and this example belongs to the best class. This treatment has the highest $lift(Rx)$: on the class distribution graph, 100% of examples(in this case, only 1 example) in the treated subset belong to the best class. In other words, treatments of very large size tend to have high $lift$ values. However, they usually select so few examples that lack statistical significance.

This problem is solved by introducing in the Best Class Support $sup(Rx)$ parameter. $sup(Rx)$ is defined as **ratio of examples belonging to the**

best class in the treated set to those in the original set. i.e.,

$$sup(Rx) = \frac{|C_{best}, Rx|}{|C_{best}|} = \frac{P(C_{best}|Rx)}{P(C_{best})}$$

$minSup$ specifies the minimum $sup(Rx)$ ratio a treatment Rx must achieve. For instance, assuming a data set contains 500 examples, among which only 50 belong to the best class. With $minSup = 80\%$, if applying a treatment Rx results in a treated set containing 100 examples, among which 45 belong to the best class. Then we have $sup(Rx) = \frac{45}{50} = 90\% > minSup$, thus Rx is considered qualified. In fact, Rx is a very good treatment, as it raises the best class percentage from $\frac{50}{500} = 10\%$ to $\frac{45}{100} = 45\%$.

In TAR3 implementation, we didn't simply reject treatments that do not meet the $minSup$. Instead, we use the $minSup$ as a regularizer that penalizes $lift(Rx)$:

$$lift(Rx) = \frac{worth(D \wedge Rx)}{worth(D)} * penalty$$

$$penalty = \begin{cases} 1 & (sup(Rx) \geq minSup) \\ \frac{sup(Rx)}{minSup} & (sup(Rx) < minSup) \end{cases}$$

Penalization ensures some potentially highly predictive treatments still be reported even though their $sup(Rx)$ are slightly below the threshold.

4.2.5 Stopping point

A theoretical drawback with any random search is that such random exploration can miss significant parts of the option space. TAR3 addresses this issue by taking the following strategy: To generate N treatments, TAR3 makes multiple iterations. Each iteration, X new treatments are generated and checked. Only qualified top N are remained in the treatment set. If the current iteration doesn't contribute any new treatments to the top N set, it is called a failure iteration. The next iteration, more treatments (X+N) are generated. The procedure stops after M failure runs in a row. In practice we found that M=[5..10] was often sufficient to return stable results.

4.2.6 Usability

TAR3 was originally designed as an experiment in reducing certain exponential time processing within TAR2. A happy side-effect is that TAR3 is actually more user-friendly than TAR2. TAR2 requires the manipulation of certain arcane parameters that must be fiddled with many times. On the other hand, TAR3's parameter set is much more succinct and, often, need not be modified from run to run.

Table 4.2 lists parameters required by TAR2 and TAR3 excluding those common to both. The replacement of threshold parameters with upper-bound ones makes parameter setting easier and more intuitive. While thresholds are domain specific, upper-bounds are less sensitive. We normally use default values or set them to some larger values, the controlling strategy employed in TAR3's random process(e.g., the stopping point) was able to accommodate domains accordingly. Further, we no longer have to run it several times to get treatments of different sizes.

TAR2	TAR3
1. $\triangle$ threshold	1. maxTreatmentNumber
2. worth threshold	2. maxRandomIterations
3. treatmentSize	3. maxTreatmentSize

Table 4.2: Different parameters required by TAR2 and TAR3.

4.3 Performance Improvement

4.3.1 Runtime In Different Domains

domain	#example	#continous	#discrete	#class	maxSize(Rx)	TAR2(sec)	TAR3(sec)
iris	150	4	0	3	2	1	< 1
wine	178	13	0	3	2	1	< 1
car	1,728	0	6	4	4	3	< 1
autompg	398	6	1	4	4	3	< 1
housing	506	13	0	4	4	4	1
pageblocks	5,473	10	0	4	2	2	1
circuit	35,228	0	18	10	6	18	6
cocomo	30,000	0	23	4	3	104	28
pilot	30,000	0	99	9	7	2842	195
reacheness	25,000	4	9	3	4	28	4
reacheness2	250,000	4	9	4	4	293	42

Table 4.3: Runtimes for TAR3 on different domains (on a 333 MHz Windows machine with 200MB of ram).

Table 4.3 compares TAR2 and TAR3 runtimes on the same datasets seen in table 4.1. Column 6 lists the maximum treatment size returned by TAR3 on each domain. Column 7 lists TAR2's runtimes with respect to maximum treatment size. In domains where $size(Rx)$ is small (e.g., the first 6 datasets), TAR2's runtimes are comparable to TAR3's. However, when $size(Rx)$ is large(e.g. the pilot domain), TAR3 is significantly faster than TAR2 (195sec vs 2842sec). The comparison shows that TAR3's random sampling is able to scale when TAR2's exponential search becomes intractable.

4.3.2 Runtime vs. Data Size

Figure 4.4 and figure 4.5 report TAR3 runtimes with respect to the number of attributes and the number of instances respectively. The datasets used

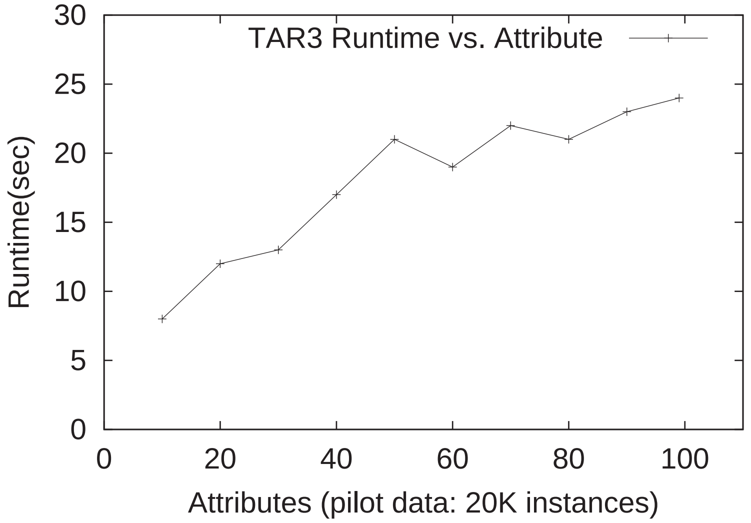


Figure 4.4: Runtime vs attributes. Datasets come from the pilot domain

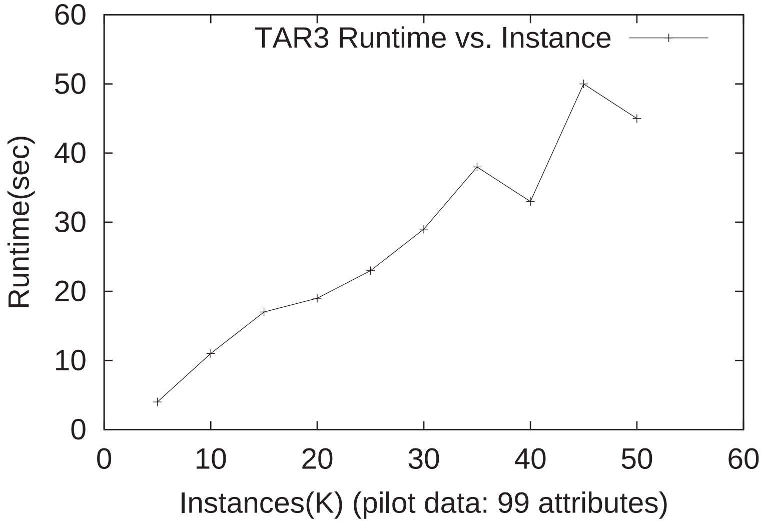


Figure 4.5: Runtime vs instances. Datasets come from the pilot domain

in both experiments come from the pilot domain (discussed in chapter 3 case study 2) because of its high dimensionality. In figure 4.4, we kept the number of instances to 20,000 and randomly chose 10, 20, 30, ... up to all 99 attributes for each trial. Similarly, in figure 4.5, the number of attributes was kept constant (using total 99 attributes) while the number of instances increased from 5,000 to 50,000. Curve in figure 4.4 fits a linear trend line with $r^2 = 0.8836$[1]; curve in figure 4.5 fits a linear trend line with $r^2 = 0.9436$. In summary, TAR3's runtime is linear in the size of training data.

4.3.3 Runtime vs. Treatment Size

Figure 4.6 shows TAR3 runtime with respect to treatment size. The datasets used in this study are the same in the TAR2 runtime evaluation experiments (77250 instances * 23 attributes). Normally, one run of TAR3 returns treatments of different sizes. For this study we forced it to return only treatments of fixed size each run. The runtime curve shown in figure 4.6 is no longer exponential, in fact it fits a logarithmic trend line with $r^2 = 0.9441$. Compared to figure 4.2, TAR2 spent near 1000 seconds for treatments of size 5 while TAR3 only needs less than 100 seconds for treatments of size 8. It is clear that TAR3's algorithm runs much more efficiently.

4.4 Experiment Result Comparison

This study aims at examining TAR3's stability in terms of the returned treatments. We wanted to know whether and to what extent would the randomness introduced in TAR3 effect its results. Our concern was that TAR3's random search would introduce an element of unreliability in the learnt the-

[1] r^2 value is the square of the correlation coefficient

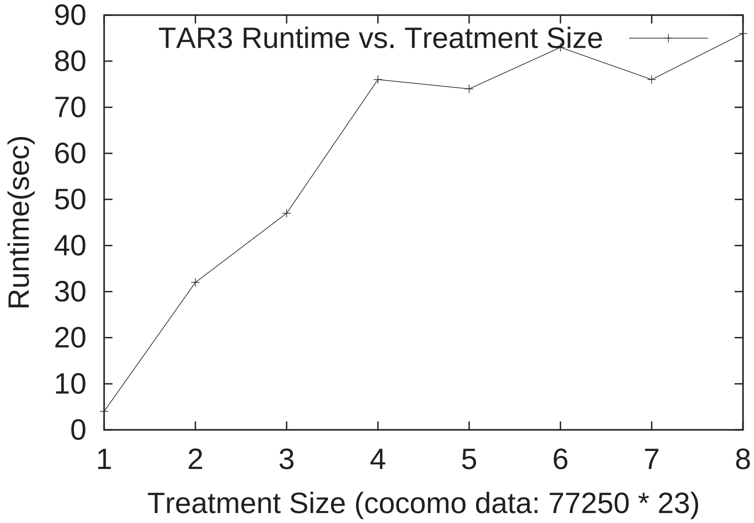


Figure 4.6: Runtime vs instances. Datasets come from the cocomo domain

ories. However, this pre-experimental fear was not realized in practice. In fact, TAR3 generates nearly the same treatments as TAR2.

Firstly, we ran TAR3 on a domain and recorded the size of the best treatment returned. We then configured TAR2 so that it would return treatments of that size on the same domain. We compare both the $lift(Rx)$ and the individual attribute-value pairs appeared in the best treatments returned by TAR2 and TAR3 respectively. Table 4.4 lists the results from 10 domains. In each domain, the best treatment returned by TAR3 has both the same $lift$ value and the same itemsets as returned by TAR2. (The notation [X..Y) means a test of $X <= attribute < Y$). In other words, the best treatment found by the two learners for each domain is identical. Table 4.5 compares results from the more complex "pilot" domain, which contains 30k examples and 99 attributes. TAR3 found a best treatment of size 10, whereas TAR2 can only handle a max size of 6. The larger treatment has higher $lift$ value than the short one, and the two treatments have 5 items in common. In summary, for 10 out of 11 cases, TAR3 found identical treatments as TAR2. For the remaining domain, TAR3 found a better treatment of larger size. Baselined on TAR2, TAR3's output is quite stable. We offer two explanations:

- Treatment learning involves a combination of search and attribute merit evaluation. Replacement of depth-first search with random sampling only changes the search strategy, resulting an improvement in efficiency. The treatment generation is still based on the underlying $confidence1$ distribution. Theoretically, as long as this evaluation process remains untouched, the two learners should return the same results.

- To control the randomness introduced by the CDF sampling method, we have designed the stopping point controlling strategy. The experiments have shown that it is empirically effective for small to medium sized datasets. However, for very large, high dimensional dataset, TAR3 may return similar, yet not identical outcomes.

4.5 Case Study: The Pilot Domain Again

The above studies dealt with small to medium sized datasets. This section describes an experiment with a very large dataset in which TAR3's treatments, while similar, were not identical.

We have discussed the “pilot” case in requirement optimization domain (see chapter 3). To compare both the performance and experimental results,

domain	attribute	range	TAR3	TAR2
iris	petal width	[1..1.6)	x	x
	petal length	[3.5..4.6)	x	x
		lift:	1.71	1.71
wine	attr11	[0.4874)	x	x
	attr12	[1.27..1.78)	x	x
		lift:	1.81	1.81
car	buying	low	x	x
	safety	high	x	x
		lift:	2.21	2.21
auto-mpg	cylinders	[4..5)	x	x
	horsepower	[46..75)	x	x
	weight	[1613..2223)	x	x
		lift:	1.97	1.97
housing	rm	[6.65..9.78)	x	x
	ptration	[12.6..15.9)	x	x
		lift:	2.35	2.35
page-block	blackand	[493..46113]	x	x
	height	[9..804]	x	x
	p_black	[0.052287)	x	x
	eccen	[0.007..2.889)	x	x
	area	[660..143993]	x	x
		lift:	9.28	9.28
circ (30k examples)	B3c	ok	x	x
	Sw2c	off	x	x
	Sw1c	on	x	x
		lift:	9.07	9.07
cocomo (30k examples)	pcap	[3..4]	x	x
	ruse	[1..2)	x	x
	acap	[3..4]	x	x
	sced	[3..4]	x	x
		lift:	1.53	1.53
reachness (25k examples)	orpMean	[6..8)	x	x
	andfMean	0.1	x	x
	noMean	[0..8)	x	x
		lift:	1.65	1.65
reachness2 (250k examples)	orpMean	[9..10]	x	x
	noMean	[0..8)	x	x
	andfMean	0.1	x	x
		lift:	1.65	1.65

Table 4.4: Best treatment returned by TAR3 and TAR2 on various domains.

domain	attribute	range	TAR3	TAR2	TAR3(best)
pilot	P44	Y	x		
(50k examples)	P317	N	x		
	P755	N	x	x	x
	P1066	N	x	x	x
	P706	Y	x	x	x
	P688	Y		x	
	P2160			x	
	P761	N	x	x	x
	P1065	N			x
	P690	Y			x
	P1069	N			x
	P2111	N			x
	P1971	Y			x
	P704	N			x
		lift:	5.93	5.84	8.16

Table 4.5: Best treatment returned by TAR3 and TAR2 on the pilot domain.

we ran TAR2 and TAR3 on the same "pilot" domain again. The model used this time is a revised version of the original one, which contains fewer details. The data set obtained after simulation has 58 attributes instead of 99. Each example is also evaluated by a pair of cost and benefit figures. According to the domain experts, we combined cost and benefit into a single attribute using the same balanced scheme but with different dividing thresholds. The combination resulted in 16 classes representing 16 levels of "goodness".

Figure 4.5 shows the initial cost-benefit distribution from the baseline simulation. The data points are widely spread across the possible cost and benefit ranges. Further, most low cost points correspond to low benefit level and high benefit points have high cost values. The desired low-cost high-benefit points are very few: less than 3% of the entire data.

We followed the same incremental learning approach as discussed in the last chapter, namely the following steps:

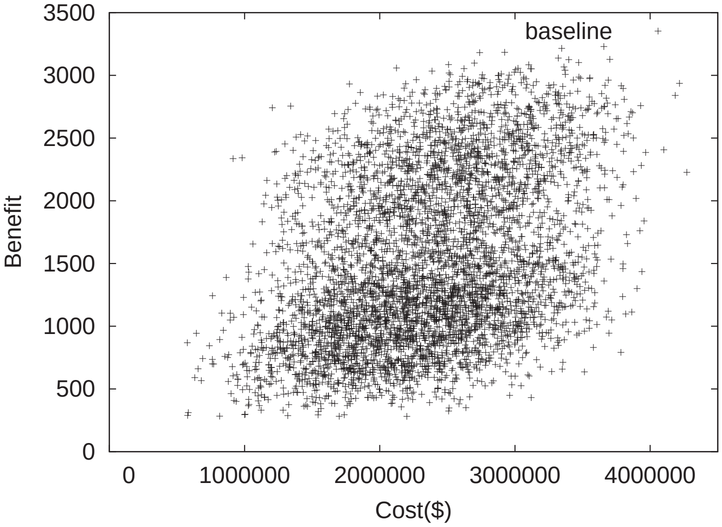


Figure 4.7: The cost-benefit distribution of the initial simulation from the pilot domain.

1. Ran TAR2 and TAR3 on the baseline (initial simulation) data, and generated two set of treatments.
2. The top ranked treatment was chosen from each treatment set. For the purpose of comparison, we didn't ask domain experts to examine the individual treatments, we simply chose the top one.
3. We then imposed the 2 chosen treatments (1 from TAR2, 1 from TAR3) on the model respectively; simulated it again and got another 2 sets of data examples.
4. Step 1-3 were repeated until the resulting distribution was so tightly clustered that domain experts agreed to stop.

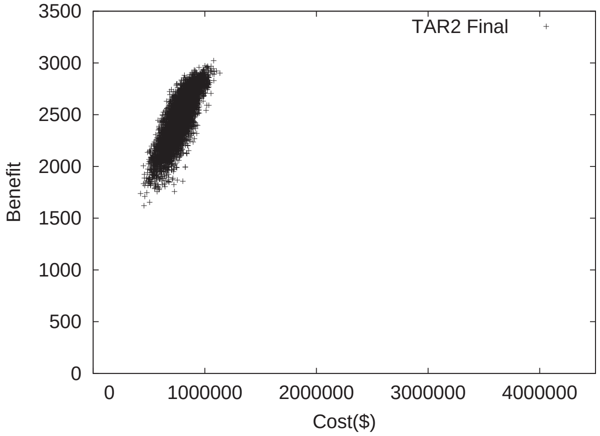


Figure 4.8: The cost-benefit distribution from executing the model of pilot domain when it was constrained after the 5^{th} iteration of TAR2.

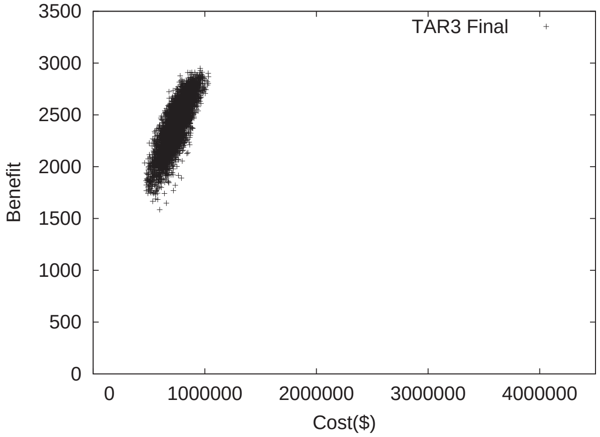


Figure 4.9: The cost-benefit distribution from executing the model of pilot domain when it was constrained after the 4^{th} iteration of TAR3.

4.5.1 Comparison of the Cost-Benefit Distribution

Figure 4.8 shows cost-benefit distribution after the 5^{th} iteration of TAR2. Compared to figure 4.5, the variation is relatively small. Most of the data points are grouped at the upper-left corner of the graph, indicating a tight cluster of low-cost high-benefit results. Figure 4.9 is the result from TAR3 experiments following the 4^{th} iteration. The two graphs are visually the same, indicating very similar results.

4.5.2 Comparison of the Best 3 Class Distribution

For a closer comparison, table 4.6 records the best 3 class distribution of each round. The best 3 out of total 16 classes correspond to a region of desired zone in which domain experts interested. TAR2 reaches the stopping point after 5 rounds, fixing total 19 attributes; TAR3 reaches the stopping point after 4 rounds, fixing total 20 attributes. At the stopping point, both TAR2 and TAR3 achieved a similar class distribution. Further learning didn't offer significant improvement (i.e., further distribution improvement is less than 5%).

TAR2	baseline	run1	run2	run3	run4	run5
size(Rx)	0	4	4	4	4	3
Class14	3%	33%	68%	22%	7%	2%
Class15	0%	1%	7%	38%	19%	5%
Class16	0%	0%	4%	28%	74%	93%
Total	**3%**	**34%**	**79%**	**88%**	**100%**	**100%**

TAR3	baseline	run1	run2	run3	run4	run5
size(T)	0	6	6	5	4	—
Class14	3%	47%	50%	11%	0%	—
Class15	0%	2%	19%	27%	7%	—
Class16	0%	1%	13%	60%	93%	—
Total	**3%**	**50%**	**82%**	**98%**	**100%**	—

Table 4.6: Comparison of the best 3 class distributions for TAR2 and TAR3 experiments.

4.5.3 Comparison of Each Round

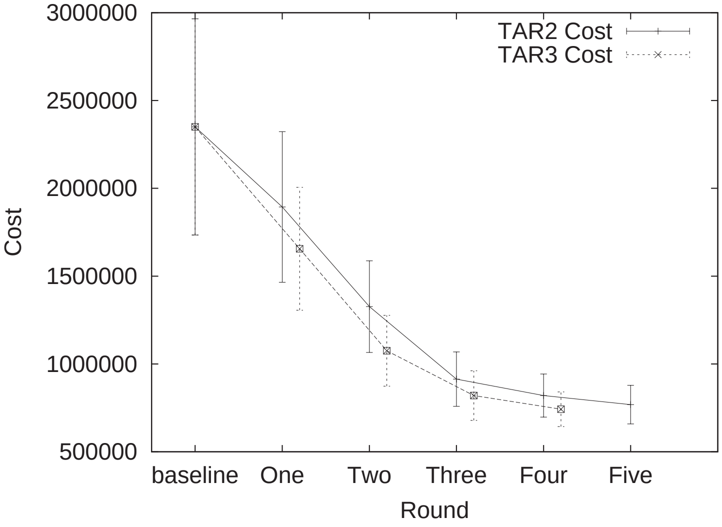


Figure 4.10: The mean and standard deviation of cost at each round.

At the beginning of this experiment, TAR2 and TAR3 started from the same point (i.e. the first baseline data) and came up with different treatments. They later followed their own path toward the final destination. Figure 4.10 compares their performance on each round in terms of the mean and standard deviation of the cost figure. Each round, TAR3 achieved lower mean cost and smaller deviation, allowing it to reach the stopping point one iteration earlier. Figure 4.11 compares the benefit figure. Again, TAR3's deviation is smaller at each round. It is interesting to notice the dip in the TAR2 curve, which indicates a slowing down of the progress. But it eventually catches up in round 4 and round 5.

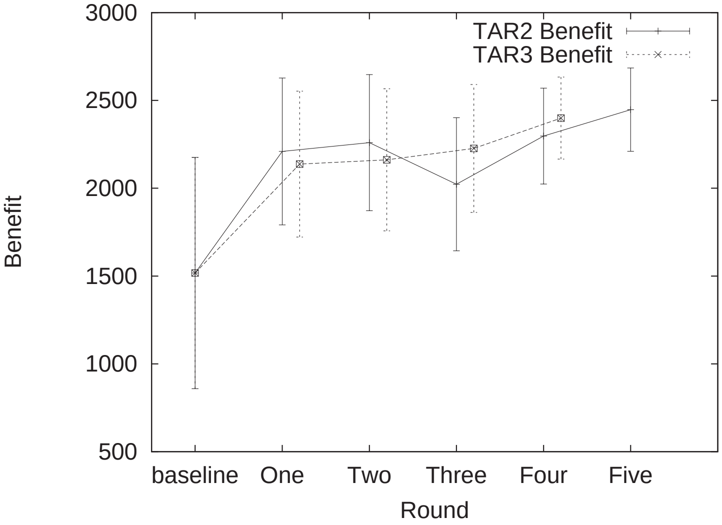


Figure 4.11: The mean and standard deviation of benefit at each round.

No.	Attribute	TAR2	TAR3	No.	Attribute	TAR2	TAR3
1	[P63=N]	✓	✓	12	[P1310=Y]	✓	✓
2	[P70=N]	✓	✓	13	[P529=N]	✓	✓
3	[P72=N]	✓	✓	14	[P544=N]	✓	✓
4	[P73=Y]	✓	✓	15	[P551=N]	✓	✓
5	[P74=N]	✓	✓	16	[P555=Y]	✓	✓
6	[P126=Y]	✓	✓	17	[P575=N]	✓	
7	[P135=N]	✓	✓	18	[P960=N]		✓
8	[P137=N]	✓	✓	19	[P1047=N]	✓	✓
9	[P145=Y]		✓	20	[P1260=Y]	✓	✓
10	[P154=Y]	✓	✓	21	[P1287=N]	✓	✓
11	[P166=N]	✓	✓	Total		19	20

Table 4.7: Comparison of the final treatments found by TAR2 and TAR3, respectively.

4.5.4 Comparison of the Final Treatments

TAR2 gave a final treatment of size 19 after 5 iterations, TAR3's final treatment is of size 20 after 4 iterations. Although in each run, they generated quite different treatments, the combined final treatments are almost the same. Table 4.7 compares the two final sets attribute by attribute, showing that they have 18 items in common.

4.5.5 Comparison of Runtimes

The data size we used is $20,000$ examples × 58 attributes at each round. Table 4.8 compares their runtimes. For reference reasons, column 3 and 5 list the size of best treatment found at that round. The average runtimes of TAR3 is only $\frac{1}{5}$ to $\frac{1}{3}$ TAR2's runtime. That is, TAR3 ran much faster even with larger treatment size.

Round	TAR2(sec)	size(T)	TAR3(sec)	size(T)	TAR3/TAR2
1	1243	4	320	6	25.8%
2	1170	4	348	6	29.7%
3	927	4	235	5	25.3%
4	650	4	126	4	19.4%
5	103	3	—	—	—

Table 4.8: Comparison of the runtimes of each round.

4.5.6 Summary

From the above case study, we have the following observations:

- In this domain, TAR3 achieved a better class distribution than TAR2 each run, and generated a slightly larger treatment.
- Their own path ended up with a similar yet not identical solution, both in terms of the cost-benefit distribution and the treatment produced.
- TAR2 reached the same final distribution after more runs, but with total less attributes fixed (i.e., the size of the final treatment is smaller

in TAR2's case).

- In this domain, TAR3's runtime is much shorter than TAR2, average $\frac{1}{5}$ to $\frac{1}{3}$ TAR2's runtime.

4.6 Conclusion

The algorithmic evaluation on TAR2 pointed out situations where its runtimes can grow exponentially. Our solution to this problem is a better learner TAR3. By adopting random sampling together with other strategies, TAR3 has made major improvement in algorithmic efficiency. Experiments have shown that on the datasets where TAR2 is exponential, TAR3 runs in linear time. We have also conducted extensive comparison to survey the stability of TAR3's treatments. It has been seen that TAR3 usually returns identical treatments as TAR2 on small to medium datasets. On high dimensional dataset, TAR3 followed a faster path to goal. The resulting distribution is better, while the final treatment is slightly different.

Specifically, the key idea to treatment learning is the $confidence1$ evaluation of individual attributes. A different search strategy should not change results but only affect efficiency. The sampling method brings in a certain degree of randomness. Still, we have shown that the controlling method we implemented is effective in practice. Given that $confidence1$ distribution represents the probability an item could be picked up in the treatment, there could be other ways to control the random process: For example, some functions could added to the distribution when computing the CDF value.

Chapter 5

Evaluation Of Treatment Learning Through Feature Subset Selection

5.1 Introduction

In previous chapters we have discussed that treatment learning is closely related to yet significantly different from both classification and association rule mining. This difference makes it not easy to directly compare treatment learner to other learning schemes. For classification, the predictive accuracy is of the main interest to most researchers. Therefore, classifiers are normally evaluated by their classification accuracy on commonly used datasets. Association rule mining are formulated as solutions to the same problem, i.e., to generate all association rules that have support and confidence greater than a user-defined minimum support and minimum confidence respectively. As a result, association rule miners are compared by their algorithmic performance, such as efficiency and scalability. Treatment learning, however, does not have a widely accepted assessment criterion. Treatments are neither

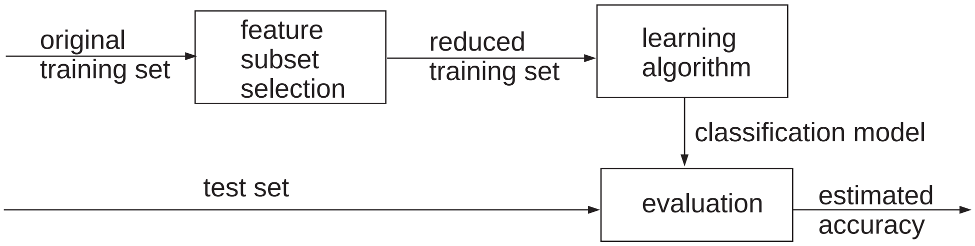


Figure 5.1: Feature subset selection as a pre-process prior to learning.

models that predict the class membership of unseen data examples nor complete set of association rules that satisfy a certain conditions. To provide a benchmark comparison with other learning methods, we approach treatment learning in the framework of feature subset selection (FSS) technique.

In machine learning applications, real world datasets usually include irrelevant, redundant and noisy attributes. To achieve best possible performance, a learning algorithm must select a relevant subset of features upon which to focus its attention. Feature subset selection is the process of identifying and removing as much of the irrelevant and redundant information as possible [HH02]. As a data engineering technique, feature subset selection is generally considered a pre-process prior to learning. Figure 5.1 illustrates the usual application approach. In the procedure, a feature subset selector takes in the original training set and outputs one with reduced dimensionality. The learning algorithm then constructs classification model using the reduced training set. The test set containing all attributes later evaluates the model and gives estimated accuracy.

Feature subset selection benefits learning in the following ways:

1. It can drastically reduce the dimensionality of the data, thus allows the learning algorithms to run faster in a smaller search space.
2. It helps the learner to ignore irrelevant, redundant and noisy features and focus on only relevant, highly predictive ones to improve its performance.
3. It results in more compact, easily understandable representation of the underlying concept.

We believe, the success of feature subset selection improving learning is an application of narrow funnel effect in the field of machine learning. For domains containing narrow funnels, features inside the funnel are much more important than those outside with respect to understanding the domain. Consequently, ignoring features outside the funnel and concentrating on those inside is sufficient, in some cases, beneficial to learning.

As discussed before, treatment learning is in fact a lightweight learning approach dedicated to identifying funnel variables. This characteristic makes it suitable for the feature subset selection task. The following sections describe an experimental evaluation of treatment learning in the framework of feature subset selection. We conducted feature subset selection using treatment learner and compared the result to conclusions seen in a recent state-of-the-art survey of FSS methods (Hall and Holmes, [HH02]). For commonly used machine learning datasets, our approach out-performs the standard FSS methods by selecting the fewest features.

5.2 The Feature Subset Selection Experiment

5.2.1 Feature Subset Selection Methods

Most feature selection techniques involve a combination of search and attribute utility estimation. Some of them use general characteristics of the data to evaluate attributes (referred to as “filters”) while others evaluate attributes by using accuracy estimates provided by the target learning algorithm (referred to as “wrappers” [Koh96]). In either case, they produce attribute ranking which demonstrates the relative merit of individual attribute for the target learning algorithm.

Hall and Holmes [HH02] provided a survey of attribute selection methods. It includes five major developments in attribute selection over the last decade as well as a classical statistical technique for dimensionality reduction. Before comparing our approach to those methods, we briefly describe them here.

Information Gain Attribute Ranking(IG)

Information Gain Attribute Ranking measures the entropy of the dataset before and after observing a feature. The difference in the entropy, called information gain [Qui92], gives a measure of the additional information about the class gained because of that attribute. Detail on this method has been explained in Chapter 2: Decision Tree Induction. This is one of the simplest and fastest method for feature ranking [SJDM98].

Relief(RLF)

Relief is an instance based learning scheme [I.K94]. It first randomly samples one example within the dataset. It then locates the nearest neighbor for that example from the same and the opposite class. The values of the nearest neighbor features are then compared to the sample and the feature scores are maintained and updated based on this. The earliest Relief algorithm could only handle two-class dataset. But it was later extend for multi-class problems by finding nearest neighbors from each different class and weighting their contributions according to each class's prior probability [I.K94]. Relief can also handle noisy data and other data anomalies by averaging the values for K nearest neighbors instead of just one.

Principle Components Analysis(PCA)

Principal component analysis is a statistical technique that reduces the dimensionality of the data by transforming the original feature space. It extracts the eigenvectors of the covariance matrix of the original features [HH02]. The eigenvectors, called principle components, define a linear transformation from the original feature space to a new uncorrelated space. Eigenvectors can be ranked according to the amount of variation in the original data that they account for. The first few transformed attributes are considered to account for most of the variation in the data and are selected. Principal components makes no use of the class attribute. It can only handle numerous attributes. To handle k-valued categorical attribute, one must first convert the attribute to k binary attributes, using "1" to denote the occurrence of the k-th value, and "0" for all other values.

Correlation-based Feature Selection(CFS)

Correlation-based Feature Selection evaluates subsets of features [M.A98]. The technique relies on a heuristic merit calculation that assigns high scores to subsets with features that are highly correlated with the class and poorly correlated with each other. Merit can find the redundant features since they will be highly correlated with the other features. Those features are ignorable for classification as they will be poor predictors of any class. To do this CFS informs a heuristic search for good subset of features via a correlation matrix.

Consistency-based Subset Evaluation(CBS)

Consistency-based Subset Evaluation is in fact a set of methods that use class consistency as an evaluation metric. The specific CBS studied by Hall and Holmes method finds the subset of features whose values divide the data into subsets with high class consistency [HT91] [HR96].

Wrapper Subset Evaluation(WRP)

Kohavi and John [Koh96] wrapped a target learner in the selection procedure to grow subsets of the available features from size 1. At each step in the growth, the target learner was called to estimate the accuracy of the model learned from the current subset. Subset grow was stopped when the addition of new features did not improve the accuracy. In their experiments, average 83% of the features in a domain could be ignored with only a minimal loss of accuracy. Wrapper tailors the search to specific target learners and usually gives better results than filters. Its main problem is overfitting and the large amounts of CPU time required.

5.2.2 The Methodology

Datasets

We used ten datasets included in Hall's experiment, all of which are from the UCI data repository [CEC98]. The datasets have a wide range of categorical and numeric features. Their sizes vary from 148 to 2310 examples. Table 5.1 summarizes these datasets.

DATA SET	INSTANCES	NUMERIC	NOMINAL	CLASSES
anneal	898	6	32	5
breast-c	286	0	9	2
credit-g	1000	7	13	2
diabetes	768	8	0	2
horsecolic	368	7	15	2
ionosphere	351	34	0	2
lymph	148	3	15	4
segment	2310	19	0	7
soybean	683	0	35	19
Vote	435	0	16	2

Table 5.1: Datasets used in the benchmark experiment, all from UCI data repository [CEC98].

Target Learning Algorithms

For each dataset in table 5.1, classification accuracy was averaged over 10-way cross validation before and after attribute selection with respect to a target learning algorithm. Both C4.5 decision tree learner and Naive Bayes classifier were used to test the effectiveness of attribute selection. They are both widely used algorithms representing two fundamentally different approaches to learning. Introduction of them can be found in the "Literature Review" chapter.

We used the WEKA (Waikato Environment for Knowledge Analysis) implementation of C4.5 release 8 (called J4.8) and Naive Bayes. WEKA[1] is a powerful open-source Java-based machine learning workbench that brings together many machine learning algorithms and tools under a common framework with a friendly GUI [WF99].

Using Treatment Learner as Attribute Selector

To accomplish feature subset selection, we followed the following steps:

1. We ran a target learner on the original dataset and obtained the initial classification accuracy by averaging over 10-way cross validation.
2. If a dataset has N classes, TAR2 was run on it N times, each time we change the class ordering in a round robin manner, e.g., each class was given the highest priority in turn. The attribute included in the top treatment was recorded for each run.
3. After N runs, we took the union of the attribute obtained in each run to get the final attribute subset of that dataset.
4. Ran the target learner on the reduced dataset containing only attributes selected by TAR2. Accuracy was again obtained by averaging over cross validation.
5. The above steps were repeated for each dataset and each target learner (namely J4.8 and Naive Bayes).

	J4.8			Naive Bayes		
DataSet	Original	After TAR2	Diff	Original	After TAR2	Diff
anneal	98.2	98.2	0	86.6	84.3	-2.3
breast-c	75.2	75.2	0	74.1	75.2	1.1
credit-g	73.9	72.3	-1.6	75.9	74.3	-1.6
diabetes	74.5	72.8	-1.7	76	74.6	-1.4
horsecolic	85.3	81.5	-3.8	78.8	79.6	0.8
ionosphere	88.6	87.8	-0.8	82.9	87.5	4.6
lymph	76.4	74.3	-2.1	81.8	77.7	-4.1
segment	97.1	96.6	-0.5	79.8	86.3	6.5
soybean	92.4	93	0.6	92.7	93	0.3
vote	95.9	96.1	0.2	90.1	94.9	4.8
	Average		-0.97	Average		0.87

Table 5.2: Classification accuracy of J4.8 and Naive Bayes before and after using TAR2 as attribute subset selector

5.2.3 The Results

Table 5.2 shows the classification accuracy on ten datasets before and after attribute selection with J4.8 and Naive Bayes. In the J4.8 case, accuracy decreases on six datasets, remains the same on two and increases on two, average difference is a drop of 0.97%. In the Naive Bayes case, accuracy decreases on four datasets and increases on six, average difference is a rise of 0.87%. The largest accuracy drop is 4.05%, and the largest accuracy improvement is 6.5%, all occurred when the target learner is Naive Bayes. On average, accuracies change is less than 1%.

Table 5.3 compares the size (measured by the number of nodes) of decision trees produced by J4.8 with and without attribute selection. After selection, tree size increased on two datasets, remained the same on one and decreased

[1] Weka is freely available at `http://www.cs.waikato.ac.nz/~ml`

DATA-SET	Before	After	Diff	$\frac{Diff}{Before}$
anneal	47	55	8	17.02%
breast-c	6	6	0	0
credit-g	140	33	-107	-76.43%
diabetes	43	5	-38	-88.37%
horsecolic	6	3	-3	-50.00%
ionosphere	35	5	-30	-85.71%
lymph	34	11	-23	-67.65%
segment	77	81	4	5.19%
soybean	93	92	-1	-1.08%
vote	11	9	-2	-18.18%

Table 5.3: Size of trees (number of nodes) produced by J4.8 with and without attribute selection

on the rest seven. For five datasets, the resulting trees are at least 50% smaller than the original ones.

DATA-SET	ORIG	IG	CFS	CNS	RLF	WRP	PC	TAR2	$\frac{TAR2}{ORIG}$
anneal	38	16.6	21.3	15.5	20.4	18.2	36.4	7	18.4%
breast-c	9	4.4	4	6.6	6.9	3.98	4.4	2	22.2%
credit-g	20	7.8	6.7	8.1	9.1	7.7	3.9	5	25%
diabetes	8	3.2	3.4	3.6	3.9	3.8	5.9	1	12.5%
horsecolic	22	3.8	3.7	2.2	3.3	4.8	2.9	2	9.1%
ionosphere	34	12.2	6.9	9.3	8.7	7.2	10.2	2	5.9%
lymph	18	6.8	5.3	4	4.5	5.9	9.2	3	16.7%
segment	19	16.4	11.9	9.5	12.6	9.2	16.4	4	21.1%
soybean	35	29.5	23.7	35	32.4	19.2	30.2	16	45.7%
vote	16	11.6	9.6	6.5	10.6	8.6	11.2	6	37.5%

Table 5.4: Number of features selected for J4.8

Table 5.4 and 5.5 show the number of attributes selected by different selection schemes for each target leaner respectively. Column 2 lists the original number of attributes of each dataset, column 2-7 list the results of 6 stan-

DATA-SET	ORIG	IG	CFS	CNS	RLF	WRP	PC	TAR2	$\frac{TAR2}{ORIG}$
anneal	38	10.1	3.7	5.4	38.9	7.1	25.4	7	18.4%
breast-c	9	3.8	7.4	5.7	5.2	2.7	3.2	2	22.2%
credit-g	20	13.2	14.3	13.6	19.9	12.4	10.7	5	25%
diabetes	8	2.7	3.6	4	5.9	2.8	4.1	1	12.5%
horsecolic	22	5.8	4.1	3.9	22.8	5.8	6.2	2	9.1%
ionosphere	34	7.9	8.1	10.5	18.1	12.6	11.7	2	5.9%
lymph	18	16.6	13.1	14.3	15.3	15	13.1	3	16.7%
segment	19	11	11.1	5	15.2	7.9	9.2	4	21.1%
soybean	35	30.9	31.3	32.7	36	25.8	20.8	16	45.7%
vote	16	1	1.7	2.6	14.9	1	3	6	37.5%

Table 5.5: Number of features selected for Naive Bayes

dard methods seen in Hall and Holmes' report [HH02]. In their experiment, they used the target learner as part of the attribute subset evaluation and averaged the result over ten 10-way cross validation. Consequently, those methods gave different results with respect to different learners. Also as a side effect, the number of attributes selected by those methods is not an integer. Instead, TAR2's attribute selection is completely independent of the target learner, hence the number of attribute selected relies only on each dataset itself. For decision tree learner, TAR2 selected fewest attributes for all datasets while for the Naive Bayes, TAR2's selections are the smallest in eight out of ten datasets. In summary, TAR2 was the best overall feature selection methods studied here; i.e. it found the smallest feature subsets and those subsets resulted in minimal or no loss in classification accuracy.

5.3 Discussion

As mentioned earlier, feature subset selection techniques can be broadly categorized into "wrappers" and "filters" depending on their interaction with the target learning algorithms. In Hall's benchmark experiment [HH02] comparing six attribute selection methods, five of them are considered "filters". "Filters" accomplished dimensionality reduction following two steps:

- "Filters" produced ranked lists of attributes either unassisted or by using a modified forward selection hill climbing search. This procedure is independent of the target learner.
- Each ranked list was cross validated with respect to the current learner to estimate the worth of the subset of the ranked attributes. That is, cross validated on the training part of each dataset to estimate the worth of the highest ranked attribute, the first two highest ranked attributes, the first three highest ranked attributes and so on.

There exists a problem in step two: the evaluation procedure still relied on the learner to make the final selection of attribute subsets, thus unavoidably made any selector partially a "wrapper". Because wrappers use the learner in the search process to evaluate features, they generally give better results in terms of classification accuracy. Unfortunately, the added computational cost is inevitable: wrappers have to invoke the target learner for every attribute subset considered during the search. Kohavi and John [KJ97] report that their Wrapper method can take up to thousands of seconds to terminate. Our treatment learner approach, on the other hand, takes a pure "filter" approach. Both the search and evaluation relied on the data itself without interference of the learning algorithm. As a result, the procedure is very fast: the total runtime for any of the domains shown in the experiment is less than ten seconds.

5.4 Conclusion

We have examined treatment learning in the framework of feature subset selection for supervised classification. The result shows that: in general, our approach can reduce the dimensionality drastically with minimal or no loss in accuracy. In some cases, it helps to improve learner's performance. We also compared it to six standard feature subset selection methods. It has been seen that features selected by TAR2 was nearly always smaller than features selected by other methods.

This study is also a supportive piece of evidence of the *narrow funnel effect*. The ten datasets on which we based our experiments were collected in real world domains. They are neither synthesized nor particularly engineered to make learning easy. Instead, they were representative of problems that naturally arise in practice. In other words, *narrow funnel effect* is common in practice. For those domains, a few attributes serve as good class indicators and simple models are adequate to describe the underlying concept. The study of using treatment learning as feature subset selector suggests that treatment learning is an ideal approach to identify funnel variables should the target domain contain any.

Chapter 6

Application Of Treatment Learning

In the 21^{th} century, we are deluged by data: transaction data, scientific data, medical data and financial data. Unless we can process the mountain of information, we are likely to be buried by irrelevant data. Ironically, many data mining tools try to generate intricate theories that are too overwhelming to understand. For example, it is difficult to understand the prediction system of a neural network merely by studying the net topology and individual node weights. As another example, on the Mushroom data set (8124 examples, 22 attributes, 2 classes, available from UCI data repository [CEC98]), the recent border-based Emerging Pattern mining algorithm found 299811 borders, each representing about 2^{18} item sets [DL99]. Although the algorithm is efficient enough to find that huge number of patterns in about 30 minutes, this is far too many results to show to an end user. In fact, the result itself might be a source for further data mining in order to provide understandable knowledge. We believe that the essence of data mining does not (only) reside on what patterns can be identified, or how efficient a miner can discover all the patterns. Rather, it is the promise to benefit decision making that makes data mining so extraordinary. The premise of treatment learning, and the reason why we use it, is that

> *showing the differences between outcomes can be much clearer than describing each separate outcome for an actionable decision.*

Treatment learner quickly identifies the key factors that most *change* (or influence) a situation instead of merely list the *description* of the current situation.

This chapter shows examples of how others have integrated treatment learning into various research frameworks to assist decision making ([DO02] [Sm02] [MSCM02] [MRoS^{+}02]). Please note that the applications were conducted through a collaboration between the author of this thesis and other researchers. The author provided implementation of treatment learner, user manuals and actively maintained the download website. Other researchers led investigations in the examples described here. We thank all the domain experts for their knowledge and insights. It is this aggregate effort that facilitated the adoption of treatment learning, and led to rewarding research discoveries in return.

6.1 Application Approach

In application domains, depending on what kind of knowledge we have, there are two scenarios:

1. We have access to some historical data about a domain. Information is hidden in the data and needs to be extracted.

2. A model expressing what is known within a domain is available. However, our knowledge about the model is usually incomplete: we may be uncertain over parts of that model or uncertain about the domain itself. Uncertainty takes the form of parameter ranges. Stochastic simulation of the model with uncertain parameters results in an option space, or a data cloud visually.

In the first scenario, *Data mining + Validation + Decision* is our general approach. Using treatment learner as the data miner, we summarize and extract knowledge from the data set to give new insights about the domain. Other miners or algorithms might also be experimented for comparison purposes. Validation takes the form of N-way cross validation to ensure stable treatments.

In the second scenario where we have access to a domain model, *Simulation + Sensitivity Analysis + Decision* is applied. Simulation is based on both categorical and continuous values. Categorical simulations draw their inputs from known operational profiles of system inputs. For continuous variables, inputs are selected at random. In a sensitivity analysis, the key factors that most influence a model are isolated. Also, recommended settings for those key factors are generated. The settings can be validated by feedback and re-simulation. Before the sensitivity analysis can be performed, there must exist a domain-specific evaluation function that can assess the simulation records.

6.2 Feasibility of Agile Process

- **Domain**: assessment of software development paradigm

- **Data Source**: simulation through the Müller/Padberg model
- **Goal**: to assess the advantage of adopting the agile process based on pair programming.
- **Reference**: “Should NASA embrace Agile Processes?” [Sm02]
- **Collaborators**: Menzies, Smith[1], Hu (support role)

6.2.1 Agile Process and Pair Programming

Agile Process (AP) is an alternate paradigm to the conventional waterfall approach to software development. It is a collection of software design practices and techniques that diverge from the heavily structured methodologies in favor of a less structured, more adaptive approach [ea01]. AP values:

- Individuals and interactions over processes and tools;
- Working software over comprehensive documentation;
- Customer collaboration over contract negotiation;
- Responding to change over following a plan.

Among the broad and diverse AP movements, Kent Beck’s *extreme programming* is one of the most popular AP approaches [Bec00]. According to Beck, extreme programming has twelve key practices, among which *pair programming* (PP) plays a key role. PP is the concept of two developers working together at a single machine, designing and writing code cooperatively. PP proponents claim that by working in pairs, developers produce code at a faster rate with fewer errors. But the idea of developer pairs leads to two scenarios, each with their own issues:

[1] West Virginia University

- **Pool:** If additional developers are added to a project from a pool of developers to create the more pairs, does the time/error advantage outweigh the additional developer costs?
- **NoPool:** If additional developers are not available, and instead, the current developers are divided into groups of two, then does the time/error advantage outweigh the additional programming time resulting from the fewer number of tasks that can be worked on at one time?

6.2.2 Müller/Padberg Model

To answer the above questions, Müller and Padberg derived a model to compare the economics of PP with those of conventional methods [MP02]. Their model calculated the Net Present Value (NPV) of the software project as a general criterion to assess both programming methods. The NPV is calculated based on a series equations. Müller and Padberg used 8 fixed parameters and 4 parameters that they considered to be key features, for which they systematically varied their values (Table 6.2.2). They used these values to calculate the NPV for the conventional and the PP method, with each of them in two situations: **Pool** - when the conventional method is using n developers, the PP method is using n pairs, or $2n$ developers; and **NoPool** - when the conventional method is using n developers, the PP method is using $\frac{n}{2}$ pairs.

Their study found that PP is advantageous when the number of pairs is equivalent to the number of developers (scenario **Pool**). In other words, n developers are more efficient than $\frac{n}{2}$ pairs. They also found that PP is advantageous in this situation under three conditions:

Parameter						
PairSpeedAdvantage	10%	20%	30%	40%	50%	
PairDefectAdvantage	5%	10%	15%	20%	25%	30%
DefectRemovalTime	5h	10h	15h			
DiscountRate	0%	25%	50%	75%	100%	

Table 6.1: Parameters systematically varied by Müller and Padberg

1. the project is of small to medium size (*ProductSize* is not large)
2. the project is of high quality ($AssetValue$ is high)
3. the need for a rapid time to market is present (*DiscountRate* is high)

6.2.3 Menzies/Smith Studies

The problem with Müller-Padberg economic model is that by only varying 4 parameters, a large number of the model's attributes were left out. To correct this, Menzies and Smith re-implemented the model for a wider range exploration. Random values within appropriate ranges (see Table 6.2.3) were generated for a larger set of attributes. In stead of the absolute NPV value, they used the NPV ratio to compare the two methods:

$$R_{NPV} = \frac{Cost(PP)}{Cost(Conventional)}$$

when $R_{NPV} > 1$, pair programming is at an advantage over conventional approaches. They generated 10,000 examples from random simulation and calculated their R_{NPV} accordingly. TAR2 was then run through the data looking for parameter ranges that lead to the largest R_{NPV} value. Based on different configurations, they conducted several groups of experiments:

Parameter	Min	Max
PairSpeedAdvantage	10%	50%
PairDefectAdvantage	5%	30%
DefectRemovalTime (hours)	5	15
DiscountRate	0%	100%
ProductSize (LOC)	10000	250000
DeveloperSalary ($)	45000	65000
ProjectLeaderSalary ($)	60000	90000
AssetValue ($)	200000	2000000
DeveloperProductivity (LOC/month)	100	500
NumberofDevelopers	4	20
DefectsPerKLOC	4	100
DefectsNotEliminated	10%	80%

Table 6.2: Parameters and ranges used by Smith and Menzies

Group 1

In study **S1,** TAR2 explored every attribute for both **NoPool** and **Pool** scenarios. This study found three attributes that had the greatest impact on the ratio: $PairSpeedAdvantage$,$PairDefectAdvantage$ and $DefectRemovalTime$. The result makes perfect sense since:

1. $PairSpeedAdvantage$ and $PairDefectAdvantage$ represent advantages that PP has over conventional methods;
2. a high $DefectRemovalTime$ would result in conventional developers working significantly more than developer pairs given the assumption that conventional methods are more defect-prone.

Group 2

Considering the above three attributes may not be changeable in practice, study **S2** ignored them for other possible features. In **S2/Pool** three items were found to be most important:

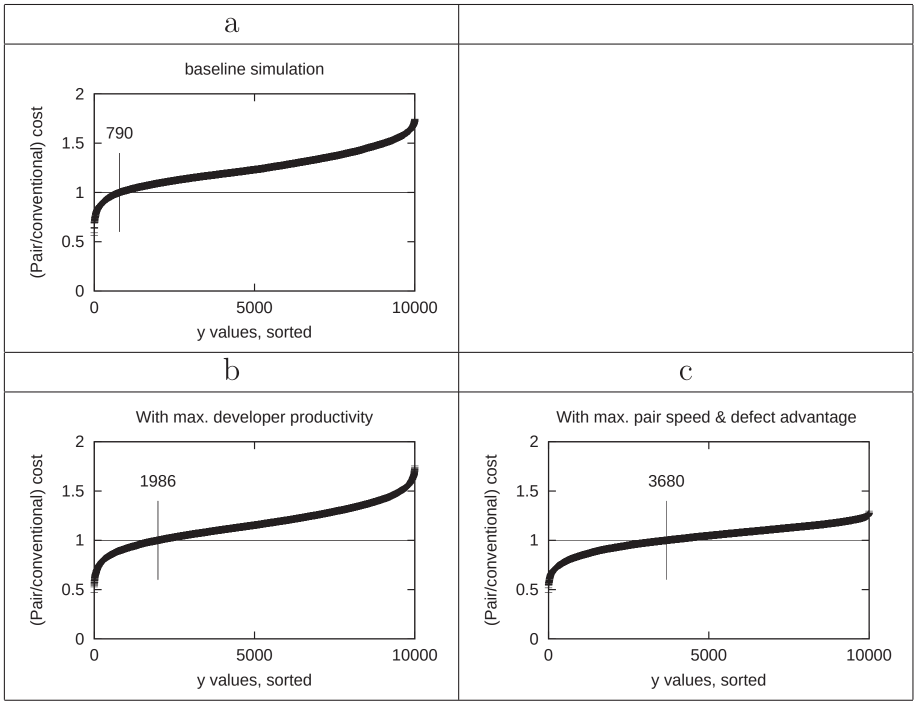


Figure 6.1: Raw data plots of a) completely random cases, b) cases with *DeveloperProductivity* set to maximum (**T1**), and c) cases with *PairSpeedAdvantage* and *PairDefectAdvantage* set to maximum (**T2**). The vertical line indicates the point where PP is no longer advantageous

1. A high *DiscountRate* (> 0.42)
2. A small *ProductSize* (10,000 to 60,000 LOC)
3. A high *AssetValue* ($\approx$ \$1.6M to \$2.0M)

In **S2/NoPool**, no significant treatments were found, indicating that, when a pool of developers is not present, PP does not have an advantage over conventional methods. Both conclusions are consistent with Müller/Padberg study described in section 6.2.2.

Group 3

To examine the effects of some particular attributes, two additional tests were conducted:

- **T1**:*DeveloperProductivity* was set to the maximum value, 500 LOC/month, to see the distribution when developers are being the most productive.
- **T2**: $PairSpeedAdvantage$ and $PairDefectAdvantage$ were set to their maximum values, 50% and 30% respectively, to see the distribution when pairs are operating at their greatest rate of advantage over individuals.

Results of these tests are shown in Figure 6.1. The vertical line in each plotting indicates the point where PP is no longer advantageous. In the baseline graph (Figure 6.1.a), PP is advantageous in only 8% examples. This low percentage increases to 20% When *DeveloperProductivity* was set to maximum (Figure 6.1.b). However, even when the two attributes that most favor PP are at their highest values, the greatest PP advantage percentage we've found is only 37%.

6.2.4 Discussion

By exploring a wide range of model simulation and summarizing using treatment learner, Menzies and Smith found pair programming only advantageous over conventional methods in a relatively small and specialized set of cases; i.e. when: a)the project is relatively small; b) an abundance of developers exists; and c)a rapid development time is demanded.

However, their results, showing consistency with the Müller/Padberg study, does not conclude the infeasibility of AP in practice. Rather, it indicates that a convincing case for AP cannot be based on the only factors encoded in this model. Other possible factors could include (e.g.) increased performance in rapid changing environments, decreased cost due to conventional requirements reworking to accommodate changes, or AP methods that do not rely on pair programming.

6.3 Software Metrics

- **Domain**: project quality analysis using software metrics
- **Data Source**: software metrics collected on NASA KC2 project.
- **Goal**: to identify metrics that are superior error predictors
- **Reference**: "Metrics That Matter" [MSCM02]
- **Collaborators**: Menzies, Di Stefano, Chapman, McGill[2], Hu (support role)

6.3.1 Background

Software metrics are attributes of software which can describe numerous things, including, but not limited to, complexity, effort, quality and reliability. Aiming at improving NASA's mission software regardless of the source, the NASA Independent Verification and Validation (IV&V) Facility creates and maintains a master repository of software metrics. Metrics are collected by reviewing requirements, code, and test results from NASA's most critical projects. A primary purpose of the repository is to identify early life cycle measures which may predict for error prone software modules. Figure 6.3

[2] West Virginia University

Metric Type	Metric	Definiton
McCabe	v(G)	Cyclomatic Complexity
	ev(G)	Essential Complexity
	iv(G)	Design Complexity
	LOC	Lines of Code
Halstead	N	Length
	V	Volume
	L	Level
	D	Difficulty
	I	Intelligent Content
	E	Effort
	B	Error Estimate
	T	Programming Time
Line Count	LOCode	Lines of Code
	LOComment	Lines of Comment
	LOBlank	Lines of Blank
	LOCodeAndComment	Lines of Code and Comment
Operator/Operand	UniqOp	Unique Operators
	UniqOpnd	Unique Operands
	TotalOp	Total Operators
	TotalOpnd	Total Operands
Branch	BranchCount	Total Branch Count

Table 6.3: Metric Groups.

outlines metrics being extensively used in NASA IV&V and in the study discussed here. Among them the McCabe complexity metrics [McC76] and Halstead metrics [Hal77] are two most popular ones that are used as a basis for predicting code errors.

6.3.2 The Experiment

NASA project KC2 is a C++ program containing over 3000 modules.[3] Among them, 521 modules were built by NASA developers and others are Commercial Off The Shelf software. Of those 521 modules, 106 were found to have various numbers of errors, ranging from 1 to 13. Software metrics information is gathered on the project especially the 512 modules to analyze the software quality. To isolate key metrics that predict for more/less errors,

[3] A module, for the purposes of our tests, is the equivalent of a C function.

Stefano et.al. divided the 512 modules into two groups according to their error rating: the 20% of modules with errors and the 80% of modules without errors. Thus, data set for our analysis contains 512 examples classified into two categories.

After performing treatment learning and 10-way cross validation on our data set, they found 2 best treatments, i.e., the best error indicators: the metrics indicating the least error was:

$$\boxed{L > 0.35}$$

and the metrics indicating most error was:

$$\boxed{LOC > 118}$$

The effects of these treatments are shown in Figure 6.2, along with the results from the customary McCabe metrics $v(G) > 10$ and $ev(G) > 4$. At the top of Figure 6.2 is the baseline class distribution. Compared to the baseline, $v(G) > 10$ and $ev(G) > 4$ are *both* good error indicators, in that they significantly alter the distribution from the baseline. It is interesting to note that $ev(G)$ actually performs better for KC2 than the more widely used $v(G)$, altering the baseline distribution by an additional 5%. However, neither of them are the *strongest* metrics to be using, since LOC and L (Halstead's Program Level) both have much better distributions. With a distribution of 97% error-free modules to 3% error-prone, $L > 0.35$ is actually a *very* strong one in this domain. For comparison purposes, a curve fitting algorithm was also applied to each metric. If it is correlated to the error rate, the R^2 value should be high (≥ 0.8). The first column of figure 6.2 includes the regres-

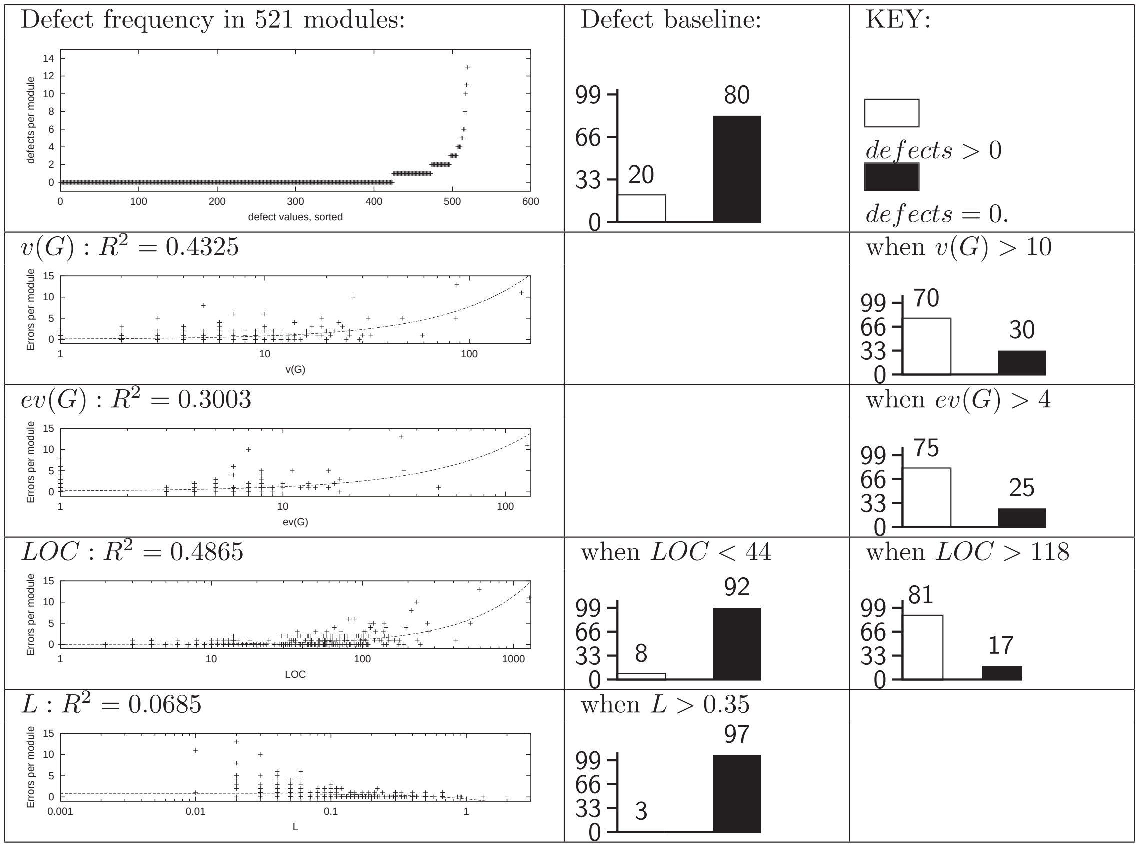


Figure 6.2: Results

sion curves[4]. In this particular project, *none* of the attributes were highly correlated to error rates. The strongest indicator L was the least correlated attribute ($R^2 = 0.0685$). This is a good indication that simplistic regression is ineffectual in finding decent error-predicting attributes. In summary, the study has shown three things which hold true in this particular domain:

- McCabe complexity metrics are *not* bad error-predictors, but others

[4] R is the correlation between output and input variables while R^2 is the coefficient of determination. R^2 represents the proportion of variability of the outputs that is accounted for by the inputs. Normally, R^2 is a measure of fit of a trend to a data series.

are better.

- LOC, a relatively cheap and easy-to-collect metric, is one of the best all-around error predictors.
- The least correlated metric, L, turned out to be the *strongest* error-free code indicator.

6.3.3 Discussion

This study uses treatment learner to seek best error-predict metrics on a particular software project. Given the complexity of correlations between metrics and module quality, treatment learner successfully found superior predictors while liner regression failed. Based on the results, it is obvious that good error predictors are project specific.

It is interesting to note that although McCabe complexity metrics are usually considered the most popular metrics, the experiment result shows that the cheap and easy to collect *LOC* metric performs exceptionally well as *both* a selector for error-prone and error-free modules. It has been suggested by other researchers that *LOC* may be a better metric to use when evaluating for error-prone code; the most notable example of this is Martin Shepperd's research [SI94]. Shepperd claimed that

> ...[Cyclomatic Complexity] is based upon poor theoretical foundations and an inadequate model of software development. The argument that the metric provides the developer with a useful engineering approximation is not borne out by the empirical evidence. Furthermore, it would appear that for a large class of software it is no more than a proxy for, and in many cases out-

performed by, lines of code (LOC).

This case study provided strong supportive evidences to Shepperd's opinion that $v(G)$ is often outperformed by LOC.

6.4 Software Inspection Policies

- **Domain**: study of software inspection policies
- **Data Source**: simulation through SE model
- **Goal**: to find the best inspection policy for a particular software development organization.
- **Reference**: "Model-based Tests of Truisms" [MRoS$^+$02]
- **Collaborators**: Menzies, Raffo[5], Hu

6.4.1 Modelling and Simulation

Software process modelling is a technique for understanding the interactions within a software development. The software process model used in this study was in two parts: a state-based simulation model built using the Statemate Magnum tool by i-Logix [6]; and a discrete event model using the Extend Simulation Language [7] [MRoS$^+$02]. This particular model has been extensively tuned and validated to a leading software development firm. It can accurately predict the impact of process changes. For example, for one very complex sub-system, the model predicted that development would take approximately

[5] Portland State University

[6] Statemate and I-Logix are registered trademarks of I-Logix Inc. (3 Riverside Drive; Andover, Massachusetts 01810 USA).

[7] Extend and Imagine That are registered trademarks r of Imagine That, Inc. (6830 Via Del Oro, Suite 230, San Jose, California, 95119 USA).

double the normal development schedule. This result was initially ignored by management as it was too long. However, months later, it was found that the model predictions corresponded quite accurately with this company's actual experience.

The model captures the phases of the company's software development process as well as the defect inspections being carried out at each phase. Each inspection is characterized by the number of staff involved which is a number drawn from distributions known to the model. There are four inspection policies:

1. do nothing;
2. do the companies current informal inspection method;
3. do a somewhat more-structured inspection process;
4. do a full formal inspection of the kind originally advocated by Micheal Fagan [Fag86].

These inspections can be conducted at various stages of the life cycle during

1. the initial functional specification;
2. after the high level design;
3. after the low level design;
4. or after the code is written.

After the number of staff involved in the inspections and the inspection policy at each stage are determined, the model predicts three main performance measures of cost, quality, and schedule using multiple regression. The output is assessed according to a domain-specific utility function. The utility function quantizes local concerns about trade-offs between cost and duration of the development process as well as the quality of the generated software. As a result, the final output is a number of defects estimated to be remaining in the software[Raf95].

6.4.2 Sensitivity Analysis

The model contains four phases of development and four inspection types at each phase. Each configuration was executed 50 times, resulting in $50 * 4^4 = 12800$ runs. The utility value of each run was calculated using the utility function and was further discretized into ten classes. The utilities are shown sorted as the *baseline* plot of Figure 6.3. Note the huge range of output values: 5,000 to 15,000.

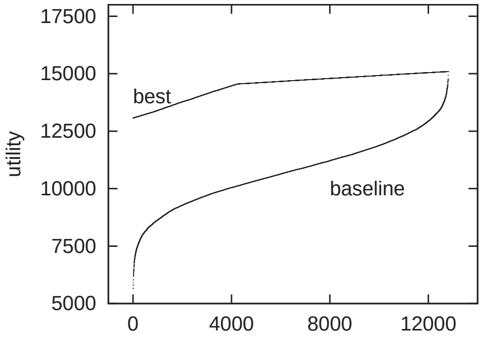


Figure 6.3: Sorted utilities generated in case study 2.

After using TAR2 to explore the simulation data, The best treatment found contains the following configuration:

- No functional specifications inspections;
- Full Fagan for low level design and code reviews.

- Baseline inspections for high level design; i.e. no change from current practice;

Given this configuration, TAR2 gave the distribution of utility values seen as the *best* plot of Figure 6.3. This preferred inspection policy increases the mean utility values seen in the baseline curve by a factor of 1.35 while reducing the standard deviation of those utilities by a factor of 2.5. This treatment was assessed via *10-way cross validation*. The *best* plot of Figure 6.3 was observed to be the average improvement seen under cross validation.

6.4.3 Discussion

When a decision tree learner was working on this complex domain, it generated a tree containing 7,206 nodes, which was far too large to understand. Instead of producing a tree, TAR2's output only mentioned the best inspection policy. Besides the succinct result, this study provides prescriptive guidance by identifying the ranges to which certain input parameters (such as inspection efficiency) must be restricted in order to achieve desired levels of performance. The paradigm of $decisions = modelling + simulations + sensitivity$ gives us the ability to examine the conditions under which the performance of a given system may be dramatically improved. Hence, it is optimistic to extend the current study to assessing standard automated software engineering methods.

6.5 Testability of Finite-State Models

- **Domain**: testability analysis of Finite-State Models

- **Data Source**: testability data gathered by a partial random search over FSMs
- **Goal**: identifies attributes that characterize easiest-to-test FSM models, therefore helping to understand what makes the FSM more or less testable.
- **Reference**: "What Makes Finite-State Models more (or less) Testable" [DO02]
- **Collaborators**: Menzies, Owen[8], Hu

6.5.1 FSM and Testability

Software systems with individual concurrent processes are often modelled as composite communicating Finite State Machines representing all possible interleavings within the system. Using a model mutator, Menzies, Owen and Cukic generated over 15,000 FSM models semi-randomly. Each FSM had parameter values drawn at random from possible ranges. The model parameters were selected to ensure that the ranges cover FSMs from real-world, and several *sanity checks* were imposed to block the generation of bizarre FSMs [MOC02]. Examples of sanity checks included things like each variable needed at least two settings and a FSM needed at least two states per machine.

A type of direct AND-OR graph could represent all possible interleavings of the individual FSMs in the original system [DO02]. In this representation, testing can be done by searching the AND-OR graph. Figure 6.4 illustrates our executable definition of testability. Given some input, if the number of

[8] West Virginia University

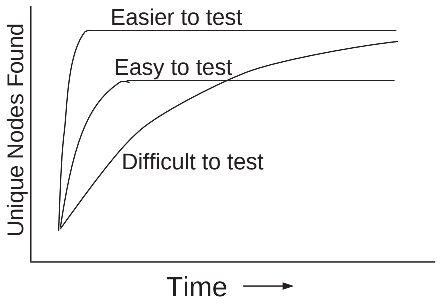


Figure 6.4: Intuitive testability interpretation of search results

unique outputs found as a result of that input rises quickly to a level plateau, a small number of tests will likely find everything it would be possible to find. Such a FSM represents a program that is *easy to test.* If the search never reaches a plateau, then even after many tests more tests might still give new information, so the program is very *difficult to test.* Menzies et.al, have built an inference engine to process the 15,000 generated FSMs. The testability was assessed via the percentage of FSM nodes reached in each run. This engine is nondeterministic in that, when there are two or more contradictory nodes to be added to the output set, the choice of which node to add is random.

6.5.2 Summarizing the Search

After searching over 15,000 semi-randomly generated models, the *time-to-plateau* parameter was calculated for each model. The result shows that plateaus were reached quickly for nearly all models, regardless of plateau height. If some method can increase the plateau height, that method would have increased the chances the odds of finding unique outputs. So the key

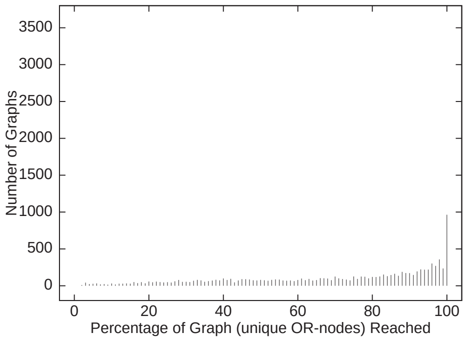


Figure 6.5: Plateau height results for 15,000 models. Average plateau height=69.39%

distinction, in terms of testability, is plateau height. Figure 6.5 shows a summary of plateau height from the search data. The average plateau height is 69.39%. The task is to analyze how FSM models yielding high search plateaus are different from those yielding low search plateaus. Specifically, what ranges of the attributes characterize the models with high plateaus?

After a series experiment with TAR2 on the search data, the treatments are summarized in table 6.4. Each treatment contains 4 attributes. It is interesting to notice that the three top parameters are low for both highly testable graphs and those difficult to test. In the real world, they would resemble the "risk factors" that polarize the outcome. For example, if somebody spent all his money buying lottery tickets he would end up either rich or poor, compared to his original situation. The bottom half of table 6.4 shows which attributes have the greatest affect on testability, given that the top three are held low. They are *state inputs*, followed by *message inputs* and *message outputs*. The result indicates that smaller FSMs are not necessarily more testable. Larger, more complex FSMs are likely to fall in the middle-to-

⟵ Better Treatments

Machines	lowest (2–4)	lowest	lowest
States	lowest (4–49)	lowest	lowest
Transitions	low (0–109)	low	low
State Inputs	high (443–737)		
Messages		(not significant)	
Message Inputs		high (389–647)	
Message Outputs			high (432–719)

Worse Treatments ⟶

Machines	lowest (2–4)	lowest	lowest
States	lowest (4–49)	lowest	lowest
Transitions	lowest (0–54)	lowest	lowest
State Inputs			lowest (0–147)
Messages		(not significant)	
Message Inputs		lowest (0–129)	
Message Outputs	lowest (0–143)		

Table 6.4: Best and worst treatments learned by TAR2.

high testability range; and simpler FSMs are likely to be either very testable or very difficult to test. It is the connectedness (the number of transition inputs and outputs), not size that is most important for testability.

6.5.3 Applying Gained Knowledge

To verify the results, another 10,000 input models were generated by setting the mutator's parameters according to TAR2's recommendation:

1. 2–5 FSMs.
2. 4–49 states.
3. 0–43 transitions.
4. 0–247 transition inputs that are states from other machines.
5. 0–10 unique consumable messages.
6. 0–229 transition inputs that are consumable messages.
7. 0–241 transition outputs that are consumable messages.

Figure 6.6 shows the search result of the new data. Compared to figure 6.5, the average plateau height was increased from 69.39% to 91.34%.

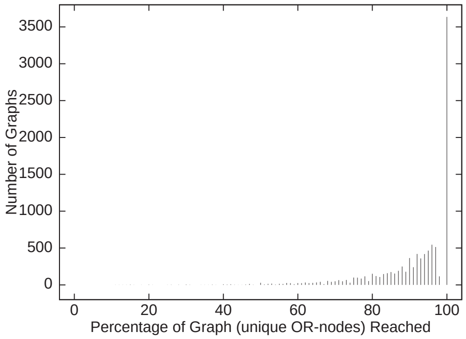


Figure 6.6: Search data for input models generated according to TAR2's suggestions—average plateau height = 91.34%.

6.5.4 Discussion

In this example, learning is manifested in the process of data summarization and knowledge acquisition. In the case when testability is assessed via plateau height, treatment learning enables the automatical learning of the features that most effect FSM's testability. Given the availability of an automatic testability assessment method, this approach is easy to generalize to other representations and other definitions of testability. The model features (parameters and their values) found by treatment learning could always be fixed and tested, resulting a verifiable feedback.

Another possibility that arises from this work is that researchers can identify design parameters that make nondeterministic FSM-based systems *more* or *less* testable. For example, given two implementations of the same requirement, researchers could favor the implementation that results in a more testable system. In other words, it is possible to *design for testability*, even for nondeterministic systems.

6.6 conclusion

The examples shown here have demonstrated, in real world domains, the capability and effectiveness of exploiting treatment learning. Treatment learner seeks minimal summaries of large data sets and shows the differences between outcomes. In model-based domains , the paradigm of *decision = simulations + sensitivity analysis* provides a fast and cheap way to assist decision making, since it:

- reduces the cost of elaborate modelling. Essential information can be extracted from even hastily built models containing much uncertainty.

- reduces the cost of extensive data collection. It is possible to *grow* data in data starve domain by simulation, and *harvest* by sensitivity analysis.

In data-present domains, the paradigm of *decisions = data mining + validation* proposes a novel and succinct KDD process to report contrasts between classes that could make a difference.

We believe treatment learning has a general applicability, and by presenting the few successful cases, we hope to stimulate wider interest in its adoption.

Chapter 7

Conclusion

7.1 Main Contributions

In this thesis, we have proposed a new learning approach called treatment learning. It addresses two central issues in data mining: (1) the understandability of learnt theories; (2) how can the learnt theories benefit decision making. We have studied this approach from the following four aspects:

- We have described the implementation detail of two treatment learners. We have compared them through algorithmic performance analysis.

- We have demonstrated, through both UCI [CEC98] data experiments and case studies, the effectiveness of using treatment learner to seek a small number of control variables that constrain the option space to a tight, near-optimal convergence.

- We have compared treatment learning with other learning schemes in the framework of feature subset selection for supervised classification. The results show that treatment learner selects smaller feature subsets than most other methods with minimal or no loss in classification accuracy.

- We have presented four real world applications of treatment learning, both in model-based domains and in data-present domains. The examples suggest two general paradigms of using treatment learning to assist decision making.

We list below the principal contributions of our research to the field of machine learning and data mining:

1. We introduce in the concept of treatment learning. Treatment learning aims at mining a small number of control variables in a large option space that can lead to better system behavior. It offers two distinguishing features to the community:
   - It produces minimal theories that are small, simple and easily understandable from the target domain.
   - It emphasizes on the interpretation of the learnt theory by human experts to inspire decision making.
2. We designed and implemented a novel mining algorithm based on the heuristic of $Confidence1$ measure.
3. We continuously work on the optimization of the initial design. The latest implementation presents an algorithm that runs in linear time in cases the original version runs exponentially.
4. We delivered a complete package of treatment learner and actively maintained a free online distribution. This package has been used extensively by other researchers (see chapter 6).

5. We point out the non-trivial observation of *narrow funnel effect* and provide treatment learning as both an evidence and an application.

6. We propose the use of treatment learner as feature subset selector and present a benchmark comparison with other standard FSS techniques.

7. We successfully applied treatment learning to various research domains and demonstrated the applicability of different approach.

7.2 Future Work

Our major topics of interest for future research are as follows:

1. Our current treatment learner uses a straightforward discretisation method, namely the *Equal Width Interval Binning*[DKS95] to discretize continuous attributes. This method divides the values into N intervals where N is pre-specified by the user. In the future, we would like to try other schemes such as the supervised discretisation developed by Fayyad and Iraini [FK93]. It combines an entropy based splitting criterion with a minimum description length stopping criterion. The best cut point is the one that makes the subintervals as pure as possible, i.e. where the *information value* is smallest. This splitting is essentially the same as the one used by the C4.5 decision tree learner [Qui86]. In that case, the best split is where the *information gain*, defined as the difference between the information value without the split and that with the split, is largest. The discretisation is then applied recursively to the two subintervals until it is time to stop. This method has an

advantage of not requiring the number of intervals to be pre-specified. It is considered to be the state-of-the-art.

2. Based on our feature subset selection experiment, we are optimistic to use treatment learner as a pre-processor prior to any target learners for supervised classification. However, the current approach involves a non-trivial procedure of alternating class ordering and combining the result together. This process can be easily automated by adding a wrapper around the learner. It will invoke the learner's core procedure for each class ordering and output the final set of selected features instead of individual treatments. This will give us a permutation that can be readily in use as feature subset selector. As a result, we'd like to experiment with more datasets (especially ones with high dimensionality) and different target classifiers to further explore the applicability of using treatment learner as feature subset selector.

3. Treatment learner involves a combination of search and attribute utility estimation. Adoption of different search techniques and estimation schemes will result in many permutations. To try other estimation schemes such as the *information gain attribute ranking* ,*Relief* [I.K94], we need to modify them such that they can reflect the domain-specific preference of classes (e.g., the weights of each class) since treatment learning is a different task from classification. Another possibility to the current $confidence1$ scheme would be to produce subset ranking instead of individual attribute ranking. This way, treatment learner operates more like standard association rule miners, in which APRIORI

property [AS94] could be exploited. Also, the increased search space dictates more sophisticated search strategy such as simulated annealing that evaluates the better nodes more times. One drawback of this approach is that it assumes complex hypothesis and makes treatment learner no longer a lightweight tool.

The first two ideas aim at enhancing some aspects of the existing treatment learner. They are straightforward to realize with the current implementation. The third idea makes some radical changes to the core algorithm. Most likely it will result in essentially different learners in the framework of treatment learning. Nevertheless, in the future we would like to promote both wider application of the current learner and some breakthroughs in the concept of treatment learning.

References

[ACDC+98] C. Abts, B. Clark, S. Devnani-Chulani, E. Horowitz, R. Madachy, D. Reifer, R. Selby, and B. Steece. COCOMO II model definition manual. Technical report, Center for Software Engineering, USC,, 1998. `http://sunset.usc.edu/COCOMOII/cocomox.html#downloads`.

[Aha97] David W. Aha. *AI review: Special Issue on Lazy Learning.* 1997.

[AS94] Rakesh Agrawal and R. Srikant. Fast algorithm for mining association rules. In *Proceedings of the 20th international conference of VLDB, page 487-499*, 1994.

[Bay98] R.J. Bayardo. Efficiently mining long patterns from database. *Proceedings of the 1998 ACM-SIGMOD International Conference on Management of Data, pp.85-93.*, 1998.

[Bec00] Kent Beck. *Extreme Programming Explained: Embrace Change.* Addison Wesley, 2000.

[BFOS84] L. Breiman, J. Friedman, R. Olshen, and C Stone. *Classification and regression trees.* 1984.

[BLM] W. Hsu B. Liu and Y. Ma. Integrating classification and association rule mining. In *Proceedings of KDD-98, New York, Aug27-31,1998*, pages 80–86.

[BP99] S.D. Bay and M.J. Pazzani. Detecting changes in categorical data: Mining contrast sets. In *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining*, 1999. Available from `http://www.ics.uci.edu/~pazzani/Publications/Publications.html`.

[BP01] S.D. Bay and M.J. Pazzani. Detecting group differences: Mining contrast sets. *Data Mining and Knowledge Discovery, 5,213-246*, 2001.

[CB94] J. Crawford and A. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In *AAAI '94*, 1994.

[CEC98] C.Blake, E.Keogh, and C.J.Merz. Uci rpository of machine learning databases, university of california, department of information oand computer science, irvine, ca, 1998.

[CFH01] S.L. Cornford, M.S. Feather, and K.A. Hicks. Ddp a tool for life-cycle risk management. In *IEEE Aerospace Conference, Big Sky, Montana*, pages 441–451, March 2001.

[CH67] T.M. Cover and P.E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory, 13, pp.21-27*, 1967.

[CW98] C.H.Cheng C.H.Cai, Ada W.C.Fu and W.W.Kwong. Mining association rules with weighted items. In *In Proceedings of International Database Engineering and Applications Symposium (IDEAS 98), Aug 1998*, 1998.

[DB96] J. Davies and D. Billman. Hierarchical categorization and the effects of contrast inconsistency in an unsupervised learning task. In *In Proceedings of the Eighteetnth Annual Conference of the Congnitive Science Society*, page 750, 1996.

[deM91] R.L. deMantaras. A distance-based attribute selection measure for decision tree induction. *Machine Learning,29,103-30*, 1991.

[DH73] R. Duda and P. Hart. *Pttern classification and scene analysis.* NewYork: John Wiley and Sons, 1973.

[D.H96] D.Heckerman. Bayesian networks for knowledge discover. *Advances in Knowledge Discovery and Data Mining, pp.273-305.*, 1996.

[DKS95] James Dougherty, Ron Kohavi, and Mehran Sahami. Supervised and unsupervised discretization of continuous features. *International Conference on Machine Learning, pp. 194-202.*, 1995.

[DL99] Guozhu Dong and Jinyan Li. Efficient mining of emerging patterns: Discovering trends and differences. In *Knowledge Discovery and Data Mining*, pages 43–52, 1999. Available from `citeseer.nj.nec.com/article/dong99efficient.html`.

[DO02] Bojan Cukic David Owen, Tim Menzies. What makes finite-state models more (or less) testable. *Submitted to the 17th IEEE international conference on automated software engineering.*, 2002.

[ea01] M. Beedle et. al. Manifesto for agile software development, 2001. Available from `http://www.agilemanifesto.org/`.

[EC00] Shinji Fujiwara Aristides Gionis Piotr Indyk Rajeev Motwani Jeffrey D. Ullman Cheng Yang Edith Cohen, Mayur Datar. Finding interesting associations without support pruning. In *In ICDE, pages 189-499, Feb 2000*, 2000.

[Fag86] M. Fagan. Advances in software inspections. *IEEE Trans. on Software Engineering*, pages 744–751, July 1986.

[Fau94] L. Fausett. *Fundamentals of Neural Networks. Architecutres, Algorithms and Applications.* Prenice Hall., 1994.

[FK93] U.M. Fayyad and K.B.Irani. Multi-interval discretisation of continuous-valued attributes. *in Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence. 1993, pp. 1022-1027*, 1993.

[FM02a] M. Feather and T. Menzies. Converging on the optimal attainment of requirements. In *In IEEE Joint Conference On Requirements Engineering ICRE'02 and RE'02, 9-13th September, University of Essen, Germany, 2002.*, 2002. Available from `http://tim.menzies.com/pdf/02re02.pdf`.

[FM02b] M.S. Feather and T. Menzies. Converging on the optimal attainment of requirements. In *IEEE Joint Conference On Requirements Engineering ICRE'02 and RE'02, 9-13th September, University of Essen, Germany*, 2002. Available from `http://tim.menzies.com/pdf/02re02.pdf`.

[Hal77] M.H. Halstead. *Elements of Software Science*. Elsevier, 1977.

[Hay99] S.S. Haykin. *Neural networks: Acomprehensive foundation.* Upper Saddle River, N.J.: Pretice Hall, 1999.

[HH02] Mark A. Hall and Geoffrey Holmes. Benchmarking attribute selection techniques for discrete class data mining. *IEEE transactions on knowledge and data engineering*, 2002.

[Hol] Robert C. Holte. *Very Simple Classification Rules Perform Well On Most Commonly Used Datasets*, pages 63–91.

[Hop82] J.J. Hopfied. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences, USA (pp.2554-2558)*, 1982.

[HR96] H.Liu and R.Setiono. A probabilistic approach to feature selection: A filter solution. *in Proceedings of 13th International Conference on Machine Learning, pp.319-327.*, 1996.

[HS66] Marin J. Hunt, E.B. and P.T. Stone. *Experiments in induction.* New York: Academic Press, 1966.

[HS86] G.E. Hinton and T.J. Sejnowski. Learning and re-learning in boltzmann machines. *Parallel distributed processing, vol.1,chap.7*, 1986.

[HT91] H.Almuallim and T.G.Dietterich. Learning with many irrelevant features. *in Proceedings of the Ninth National Conference on Artificial Intelligence, pp.547-552.*, 1991.

[I.K94] I.Kononenko. Estimating attributs: Analysis and extensions of relief. *in Proceedings of the Seventh European Conference on Machine Learning, pp.171-182.*, 1994.

[JH01] Micheline Kamber Jiawei Han. *Data Mining: Concepts and Techniques*. 2001.

[KJ97] Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artificial Intelligence Volum 97, pages 273-324*, 1997.

[Koh96] R. Kohavi. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. *Proceedings of the second international conference on knowledge discovery and data mining,pp202-207.*, 1996.

[KW01] David Cheung Francis Chin Ke Wang, Yu He. Mining confident rules without support requirement. In *the 10th ACM International Conference on Information and Knowledge Management (CIKM 2001), Atlanta*, 2001.

[LDR00] Jinyan Li, Guozhu Dong, and Kotagiri Ramamohanarao. Making use of the most expressive jumping emerging patterns for classification. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 220–232, 2000. Available from `citeseer.nj.nec.com/li00making.html`.

[LK98] D.I. Lin and Z.M. Kedem. Pincer-search: A new algorithm for discovering the maximum frequent set. *Proceedings of the 6th International Conference on Extending Database Technology, pp.105-119.*, 1998.

[M.A98] M.A.Hall. Correlation-based feature selection for machine learning. *Ph.D. thesis, Dept. of Computer Science, University of Waikato, Hamilton, New Zealand.*, 1998.

[McC76] T.J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, 2(4):308–320, December 1976.

[MCF$^+$02] T. Menzies, E. Chiang, M. Feather, Y. Hu, and J.D. Kiper. Condensing uncertainty via incremental treatment learning. In *Annals of Software Engineering*, 2002. Available from `http://tim.menzies.com/pdf/02itar2.pdf`.

[MENW99] T.J. Menzies, S. Easterbrook, Bashar Nuseibeh, and Sam Waugh. An empirical investigation of multiple viewpoint reasoning in requirements engineering. In *RE '99*, 1999. Available from `http://tim.menzies.com/pdf/99re.pdf`.

[MH01a] T. Menzies and Y. Hu. Constraining discussions in requirements engineering. In *First International Workshop on Model-based*

Requirements Engineering, 2001. Available from http://tim.menzies.com/pdf/01lesstalk.pdf.

[MH01b] T. Menzies and Y. Hu. Reusing models for requirements engineering. In *First International Workshop on Model-based Requirements Engineering*, 2001. Available from http://tim.menzies.com/pdf/01reusere.pdf.

[MH01c] T. Menzies and Y. Hu. Reusing models for requirements engineering. In *Submitted to the first International Workshop on Model-based Requirements Engineering*, 2001. Available from http://tim.menzies.com/pdf/01reusere.pdf.

[MH02] T. Menzies and Y. Hu. Agents in a wild world. In *Book chapter, submitted to Formal Approaches to Agent-Based Systems*, 2002.

[Min89] J. Mingers. *An empirical comparison of selection measures fordecision-tree induction*. 1989.

[MK01] T. Menzies and J.D. Kiper. How to argue less. In *Submitted to the ACM CIKM 2001: the Tenth International Conference on Information and Knowledge Management*, 2001. Available from http://tim.menzies.com/pdf/01jane.pdf.

[MOC02] T. Menzies, D. Owen, and B. Cukic. Saturation effects in testing of formal models. In *ISSRE 2002*, 2002. Available from http://tim.menzies.com/pdf/02sat.pdf.

[MP02] Matthias M. Mller and Frank Padberg. Extreme programming from an engineering economics viewpoint. In *Proceedings of the*

Fourth International Workshop on Economics-Driven Software Engineering Research (EDSER), 2002.

[MRoS$^+$02] Tim Menzies, David Raffo, Siri on Setamanit, Ying Hu, and Sina Tootoonian. Model-based tests of truisms. In *Proceedings of IEEE ASE 2002*, 2002. Available from `http://tim.menzies.com/pdf/02truisms.pdf`.

[MSCM02] Tim Menzies, Justin S. Di Stefano, Mike Chapman, and Ken McGill. Metrics that matter. *Submitted to the 27th Annual IEEE/NASA Software Engineering Workshop.*, 2002.

[Qui86] R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[Qui88] J.R Quinlan. *Decision trees and multi-valued attributes*. Oxford, UK:Oxford University Press, 1988.

[Qui92] R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufman, 1992. ISBN: 1558602380.

[Raf95] D.M. Raffo. Modeling software processes quantitatively and assessing the impact of potential process changes of process performance, 1995. Ph.D. thesis, Manufacturing and Operations Systems, Carnegie Mellon University.

[RH97] Y. Dan Rubinstein and Trevor Hastie. Discriminative vs informative learning. In *Knowledge Discovery and Data Mining*, pages 49–53, 1997. Available from `citeseer.nj.nec.com/rubinstein97discriminative.html`.

[Ros62] F. Rosenblatt. *Principles of neurodynamics: Perceptrons andthe theory of brain mechanisms.* Washington D.C.:Spartan Books, 1962.

[Rym92] R. Rymon. Search through systematic set enumeration. *In 3rd International Conference on Principles of Knowledge Representation and Reasoning.*, 1992.

[SI94] M. Shepperd and D.C. Ince. A critique of three metrics. *The Journal of Systems and Software*, 26(3):197–210, September 1994.

[SJDM98] S.Dumais, J.Platt, D.Heckerman, and M.Sahami. Inductive learning algorithms and representations for text categorization. *in Proceedings of the International Conference on Information and Knowledge Management, pp.148-155.*, 1998.

[Sm02] Jefferey Smith and Tim menzies. Should nasa embrace agile processes? *Submitted to the 27th Annual IEEE/NASA Software Engineering Workshop.*, 2002.

[SMT91] J. Shavlik, R. Mooney, and G. Towell. Symbolic and neural learning algorithms: An experimental comparison. *Machine Learning, 6(2): 111–143*, 1991.

[TA93] Rakesh Agrawal T.Imeilinski and A.Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD Conference, Washington DC, USA*, 1993.

[UPSS96] U.M.Fayyad, G. Piatetsky-Shapiro, and P. Smyth. *Advances in knowledge discovery and data mining, 1-34*. AAAI/MIT Press, 1996.

[Web00] Geoffrey I. Webb. Efficient search for association rules. In *Proceeding of KDD-2000 Boston, MA*, 2000.

[WF99] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1999.

[WZH00] Ke Wang, Senqiang Zhou, and Yu He. Growing decision trees on support-less association rules. In *Knowledge Discovery and Data Mining*, pages 265–269, 2000.

[ZPOL97] M.J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithm for fast discovery of association rules. *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining, pp.283-286.*, 1997.

Appendix A

User Manual for TAR3 Treatment Learner

A.1 Getting TAR3

The TAR3 treatment learner is distributed under the GNU General Public License and is freely available at the online distribution[1]. For installation, simply download the newest TAR3 package (dispatchTAR3.zip) and unzip it to your local computer. The whole package contains the following file structure:

- /bin: folder where all the executables reside
- /doc: related publications and user manual
- /sample: sample data sets and their configuration files
- /source: C source code

[1] `http://www.ece.ubc.ca/~yingh/`

Table A.1: Parameters seen in the configuration file.

Name	Meaning	Default
granularity	how many intervals should a continuous attribute be divided	3
maxNumber	maximum number of treatments wanted	30
minSize	minimum treatment size expected in a single treatment	1
maxSize	maximum treatment size allowed in a single treatment	5
randomTrials	maximum random trials tried before stop	1
futileTrials	number of successive futile trials to be completed before stop	5
bestClass	percentage of best class examples expected to be remained in the treated set	50%

A.2 Configuration File

Table A.1 lists the parameters used in the configuration file (xx.cfg). Note that the order of the parameters are not important, and if a parameter is missing, TAR3 will take the default value as listed in table A.1.

TAR3 adopts a random sampling algorithm to draw treatments from the underlying distribution. Parameter `randomTrials` and `futileTrials` are part of the randomness control. Suppose we set:

```
maxNumber = 100
randomTrials = 50
futileTrials = 10
```

In each random trial, TAR3 generates a set of treatments and maintains a list of 100 top ranked treatments. If a random trial doesn't contribute new treatments into that list (e.g., treatments generated in that trial have lower rank than those already in the list), it is called a futile trial. The process stops after completing 50 random trials or after 10 successive futile trials are reached. Empirically, setting `randomTrials` between 30 to 60 and `futileTrials` between 5 to 10 are usually sufficient to get stable treatments.

A.3 Name File

The `.names` file consists of a series of sections, each of which has restrictions and format. Blank lines, spaces, and tabs may be used to make the file more readable and have no significance. The vertical bar character(|) appearing anywhere on a line causes the rest of that line to be ignored, and can be used to incorporate comments in the file.

A.3.1 Name Restriction

1. A name cannot be the single character "?"

2. The special characters comma(,), colon(:), vertical bar(|), and backslash have particular meanings and must be escaped (preceded by a backslash character) if they appear in a name.

3. A period(.) may appear in a name provided it is not followed by a space.

4. Embedded spaces are also permitted in a name, but multiple whitespace characters (spaces and tabs) are replaced by a single space.

A.3.2 Class Format

1. The first entry in the names file gives the class names, separated by commas.

2. There must be at least two class names.

3. Classes are ordered from the domain-specific point of view, with the worst first, the best last.

A.3.3 Attribute Format

1. An attribute entry begins with its name followed by a colon, and then a specification of the values it can take.

2. `continuous`: indicates that the attribute has numeric values, either integer or floating point.

3. A list of names separated by commas: indicates that the attribute has discrete values and specifies them explicitly. The order of attribute values is arbitrary.

A.3.4 Optional Sections

TAR2 takes the three sections as inputs to restrict the data processing scope of a particular data set.

- NOW section: NOW specifies the current status of the data, i.e., only those satisfy NOW criteria will be read in and processed. This data pre-process could always be obtained by using other tools.

- CHANGES section: CHANGES represents some desired zone within the data set that the user wishes to approach. Only attribute ranges specified in CHANGES could appear in the treatments.
- SCORE section: SCORE encodes user's preference of the classes. User can assign a specific score (weight) to a class. Without user specification, TAR3 scores the classes according to a default scoring function.

The above three sections are optional, but once they appear, their relative order is important. e.g., CHANGES section must be after NOW section and SCORE section must be the last.

A.3.5 Little Language

A little language is designed to specify attribute ranges in NOW and CHANGES sections, for example:

- `Attribute1:true`:
  all possible values are acceptable
- `Attribute2:ignore`:
  none values are acceptable
- `Attribute3:a, b, c`:
  for categorical attribute, only values a, b, c are acceptable
- `Attribute4:[-;10), [20;30], [50;-)`:
  for continuous attribute,the acceptable ranges are: $x < 10$ OR $20 <= x <= 30$ OR $x <= 50$

A.4 Command Line

Suppose the data set to use is `c:/tar3/data/myDataset.data`. The following files are required to be placed into the same folder:

- data file: `c:/tar3/data/myDataset.data`
- name file: `c:/tar3/data/myDataset.names`
- configuration file: `c:/tar3/data/myDataset.cfg`

To invoke TAR3, issue the command: (suppose `tar3.exe` resides in `c:/tar3/bin`)

```
cd c:/tar3/data
c:/tar3/bin/tar3 myDataset
```

A.5 Cross-Validation

TAR3 also comes with a cross-validation facility (`/bin/xval.exe`). This program is compatible with both TAR2(v2.2) and TAR3. To invoke it, use one of the following three commands:

- `xval tar2 fileName N`:
  N-way cross validation with tar2 on `fileName.data`
- `xval tar3 fileName N`:
  N-way cross validation with tar3 on `fileName.data`
- `xval -p   fileName N`:
  perform file-split on `fileName.data`

The current directory must be the one where the data files reside. If tarX.exe and xval.exe are in different folders from the current directory, full path must be specified. For example, to perform 10-way cross validation on myDataset.data, we issue the command:

```
cd c:/tar3/data
c:/tar3/bin/xval  c:/tar3/bin/tar3 myDataset 10
```

xval.exe first splits the data file in to N .data file and N .test files, resulting in:

```
XDF[0..N-1].data
XDF[0..N-1].test
```

Then it invokes tar2 or tar3 N times, generating N output files plus one summary file. After done, it automatically delete XDF*.data and XDF*.test