AN ABSTRACT OF THE DISSERTATION OF

Valentina Bayer Zubek for the degree of Doctor of Philosophy in Computer Science presented on July 16, 2003.

Title: Learning Cost-Sensitive Diagnostic Policies from Data.

Abstract approved:

Thomas G. Dietterich

In its simplest form, the process of diagnosis is a decision-making process in which the diagnostician performs a sequence of tests culminating in a diagnostic decision. For example, a physician might perform a series of simple measurements (body temperature, weight, etc.) and laboratory measurements (white blood count, CT scan, MRI scan, etc.) in order to determine the disease of the patient. A diagnostic policy is a complete description of the decision-making actions of a diagnostician under all possible circumstances. This dissertation studies the problem of learning diagnostic policies from training examples. An optimal diagnostic policy is one that minimizes the expected total cost of diagnosing a patient, where the cost is composed of two components: (a) measurement costs (the costs of performing various diagnostic tests) and (b) misdiagnosis costs (the costs incurred when the patient is incorrectly diagnosed). The optimal policy must perform diagnostic tests until further measurements do not reduce the expected total cost of diagnosis.

The dissertation investigates two families of algorithms for learning diagnostic policies: greedy methods and methods based on the AO* algorithm for systematic search. Previous work in supervised learning constructed greedy diagnostic policies that either ignored all costs or considered only measurement costs or only misdiagnosis costs. This research recognizes the practical importance of costs incurred by performing measurements and by making incorrect diagnoses and studies the tradeoff between them. This dissertation develops improved greedy methods. It also introduces a new family of learning algorithms based on systematic search. Systematic search has previously been regarded as computationally infeasible for learning diagnostic policies. However, this dissertation describes an admissible heuristic for AO* that enables it to prune large parts of the search space. In addition, the dissertation shows that policies with better performance on an independent test set are learned when the AO* method is regularized in order to reduce overfitting.

Experimental studies on benchmark data sets show that in most cases the systematic search methods produce better diagnostic policies than the greedy methods. Hence, these AO*-based methods are recommended for learning diagnostic policies that seek to minimize the expected total cost of diagnosis.

Learning Cost-Sensitive Diagnostic Policies from Data

by
Valentina Bayer Zubek

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented July 16, 2003
Commencement June 2004

Doctor of Philosophy dissertation of Valentina Bayer Zubek presented on July 16, 2003.

APPROVED:

Major Professor, representing Computer Science

Director of the School of Electrical Engineering and Computer Science

Dean of the Graduate School

I understand that my dissertation will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my dissertation to any reader upon request.

Valentina Bayer Zubek, Author

ACKNOWLEDGMENTS

To the Holy Trinity, thank You for my life, for the people around me and for this beautiful world. Thank you for giving me hope and showing me there is life outside my narrow research universe, in the beauty of grass, flowers, trees, mountains, oceans and in the songs of birds. Thank you for lifting up my heart and letting the sun shine on me after the rain. Thank you for Oregon. Thank you for carrying me through the adversities of the last year, and making me appreciate how fortunate one is to be healthy and able to do research in a quiet environment in a rich country.

To my advisor, Professor Tom Dietterich, thank you for your trust in me. Thank you for your patience in making me understand, in teaching me how to do research, how not to get caught up in details (I mostly do!) and how to improve my writing. I learned a lot from you. Thank you for your time (I always demanded more!), scientific and personal guidance, thank you for supporting me as a research assistant and introducing me to the scientific community. Thank you for your generosity, enthusiasm and patience, and for being a great teacher. Thank you for your careful reading of my thesis and for allowing me to do pure research during the last year. I will always be proud to have been your student. And thank you for encouraging me in several key moments. And for being so understanding with my backpacking endeavors. You are the reason I came to Oregon State University and the optimistic scientific wave that finally brought me to the shore of completing my Ph.D.

To Professor Prasad Tadepalli, thank you for being one of the most kind professors I have met. To me, you are the image of the perfect teacher, knowledgeable, patient and caring. Thank you for taking time to clarify my questions whenever I knocked on your door, for advising me and encouraging me to continue when I was ready to give up. Thank you for your comments on my thesis, research and feedback on talks.

Thank you to my other committee members, Professors Bella Bose, Bruce D'Ambrosio and Paul Adams, for their comments on my thesis and for fruitful discussions. Professor Bose was my temporary advisor when I came here with a fresh B.S. and he warned me not to pursue a Ph.D. degree unless I really like what I am doing. Little did I know at that time what research was like; no wonder it took a long while to learn, fail, like it, dislike it, and finally finishing it up (though my advisor will disagree, saying that research is never done). This is my advice to prospective Ph.D. students: sooner or later you will regret starting a Ph.D. There will be moments when you are totally alone (though still in the hand of God), when neither your family, friends, colleagues nor advisor will be there to help you. You will fall. But you can always rise.

Thank you to the other faculty in our Computer Science department, my professors over the years who taught me so much: Paul Cull, Michael Quinn, Margaret Burnett and Tim Budd. Thank you to Greg Rothermel, Ron Metoyer and Jon Herlocker for career advice.

Thank you to the people in my research group: Dragos Margineantu, Xin Wang, Bill Langford, Chandra Reddy, Tom Amoth, Eric Chown, Tony Fountain, Wes Pinchot, Dan Forrest, Adam Ashenfelter, Ashit Gandhi, Saket Joshi, Diane Damon, Poncho Wu. Warm thanks to Xin Wang for her kindness, generosity in helping me with programming questions and for special moments snowshoeing, dune buggying. She can be such a joy! Thank you to Dragos Margineantu for greeting me at the airport when I first came to America, for research advice and for his big heart. Thank you to all my colleagues for feedback on my research and nice talks on life issues.

Thank you to my fellow Ph.D. students in Dearborn Hall: Alexey Malishevsky, Scott Burgess, Bader AlMohammad, Paul Oprisan, Mohammad Borujerdi, Laura Beckwith and Doug Chow. Special thanks to Alexey and Paul for help with formatting this thesis.

Thank you to Jalal Haddad, our computer support person, for never tiring of my problems.

Thank you to the people in our Computer Science office, always helpful and cheerful: Bernie Feyerherm, Nancy Brown, Janice Abele and Brian Lindsley.

Thank you to our janitor/custodian, Eric Wukovitz, who brought a ray of humanity to my abstract mental life during many evenings.

Thank you to the graduate students in the Physics department with whom we shared many Thanksgivings and picnics. Thank you to Dr. William Hetherington, Dr. Tomasz Giebultowicz and Tim Taylor, in the Physics department, with whom we had many interesting conversations.

Seven years in grad school! Thank you Oregon State University for many cultural events, talks, presentations and conferences on larger issues, thank you for a beautiful campus!

Thank you to these fellow researchers who were very generous with their time, scientific expertise and personal advice: Michael Littman, Tony Cassandra, Peter Turney, Eric Hansen, Milos Hauskrecht, Ron Parr, Pedro Domingos, Omid Madani, Blai Bonet, Andrew Ng, Judy Goldsmith and Rich Sutton. Thank you to Amy Greenwald for recommending me for the "Research, Careers, and Computer Science: A Maryland Symposium," where I benefited from comments on my recent research from Dana Nau, Daniel Jimenez and Kiri Wagstaff. Thank you to Manuela Veloso who advised me not to let people tell me how good my paper is. Thank you for the feedback I received on earlier stages of my research on cost observable MDPs at the AAAI Doctoral Consortium from Raymond Mooney, Janyce Wieb and Mary Harper. Thank you to David Brogan for his career advice during a DARPA workshop.

Thank you to the friends I made at the AAAI, ICML and UAI conferences: Michail Lagoudakis, Pascal Poupart, Ion Muslea, Adrian Silvescu and Doina Caragea, Marina Meila, Doina Precup and Stephen Majercik. Thank you to Prof. Judea Pearl for his cheerfulness at a UAI banquet; hope it stays with you.

Thank you to the National Science Foundation and the Air Force Office of Scientific Research for supporting my research. I want to acknowledge grants IRI-9626584, F49620-98-1-0375 and IIS-0083292.

Thank you to my physical therapists, Cari Gleason, Rich Lague, Guido Van Ryssegem, Pamela Hough and Mike Joki, to ergonometrist Bryan McCampbell and to Ron Stewart, without whom I would not have been able to type this thesis. Thank you to my doctors LaDonna Johnson and William Ferguson. Dr. Ferguson was right, the body heals itself from repeated motion injuries, though it may take a whole year. To whoever reads this, please make sure you take frequent breaks from the computer and correct your posture! Thank you Janet Beary, Sandy Baer and J'aime Phelps, and thank you Cari, Rich and LaDonna for your care and help over all these years.

To our Romanian community in which I spent nice hours laughing, talking and eating, thank you for the memories of picnics, New Year's Celebrations, Christmas and Easter, weddings, birthdays and funerals. Thank you to Monica and Dan Onu, George and Cleo Hirsovescu, Costel Vasiliu, Gabi Zapodeanu, Dragos and Daciana Margineantu, Andra and Nicu Vulpanovici, Telu and Mihaela Popescu, Lucian Chis, Angie and Catalin Doneanu, Veronica and Iulian Mart, Paul Oprisan, Dorina and Adrian Avram, Gabriela and Dan Chirica, Peter and Erika Kiss, Ovidiu Iancu, Corina Constantinescu, Diana Luca, Silvia Stan, Corina Anghel, Cristi Dogaru, Cornelia, Tony and Beatrice Pintea.

Thank you to Nicu Vulpanovici who passed away before I could get to know him better and appreciate him more, whom I loved like the brother I never had.

Thank you to my professors and teachers in Romania: Octavian Stanasila, Maria Honciuc, Ion Constantin, Valentin Matrosenco, Rodica Munteanu, Georgeta Paunescu, Anisoara Paun, Silviu Calinescu and Cristian Giumale.

Thank you to my Romanian friends who did not forget me over these years: Corina Cadariu, Nicoleta Pomojnicu (now Ruckensteiner), Raluca Capatina-Rata, Miki Dabu (now Morgan), Anca Duica, Alex Moldovanu, Cosmin Deciu, Dana Dragomirescu, Bogdan Panghe, Laura Stan, Oana Popescu, Simona Popescu (now Condurateanu), Dan Sevcenco, Costin "Coco" Mihaila and Jeanina Giurea.

Thank you to my international friends: Asami Onada, Toshiko Noda (now Sesselmann), Elena Camarillo Banuelos (now Westbrook), Szilvia Pelikan, Trina Siebert, Dave Stephen and Jill Anthony, Julie and Erwin Schutfort, Roger Klein, Emily Townsend (a most gentle soul), Kyle Winkler and to my host family Judy and Roland deSzoeke.

Thank you to Father Stephen Soot and Mona, and to the people in our beautiful St. Anne Orthodox Church, and to the kids.

Thank you to the authors of several of my favorite books, The Lord of the rings, The way of a pilgrim, To kill a mockingbird. Thank you Dostoyevski. Thank you Ioan Dan for your wonderful adventure books, that still make me laugh even if I know them by heart.

To the people of my immediate family, my grandparents Florina, Virgil and Magdalena, uncles Rica and Corny, and aunt "tusica" Ana, to all my relatives, friends and professors that I knew and loved and are no more, my heart and prayers are always with you.

Thank you to grandfather "tataie" Constantin, uncle Marian and aunt Anisoara, cousin Mariana and Tatiana, "verita" Ana, uncle Vergica, aunts "tanti" Ana and Mariana, uncle Aurel, and to my family-in-law Cathy and Brian, aunts Alice, Catherine and Rita.

Thank you to our godparents, Daniel and Viorica Adam, for their love and generosity.

Thank you to Anna Zubek, my mother-in-law, for her love and gentle nature.

To my husband, Edward Zubek, thank you for the ups and downs and for putting up with me. Ed, thank you for your forgiveness. Thank you for sharing ideas and stories, for teaching me about the environment, for being there most of the time to share laughter or dry a tear. Thank you for your patience and kindness, and for your love and care. Thank you for sleeping under the stars, for swimming in the lakes, for holding my hand on narrow traverses and when crossing glacier streams, for enjoying the pure beauty of wildflowers and snow capped mountains. Without you, I would not have been there. Without you, I would not be here. Thank you for help with typing and editing. Open your heart and fear not, for with love everything is possible.

To my parents, Constantina and Mihai Bayer, I love you and I missed being away from you all these years. Thank you for your love and sacrifices and for encouraging me to finish my Ph.D. Thank you for your patience and help while I prepared the final manuscript of this thesis. You were the motivating force behind me at all times. Thank you for giving me everything. I love you. Multumesc mami si tati pentru ca ati suportat si nu v-ati plins niciodata de faptul ca am plecat la studii de doctorat si am trait departe unii de altii sapte ani. Multumesc pentru dragostea voastra, pentru cum m-ati crescut sa fiu un om. Multumesc pentru ca m-ati sprijinit in orice moment, si pentru toate momentele petrecute impreuna. Ma gindesc la voi mereu si imi pare rau de supararile produse. America poate sa ma fi schimbat in mai toate privintele, dar nu in dragostea pentru voi. Fara incurajarea voastra, n-as fi terminat aceasta teza. Multumesc pentru ajutorul si rabdarea voastra in timpul prepararii manuscrisului final al tezei. Va multumesc pentru tot ce sinteti si faceti pentru mine. Va multumesc pentru ca mi-ati dat tot ce aveti. Va iubesc.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF TABLES

LIST OF TABLES (Continued)

LIST OF TABLES (Continued)

LIST OF TABLES (Continued)

LIST OF TABLES (Continued)

LIST OF APPENDIX FIGURES

LIST OF APPENDIX TABLES

DEDICATION

Sfintei Treimi, pentru tot.

Parintilor mei, pentru ca sint intotdeauna cu mine.

Sotului meu, pentru calatoria noastra impreuna.

To the Holy Trinity, for everything.

To my parents, for always being with me.

To my husband, for our journey together.

Learning Cost-Sensitive Diagnostic Policies from Data

CHAPTER 1

INTRODUCTION

We confront uncertainty every day. To decide how to act, we envision different outcomes of our actions and we plan ahead from each contingency, assessing and weighing the risks and benefits of different courses of action. Sequential decision problems involve decision making in uncertain environments, sensing, and reasoning about utilities. We encounter examples of this sequential decision making process when we make travel, business, or future career plans.

The goal of this dissertation is to develop practical and near-optimal algorithms for learning (from training examples) how to solve cost-sensitive classification tasks when there are costs for measuring each attribute and costs for making misclassification errors. This is important in many domains, from medicine to automotive troubleshooting to computer fault detection and diagnosis.

Traditionally, machine learning [47, 72] has built classifiers to minimize the expected number of errors (also known as the 0/1 loss). This overlooks the importance of estimating the cost of the classification process. For example, to diagnose a cold most accurately, a CT scan can be performed, but such an expensive test will never be ordered in practice for such a minor condition. Some classification errors may be more expensive than others (for example, declaring a sick patient to be healthy can be more expensive than declaring a healthy patient to be sick). The performance of a 0/1 loss classifier, or, more generally, of a classifier that seeks to minimize either expected test costs or expected misclassification costs but ignores the others, will inherently suffer when evaluated with both types of costs.

This dissertation argues for the need to consider both kinds of costs together, because it is not rational to minimize one without the other. There are two types of actions in our framework: (a) *measurement actions* (also called measurements, diagnostic tests, or simply *tests*) and (b) *classification actions* (or diagnoses). Both have costs: *measurement costs* and *misclassification costs.* Measurement actions are purely observational, they do not change the true class. They are needed to get more information, to reduce uncertainty, to guide us closer to the true class that we aim to discover.

Let us consider the task of diagnosing diabetes. The diagnostician performs a sequence of tests, gathers their results, and then makes a diagnosis (in real life, she will also prescribe a treatment, but we do not consider this step). She may ask a series of questions (such as the patient's age, health history, family history of medical conditions), perform simple measurements (measure body mass index, blood pressure) and order lab tests (glucose, insulin). Each measurement has an associated cost — some are cheaper (i.e., measuring the weight and calculating the body mass index), and some are more expensive (i.e, the blood tests). The diagnostician analyzes the results of each test selected and decides whether there is enough information to make a diagnosis or whether more tests are needed. When making a diagnosis, she must take into account the likelihood of each disease and the costs of the misdiagnoses. For example, diagnosing a diabetic patient as healthy can incur costs (such as the cost of aggravating the patient's medical condition); diagnosing a healthy patient as having diabetes can also incur costs (such as the cost of unnecessary treatments).

A *diagnostic policy* (simply, a *policy*) specifies what test to perform next, based on the outcomes of the previous tests, and when to stop (by choosing to classify). A diagnostic policy takes the form of a decision tree whose nodes specify tests and whose leaves specify classification actions. The interesting decision problem is how to choose among different policies. Shall we classify now, shall we perform a single high-cost test, or shall we perform sequences of cheap tests before classifying?

Thesis Objective

We define *cost-sensitive learning* as the problem of learning diagnostic policies minimizing expected total costs, given as inputs a training set of labeled examples, the measurement costs, and the misclassification costs. This dissertation is about learning, not planning, and learning is done from data. Each example records the results of all tests and has an associated class (diagnosis). An example can be seen as a set of attribute values, "*attribute*" being used interchangeably for measurement/test. We assume there are *no missing attribute values.* The training data is used to compute probabilities of measurements' outcomes and probabilities of classes conditioned on those outcomes.

It is interesting to note that the problem of learning good diagnostic policies is difficult only when the cost of testing is comparable to the cost of misclassification. If the tests are very cheap compared to misclassification costs, then it is optimal (or close to optimal) to measure all of them to gain as much information as possible. If the tests are very expensive compared to misclassification costs, then it is optimal (or close to optimal) to classify directly without measuring anything.

We will evaluate diagnostic policies by their expected total costs. This is a widely accepted measure, but it is not the only possible one. In some settings, worst-case or best-case cost might make more sense. In other settings, there may be multiple evaluation criteria that cannot be summarized as a single measure of cost. For example, when planning a trip from Portland, Oregon to Glacier National Park, you may want to minimize the expense, maximize the enjoyment, minimize environmental impact, and minimize transit time. There are tradeoffs among these measures, and it may be difficult or impossible to summarize them in a single cost measure.

This thesis defines and studies two families of algorithms for learning diagnostic policies.

The first family consists of greedy methods closely related to standard top-down decision tree algorithms ([9, 63]). These methods construct the diagnostic policy top-down by selecting attributes to measure based on a one-step lookahead search, with some costs involved.

The second family of algorithms consists of new methods based on the AO* algorithm for systematic search of AND/OR graphs. AO* computes the optimal policy on the training data. It performs a systematic search of the space of all diagnostic policies, but avoids exhaustive search by pruning parts of the search space that provably cannot contain the optimal policy. Systematic search of the space of diagnostic policies has previously been regarded as computationally infeasible, because the search space is super-exponential in size.

This dissertation explores several ways of making systematic search feasible while learning good policies. First, we note that the size of the training data has a profound effect on the size of the AO* search graph. When training on data sets of ordinary sizes (less than 10,000 examples), the AO* graph is much smaller than it would be for exhaustive search, because many sequences of test outcomes have zero probability, since they do not appear in the training data. Second, we introduce an admissible heuristic that allows AO* to prune large parts of the search space. Third, we introduce several modifications of AO* to reduce the risk of overfitting during learning. These modifications are called "regularizers." Some of these have the side-effect of further pruning the search space. Finally, we explore a form of "statistical pruning" that deletes branches from the search space that are unlikely to lead to good solutions.

The dissertation presents experimental evidence to support the following theses:

Thesis 1: Systematic search (with an appropriate admissible heuristic and regularizers) is computationally feasible for real-world cost-sensitive learning problems.

Thesis 2: A method called SP-L, which combines AO* search, Laplace corrections, and statistical pruning, gives the most robust overall performance when measured on independent test data.

The dissertation is organized as follows. Chapter 2 formalizes the cost-sensitive learning problem addressed in this work and reviews previous work on this and similar problems. Chapters 3 and 4 present, respectively, the greedy and systematic search families of cost-sensitive learning algorithms. Chapter 5 presents a series of experiments that measure the efficiency and effectiveness of the various methods on real-world data sets. Chapter 6 presents the contributions of the thesis and discusses future work extending our cost-sensitive learning framework.

CHAPTER 2

COST-SENSITIVE LEARNING (CSL)

This chapter formally introduces the problem of cost-sensitive learning. We define cost-sensitive learning (CSL) as the problem of learning diagnostic policies that minimize the expected total cost of diagnostic tests and classification errors based on a set of training examples. In this formulation, CSL borrows from both supervised learning and Markov Decision Problems. Indeed, as in supervised learning, we want to learn an hypothesis predicting the class of new, unseen examples, from a set of labeled training examples. But our objective function is cost-sensitive, and subject to its minimization, we want to learn in which order to perform the diagnostic tests followed by classification actions. This is a sequential decision problem that can be modeled by a Markov Decision Problem (MDP). Given the costs of all actions (diagnostic tests and classifications) and given a set of training examples from which we can compute the transition probability model, we can define an MDP whose solutions are optimal diagnostic policies. We present exact and approximate methods for solving this MDP in the next two chapters. Here, we briefly describe supervised learning and the MDP framework, then we show how the CSL problem can be formulated as an MDP. The relevant notations and concepts are illustrated on a simple diagnostic task. We discuss restrictions of our framework and possible extensions. Then we review relevant literature.

2.1 Supervised Learning

An application of supervised learning is the PAPNET technology, a computer-assisted Pap smear test that classifies cells as normal or precancerous (for this FDA-approved PAPNET Testing System, Neuromedical Systems, INC. holds two U.S. patents; see http://www.fda.gov/fdac/features/896_pap.html and http://www.hknet.com/Papnet). The system (a neural network) was trained on images of cervical cells having different shapes and colors. The images were manually labeled as benign or malignant. The neural network learns patterns for identifying each type of cell; for example, large and misshapen nuclei can signify cancer. After training, the network is used to quickly scan new slides and rank the top 128 most abnormal cells. These are then presented to a human for evaluation. Traditionally, humans scanning the cells under a microscope miss from $10\% - 30\%$ of abnormal cases. The automation of the test reduced human fatigue by eliminating 98% of the work, and it increased the detection of abnormalities up to 30%.

This is an instance of supervised learning. Formally, given a sample of labeled examples (x, y) drawn from a distribution $D(x, y)$, where x is a vector of *attributes* and y is its label, the task is to learn a *hypothesis* h that labels x with the most likely class. The predicted class $h(x)$ is denoted by $\hat{y}$. The distribution $D(x, y)$ from which the labeled examples are drawn can be factored into two probability distributions, a class probability $P(y)$ and a conditional probability $P(x|y)$.

The attributes can be symbolic or numeric (discrete or continuous). The labels can be discrete (in which case the task is called *classification*, and the labels are called *classes*) or continuous (in which case the task is called *regression*, for example, predicting the temperature in a furnace).

Our CSL framework assumes the attributes are numeric and the labels are discrete, so it focuses on classification tasks. The labels are called the observed classes.

In traditional classification tasks, the goal is to find an hypothesis h that minimizes the expected number of *misclassification* errors, that is, the expected number of examples incorrectly classified:

$$\min E_{(x,y)\sim D}[L(h(x), y)] = \min \sum_{(x,y)} D(x, y) L(h(x), y), \tag{2.1}$$

where the loss function $L(h(x), y)$ is 1 when $h(x) \neq y$ and 0 otherwise. These classifiers are also known as minimizing the expected 0/1 loss, and their underlying assumption is that misclassification errors have the same cost, and, in addition, no attention is

TABLE 2.1: Notations for the examples' predictions versus their observed classes. The class $y = 1$ is interpreted as having the disease, and $y = 0$ as not having it.

	observed class	
	$y = 0$	$y = 1$
$\hat{y} = 0$	true negatives	false negatives
$\hat{y} = 1$	false positives	true positives

paid to attribute costs.

Cost-sensitive learning is an extension of the classification task of supervised learning. First, it takes into account different costs for misclassification errors. It also considers attribute costs because it realizes that there is a cost associated with obtaining each attribute value. The objective of cost-sensitive learning is to minimize the expected total cost.

In supervised learning, for the case when there are only two classes (0 and 1), we call a *positive* example one whose class is $y = 1$ and a *negative* example one whose class is $y = 0$. It helps to think of this in terms of medical diagnosis. A patient is "diabetes positive" if he has the disease ($y = 1$) and "diabetes negative" if he does not ($y = 0$). The classes assigned by the hypothesis can be correct or not, so we talk about "true positives" and "true negatives", when the predictions match the observed classes, and "false positives" and "false negatives" when they do not; see Table 2.1. Let $\hat{y} = h(x)$ be the class predicted by hypothesis h. Then a false positive is an example (x, y) where $\hat{y} = 1$ and $y = 0$ (a healthy patient was diagnosed with diabetes). Similarly, a *false negative* example was assigned class $\hat{y} = 0$ when in fact its observed class is $y = 1$ (a sick patient was diagnosed to be healthy).

After an hypothesis is learned, we want to see how good it is at predicting the classification of new, unseen examples. The strategy is to divide the data into two sets, a *training set* and a *test set.* Learning is done on the training set; then the hypothesis is evaluated on the test set. Because we know the observed labels of the test examples, we can compare them to the predicted labels of the hypothesis and compute the number of errors (false positives and false negatives).

One of the most serious problems facing learning from training data is *overfitting.* Overfitting is picking up noise or regularities from the training data that are not characteristic of the entire data population. In the extreme case, the hypothesis can memorize perfectly the training data, but it has a reduced power of generalization on new examples, so it is almost useless. Sometimes it is better (in terms of generalization power, i.e., performance on the test set), not to be perfect on the training data. Quoting Professor Thomas Dietterich, "sometimes it is optimal to be suboptimal." We postpone discussion of overfitting until Chapter 5, but we wanted to mention that the CSL problem is not immune to it, since it learns from data.

2.2 Markov Decision Problems (MDPs)

Cost-sensitive learning requires learning (optimal) diagnostic policies, which are sequences of decisions made under uncertainty, each decision depending on previous decisions and their outcomes. The notations of the Markov Decision Problem will be very useful in formalizing the notions of diagnostic policies and the objective function of minimizing expected total costs.

An MDP is a mathematical model for describing the interaction of an *agent* (learner and decision maker) with an *environment.* At each step the agent perceives the state of the world s_t, based on which it chooses an *action* a_t and receives a reward r_{t+1} (or pays a *cost* c_{t+1}). Then it perceives the resulting state s_{t+1}, chooses another action a_{t+1}, and the cycle repeats. The action a_t is selected from the set of actions available in state s_t, $a_t \in A(s_t)$. The agent's goal is to choose its actions such as to maximize the long term rewards from the environment. In terms of costs, this translates into minimizing expected total costs, since costs can be seen as negative rewards. We will present the MDP model in terms of costs (though traditionally reward functions are employed).

If the state representation contains all the relevant information for future decisions, then it is said to have the *Markov property.* In this case, the probability distributions of the next state s_{t+1} and cost c_{t+1} only depend on the current state s_t and action a_t, and not on the history of past states, actions, and costs:

$$P(s_{t+1}, c_{t+1}|s_t, a_t, c_t, s_{t-1}, a_{t-1}, \ldots, c_1, s_0, a_0) = P(s_{t+1}, c_{t+1}|s_t, a_t).$$

Formally, a (discrete) Markov Decision Problem ([60]) is a tuple

$$\langle S_0, S, A, P_{tr}(S|S, A), C(S, A, S)\rangle,$$

where S_0 is the set of initial world states, S is the set of all world states, A is the set of actions, $P_{tr}(s_{t+1}|s_t, a_t)$ is the *transition probability* of moving to state s_{t+1} at time $t+1$, after performing action a_t in state s_t at time t, and $C(s_t, a_t, s_{t+1})$ is the expected immediate cost for performing action a_t in s_t and making a transition to s_{t+1}. We assume the *state and action sets are finite.* There is no discount factor, because we are interested in *episodic* tasks (i.e., tasks that terminate after a finite number of actions have been executed).

An action can have a *deterministic* or a *stochastic* effect. If it is deterministic, from a state s_t the transition is to a single next state with probability 1. If it is stochastic, there may be more than one possible resulting state. We write $P_{tr}(s_{t+1}|s_t, a_t)$ for the probability of moving to state s_{t+1} at time $t+1$, after performing action a_t in state s_t at time t. Note that $\sum_{s_{t+1} \in S} P_{tr}(s_{t+1}|s_t, a_t) = 1$ (so this is a probability distribution over the next states).

The *cost function* $C(s_t, a_t, s_{t+1})$ is the expected cost associated with the transition from s_t to s_{t+1}, after performing action a_t. The expectation is with respect to unknowns in the environment, $C(s_t, a_t, s_{t+1}) = E\{c_{t+1}|s_t, a_t, s_{t+1}\}$.

A (deterministic) *policy* is a mapping from states to actions $\pi : S \to A$, and it chooses an action to take in each state. The *value* of a state s under a fixed policy π, $V^\pi(s)$, is the expected sum of future costs incurred when starting in state s and following π afterwards ([66], chapter 3):

$$V^\pi(s) = E_\pi \left\{ \sum_k c_{t+k+1} \middle| s_t = s \right\}.$$

The expectation is taken with respect to the randomness in the effects of the actions and in the cost function. For episodic tasks, a *terminal state* s_f is eventually reached after a finite number of actions, and $V^\pi(s_f) = 0, \forall \pi$.

The value function V^π of a policy π satisfies the following recursive relationship, known as the *Bellman equation for* V^π:

$$V^\pi(s) = \sum_{s' \in S} P_{tr}(s'|s, \pi(s)) \times \left[C(s, \pi(s), s') + V^\pi(s')\right], \forall \pi, \forall s. \tag{2.2}$$

This can be viewed as a *one-step lookahead* from state s to each of the next states s' reached after executing $\pi(s)$. Given a policy π, the value of state s can be computed from the value of its successor states, by adding the expected costs of the transitions, then weighting them by the transition probabilities.

In addition to V^π, it is useful to define the action value function Q^π as follows:

$$Q^\pi(s, a) = \sum_{s' \in S} P_{tr}(s'|s, a) \times \left[C(s, a, s') + V^\pi(s')\right].$$

Note that $V^\pi(s) = Q^\pi(s, \pi(s))$.

Solving the MDP means finding a policy that minimizes the expected sum of costs, in other words, finding a policy with minimum value function. Such a policy is called an *optimal policy* (there can be several of them), and it chooses the best action in each state. Policies can be partially ordered according to their values. A policy π_1 is better than π_2 if its policy value is smaller: $V^{\pi_1}(s) \leq V^{\pi_2}(s), \forall s$. An optimal policy π^* has the minimum value function in every state, so it is better than or equal to all other policies. All optimal policies share the optimal value function V^*, with $V^*(s) = \min_\pi V^\pi(s), \forall s$, and the same optimal Q-function Q^*, with $Q^*(s, a) = \min_\pi Q^\pi(s, a), \forall (s, a)$.

The *optimal value function* V^*, sometimes called the value function of the MDP, satisfies the *Bellman optimality equations* (one equation for every state):

$$V^*(s) = \min_a \sum_{s' \in S} P_{tr}(s'|s, a) \times \left[C(s, a, s') + V^*(s')\right], \forall s. \tag{2.3}$$

Similarly, the Bellman optimality equations for Q^* are (one equation for every state-action pair):

$$\begin{aligned} Q^*(s,a) &= \sum_{s' \in S} P_{tr}(s'|s,a) \times \left[C(s,a,s') + V^*(s')\right] \\ &= \sum_{s' \in S} P_{tr}(s'|s,a) \times \left[C(s,a,s') + \min_{a'} Q^*(s',a')\right], \forall s, \forall a. \end{aligned} \tag{2.4}$$

$Q^*(s,a)$ is the value of executing action a in state s, followed by executing the optimal policy in the resulting states s'.

We also write $V^*(s) = \min_a Q^*(s,a)$. If we know the optimal value function V^*, the probability transition model and the cost function, then any policy that is greedy with respect to V^* is an optimal policy:

$$\pi^*(s) = \arg\min_a \sum_{s' \in S} P_{tr}(s'|s,a) \times \left[C(s,a,s') + V^*(s')\right].$$

If the Q^* function is known, then we can compute the optimal policy directly, without doing the one-step lookahead: $\pi^*(s) = \arg\min_a Q^*(s,a)$.

Value iteration is an algorithm that solves MDPs by iteratively computing their value functions. Value iteration belongs to a class of methods, called *dynamic programming* methods, that compute V^* by solving the Bellman optimality equations 2.3 or 2.4. These methods require a model of the transition probabilities and of the cost function. In value iteration, at iteration $i+1$, the value function is updated using a one-step lookahead based on the value function computed in the previous iteration i,

$$V_{i+1}(s) := \min_a \sum_{s' \in S} P_{tr}(s'|s,a) \times \left[C(s,a,s') + V_i(s')\right]. \tag{2.5}$$

We also say that the value function has been *backed up*. The value function may be initialized with arbitrary values (though it is easier to understand the update process if the initial values are zero). For finite MDPs in which the terminal state is reached after a finite number of steps, the value iteration algorithm converges to the optimal value function V^*.

Value iteration can also be interpreted as turning the Bellman optimality equation 2.3 into an update rule. Based on the value function V_i, the policy can also be updated $\pi_{i+1}(s) := \arg\min_a \sum_{s' \in S} P_{tr}(s'|s,a) \times \left[C(s,a,s') + V_i(s')\right]$.

An MDP is said to be *acyclic* if starting from the initial state(s) the state transitions form a DAG ending in the terminal state. In an acyclic MDP, value iteration can compute the optimal value function V^* in a single *sweep* through the state space, backing up values starting from the terminal state (which has zero value) and proceeding through the DAG all the way to the initial state(s):

$$V(s) := \min_a \sum_{s' \in S} P_{tr}(s'|s,a) \times \left[C(s,a,s') + V(s')\right] . \tag{2.6}$$

2.3 Formal Description of the Cost-sensitive Learning Problem as an (Acyclic) MDP

The decision making process in cost-sensitive learning involves performing diagnostic tests (observation actions), gathering their results, based on which more tests will be chosen, with the purpose of gathering enough information about the class of the example (patient) to be able to identify it.

Cost-sensitive learning combines aspects of both supervised learning and MDPs. Like supervised learning, the data consist of examples of the form (x, y). But as in MDPs, the goal is to learn a decision-making policy for making a sequence of decisions that minimizes the expected total cost. We will use terms and notations from both fields. For example, we will call the hypothesis a policy, a term from MDPs, though all policies take the form of decision trees (a structure frequently used in supervised learning). We will use measurements, actions, tests, and attributes interchangeably, and we will also use classification and diagnosis as synonyms.

Informally, given a training set of labeled examples where *all attributes are measured*, and given attribute costs and misclassification costs, the goal of cost-sensitive learning is to learn a policy with low expected costs on new examples.

The problem of cost-sensitive learning can be represented as a Markov Decision Problem (MDP). We assume that the class we want to identify (belonging to a discrete set of classes $\{1, 2, \ldots, K\}$) is part of the environment and that the policy cannot modify it; that is, all the tests are *pure observations*. In other words, none of the measurement actions of the CSL problem can make the patient sick nor cure the patient. This is not true, for example, in automobile diagnosis where the policy may include repair actions (e.g., replace spark plugs), nor in medicine if the policy is permitted to include attempted therapies (e.g., try an injection of steroids and observe the results). We also assume that the order in which we perform the measurements does not influence the values of other measurements, though it may affect the value of a policy. Our CSL formulation only requires the identification of the class, subject to minimizing the expected total cost, assuming the testing process has no side effects.

We begin by defining the actions A of this MDP. We assume that there are N measurement actions (tests) and K classification actions. Measurement action n (denoted x_n) returns the value of attribute x_n, which we assume is a discrete variable with possible values $v_1, ..., v_{V_n}$. Classification action k (denoted f_k) is the act of classifying the example (patient) into class $y = k$.

We emphasize that *in this thesis, x_n denotes a variable, not a value. x_n is the attribute and v is its value.*

Now let us define the states S of the MDP. A state is a complete history of all past observations, and because it retains all the relevant information for future decisions, this state representation has the Markov property. We emphasize again that the observed class y is not part of the state representation. There is a single initial world state, s_0, also called the start state, so $S_0 = \{s_0\}$. In the start state, no attributes have been measured (so we write $s_0 = \{\}$). The set of all world states S contains one state for each possible combination of measured attributes, as found in the training data. For example, the state {age = old, insulin = low} is a state in which the "age" attribute has been measured to have the value "old" and the "insulin" attribute has been measured to have the value "low". We assume that *once an attribute is measured, it cannot be tested again* (since all attributes are measured in the training data, by measuring the attribute we obtain its value and there is no reason to measure it again to get the same value). The set $A(s)$ of actions executable in state s consists of those attributes not yet measured and all the classification actions.

There is also a special terminal state s_f. Every classification action makes a transition to s_f with probability 1 (i.e., once a classification is made, the task terminates). By definition, no actions are executable in the terminal state, $A(s_f) = \{\}$, and its value function is zero, $V^{\pi}(s_f) = 0, \forall \pi$.

If all attributes x_n have the same arity, $V_n = V, \forall n$, then the total number of states of the MDP is $(V+1)^N + 1$, because an attribute can be either measured (so it will have one of the V values) or not yet measured, plus we have the terminal state. But on a given training set, not all of these states may be *reachable*, because not all combinations of attribute values may be observed in the training examples. For N attributes and m training examples, the upper bound on the number of reachable states is $2^N \cdot m$.

We now define the transition probabilities and the expected immediate costs of the MDP. For measurement action x_n executed in state s, the result state s' will be $s' = s \cup \{x_n = v\}$, where v is one of the possible values of x_n. The probability of this transition will depend on the values of all of the previously-measured attributes, which are stored in state s: $P_{tr}(s'|s, x_n) = P(x_n = v|s)$. The expected cost of this transition is $C(s, x_n, s')$. We assume that the cost of measurement action x_n depends only on the action itself and not on which other measurements have already been performed (so it does not depend on s), nor on its measured value v (so it does not depend on s'), and we also assume that it does not depend on the observed class y. We believe these assumptions can be relaxed without substantial changes to our framework.

The expected cost of a classification action f_k depends on the observed class y of the example. Let $MC(f_k, y)$ be the misclassification cost of guessing class k when the observed class is y. Because the observed class y of an example is not known to the learner, the cost of a classification action (which depends on y) performed in state s must be viewed as a random variable whose value is $MC(f_k, y)$ with probability $P(y|s)$, which is the probability that the observed class is y given the current state s. Fortunately, to compute the optimal policy for the MDP, we only need the expected cost of each action. The expected cost of classification action f_k in state s is

$$C(s, f_k) = \sum_y P(y|s) \cdot MC(f_k, y), \tag{2.7}$$

which is independent of y. $C(s, f_k)$ is a shorthand notation for $C(s, f_k, s_f)$, and we omit s_f because classification actions transition to the terminal state s_f with probability 1. To keep a consistent notation, we also write $C(s, x_n)$ (instead of $C(s, x_n, s'))$ for the expected cost of measurement action x_n, though with our assumptions it could just be written as $C(x_n)$. This allows us to write the expected immediate cost as $C(s, a)$ where the action a can be either a measurement or a classification action.

We will say that an example *matches* a state s if the example agrees with the attribute values defining s.

Given a training set of labeled examples with no missing attribute values, we can directly estimate the MDP's transition probabilities from the training set. $P_{tr}(s'|s, x_n)$ is estimated as the number of training examples that match state s' (where $x_n = v$) divided by the number of training examples that match s. We can also estimate the class probabilities $P(y|s)$ needed to compute the expected costs, $C(s, f_k)$, of the classification actions. $P(y|s)$ is the fraction of training examples matching state s that belong to class y.

A policy π for an MDP is a mapping from states to actions. Note that in the cost-sensitive learning problem, all policies are *proper*, that is, they reach the terminal state with probability 1. The terminal state is always reached, because only finitely-many measurement actions can be executed after which any classification action will cause the MDP to enter the terminal state.

Because the states of our MDP record the values of measured attributes, and because each attribute can only be measured once, the MDP for CSL problems is acyclic. For a given start state s_0, the CSL policy is a *decision tree*. The root of the tree is the start state of the MDP. Each internal node in the tree corresponds to a state that is reached by π in the MDP, and the attribute tested in the node is the action chosen by π. The branches descending from such a node specify each of the values of the attribute, and lead to new nodes. The classification labels in the leaves of the tree are the classification actions of the MDP. All diagnostic policies are decision trees, so the hypothesis space of decision trees contains the optimal policy π^*.

Since building decision trees that minimize the 0/1 loss is NP-complete [29], and this problem can be reduced to our cost-sensitive learning problem (by having all attribute costs be zero, and having all error costs be one) it follows that the CSL problem is also NP-complete.

The value function of a policy, $V^\pi(s)$, is the expected total cost of following policy π starting in state s until the terminal state is reached. If we knew the true underlying distribution $D(x, y)$, we could compute the true value function V^π. But we only have a data sample from $D(x, y)$, which we divide into a training data set and a test data set. The training data set is used by the learning algorithm to construct the policy, so using it to evaluate the policy gives a biased estimate of the true value function. The independently chosen test data set gives an unbiased estimate of V^π. Nevertheless, both training and test data sets can only produce estimates of the true value function V^π.

Any policy π we learn will start in the initial state s_0. In general, by *the policy value* we mean the value of the policy in the start state, $V^\pi(s_0)$. The goal of CSL is to learn policies that minimize $V^\pi(s_0)$ on the test data (actually on the true distribution, but we do not know it at learning time).

The optimal policy π^* (on the training data) minimizes $V^\pi(s)$ for all states, and its optimal value is $V^*(s)$. With our CSL notations, we can rewrite equations 2.2 and 2.3 as follows. The Bellman equation for policy values becomes

$$V^\pi(s) = C(s, \pi(s)) + \sum_{s' \in S} P_{tr}(s'|s, \pi(s)) \times V^\pi(s'), \tag{2.8}$$

and the Bellman optimality equation becomes

$$V^*(s) = \min_{a \in A(s)} \left[C(s, a) + \sum_{s' \in S} P_{tr}(s'|s, a) \times V^*(s') \right] = \min_{a \in A(s)} Q^*(s, a). \tag{2.9}$$

Equation 2.8 can be further rewritten, depending on whether $\pi(s)$ is a measurement action x_n with possible values v, or if it is a classification action f_k. In the former case,

$$V^\pi(s) = Q^\pi(s, x_n) = C(s, x_n) + \sum_v P(x_n = v|s) \times V^\pi(s \cup \{x_n = v\}). \tag{2.10}$$

In the latter case,

$$V^\pi(s) = Q^\pi(s, f_k) = C(s, f_k), \tag{2.11}$$

because $V^\pi(s_f) = 0$ by definition.

During learning and evaluation, we will need to estimate $V^\pi(s_0)$ from a set of labeled examples. There are two methods for doing this. One is to use the labeled examples to estimate various probabilities which can then be plugged into equations 2.10 and 2.11. Specifically, we need to estimate $P(y|s)$ for each state s where a classification action is chosen. This allows us to compute $C(s, f_k)$ according to 2.7. And we need to estimate $P(x_n = v|s)$ for each state reached by π and for the attribute $x_n = \pi(s)$ chosen in state s.

The second method for computing $V^\pi(s_0)$ is to take advantage of the fact that the policy π has the form of a decision tree. This means that each example, when classified by the tree, will follow a path from the root to one of the leaves. To compute $V^\pi(s_0)$, do the following: for each example in the data set, compute the cost of measuring each attribute tested along the path followed by this example. Then compare the classification action f_k in the leaf reached by this example to the observed class y and add in the misclassification cost $MC(f_k, y)$. This gives the total cost of classifying that example. Average this over all of the examples to obtain the estimated value of the policy on the given data set.

There can be examples with the same attribute values but different classes; this reflects a reality in medicine, for example, where based on the results of a limited number of tests, a sick patient and a healthy patient may appear the same. Even if the observed class label of an example is wrong, our policy will incur a misclassification cost for not matching it, because we have no way of knowing whether the label is correct.

Because the MDP corresponding to the cost-sensitive learning problem is acyclic, we can employ the single sweep value iteration update 2.6. With our notations,

$$V(s) := \min_{a \in A(s)} \left[C(s,a) + \sum_{s' \in S} P_{tr}(s'|s,a) \times V(s') \right]. \tag{2.12}$$

Alternative Formulation of the Cost-sensitive Learning Problem as a POMDP

In this thesis, the cost-sensitive classification problem is formulated as an MDP where the states are combinations of measured attributes. There is a different formulation of the CSL problem as a POMDP (Partially Observable MDP) with belief states over the possible classes of the problem. The belief states also need to specify the set of remaining actions (because the same class probability distribution $P(y|s)$ can appear after different sets of tests were measured). This is another way of formulating the cost-sensitive learning problem using belief states of the form $\langle P(s), A(s) \rangle$. The underlying MDP for this POMDP is trivial, since its states are simply the classes of the problem, so no measurement action needs to be performed, and $V^*_{MDP}(s) = 0$ (assuming classifying in the correct class has zero cost), and $\pi^*(s = \{y = k)\} = f_k$.

2.4 Example of Diagnostic Policies

Let us illustrate the above notions with a simple example. Suppose the task is to learn how to diagnose diabetes, and we have a dataset of 100 labeled examples with two tests, Body Mass Index (BMI) and Insulin, and two classes, Healthy and Diabetes. The training dataset is described in Table 2.2.

Tables 2.3 and 2.4 illustrate the costs for tests and for misclassifications. BMI has a lower cost, $C(BMI) = 1 < C(Insulin) = 22.78$. The misclassification costs are asymmetric: the false negatives are more expensive than the false positives, $MC(f = Healthy, y = Diabetes) > MC(f = Diabetes, y = Healthy)$. Correct classifications have zero costs.

From the training data set we can compute maximum likelihood estimates of the

TABLE 2.2: Training set of labeled examples for a simple diabetes diagnosis task.

Body Mass Index	Insulin	Class	# examples
small	low	Healthy	15
small	low	Diabetes	2
small	high	Healthy	30
small	high	Diabetes	3
large	low	Healthy	12
large	low	Diabetes	28
large	high	Healthy	8
large	high	Diabetes	2

MDP probabilities, for example $P(BMI = large|\{\}) = .5$, computed as the ratio between the 50 examples matching state $\{BMI = large\}$ and the total number of training examples (100). Similarly, $P(Insulin = high|\{BMI = large\}) = (8 + 2)/50 = 10/50 = .2$. The class probabilities are estimated as the ratio between the number of examples matching a state and having a certain class, and the total number of examples matching the state. For example, $P(y = Healthy|\{BMI = small\}) = (15 + 30)/50 = 45/50 = 0.9$, and $P(y = Healthy|\{BMI = large, Insulin = low\} = 12/(12 + 28) = 12/40 = .3$.

In our notation, $A(s)$ is the set of actions executable in state s: those attributes not measured in s and all the classification actions. For example, $A(\{BMI = small\}) = \{Insulin, f = Healthy, f = Diabetes\}$.

Figure 2.1 illustrates a simple policy π_1. We describe the policy and show how can we compute its value in a single sweep, starting at the leaves. In the start state {}, the BMI (Body Mass Index) attribute is measured. If the result of this test is {BMI = small}, a classification action is made, classifying as Healthy. This is an incorrect

TABLE 2.3: Test costs.

Test	Cost
BMI	\$1.00
Insulin	\$22.78

TABLE 2.4: Misclassification costs.

	observed class y	
	Healthy	Diabetes
predicted class $\hat{y}$ = Healthy	\$0	\$100
predicted class $\hat{y}$ = Diabetes	\$80	\$0

classification 10% of the time, therefore the expected cost of this classification action is $C(\{BMI = small\}, f = Healthy) = .9*0+.1*100 = 10$, according to equation 2.7. This is also the value of the leaf state, $V^{\pi_1}(\{BMI = small\}) = 10$.

If the result of the Body Mass Index test is {BMI = large}, the policy π_1 measures Insulin, after which it classifies. The values of the leaf states are $V^{\pi_1}(\{BMI = large, Insulin = low\}) = C(\{BMI = large, Insulin = low\}, f = Diabetes) = .3 * 80 + .7 * 0 = 24$ and $V^{\pi_1}(\{BMI = large, Insulin = high\}) = C(\{BMI = large, Insulin = high\}, f = Healthy) = .8 * 0 + .2 * 100 = 20$.

The value of the state {BMI = large} is backed up according to equation 2.10, $V^{\pi_1}(\{BMI = large\}) = C(\{BMI = large\}, Insulin) + P(Insulin = low|\{BMI = large\})*V^{\pi_1}(\{BMI = large, Insulin = low\})+P(Insulin = high|\{BMI = large\})* V^{\pi_1}(\{BMI = large, Insulin = high\}) = 22.78 + .8 * 24 + .2 * 20 = 45.98$. Finally, the value of the entire diagnostic policy π_1 is $V^{\pi_1}(\{\}) = C(\{\}, BMI) + P(BMI =$

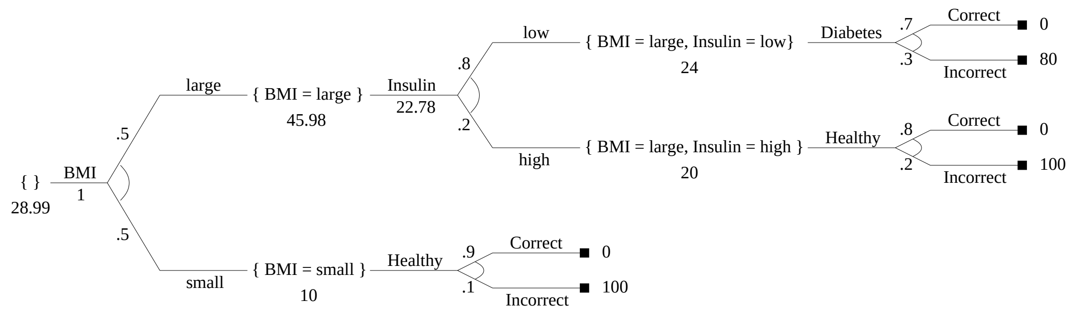


FIGURE 2.1: Diagnostic policy π_1. Internal nodes contain states delimited by $\{,\}$, and specify which attribute (BMI, Insulin) is tested. Leaves are labeled with classification actions (Healthy, Diabetes) and we write the probabilities of that classification being correct and incorrect. We write the costs of attributes underneath them, and the misclassification costs next to the solid squares. The policy values of states are written under the states.

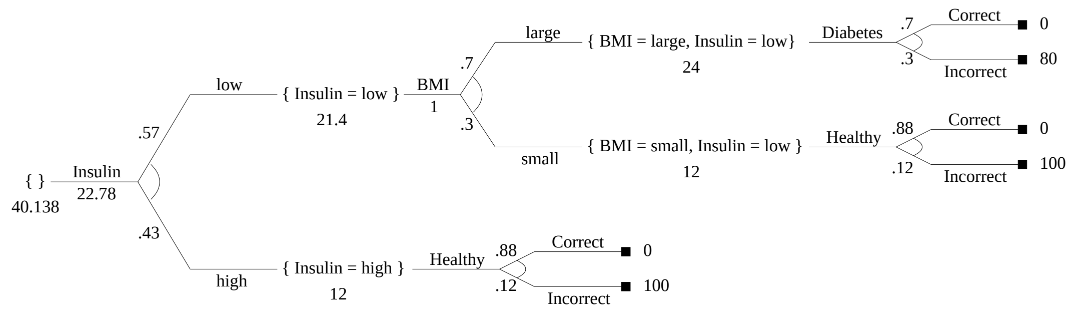


FIGURE 2.2: Another diagnostic policy π_2, making the same classification decisions as π_1, but with a changed order of attributes, and therefore with a different policy value.

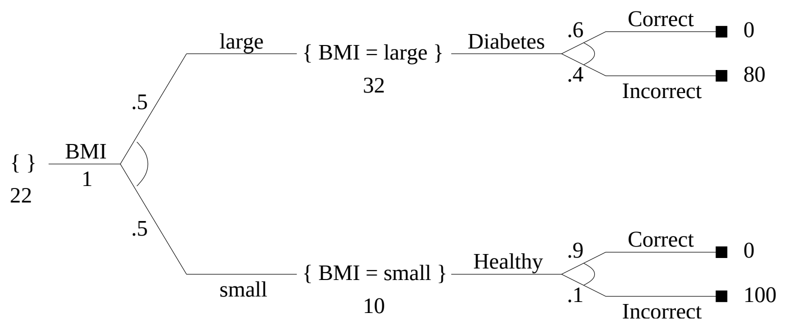


FIGURE 2.3: Optimal diagnostic policy on the training data.

$large|\{\}) * V^{\pi_1}(\{BMI = large\} + P(BMI = small|\{\}) * V^{\pi_1}(\{BMI = small\}) = 1 + 0.5 * 45.98 + 0.5 * 10 = 28.99$.

It is interesting to notice that the order in which attributes are measured has a large effect on the value of the policy. Indeed, if Insulin is measured first, then policy π_2 from Figure 2.2 has a larger policy value $V^{\pi_2}(\{\}) = 40.138 > V^{\pi_1}(\{\}) = 28.99$. Policy π_2 still makes the same classification decisions as π_1 for all the training examples, but it costs much more.

A simpler policy π_3, measuring only BMI and then classifying (see Figure 2.3), has a better policy value than π_1, $V^{\pi_3}(\{\}) = 22$; in fact, this policy is the optimal policy π^* on the training data.

Unlike with standard 0/1 loss, the majority class is not necessarily the best cost-sensitive class. In this problem, for example, $P(y = Healthy|\{\}) = .65$ and $P(y = Diabetes|\{\}) = .35$, so the expected classification costs are $C(\{\}, f = Healthy) = .65 * 0 + .35 * 100 = 35$ and $C(\{\}, f = Diabetes) = .65 * 80 + .35 * 0 = 52$. Therefore the majority class (Healthy) is also the cheapest diagnosis. But if the misclassification costs of false negatives is doubled, that is, if $MC(f = Healthy, y = Diabetes) = 200$, then $C(\{\}, f = Healthy) = .35 * 200 = 70 > C(\{\}, f = Diabetes) = 52$, so the minority class (Diabetes) becomes the cheapest diagnosis.

2.5 Assumptions and Extensions of Our Cost-sensitive Learning Framework

In the following subsections we discuss several important issues related to cost-sensitive learning problems. We introduce first the general setting, then we discuss the assumptions of our CSL framework and its possible extensions.

2.5.1 Complex Attribute Costs and Misclassification Costs

2.5.1.1 Attribute Costs

Using Turney's terminology for *conditional test costs* [70], the measurement cost of attribute x_n may depend on

- prior test selection (e.g., blood tests can share a common cost of collecting the blood).
- the results of prior tests (e.g., drawing blood from a newborn is more costly than from an adult; in this case, the result of a previous test "observe patient age" influences the cost of the next test "draw blood").
- the outcome of x_n (e.g., in computer network diagnosis [41], doing a "ping" to measure the round-trip-time to a host is very fast if the host is reachable but waits 20 seconds for a timeout if the host is down or not reachable).
- the class (e.g, tests can become more expensive for patients in critical medical condition).

In general, measurement costs can depend on the action performed x_n, the current state s (history of prior measurements and their values), the resulting states s' (since $s' = s \cup \{x_n = v\}$, this is a dependency on the outcomes of x_n), and the class of the example y, so the most general form of the cost function is $C(s, x_n, s', y)$. In order to reason with complex test costs, we would first need to acquire them, either from training data or from being told. Note that we cannot learn cost dependencies from our existing, order-free, training data. Once the cost model is known, we can easily incorporate more complex test costs in our framework than the current test costs $C(s, x_n)$.

Since the state s can also be described in terms of the matching training examples (these are the examples that agree with the attribute values defining s), state-dependent costs are equivalent to example-dependent costs. This holds for both attribute and misclassification costs. For attribute costs,

$$C(s, x_n) = \sum_v P(x_n = v|s) \cdot \sum_y C(s, x_n, (s \cup \{x_n = v\}, y)), \tag{2.13}$$

where we also write $s' = s \cup \{x_n = v\}$.

2.5.1.2 Misclassification Costs

Misclassification costs can also be example-dependent, $MC(f_k, x, y)$, where for each example (x, y), x is the vector of attributes, y is the observed class, and f_k is the predicted class. For example, donation amounts in solicitation campaigns depend on the individual; purchasing amounts in catalogue mailing also depend on the customer; see [77, 56] for more details about the application of direct marketing.

In general, we can assume there is a distribution $P(x, y|s)$ of examples that match state s. Then we can estimate the expected cost of classification action f_k in state s as:

$$C(s, f_k) = E_P[MC(f_k, x, y)] = \sum_{x,y} P(x, y|s) MC(f_k, x, y). \tag{2.14}$$

In our experimental studies, we considered the simple case where test costs are constant (they depend only on the attribute x_n, $C(x_n)$), and misclassification costs are constant as well (they do not depend on examples, just on the predicted and observed class). But we can incorporate both attribute and misclassification costs of these more complex forms in our CSL framework, as described above in Equations 2.13 and 2.14, and the model remains an MDP because the Markov property still holds.

2.5.1.3 Costs Unknown at Learning Time

We assume test costs and misclassification costs are known during the learning phase. If costs are not known at learning time, we may still be able to learn some policies based on bounds of costs, and prune some policies. Let B represent a set of bounds on test and misclassification costs. We could prune a policy π_1 if $\min_{b \in B} V_b^{\pi_1} > \max_{b \in B} V_b^{\pi_2}$, where V_b^{π} is the value of policy π for cost values b.

If costs are known only at execution time and real-time decisions are needed, this rules out systematic search methods, because of their computational cost. Greedy algorithms can still be applied.

2.5.2 Complex Actions

We list several types of complex actions, mainly to suggest directions of future work. Our CSL framework can not currently address them.

2.5.2.1 Repeated Tests

We assume an attribute can be tested only once. But in general, tests may be repeated. For example, to diagnose a cracked tooth, the biting test and the cold-sensitivity test can be applied several times.

To capture repeated tests, the state representation will need to specify the history of repeated tests. The difficult part for learning is obtaining sufficient training data for repeated tests.

2.5.2.2 Side-effects of Tests

We assume our tests are pure observations, followed by a single diagnosis. In general, tests may have many kinds of side-effects that would require changing our approach.

1. The execution of one test may influence the costs of other tests. Suppose a test on the transmission of a car requires removing the engine from the car. Once the engine is out, other tests which also required removing the engine (e.g.,

checking the clutch) become much cheaper. Conversely, tests that require that the engine is in the car become much more expensive (because the engine needs to be put back into the car).

2. The execution of one test may influence the results of other tests. For example, if you administer a glucose intolerance test (by asking the patient to drink a very sweet drink and then measuring blood glucose), this will change the results of doing a measurement of blood insulin. Hence, blood insulin (and baseline glucose levels) should be measured before the glucose intolerance test.

3. The execution of a test may change the class variable (the disease of the patient). These tests are usually called attempted repairs or attempted therapies. A test like "administer penicillin and see if it cures the patient" clearly has the potential to change the class variable.

The first case can be handled by extending the MDP state to include a variable that indicates whether the engine is in or out of the car. The test cost information must then specify the cost of each test for both cases (engine in, engine out) and it must indicate whether the test causes the engine to change state. This doubles the number of states in the MDP, but otherwise our approach can be applied.

The second and third cases invalidate our approach to training data. In our approach, the training data consist of an unordered list of tests and their results, and we assume that the tests would give the same results for all possible orderings. For the second and third cases, this is not true, so our training data would need to record the actual order in which the tests were performed. In the worst case, we would need to provide training examples exhibiting all possible test orderings so that we could detect all possible interactions.

In problems where therapies or repairs are available, our simple model of diagnosis (make observations, then predict the disease of the patient) does not make sense. Instead, the problem can be formalized as one of performing a mix of observations and repairs until the problem is solved (i.e., the patient is healthy, the car is working, etc.).

We emphasize that this is a different problem definition than the CSL problem studied in this thesis, because the goal is repair or treatment rather than just diagnosis.

2.5.2.3 Delayed Test Results

We assume the result of a test is received immediately (i.e., without a time delay). In medicine, this is true for some tests but not all. For example, a blood test might require an hour; a test for strep throat might require 48 hours; a cancer biopsy might require several days. Furthermore, these tests may measure multiple attributes at once (for example, a blood lipid profile returns triglycerides, HDL, VLDL and LDL cholesterol). A physician may order one of these tests and then proceed with other diagnostic tests while waiting for the results. The CSL problem with delayed test results is no longer an MDP, but a Semi-MDP [60, 12], because actions take variable amounts of time.

2.5.3 CSL Problem Changes in Time

In real life, the CSL problem will change in time. This means that the MDP changes. Here are some possible causes of the changes in the MDP model:

- New training data arrives. This is more and more the case as hospitals and businesses keep track of records.

- New tests are discovered (for example, the test A1c for diabetic patients). Some tests become obsolete.

- New outcomes of old tests or treatments appear (for example, resistance to antibiotics).

- New diseases appear (such as SARS). Some diseases are eradicated.

- Costs change in time.

There is no easy way to deal with these changes in the MDP model. It is likely that the policy needs to be re-learned. In special cases, incremental updates are possible, or parts of the old search space may be reused.

2.5.4 Missing Attribute Values

Our work assumes that in each training example, every attribute has been measured. We can imagine that the training data was collected and labeled by an "observe-everything" exploration policy. Because of this, we can evaluate any policy on the collected training data. This kind of data can be collected in medical research studies. However, in observational data, such as data extracted from hospital records, such training examples are not likely to be observed. The reason is that physicians follow their own diagnostic policies, and of course these policies attempt to tradeoff the cost of tests against the cost of misclassifications.

It is possible to learn the optimal policy π^* by observing training examples collected under some other policy π provided that three conditions hold:

1. The policy π has a non-zero probability of reaching every state s that can be reached.

2. The policy π has a non-zero probability of executing every test in every reachable state s.

3. The policy π is only selecting actions based on the observed results of executed tests. That is, no outside information sources are being used.

Under these conditions, the transition probabilities of the MDP can be estimated and the methods described in this thesis can still be applied.

Unfortunately, these conditions are unlikely to be satisfied in practice by a physician following his or her own policy. The conditions could be guaranteed by modifying the physician's policy so that with a small probability every possible action is considered at each step (e.g., by slightly randomizing the physician's policy). Such techniques have been applied in reinforcement learning to guarantee that the optimal policy π^* can be learned [66].

2.5.5 Multiple Classes

Classification problems often have more than 2 classes, and this is easily accommodated in our approach. More classification actions simply add a constant amount of computation when evaluating every state s through the computation of their expected costs: $C(s, f_k) = \sum_y P(y|s) \cdot MC(f_k, y)$. Neither the greedy nor the systematic algorithms introduced in this thesis are limited to binary classes.

The most difficult problem is getting enough training data for all these classes. There is an additional overhead for determining the misclassification costs $MC(f_k, y)$ for more than two classes.

2.5.6 Continuous Attributes

Most real problems have continuous attributes, but the systematic search algorithms described in this thesis require attributes to have a small number of discrete values. The simplest approach to handle this is to consider all possible values as the outcomes of a test. This will increase the risk of overfitting for the algorithms (especially for the systematic search ones), since the state space becomes larger. Overfitting happens because data becomes fragmented and all decisions based on a smaller sample are less accurate. Overfitting also affects a different approach in which we change the decision nodes in the policy from "test attribute x_i" to "test if $x_i \leq$ threshold", with binary outcomes; in this case we will charge the cost of measuring x_i only once. Greedy algorithms introduced in Chapter 3 can easily deal with continuous attributes, providing the branching factor is not huge.

If the measured attribute values are continuous, our systematic search algorithms require that they be discretized prior to learning. We recommend no more than 10 values for the discretized attributes in order to avoid small data partitions and the ensuing overfitting. In this dissertation, we discretized continuous attributes in three partitions.

For systematic algorithms, we might use a simple greedy algorithm at each internal node for choosing a threshold for each continuous attribute.

2.5.7 Objective Function

Different CSL problems may have different objective functions. For example, in a bomb-threat the objective is to to find (and disable) the bomb, while minimizing time and loss of human lives and property.

If the objective function can be expressed as a single utility function (in terms of "costs"), then our CSL framework is applicable. Our goal is to find the diagnostic policy that minimizes the expected total cost. "Cost" can be time, etc., for example in a medical emergency the objective is to minimize the time to find the cause of the injury (e.g., which internal organ is failing?).

We can not address multiple objective functions.

2.6 Literature Review for the Cost-sensitive Learning Problem in Machine Learning

Machine learning has tackled several different settings for the classification problem, mentioned here in historical order:

1. **classifiers minimizing 0/1 loss** (see 2.1). This has been the main focus of machine learning, from which we mention only CART [9] and C4.5 [63]. These are standard top-down decision tree algorithms. C4.5 introduced the information gain as a heuristic for choosing which attribute to measure in each node. CART uses the GINI criterion.

   Weiss et al. [75] proposed an algorithm for learning decision rules of a fixed length for classifications in a medical application; there are no costs (the goal is to maximize prediction accuracy). Their paper also defines the commonly used medical terms of sensitivity and specificity of tests from a machine learning perspective.

2. **classifiers sensitive only to attribute costs:** Norton [51], Nunez [52] and Tan [68, 67]. The splitting criterion of these decision trees combines information gain and attribute costs. These policies are learned from data, and their objective is to maximize accuracy (equivalently, to minimize the expected number of classification errors) and to minimize expected costs of attributes.

   A related problem is the *test sequencing problem* [53], from electronic systems testing. Pattipati and Alexandridis point out that "the test sequencing problem belongs to the general case of binary identification problems that arise in botanical and zoological field work, plant pathology, medical diagnosis, computerized banking and pattern recognition." The objective of the test sequencing problem is to unambiguously (deterministically) identify the system state (either one of the faulty states, or the fault-free state) by performing tests with minimum expected total cost. The assumptions are that only one of the system states occurs (or equivalently, the faults are mutually exclusive), the probability distribution over the system states is given and so is the binary diagnostic matrix (which tells if a test detects a fault or not). The test sequencing problem is a simplified version of the cost-sensitive classification problem, because faults are identifiable with probability 1.0, and therefore there are no misclassification costs.

3. **classifiers sensitive only to misclassification costs:** Breiman and al. [9], Hermans et al. [28], Gordon and Perlis [21], Pazzani et al. [54], Knoll et al. [32], Fawcett and Provost [17], Gama [20], Margineantu [43], Zadrozny and Elkan [77], and Ikizler [30]. This problem setting assumes that all data is provided at once, therefore there are no costs for measuring attributes and only misclassification costs matter; this is not a sequential decision making problem. The objective is to minimize the expected misclassification costs.

This work can be further divided depending at which point in the learning process the knowledge about misclassification costs becomes available:

(a) **misclassification costs known during the learning of classifiers** (CART [9], MetaCost [14], post-pruning of decision trees using misclassification costs (Bradford et al. [8], Kukar and Kononenko [35], Webb [74])).

(b) **misclassification costs not known until execution time**. Two strategies are employed.

The first one learns cost-insensitive classifiers with improved conditional class probabilities $P(y|x)$, then classifies each test example x into the class with the minimum expected cost:

$$\hat{y}_{opt}(x) = \operatorname*{argmin}_{\hat{y}} \sum_{y} P(y|x) L(\hat{y}, y).$$

This approach includes logistic regression, Friedman and Stuetzle's projection pursuit regression [19], Naive Bayes, Domingos and Provost's B-PETs [15], Margineantu and Dietterich's B-LOTs [44].

The second strategy learns a range of operating points on an ROC curve. When costs become known at execution time, an operating point is chosen (Provost and Fawcett's ROC convex hull [58] and [59]).

4. **classifiers sensitive to both attribute costs and misclassification costs.** More recently, researchers have begun to consider both test and misclassification costs: [71, 22]. The objective is to minimize the expected total cost of tests and misclassifications. Both algorithms learn from data as well.

Peter Turney in 1995 [71] developed ICET, a cost-sensitive algorithm that employs genetic search to tune parameters used to construct decision trees. Each decision tree is built using Nunez' criterion (described in Section 3.2), which selects attributes greedily, based on their information gain and costs. Turney's method adjusts the test costs to change the behavior of Nunez' heuristic so that it builds different trees. These trees are evaluated on an internal holdout data set using the real test costs and misclassification costs. After several trials, the best set of test costs found by the genetic search is used by the Nunez' heuristic to build the final decision tree on the entire training data set. Because Turney simply modifies the attribute selection in C4.5 to add attribute costs when implementing the Nunez' criterion, his algorithm can deal with continuous attributes and with missing attribute values.

Turney's is a seminal paper laying the foundations of cost-sensitive learning with both attribute costs and misclassification costs. Turney compares his algorithm with C4.5 and with algorithms sensitive only to attribute costs (Norton, Nunez and Tan). He does not compare ICET with algorithms sensitive to misclassification costs only, because in his experiments he used simple misclassification cost matrices (equal costs on diagonal, equal costs off diagonal) which make algorithms sensitive only to misclassification costs equivalent to minimizing 0/1 loss. ICET outperformed the simpler greedy algorithms on several medical domains from the UCI repository.

Greiner, Grove & Roth's 2002 paper [22] is a theoretical work on a dynamic programming algorithm (value iteration) searching for best diagnostic policies measuring at most a constant number of attributes. They compute a theoretical sample size for which value iteration comes within ϵ of V^*, with probability at least $1 - \delta$ (a PAC-learning result). Their theoretical bound is not applicable in practice, because it requires a specified amount of training data in order to obtain close-to-optimal policies.

2.7 Related Work in Decision-theoretic Analysis

Decision analysis has addressed a problem similar to cost-sensitive learning, called *troubleshooting.* In the troubleshooting problem, it is known that the system is faulty, and the objective is to repair it, while minimizing the expected total cost of repairs and observations. It is assumed that a model is given (usually a Bayesian network) from which the necessary probabilities can be inferred. The Bayesian network is usually elicited from experts as opposed to being learned from training data.

The troubleshooting problem is slightly different from the cost-sensitive learning problem. The purpose is fault repair, not diagnosis. In addition to pure observation actions (similar to our tests), troubleshooting has repair actions that affect the state of the device (change the class, in our terminology). More precisely, they can change the state of the device from faulty to functioning. Repairs are similar to our classification actions, but a repair action is terminal only when it fixes the device. The costs of the observation and repair actions are deterministic and independent. There is no learning from data; instead a model is assumed.

Heckerman et al. [27] characterize a class of problems for which the optimal policy can be computed greedily, assuming there is a single faulty component and there are only repair actions with independent costs. Let there be n components in the system. Let C_i^r be the cost of repairing component c_i and p_i the probability (in the start state) that repairing this component fixes the system. Repairing a component does not affect other components. The assumption is that after fixing all components, the system will be in working condition (or else a service call, of known cost, can be made, and it guarantees to get the system to function). With the above assumptions, the repair policy is optimal; it performs a sequence of repair actions in decreasing order of $\frac{p_i}{C_i^r}$, until the system is working. We can think of this repair sequence as a macro terminal action (it will eventually fix the system). We will also refer to the repair sequence as the "stop testing" policy.

When both repair actions and observation actions are available, Heckerman et al. construct a policy using the one-step value of information (VOI) heuristic. The idea is to compute the value of information of each possible observation action and choose the action with the largest VOI. The value of information of an observation action is the difference in expected cost of performing the repair sequence now, versus performing first the observation, then executing the corresponding repair sequence (after computing the repair probabilities p_i based on the outcomes of the observation). If the VOI of all observations is negative, the repair sequence is executed without performing any observations. If VOI is positive, the best observation action is executed and then the VOI values are recomputed based on its outcomes. In the domain of troubleshooting car problems, the expected cost of the one-step VOI policy was close to optimal and this policy outperformed simpler planners.

On a different problem, Fountain et al. [18] applied iterative one-step VOI to decide which integrated circuit chip (die) on a wafer should be tested next, and when to stop testing. The problem is to compute a policy for choosing which dice to test on a wafer in order to maximize the total profit resulting from testing, packaging, and selling the ICs.

Each die can be subjected to a die-level functional test to determine if it operates correctly. Traditionally, all dice on a wafer are tested, the bad ones are inked, the wafers are shipped to a different location, cut apart (discarding the inked ones), packaged (or assembled into ICs), tested one more time in package form (these are called package tests) and finally sold. Since package tests double check the quality of the ICs, die-level functional tests are only needed to avoid packaging defective dice (and therefore cut costs), and also to diagnose problems with the manufacturing process.

Fountain et al. fit a Naive Bayes model to historical data from the Hewlett Packard company, hypothesizing that there is a class (or fault) variable that causes each individual die to fail, independently of the rest. Since this data is unlabeled, the authors experimentally set the number of classes to four, then employ the EM algorithm to learn the parameters of the Naive Bayes model. Interestingly, Fountain et al. divided the data into training data, to which they fit the Naive Bayes, and test data, on which the learned policy is evaluated.

There are 209 dice on a wafer, which creates too large a branching factor to attempt exhaustive search. The one-step VOI was applied to decide between testing one more die (after which it performs "stop testing" policies), and "stop testing" now. Fountain et al. only perform this greedy policy at run time, based on the actual results of tests performed so far, rather then at a separate learning time.

Computing the value of the "stop testing" policy is more complex than our computation of the expected costs of classifications actions. An inking decision must be made for each die; note that the inference of probabilities for the inking decision is easy because of the simple Naive Bayes model. There is an additional decision of whether to discard the entire wafer. Then, if the wafer is not discarded, the costs of shipping, cutting, packaging, package testing, and finally selling the good ICs must be computed.

The greedy VOI policy produces more net profit than simple policies that test all dice exhaustively or that test no dice at all (at the functional level). Moreover, the one-step VOI policy detects abnormal wafers (so it can give feedback to the fabrication process in real time) and is robust to changes in testing costs.

Both troubleshooting and die-level testing problems are MDPs where each state stores the history of test outcomes. Both application problems employ a model to infer the probabilities and perform greedy search (the one-step VOI) to compute policies. The action space includes tests, but it does not include classification actions, though it has similar terminal actions (which enable the one-step VOI policy to compute the value of the "stop testing" policy).

2.8 Summary

This chapter formalized the cost-sensitive learning problem by using terminology from supervised learning and the MDP framework. We borrowed the format of the data (sets of labeled examples) and the task of predicting the class from supervised learning. We borrowed the notion of sequential decision processes and the goal of minimizing the expected total cost from MDPs. We showed that diagnostic policies take the form of decision trees. We demonstrated that the order in which diagnostic tests are performed is important. We identified restrictions of our framework and discussed related work. The following chapters delve into the design space of CSL algorithms. We introduce greedy methods in Chapter 3, and systematic search methods in Chapter 4.

CHAPTER 3

GREEDY SEARCH FOR DIAGNOSTIC POLICIES

This chapter describes a design space of greedy search algorithms for diagnostic policies. In Chapter 5 we will perform an experimental comparison of various algorithms in this space. *Greedy algorithms* perform a limited lookahead search, using information specific to the problem (informativeness of attributes, attribute costs or misclassification costs or both, etc.), but once they commit to a choice of an attribute to test, that choice is final; that is, usually they can not revise their choices, and the policies they produce will not be optimal in general.

We start by reviewing several existing algorithms whose attribute selection heuristics combine attribute information gains and attribute costs, but which ignore misclassification costs. Then we extend these algorithms by adding Laplace corrections and by choosing classification actions sensitive to misclassification costs. Last we describe the well-known one-step value of information (VOI) algorithm from decision theory.

Even if the greedy policies are, in general, not optimal, on many problems their performance can be good enough, and they have the advantage of fast execution. They also require less information to be extracted from the training examples, which reduces the risk of overfitting.

3.1 General Description of Greedy Algorithms

The greedy algorithms described here employ a top-down strategy of selecting attributes. They differ in the way they select attributes, in the stopping conditions, and in the way they choose classification actions. Table 3.1 gives the pseudocode for greedy search, and we will describe how each method addresses the numbered steps.

TABLE 3.1: The GREEDY search algorithm. Initially, the function Greedy() is called with the start state s_0.

```
function Greedy(state $s$) returns a policy $\pi$ (in the form of a decision tree).

(1) if (stopping conditions are not met)
(2)     select attribute $x_n$ to test;
        set $\pi(s) = x_n$;
        for each value $v$ of the attribute $x_n$ add the subtree
            Greedy(state $s \cup \{x_n = v\}$);
    else
(3)     classify state $s$ in $best_f$, set $\pi(s) = best_f$;
(4) post-prune(state s).
```

In their search through the space of decision trees, the greedy algorithms never backtrack to reconsider other attributes, but we will see that post-pruning (step 4) may replace an attribute and the subtree underneath it by a classification action, but it does not consider measuring other attributes.

3.2 InfoGainCost Methods

InfoGainCost methods grow a policy recursively top down. At each node, they choose to measure the attribute with the maximum value of the InformationGainCost criterion, defined below. After an attribute has been chosen, the training examples are partitioned according to their values for the selected attribute. This process is repeated recursively for each of the resulting child nodes. The leaf nodes specify the classification actions.

These methods pay attention to test costs and accuracy. Note that, conceptually, this is equivalent to assuming huge misclassification costs, rather than zero misclassification costs (because zero costs for errors imply direct classification in any of the classes, without measuring any attributes, and the classification problem is trivial).

The InfoGainCost methods specialize the generic algorithm of Table 3.1 as follows. For line (1) they use the C4.5 stopping conditions described below. For line (3) they *classify in the majority class.* Line (4) is a post-pruning step using C4.5's *pessimistic post-pruning* (with an added optional flavor of Laplace correction).

A node with state s is classified according to the majority class, $best_f = \arg\max_y P(y|s)$, when one of these *stopping conditions* is satisfied (these are the *C4.5 stopping conditions*):

- there are no more attributes to be tested,
- all matching examples have the same class (pure node), or
- no attribute splits the data into at least 2 subsets with ≥ 2 examples.

As the policy is grown, nodes can be *collapsed* (a node will be made into a leaf if its children make more misclassification errors than it does).

We will consider several ways of implementing step (2). All of them employ in some way the information gain (or mutual information) of an attribute. The *information gain* of attribute x_n in state s is the expected reduction in class entropy by splitting on this attribute:

$$ig \stackrel{\text{def}}{=} InfoGain(s, x_n) = Entropy(s) - \sum_v P(x_n = v|s) \cdot Entropy(s \cup \{x_n = v\}),$$

where $Entropy(s) = -\sum_y P(y|s) \cdot \log_2 P(y|s)$.

In state s, for attribute x_n not yet measured, with attribute cost $C(s, x_n)$, we define the *InformationGainCost*(s, x_n) criterion to be one of the following variants:

- *ig*. This selection criterion is borrowed from ID3 and C4.5. This method, denoted **InfoGain** (or just **IG**), is cost-insensitive. It ignores both attribute costs and misclassification costs.

- $ig/C(s, x_n)$ (**Norton**'s criterion [51]).
- $ig \cdot ig/C(s, x_n)$ (**Tan**'s criterion [68, 67]).
- $(2^{ig} - 1)/(C(s, x_n) + 1)$ (**Nunez**' criterion [52] — slightly modified here).

At each internal node in the decision tree, the attribute with the maximum InformationGainCost value will be selected (step (2)),

$$\pi(s) = \arg \max_{x_n \in A(s)} InformationGainCost(s, x_n),$$

and the training examples will be recursively partitioned according to the value of the selected attribute. Since we require that an attribute be tested only once, the selected attribute becomes unavailable for future selection. This is easily checked by examining the state s and only computing InformationGainCost(s, x_n) for attributes that have no assigned value, so they have not yet been measured, $x_n \in A(s)$.

While the last three methods, Norton, Tan and Nunez, do not have a clear cost-sensitive objective (in our opinion, they strive for good classifications with low expected attribute costs), they do bias the search in favor of low-cost attributes; indeed, attributes with smaller cost $C(s, x_n)$ will have a larger InformationGainCost and will be placed closer to the root than in ID3 or C4.5. Nevertheless, the greedy selection of an attribute that appears to contribute most to reducing classification errors and least to the testing costs ignores interactions among attributes, and may just postpone selecting important, but expensive, tests.

For step (4), we apply a post-pruning procedure in all four variants. Consider any node in the policy matching n training examples, out of which n_y belong to class y. The error rate of classifying into the majority class $best_f = \arg\max_y P(y|s)$ is $p = 1 - max_y P(y|s)$.

The number of errors at a node has a binomial distribution with parameters (n, p), n being the sample size (the training examples matching the node) and p the proportion of failures. We pessimistically estimate the error rate as the upper bound of the 75% confidence interval for this binomial distribution, that is,

$$p + z_c \times \sqrt{p \cdot (1-p)/n} + 0.5/n,$$

where $0.5/n$ is the correction for continuity (because we use the normal distribution to approximate the sampling distribution of a proportion, which is not continuous) and $z_c = 1.15$ is the confidence coefficient corresponding to a 75% confidence interval. When $p = 0$, the new estimate for error rate is $0.5/n > 0$.

The pessimistic estimate of the number of errors is obtained by multiplying n with this upper bound for error rate. A node will be pruned if the sum of its children's pessimistic errors is greater than or equal to its own pessimistic error, if the node were classified. This pruning may help reduce overfitting. Indeed, if we do not prune, the policy is likely to overfit the data, by overestimating how informative the attributes are, and we will end up paying too much for them.

In Section 5.3.1, we study whether it is useful to apply a Laplace correction (of +1) to the error rate p before computing the pessimistic bound. The Laplace-corrected error rate is $p^L = \frac{\sum_{y \neq best_f}(n_y+1)}{n+K} = \frac{K-1+\sum_{y \neq best_f} n_y}{n+K}$, where K is the number of classes. This can be viewed as adding a "fake" example to each class before computing the probabilities.

In Section 2.4, policy π_1 in Figure 2.1 is the policy constructed by the InfoGain method (in this simple problem, all the InfoGainCost methods construct the same policy). In the start state s_0, the attribute BMI is both cheaper and more informative than the attribute Insulin: $C(BMI) = 1 < C(Insulin) = 22.78$ and $InfoGain(s_0, BMI) = 0.21 > InfoGain(s_0, Insulin) = 0.14$. In state $\{BMI = small\}$, π_1 first measures Insulin and classifies as Healthy in the resulting states. But since the number of errors of this subtree is equal to the number of errors of classifying Healthy in $\{BMI = small\}$, the subtree is collapsed. In state $\{BMI = large\}$, Insulin is chosen to be measured. After measuring Insulin, there are no more attributes to test, so the algorithm will choose to classify. In this problem, it so happened that the majority class in all the policy's leaf states is also the action with minimum expected cost. In state $s_1 = \{BMI = small\}$, $P(Healthy|s_1) = .9$, in state $s_3 = \{BMI = large, Insulin = low\}$, $P(Diabetes|s_3) = .7$ and in state $s_4 = \{BMI = large, Insulin = high\}$, $P(Healthy|s_4) = .8$. Pessimistic post-pruning (with or without Laplace correction) does not change the policy constructed by the InfoGainCost methods (it does not help in this case).

Note that policy π_1, constructed in this way, is not the optimal policy. The optimal policy (Figure 2.3) tests BMI and then classifies.

3.3 Modified InfoGainCost Methods (MC+InfoGainCost)

We now describe four new greedy methods, which we call MC+InfoGainCost methods, that are sensitive to both attribute costs and misclassification costs. These methods inherit steps (1) and (2) in Table 3.1 from the InfoGainCost methods, but they modify steps (3) and (4) as follows:

- in step (3), they classify into the class with the minimum expected classification cost, $best_f = \arg\min_{f_k} C(s, f_k)$ (see equation 2.7, $C(s, f_k) = \sum_y P(y|s) \cdot MC(f_k, y)$), instead of classifying into the majority class (with minimum error rate).

- in step (4), if $\pi(s) = x_n$, they prune this action if $C(s, best_f) \leq Q^\pi(s, x_n)$ (see equation 2.10, $Q^\pi(s, x_n) = C(s, x_n) + \sum_v P(x_n = v|s) \times V^\pi(s \cup \{x_n = v\})$); so when the Q value of (s, x_n) surpasses the expected cost of the best classification action, the node corresponding to state s is turned into a leaf that classifies into $best_f$.

As we mentioned in the previous section, all the InfoGainCost methods constructed an identical policy for the problem in Section 2.4, the policy shown in Figure 2.1, and this is also the policy grown by our MC+InfoGainCost methods before the post-pruning phase, because the majority classes in the leaf states happen to also have the minimum expected cost. Indeed, in state $s_1 = \{BMI = small\}$, $C(s_1, Healthy) = 10 < C(s_1, Diabetes) = 72$, in state $s_3 = \{BMI =$

$large, Insulin = low\}$, $C(s_3, Diabetes) = 24 < C(s_3, Healthy) = 70$ and in state $s_4 = \{BMI = large, Insulin = high\}$, $C(s_4, Healthy) = 20 < C(s_4, Diabetes) = 64$.

We now explain how post-pruning works. During the top-down phase, the greedy policy chooses to measure the remaining attribute, Insulin, in state $s_2 = \{BMI = large\}$. Because $Q^{\pi}(s_2, Insulin) = 45.98 > C(s_2, best_f = Diabetes) = 32$, the subtree rooted at s_2 will be pruned and replaced by the classification action Diabetes. In state $s_1 = \{BMI = small\}$, the greedy policy tested Insulin as well, and classified as Healthy in the resulting states. Pruning will eliminate this extra test, because the cost of the subtree is $C(s_1, Insulin) + C(s_1, Healthy)$, which exceeds the cost of classifying directly into s_1 as Healthy, $C(s_1, Healthy)$. This example shows that this type of pruning, based on misclassification costs, is more general than collapsing, which was based on errors made by the majority class, and therefore collapsing is not needed in the MC+InfoGainCost methods. In the start state s_0, there is no pruning, because the cost of the best classification action $C(s_0, Healthy) = 35$ exceeds $Q^{\pi}(s_0, BMI) = 22$ (this cost reflects the new policy value in s_2 after pruning, $V^{\pi}(s_2) = 32$).

Note that, during the pruning phase, we need to compute the classification action $best_f$ with minimum expected cost in every state reached by the policy. For example, in state $s_2 = \{BMI = large\}$, $best_f = Diabetes$ because $C(s_2, Diabetes) = 32 < C(s_2, Healthy) = 60$. In this problem, the MC+InfoGainCost policy is optimal (see the policy in Figure 2.3).

We have studied the option of adding a *Laplace correction* of +1 to the probabilities $P(y|s)$ and $P(x_n = v|s)$ involved in the computation of $C(s, f_k)$ and $Q^{\pi}(s, x_n)$ in steps (3) and (4). This is accomplished by adding a fake example to each of the possible cases, as described below.

If the estimated class probability is $P(y|s) = n_y/n$, where n is the number of training examples matching state s and out of them, n_y belong to class y, then the Laplace corrected class probability is

$$P^L(y|s) = \frac{n_y + 1}{n + K}, \tag{3.1}$$

where K is the number of classes. A fake example was added to each of the classes when computing $P^L(y|s)$.

Let n_v be the number of training examples matching state $s' = s \cup \{x_n = v\}$ — in other words, n_v is the number of training examples matching state s with value v for attribute x_n. If the estimated transition probability is

$$P(x_n = v|s) = P_{tr}(s'|s, x_n) = n_v/n,$$

then the Laplace corrected transition probability is

$$P^L(x_n = v|s) = \frac{n_v + 1}{n + Arity(x_n)}. \tag{3.2}$$

A fake example was added to each of the attribute values when computing $P^L(x_n = v|s)$.

Let us assume there are no training examples matching state $s' = s \cup \{x_n = v\}$, so $P(x_n = v|s) = 0$, and we originally classified state s' in the best class of s. With the Laplace correction, $P^L(x_n = v|s) = 1/(n + V_n)$, and $P^L(y|s') = 1/K$. Therefore, $C^L(s', f_k) = \sum_y MC(f_k, y)/K$, and state s' will be classified into the class with minimum average misclassification costs. The Laplace correction has no effect on the example policy above.

3.4 One-step Value of Information (VOI)

This is another method that pays attention to both attribute costs and misclassification costs. The one-step VOI algorithm iteratively selects the attribute that looks best according to the following simple lookahead: measure the attribute; then for each of its possible values classify into the best class (with minimum expected cost). Once an attribute is selected, the process is repeated in the resulting states. The method stops when there are no more attributes to measure (we write this $A(s) = \emptyset$) or when it is cheaper to classify. Intuitively, the one-step VOI repeatedly asks the question, in every state it reaches: is it worth testing one more attribute, after which it classifies, or should it stop now and classify?

We define the one-step lookahead value of attribute x_n in state s, called 1-step-LA(s, x_n), to be equal to the cost of measuring this attribute plus the expected cost of making the best classifications in the resulting states:

$$\text{1-step-LA}(s, x_n) \overset{\text{def}}{=} C(s, x_n) + \sum_v P(x_n = v|s) \times \min_{f_k} C(s \cup \{x_n = v\}, f_k). \quad (3.3)$$

The one-step value of information of attribute x_n in state s weighs the value of classifying now against the value of measuring x_n, followed by classification. By definition, $VOI(s, x_n)$ is the difference in expected costs between classifying before and after the attribute is measured:

$$VOI(s, x_n) \overset{\text{def}}{=} C(s, best_f) - \text{1-step-LA}(s, x_n).$$

Step (2) in Table 3.1 selects attribute x_n^* with $\min_{x_n}$ 1-step-LA(s, x_n), or, equivalently, the attribute with the maximum $VOI(s, x_n)$. Attribute x_n^* is measured only if it has a strictly positive value of information, $VOI(s, x_n^*) > 0$, or $C(s, best_f) >$ 1-step-LA(s, x_n^*), otherwise it is cheaper to classify in state s.

Step (3), selecting the best classification action, is the same as for the MC+InfoGainCost methods, so $best_f = \arg\min_{f_k} C(s, f_k)$.

Steps (1) and (2) are tightly related. In fact, during step (1), we first calculate $best_f$, the classification action with the minimum expected cost. If $A(s) = \emptyset$ we classify into $best_f$, otherwise we calculate the one-step lookahead value of each $x_n \in A(s)$ as in equation 3.3. In step (2) we select the attribute x_n^* with the minimum 1-step-LA(s, x_n) and compare this value with $C(s, best_f)$. If classifying is cheaper, then we stop and classify, otherwise we measure the attribute and add subtrees for its outcomes. We formally list the operation of each of the steps in Table 3.1, then we rewrite the one-step VOI algorithm in Table 3.2. With our notations, one-step VOI is the algorithm, VOI is the policy it learns, and VOI is the function in Table 3.2.

- in step(1), stop when $A(s) = \emptyset$ or $C(s, best_f) \le min_{x_n \in A(s)}$1-step-LA$(s, x_n)$.

- in step (2), select attribute $x_n^* = \arg\min_{x_n \in A(s)}$ 1-step-LA(s, x_n).

TABLE 3.2: The ONE-STEP VALUE OF INFORMATION (VOI) search algorithm. Initially, the function VOI() is called with the start state s_0.

```
**function** VOI(state $s$) **returns** a policy (in the form of a decision tree).
Compute $best_f = \arg\min_{f_k} C(s, f_k)$.
Compute $x_n^* = \arg\min_{x_n \in A(s)}$ 1-step-LA$(s, x_n)$, where
    $1\text{-step-LA}(s, x_n) \overset{\text{def}}{=} C(s, x_n) + \sum_v P(x_n = v|s) \times \min_{f_k} C(s \cup \{x_n = v\}, f_k).$
if $(A(s) = \emptyset)$ or $(C(s, best_f) \le 1\text{-step-LA}(s, x_n^*))$
  classify into $best_f$, set policy $\mathsf{VOI(s)} = best_f$.
else
  select attribute $x_n^*$ to test;
  set policy $\mathsf{VOI(s)} = x_n^*$;
  for each value $v$ of the attribute $x_n^*$ add the subtree
      VOI(state $s \cup \{x_n^* = v\}$);
```

- in step (3), classify into $best_f = \arg\min_{f_k} C(s, f_k)$.

- we show that step (4) is not necessary, because the VOI policy already satisfies the condition $Q^\pi(s, x_n) < C(s, best_f)$ if $\pi(s) = x_n$.

We can summarize the definition of the VOI policy computed by the one-step VOI algorithm as

$$\mathsf{VOI}(s) = \arg \min_{f_k, x_n \in A(s)} \{C(s, f_k), 1\text{-step-LA}(s, x_n)\},$$

breaking ties in favor of classification actions, after which we prefer the action with the lowest index. If there are no more attributes to be tested, that is $A(s) = \emptyset$, then we classify into the class with minimum expected cost.

Theorem 3.4.1 proves that no pruning (step (4)) is necessary, because pruning is built-in.

Theorem 3.4.1 *Let s be any of the non-leaf states reached by the* VOI *policy π, and let $\pi(s) = x_n$. Then $Q^\pi(s, x_n) \leq$ 1-step-LA$(s, x_n) < C(s, best_f)$, where $best_f = \arg\min_{f_k} C(s, f_k)$.*

Proof by induction:

Because $\pi(s) = x_n$, it follows from the definition of one-step VOI (and from the way we break ties in favor of classification actions), that 1-step-LA$(s, x_n) < C(s, best_f)$. So we only need to show that $Q^\pi(s, x_n) \leq$ 1-step-LA(s, x_n).

Base case

Let s be any state reached by π whose children $s' = s \cup \{x_n = v\}$ are leaf states, so $\pi(s') = \arg\min_{f_k} C(s', f_k)$, and $V^\pi(s') = min_{f_k} C(s', f_k)$. Recall that $P(x_n = v|s) = P_{tr}(s'|s, x_n)$ so we will be using the second notation for brevity. Then

$$\begin{aligned} Q^\pi(s, x_n) &\overset{\text{def}}{=} C(s, x_n) + \textstyle\sum_{s'} P_{tr}(s'|s, x_n) \times V^\pi(s') \\ &= C(s, x_n) + \textstyle\sum_{s'} P_{tr}(s'|s, x_n) \times min_{f_k} C(s', f_k) \overset{\text{def}}{=} \text{1-step-LA}(s, x_n), \end{aligned}$$

therefore $Q^\pi(s, x_n) =$ 1-step-LA(s, x_n). This establishes the base case.

Now let s be any state that has at least one non-leaf child. We assume all its non-leaf children s' satisfy the induction hypothesis:

Induction hypothesis

If $\pi(s') = x'_n$, then $Q^\pi(s', x'_n) \leq$ 1-step-LA$(s', x'_n) < C(s', best'_f)$, where $best'_f = \arg\min_{f_k} C(s', f_k)$.

Since $\pi(s') = x'_n$, it follows that $V^\pi(s') = Q^\pi(s', x'_n)$, and from the induction hypothesis, by transitivity of $<$, we have $V^\pi(s') < C(s', best'_f)$.

If s has leaf children s', then by definition $V^\pi(s') = C(s', best'_f)$.

Then, because s has at least one non-leaf child s' for which $V^\pi(s') < C(s', best'_f)$, we have

$$\begin{aligned} Q^\pi(s, x_n) &\overset{\text{def}}{=} C(s, x_n) + \textstyle\sum_{s'} P_{tr}(s'|s, x_n) \times V^\pi(s') \\ &< C(s, x_n) + \textstyle\sum_{s'} P_{tr}(s'|s, x_n) \times C(s', best'_f) \overset{\text{def}}{=} \text{1-step-LA}(s, x_n), \end{aligned}$$

therefore $Q^\pi(s, x_n) <$ 1-step-LA(s, x_n). **Q.E.D.**

In Section 2.4, the policy in Figure 2.3 is the policy constructed by the one-step VOI method, and in this simple problem this policy is optimal on the training data. We will explain in detail the construction of the VOI policy for this problem, following the steps in Table 3.2. We first invoke the function in the start state s_0, VOI(s_0). We compute $best_f$ in s_0 to be Healthy, because $C(s_0, Healthy) = 35 < C(s_0, Diabetes) = 52$. Next we compute the one-step lookahead value of attributes BMI and Insulin, 1-step-LA$(s_0, BMI) = 22 <$ 1-step-LA$(s_0, Insulin) = 49.38$, so $x_n^* = BMI$. In fact, we do not even have to compute 1-step-LA$(s_0, Insulin)$; it is enough to notice that

$$\text{1-step-LA}(s_0, Insulin) \geq C(s_0, Insulin) = 22.78 > \text{1-step-LA}(s_0, BMI) = 22.$$

Because $C(s_0, Healthy) >$ 1-step-LA(s_0, BMI), BMI will be chosen and $\mathsf{VOI}(s_0) = BMI$.

Next we recursively call the one-step VOI function in the resulting states $s_1 = \{BMI = small\}$ and $s_2 = \{BMI = large\}$. In state s_1, $best_f = Healthy$ because $C(s_1, Healthy) = 10 < C(s_1, Diabetes) = 72$. There is a single attribute remaining, Insulin, and 1-step-LA$(s_1, Insulin) = 32.78$. Since $C(s_1, Healthy) <$ 1-step-LA$(s_1, Insulin)$, we classify s_1 as Healthy and set $\mathsf{VOI}(s_1) = Healthy$. Again, we could notice right away that

$$\text{1-step-LA}(s_1, Insulin) \geq C(s_1, Insulin) = 22.78 > C(s_1, Healthy) = 10.$$

Similarly, in state s_2, $best_f = Diabetes$ because

$$C(s_2, Diabetes) = 32 < C(s_2, Healthy) = 60.$$

There is a single attribute remaining, and 1-step-LA$(s_2, Insulin) = 45.98$. Since $C(s_2, Diabetes) <$ 1-step-LA$(s_2, Insulin)$, we classify s_2 as Diabetes and set $\mathsf{VOI}(s_2) = Diabetes$.

The resulting one-step VOI policy is shown in Figure 2.3. On this problem with only two attributes, the one-step VOI policy is optimal because

- it chose the best attribute in the start state, $\mathsf{VOI}(s_0) = \pi^*(s_0)$, and

- in the resulting states s' there is a single attribute left to measure, after which classification is the only possibility, so the myopic computation of one-step VOI is optimal in these states s', 1-step-LA$(s', x_n) = Q^*(s', x_n)$.

After comparing with $C(s', best_f)$, $\mathsf{VOI}(s') = \pi^*(s')$. The same explanation applies to the MC+InfoGainCost methods for this problem, because their policy π has $Q^\pi(s', x_n) = Q^*(s', x_n)$, in the states s' where there is only one unmeasured attribute.

This is a trivial result that holds in general: for any state s with a single attribute x_n left to measure, and for any policy that classifies in the cheapest class after measuring x_n, we have $Q^\pi(s, x_n) = Q^*(s, x_n)$. There could be more than one such policy if there is a tie among classification actions in the leaves.

Laplace correction can be applied to the probabilities $P(y|s)$ and $P(x_n = v|s)$ as in equations 3.1 and 3.2. These can then be used in the computation of $best_f$ and x_n^* in Table 3.2. The Laplace correction has no effect on the policy computed in the above example.

In General, the VOI *Policy is Not Optimal*

Because the one-step lookahead values are based on classifications in the resulting states s', and the classification costs overestimate the optimal value function,

$$C(s', best_f) \geq V^*(s') \stackrel{\text{def}}{=} \min_{x_n \in A(s')} \left\{ C(s', best_f), Q^*(s', x_n) \right\},$$

it follows that 1-step-LA$(s, x_n) \geq Q^*(s, x_n)$. Therefore the choice of actions of the VOI policy is not optimal. Indeed,

$$\mathsf{VOI}(s) = \arg \min_{f_k, x_n \in A(s)} \left\{ C(s, f_k), \text{1-step-LA}(s, x_n) \right\},$$

while

$$\pi^*(s) = \arg \min_{f_k, x_n \in A(s)} \left\{ C(s, f_k), Q^*(s, x_n) \right\},$$

so there is no guarantee that the VOI policy is optimal. Of course, to prove that a policy is suboptimal we need to compare its value function with the optimal value function in the start state, $V^*(s_0)$, because there can be several tied optimal policies.

The proof that the value of the VOI policy overestimates the optimal value function V^* is more complicated. Let $\pi = \mathsf{VOI}$, and let V^π be the value function of the VOI policy. Let s be a state reached by the VOI policy. There are two cases in which the VOI policy can be suboptimal:

1. it stops testing too early, that is $\pi(s) = best_f$, while $\pi^*(s) = x_n$. Because $V^*(s) < C(s, best_f) = V^\pi(s)$, it can be proven by induction that $V^*(s_0) < V^\pi(s_0)$, where s_0 is the start state. In this case $\pi = \mathsf{VOI}$ is suboptimal.

2. it chooses the wrong attribute to test, that is, $\pi(s) = x_1$, $\pi^*(s) = x_2$, and $x_1 \neq x_2$. Nevertheless, the two policies may still be equally good (i.e., $V^*(s_0) = V^\pi(s_0)$), if subsequent attributes measured by π improve its value.

We present an example of case 1, where the one-step VOI policy is not optimal. Consider the case where there are two binary attributes x_1 and x_2, and the class y is the XOR function, $y = x_1 \otimes x_2$. Assume there are equal numbers of examples for each combination of attributes. The attribute costs are c_1 and c_2, with $0 < c_1 < c_2$, and the misclassification costs are zero if $\hat{y} = y$ and m if $\hat{y} \neq y$, with $c_1 + c_2 < 0.5 \cdot m$. In s_0, before any attributes are measured, $P(y) = 0.5$, so both classification actions are equally expensive, with $C(s_0, best_f) = 0.5 \cdot m$. After measuring either attribute, the class probabilities do not change, so in the resulting states s', $C(s', best_f) = C(s_0, best_f)$. Therefore the one-step lookahead values are 1-step-LA$(s_0, x_1) = c_1 + C(s_0, best_f)$ and 1-step-LA$(s_0, x_2) = c_2 + C(s_0, best_f)$, with 1-step-LA$(s_0, x_1) <$ 1-step-LA(s_0, x_2) (so x_1 is cheaper according to the 1-step-LA). Because $C(s_0, best_f) <$ 1-step-LA$(s_0, x_1) = c_1 + C(s_0, best_f)$, that is, in s_0 classifying is cheaper than testing the best attribute and classifying afterwards, the one-step VOI policy chooses to classify in s_0, and $\mathsf{VOI}(s_0) = best_f$. The value of the VOI policy in s_0 is $C(s_0, best_f)$. On the other hand, the optimal policy achieves perfect classification after measuring both attributes, and $V^*(s_0) = c_1 + c_2$. By construction, this is less than $C(s_0, best_f) = 0.5 \cdot m$. Therefore the VOI policy is not optimal (and would not be optimal even if the attributes had equal costs).

In practice, when learning from data, what matters is not how close the one-step VOI policy comes to the optimal policy on the training data, but how close it gets to the optimal policy on the true underlying distribution of the data. Being suboptimal on the training data may reduce overfitting and trigger better performance on a different sample (in particular, the test set).

The same reasoning applies to show that the VOI policy may stop testing earlier than the InfoGainCost methods. In the case of the XOR example, greedy policies selecting attributes based on InfoGain will be able to compute the optimal policy even if the information gain of each attribute is zero, because they keep testing attributes until the C4.5 stopping conditions are satisfied.

3.5 Implementation Details for Greedy Algorithms

In our implementation, we *break ties* following this rule: classifying is preferred to testing attributes. In the case of ties among classification actions, we prefer the class with the lowest index; and in case of ties among attributes, we break the tie in favor of the attribute with the lowest index.

If there are no training examples traveling down to state $s \cup \{x_n = v\}$, that is, $P(x_n = v|s) = 0$, then we ignore that branch in the calculation of $InfoGain(s, x_n)$ and 1-step-LA(s, x_n), and the state $s \cup \{x_n = v\}$ will be classified into the best class of state s, $best_f$. For the InfoGainCost methods, $best_f$ is the majority class; for the MC+InfoGainCost methods and for the one-step VOI, $best_f$ is the class with the minimum expected cost.

For InfoGainCost and MC+InfoGainCost methods, we used information gain, and not C4.5's information gain ratio, because we discretized the attributes in at most three levels, so we do not have the risk of attributes over-splitting the data.

The InfoGainCost methods were designed to solve a different problem, one that maximizes accuracy while minimizing attribute costs. We still study them in our CSL framework, which considers both test costs and misclassification costs. But of course in our CSL framework, we always evaluate a policy using both test costs and misclassification costs.

3.6 Summary

We described several greedy algorithms that construct suboptimal policies based on a limited lookahead. The next chapter describes systematic search algorithms that produce optimal policies (optimal on the training data).

CHAPTER 4

SYSTEMATIC SEARCH FOR DIAGNOSTIC POLICIES

This chapter describes a systematic search algorithm that computes an optimal cost-sensitive diagnostic policy, optimal with respect to the training data. The natural search process for this optimal policy is in the space of AND/OR graphs, and the algorithm is called AO*. This algorithm belongs to the family of best-first algorithms guided by admissible heuristics. As Hansen points out in [25], heuristic search has an advantage over dynamic programming methods when searching from a given start state, because it computes an optimal policy but it does not need to evaluate the entire state space.

In this chapter, we introduce notations for AND/OR graphs, we define an admissible heuristic, and we describe the implementation of the AO* algorithm for CSL problems. The search process is guided by a lower and an upper bound to the optimal value function. These bounds provide significant cutoffs in the search space. The bounds correspond to value functions computed for an optimistic and a realistic policy. The realistic policy is of particular interest, because we may stop the search process at any time and return this policy.

Because we aim to avoid overfitting, we introduce several regularizers into the AO* algorithm. As a result, AO* with regularization will no longer compute an optimal policy on the training data, but rather a policy whose quality will hopefully be better on the test data. These regularizers modify AO* by (a) imposing a memory limit on the search space, (b) computing probabilities using the Laplace correction, (c) pruning the search space in various ways, and (d) stopping training early using a holdout set. We also initialized the AO* search graph with a greedy policy (any of the MC+InfoGainCost methods), which may speed up the search, without changing the final policy.

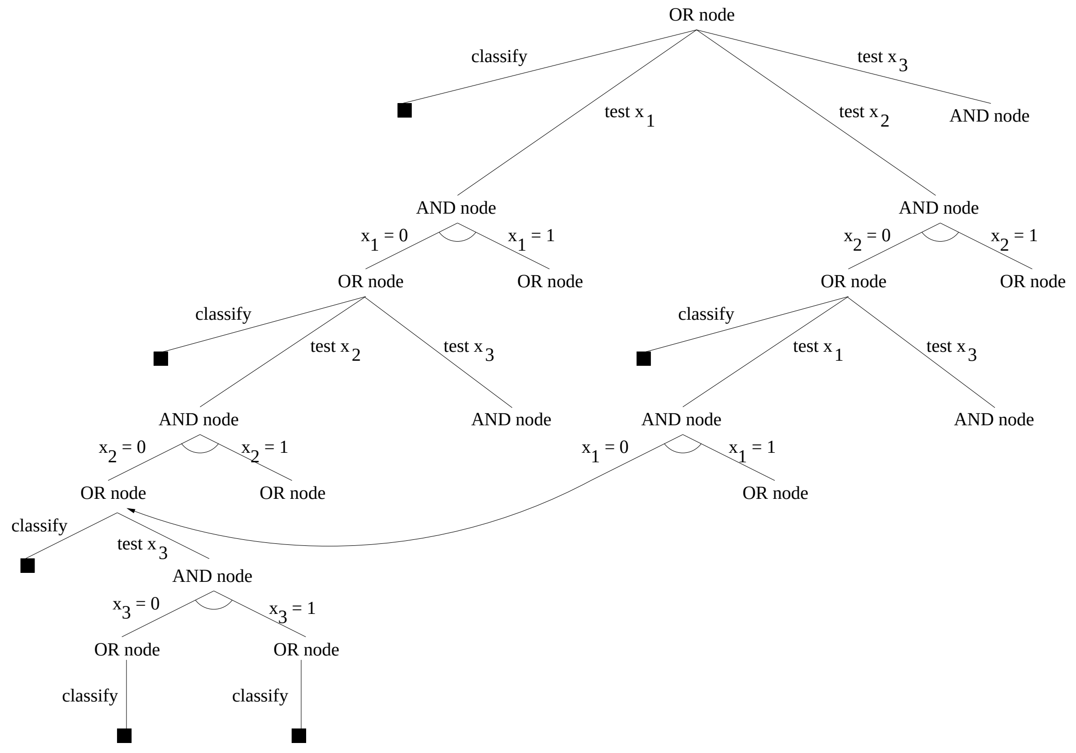


FIGURE 4.1: The AND/OR search space is a directed acyclic graph (DAG). This figure shows part of the AND/OR graph for a classification problem with 3 binary attributes (e.g., BMI, Glucose, and Insulin); "classify" is the terminal action choosing among several class labels (e.g., Healthy, Diabetes Type I, and Diabetes Type II). OR nodes have a choice between classifying and measuring one of the remaining attributes. If classification is chosen, the OR node is called a leaf node (marked by the solid square). AND nodes (marked by arcs) specify the outcomes of the tests; all test results must be considered. The OR node with state $\{x1 = 0, x_2 = 0\}$ was generated for the first time on the leftmost branch, by measuring first x_1, then x_2. The branch measuring $x_2 = 0$ then $x_1 = 0$ points to the same OR node. When the same test results are obtained in a different order, a single OR node is stored in the graph.

4.1 AND/OR Graphs

Figure 4.1 shows a simple AND/OR graph. An AND/OR graph is a directed graph composed of two kinds of nodes, AND nodes and OR nodes. The root of the graph is an OR node. Every OR node has out-going links to AND nodes which are called the *children* of the OR node. Every AND node has out-going links to OR nodes, which are called the *children* of the AND node.

In cost-sensitive learning, an OR node corresponds to a state s in the CSL MDP. It stores a representation of s, a current policy $\pi(s)$, which specifies what action to take in state s, and a current value function $V(s)$. (Below, we will describe other information stored in the OR nodes.) The root OR node corresponds to the start state $s_0 = \{\}$.

Each AND child of an OR node represents one of the possible measurement actions $x_n \in A(s)$ that can be taken in s. Each AND node stores $Q(s, x_n)$, the expected cost of measuring x_n and then continuing with the current policy. Each OR child of an AND node represents one of the possible states $s' = s \cup \{x_n = v\}$ that results from measuring attribute x_n; for each value v of the attribute there is a state s'.

If the current policy in an OR node chooses a classification action instead of a measurement action, then we will say that the OR node is *classified.* A classified OR node will always be classified into the best class according to the training data: $\pi(s) = best_f = \arg\min_{f_k} C(s, f_k)$.

A *complete* diagnostic policy consists of a subtree of the AND/OR graph. The subtree begins at the root OR node and contains only one of its AND children, the one corresponding to the measurement action x_n chosen at the root of the diagnostic policy. The subtree contains all of the children of the AND node (which correspond to the possible outcomes of the measurement action). This pattern continues: at each OR node, only one child is included; at each AND node, all of the children are included (hence the names!). The leaves of the diagnostic policy correspond to classified OR nodes.

The AND/OR graph search works by growing the AND/OR graph and updating the Q values in the AND nodes, the V values and the policy π in the OR nodes until a termination condition is satisfied. At termination, the subtree starting at the root node and selected by the current policy π in each OR node will be a complete optimal policy for the CSL problem.

The graph is initialized by creating the root node and AND children for each of the possible measurement actions. The graph is then expanded by iteratively choosing an *unexpanded AND node* (corresponding, say, to attribute x_n), creating an OR node for each possible outcome v of measuring x_n, and creating an unexpanded AND node for each possible child of those OR nodes. There are two exceptions to this: a) there are no possible AND children of an OR node, because all possible measurement actions have been performed, and b) classifying is the cheapest action in an OR node (though such an OR node may have AND nodes previously expanded and may also have unexpanded AND nodes). The resulting OR node is called a *leaf OR node*, and it must be classified. Hence, at all times, the leaves of the AND/OR graph consist either of unexpanded AND nodes or of leaf OR nodes.

The current diagnostic policy π is *incomplete* if it reaches some OR node s such that $\pi(s)$ chooses a measurement action that leads to an unexpanded AND node.

It is important to note that the AND/OR graph is not itself a tree. This is because there may be multiple paths from the root to an OR node. Consider, for example, an OR node corresponding to the state $\{x_1 = 0, x_2 = 0\}$. This node could be reached by a path from the root that first tests x_1 and then tests x_2 or by a path that first tests x_2 and then tests x_1. Hence, the AND/OR graph is a DAG. In general, if an OR node corresponds to a state s where n attributes have already been measured, then there can exist up to $n!$ paths from the root to s. Obviously, an important goal in AND/OR graph search is to find the optimal policy without completely expanding the entire graph.

Every state s corresponds to exactly one OR node in the AND/OR graph, and each state-action pair (s, x_n) corresponds to a child AND node. Because there is a one-to-one relationship between OR nodes and states, we will denote OR nodes by their states and use the terms "OR node" and "state" interchangeably. (We store the OR nodes in a hash table to avoid creating multiple OR nodes for a single state s.) Similarly, we will use the terms "AND node" and "state-action pair" interchangeably.

We visualize the AND/OR graph as growing downwards. Given a set of training examples, it is possible to "drop" them through the AND/OR graph as follows. All of the examples start at the root node. At each OR node, the examples are "cloned" and sent down *all* of the AND children. At each AND node, the examples are sent to the OR child whose state they match (an example matches a state if the example agrees with the attribute values defining the state). When multiple paths reach an OR node, the node will receive the same "cloned" training examples along all of them, and the "clones" can be reunited. Equivalently, we can view each OR node as storing pointers to the training examples that match its corresponding state.

Suppose that we have expanded the AND/OR graph completely, so that each leaf is a leaf OR node. We will initialize the current policy π in each node to be undefined. Now imagine that we have "dropped" a set of training examples through the graph. We can then compute the optimal policy and value function by performing value iteration over the AND/OR graph. Because the graph is acyclic, we can do this in a single pass. We start at the leaf OR nodes and classify each such node s into $best_f = \arg\min_{f_k} C(s, f_k)$. Then we interleave the following two steps in any order until we reach the root of the AND/OR graph:

- Find an AND node (s, x_n) all of whose OR children have defined policies π. Compute the Q value $Q(s, x_n) = C(s, x_n) + \sum_v P(x_n = v|s) \times V(s \cup \{x_n = v\})$.

- Find an OR node s all of whose AND children have computed their Q values. Compute the value of classifying s, $\min_{f_k} (Q(s, f_k) = C(s, f_k))$, and the value of measuring an attribute, $\min_{x_n \in A(s)} Q(s, x_n)$, and choose the action with the smallest Q value. Set the current policy $\pi(s)$ to this action.

The action $\pi(s_0)$ computed at the root of the AND/OR graph will be the root of an optimal diagnostic policy. The above computations are equivalent to ExpectiMin, by replacing AND nodes with EXPECT nodes, and OR nodes with MIN nodes.

The goal of the rest of this chapter is to describe algorithms that selectively expand the AND/OR graph and that selectively update the current policy π and the current

Q and V values so that the optimal policy can be found without expanding the entire AND/OR graph.

At various points in our description, it will be useful to ignore the AND nodes and view the AND/OR graph as a DAG of OR nodes. For this purpose, we will say that OR node s_1 is a *parent* of node s_2 if s_2 can be reached from s_1 by traversing exactly one AND node. Conversely, we will say that s_2 is a *successor* of s_1. If at least one AND node must be crossed to get from s_1 to s_2, then s_2 is a *descendant* of s_1, and s_1 is an *ancestor* of s_2.

4.2 AO* Algorithm

4.2.1 Overview of the AO Algorithm*

The AO* algorithm [50] computes an optimal solution graph of an AND/OR graph, given an admissible heuristic. A heuristic is admissible if it never over-estimates the optimal cost of getting from a state s to a terminal state.

During its search, AO* considers partial (or incomplete) policies in which not all nodes have been expanded. The AO* algorithm repeats the following steps: in the current best partial policy, it selects an AND node to expand; it expands it; and then it recomputes (bottom-up) the optimal value function and policy of the revised graph. The algorithm terminates when the best decision in all leaf OR nodes of the current policy is to classify.

In Nilsson's description of the AO* algorithm, at any given time, only one type of policy is considered. We call this policy the *optimistic policy*, π^{opt}. Its value function V^{opt} is a lower bound on V^*. This is enough to compute an optimal policy π^* [45]. The optimistic policy π^{opt} is a partial, incomplete policy because it does not end with all terminal leaf OR nodes, since some of the AND nodes are unexpanded. When π^{opt} is a complete policy, it is in fact an optimal policy π^*.

In our algorithms, it is useful to introduce a second policy, π^{real}, which is called the *realistic policy*. Its value function V^{real} is an upper bound on V^*. This policy is always a complete policy, so it is executable at any time after any iteration of AO*,

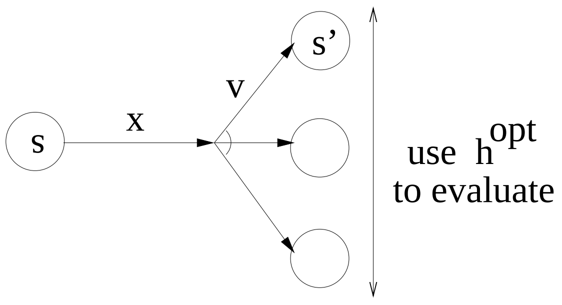


FIGURE 4.2: $Q^{opt}(s,x)$ for unexpanded AND node (s,x) is computed using one-step lookahead and h^{opt} to evaluate the resulting states s'. x is an attribute not yet measured in state s, and v is one of its values.

while the optimistic policy is only complete when AO* terminates. We will use this anytime feature of the realistic policy to find an optimal stopping point for the AO* algorithm in a version of early stopping. We also use the realistic values to help choose which AND nodes to expand.

Basically we expand π^{opt} of the OR node with the largest gap between V^{real} and V^{opt} (so with the largest uncertainty about V^*), weighted by the "popularity" of this node (how easy is it to reach it from the root, according to π^{opt}; note that this is a path, because in a given policy there is a single way of reaching an OR node from the root).

We next introduce the notion of admissible heuristic that estimates the values of unexpanded AND nodes, and thus guides the search process of which AND node to expand next.

4.2.2 Admissible Heuristic

Our admissible heuristic provides an optimistic estimate, $Q^{opt}(s,x)$, of the expected cost of an unexpanded AND node (s,x). It is based on an incomplete two-step lookahead search (see Figure 4.2). The first step of the lookahead search computes $Q^{opt}(s,x) = C(s,x) + \sum_{s'} P_{tr}(s'|s,x) \cdot h^{opt}(s')$. Here s' iterates over the states resulting from measuring attribute x. The second step of the lookahead is defined by the function $h^{opt}(s') = \min_{a' \in A(s')} C(s', a')$, which is the minimum over the cost of each classification action and the cost of each remaining attribute x' in s'. That is, rather than considering the states s'' that would result from measuring x', we only consider the cost of measuring x'. It follows immediately that $h^{opt}(s') \leq V^*(s')\ \forall s'$, because $C(s', x') \leq Q^*(s', x') = C(s', x') + \sum_{s''} P_{tr}(s''|s', x') \cdot V^*(s'')$. The key thing to notice is that the cost of measuring a single attribute x' is less than, or equal to, the cost of any policy that begins by measuring x', because the policy must pay the cost of at least one more action (classification or measurement) before entering the terminal state s_f.

4.2.3 Optimistic Values and Optimistic Policy

The definition of the optimistic Q value Q^{opt} can be extended to apply to all AND nodes as follows:

$$Q^{opt}(s,a) = \begin{cases} C(s,a) \\ \qquad \text{if } a = f \text{ (a classification action)} \\ C(s,a) + \sum_{s'} P_{tr}(s'|s,a) \cdot h^{opt}(s') \\ \qquad \text{if } (s,a) \text{ is unexpanded} \\ C(s,a) + \sum_{s'} P_{tr}(s'|s,a) \cdot V^{opt}(s') \\ \qquad \text{if } (s,a) \text{ is expanded} \end{cases}$$

where $V^{opt}(s) \overset{\text{def}}{=} \min_{a \in A(s)} Q^{opt}(s,a)$.

The optimistic policy is $\pi^{opt}(s) = \operatorname{argmin}_{a \in A(s)} Q^{opt}(s,a)$. Every OR node s stores its optimistic value $V^{opt}(s)$ and policy $\pi^{opt}(s)$, and every AND node (s,a) stores $Q^{opt}(s,a)$. Theorem 4.2.1 proves that Q^{opt} and V^{opt} form an admissible heuristic.

Theorem 4.2.1 *For all states s and all actions $a \in A(s)$, $Q^{opt}(s,a) \leq Q^*(s,a)$, and $V^{opt}(s) \leq V^*(s)$.*

Proof by induction:

Base case for Q^{opt}: If a is a classification action f, then $Q^{opt}(s,f) = C(s,f) = Q^*(s,f)$. If (s,a) is unexpanded, then $Q^{opt}(s,a) = C(s,a) + \sum_{s'} P_{tr}(s'|s,a) \cdot h^{opt}(s') \leq C(s,a) + \sum_{s'} P_{tr}(s'|s,a) \cdot V^*(s') = Q^*(s,a)$, because $h^{opt}(s') \leq V^*(s')$.

Base case for V^{opt}: If all attributes were measured in state s, then classifying is only possibility, so $V^{opt}(s) = \min_f C(s,f) = V^*(s)$.

Let s_u be any state all of whose AND nodes (s_u, a) are unexpanded. Then $V^{opt}(s_u) \leq V^*(s_u)$ follows from the base case of Q^{opt}.

Any other state s_e in the graph has at least one AND node (s_e, a) previously expanded. Let s_c be any of the resulting states of (s_e, a).

Induction hypothesis: If $V^{opt}(s_c) \leq V^*(s_c), \forall s_c$, then $Q^{opt}(s_e,a) \leq Q^*(s_e,a), \forall a \in A(s_e)$, and $V^{opt}(s_e) \leq V^*(s_e)$.

We only need to consider the case of an expanded action $a \in A(s_e)$, because the other situations are covered by the base case of Q^{opt}. By definition, $Q^{opt}(s_e,a) = C(s_e,a) + \sum_{s'} P_{tr}(s'|s_e,a) \cdot V^{opt}(s')$. Because s' is one of the resulting states of (s_e,a), we can apply the induction hypothesis with $s_c = s'$, so $V^{opt}(s') \leq V^*(s')$, hence $Q^{opt}(s_e,a) \leq Q^*(s_e,a)$. It follows that $Q^{opt}(s_e,a) \leq Q^*(s_e,a), \forall a \in A(s_e)$, and $V^{opt}(s_e) \leq V^*(s_e)$. **Q.E.D.**

4.2.4 Realistic Values and Realistic Policy

We also introduce an upper bound, $V^{real}(s)$, which is an overestimate of the value of the optimal policy rooted at s. Every OR node s stores a realistic value $V^{real}(s)$ and policy $\pi^{real}(s)$, and every AND node (s,a) stores a realistic Q value, $Q^{real}(s,a)$. For $a \in A(s)$ define

$$Q^{real}(s,a) = \begin{cases} C(s,a) & \\ \quad \text{if } a = f \text{ (a classification action)} & \\ C(s,a) + \sum_{s'} P_{tr}(s'|s,a) \cdot V^{real}(s') & \\ \quad \text{if } (s,a) \text{ is expanded} & \\ \text{ignore} \quad \text{if } (s,a) \text{ is unexpanded} & \end{cases}$$

and $V^{real}(s) = \min_{a \in A'(s)} Q^{real}(s,a)$, where the set $A'(s)$ is $A(s)$ without the unexpanded actions. The realistic policy is $\pi^{real}(s) = \operatorname{argmin}_{a \in A'(s)} Q^{real}(s,a)$.

In the current graph expanded by AO*, assume that we ignore all unexpanded AND nodes (s,a). We call this graph the *realistic* graph. The current realistic policy is the best policy (according to the training data) from this realistic graph.

Theorem 4.2.2 *The realistic value V^{real} is an upper bound: $V^*(s) \leq V^{real}(s), \forall s$.*

Proof: Base case: By definition, a leaf OR node s in the realistic graph has $\pi^{real}(s) = f$, where f is the best classification action in s. Therefore $V^{real}(s) = C(s,f) \geq V^*(s)$, because we ignore unexpanded AND nodes (s,x_n), and it could well be that $C(s,f) \geq Q^*(s,x_n)$. Induction step: The internal nodes in the graph compute their realistic values using a one-step Bellman backup based on realistic values of the next states, $V^{real}(s) = \min\limits_{a \in A'(s)} [C(s,a) + \sum\limits_{s'} P_{tr}(s'|s,a) \cdot V^{real}(s')]$. **Q.E.D.**

Corollary 4.2.3 *If a is an expanded action in state s or a classification action, then $Q^*(s,a) \leq Q^{real}(s,a)$.*

Proof: If a is a classification action f, then $Q^{real}(s,f) = C(s,f) = Q^*(s,f)$. If a is an expanded action, the proof is immediate from the definition of $Q^{real}(s,a)$ based on one-step lookahead, and using Theorem 4.2.2 in the resulting states s'. **Q.E.D.**

4.2.5 Selecting a Node for Expansion

In the current optimistic policy π^{opt}, we choose to expand the unexpanded AND node $(s, \pi^{opt}(s))$ with the largest impact on the root, defined as

$$\operatorname*{argmax}_s \, [V^{real}(s) - V^{opt}(s)] \cdot P_{reach}(s|\pi^{opt}),$$

where $P_{reach}(s|\pi^{opt})$ is the probability of reaching state s from the root, following policy π^{opt}. The difference $V^{real}(s) - V^{opt}(s)$ tells how much we expect the value of state s to change if we expand $\pi^{opt}(s)$.

The rationale for this heuristic is based on the observation that AO* terminates when $V^{opt}(s_0) = V^{real}(s_0)$. Therefore, we want to expand the node that makes the biggest step toward this goal.

4.2.6 Our Implementation of AO* (High Level)

Our implementation of AO* is the following:

```
repeat
    select an AND node $(s, a)$ to expand (using $\pi^{opt}, V^{opt}, V^{real}$).
    expand $(s, a)$.
    do bottom-up updates of $Q^{opt}, V^{opt}, \pi^{opt}$ and of $Q^{real}, V^{real}, \pi^{real}$.
until there are no unexpanded nodes reachable by $\pi^{opt}$.
```

It is also possible to expand more AND nodes per iteration. We did not explore this.

The only way the value function of a state s changes is if one of its descendants s_d has changed its value.

Theorem 4.2.4 *For all states s in the graph expanded so far, subsequent iterations of AO* increase their optimistic values: $V_i^{opt}(s) \leq V_{i+1}^{opt}(s)$.*

Proof: For optimistic values, any change in V^{opt} is triggered by the expansion of an AND node. Say unexpanded AND node (s_d, a) was chosen for expansion during iteration i of the AO* algorithm; then $\pi_i^{opt}(s_d) = a$ and $V_i^{opt}(s_d) = Q_i^{opt}(s_d, a) = C(s_d, a) + \sum_{s'} P_{tr}(s'|s_d, a) \cdot h^{opt}(s')$. After expansion, $Q_{i+1}^{opt}(s_d, a) = C(s_d, a) + \sum_{s'} P_{tr}(s'|s_d, a) \cdot V_{i+1}^{opt}(s')$, where s' are the states in the newly created OR nodes (i.e., the children of the AND node (s_d, a)), so all their AND node children, if any, are currently unexpanded. Therefore $V_{i+1}^{opt}(s') = \min_{a' \in A(s')} Q_{i+1}^{opt}(s', a') \geq \min_{a' \in A(s')} C(s', a') = h^{opt}(s')$, which implies $Q_i^{opt}(s_d, a) \leq Q_{i+1}^{opt}(s_d, a)$. Since only the Q^{opt} value of AND node (s_d, a) has changed in state s_d, we have $V_i^{opt}(s_d) \leq V_{i+1}^{opt}(s_d)$ and this can only trigger an increase (or no change) in V^{opt} in the ancestors of state s_d. **Q.E.D.**

Theorem 4.2.5 *For all states s in the graph expanded so far, subsequent iterations of AO^* decrease their realistic values: $V_{i+1}^{real}(s) \leq V_i^{real}(s)$.*

Proof: For V^{real}, the change comes after the expansion of an unexpanded AND node (s_d, a_u). $V_i^{real}(s_d) = \min_{a \in A'(s_d)} Q_i^{real}(s_d, a)$, where the set of actions $A'(s_d)$ includes classification actions and previously expanded attributes (if any). After the expansion of a_u, $A'(s_d)$ will include a_u, and $V_{i+1}^{real}(s_d) = \min(V_i^{real}(s_d), Q_{i+1}^{real}(s_d, a_u)) \leq V_i^{real}(s_d)$ and this can only trigger a decrease (or no change) in V^{real} in the ancestors of state s_d. **Q.E.D.**

Theorem 4.2.6 *For an unexpanded AND node (s, a), if $V^{real}(s) < Q^{opt}(s, a)$, then action a does not need to be expanded (so it can be pruned).*

Proof: In this case, $V^{opt}(s) \leq V^*(s) \leq V^{real}(s) < Q^{opt}(s, a) \leq Q^*(s, a)$. So $V^{opt}(s) < Q^{opt}(s, a) \Rightarrow \pi^{opt}(s) \neq a$, and $V^*(s) < Q^*(s, a) \Rightarrow \pi^*(s) \neq a$. This theorem shows that our heuristic provides a cutoff in node expansions when there exists a complete policy π^{real} with smaller expected cost. Note that in future iterations of AO*, V^{real} can stay the same or it can decrease (Theorem 4.2.5), so it remains cheaper than $Q^{opt}(s, a)$. **Q.E.D.**

The AO* algorithm can be viewed as an "anytime" algorithm. If we want to stop the algorithm after a certain number of nodes have been expanded or if we run out of memory, then we can return the realistic policy π^{real} computed up to that point.

4.2.7 AO* for CSL Problems, With an Admissible Heuristic, Converges to the Optimal Value Function V^*

As more nodes are expanded, the optimistic values V^{opt} increase, becoming tighter lower bounds to the optimal values V^*, and the realistic values V^{real} decrease, becoming tighter upper bounds. They converge to the value of the optimal policy on the training data: $V^{opt}(s) = V^{real}(s) = V^*(s)$, $\forall s$ reachable by π^*. When π^{opt} becomes a complete policy (so all the leaf OR nodes in π^{opt} are classified), it is equal to π^{real} because it is the best complete policy in the graph, and it is actually π^*.

Theorem 4.2.7 *If h^{opt} is an admissible heuristic, then the AO^* algorithm for CSL problems converges to an optimal policy π^*, and $V^{opt}(s_0) = V^*(s_0)$.*

The proof is based solely on V^{opt}, because V^{real} is not necessary for the convergence of AO*. The stopping condition does not depend on V^{real}, and we could expand AND nodes $(s, \pi^{opt}(s))$ in state s with $\max_s V^{opt}(s) \times P_{reach}(s|\pi^{opt})$.

In the worst case, AO* does exhaustive search (in a finite graph) until its π^{opt} becomes a complete policy, so the algorithm always converges.

When AO* converges, its π^{opt} is a complete policy. In the following, by $V^{opt}(s)$ we denote the optimistic values, when the algorithm terminates, of the states s reachable by π^{opt}.

Proof by induction:

Base case

The leaf OR nodes s of π^{opt} are classified, so $V^{opt}(s) = C(s, best_f) = \min_{a \in A(s)} Q^{opt}(s, a)$. Therefore $C(s, best_f) \leq Q^{opt}(s, a), \forall a \in A(s)$. In Theorem 4.2.1 we proved that because h^{opt} is an admissible heuristic, $Q^{opt}(s, a) \leq Q^*(s, a), \forall a \in A(s)$. Therefore $C(s, best_f) \leq Q^*(s, a), \forall a \in A(s)$, which implies $V^*(s) = C(s, best_f) = V^{opt}(s)$. This proves that in the leaves s of π^{opt}, $V^{opt}(s) = V^*(s)$.

Induction hypothesis

Let s be a state in an internal node of π^{opt}, such that all its successor OR nodes s' through π^{opt} have $V^{opt}(s') = V^*(s')$. Then $V^{opt}(s) = V^*(s)$.

Let $\pi^{opt}(s) = a$, therefore $V^{opt}(s) = Q^{opt}(s, a)$. Because s is an internal node of π^{opt}, it follows that (s, a) was expanded, and by definition $Q^{opt}(s, a) = C(s, a) + \sum_{s'} P_{tr}(s'|s, a) \times V^{opt}(s')$. Because the induction hypothesis assumes that $V^{opt}(s') = V^*(s')$, we have $Q^{opt}(s, a) = Q^*(s, a)$.

$V^{opt}(s) = Q^{opt}(s, a)$ implies $Q^{opt}(s, a) \leq Q^{opt}(s, a'), \forall a' \in A(s)$. Because $Q^{opt}(s, a') \leq Q^*(s, a'), \forall a' \in A(s)$, according to Theorem 4.2.1, it follows that $Q^{opt}(s, a) = Q^*(s, a) \leq Q^{opt}(s, a') \leq Q^*(s, a'), \forall a' \in A(s)$. Therefore $Q^*(s, a) \leq Q^*(s, a'), \forall a' \in A(s)$, so $V^*(s) = \min_{a' \in A(s)} Q^*(s, a') = Q^*(s, a)$. Since $V^{opt}(s) = Q^{opt}(s, a) \Rightarrow V^{opt}(s) = V^*(s)$.

We proved that for all states s reached by the optimistic policy π^{opt} when AO* terminates, $V^{opt}(s) = V^*(s)$. Therefore this policy is an optimal policy. Because the initial state s_0 is part of any policy, therefore also of this policy, we have $V^{opt}(s_0) = V^*(s_0)$. **Q.E.D.**

Corollary 4.2.8 *When algorithm AO* terminates, $V^{opt}(s) = V^{real}(s) = V^*(s)$ for all states s reached by π^*, and $\pi^{opt}(s) = \pi^{real}(s) = \pi^*(s)$.*

Proof: The proof that $\pi^{opt}(s) = \pi^{real}(s)$ at convergence follows immediately from the fact that π^{opt} is now a complete policy. More precisely, it is the best complete policy in the graph expanded so far. Therefore it is equal to the realistic policy (since both policies break ties in the same way). And we already showed that the optimistic policy is an optimal policy π^*. **Q.E.D.**

4.2.8 Pseudocode and Implementation Details for the AO* Algorithm

In our current implementation, we store more information than necessary, but this makes the description and the implementation clearer. We store Q, V, π, though it would be enough to store just the Q function, because the policy π and the V function can be computed from Q.

4.2.8.1 Data Structures

An OR node stores

- the corresponding state s,
- the best classification action in s, $best_f$ according to the training data, and its expected cost $C(s, best_f) = \min_{f_k} C(s, f_k)$ (We store these so we do not have to compute them repeatedly.),
- the current optimistic policy, $\pi^{opt}(s)$, and its optimistic value, $V^{opt}(s)$,
- the current realistic policy, $\pi^{real}(s)$, and its realistic value, $V^{real}(s)$,

- a flag that marks this node as solved,
- a list of (pointers to) children AND nodes, for all attributes not yet measured in s, and
- a list of (pointers to) parent OR nodes, along with markers set to 1 if s is reached by $\pi^{opt}(parent)$.

Because our graph is a DAG and we do not want to generate the same OR node multiple times, the OR nodes are stored in a hash table. Before generating a new OR node with state s, we double check, using the measured attributes of state s, that such an OR node does not exist already. If it does, we only update the links to its parents.

An AND node (s, a) stores

- the action a that measures attribute x_n,
- a flag that marks this node as expanded or not (it also marks if the AND node was pruned by the statistical pruning regularizer SP, see below),
- the optimistic Q-value, $Q^{opt}(s, a)$,
- the realistic Q-value, $Q^{real}(s, a)$,
- a vector of probabilities for each of the possible values v of attribute x_n, $P(x_n = v|s)$, computed from the training data (We store these probabilities because they will be used in future updates of the value functions.), and
- a vector of (pointers to) children OR nodes corresponding to states $s \cup \{x_n = v\}$.

An AND node (s, x_n) does not need to store the state s, because it is stored in the parent OR node.

TABLE 4.1: Pseudocode for the AO* algorithm.

```
function AO*(int Mem-limit) returns a complete policy.

iteration $i = 0$;
Memory = 0;
create hash-table;
(1) OR node * root = create-OR-node($s_0$, (OR Node *) 0, hash-table, Memory);
(2) while ((Memory < Mem-limit) && (root not solved)){
        i++;
(3)     in current $\pi^{opt}$, select fringe OR node $s$ with
            AND node $(s, a)$ to expand $(\pi^{opt}(s) = a)$.
(4)     expand AND node $(s, a)$ by creating children OR nodes $s'$.
        update $A'(s)$ with the newly expanded action $a$, $A'(s) = A'(s) \cup \{a\}$.
(5)     do bottom-up updates of $Q^{opt}, V^{opt}, \pi^{opt}$ for $s$ and its ancestors.
(6)     do bottom-up updates of $Q^{real}, V^{real}, \pi^{real}$ for $s$ and its ancestors.
}
return last $\pi^{real}$.
```

4.2.8.2 Pseudocode

Table 4.1 gives the pseudocode for the AO* algorithm, and we will describe each step of it. Step (1) is described in Table 4.2, step (4) in Table 4.3 and steps (5) and (6) in Tables 4.4 and 4.5.

Step(1): Creating a New OR Node

First we compute the classification action $best_f$ with the minimum expected cost in state s, using the class probabilities estimated from the training data (see equation 2.7, $C(s, f_k) = \sum_y P(y|s) \cdot MC(f_k, y)$). We initialize the realistic policy to be $best_f$ and

TABLE 4.2: Creating a new OR node. If $s = s_0$ (for the root node), there is no parent.

```
**function** create-OR-node(state $s$, OR node * parent, hash-table, int & Memory)
**returns** an OR node.

compute $best_f$ in $s$, $best_f = \arg\min_{f_k} C(s, f_k)$,
    and set $\pi^{real}(s) = best_f, V^{real}(s) = C(s, best_f)$.
$A(s) = \{$classification actions and attributes not measured in $s\}$.
$A'(s) = \{$classification actions$\}$.
for every attribute $a$ not measured in $s$, compute
    $Q^{opt}(s,a) = C(s,a) + \sum_{s'} P_{tr}(s'|s,a) \cdot h^{opt}(s')$.
$\pi^{opt}(s) = \arg\min_{a \in A(s)} Q^{opt}(s,a)$, $V^{opt}(s) = \min_{a \in A(s)} Q^{opt}(s,a)$.
**if** ($\pi^{opt} = best_f$)
    solved = 1.
**else**
    solved = 0;
    for every attribute $a$ not measured in $s$,
        **if** $Q^{opt}(s,a) < C(s, best_f)$
            create AND node $(s,a)$.
        **else**
            remove $a$ from $A(s)$.
**if** (parent)
    add parent to the list of parents.
create OR node $n$.
store pointer to OR node $n$ in the hash-table.
update Memory by counting memory for OR node $n$
    and its children AND nodes.
return OR node $n$.
```

the realistic value $V^{real}(s)$ to be its cost.

Then, for every attribute a not yet measured in s (if any), we compute $Q^{opt}(s,a)$ using the optimistic heuristic h^{opt} in the resulting states s'. Then we compute the optimistic policy choosing between the best classification action (recall that $Q^{opt}(s, best_f) = C(s, best_f)$) and the attribute with the smallest heuristic estimate. We break ties following the rules in section 4.2.8.3.

If $\pi^{opt} = best_f$ (so $V^{opt}(s) = C(s, best_f)$), then we do not need to store any of the AND nodes (here, the optimistic heuristic provides some cutoff), and the newly created OR node will be marked as solved. Otherwise, a new AND node is created for each attribute a whose $Q^{opt}(s,a)$ is strictly less than the expected cost of $best_f$. If $C(s, best_f) \leq Q^{opt}(s,a)$, then the AND node (s,a) does not need to be stored (*admissible heuristic cutoff*), since according to Theorem 4.2.1 $Q^{opt}(s,a) \leq Q^*(s,a)$, therefore $C(s, best_f) \leq Q^*(s,a)$, and because we break ties in favor of classifying, $\pi^*(s) \neq a$. If such an (s,a) were to be expanded, its $Q^{opt}(s,a)$ could only increase, and it will never beat $C(s, best_f)$, so a will never become $\pi^{opt}(s)$.

When creating an AND node (s,a), we set its action a, we mark it as unexpanded (expanded $= 0$), and we store its computed $Q^{opt}(s,a)$, we store a default value for $Q^{real}(s,a)$ (since the AND node is unexpanded, its realistic value is not defined), and we compute the probability distribution $P(a = v|s)$ from the training data.

If the call to create-OR-node specifies a parent, then we add this parent to the list of parents of the new OR node. We finally create the new OR node with state s, $best_f$ and its expected cost, π^{opt} and V^{opt}, π^{real} and V^{real}, solved, children AND nodes and updated list of parents. We add the newly created OR node to the hash table (using the values of the measured attributes in state s as the hashing function). The global variable Memory keeps track of the total amount of memory consumed by the AND/OR graph. When updating Memory, we count the memory used by this OR node, its AND children (if any), the memory to store a link to its parent and to store a pointer to this OR node in the hash-table.

The root gets added to the hash-table as the first OR node created. For all other

OR nodes, we first check that they are not already in the hash-table, before storing them.

Step(2): OR Node Becomes Solved

In the following, we explain what it means for an OR node to be solved. This is a general concept, though in step (2) of Table 4.1 it is applied for the root node. An OR node becomes *solved* when all its children OR nodes, reached by π^{opt}, are solved. The base case for this recursive definition is a leaf OR node with state s, which is solved when $\pi^{opt}(s) = best_f$. This leaf OR node is called a *terminal leaf*, because the classification action deterministically transitions from s to the terminal state s_f.

Definition 4.2.1 *An OR node with state s is solved if it is a leaf OR node and $\pi^{opt}(s) = best_f$, or if it is an internal OR node and all its children OR nodes, reached by π^{opt}, are solved.*

Once an OR node s becomes solved, it stays solved. Indeed, its optimistic policy is a complete policy, the best one in the subgraph rooted at this OR node. With a proof similar to Theorem 4.2.7, it can be shown by induction that $\pi^{opt}(s) = \pi^*(s)$, starting with the leaf nodes that have optimal classification actions. After updating the OR node's realistic value, we will also have $\pi^{real}(s) = \pi^{opt}(s) = \pi^*(s)$, because π^{opt} is the best complete policy in the graph expanded so far, and therefore it is the realistic policy. The solved node stays solved because its choice for π^{opt} will not be changed, as we see in the code for selecting the AND node to expand.

Step(3): Select Node to Expand

We traverse the current optimistic policy π^{opt} depth-first, keeping track of the leaf OR node s with unexpanded $\pi^{opt}(s)$ having the largest score

$$\left[V^{real}(s) - V^{opt}(s)\right] \cdot P_{reach}(s|\pi^{opt}).$$

In fact, we are interested in the leaf OR node s, though we expand its AND node $(s, \pi^{opt}(s))$; the reason for this will become apparent in the description of the next steps of the AO* algorithm.

The OR node returned by the depth-first search of π^{opt} is called a *fringe* node. Since the root is not solved, there must be at least one unexpanded AND node in π^{opt}, so the fringe node exists. In case there are multiple leaf nodes tied for the best score, we return the first one reached in the depth-first traversal of π^{opt}, though we do have to visit all (nonterminal) leaf nodes of π^{opt}.

In general, note that an OR node s is a leaf with respect to a particular policy π, in the sense that the AND node $(s, \pi(s))$ is unexpanded, though in the graph this OR node may have other AND child nodes, measuring other attributes, already expanded.

$P_{reach}(s|\pi^{opt})$ can be computed by counting the number of training examples reaching state s divided by the total number of training examples. This is due to the fact that in any policy π reaching state s, the probability $P_{reach}(s|\pi)$ is independent of the order in which the attributes in s are measured. $P_{reach}(s|\pi)$ is independent of the policy — the proof is immediate using the chain rule for probabilities and the commutativity and associativity of the multiplication operation. In fact, $P_{reach}(s|\pi)$ is equal to the state probability $P(s)$, defined as the fraction of training examples that match state s. So all paths reaching a state s from the root have the same probability.

We can stop the search down a branch of π^{opt} if we find a solved OR node, because the optimistic policy under that node is complete (all its leaf OR nodes are classified), therefore there are no unexpanded AND nodes in it. This proves that once an OR node becomes solved, it stays solved. It also makes the implementation more efficient.

Step(4): Expand AND Node

Table 4.3 shows the details for expanding an AND node (s, a). The probability $P(a = v|s)$ has been computed and stored in the AND node at its creation. If the child OR node with state $s' = s \cup \{a = v\}$ was already created, then it is stored in the hash-table, and it is sufficient to mark the connector from the child OR node to its parent fringe OR node as being part of the optimistic policy (this is useful for future updates of optimistic values and policy). Otherwise, we need to create a new

TABLE 4.3: Expanding AND node (s, a) of fringe OR node with state s.

```
function expand-AND-node(OR node * fringe, hash-table, int & Memory).

s = state of fringe OR node.
a = $\pi^{opt}(s)$.
for every value $v$ of attribute $a$
  if ($P(a = v|s) > 0$)
    let state $s' = s \cup \{a = v\}$.
    if (state $s'$ is already in the hash-table)
      let child be its OR node.
      add fringe to the child's list of parent OR nodes.
      mark the link/connector child-fringe as being part of $\pi^{opt}$.
      update Memory to count the link to this new parent.
    else
      OR node * child = create-OR-node($s'$, fringe, hash-table, Memory).
    add child with state $s'$ as a child OR node for AND node $(s, a)$.
    mark AND node $(s, a)$ as expanded (expanded = 1).
  else
    do not store anything.
    when evaluating policy on test or validation data, classify examples
    reaching $s' = s \cup \{a = v\}$ into the same class as $best_f$ in state $s$.
```

OR node. In either case, the child OR node with state s' is added below the AND node (s, a), corresponding to the outcome $a = v$.

Note that if the probability of reaching state s' from state s is zero according to the training data ($P_{tr}(s'|s, a) = P(a = v|s) = 0$), then we do not add an OR node. When we evaluate additional data points and if some of them travel down the policy from state s to state s', we will classify them into the best classification action of state s.

Step(5): Bottom-up Updates of Optimistic Values and Policy

For both optimistic and realistic updates, $V_{i+1}(s)$ are the new values at the end of iteration $i + 1$. All states s in the graph whose values were not modified in iteration $i + 1$ have $V_{i+1}(s) = V_i(s)$. The same holds true for Q values and policies. The changes propagate from bottom-up, so when updating the value of state s we already computed the value V_{i+1} of its children states. The action set $A(s)$ of valid actions in any state s does not change from one iteration of AO* to another.

Table 4.4 details the updates of optimistic values and policies. Starting with the fringe node, we propagate changes in the optimistic values upward in the graph by pushing OR nodes, which can be affected by these changes, onto a queue (called the optimistic queue in order to differentiate it from the queue used for realistic updates).

Only the Q values for expanded AND nodes need to be updated, based on the already computed V^{opt} of their children OR nodes. For every OR node in the optimistic queue, it is necessary to perform the updates for all its expanded AND nodes.

After updating the Q values, we recompute π^{opt} and V^{opt}. If the optimistic policy has changed, we need to mark its new successor OR nodes as being reachable by π^{opt}, and unmark the connectors for the old policy π^{opt}. It is important to have a marker between a child OR node and its parent OR node, if the child is reached from the parent by following π^{opt}. These markers, which are similar to reversed links, will be useful after children OR nodes update their V^{opt} and these changes need to be propagated to their marked parents, which will be pushed onto the optimistic queue.

TABLE 4.4: Updating V^{opt}, Q^{opt} and π^{opt} after the expansion of π^{opt} in fringe OR node.

```
function optimistic-update(OR node * fringe).

push fringe OR node onto the optimistic queue.
while (optimistic queue not empty){
   pop OR node $n$ with state $s$ from the optimistic queue.
   recompute $Q^{opt}(s,a)$ for all expanded AND nodes $(s,a)$,
     $Q^{opt}_{i+1}(s,a) := C(s,a) + \sum_{s'} P_{tr}(s'|s,a) \cdot V^{opt}_{i+1}(s')$.
   for unexpanded AND nodes $(s,a)$, and for $a = best_f$,
     $Q^{opt}_{i+1}(s,a) := Q^{opt}_{i}(s,a)$.
   $\pi^{opt}_{i+1}(s) := \arg\min_{a \in A(s)} Q^{opt}_{i+1}(s,a)$, $V^{opt}_{i+1}(s) := \min_{a \in A(s)} Q^{opt}_{i+1}(s,a)$.
   if ($\pi^{opt}_{i+1}(s) = best_f$)
     mark OR node $n$ as solved.
   else
     if (AND node $(s, \pi^{opt}_{i+1}(s))$ is expanded)
       mark all its child OR nodes as being reachable by $\pi^{opt}$ from node $n$.
     if ($(\pi^{opt}_{i}(s)) \neq \pi^{opt}_{i+1}(s)))$
       mark all the child OR nodes of $\pi^{opt}_{i}(s)$ as unreachable by $\pi^{opt}$ from
       node $n$.
     if (all successor OR nodes through $\pi^{opt}_{i+1}$ are solved)
       label OR node $n$ as solved.
     if ((OR node $n$ solved) or ($V^{opt}_{i+1}(s) > V^{opt}_{i}(s)$))
       push onto the optimistic queue all marked parents of OR node $n$.
}
```

If an OR node becomes solved, or its V^{opt} changes (which can only be an increase in value, according to Theorem 4.2.4), then all its marked parents get pushed onto the optimistic queue. The connectors to the parent OR nodes were marked as being part of π^{opt} in previous iterations of AO*. Such a marked OR node gets pushed only once, per iteration, onto the optimistic queue, because its π^{opt} specifies only one action, therefore a single child OR node will push it.

The optimistic updates are triggered by expanding an AND node of the fringe OR node s_d. After expanding the AND node (s_d, a) by creating its OR children, we update its $Q^{opt}(s_d, a)$ value (which was initially computed using the admissible heuristic h^{opt}). If this leads to an increase in $V^{opt}(s_d)$, then this increased value needs to be propagated in the graph expanded so far, to all the ancestors of the fringe OR node that can be affected by it.

The trick that saves computation time, instead of updating all ancestors of state s_d, is to update only those ancestors that reach state s_d through their optimistic policies. Because their descendants' optimistic values have increased (assuming $V^{opt}(s_d)$ has increased), they may only influence V^{opt} of their ancestors if they are reached by optimistic policies, otherwise there already exist other actions with smaller Q^{opt} than theirs. This efficient way of computing V^{opt} is correct because at the end of each AO* iteration, every OR node in the graph has the correct π^{opt} and V^{opt}.

If $V_i^{opt}(s_d) < V_{i+1}^{opt}(s_d)$, we push onto the optimistic queue each parent OR node of state s_d whose link/connector is marked as being part of some π^{opt}. These parents, when their turn to be updated comes, will push their marked parents onto the queue, and this continues until the queue is empty.

Step(6): Bottom-up Updates of Realistic Values and Policy

Table 4.5 details the updates of realistic values and policies. After expanding an AND node (s, a) and computing its $Q^{real}(s, a)$, it is necessary to recompute the $V^{real}(s)$ value of its parent OR node. If this results in a decrease (Theorem 4.2.5 shows that realistic values decrease or stay the same), then this needs to be propagated to the

TABLE 4.5: Updating V^{real}, Q^{real} and π^{real} after the expansion of π^{opt} in fringe OR node.

```
**function** realistic-update(OR node * fringe).

push fringe OR node onto the realistic queue.
while (realistic queue not empty){
    pop OR node $n$ with state $s$ from the realistic queue.
    recompute $Q^{real}(s,a)$ for all expanded AND nodes $(s,a)$,
        $Q^{real}_{i+1}(s,a) := C(s,a) + \sum_{s'} P_{tr}(s'|s,a) \cdot V^{real}_{i+1}(s')$.
    ignore unexpanded AND nodes $(s,a)$.
    for $a = best_f$, $Q^{real}_{i+1}(s,a) := Q^{real}_{i}(s,a) = C(s, best_f)$.
    $\pi^{real}_{i+1}(s) = \arg\min_{a \in A'(s)} Q^{real}_{i+1}(s,a)$, $V^{real}_{i+1}(s) = \min_{a \in A'(s)} Q^{real}_{i+1}(s,a)$.
    if $(V^{real}_{i+1}(s) < V^{real}_{i}(s))$
        push onto the realistic queue all parents of OR node $n$.
}
```

ancestor OR nodes of OR node s. We again employ a queue to do this, called the *realistic queue*. Unlike with V^{opt}, we cannot employ the time-saving trick of only updating OR nodes that reach the fringe OR node by following an optimistic policy. When $V^{real}(s)$ decreases, this, in turn, decreases the Q^{real} in ancestors of s, and the corresponding actions may become π^{real} in their nodes. Therefore we must push all of the ancestor OR nodes of state s onto the realistic queue. It is possible for an OR node to be pushed more than once, per iteration, onto the realistic queue.

4.2.8.3 Tie Breaking

As in the previous chapter on greedy search for diagnostic policies, we *break ties* following this rule: classifying is preferred to testing attributes, and in case of ties among classification actions we prefer the one with the lowest index. In case of ties among attributes, we break the tie in favor of the attribute with the lowest index. Both optimistic and realistic policies follow these rules for breaking ties.

The code could be improved by selecting a solved AND node in a tie, where an AND node becomes solved when all its children OR nodes are solved.

This concludes our description of the AO* algorithm. The Appendix A includes additional notes on our AO* implementation.

4.3 Regularizers

In order to reduce the risk of overfitting the training data, we introduce several regularization techniques. Overfitting tends to occur when the learning algorithm extracts too much detailed information from the training data. This can occur when the learning algorithm computes too many probabilities from the data or when the learning algorithm bases its decisions on only a small number of training examples. The most extreme case arises when AO* decides that a node in the graph has probability 0 of being reached because none of the training examples reach that node.

The risk of overfitting increases as AO* expands more nodes. There are several reasons. First, each additional expansion requires computing additional probabilities from the training data. Second, additional expansions may cause the AND/OR graph to grow deeper, which typically means that fewer training examples are reaching the newly expanded nodes, so the computed probabilities are based on fewer training examples. Hence, many of the regularizers that we describe in this section act to prevent AO* from expanding all of the nodes it would otherwise expand.

Recall that an example *matches* a state s if the example agrees with the attribute values defining s, as defined in Chapter 2. This notion is used in several of the regularizers below.

4.3.1 Memory Limit

One way to regularize AO* is to set a limit on the amount of memory that the AND/OR graph can consume. When this memory limit is reached, we terminate AO* and return the current realistic policy π^{real}.

4.3.2 Laplace Correction (L)

A second way to regularize AO* is to prevent it from interpreting the complete absence of examples at a node as implying that the node is reached with probability 0. This can be accomplished by applying a Laplace correction to all probabilities calculated from the training data.

A Laplace correction of +1 is applied to both class probabilities $P(y|s)$ and transition probabilities $P(x_n = v|s)$ computed from the training data, as described in equations 3.1 and 3.2, which we rewrite below. Laplace correction does not affect the test data, nor the holdout data. Note that changes in class probability $P(y|s)$ imply a change in expected cost of classification actions $C(s, f)$. These changes in probabilities (and expected costs of classification actions) induce a change in the MDP.

Let $P(y|s) = n_y/n$ be the class probability, where n is the number of training examples matching state s and out of them, n_y examples have class y. Then the Laplace corrected class probability is $P^L(y|s) = (n_y + 1)/(n + K)$, where K is the number of classes. Intuitively, each class has been given an extra training example.

Let $P(x_n = v|s) = n_v/n$ be the transition probability, where out of the n training examples matching state s, n_v have value v for attribute x_n. With the Laplace correction, the transition probability becomes $P^L(x_n = v|s) = (n_v+1)/(n+Arity(x_n)$. This is equivalent to creating an extra training example for each value of each attribute.

In the AO* algorithm, whenever a probability is computed from the training data, and the Laplace option is on, we apply the above Laplace corrections. This causes all probabilities to be non-zero. As a result, the AND/OR graph usually gets larger.

If the transition probability $P(x_n = v|s)$ is zero, in the original AO* code for expanding the AND node (s, x_n) in Table 4.3 we do not create an OR node for state $s' = s \cup \{x_n = v\}$. But with the Laplace correction this transition probability becomes greater than zero, and that is why we reach state s' now. In create-OR-node() for state s' of Table 4.2, we classify this node using the Laplace-corrected class probabilities.

4.3.3 Statistical Pruning (SP)

Even though the admissible heuristic can prune parts of the search space, its benefits may be limited (for example, if the attribute costs are small). We would like to prune additional parts of the search space. The **statistical motivation** is the following: given a small training data sample, there are many pairs of diagnostic policies that are statistically indistinguishable. Ideally, we would like to prune all policies in the AND/OR graph that are statistically indistinguishable from the optimal policies. Since this is not possible without first expanding the graph, we need a heuristic that approximately implements the following indifference principle:

Indifference Principle. *Given two diagnostic policies whose values are statistically indistinguishable based on the training data set, a learning algorithm can choose arbitrarily between them.*

This heuristic is called *statistical pruning* (abbreviated SP), and is applied in each fringe node s to the currently unexpanded optimistic policy $\pi^{opt}(s)$ and the current realistic policy $\pi^{real}(s)$. The action specified by $\pi^{opt}(s)$ (in fact, the AND node

$(s, \pi^{opt}(s)))$ will be pruned from the graph if a statistical test cannot reject the null hypothesis that $V^{opt}(s) = V^{real}(s)$. In other words, between an incomplete policy π^{opt} and a complete policy π^{real}, we prefer the last one. The statistical test checks if $V^{opt}(s)$ falls inside a confidence interval around $V^{real}(s)$.

This is a pruning heuristic applied as the AO*'s AND/OR graph is grown. The idea is to design a statistical test for pruning an unexpanded AND node (s, a), similar to the admissible heuristic's pruning when $V^{real}(s) \leq Q^{opt}(s, a)$. The purpose is to prune useless nodes, and thereby reduce the size of search space, which may also reduce overfitting.

When an AND node (s, a) is selected for expansion, we first check to see if this AND node should be pruned instead. If it can be pruned, we mark this AND node as pruned (expanded $= 2$). This action will then be ignored in further computations.

In our current implementation, this pruning of actions is irreversible. While we refer to pruning of AND nodes, note that once an AND node is pruned, the entire graph that could have potentially been constructed under this node is also pruned.

The SP option modifies steps (4) and (6) of the AO* algorithm from Table 4.1. We rewrite the pseudocode for AO*+SP in Table 4.6.

By marking an AND node (s, a) as pruned, the effect is that it is eliminated from the set $A(s)$ of actions valid in state s. Therefore in the updates of Q^{opt} and Q^{real}, in steps (5) and (6), we ignore pruned actions. This implies immediately that a pruned action cannot be part of π^{opt}, and therefore pruned actions will be ignored in step (3) as well.

We can also free the memory for the pruned AND node, since it no longer matters in the graph.

When we prune the current $\pi^{opt}(s) = a$ in the fringe node, we need to recompute its new optimistic policy $\pi^{opt}(s)$ from the set of actions $A(s) \backslash \{a\}$. This may increase $V^{opt}(s)$ and/or render the fringe node solved, therefore leading to optimistic updates in its marked ancestors. The code in Table 4.4 can be used without any change.

If the AND node (s, a) of the fringe OR node was pruned, there is no change in the

TABLE 4.6: Pseudocode for the AO* algorithm with statistical pruning SP.

```
function AO*+SP(int Mem-limit) returns a complete policy.

iteration $i = 0$;
Memory = 0;
create hash-table;
(1) OR node * root = create-OR-node($s_0$, (OR Node *) 0, hash-table, Memory);
(2) while ((Memory < Mem-limit) && (root not solved)){
        i++;
(3)     in current $\pi^{opt}$, select fringe OR node $s$ with
            AND node $(s, a)$ to expand $(\pi^{opt}(s) = a)$.
(4)     if ((SP-option = on) and (AND node (s,a) needs pruning))
            mark AND node (s,a) as pruned (expanded = 2).
            $A(s) = A(s) - \{a\}$.
            do not update $A'(s)$.
            free memory for this AND node.
        else
            expand AND node $(s, a)$ as usual (Table 4.3).
            update $A'(s)$ with the newly expanded action $a$,
                $A'(s) = A'(s) \cup \{a\}$.
(5)     do bottom-up updates of $Q^{opt}, V^{opt}, \pi^{opt}$.
(6)     if (AND node $(s, a)$ was expanded)
            do bottom-up updates of $Q^{real}, V^{real}, \pi^{real}$ for $s$ and its ancestors.
}
return last $\pi^{real}$.
```

TABLE 4.7: Checking if an unexpanded AND node $(s, \pi^{opt}(s))$ needs to be statistically pruned. The statistical test is a one-sided test checking if $V^{opt}(s)$ belongs to a normal confidence interval for $V^{real}(s)$, for a specified confidence level $1 - \alpha_{SP}$.

```
**function** AND-node-needs-pruning(OR node * fringe) **returns** prune=yes/no.

s = state of fringe OR node with unexpanded AND node $(s, \pi^{opt}(s))$.
$n$ = number of training examples matching state $s$.
**if** $(n < 2)$
    prune = yes.
**else**
    **for** every training example $te_j$ matching state $s$
        cost-real$_j$ = cost of $te_j$ when processed by $\pi_{real}(s)$.
    compute a $1 - \alpha_{SP}$ normal confidence interval
        for the costs cost-real$_j$, called $CI(V^{real}(s))$.
    **if** $(V^{opt}(s) \in CI(V^{real}(s)))$
        prune = yes.
    **else**
        prune = no.
return prune.
```

realistic graph, so no realistic updates are necessary (π^{real} stays the same). Indeed, (s, a) was previously unexpanded, and after pruning it will be ignored.

If the SP-option is off, or the AND node does not need to be pruned, the AO* algorithm continues as before.

Table 4.7 presents the details of how an AND node can become pruned. Note that the fringe node with state s may have other actions previously expanded, so its realistic policy $\pi^{real}(s)$ may be different from $best_f$ and $V^{real}(s)$ can be less than $C(s, best_f)$. Because state s is reached by π^{opt}, there must be at least one training example matching it. If indeed there is a single training example matching state s, we will prune the action. Otherwise, we compute a confidence interval for the costs of training examples traveling down $\pi^{real}(s)$. Let te be a training example matching state s, then we can recursively define $cost(te, \pi^{real}(s))$ as the cost of this training example (with class $te.y$) when processed according to the policy π^{real}. If π^{real} measures an attribute a, and v is the value of the training example for that attribute, $te[a] = v$, then te travels down to the state $s' = s \cup \{a = v\}$, and so on, until it reaches a leaf. $cost(te, \pi^{real}(s))$ sums the costs of attributes measured along the branch of $\pi^{real}(s)$ followed by the training example te, and the misclassification cost of predicting the class $best_f$ in the leaf.

$$cost(te, \pi^{real}(s)) \overset{\text{def}}{=} \begin{cases} MC(best_f, te.y) \\ \quad \text{if } \pi^{real}(s) = best_f \\ C(s,a) + cost(te, \pi^{real}(s')), \\ \quad \text{if } \pi^{real}(s) = a \neq best_f, te[a] = v, s' = s \cup \{a = v\}. \end{cases}$$

The confidence interval for $V^{real}(s)$ is the normal confidence interval (for a specified value of the confidence level $1 - \alpha_{SP}$) for the costs of n training examples te_j matching state s evaluated according to $\pi^{real}(s)$. The mean of cost-real$_j$ = $cost(te_j, \pi^{real}(s))$ is $\mu = V^{real}(s)$ and the standard deviation is

$$\sigma = \sqrt{\frac{\sum_j \text{cost-real}_j^2 - (\sum_j \text{cost-real}_j)^2/n}{n-1}}.$$

The confidence interval is then defined as

$$CI(V^{real}(s)) = \left[\mu - z_{SP} \cdot \frac{\sigma}{\sqrt{n}}, \mu + z_{SP} \cdot \frac{\sigma}{\sqrt{n}}\right],$$

where z_{SP} is the confidence coefficient corresponding to a $1 - \alpha_{SP}$ confidence interval.

Because $V^{opt}(s) \leq V^*(s) \leq V^{real}(s)$, this implies that the optimistic value is always less than the upper bound of the normal confidence interval for $V^{real}(s)$, $V^{opt}(s) \leq V^{real}(s) + z_{SP} \cdot \frac{\sigma}{\sqrt{n}}$. Therefore we only need to check where $V^{opt}(s)$ is situated versus the lower bound of the confidence interval.

It is interesting to note that if $V^{opt}(s)$ is inside the interval $CI(V^{real}(s))$, it will remain inside the interval even if further expansions increase $V^{opt}(s)$; this only holds for the current V^{real}.

At the current iteration i of the AO*+SP algorithm, $V_i^{opt}(s) = Q_i^{opt}(s,a) \leq V_i^{real}(s)$. If $Q_i^{opt}(s,a) \in CI(V_i^{real}(s))$, then action a will be pruned. From Theorem 4.2.1, $Q_i^{opt}(s,a) \leq Q^*(s,a)$, so $Q^*(s,a)$ is also in the confidence interval for the current $V_i^{real}(s)$ or exceeds the upper confidence limit. This only says that the optimal policy under AND node (s,a) is statistically indistinguishable or worse than the current realistic policy. It does not rule out the possibility that the realistic policy may improve in future iterations. If this happens, its realistic value will decrease and it is possible that its confidence interval will shrink as well. As a result, at some future iteration l, we may have $Q_i^{opt}(s,a) < V_l^{real}(s) - z_{SP} \cdot \frac{\sigma_l}{\sqrt{n}}$, and in retrospect, action a should not have been pruned at iteration i.

If the Laplace option is on, then besides correcting all the probabilities, we modify the statistical test to center the confidence interval around V^{real} computed with Laplace corrections. The standard deviation σ is still computed based on the n real training examples traveling down π_L^{real}. Note that V^{real} with Laplace corrections is not the mean of $cost(te, \pi_L^{real}(s))$ anymore.

In [2] we described a version of the SP heuristic that employed a pair-difference statistical test for the costs of training examples traveling down π^{real} and π^{opt}. On synthetic problems, the two statistical tests proved to be similar. We prefer the statistical test shown here, because we have a better theoretical understanding of it.

4.3.4 Pessimistic Post-Pruning (PPP) Based on Misclassification Costs

PPP is a post-pruning heuristic, similar to C4.5's pessimistic post-pruning described in Section 3.2. It is a pessimistic heuristic, because it exaggerates the expected classification cost as being the upper bound of a confidence interval. This PPP is applied to the final realistic policy computed by the learning algorithm.

In the following, we describe PPP for any complete policy π. In a complete policy

π, every internal node has $\pi(s)$ expanded, and every leaf node has $\pi(s) = best_f$ (classification action with minimum expected cost).

First we need to define an *upper bound* for the value of policy π in state s, $V^{\pi}(s)$.

$$UB(s, \pi(s)) \overset{\text{def}}{=} \begin{cases} UpperBoundMC(s, best_f) \\ \quad \text{if } \pi(s) = best_f \\ C(s,a) + \sum_{s'} P_{tr}(s'|s,a) \cdot UB(s') \\ \quad \text{if } \pi(s) = a \neq best_f. \end{cases}$$

If the policy classifies state s, $\pi(s) = best_f$, then $UpperBoundMC(s, best_f)$ is an upper bound for $C(s, best_f)$. It is computed by constructing a normal confidence interval (for a specified value of the confidence level $1-\alpha_{PPP}$) for the misclassification costs of all n training examples matching state s, when classified in $best_f$, and taking the upper bound of this confidence interval. Let $cost_j = MC(best_f, te_j.y)$ be the misclassification cost of classifying training example j (matching state s) with class y in $best_f$. We compute the $1 - \alpha_{PPP}$ normal confidence interval for all $cost_j$, and return

$$UpperBoundMC(s, best_f) \overset{\text{def}}{=} \mu + z_{PPP} \times \frac{\sigma}{\sqrt{n}},$$

where the mean of $cost_j$ is $\mu = C(s, best_f)$, the standard deviation is

$$\sigma = \sqrt{\frac{\sum_j cost_j^2 - (\sum_j cost_j)^2/n}{n-1}},$$

and z_{PPP} is the confidence coefficient corresponding to a $1-\alpha_{PPP}$ confidence interval. If there is a single training example matching s (i.e., $n = 1$), then we return a large value, $UpperBoundMC(s, best_f) = \infty$, so the policy in the parent node of s will be pruned.

Table 4.8 describes an implementation of the PPP, which destructively modifies the structure of the policy π. The function returns the new upper bound for the value of state s, $UB(s) \overset{\text{def}}{=} \min(UpperBoundMC(s, best_f), UB(s, \pi(s)))$. The function first computes $UpperBoundMC(s, best_f)$ for a state s. If s belongs to an internal node, it computes $UB(s, \pi(s))$ based on $UB(s')$ for its children s' reached through $\pi(s)$. Then

TABLE 4.8: Pessimistic Post-Pruning for a complete policy π (in our case, π^{real}). In case of pruning, the structure of the policy is modified.

```
function PPP(state $s$) returns an upper bound $UB(s)$.

$best_f$ = best classification action in state $s$.
compute UBMC = $UpperBoundMC(s, best_f)$.
if $(\pi(s) = best_f)$
    return UBMC.
else
    let a = $\pi(s)$.
    initialize UB-policy = $C(s, a)$.
    for every value $v$ of attribute $a$
        if $(P(a = v|s) > 0)$
            UB-policy += $P(a = v|s)\cdot$ PPP$(s \cup \{a = v\})$.
    if (UBMC $\leq$ UB-policy)
        $\pi(s) := best_f$ (action $a$ was pruned).
        return UBMC.
    else
        return UB-policy (called $UB(s, \pi(s))$ in text).
```

it decides if the action $\pi(s)$ needs to be pruned. This happens when the upper bound on the cost of classifying is less than the upper bound on the value of the policy $\pi(s)$, $UpperBoundMC(s, best_f) \leq UB(s, \pi(s))$. When pruning $\pi(s)$, the effect is that we reset $\pi(s)$ to be $best_f$.

If the Laplace option is on, besides correcting all the probabilities we also add one fake example in each class when computing $UpperBoundMC(s, best_f)$. Basically, we compute the normal confidence interval for the misclassification costs of $n + K$ training examples, where K is the number of classes, by adding one $MC(best_f, y)$ for each class y.

Bradford et al. [8] present a simpler pruning method for decision trees, based only on misclassification costs. A decision tree is first grown for the 0/1 loss (using a top-down induction method, based on information gain of attributes). All class probabilities are then Laplace-corrected, and the best classification action $best_f$ is the one with minimum expected misclassification cost, $C_L(s, best_f)$. Starting from the leaves of the decision tree, they prune a node's action when its children's expected misclassification costs are larger than the node's expected misclassification cost. Mathematically, define $UB(s) = \min(C_L(s, best_f), UB(s, \pi(s)))$, where

$$UB(s, \pi(s)) \overset{\text{def}}{=} \begin{cases} C_L(s, best_f) & \text{if } \pi(s) = best_f \\ \sum_{s'} P_{tr}(s'|s, a) \cdot UB(s') & \text{if } \pi(s) = a \neq best_f. \end{cases}$$

In their paper, there is no Laplace correction for transition probabilities $P_{tr}(s'|s, a)$. The action $\pi(s)$ in an internal node is pruned when $C_L(s, best_f) \leq UB(s, \pi(s))$.

4.3.5 Early Stopping (ES)

Early stopping employs an internal validation set to decide when to halt AO*. The training data is split in half. One half is used as the subtraining data, and the other half as the holdout data. AO* is trained on the subtraining data, and after every iteration, its policy is evaluated on the holdout data. The best realistic policy, according to the holdout data, is returned at the end of AO* search.

Note that we are not reconstituting the full training set. That is, we do not rerun

AO* on the entire training data until the iteration at which the best policy on the holdout data was found.

If the Laplace option is on, the transition and class probabilities computed from the subtraining data are Laplace-corrected, but those on the holdout data are not.

4.3.6 Dynamic Method

This method decides internally if it should apply Laplace correction to its probabilities or not. It first splits the training data in half (50% subtraining and 50% holdout). It runs AO* on the subtraining data, then it runs AO*+Laplace on the subtraining data (until convergence or the memory limit is reached), and then it evaluates their final policies π^{real} on the holdout data. The dynamic method chooses the best configuration (to apply Laplace or not) according to the holdout data ($\text{argmin}_{Laplace-option=on/off} V_{holdout}^{\pi^{real}(Laplace-option)}$). Then it runs AO* on the entire training data with the chosen configuration and returns the resulting policy.

The Dynamic method is not related to Early Stopping, though both make decisions using an internal holdout data. Dynamic only uses the holdout data to evaluate the last policy learned on the subtraining data, while Early Stopping evaluates the policy learned at each iteration on the holdout data to find the best stopping point. Also, Early Stopping does not rerun on the whole training set, while Dynamic does.

4.3.7 AND/OR Graph Initialized with a Known Policy

AO* does not need to start with an empty AND/OR graph. It can instead be initialized with an AND/OR graph corresponding to a known policy π_0. For example, we initialized the AO* search with one of the greedy policies MC+InfoGainCost described in Section 3.3 (with modified stopping conditions which use our optimistic heuristic). If π_0 is very good, this might allow AO* to run faster and obtain better cutoffs from its admissible heuristic.

If AO* has enough resources to terminate, it will compute the same optimal policy

TABLE 4.9: Pseudocode for initializing the AO* algorithm with a policy π_0.

function grow-initial-policy(state s, policy π_0, OR node * parent, hash-table, int & Memory) **returns** an OR node.

```
OR node * n = create-OR-node($s$, parent, hash-table, Memory).
$a_0 = \pi_0(s)$.
if (AND node $(s, a_0)$ is stored in $n$, i.e., $Q^{opt}(s, a_0) < C(s, best_f)$)
    expand AND node $(s, a_0)$ by calling grow-initial-policy()
        recursively in the resulting states $s'$.
    mark AND node $(s, a_0)$ as expanded (expanded = 1).
    update $A'(s)$ with the newly expanded action $a_0$.
    do optimistic updates for node $n$.
    do realistic updates for node $n$.
return $n$.
```

π^* with or without the initial policy π_0.

Table 4.9 has the pseudocode for initializing the AND/OR graph. In Table 4.1, replace step (1) with

OR node * root = grow-initial-policy(s_0, π_0, (OR Node *) 0, hash-table, Memory).

The initial policy is grown top-down, creating OR nodes as in Table 4.2, until classifying is cheaper, $C(s, best_f) \leq Q^{opt}(s, a)$ for all attributes a not yet measured in s. If that is not the case, we select attribute $a_0 = \pi_0(s)$ and expand it if it is cheaper than classifying, according to our heuristic function. If $C(s, best_f) \leq Q^{opt}(s, a_0)$, there is no point in expanding a_0 (in fact, the AND node (s, a_0) is not even stored in its parent OR node).

In this function, expanding AND node (s, a_0) involves calling the function grow-initial-policy() recursively in each of the resulting states s' if $P_{tr}(s'|s, a_0) > 0$; note that it is not necessary to check if these states s' are in the hash table, because they were not generated before (the policy is a tree, so a state does not repeat).

As we return from the function's recursive calls, we update the optimistic values and policy, and also the realistic values and policy. Recall that a_0 is the only action expanded in node n, so we only need to update its Q^{opt} value and compute its Q^{real} value (this computation is done for the first time, since a_0 just became expanded):

$Q^{opt}(s, a_0) := C(s, a_0) + \sum_{s'} P_{tr}(s'|s, a_0) \cdot V^{opt}(s')$.

$Q^{real}(s, a_0) := C(s, a_0) + \sum_{s'} P_{tr}(s'|s, a_0) \cdot V^{real}(s')$.

Next we update $V^{opt}(s)$ and $\pi^{opt}(s)$. Since only AND node (s, a_0) is expanded in n, all other actions are unexpanded and have not modified their Q^{opt} values. If a_0 was different from the old $\pi^{opt}(s)$, we only need to compare the updated $Q^{opt}(s, a_0)$ with the old $V^{opt}(s)$; if they were equal, we need to check if after expansion, a_0 remains equal to $\pi^{opt}(s)$. After updating $V^{opt}(s)$, $\pi^{opt}(s)$,

```
if ($\pi^{opt}(s) = best_f$)
    mark OR node $n$ as solved.
else
    if ($\pi^{opt}(s) = a_0$)
        mark connectors for $n$ and its children OR nodes through $a_0$ as part of $\pi^{opt}$.
        mark $n$ as solved if all its children OR nodes through $a_0$ are solved.
    else
        do nothing ($\pi^{opt}(s)$ is unexpanded, there is nothing to mark).

if ($Q^{real}(s, a_0) < C(s, best_f)$)
    $\pi^{real}(s) := a_0$.
    $V^{real}(s) := Q^{real}(s, a_0)$.
```

4.3.8 Combining Regularizers

These regularizers can be combined in many ways. The memory limit Mem-limit is always present, although it may be set too large to have any effect. We implemented the following combinations of regularizers: AO*, AO*+L, AO*+SP, AO*+SP+L, AO*+PPP, AO*+PPP+L, AO*+SP+PPP, AO*+SP+PPP+L, AO*+ES, AO*+ES+L, Dynamic, AO*+(MC+InfoGainCost). The next chapter will discuss the best performing regularizers.

4.4 Review of AO* Literature

We need to mention from the very beginning that none of the uses of AO* in literature, to our knowledge, was applied to learning from data; instead, a model of the problem (either an MDP, or a POMDP) was assumed; therefore these algorithms fall into the planning, not learning, category. The AO* algorithm has been studied extensively both in the Artificial Intelligence and the Operations Research communities. This AO* literature review is organized from the most general to the more specific.

4.4.1 AO* Relation with A*

AO* is similar to A*, except that it is for AND/OR graphs, while A* is for graphs with only one type of nodes (OR nodes). AO*'s expanding leaf AND nodes in π^{opt} is the analog of A*'s expanding nodes with minimum $f(n) = g(n) + h(n)$. An optimal policy in AO* corresponds to an optimal path in A*. Given admissible heuristics, both algorithms terminate with optimal solutions. When AO* terminates, it finds an optimal policy with value $V^*(s_0)$; when A* terminates, it finds an optimal path with value f^*. In AO*, if the heuristic h^{opt} is admissible, then the optimistic values of states s stored in the OR nodes, $V^{opt}(s)$, will increase during the search (until they become the optimal values, $V^*(s)$); in A*, if the heuristic h is admissible, then the $f(n)$ values of the sequence of nodes expanded by A* will increase.

4.4.2 AO* Notations, Implementations, and Relation with Branch-and-Bound

Our definitions of AND and OR nodes are similar to those of Martelli and Montanari [45], Chakrabarti et al. [11], Pattipati and Alexandridis [53], Qi [61] and Hansen [24]. An OR node specifies the choice of an action. It is called an OR node because its solution involves the selection of only one of its children. An AND node specifies the outcomes of an action. It is called an AND node because in order to solve it, all its children must be solved. These definitions are the reverse of Nilsson's

[50] in which the type of a node is determined by the relation to its parent.

There are several implementations of AO*: two by Martelli and Montanari [45, 46], one by Nilsson [50], and one by Mahanti and Bagchi [42]. The first three implementations are practically identical. Martelli and Montanari are the first to recognize that dynamic programming techniques that discover common subproblems can be applied to search AND/OR graphs and to compute the optimal solution. Martelli and Montanari [45] show that the AO* algorithm with an admissible heuristic converges to an optimal policy π^* (if complete policies exist, reaching the terminal state from the start state, which is always the case in our finite CSL problem).

Our implementation follows the framework of Nilsson's, with specific additions for the cost-sensitive learning problem. To implement AO* following his description, one only needs an admissible heuristic that optimistically estimates (that is, it underestimates) the value of reaching the terminal state s_f from any state s. Our analysis is more complex than Nilsson's, because he does not explicitly differentiate between AND nodes and OR nodes when describing the AO* algorithm for graphs, though he makes the distinction between the two nodes when discussing AND/OR trees. He calls AND/OR graphs hypergraphs, and their hyperarcs/hyperlinks connectors, so instead of arcs/links connecting pairs of nodes in ordinary graphs, connectors connect a parent node with a set of successor nodes. The complete solution (no longer a path, but a hyperpath), is represented by an AND/OR subgraph, called a solution graph (with our notation, this is a policy). These connectors require a new algorithm, AO*, for the AND/OR graphs, instead of the A* algorithm for ordinary graphs.

Nilsson does not refer to the nodes of an AND/OR graph as being AND nodes or OR nodes, because in general a node can be seen as both an OR node and an AND node, plus his notation is the reverse of ours, calling the node type with respect to its parent, not the node itself. Pearl [55] notes as well that an AND link and an OR link can point to the same node. Nevertheless, in our CSL problem, with our definition for AND nodes and OR nodes, we have a clear separation between the two types of nodes (a node cannot belong to both types). Chakrabarti et al. [11] call them pure

AND and OR nodes (unlike the nodes of mixed type allowed by Nilsson [50]).

AND/OR graph search algorithms have been studied by many authors. At first, the algorithms worked on an *implicit* graph (the entire graph that can be generated), which was assumed to be acyclic [46, 50, 42, 11]. An *explicit* graph is the part of the graph generated during the search process. Graphs with cycles were usually solved by unfolding the cycles. For a recent review of AO* algorithms for searching both explicit and implicit AND/OR graphs with cycles see [31].

Branch-and-bound algorithms use lower and upper bounds to prune non-optimal branches, without generating and evaluating the entire AND/OR graph. Kumar and Kanal [36] explain the relationship between branch-and-bound algorithms from Operations Research and heuristic search algorithms from Artificial Intelligence (including alpha-beta [33], AO*, B* [3], and SSS* [65]). Other relevant articles, which outline AO* as a branch-and-bound procedure are [49, 37, 38]. They show that AO* is a special case of a general branch-and-bound formulation.

4.4.3 Theoretical Results on AO*

Memory-bounded AO*

AO* may require memory exponential in the size of the optimal policy. Chakrabarti et al. [10] propose running AO* in restricted memory by pruning unmarked nodes (that are not part of some π^{opt} in the graph) when the available memory is reached. This method still computes the optimal value function, trading-off space for time (if the pruned nodes are needed again, they must be generated again).

Inadmissible Heuristics

If the heuristic h is inadmissible, but within ϵ of the optimal value function V^*, then it is straightforward to compute a bound on the value of the suboptimal policy learned with h. Indeed, if $h(s) - V^*(s) \leq \epsilon, \forall s$, then the maximal error of the policy π computed by AO* with heuristic h is ϵ, $V^\pi - V^* \leq \epsilon$ (see [11] and [61], page 31).

Chakrabarti et al. [11] showed that the optimal policy can be computed with an inadmissible heuristic if if its weight is shrunk. If the heuristic function can be decomposed into $f = g + h$, where g is the cost incurred so far, and h is the heuristic estimating the remaining cost, then AO* with the weighted heuristic $(1-w)\cdot g + w\cdot h$, where $0 \leq w \leq 1$, compute an optimal policy π^* even for an overestimating (inadmissible) heuristic h. The weight w is such that $w < \frac{1}{1+\epsilon}$, where ϵ is the maximum distance between the inadmissible heuristic h and V^*. Note that for node n, $f(n) = g(n) + h(n)$ is the estimated cost of the cheapest policy through the node n; since the policy is a decision tree, node n occurs only once in it, therefore $g(n)$ is the cost of the path from the root to n.

Influence of Heuristic on Nodes Expanded

Chakrabarti et al. [11] show that a more accurate admissible heuristic in AO* has a smaller worst-case set of nodes expanded. That is, if h_2^{opt} heuristic is closer to the optimal value function than h_1^{opt}, $h_1^{opt} \leq h_2^{opt} \leq V^*$, then the largest set of nodes expanded by AO* with the h_2^{opt} heuristic is a subset of the largest set of nodes expanded by AO* with the h_1^{opt} heuristic.

4.4.4 POMDPs

Partially Observable MDPs (POMDPs) are MDPs in which the state of the world is not known with certainty. Instead, observations reveal information about the state of the world. The states of the POMDP are called *belief or information states.* Instead of minimizing expected costs, usually POMDPs maximize expected values.

Heuristic search methods (either branch-and-bound or AO*) have been applied to approximately solve infinite-horizon POMDPs from a single initial belief state. Satia and Lave [64] proposed a branch-and-bound algorithm for finding optimal and ϵ-optimal POMDP policies; Larsen and Dyer [39] improve upon this work. Washington [73] used the value function of the underlying MDP to define lower and upper bounds in AO*. Hansen [24] developed a heuristic search algorithm that combines AO* with policy iteration for approximately solving infinite-horizon POMDPs (from a given start state) by searching in the policy space of finite-state controllers. For a recent review of approximation methods for solving POMDPs, which also includes a systematic presentation of lower and upper bounds for POMDPs, see Hauskrecht [26].

Hansen's LAO* algorithm [25] is a generalization of AO* that solves MDPs with cycles by using a dynamic programming method (either value iteration or policy iteration) for the bottom-up update of the optimistic value function (and policy). This is necessary because in the general case of an MDP with cycles, we can not perform a single sweep of value iteration through the state space from the fringe node to the root node to update the value function. Bonet and Geffner [7, 6] follow up with heuristic search algorithms for solving MDPs with cycles, which converge faster than value iteration, RTDP or Hansen's LAO*. RTDP [1] is a Real Time Dynamic Programming Algorithm that bypasses full dynamic programming updates in MDPs by only updating the values of states reached from an initial state during repeated trials of executing a greedy policy; a heuristic is used to initialize the value function. If the heuristic is admissible, then RTDP converges in the limit to the optimal policy. RTDP extends Korf's Learning Real Time A* (LRTA*) algorithm [34] to asynchronous dynamic programming with stochastic actions. LRTA* can be seen as a real-time version of A*, and RTDP is the real-time version of AO* and LAO*. In terms of how they represent solutions, A* outputs a simple path (a sequence of actions), AO* outputs a directed acyclic graph, and LAO* outputs a cyclic graph (a finite-state controller).

In [2] we describe AO* for the cost-sensitive learning problem with both attribute costs and misclassification costs, formulating it as an acyclic MDP. A related paper is Bonet and Geffner's [5] which learns decision trees from data by formulating this problem as a POMDP and using the RTDP version for POMDPs algorithm to solve it. Even though POMDPs can accommodate both attribute costs and misclassification costs, Bonet and Geffner only evaluated the learned decision trees in terms of accuracy and compared them to ID3 and C4.5.

4.4.5 Decision-theoretic Analysis

Qi [61] showed how to represent finite acyclic MDPs with a unique start state by *decision graphs* (which are acyclic AND/OR graphs with an evaluation function). His definitions for AND and OR nodes are similar to ours. An OR node represents the choice of an action and is called a *choice* node in decision graphs. An AND node represents a set of possible observations and is called a *chance* node.

Qi develops and extends decision graph search algorithms and applies them to decision problems in various forms (decision trees, MDPs and influence diagrams) by first transforming the problems into the decision graph representation. He describes three algorithms for decision graph search, all of which compute the optimal value function when admissible heuristics are used: depth-first search (DFS) for decision trees, AO*, and iterative deepening search. His depth-first search algorithm uses an admissible heuristic and a branch-and-bound pruning technique (an upper bound β-value, similar to the α-β method [33]). He also describes an anytime version of it outputting decision trees whose values improve monotonically. The AO* implementation follows Nilsson's [50]. Qi implemented two iterative deepening search algorithms: DFS with increasing depth bounds as cutoffs (which is probably a better algorithm than Greiner et al.'s [22]), and DFS with a lower and an upper bound on costs.

Qi's work is mostly theoretical, and he assumes the probability model is given. His main contribution is to transform decision problems given in the form of influence diagrams into decision graphs (this is an intermediate representation), and use decision graph search algorithms to compute an optimal policy, therefore proposing a new method for influence diagram evaluation [62].

4.4.6 Test Sequencing Problem

Pattipati and Alexandridis [53] solved the test sequencing problem of detecting faults in electronic systems by applying AO*. Though this problem only uses test costs, and no misclassification costs, we discuss it in detail below, hoping that it will encourage the machine learning community to explore this work, though it does not involve learning from data. A recent review appears in [78].

The test sequencing problem is a simpler version of the cost-sensitive classification problem. The original test sequencing problem (also known as the test planning problem) is defined by a set of system states (faulty states and a fault-free state), their (a priori) probability distribution, a set of binary tests, the costs of these tests, and a binary diagnostic matrix. The problem has three assumptions: (*i*) only one of the system states occurs; (*ii*) the available tests can identify the system states unambiguously (this is the reason for which misclassification costs are not relevant to this problem, since they assume a correct diagnosis is always possible, and this incurs no cost); and (*iii*) tests do not affect the system states. The objective is to deterministically identify the presence of any system state while minimizing the expected test costs. Pattipati and Alexandridis [53] prove that the test sequencing problem is NP-complete (by reducing the NP-complete exact cover by 3-sets problem to it).

A solution to the test sequencing problem takes the form of a decision tree. The root corresponds to the entire set of system states, and each leaf corresponds to a single system state. Each test in a node separates the system states into two disjoint subsets (such tests are called *symmetrical*). [78] extends the test sequencing problem from binary to multiple-valued tests, and from symmetrical to asymmetrical tests (after performing a test, the resulting sets can have system states in common). They still assume perfect classification is possible.

The expected test costs of such a decision tree has a simple computation, as the sum of test costs in each branch of the decision tree, weighted by the probability of the system state that appears in the leaf. The optimal solution to the test sequencing problem is the decision tree with minimum expected test costs.

Pattipati and Alexandridis observe that the test sequencing problem is an MDP whose solution is an optimal AND/OR binary decision tree. An MDP state is a set of system states. They also note that dynamic programming methods can compute the optimal solution, but since the implementation is $O(3^N)$, where N is the number of tests, this is impractical for more than 12 tests.

Approximation (Greedy) Methods for the Test Sequencing Problem

Greedy heuristics have been employed on the test sequencing problem. One-step lookahead algorithms were applied to select the test that maximizes the information gain per unit cost of test or that maximizes a "separation" heuristic. In their terminology, the information gain for a binary test t in an MDP state s is

$$IG(s,t) = -[P(t=0|s) \cdot \log_2 P(t=0|s) + P(t=1|s) \cdot \log_2 P(t=1|s)],$$

which, for symmetrical tests, can be shown to be equivalent to the information gain formula commonly used in machine learning (see Section 3.2). The "separation" heuristic saves time by replacing the above information gain computation with $P(t=0|s) \cdot P(t=1|s)$, which produces the same decision tree when the test costs are equal. On realistic problems the cost of the diagnostic procedure generated by the one-step lookahead algorithms is rarely twice the cost of the optimal solution.

Systematic Search Methods for the Test Sequencing Problem

To compute the optimal solution to the test sequencing problem, Pattipati and Alexandridis employ AO* with two admissible heuristics, one derived from Huffman coding and the other being based on the entropy of system states. These two heuristics require equal test costs in order to be admissible. For simplicity, unit test costs are used, since their objective function is linear in test costs. For unit test costs, the expected total test cost is exactly the expected number of tests required to reach pure leaves. The admissible heuristic estimates this expected number of tests as the expected length of the Huffman binary coding for the system states. The Huffman code is free to make up its own tests rather than use only the tests given in the diagnostic matrix. Therefore, it may need fewer tests than will be needed when the tests in the diagnostic matrix are used.

AO* with an admissible heuristic based on a modified Huffman code is also used for a network troubleshooting problem [40]. The heuristic combines information about component failure rates and test costs. The AO* algorithm is then applied to generate a network troubleshooting expert system which minimizes the expected troubleshooting cost.

4.4.7 Relation of CSL with Reinforcement Learning

The *reinforcement learning (RL)* framework studies methods for learning how to solve MDPs. RL algorithms can be either model-based (they learn the MDP model) or model-free (they learn a policy or value function by direct interaction with the environment). In reinforcement learning the training data is collected interactively, online, as tuples of the form $\langle s_t, a_t, c_{t+1}, s_{t+1} \rangle$, while in CSL the training data has been collected in advance.

It is common in RL to collect the training data using a policy (known as the exploration policy) which is different from the policy being learned. Popular exploration policies include ϵ-greedy and Boltzmann exploration. These policies have the property that they execute every action in every state infinitely often. This is sufficient to prove convergence of many RL algorithms to the optimal policy.

In our CSL formulation, we can imagine that the training data was collected and labeled by an "observe-everything" exploration policy (see Section 2.5.4). This policy is used to collect and label the training data. Because the exploration policy has measured all available attributes, we can evaluate any policy on the collected training data. In general, it is difficult to evaluate a policy by observing data gathered by another policy. The optimal policy (on the training data) can still be learned if the policy that gathered the data has non-zero probability of visiting every state and trying every action. This problem is known as the problem of "off-policy" learning in reinforcement learning (see [66]).

We tried solving synthetic CSL problems using model-free RL algorithms, including Q-learning and Sarsa. We discovered that we needed a very small learning rate to obtain convergence. The reason is that the one-step cost for a classification action is highly stochastic. Hence, the results of many trials must be averaged to get stable results. This observation pointed us to the necessity of learning a model, and we realized once we had the model that we could perform heuristic search using AO* since our CSL problem has a unique start state and is acyclic. Our "model" consists of the training data, and we use it to compute probabilities as needed.

AO* can be viewed as expanding the state space (through exploration) and then updating the value function via local propagation. According to this view, it is an efficient implementation of prioritized sweeping [48, 57] for MDPs with a single start state and no cycles.

Pednault et al. [56] and the work of Zadrozny [76] also discuss the relationship between sequential cost-sensitive decision making processes and RL, though they only consider misclassification costs. Their application domain is targeted marketing.

4.5 Summary

This chapter showed how to solve the cost-sensitive problem using the AO* algorithm. We introduced an admissible heuristic and a new policy (called the realistic policy) whose quality improves with future iterations, therefore turning AO* into an anytime algorithm. We presented several levels of abstraction of the algorithm, including a detailed implementation. To reduce overfitting, we proposed several regularizers for AO*. The next chapter describes experimental results of applying greedy and systematic algorithms to the CSL problem.

CHAPTER 5

EXPERIMENTAL STUDIES

This chapter describes an extensive experimental study of the various greedy and systematic CSL algorithms presented in Chapters 3 and 4. The goal is to identify one or more practical, fast algorithms that learn good cost-sensitive learning policies on real problems.

Here are the main questions this chapter addresses:

- Which algorithm is the best among all the CSL algorithms proposed in Chapters 3 and 4? If there is no overall winner, which is the most robust algorithm, and where does it fail?

These questions are not trivial. It is hard to say which one of several algorithms is the best; the difficulty is enhanced by the fact that we have multiple criteria:

- quality of the learned policies,
- computational efficiency, and
- ease of implementation.

We want efficient algorithms (in terms of time and memory), and we also want algorithms that produce good policies. An easy-to-implement algorithm is also desirable. But the most important thing is the quality of the learned policy.

This chapter is organized as follows: first we introduce the UCI domains, define the test costs and misclassification costs, and explain how we divide the data into training and test sets. To answer the question of which is the best overall algorithm, we compare the performance of the algorithms on the test sets; several evaluation methods are presented, with different levels of abstraction. We illustrate overfitting in anytime graphs, which confirm the need for regularization in AO*. The Results section compares the algorithms on several domains. The Discussion section answers the main questions of the chapter. The chapter ends with insights on how the performance of the algorithms is affected by differences in costs, regularizers, and the type of search.

5.1 Experimental Setup

5.1.1 UCI Domains

We performed experiments on five cost-sensitive learning problems based on real data sets found at the University of California at Irvine (UCI) repository [4]. The five problems are listed here along with a short name in parentheses that we will use to refer to them: Pima Indians Diabetes (pima), Liver disorders (bupa), Cleveland Heart Disease (heart), the original Wisconsin Breast Cancer (breast-cancer), and SPECT (spect). The domains were chosen for two reasons. First, they are all real medical diagnosis domains. Second, attribute costs have been provided for three of them (pima, bupa, and heart) by Peter Turney [71].

Before describing the five domains in detail, we describe some pre-processing steps that were applied to all of the domains. First, all training examples that contained missing attribute values were removed from the data sets. Second, if a data set contained more than two output classes, some of the classes were merged so that only two classes (healthy and sick) remained. Third, any existing division of the data into training and test sets was ignored, and the data were simply merged into a single set. As we will describe later, we performed multiple random divisions of the available data into training and testing sets instead. Each real-valued attribute x_j was discretized into 3 levels (as defined by two thresholds, θ_1 and θ_2) such that the discretized variable takes on a value of 0 if $x_j \leq \theta_1$, a value of 1 if $\theta_1 < x_j \leq \theta_2$ and a value of 2 otherwise. The values of the thresholds are chosen to maximize the mutual information (InfoGain) between the discretized variable and the class. The mutual information was computed using the entire data set.

TABLE 5.1: BUPA Liver Disorders (bupa).

attrib	name	partitions	infogain	cost(attribute)
x_1	mcv	3	0.0428408	7.27
x_2	alkphos	3	0.0203466	7.27
x_3	sgpt	3	0.0369976	7.27
x_4	sgot	3	0.0557107	7.27
x_5	gammagt	3	0.0599615	9.86

We now describe the five domains in details.

5.1.1.1 BUPA Liver Disorders Domain (bupa)

In this domain, the goal is to diagnose whether a patient has a liver disorder that might arise from excessive alcohol consumption. The diagnosis is made based on five blood tests. This data set also contains a "selector" attribute which was included to record how the data had been divided into training and test sets in an earlier study. We deleted this attribute.

There are 345 examples with no missing values, 5 numerical attributes and 2 classes (0 = healthy liver, 1 =sick). 169 of the examples (48.98%) are healthy. The attribute costs, donated by Peter Turney, are 7.27 and 9.86 (see Table 5.1). The attributes were discretized into 3 partitions.

5.1.1.2 Pima Indians Diabetes Domain (pima)

In this domain, the goal is to diagnose whether a patient has diabetes. The patients are females of at least 21 years of age and of Pima Indian heritage. There are 768 examples with no missing values, 8 numerical attributes, and two classes (0 = healthy,

TABLE 5.2: Pima Indians Diabetes (pima).

attrib	name	partitions	infogain	cost(attribute)
x_1	times pregnant	3	0.0460019	1
x_2	glucose tol	3	0.173206	17.61
x_3	diastolic bp	3	0.0204929	1
x_4	triceps	3	0.0387898	1
x_5	insulin	3	0.0613776	22.78
x_6	mass index	3	0.0891406	1
x_7	pedigree	3	0.0285451	1
x_8	age	3	0.0818478	1

1 = tested positive for diabetes). 500 of the examples (65.10%) are healthy. The attribute costs, donated by Peter Turney, are 1, 17.61 and 22.78 (see Table 5.2).

The attributes were discretized in 3 partitions (note that in [2] we used a binary discretization).

5.1.1.3 Heart Disease Domain (heart)

In this domain, the goal is to diagnose whether a patient has a heart disease. From the original data set called "processed.cleveland.data" we first eliminated the 6 examples that had missing values. This left 297 examples. There are 4 different types of heart disease in the database. To create a 2-class problem, we grouped values 1, 2, 3, 4 together as class 1 (sick = presence of heart disease), and retained value 0 (absence of heart disease). 160 of the examples (53.87%) are healthy. There are 13 numerical attributes (3 are binary; we discretized the other 10 into 3 partitions). The attribute costs, donated by Peter Turney, are 1, 5.20, 7.27, 15.50, 87.30, 100.90 and 102.90 (see Table 5.3).

TABLE 5.3: Heart Disease (heart).

attrib	name	partitions	infogain	cost(attribute)
x_1	age	3	0.0752612	1
x_2	sex	2	0.057874	1
x_3	cp	3	0.196691	1
x_4	trestbps	3	0.0289602	1
x_5	chol	3	0.0398084	7.27
x_6	fbs	2	7.2228e-06	5.2
x_7	restecg	3	0.0234737	15.5
x_8	thalach	3	0.149107	102.9
x_9	exang	2	0.132295	87.3
x_{10}	oldpeak	3	0.155641	87.3
x_{11}	slope	3	0.108775	87.3
x_{12}	ca	3	0.18437	100.9
x_{13}	thal	3	0.210234	102.9

TABLE 5.4: Breast Cancer (breast-cancer).

attrib	name	partitions	infogain	cost(attribute)
x_1	clump thickness	3	0.434685	1
x_2	uniformity of cell size	3	0.664429	1
x_3	uniformity of cell shape	3	0.641162	1
x_4	marginal adhesion	3	0.446273	1
x_5	single epithelial cell size	3	0.524958	1
x_6	bare nuclei	3	0.587182	1
x_7	bland chromatin	3	0.537129	1
x_8	normal nucleoli	3	0.479613	1
x_9	mitoses	3	0.206436	1

5.1.1.4 Breast Cancer Domain (breast-cancer)

This is the original Wisconsin Breast Cancer Domain. The goal in this domain is to diagnose a palpable breast mass as benign or malignant.

We first eliminated the 16 examples that had missing values, which left 683 examples. We also removed the first attribute (id number) because it is not useful for diagnosis. The remaining 9 attributes (see Table 5.4) were discretized into 3 partitions. There are two classes (healthy = benign tumor, sick = malignant tumor). 444 of the examples are healthy (65%).

Since there are no attribute costs, we set them all to be 1.0. Note that all the n tests are actually the n observations of a single test (a fine needle aspiration of a palpable breast mass), but we treat them as n different tests, in order to evaluate the effectiveness of our cost-sensitive learning algorithms.

5.1.1.5 SPECT Heart Domain(spect)

The goal in this domain is to diagnose cardiac problems based on Single Proton Emission Computed Tomography (SPECT) images. The data set of 267 images was first processed to extract 44 continuous attributes, then further processed to obtain 22 binary attributes. Since the data is already in binary form, we did not need to discretize it.

There are 267 examples with no missing values, 22 binary attributes (see Table 5.5) and two classes (0 = normal, 1 = abnormal). 55 of the examples (20.6%) are labeled as healthy (class 0). Since there are no attribute costs, we set them all to be 1.0. As with breast-cancer, we treat these attributes as 22 separate tests for purposes of evaluating cost-sensitive learning. In the real application, all 22 attributes would be computed simultaneously from a SPECT image.

5.1.2 Setting the Misclassification Costs (MC)

While attribute costs can usually be found, misclassification costs are harder to assign. In medical diagnosis, if the cost of incorrect diagnoses is set to incorporate costs for being sued, the MC will be so high compared to attribute costs that most or all of the tests will be performed in order to get as much knowledge about the disease as possible.

For the UCI domains, we wanted a reasonable range of MC to attribute costs, preferably such that we do not encounter the trivial cases of policies testing no attributes or testing all attributes. We believe the question of how to set MC for realistic synthetic domains is an interesting open problem, and one that deserves deeper analysis. Even when MC are known for a domain, it helps to vary them in order to see how flexible the CSL methods are to changes in the relative costs of attributes and misclassifications.

Let $E[tc]$ be the expected $\underline{t}$est $\underline{c}$osts for a policy on a data set. This is the average of the costs of attributes tested by the policy on each example in the data set. Our idea for setting MC is the following:

TABLE 5.5: Spect (spect).

attrib	name	partitions	infogain	cost(attribute)
x_1	F1	2	0.028932	1
x_2	F2	2	0.037068	1
x_3	F3	2	0.038135	1
x_4	F4	2	0.0326777	1
x_5	F5	2	0.0231747	1
x_6	F6	2	0.0334025	1
x_7	F7	2	0.040793	1
x_8	F8	2	0.0672918	1
x_9	F9	2	0.0207391	1
x_{10}	F10	2	0.0334099	1
x_{11}	F11	2	0.0358317	1
x_{12}	F12	2	0.0362212	1
x_{13}	F13	2	0.111126	1
x_{14}	F14	2	0.024555	1
x_{15}	F15	2	0.0327299	1
x_{16}	F16	2	0.060274	1
x_{17}	F17	2	0.051615	1
x_{18}	F18	2	0.0471958	1
x_{19}	F19	2	0.0161291	1
x_{20}	F20	2	0.0450407	1
x_{21}	F21	2	0.0708648	1
x_{22}	F22	2	0.0610843	1

- grow a simple policy for some initial large MC and measure its expected test costs, denoted $\max E[tc]$. Presumably this policy will test quite a few attributes, so it will have a large $E[tc]$.

- prune this policy for smaller MC values decreasing to zero. The pruned policies test fewer and fewer attributes, therefore $E[tc]$ decreases (all the way to zero when the policy classifies without measuring any attributes).

- several values of MC will be chosen such that the $E[tc]$ of their policies are well spaced. Our method dynamically samples MC in interesting regions where $E[tc]$ shows a rapid change (which presumably means different policies).

Our algorithm applies to 2-class problems with zero-diagonal misclassification costs matrices. The misclassification costs that will be produced depend on the choice of the data set. In our experiments, we used each domain's entire data set to set the misclassification costs. This, of course, can be modified to use any data set (for example, the training data set). We used the entire data set because we will have different replicas of the data, with different training and test sets, and we wanted the same misclassification costs for the entire domain (that implies the same misclassification costs for all replicas). We emphasize again that each domain has its own misclassification costs.

On each domain, we apply the following steps to set its misclassification costs. We assume that in the initial state s_0, before any attributes were measured, both classification actions f_0 (classify in class 0) and f_1 (classify in class 1) have equal expected cost $C(s_0, f_0) = C(s_0, f_1)$. By definition (see 2.7), the expected cost of classification action f_k in state s is $C(s, f_k) = \sum_y P(y|s) \times MC(f_k, y)$. For a zero-diagonal MC matrix there are no misclassification costs for correct classification, that is, $MC(f_k, y) = 0$ when $f_k = y$, so the above equation becomes

$$P(y = 1|s_0) \cdot MC(f_0, y = 1) = P(y = 0|s_0) \cdot MC(f_1, y = 0),$$

or

$$\frac{MC(f_0, y=1)}{MC(f_1, y=0)} = \frac{P(y=0|s_0)}{P(y=1|s_0)} = r,$$

where we denote this ratio by r. The class probabilities $P(y|s_0)$ are computed on the entire data set, so we know r. With the notations introduced in Chapter 2, $MC(f_0, y = 1)$ are the misclassification costs of false negatives, $MC(fn)$, and $MC(f_1, y = 0)$ are the misclassification costs of false positives, $MC(fp)$. We can rewrite $\frac{MC(fn)}{MC(fp)} = r$, or $MC(fn) = r \cdot MC(fp)$. Denote $MC(fp) = m$. We will choose values for m. Then we will set the misclassification matrix to

$$MC = \begin{pmatrix} 0 & r \cdot m \\ m & 0 \end{pmatrix}.$$

This adjustment in misclassification costs reflects realistic behavior where the rare cases are usually more expensive to misclassify. Indeed, when $r = P(y = 0)/P(y = 1) < 1$, as is the case in bupa and spect, there are fewer negative than positive examples (i.e., there are fewer healthy patients). In this case, misclassifying the negatives is more expensive, $MC(fn) < MC(fp)$. Similarly, when $r > 1$, there are fewer positives than negatives (there are fewer sick patients). In such cases, misclassifying the positives is more expensive, $MC(fn) > MC(fp)$.

We initially set

$$MC = \begin{pmatrix} 0 & r \cdot lm \\ lm & 0 \end{pmatrix},$$

where m was assigned a large value lm. Using this MC matrix, we grow a greedy policy π^G top-down based on our optimistic heuristic h^{opt} (Section 4.2.2), so

$$\pi^G(s) = \operatorname*{argmin}_{k,i} \left(C(s, f_k), Q^{opt}(s, \text{unexpanded } x_i) \right).$$

Attributes are tested until classifying becomes cheaper. To set lm, we try several large values of m to determine a value beyond which $E[tc]$ does not increase further. This value is selected as lm. Let π^G be the policy corresponding to lm. If no matter how high lm is set, $\max E[tc] = 0$, then our method for setting MC cannot be applied, and it also means that the optimal policy (on the entire data set of that domain) is to classify directly.

After π^G is grown, we decrease m from lm to zero, in unit steps. Each node s in π^G has a threshold value $m_t(s)$ such that $m < m_t(s)$ causes the policy under s to be pruned from π^G. In that case, we set $\pi^G(s) = best_f = \text{argmin}_k C(s, f_k)$; note that, because of the structure of the MC matrix, the best classification action in a state s, $best_f$, is the same for all values of m. Hence, as m decreases, π^G will be gradually pruned, bottom-up, as m becomes lower than the thresholds of some of the nodes. Since π^G has a finite number of internal nodes, there will be a finite number of pruned policies. As m decreases and more nodes of π^G are pruned, $E[tc]$ and the value of the pruned policies decrease monotonically with m. Nevertheless, the expected misclassification costs do not necessarily decrease as m decreases.

Once we have generated all of the possible pruned versions of π^G, we choose (at most) five values of m, as follows. The largest m is set to be the smallest value with $E[tc] = \max E[tc]$. The smallest m is chosen to be the largest value that gave $E[tc] = 0$. We then choose 3 values of m whose policies have their $E[tc]$ closest in Euclidian distance to 1/4, 1/2, and 3/4 of $\max E[tc]$. In case these desired $E[tc]$ values are close to a plateau of $E[tc]$, we choose a value of m at the beginning of the plateau (if the desired $E[tc]$ is less than the plateau's $E[tc]$), a value of m in the middle of the plateau (if the desired $E[tc]$ is equal to the plateau's $E[tc]$), and a value of m at the end of the plateau (if the desired $E[tc]$ is greater than the plateau's $E[tc]$).

For example, in Figure 5.1 on spect, the selected m values are: $m = 27$ (largest m whose pruned policy tests no attributes), $m = 33$ (m at the end of the second plateau, closest to $1/4 \cdot \max E[tc] = 2.02715$), $m = 34$ (m at the beginning of the third plateau, closest to $1/2 \cdot \max E[tc] = 4.05431$), $m = 57$ (m with closest $E[tc]$ to $3/4 \cdot \max E[tc] = 6.08146$) and $m = 249$ (m at the beginning of the plateau with $\max E[tc]$). Note that the graph exhibits a staircase increase of E[tc] with m.

The 5 chosen values of m define 5 different MC matrices, though sometimes two of the matrices may be very close in value. We denote them by MC1, MC2, MC3, MC4

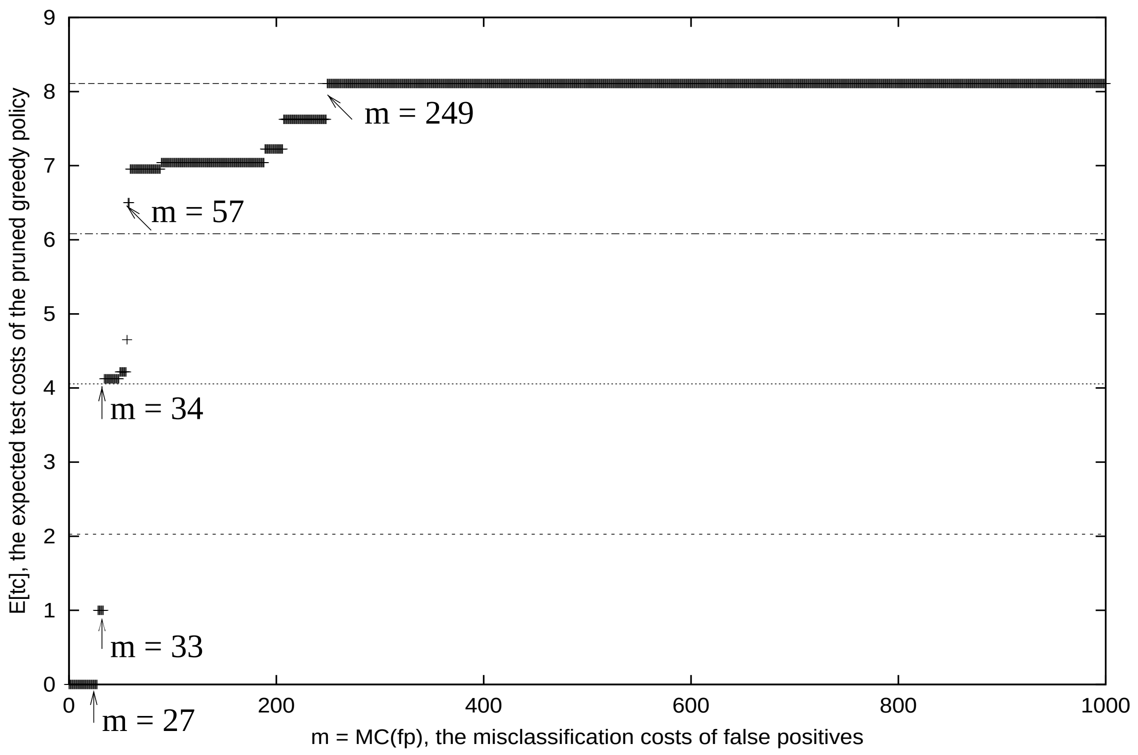


FIGURE 5.1: Spect domain, selecting five values for $m = MC(fp)$, used to set the misclassification cost matrices. A greedy policy π^G is grown first using a large value of m. We decrease m from that large value to zero. Each m defines a misclassification costs matrix, which is used to post-prune the initial policy π^G. As m decreases, more nodes of π^G are pruned, until the entire policy is pruned. The number of pruned policies is finite; for each of them we measure the expected test costs, $E[tc]$. The selected values of m produce policies whose $E[tc]$ are closest to five equally-spaced target values for $E[tc]$ in the interval from 0 to $\max E[tc]$.

and MC5. The matrices with larger indexes (like MC5) have larger misclassification costs. These large costs reduce the ratio of test costs to misclassification costs, and therefore testing attributes will become cheaper in comparison to classifying. This makes the problems more difficult (at least for the systematic search algorithms). We sometimes refer to the misclassification costs as small, medium, and large.

The Appendix B lists the values for the five misclassification costs used for each domain, in Tables B.1, B.2, B.3, B.4 and B.5.

5.1.3 Training Data, Test Data, Memory Limit

Experimental comparisons of learning algorithms must take into account two sources of random variation: variation due to the choice of the training data and variation due to the choice of the test data [13]. This is especially true for cost-sensitive learning, because different training examples can have different costs, so the addition or subtraction of a single example from the training or test set can produce a big change in the measured performance of a learning algorithm.

To measure variation due to the choice of the training data, we repeated our experiments 20 times by generating 20 random training/test splits. To measure variation due to the choice of the test data, we applied the BDeltaCost statistical test, which applies bootstrap sampling to simulate (and thereby measure) test set variability. BDeltaCost is described in Section 5.1.5.3.

For each UCI domain, the transformed data (2 classes, discretized attributes with no missing values) is used to generate 20 random splits into training sets (two thirds of data) and test sets (one third of data), with sampling stratified by class. Such a split (training data, test data) is called a *replica.* On each domain, the same replicas are used with the five misclassification cost matrices.

These replicas have overlapping test sets (and training sets), so they are not independent. Section 5.1.5.5 discusses the effect of having non-independent replicas on our analysis.

Memory Limit

Given enough resources, AO* converges to an optimal policy on the training data. Similarly, AO* with various regularizers converges to an optimal policy on the MDP defined by the training data and the regularizer. For large domains (with large number of attributes), the AND/OR graph grows very large, especially in the following cases:

- the attributes are not very informative,
- the attributes are cheap relative to the misclassification costs, so the AO*'s admissible heuristic will not perform its cutoff,
- the optimal policy is very deep,
- there are many policies tied to the optimal one and the systematic search algorithm needs to expand them to prove to itself that there is no better alternative.

To make systematic search feasible, we need to prevent the AND/OR graph from growing too large. We do this by imposing a limit of 100 MB on the amount of memory that the AND/OR graph can use. Each time an AND or OR node is created, we update a count of the amount of space required to store the new node(s). We actually count the amount of space required by a "theoretical" optimized implementation, since our actual implementation is not optimized and stores extra debugging and measurement information in each node. As a result, the 100 MB "theoretical" limit was in practice a 500 MB limit. The greedy algorithms always converged within the memory limit. The systematic algorithms, based on AO*, converge within this memory limit most of the time (spect is the exception, for large misclassification costs). If this memory limit is reached before the systematic algorithm converged, the best realistic policy found so far is returned.

It is interesting to note that even on a domain with many attributes, a small training set may severely limit the number of reachable states (that is, the possible combinations of attribute values), and thereby limit the AND/OR graph to a manageable size.

5.1.4 Notations for the Cost-Sensitive Algorithms

The following algorithms will be compared in this chapter. In this chapter's text and in the graphs, we chose an abbreviation of the full name of the algorithms introduced before. A Laplace correction is indicated by an "L" suffix added to the name.

- Nor, Nor-L denote Norton with pessimistic post-pruning and Norton with pessimistic post-pruning and Laplace correction, from Section 3.2. The pessimistic post-pruning is similar to C4.5; the Laplace correction is only done in the pruning phase, not in computing InfoGain.

- MC-N, MC-N-L denote MC+Norton and its Laplace version from Section 3.3. Both algorithms employ a pessimistic post-pruning based on test costs and misclassification costs. MC-N-L has Laplace correction for transition and class probabilities. As the policy is grown, Laplace is applied for $P(y|s)$ used in computing $C(s, f)$, but not for InfoGain; in the pruning phase, Laplace is also used for $P(x_n = v|s)$, and it was used in $C(s, f)$.

- VOI, VOI-L denote the one-step Value of Information and its Laplace version from Section 3.4. In VOI-L, the Laplace correction is applied to transition and class probabilities as the 1step VOI policy is grown; there is no need for pruning.

- AO*, AO*-L denote AO*, described in Section 4.2, and AO* with the Laplace regularizer from Section 4.3.2.

- ES, ES-L denote AO* with our Early Stopping regularizer, and its Laplace version from Section 4.3.5.

- SP, SP-L denote AO* with our Statistical Pruning regularizer, and its Laplace version, from Section 4.3.3. In SP, we used a 95% confidence interval.

- PPP, PPP-L denote AO* with our Pessimistic Post-Pruning regularizer, based on misclassification costs, and its Laplace version, from Section 4.3.4. In PPP, we also used a 95% confidence interval.

Other Considerations

We chose Norton as the representative of the InfoGainCost methods because the others, Tan and Nunez, have performance very similar to Norton. Norton also has the simplest, most intuitive mathematical formula for including test costs ($\frac{InfoGain}{\text{attribute cost}}$). The InfoGain method does not use test costs, and overall, its performance was quite poor, even with Laplace corrections.

On spect and breast-cancer, because all attributes have the same cost (= 1), all InfoGainCost methods are identical, and we could choose any of them (we choose Norton). The same holds for MC+InfoGainCost, where we choose MC+Norton.

We do not describe the performance of the Dynamic algorithm, from Section 4.3.6, because it is not any better than AO*-L.

We do not describe the performance of AO* with both the statistical pruning SP and pessimistic post-pruning PPP regularizers, nor its Laplace version, because they were not any better than SP or PPP or their Laplace versions.

Nor do we describe the performance of AO* initialized with one of the MC+InfoGainCost methods, described in Section 4.3.7. According to BDeltaCost, these methods were tied with, or worse than, AO*. Indeed, if AO* has enough resources to terminate, the initialization phase has no effect on the quality of the learned policy. If AO* reached the memory limit before convergence, which happens for spect, BDeltaCost shows that AO* still terminates with a better policy; in that case no initialization of AO* with any of the MC+InfoGainCost policies has provided enough memory savings to obtain convergence within this memory limit, nor has it produced a better policy when reaching the memory limit.

We performed experiments on synthetic problems for the setting of the confidence levels $1 - \alpha_{SP}$ and $1 - \alpha_{PPP}$ used in Statistical Pruning and Pessimistic Post-Pruning. We tried values corresponding to ten different confidence intervals: $50\%, 60\%, 70\%, 80\%, 90\%, 95\%, 99\%, 99.99\%, 99.9999\%$ and 99.999999%, but no single value was consistently good. Therefore we used a conventional value corresponding to a 95% confidence interval (so the confidence coefficients are $z_{SP} = z_{PPP} = 1.96$).

5.1.5 Evaluation Methods

The goal of this dissertation is to identify good CSL algorithms, taking into account the variance in the data replicas, the influence of relative sizes of test costs and misclassification costs, and the nature of the domain. We manipulate the relative costs of testing and misclassification by varying the misclassification costs while holding the test costs fixed.

To evaluate each algorithm, we train it on the training set and then compute its average total cost (measurement cost and misclassification cost) on the test set. We will denote the average test set cost by V_{test}. We seek algorithms with low V_{test}, because these algorithms are producing the minimum cost diagnostic policies.

While the quality of the policy as measured by V_{test} is the most important quantity to measure, we also measured two other statistics. One is the error rate (i.e., the fraction of test examples misclassified). The other is the average number of attributes tested (denoted *eatp*). These two quantities allow us to observe the tradeoff between misclassification and measurements. They also help us determine whether an algorithm is overfitting (e.g. if eatp is very high or error rate is very low).

We describe below the computation of V_{test}, the other measurements, and the BDeltaCost statistical test. BDeltaCost is a statistical test for deciding whether one policy is better than another. It compares the V_{test} for two policies and decides if there is a statistically significant difference between their values.

5.1.5.1 Measurements of Interest; The Most Important One is the Quality of the Policy on the Test Set V_{test}

We describe how to evaluate any complete policy π on some sample data (which could be the training data, validation or holdout data, or the test data).

There are several measurements we are interested in. We define these measurements for a single example in the sample data. Then we average over all examples to obtain the measurement value on the sample. For a given example and a given policy, the measurements are: test costs, misclassification cost, number of attributes tested and error (if the example was misclassified).

Let ex be any of the n examples in *sample*. Denote by $m(ex, \pi, s_0)$ the measurement value of the example ex following policy π starting at the root with state s_0. The measurement value on the *sample* is $m^{\pi}_{sample} = \frac{\sum_{ex} m(ex,\pi,s_0)}{n}$.

The example travels down the decision tree of the policy to a leaf node, with classification action $best_f$. If $\pi(s)$ measures attribute a, and v is the value of the example for this attribute, $ex[a] = v$, then the example will travel to the next state $s' = s \cup \{a = v\}$.

$$\text{test-costs}(ex, \pi, s) \overset{\text{def}}{=} \begin{cases} 0, & \text{if } \pi(s) = best_f \\ C(s, a) + \text{test-costs}(ex, \pi, s'), & \text{if } \pi(s) = a \neq best_f. \end{cases}$$

$$\text{attrib-tested}(ex, \pi, s) \overset{\text{def}}{=} \begin{cases} 0, & \text{if } \pi(s) = best_f \\ 1 + \text{attrib-tested}(ex, \pi, s'), & \text{if } \pi(s) = a \neq best_f. \end{cases}$$

The misclassification cost and the error are defined when the example ex reaches its corresponding leaf node in the policy. Let $best_f$ be the classification action in the leaf node reached by example ex, and let $ex.y$ be the class of this example. The *misclassification cost* is

$$mc(ex, \pi, s_0) = MC(best_f, ex.y),$$

and the *error* is

$$error(ex, \pi, s_0) \overset{\text{def}}{=} \begin{cases} 0, & \text{if } best_f = ex.y \\ 1, & \text{otherwise.} \end{cases}$$

The total cost of an example ex down π is

$$cost(ex, \pi) = \text{test-costs}(ex, \pi, s_0) + mc(ex, \pi, s_0).$$

The value of the policy π on the *sample* data is

$$V^{\pi}_{sample} = \frac{\sum_{ex} \text{test-costs}(ex, \pi, s_0) + mc(ex, \pi, s_0)}{n}.$$

The expected number of attributes tested by π on the *sample* data is

$$\text{eatp}^{\pi}_{sample} = \frac{\sum_{ex} \text{attrib-tested}(ex, \pi, s_0)}{n}$$

(this can be interpreted as the average depth of policy π on the *sample*), and the error rate of π on the *sample* data is

$$\text{error-rate}^{\pi}_{sample} = \frac{\sum_{ex} error(ex, \pi, s_0)}{n}.$$

We are interested in the algorithms' performance on the test data. To measure this we replace *sample* by the test data in the above definitions. Thus we obtain V_{test}, eatp_{test} and error-rate_{test}; *test* is an abbreviation for test data, and it is only used as such in these measurement notations, in all other contexts "test" means measuring the value of an attribute.

The policies produced by both greedy and systematic search algorithms are evaluated using these measurements. Note that in our CSL framework, a policy value is always computed using both test costs and misclassification costs, no matter how the policy was learned (recall that some of the greedy methods only pay attention to test costs).

5.1.5.2 Computational Efficiency

The policy quality is the main factor in deciding if an algorithm is good or not, but, because we are interested in practical algorithms, we also compute the amount of memory and CPU time used by each algorithm to learn its policy.

5.1.5.3 BDeltaCost – a Statistical Test Comparing the Expected Costs of Two Diagnostic Policies

BDeltaCost analyzes the costs of individual test examples to try to decide if the V_{test} of one policy π_1 is statistically different from the V_{test} of another policy π_2. Given two policies, we want to test the null hypothesis H_0 that the two policies are tied (have the same value) on the test data against the hypothesis that one policy is better than the other.

Let $ex_1, \ldots, ex_n$ be our n test data points. To apply BDeltaCost, we must first compute an array Δ whose i-th element is the difference in the cost of processing ex_i (adding up both test costs and misclassification costs) using two different policies π_1 and π_2:

$$\Delta[i] = cost(ex_i, \pi_1) - cost(ex_i, \pi_2).$$

Note that the average value of the elements in Δ is equal to the difference in the V_{test} of the two policies: $V_{test}^{\pi_1} - V_{test}^{\pi_2}$. The goal of BDeltaCost is to decide if this average difference is less than zero, equal to zero, or greater than zero, which will tell us whether π_1 is better than, tied with, or worse than π_2.

BDeltaCost is based on the idea of using bootstrap replicates [16] to estimate a confidence interval for a parameter from a random data sample. A *bootstrap replicate* is a new data sample, constructed from the original data sample by drawing data points at random, with replacement. Replacement means that after a data point is drawn, it is put back into the sample (replaced) so that it can be drawn again. The bootstrap replicate has the same size as the original data sample. The basic idea of a bootstrap confidence interval is to construct a large number N_b of bootstrap replicate samples, compute the parameter separately for each sample, and then look at the amount of variation in the computed parameter values.

For BDeltaCost, the sample is the array Δ. BDeltaCost constructs $B = 1000$ bootstrap replicates of Δ and computes the average value $\mu[b]$ for each replica ($b = 1, \ldots, 1000$). It then sorts the array μ and forms a confidence interval whose lower bound is $\mu[26]$ and whose upper bound is $\mu[975]$. This interval contains 950 of the 1000 computed averages, so it provides a 95% confidence interval for the expected cost difference $E[cost(ex, \pi_1) - cost(ex, \pi_2)]$. If this confidence interval contains 0, then we cannot reject the null hypothesis that the expected cost difference between the two policies is zero (i.e., the policies are tied). If the entire confidence interval lies below zero (i.e., $\mu[975] < 0$), then we reject the null hypothesis in favor of the conclusion that π_1 has lower expected cost. If the entire confidence interval lies above zero (i.e., $\mu[26] > 0$), then we reject the null hypothesis in favor of the conclusion that π_1 has higher expected cost. Table 5.6 gives the pseudo-code for the BDeltaCost procedure.

Note that because we employ a 95% confidence level, BDeltaCost has a 0.05 probability of making a Type I error (i.e., of rejecting the null hypothesis when in fact it is true). This means that when we compare two algorithms on 20 data replicas with BDeltaCost, one out of the 20 replica results showing a statistical difference may be wrong.

BDeltaCost measures variability with respect to the random choice of the test data. A small test set will result in wide bounds for the confidence interval. A large test set will give a much tighter confidence interval. BDeltaCost does not measure variability with respect to the random choice of the training data. It is for this reason that we create 20 train/test replicas and apply BDeltaCost separately to policies trained on each replica. The variation across the 20 replicas (even though they are not truly independent data sets) gives some idea of the variability due to the random choice of training set. If in the 20 replicas, one algorithm wins half the time and loses half the time, this shows that there is a high degree of variability and no clear winner can be identified. But if one algorithm wins half the time and the other algorithm never wins, this increases our confidence that the first algorithm is superior to the second.

Unfortunately, pairwise statistical tests only support pairwise conclusions—there is no transitivity property. Hence, if policy π_1 is tied with policy π_2, and π_2 is tied with π_3, according to BDeltaCost, we can not conclude anything about the relationship between π_1 and π_3. For this reason, we apply BDeltaCost separately to each pair of policies.

Finally, notice that BDeltaCost only tells us if one algorithm's policy is better than another's. It does not give a measure of the magnitude of the difference. To

TABLE 5.6: The BDeltaCost statistical test compares the expected costs of two diagnostic policies π_1 and π_2 on the test data. The array Δ has n cost differences, $cost(ex_i, \pi_1) - cost(ex_i, \pi_2), i = 1, \ldots, n$, one for each test example ex_i. A confidence interval CI based on B bootstrap replicates of Δ is constructed for a specified confidence level $1 - \alpha$. The position of zero relative to the confidence interval determines the win/tie/loss for the two policies.

```
**function** BDeltaCost(sample $\Delta$ of size $n$, number of bootstrap replicates $B$) **returns**
the result of the statistical test.

for $b$ from 1 to $B$
    $\mu[b] = 0$;
    for $j$ from 1 to $n$
        draw $r$, a random number between $1, n$.
        $\mu[b] = \mu[b] + \Delta[r]$;
    $\mu[b] = \mu[b]/n$; mean of bootstrap replicate $b$.
sort the $B$ means $\mu$ in increasing order.
$lb = \lfloor \frac{\alpha}{2} \cdot B \rfloor + 1$; $\mu[lb]$ is the lower bound of the CI.
$ub = B - lb$; $\mu[ub]$ is the upper bound of the CI.
if $(0 < \mu[lb])$
    $res = -1$; $\pi_1$ is worse than $\pi_2$ (*loss* for $\pi_1$).
else
    if $(\mu[ub] < 0)$
        $res = 1$; $\pi_1$ is better than $\pi_2$ (*win* for $\pi_1$).
    else
        $res = 0$; $\pi_1$ is tied to $\pi_2$ (*tie*).
return $res$.
```

understand this, we also need to look at the size of the difference in V_{test} values.

5.1.5.4 Synthesis of Evaluation Methods

In our experiments, we run each algorithm on each combination of domain, misclassification cost matrix MC, and train/test replica. We denote this by (domain, MC, replica). For each pair of algorithms, we apply BDeltaCost, which reports a win, tie, or loss for one of the algorithms over the other.

Once we obtain these results, we summarize them at several different levels of aggregation in order to gain an understanding of the general patterns of behavior of the different methods. There are three levels of aggregation: the replica level, the MC level, and the domain level:

1. *Replica level:* for each (domain, MC), we present a graph showing the V_{test} of each pair of algorithms along with the results of the BDeltaCost test. This helps visualize the variability across the different train/test replicas.

2. *MC level:* for each (domain, MC), we present two summaries of performance across the replicas:

   - Graphs of *average V_{test} across all replicas (with normal confidence intervals)* for each algorithm. Each graph displays these measurements for each algorithm and allows us to compare them.
   - Tables of *cumulative BDeltaCost results for pairs of algorithms.* These tables sum the wins, ties, and losses for each pair of algorithms across the replicas.

3. *Domain level*: for each (domain), we summarize the BDeltaCost results using a chess metric that weighs the wins, ties, and losses of each algorithm against all the others. This metric will be introduced in the Section 5.4.1.

All of these different ways of summarizing the results hide some details. For example, the average V_{test} can be affected by one particularly easy replica (with small V_{test}), or by one particularly difficult replica (with large V_{test}).

Note that it does not make sense to summarize the performance of algorithms summing over domains (or over misclassification cost matrices), because the difficulty of each CSL problem is different in each domain and the MC matrices are not comparable across domains.

5.1.5.5 Effect of Non-independent Replicas on Interpretation of the Results

Our data replicas have overlapping test sets (and training sets), so they are not independent. We discuss the impact this has on our interpretation, when combining results from different replicas.

Average and normal confidence intervals of V_{test} over replicas

Each algorithm is run on the 20 replicas. For each replica, it learns a policy which we evaluate on the test data and thus obtain 20 V_{test} estimates. For each algorithm, we average these V_{test} estimates over the 20 replicas, and compute a normal confidence interval for them. We know the replicas are not independent, so the resulting confidence intervals underestimate the true variability of the learned policies and the V_{test} values. Hence, the confidence intervals will be too narrow. But the normal confidence interval still gives some idea of the variability of V_{test} for an algorithm and if the intervals of two methods overlap, this is evidence that there is no difference in their V_{test} values.

Cumulative BDeltaCost results over replicas

For each domain and MC matrix, we obtain 20 BDeltaCost results. We will sum these wins, ties, losses over all 20 replicas. However, an overall winning score for one algorithm does not prove that it is superior, because the replicas are not independent. Nonetheless, if the accumulated BDeltaCost results show that the two policies are

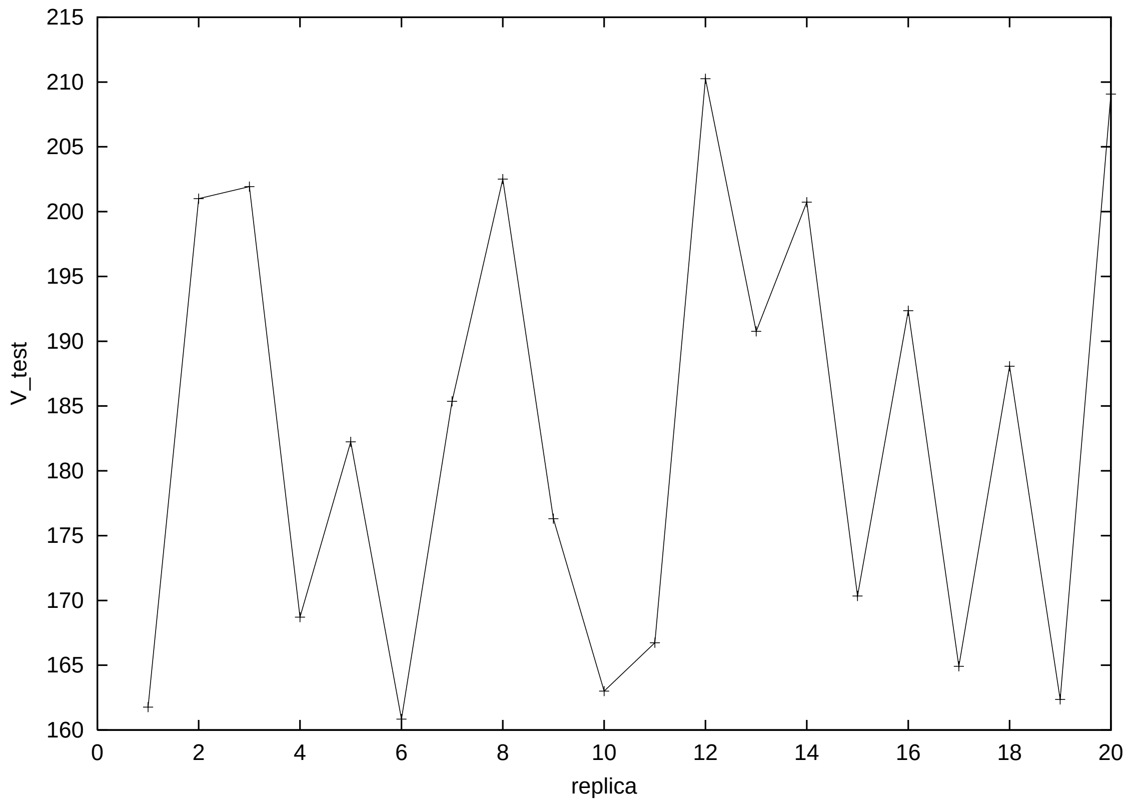


FIGURE 5.2: Heart, MC3, V_{test} of AO* shows high variability across replicas.

mostly tied, then they are likely to be statistically indistinguishable. If we get mostly wins for one policy, that is weak evidence in favor of the superiority of that policy. If we get consistent weak evidence across multiple domains, then we can strengthen the claim that one policy is better than another, since each domain is truly independent.

How variable are the replicas?

Figure 5.2 plots V_{test} for AO* on the heart problem, MC3. It shows that there is huge variation (from 160 to 210) in the V_{test}. Some replicas are harder and some are easier. This high degree of variation reassures us that our replicas are doing a good job of

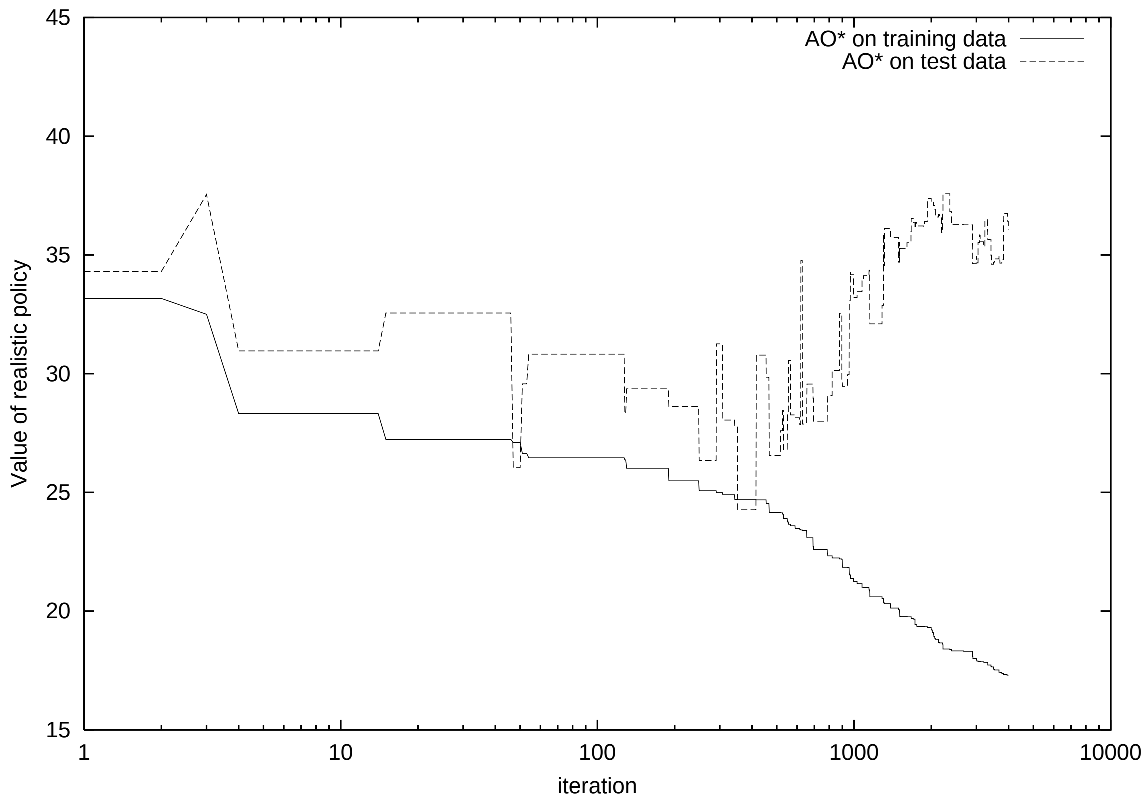


FIGURE 5.3: Anytime graph of AO* on pima, MC3, one of the replicas. The best realistic policy, according to the test data, was discovered after 350 iterations, after which AO* overfits.

simulating the variation due to the choice of training and test sets and reduces our concern about the validity of combining results across replicas.

5.2 Overfitting

When an algorithm learns patterns specific to the training data and has low generalization power on new, unseen data, we say that it suffers from *overfitting*. The algorithm learned its training examples too well, including both the signal and the noise, and it is not able to generalize well on different samples drawn from the true distribution.

CSL algorithms are prone to overfitting as are any algorithms learning from data. Since our goal is to determine which algorithms produce good policies (as evaluated on an independent test set), it is important to study how overfitting affects the quality of the learned policy. We will address the following questions: How much of a problem is overfitting? How important is overfitting reduction? How good are the regularizers at reducing overfitting?

To answer these questions, we will discuss so-called anytime graphs and the effect of the regularizers on the learning process. An anytime graph plots V_{train} and V_{test} as a function of the number of AND nodes expanded in the AND/OR graph (the number of expanded AND nodes is also the number of iterations). Figure 5.3 shows an anytime graph for AO* on the pima problem (MC3). Notice that AO*'s best realistic policy (the policy with minimum V_{test}) was discovered after only 350 iterations. After this point, V_{test} increases, which is the classic sign of overfitting. The quality of the policies learned afterwards continues to improve monotonically on the training data, but their performance on the test data gets worse. Upon convergence, AO* has learned the optimal policy on the training data, but this policy performs badly on the test data. This anytime graph demonstrates that overfitting is a problem for AO* and confirms the need for regularizers.

One of the regularizers described in Chapter 4 is Early Stopping. It uses a holdout data set to choose a stopping point. In this particular case (pima, MC3), the performance of realistic policies on the holdout data tracks their performance on the test data, so ES is able to find a good stopping point.

Another regularizer introduced in Chapter 4 is Laplace correction. Figure 5.4 shows that AO* with the Laplace regularizer gives worse performance on the training data but a better performance on the test data than AO*. Despite this improvement Figure 5.4 shows that AO* with Laplace still overfits: a better policy that was learned early on is discarded later for a worse one.

All of the regularizers we proposed, including Early Stopping, Laplace and Statistical Pruning, may expand different parts of the search space, and this makes it hard to analyze them. In general, ES and SP expand an AND/OR graph of roughly the same size as AO*, while the Laplace regularizer (AO*-L, ES-L, SP-L) increases the size of search space. Pessimistic Post-Pruning does not affect the search space since it is a post-pruning method.

Figure 5.5 shows that the VOI policy is suboptimal on the training data, but outperforms AO*'s last policy on the test data. Nevertheless, AO* has discovered several policies that are better on the test set than the VOI policy. Overfitting hurts AO* to the point where greedy simpler methods perform better on the test set. Regularizing AO* will give it an edge over the greedy methods on several domains.

Notice that while we proposed regularizers primarily for systematic search, regularization may also be helpful for greedy search. Therefore we will also measure the effectiveness of the Laplace correction regularizer on the greedy algorithms. Nevertheless, overfitting is more of a problem for systematic algorithms, because they do more thorough search in the policy space. In particular, if AO* has enough resources to terminate, it will compute the optimal policy on the training data, but this policy may be very bad when evaluated on the test set.

In conclusion, overfitting confirms the need for regularizers. Overfitting can also be indirectly hypothesized by noticing that policies have large values and large average depths.

5.3 Results

In order to answer the main questions of this chapter: "Which algorithm is the best one? If there is no overall winner, which is the most robust algorithm and where does it fail?", we first describe the results on each domain. The main questions break down in several others: Which algorithm is the best on each domain? Which algorithm is the worst? How do the greedy algorithms compare with the systematic search ones? How does Laplace correction influence V_{test}? How does AO* perform, and

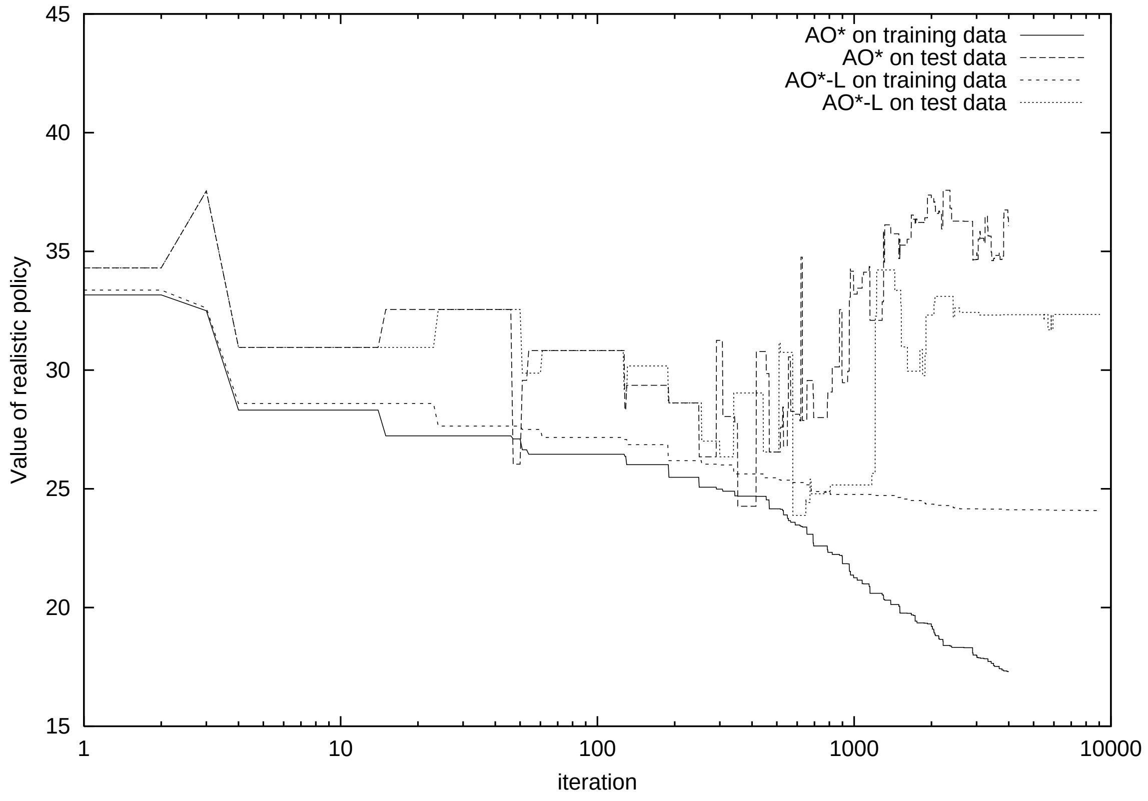


FIGURE 5.4: Anytime graphs of AO* and AO*-L on pima, MC3. The Laplace regularizer helps AO*, both in the anytime graph and in the value of the last policy learned.

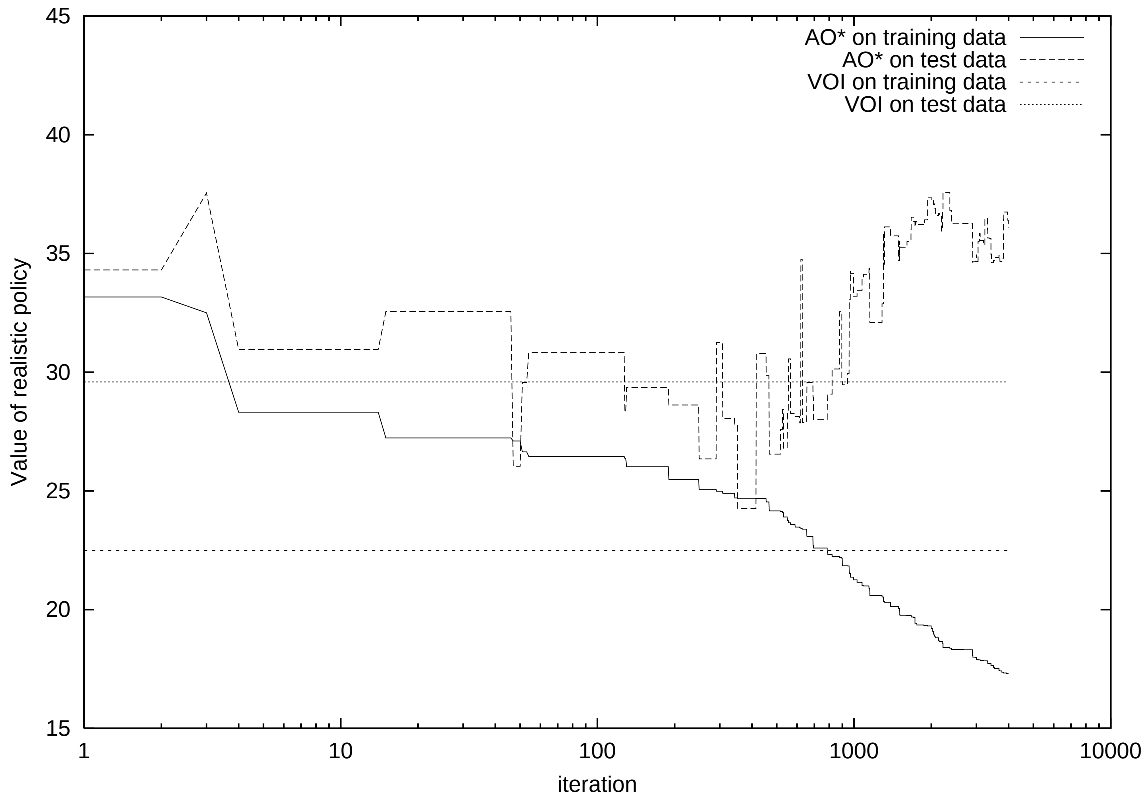


FIGURE 5.5: In its anytime graph, AO* learns a better policy than VOI, measured on the test data, but then forgets it. The optimal policy learned by AO* on the training data (its last policy) has worst performance on the test data than VOI. Upon convergence (at iteration 3990), AO* has found a better policy on the training data than VOI, which confirms that the VOI policy is suboptimal on the training data.

TABLE 5.7: Abbreviations.

CSL	cost-sensitive learning
MC	misclassification cost level (MC1, MC2, MC3, MC4, MC5)
CI	confidence interval
Nor, Nor-L	Norton, and Norton with Laplace
MC-N, MC-N-L	MC+Norton, and MC+Norton with Laplace
VOI, VOI-L	Value of Information, and with Laplace
AO*, AO*-L	AO*, and AO* with Laplace
ES, ES-L	AO* with Early Stopping, and with Laplace
SP, SP-L	AO* with Statistical Pruning, and with Laplace
PPP, PPP-L	AO* with Pessimistic Post-Pruning, and with Laplace

which regularizers help it the most? How do different misclassification cost matrices influence the best and the worst algorithm in a domain?

Table 5.7 lists the abbreviations used in this chapter; for more information on each algorithm, see Section 5.1.4.

5.3.1 Laplace Correction Improves All Algorithms

Before describing the experimental results in each of the domains, we first discuss the most important overall pattern. Laplace regularization generally improves or leaves unaffected every one of the seven search methods (VOI, MC-N, Nor, AO*, ES, SP, and PPP).

Table 5.8 shows the total (wins, ties, losses) for each greedy search method with Laplace correction compared against itself without Laplace correction. Notice that for MC1, the Laplace-corrected method always either wins or ties the method without Laplace. With a few exceptions, this is true for the other MC levels, except for MC5 where Laplace correction appears to hurt the performance of Nor on the spect data set.

Table 5.9 shows the same information for the systematic search algorithms. Here again, the Laplace regularizer almost always beats or ties against the unregularized algorithm. There is no case where it clearly gives poor results.

For systematic algorithms, AO* benefits by far the most from the Laplace correction. This is to be expected, since ES, SP and PPP have already brought their own regularization to AO*.

For greedy algorithms, Laplace helped Nor the most. Note that we extend the term "Laplace-corrected algorithms" to all algorithms, even though Nor-L is not a Laplace-corrected version of Nor, since Laplace is only used to correct the error rate in the pessimistic post-pruning. This study shows that it is useful the apply a Laplace correction during the post-pruning phase of Norton, and it answers the question raised in Section 3.2.

Because Laplace regularization is clearly good, we decided to simplify the remainder of this chapter by using BDeltaCost to compare only the seven search algorithms with Laplace correction. Hence, we will consider only the 21 pairs of Laplace-corrected algorithms rather than the 91 pairs of all algorithms (with and without Laplace correction).

5.3.2 Results on the bupa Domain

Figure 5.6 and Tables 5.10, 5.11, 5.12, 5.13, 5.14, 5.15 and 5.16 present the test set results for the bupa domain. Figure 5.6 shows the V_{test} for each algorithm averaged over the 20 replicas. The error bars show a 95% normal confidence interval for the mean. The tables report the (wins, ties, and losses) according to the BDeltaCost test for each pair of algorithms, summed over the 20 replicas. From these results, we note the following:

- The best algorithm is ES-L, although its confidence interval overlaps the confidence intervals for all the other systematic algorithms and the confidence inter-

TABLE 5.8: The effect of Laplace correction on each greedy search algorithm, across all domains. Each table entry has (wins, ties, losses) of algorithm 1 (alg1) over algorithm 2 (alg2), summed over all replicas.

alg1	alg2	domain	MC1	MC2	MC3	MC4	MC5
VOI-L	VOI	bupa	1, 19, 0	1, 19, 0	2, 18, 0	1, 19, 0	2, 18, 0
VOI-L	VOI	pima	0, 20, 0	1, 18, 1	2, 17, 1	1, 18, 1	3, 16, 1
VOI-L	VOI	heart	0, 20, 0	2, 18, 0	3, 17, 0	2, 18, 0	2, 18, 0
VOI-L	VOI	b-can	0, 20, 0	0, 20, 0	6, 14, 0	1, 19, 0	2, 17, 1
VOI-L	VOI	spect	1, 19, 0	3, 17, 0	3, 17, 0	0, 19, 1	0, 19, 1
MC-N-L	MC-N	bupa	1, 19, 0	1, 19, 0	2, 18, 0	2, 18, 0	0, 20, 0
MC-N-L	MC-N	pima	1, 19, 0	8, 12, 0	16, 4, 0	8, 12, 0	2, 18, 0
MC-N-L	MC-N	heart	0, 20, 0	8, 12, 0	7, 13, 0	1, 19, 0	1, 19, 0
MC-N-L	MC-N	b-can	0, 20, 0	0, 20, 0	2, 18, 0	3, 17, 0	12, 8, 0
MC-N-L	MC-N	spect	1, 19, 0	5, 15, 0	6, 14, 0	11, 8, 1	4, 16, 0
Nor-L	Nor	bupa	6, 14, 0	6, 14, 0	4, 16, 0	4, 16, 0	2, 18, 0
Nor-L	Nor	pima	20, 0, 0	19, 1, 0	15, 5, 0	10, 10, 0	2, 16, 2
Nor-L	Nor	heart	20, 0, 0	6, 14, 0	3, 17, 0	0, 20, 0	0, 20, 0
Nor-L	Nor	b-can	18, 2, 0	17, 3, 0	15, 5, 0	15, 5, 0	7, 13, 0
Nor-L	Nor	spect	6, 14, 0	2, 18, 0	1, 19, 0	0, 20, 0	0, 13, 7

TABLE 5.9: The effect of Laplace correction on each systematic search algorithm, across all domains. Each table entry has (wins, ties, losses) of algorithm 1 (alg1) over algorithm 2 (alg2), summed over all replicas.

alg1	alg2	domain	MC1	MC2	MC3	MC4	MC5
AO*-L	AO*	bupa	4, 16, 0	4, 16, 0	4, 16, 0	4, 16, 0	5, 15, 0
AO*-L	AO*	pima	3, 17, 0	9, 11, 0	15, 5, 0	11, 9, 0	8, 12, 0
AO*-L	AO*	heart	0, 20, 0	11, 9, 0	13, 7, 0	2, 18, 0	2, 18, 0
AO*-L	AO*	b-can	0, 20, 0	0, 20, 0	3, 16, 1	2, 18, 0	12, 8, 0
AO*-L	AO*	spect	6, 14, 0	10, 10, 0	10, 10, 0	5, 15, 0	0, 19, 1
ES-L	ES	bupa	3, 17, 0	3, 17, 0	1, 19, 0	1, 19, 0	2, 18, 0
ES-L	ES	pima	0, 20, 0	0, 19, 1	3, 17, 0	1, 19, 0	10, 10, 0
ES-L	ES	heart	1, 19, 0	5, 15, 0	3, 17, 0	1, 19, 0	1, 19, 0
ES-L	ES	b-can	0, 20, 0	0, 20, 0	2, 17, 1	4, 12, 4	3, 15, 2
ES-L	ES	spect	0, 20, 0	0, 20, 0	0, 20, 0	0, 20, 0	1, 19, 0
SP-L	SP	bupa	0, 20, 0	0, 20, 0	3, 17, 0	3, 17, 0	3, 17, 0
SP-L	SP	pima	0, 20, 0	4, 16, 0	13, 7, 0	6, 14, 0	8, 12, 0
SP-L	SP	heart	0, 20, 0	7, 13, 0	5, 15, 0	2, 17, 1	2, 17, 1
SP-L	SP	b-can	0, 20, 0	0, 20, 0	0, 19, 1	0, 19, 1	4, 16, 0
SP-L	SP	spect	2, 18, 0	6, 14, 0	5, 15, 0	7, 13, 0	2, 18, 0
PPP-L	PPP	bupa	3, 17, 0	3, 17, 0	4, 16, 0	4, 16, 0	1, 19, 0
PPP-L	PPP	pima	0, 20, 0	1, 18, 1	1, 18, 1	1, 18, 1	7, 13, 0
PPP-L	PPP	heart	0, 19, 1	1, 19, 0	0, 20, 0	1, 18, 1	1, 18, 1
PPP-L	PPP	b-can	0, 20, 0	0, 20, 0	1, 18, 1	1, 19, 0	9, 11, 0
PPP-L	PPP	spect	7, 13, 0	9, 11, 0	7, 13, 0	7, 13, 0	0, 19, 1

vals for VOI and VOI-L. The BDeltaCost Table 5.14 confirms the superiority of ES-L. This is the only domain in the thesis where one algorithm never loses to any other algorithm.

- The worst algorithm is Nor. In general, the confidence intervals of the systematic algorithms were below the confidence intervals of greedy algorithms, with the exception of VOI and VOI-L. Even with the Laplace correction, Nor-L and MC-N-L fared badly (see Tables 5.12 and 5.11).

- The misclassification costs do not change the best and the worst algorithm on bupa.

- All other algorithms have similar performance. SP-L and PPP-L are slightly better than the other algorithms according to BDeltaCost (Tables 5.15 and 5.16), followed by VOI-L and AO*-L (Tables 5.10 and 5.13).

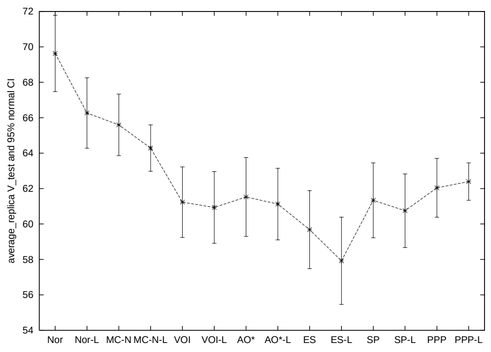
- The Laplace correction slightly improved average V_{test} for most algorithms, although the confidence intervals for an algorithm and its Laplace correction overlap.

5.3.3 Results on the pima Domain

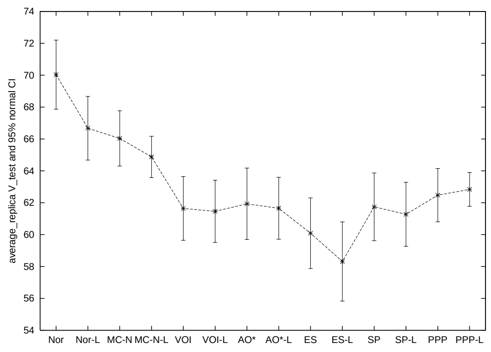
Figure 5.7 and Tables 5.17, 5.18, 5.19, 5.20, 5.21, 5.22 and 5.23 present the test set results for the pima domain. From these results, we note the following:

- The best algorithm changes depending on the misclassification costs. VOI and VOI-L are consistently good, and on the misclassification cost level MC5 they are by far the best algorithms. Several other algorithms are good. For example, on MC2 Nor-L and MC-N-L are slightly better than VOI-L (see Table 5.17).

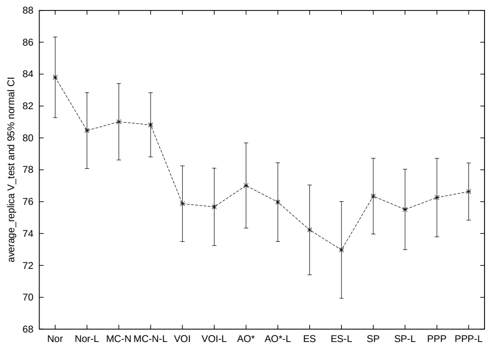
- The worst algorithm depends on misclassification costs. On MC1 and MC2, it is Nor; on medium and large MC, it is AO*.



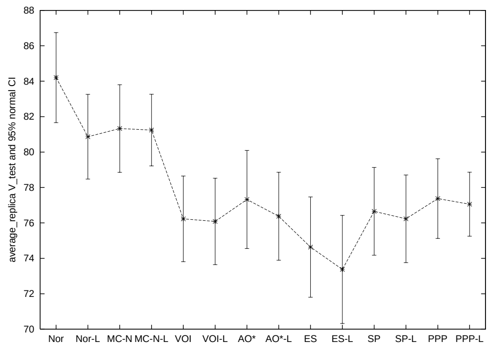
(a) MC1



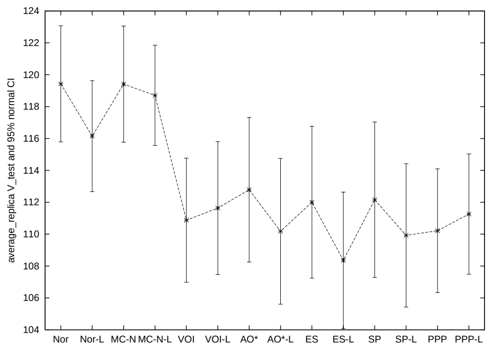
(b) MC2



(c) MC3



(d) MC4



(e) MC5

FIGURE 5.6: Bupa domain. Graphs of average V_{test} over replicas, and its 95% normal confidence interval (CI). Note that MC1 $\simeq$ MC2, MC3 $\simeq$ MC4.

TABLE 5.10: Bupa, BDeltaCost of VOI-L paired with each of the Laplace corrected algorithms. Each table entry has (wins, ties, losses) of alg1 over alg2.

alg1	alg2	MC1	MC2	MC3	MC4	MC5
VOI-L	MC-N-L	2, 18, 0	2, 18, 0	3, 17, 0	3, 17, 0	5, 15, 0
VOI-L	Nor-L	8, 12, 0	8, 12, 0	6, 14, 0	6, 14, 0	3, 17, 0
VOI-L	AO*-L	0, 20, 0	0, 20, 0	0, 20, 0	0, 20, 0	2, 18, 0
VOI-L	ES-L	0, 15, 5	0, 15, 5	0, 16, 4	0, 16, 4	0, 17, 3
VOI-L	SP-L	0, 16, 4	0, 16, 4	0, 18, 2	0, 20, 0	1, 19, 0
VOI-L	PPP-L	0, 17, 3	0, 17, 3	0, 17, 3	0, 17, 3	0, 19, 1

TABLE 5.11: Bupa, BDeltaCost of MC-N-L paired with each of the Laplace corrected algorithms. Each table entry has (wins, ties, losses) of alg1 over alg2.

alg1	alg2	MC1	MC2	MC3	MC4	MC5
MC-N-L	VOI-L	0, 18, 2	0, 18, 2	0, 17, 3	0, 17, 3	0, 15, 5
MC-N-L	Nor-L	2, 16, 2	2, 16, 2	1, 17, 2	1, 17, 2	0, 18, 2
MC-N-L	AO*-L	0, 18, 2	0, 18, 2	0, 17, 3	0, 17, 3	0, 15, 5
MC-N-L	ES-L	0, 16, 4	0, 16, 4	0, 14, 6	0, 14, 6	0, 15, 5
MC-N-L	SP-L	0, 17, 3	0, 17, 3	0, 15, 5	0, 17, 3	0, 14, 6
MC-N-L	PPP-L	0, 18, 2	0, 18, 2	0, 15, 5	0, 15, 5	0, 17, 3

TABLE 5.12: Bupa, BDeltaCost of Nor-L paired with each of the Laplace corrected algorithms. Each table entry has (wins, ties, losses) of alg1 over alg2.

alg1	alg2	MC1	MC2	MC3	MC4	MC5
Nor-L	VOI-L	0, 12, 8	0, 12, 8	0, 14, 6	0, 14, 6	0, 17, 3
Nor-L	MC-N-L	2, 16, 2	2, 16, 2	2, 17, 1	2, 17, 1	2, 18, 0
Nor-L	AO*-L	0, 12, 8	0, 12, 8	0, 15, 5	0, 15, 5	0, 19, 1
Nor-L	ES-L	0, 10, 10	0, 10, 10	0, 16, 4	0, 16, 4	0, 17, 3
Nor-L	SP-L	0, 10, 10	0, 10, 10	0, 14, 6	0, 15, 5	0, 18, 2
Nor-L	PPP-L	0, 14, 6	0, 14, 6	0, 14, 6	0, 14, 6	0, 16, 4

TABLE 5.13: Bupa, BDeltaCost of AO*-L paired with each of the Laplace corrected algorithms. Each table entry has (wins, ties, losses) of alg1 over alg2.

alg1	alg2	MC1	MC2	MC3	MC4	MC5
AO*-L	VOI-L	0, 20, 0	0, 20, 0	0, 20, 0	0, 20, 0	0, 18, 2
AO*-L	MC-N-L	2, 18, 0	2, 18, 0	3, 17, 0	3, 17, 0	5, 15, 0
AO*-L	Nor-L	8, 12, 0	8, 12, 0	5, 15, 0	5, 15, 0	1, 19, 0
AO*-L	ES-L	0, 15, 5	0, 15, 5	0, 16, 4	0, 16, 4	0, 18, 2
AO*-L	SP-L	0, 16, 4	0, 16, 4	0, 18, 2	0, 20, 0	0, 19, 1
AO*-L	PPP-L	0, 17, 3	0, 17, 3	0, 17, 3	0, 17, 3	0, 17, 3

TABLE 5.14: Bupa, BDeltaCost of ES-L paired with each of the Laplace corrected algorithms. Each table entry has (wins, ties, losses) of alg1 over alg2.

alg1	alg2	MC1	MC2	MC3	MC4	MC5
ES-L	VOI-L	5, 15, 0	5, 15, 0	4, 16, 0	4, 16, 0	3, 17, 0
ES-L	MC-N-L	4, 16, 0	4, 16, 0	6, 14, 0	6, 14, 0	5, 15, 0
ES-L	Nor-L	10, 10, 0	10, 10, 0	4, 16, 0	4, 16, 0	3, 17, 0
ES-L	AO*-L	5, 15, 0	5, 15, 0	4, 16, 0	4, 16, 0	2, 18, 0
ES-L	SP-L	1, 19, 0	1, 19, 0	0, 20, 0	3, 17, 0	2, 18, 0
ES-L	PPP-L	1, 19, 0	1, 19, 0	0, 20, 0	0, 20, 0	0, 20, 0

TABLE 5.15: Bupa, BDeltaCost of SP-L paired with each of the Laplace corrected algorithms. Each table entry has (wins, ties, losses) of alg1 over alg2.

alg1	alg2	MC1	MC2	MC3	MC4	MC5
SP-L	VOI-L	4, 16, 0	4, 16, 0	2, 18, 0	0, 20, 0	0, 19, 1
SP-L	MC-N-L	3, 17, 0	3, 17, 0	5, 15, 0	3, 17, 0	6, 14, 0
SP-L	Nor-L	10, 10, 0	10, 10, 0	6, 14, 0	5, 15, 0	2, 18, 0
SP-L	AO*-L	4, 16, 0	4, 16, 0	2, 18, 0	0, 20, 0	1, 19, 0
SP-L	ES-L	0, 19, 1	0, 19, 1	0, 20, 0	0, 17, 3	0, 18, 2
SP-L	PPP-L	0, 20, 0	0, 20, 0	0, 20, 0	0, 18, 2	0, 18, 2

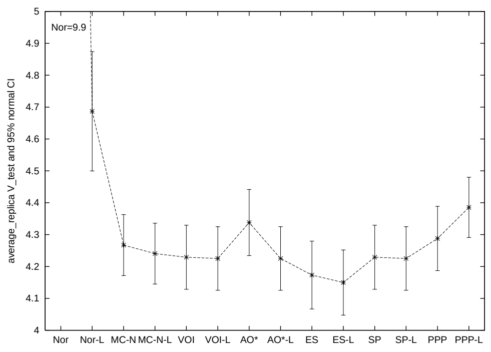
TABLE 5.16: Bupa, BDeltaCost of PPP-L paired with each of the Laplace corrected algorithms. Each table entry has (wins, ties, losses) of alg1 over alg2.

alg1	alg2	MC1	MC2	MC3	MC4	MC5
PPP-L	VOI-L	3, 17, 0	3, 17, 0	3, 17, 0	3, 17, 0	1, 19, 0
PPP-L	MC-N-L	2, 18, 0	2, 18, 0	5, 15, 0	5, 15, 0	3, 17, 0
PPP-L	Nor-L	6, 14, 0	6, 14, 0	6, 14, 0	6, 14, 0	4, 16, 0
PPP-L	AO*-L	3, 17, 0	3, 17, 0	3, 17, 0	3, 17, 0	3, 17, 0
PPP-L	ES-L	0, 19, 1	0, 19, 1	0, 20, 0	0, 20, 0	0, 20, 0
PPP-L	SP-L	0, 20, 0	0, 20, 0	0, 20, 0	2, 18, 0	2, 18, 0

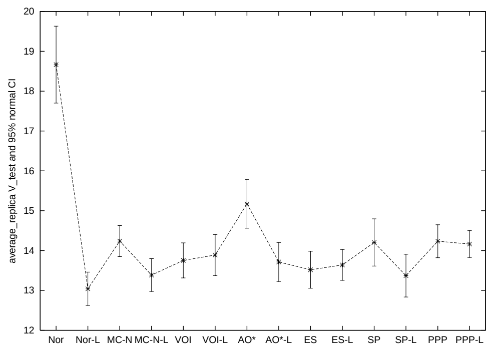
- AO* performed badly on pima. All the regularizers helped it, except on MC5, where ES did not. These findings, according to the normal confidence intervals, are confirmed by BDeltaCost results (Appendix Table B.7).

- Among the regularized AO* methods, SP-L and ES-L are the best (see Tables 5.22 and 5.21). SP-L has several wins over AO*-L, ES-L and PPP-L.

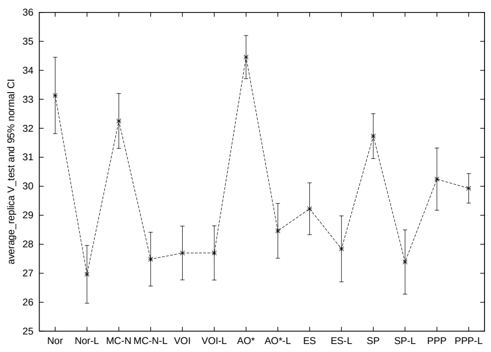
In [2] we presented experiments on a version of the pima problem, with a binary discretization of the attributes and slightly different misclassification costs (100 for false negatives and 80 for false positives), and we concluded that SP was slightly better than AO*. On the current version of pima, we confirm these results: BDeltaCost produces 3, 6, 5, 3, and 4 wins of SP over AO*, with no losses, for each of the 5 MCs. However, SP is no longer the best algorithm: BDeltaCost shows that VOI-L has 0, 2, 10, 8, and 11 wins over SP, with no losses, and the Laplace version SP-L wins over SP on 0, 4, 13, 6, and 8 replicas with no losses. Table 5.17 comparing VOI-L and SP-L shows VOI-L losing on 2 replicas on MC2 and winning on more replicas on large MCs. In general, there is no single best algorithm across all MCs, but VOI-L is probably



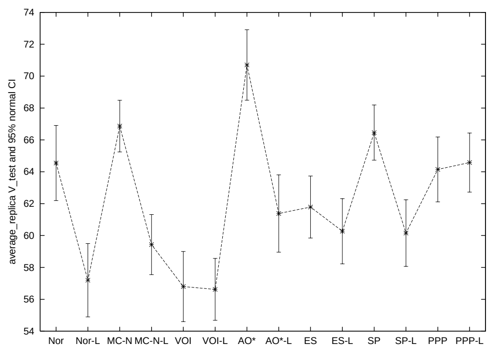
(a) MC1



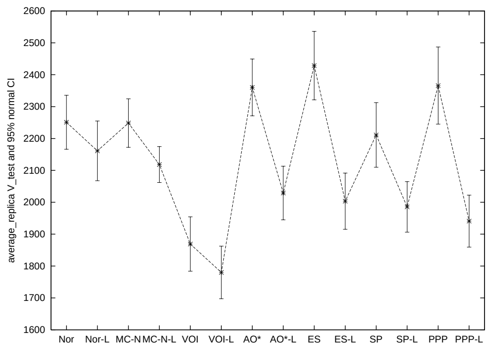
(b) MC2



(c) MC3



(d) MC4



(e) MC5

FIGURE 5.7: Pima domain. Graphs of average V_{test} over replicas, and its 95% normal confidence interval (CI).

TABLE 5.17: Pima, BDeltaCost of VOI-L paired with each of the Laplace corrected algorithms. Each table entry has (wins, ties, losses) of alg1 over alg2.

alg1	alg2	MC1	MC2	MC3	MC4	MC5
VOI-L	MC-N-L	0, 20, 0	0, 17, 3	0, 19, 1	2, 18, 0	5, 15, 0
VOI-L	Nor-L	4, 16, 0	0, 18, 2	1, 18, 1	1, 19, 0	6, 14, 0
VOI-L	AO*-L	0, 20, 0	1, 17, 2	1, 19, 0	3, 17, 0	8, 12, 0
VOI-L	ES-L	1, 19, 0	1, 17, 2	2, 18, 0	3, 17, 0	5, 15, 0
VOI-L	SP-L	0, 20, 0	0, 18, 2	0, 20, 0	3, 17, 0	4, 16, 0
VOI-L	PPP-L	0, 20, 0	0, 20, 0	1, 19, 0	7, 13, 0	3, 16, 1

TABLE 5.18: Pima, BDeltaCost of MC-N-L paired with each of the Laplace corrected algorithms. Each table entry has (wins, ties, losses) of alg1 over alg2.

alg1	alg2	MC1	MC2	MC3	MC4	MC5
MC-N-L	VOI-L	0, 20, 0	3, 17, 0	1, 19, 0	0, 18, 2	0, 15, 5
MC-N-L	Nor-L	4, 16, 0	1, 19, 0	2, 16, 2	0, 18, 2	3, 16, 1
MC-N-L	AO*-L	0, 20, 0	3, 16, 1	3, 17, 0	2, 18, 0	0, 19, 1
MC-N-L	ES-L	1, 17, 2	1, 18, 1	2, 16, 2	1, 18, 1	1, 19, 0
MC-N-L	SP-L	0, 20, 0	0, 19, 1	2, 17, 1	1, 19, 0	0, 19, 1
MC-N-L	PPP-L	0, 20, 0	2, 18, 0	1, 19, 0	4, 16, 0	0, 19, 1

TABLE 5.19: Pima, BDeltaCost of Nor-L paired with each of the Laplace corrected algorithms. Each table entry has (wins, ties, losses) of alg1 over alg2.

alg1	alg2	MC1	MC2	MC3	MC4	MC5
Nor-L	VOI-L	0, 16, 4	2, 18, 0	1, 18, 1	0, 19, 1	0, 14, 6
Nor-L	MC-N-L	0, 16, 4	0, 19, 1	2, 16, 2	2, 18, 0	1, 16, 3
Nor-L	AO*-L	0, 16, 4	1, 19, 0	4, 15, 1	3, 17, 0	1, 16, 3
Nor-L	ES-L	0, 14, 6	4, 16, 0	1, 17, 2	1, 18, 1	0, 18, 2
Nor-L	SP-L	0, 16, 4	0, 19, 1	0, 19, 1	2, 18, 0	0, 17, 3
Nor-L	PPP-L	0, 19, 1	2, 18, 0	2, 18, 0	5, 15, 0	0, 15, 5

TABLE 5.20: Pima, BDeltaCost of AO*-L paired with each of the Laplace corrected algorithms. Each table entry has (wins, ties, losses) of alg1 over alg2.

alg1	alg2	MC1	MC2	MC3	MC4	MC5
AO*-L	VOI-L	0, 20, 0	2, 17, 1	0, 19, 1	0, 17, 3	0, 12, 8
AO*-L	MC-N-L	0, 20, 0	1, 16, 3	0, 17, 3	0, 18, 2	1, 19, 0
AO*-L	Nor-L	4, 16, 0	0, 19, 1	1, 15, 4	0, 17, 3	3, 16, 1
AO*-L	ES-L	1, 19, 0	2, 17, 1	1, 18, 1	0, 18, 2	1, 17, 2
AO*-L	SP-L	0, 20, 0	1, 15, 4	0, 15, 5	1, 17, 2	1, 15, 4
AO*-L	PPP-L	0, 20, 0	1, 19, 0	0, 20, 0	4, 16, 0	0, 18, 2

TABLE 5.21: Pima, BDeltaCost of ES-L paired with each of the Laplace corrected algorithms. Each table entry has (wins, ties, losses) of alg1 over alg2.

alg1	alg2	MC1	MC2	MC3	MC4	MC5
ES-L	VOI-L	0, 19, 1	2, 17, 1	0, 18, 2	0, 17, 3	0, 15, 5
ES-L	MC-N-L	2, 17, 1	1, 18, 1	2, 16, 2	1, 18, 1	0, 19, 1
ES-L	Nor-L	6, 14, 0	0, 16, 4	2, 17, 1	1, 18, 1	2, 18, 0
ES-L	AO*-L	0, 19, 1	1, 17, 2	1, 18, 1	2, 18, 0	2, 17, 1
ES-L	SP-L	0, 19, 1	1, 18, 1	0, 18, 2	1, 19, 0	1, 18, 1
ES-L	PPP-L	3, 16, 1	0, 20, 0	3, 16, 1	2, 17, 1	1, 17, 2

TABLE 5.22: Pima, BDeltaCost of SP-L paired with each of the Laplace corrected algorithms. Each table entry has (wins, ties, losses) of alg1 over alg2.

alg1	alg2	MC1	MC2	MC3	MC4	MC5
SP-L	VOI-L	0, 20, 0	2, 18, 0	0, 20, 0	0, 17, 3	0, 16, 4
SP-L	MC-N-L	0, 20, 0	1, 19, 0	1, 17, 2	0, 19, 1	1, 19, 0
SP-L	Nor-L	4, 16, 0	1, 19, 0	1, 19, 0	0, 18, 2	3, 17, 0
SP-L	AO*-L	0, 20, 0	4, 15, 1	5, 15, 0	2, 17, 1	4, 15, 1
SP-L	ES-L	1, 19, 0	1, 18, 1	2, 18, 0	0, 19, 1	1, 18, 1
SP-L	PPP-L	0, 20, 0	2, 18, 0	3, 17, 0	3, 17, 0	0, 19, 1

TABLE 5.23: Pima, BDeltaCost of PPP-L paired with each of the Laplace corrected algorithms. Each table entry has (wins, ties, losses) of alg1 over alg2.

alg1	alg2	MC1	MC2	MC3	MC4	MC5
PPP-L	VOI-L	0, 20, 0	0, 20, 0	0, 19, 1	0, 13, 7	1, 16, 3
PPP-L	MC-N-L	0, 20, 0	0, 18, 2	0, 19, 1	0, 16, 4	1, 19, 0
PPP-L	Nor-L	1, 19, 0	0, 18, 2	0, 18, 2	0, 15, 5	5, 15, 0
PPP-L	AO*-L	0, 20, 0	0, 19, 1	0, 20, 0	0, 16, 4	2, 18, 0
PPP-L	ES-L	1, 16, 3	0, 20, 0	1, 16, 3	1, 17, 2	2, 17, 1
PPP-L	SP-L	0, 20, 0	0, 18, 2	0, 17, 3	0, 17, 3	1, 19, 0

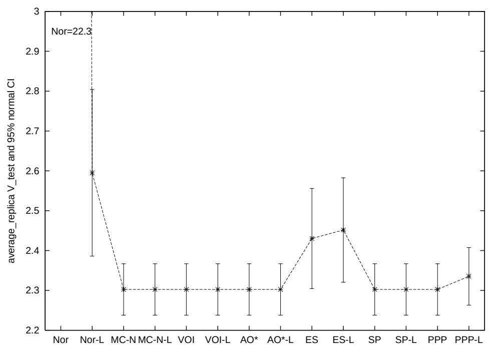
the most robust on this problem.

5.3.4 Results on the heart Domain

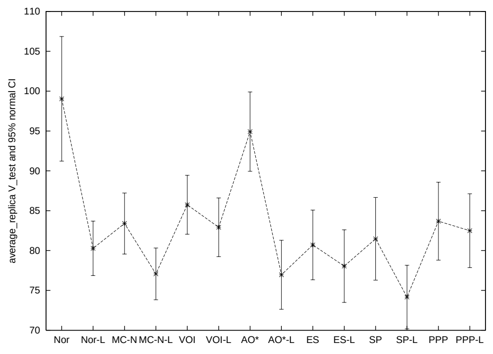
Figure 5.8 and Tables 5.24, 5.25, 5.26, 5.27, 5.28, 5.29 and 5.30 present the test set results for the heart domain. From these results, we note the following:

- The best algorithm is SP-L (see Table 5.29). From the greedy algorithms, MC-N-L comes next (see Table 5.25).

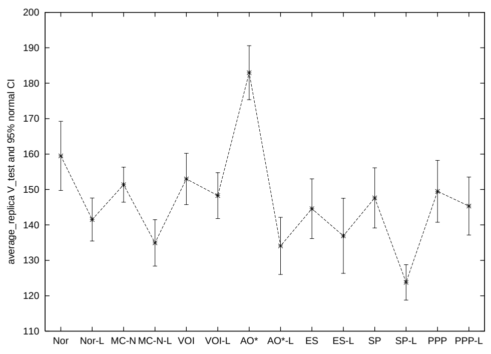
- The worst algorithm is Nor on small misclassification costs, and VOI and VOI-L on large misclassification costs. AO* performed badly for every MC. All the regularizers significantly helped AO*, although AO*-L was not particularly good on heart (Table 5.27). VOI-L also fared badly on heart, according to Figure 5.8, although the BDeltaCost in Table 5.24 finds it mostly tied with the best algorithms on heart (SP-L and MC-N-L).



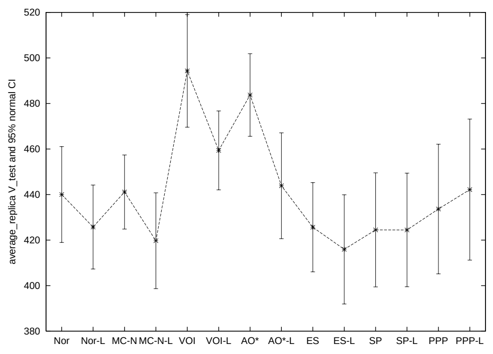
(a) MC1



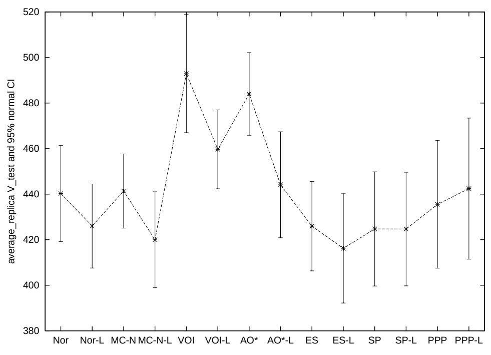
(b) MC2



(c) MC3



(d) MC4



(e) MC5

FIGURE 5.8: Heart domain. Graphs of average V_{test} over replicas, and its 95% normal confidence interval (CI). Note that MC4 $\simeq$ MC5.

TABLE 5.24: Heart, BDeltaCost of VOI-L paired with each of the Laplace corrected algorithms. Each table entry has (wins, ties, losses) of alg1 over alg2.

alg1	alg2	MC1	MC2	MC3	MC4	MC5
VOI-L	MC-N-L	0, 20, 0	0, 20, 0	0, 20, 0	0, 19, 1	0, 19, 1
VOI-L	Nor-L	6, 14, 0	0, 17, 3	0, 15, 5	0, 17, 3	0, 17, 3
VOI-L	AO*-L	0, 20, 0	0, 19, 1	0, 20, 0	0, 19, 1	0, 19, 1
VOI-L	ES-L	2, 18, 0	0, 19, 1	1, 18, 1	0, 20, 0	0, 20, 0
VOI-L	SP-L	0, 20, 0	0, 18, 2	0, 19, 1	0, 20, 0	0, 20, 0
VOI-L	PPP-L	1, 19, 0	0, 17, 3	0, 17, 3	0, 17, 3	0, 17, 3

TABLE 5.25: Heart, BDeltaCost of MC-N-L paired with each of the Laplace corrected algorithms. Each table entry has (wins, ties, losses) of alg1 over alg2.

alg1	alg2	MC1	MC2	MC3	MC4	MC5
MC-N-L	VOI-L	0, 20, 0	0, 20, 0	0, 20, 0	1, 19, 0	1, 19, 0
MC-N-L	Nor-L	6, 14, 0	0, 20, 0	0, 20, 0	0, 20, 0	0, 20, 0
MC-N-L	AO*-L	0, 20, 0	3, 16, 1	2, 18, 0	2, 17, 1	2, 17, 1
MC-N-L	ES-L	2, 18, 0	2, 17, 1	2, 16, 2	0, 20, 0	0, 20, 0
MC-N-L	SP-L	0, 20, 0	1, 17, 2	1, 17, 2	1, 19, 0	1, 19, 0
MC-N-L	PPP-L	1, 19, 0	1, 19, 0	1, 19, 0	0, 20, 0	0, 20, 0

TABLE 5.26: Heart, BDeltaCost of Nor-L paired with each of the Laplace corrected algorithms. Each table entry has (wins, ties, losses) of alg1 over alg2.

alg1	alg2	MC1	MC2	MC3	MC4	MC5
Nor-L	VOI-L	0, 14, 6	3, 17, 0	5, 15, 0	3, 17, 0	3, 17, 0
Nor-L	MC-N-L	0, 14, 6	0, 20, 0	0, 20, 0	0, 20, 0	0, 20, 0
Nor-L	AO*-L	0, 14, 6	0, 19, 1	0, 20, 0	0, 19, 1	0, 19, 1
Nor-L	ES-L	1, 14, 5	0, 20, 0	1, 19, 0	0, 20, 0	0, 20, 0
Nor-L	SP-L	0, 14, 6	0, 18, 2	0, 19, 1	0, 20, 0	0, 20, 0
Nor-L	PPP-L	0, 15, 5	1, 19, 0	0, 20, 0	1, 19, 0	1, 19, 0

TABLE 5.27: Heart, BDeltaCost of AO*-L paired with each of the Laplace corrected algorithms. Each table entry has (wins, ties, losses) of alg1 over alg2.

alg1	alg2	MC1	MC2	MC3	MC4	MC5
AO*-L	VOI-L	0, 20, 0	1, 19, 0	0, 20, 0	1, 19, 0	1, 19, 0
AO*-L	MC-N-L	0, 20, 0	1, 16, 3	0, 18, 2	1, 17, 2	1, 17, 2
AO*-L	Nor-L	6, 14, 0	1, 19, 0	0, 20, 0	1, 19, 0	1, 19, 0
AO*-L	ES-L	2, 18, 0	2, 17, 1	2, 17, 1	1, 19, 0	1, 19, 0
AO*-L	SP-L	0, 20, 0	1, 16, 3	1, 15, 4	1, 13, 6	1, 13, 6
AO*-L	PPP-L	1, 19, 0	2, 18, 0	1, 19, 0	2, 18, 0	2, 18, 0

TABLE 5.28: Heart, BDeltaCost of ES-L paired with each of the Laplace corrected algorithms. Each table entry has (wins, ties, losses) of alg1 over alg2.

alg1	alg2	MC1	MC2	MC3	MC4	MC5
ES-L	VOI-L	0, 18, 2	1, 19, 0	1, 18, 1	0, 20, 0	0, 20, 0
ES-L	MC-N-L	0, 18, 2	1, 17, 2	2, 16, 2	0, 20, 0	0, 20, 0
ES-L	Nor-L	5, 14, 1	0, 20, 0	0, 19, 1	0, 20, 0	0, 20, 0
ES-L	AO*-L	0, 18, 2	1, 17, 2	1, 17, 2	0, 19, 1	0, 19, 1
ES-L	SP-L	0, 18, 2	0, 16, 4	0, 16, 4	0, 18, 2	0, 18, 2
ES-L	PPP-L	1, 17, 2	2, 18, 0	1, 19, 0	2, 17, 1	2, 17, 1

TABLE 5.29: Heart, BDeltaCost of SP-L paired with each of the Laplace corrected algorithms. Each table entry has (wins, ties, losses) of alg1 over alg2.

alg1	alg2	MC1	MC2	MC3	MC4	MC5
SP-L	VOI-L	0, 20, 0	2, 18, 0	1, 19, 0	0, 20, 0	0, 20, 0
SP-L	MC-N-L	0, 20, 0	2, 17, 1	2, 17, 1	0, 19, 1	0, 19, 1
SP-L	Nor-L	6, 14, 0	2, 18, 0	1, 19, 0	0, 20, 0	0, 20, 0
SP-L	AO*-L	0, 20, 0	3, 16, 1	4, 15, 1	6, 13, 1	6, 13, 1
SP-L	ES-L	2, 18, 0	4, 16, 0	4, 16, 0	2, 18, 0	2, 18, 0
SP-L	PPP-L	1, 19, 0	4, 16, 0	4, 16, 0	1, 18, 1	1, 18, 1

TABLE 5.30: Heart, BDeltaCost of PPP-L paired with each of the Laplace corrected algorithms. Each table entry has (wins, ties, losses) of alg1 over alg2.

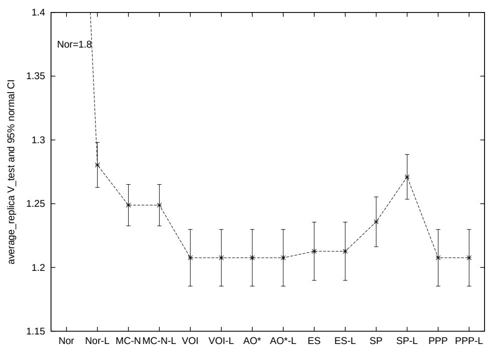
alg1	alg2	MC1	MC2	MC3	MC4	MC5
PPP-L	VOI-L	0, 19, 1	3, 17, 0	3, 17, 0	3, 17, 0	3, 17, 0
PPP-L	MC-N-L	0, 19, 1	0, 19, 1	0, 19, 1	0, 20, 0	0, 20, 0
PPP-L	Nor-L	5, 15, 0	0, 19, 1	0, 20, 0	0, 19, 1	0, 19, 1
PPP-L	AO*-L	0, 19, 1	0, 18, 2	0, 19, 1	0, 18, 2	0, 18, 2
PPP-L	ES-L	2, 17, 1	0, 18, 2	0, 19, 1	1, 17, 2	1, 17, 2
PPP-L	SP-L	0, 19, 1	0, 16, 4	0, 16, 4	1, 18, 1	1, 18, 1

5.3.5 *Results on the breast-cancer Domain*

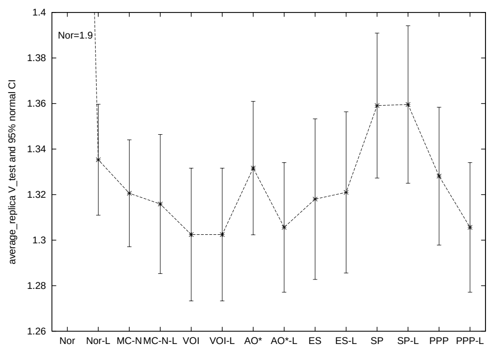
There are some discrepancies between the results of Figure 5.9 and the BDeltaCost Tables 5.31, 5.32, 5.33, 5.34, 5.35, 5.36 and 5.37, therefore we need to look at both of them to draw conclusions.

- The best algorithm changes depending on misclassification costs. MC-N-L and SP-L are consistently good according to the BDeltaCost Tables 5.32 and 5.36. VOI-L is good for small misclassification costs.

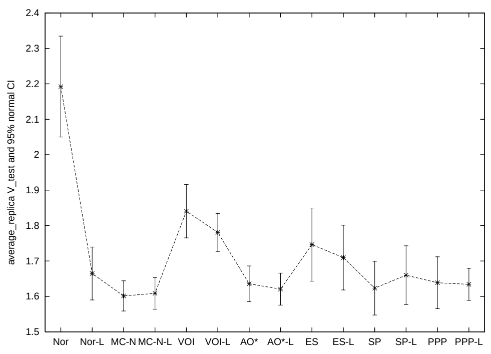
- The worst algorithm changes depending on misclassification costs. On all misclassification costs except MC5, Nor's confidence interval is above the others. AO* was significantly bad on MC5. VOI-L and ES-L performed badly on larger misclassification costs (see Tables 5.31 and 5.35).



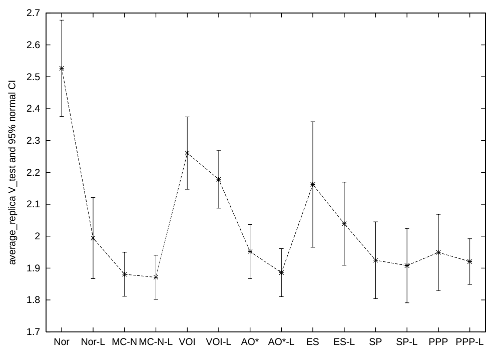
(a) MC1



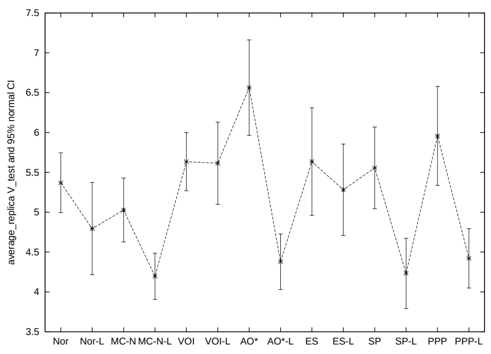
(b) MC2



(c) MC3



(d) MC4



(e) MC5

FIGURE 5.9: Breast-cancer domain. Graphs of average V_{test} over replicas, and its 95% normal confidence interval (CI). Note that MC1 $\simeq$ MC2.

TABLE 5.31: Breast-cancer, BDeltaCost of VOI-L paired with each of the Laplace corrected algorithms. Each table entry has (wins, ties, losses) of alg1 over alg2.

alg1	alg2	MC1	MC2	MC3	MC4	MC5
VOI-L	MC-N-L	1, 19, 0	1, 19, 0	0, 17, 3	0, 14, 6	0, 14, 6
VOI-L	Nor-L	7, 13, 0	0, 20, 0	0, 18, 2	0, 16, 4	0, 14, 6
VOI-L	AO*-L	0, 20, 0	0, 20, 0	0, 18, 2	0, 14, 6	0, 16, 4
VOI-L	ES-L	0, 19, 1	0, 20, 0	1, 18, 1	0, 17, 3	0, 19, 1
VOI-L	SP-L	1, 19, 0	1, 19, 0	0, 20, 0	0, 14, 6	0, 15, 5
VOI-L	PPP-L	0, 20, 0	0, 20, 0	0, 20, 0	0, 17, 3	0, 17, 3

TABLE 5.32: Breast-cancer, BDeltaCost of MC-N-L paired with each of the Laplace corrected algorithms. Each table entry has (wins, ties, losses) of alg1 over alg2.

alg1	alg2	MC1	MC2	MC3	MC4	MC5
MC-N-L	VOI-L	0, 19, 1	0, 19, 1	3, 17, 0	6, 14, 0	6, 14, 0
MC-N-L	Nor-L	0, 20, 0	0, 20, 0	2, 18, 0	2, 17, 1	3, 15, 2
MC-N-L	AO*-L	0, 19, 1	0, 19, 1	1, 19, 0	1, 19, 0	6, 14, 0
MC-N-L	ES-L	0, 19, 1	1, 18, 1	2, 18, 0	4, 16, 0	4, 16, 0
MC-N-L	SP-L	0, 20, 0	0, 20, 0	2, 18, 0	2, 17, 1	1, 18, 1
MC-N-L	PPP-L	0, 19, 1	0, 19, 1	1, 19, 0	2, 18, 0	2, 16, 2

TABLE 5.33: Breast-cancer, BDeltaCost of Nor-L paired with each of the Laplace corrected algorithms. Each table entry has (wins, ties, losses) of alg1 over alg2.

alg1	alg2	MC1	MC2	MC3	MC4	MC5
Nor-L	VOI-L	0, 13, 7	0, 20, 0	2, 18, 0	4, 16, 0	6, 14, 0
Nor-L	MC-N-L	0, 20, 0	0, 20, 0	0, 18, 2	1, 17, 2	2, 15, 3
Nor-L	AO*-L	0, 13, 7	0, 20, 0	0, 18, 2	2, 17, 1	4, 13, 3
Nor-L	ES-L	0, 14, 6	1, 19, 0	2, 17, 1	3, 16, 1	3, 15, 2
Nor-L	SP-L	0, 19, 1	1, 19, 0	0, 19, 1	1, 19, 0	2, 15, 3
Nor-L	PPP-L	0, 13, 7	0, 20, 0	0, 19, 1	2, 17, 1	3, 13, 4

TABLE 5.34: Breast-cancer, BDeltaCost of AO*-L paired with each of the Laplace corrected algorithms. Each table entry has (wins, ties, losses) of alg1 over alg2.

alg1	alg2	MC1	MC2	MC3	MC4	MC5
AO*-L	VOI-L	0, 20, 0	0, 20, 0	2, 18, 0	6, 14, 0	4, 16, 0
AO*-L	MC-N-L	1, 19, 0	1, 19, 0	0, 19, 1	0, 19, 1	0, 14, 6
AO*-L	Nor-L	7, 13, 0	0, 20, 0	2, 18, 0	1, 17, 2	3, 13, 4
AO*-L	ES-L	0, 19, 1	0, 20, 0	1, 19, 0	4, 16, 0	4, 14, 2
AO*-L	SP-L	1, 19, 0	1, 19, 0	1, 19, 0	1, 18, 1	0, 14, 6
AO*-L	PPP-L	0, 20, 0	0, 20, 0	0, 20, 0	1, 19, 0	2, 13, 5

TABLE 5.35: Breast-cancer, BDeltaCost of ES-L paired with each of the Laplace corrected algorithms. Each table entry has (wins, ties, losses) of alg1 over alg2.

alg1	alg2	MC1	MC2	MC3	MC4	MC5
ES-L	VOI-L	1, 19, 0	0, 20, 0	1, 18, 1	3, 17, 0	1, 19, 0
ES-L	MC-N-L	1, 19, 0	1, 18, 1	0, 18, 2	0, 16, 4	0, 16, 4
ES-L	Nor-L	6, 14, 0	0, 19, 1	1, 17, 2	1, 16, 3	2, 15, 3
ES-L	AO*-L	1, 19, 0	0, 20, 0	0, 19, 1	0, 16, 4	2, 14, 4
ES-L	SP-L	2, 18, 0	1, 19, 0	1, 18, 1	1, 15, 4	0, 14, 6
ES-L	PPP-L	1, 19, 0	0, 20, 0	0, 19, 1	0, 17, 3	0, 17, 3

TABLE 5.36: Breast-cancer, BDeltaCost of SP-L paired with each of the Laplace corrected algorithms. Each table entry has (wins, ties, losses) of alg1 over alg2.

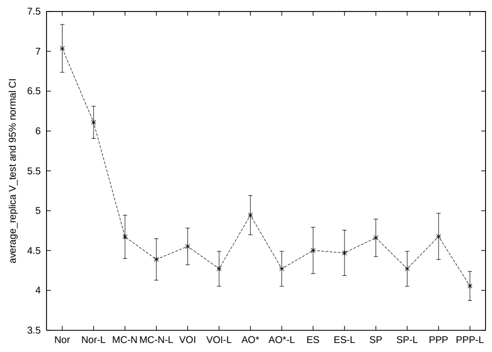
alg1	alg2	MC1	MC2	MC3	MC4	MC5
SP-L	VOI-L	0, 19, 1	0, 19, 1	0, 20, 0	6, 14, 0	5, 15, 0
SP-L	MC-N-L	0, 20, 0	0, 20, 0	0, 18, 2	1, 17, 2	1, 18, 1
SP-L	Nor-L	1, 19, 0	0, 19, 1	1, 19, 0	0, 19, 1	3, 15, 2
SP-L	AO*-L	0, 19, 1	0, 19, 1	0, 19, 1	1, 18, 1	6, 14, 0
SP-L	ES-L	0, 18, 2	0, 19, 1	1, 18, 1	4, 15, 1	6, 14, 0
SP-L	PPP-L	0, 19, 1	0, 19, 1	0, 19, 1	1, 18, 1	3, 16, 1

TABLE 5.37: Breast-cancer, BDeltaCost of PPP-L paired with each of the Laplace corrected algorithms. Each table entry has (wins, ties, losses) of alg1 over alg2.

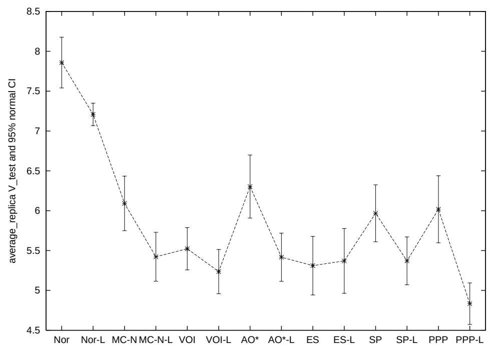
alg1	alg2	MC1	MC2	MC3	MC4	MC5
PPP-L	VOI-L	0, 20, 0	0, 20, 0	0, 20, 0	3, 17, 0	3, 17, 0
PPP-L	MC-N-L	1, 19, 0	1, 19, 0	0, 19, 1	0, 18, 2	2, 16, 2
PPP-L	Nor-L	7, 13, 0	0, 20, 0	1, 19, 0	1, 17, 2	4, 13, 3
PPP-L	AO*-L	0, 20, 0	0, 20, 0	0, 20, 0	0, 19, 1	5, 13, 2
PPP-L	ES-L	0, 19, 1	0, 20, 0	1, 19, 0	3, 17, 0	3, 17, 0
PPP-L	SP-L	1, 19, 0	1, 19, 0	1, 19, 0	1, 18, 1	1, 16, 3

TABLE 5.38: Spect, BDeltaCost of VOI-L paired with each of the Laplace corrected algorithms. Each table entry has (wins, ties, losses) of alg1 over alg2.

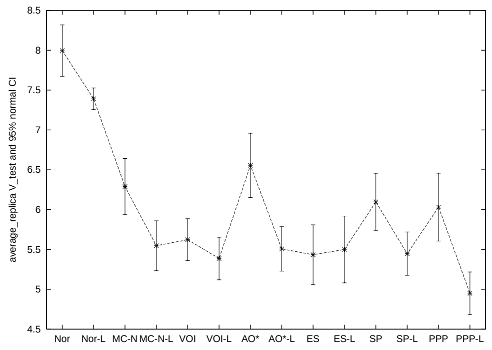
alg1	alg2	MC1	MC2	MC3	MC4	MC5
VOI-L	MC-N-L	2, 18, 0	2, 18, 0	2, 18, 0	3, 17, 0	5, 12, 3
VOI-L	Nor-L	9, 11, 0	5, 15, 0	5, 15, 0	5, 15, 0	10, 10, 0
VOI-L	AO*-L	0, 20, 0	1, 19, 0	1, 19, 0	3, 16, 1	4, 15, 1
VOI-L	ES-L	2, 18, 0	2, 18, 0	1, 19, 0	3, 16, 1	1, 19, 0
VOI-L	SP-L	0, 20, 0	1, 19, 0	1, 19, 0	3, 15, 2	4, 15, 1
VOI-L	PPP-L	0, 19, 1	1, 18, 1	0, 19, 1	2, 16, 2	4, 16, 0



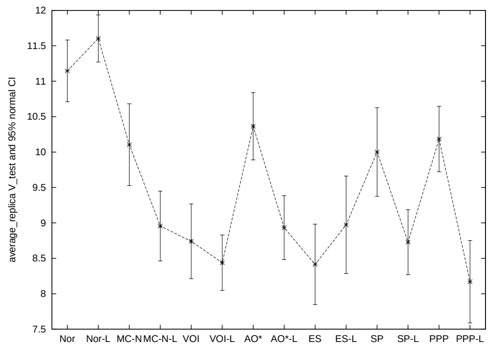
(a) MC1



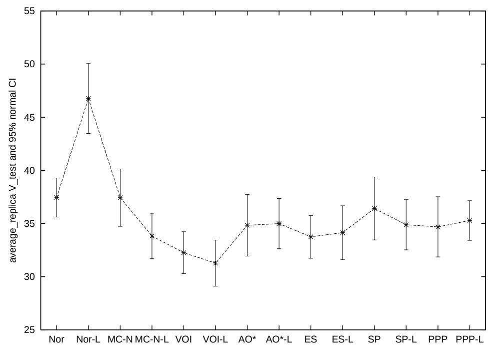
(b) MC2



(c) MC3



(d) MC4



(e) MC5

FIGURE 5.10: Spect domain. Graphs of average V_{test} over replicas, and its 95% normal confidence interval (CI). Note that MC2 $\simeq$ MC3.

TABLE 5.39: Spect, BDeltaCost of MC-N-L paired with each of the Laplace corrected algorithms. Each table entry has (wins, ties, losses) of alg1 over alg2.

alg1	alg2	MC1	MC2	MC3	MC4	MC5
MC-N-L	VOI-L	0, 18, 2	0, 18, 2	0, 18, 2	0, 17, 3	3, 12, 5
MC-N-L	Nor-L	9, 11, 0	5, 15, 0	5, 15, 0	3, 17, 0	8, 12, 0
MC-N-L	AO*-L	0, 18, 2	0, 19, 1	0, 19, 1	1, 16, 3	2, 17, 1
MC-N-L	ES-L	1, 18, 1	0, 20, 0	1, 19, 0	0, 18, 2	2, 17, 1
MC-N-L	SP-L	0, 18, 2	0, 19, 1	0, 19, 1	1, 16, 3	1, 18, 1
MC-N-L	PPP-L	0, 18, 2	0, 18, 2	0, 18, 2	1, 15, 4	0, 20, 0

TABLE 5.40: Spect, BDeltaCost of Nor-L paired with each of the Laplace corrected algorithms. Each table entry has (wins, ties, losses) of alg1 over alg2.

alg1	alg2	MC1	MC2	MC3	MC4	MC5
Nor-L	VOI-L	0, 11, 9	0, 15, 5	0, 15, 5	0, 15, 5	0, 10, 10
Nor-L	MC-N-L	0, 11, 9	0, 15, 5	0, 15, 5	0, 17, 3	0, 12, 8
Nor-L	AO*-L	0, 11, 9	0, 15, 5	0, 15, 5	0, 17, 3	0, 15, 5
Nor-L	ES-L	0, 13, 7	0, 13, 7	0, 13, 7	0, 17, 3	0, 13, 7
Nor-L	SP-L	0, 11, 9	0, 15, 5	0, 15, 5	0, 17, 3	0, 14, 6
Nor-L	PPP-L	0, 11, 9	0, 11, 9	0, 11, 9	0, 16, 4	0, 16, 4

TABLE 5.41: Spect, BDeltaCost of AO*-L paired with each of the Laplace corrected algorithms. Each table entry has (wins, ties, losses) of alg1 over alg2.

alg1	alg2	MC1	MC2	MC3	MC4	MC5
AO*-L	VOI-L	0, 20, 0	0, 19, 1	0, 19, 1	1, 16, 3	1, 15, 4
AO*-L	MC-N-L	2, 18, 0	1, 19, 0	1, 19, 0	3, 16, 1	1, 17, 2
AO*-L	Nor-L	9, 11, 0	5, 15, 0	5, 15, 0	3, 17, 0	5, 15, 0
AO*-L	ES-L	2, 18, 0	1, 19, 0	1, 19, 0	1, 17, 2	1, 18, 1
AO*-L	SP-L	0, 20, 0	0, 20, 0	0, 19, 1	0, 18, 2	0, 19, 1
AO*-L	PPP-L	0, 19, 1	0, 19, 1	0, 19, 1	1, 17, 2	3, 15, 2

TABLE 5.42: Spect, BDeltaCost of ES-L paired with each of the Laplace corrected algorithms. Each table entry has (wins, ties, losses) of alg1 over alg2.

alg1	alg2	MC1	MC2	MC3	MC4	MC5
ES-L	VOI-L	0, 18, 2	0, 18, 2	0, 19, 1	1, 16, 3	0, 19, 1
ES-L	MC-N-L	1, 18, 1	0, 20, 0	0, 19, 1	2, 18, 0	1, 17, 2
ES-L	Nor-L	7, 13, 0	7, 13, 0	7, 13, 0	3, 17, 0	7, 13, 0
ES-L	AO*-L	0, 18, 2	0, 19, 1	0, 19, 1	2, 17, 1	1, 18, 1
ES-L	SP-L	0, 18, 2	0, 19, 1	0, 19, 1	2, 17, 1	1, 18, 1
ES-L	PPP-L	0, 17, 3	0, 18, 2	0, 18, 2	2, 17, 1	3, 16, 1

TABLE 5.43: Spect, BDeltaCost of SP-L paired with each of the Laplace corrected algorithms. Each table entry has (wins, ties, losses) of alg1 over alg2.

alg1	alg2	MC1	MC2	MC3	MC4	MC5
SP-L	VOI-L	0, 20, 0	0, 19, 1	0, 19, 1	2, 15, 3	1, 15, 4
SP-L	MC-N-L	2, 18, 0	1, 19, 0	1, 19, 0	3, 16, 1	1, 18, 1
SP-L	Nor-L	9, 11, 0	5, 15, 0	5, 15, 0	3, 17, 0	6, 14, 0
SP-L	AO*-L	0, 20, 0	0, 20, 0	1, 19, 0	2, 18, 0	1, 19, 0
SP-L	ES-L	2, 18, 0	1, 19, 0	1, 19, 0	1, 17, 2	1, 18, 1
SP-L	PPP-L	0, 19, 1	0, 19, 1	0, 19, 1	1, 18, 1	2, 16, 2

TABLE 5.44: Spect, BDeltaCost of PPP-L paired with each of the Laplace corrected algorithms. Each table entry has (wins, ties, losses) of alg1 over alg2.

alg1	alg2	MC1	MC2	MC3	MC4	MC5
PPP-L	VOI-L	1, 19, 0	1, 18, 1	1, 19, 0	2, 16, 2	0, 16, 4
PPP-L	MC-N-L	2, 18, 0	2, 18, 0	2, 18, 0	4, 15, 1	0, 20, 0
PPP-L	Nor-L	9, 11, 0	9, 11, 0	9, 11, 0	4, 16, 0	4, 16, 0
PPP-L	AO*-L	1, 19, 0	1, 19, 0	1, 19, 0	2, 17, 1	2, 15, 3
PPP-L	ES-L	3, 17, 0	2, 18, 0	2, 18, 0	1, 17, 2	1, 16, 3
PPP-L	SP-L	1, 19, 0	1, 19, 0	1, 19, 0	1, 18, 1	2, 16, 2

5.3.6 Results on the spect Domain

Figure 5.10 and Tables 5.38, 5.39, 5.40, 5.41, 5.42, 5.43 and 5.44 present the test set results for the spect domain. From these results, we note the following:

- The best algorithm is VOI-L (see Table 5.38).
- The worst algorithms are Nor and Nor-L. Pruning (with Laplace correction) hurt Nor on MC5 (that is, Nor-L was significantly worse).
- All the Laplace-corrected systematic algorithms have similar performance and are all very good (see Tables 5.41, 5.42, 5.43 and 5.44). PPP-L is slightly better, and ES-L is slightly worse. ES also performed well.
- AO* performs badly on spect. Of its regularizers, the Laplace-corrected ones and ES helped the most.

5.3.7 Summary of Algorithms' Performance

After analyzing the average V_{test} and normal confidence intervals across replicas and the cumulative BDeltaCost results, we formulated the following hypotheses in answer to the questions at the beginning of this Results section:

1. Question: Which algorithm is the best on each domain? Which algorithm is the worst?

   Answer: Of the 14 algorithms we compared, there is no single overall best algorithm. Nevertheless, systematic search algorithms are best or close to the best algorithms. In particular, SP-L is robust across all domains. Table 5.45 summarizes our observations for the five domains.

   Nor is consistently the worst algorithm, and sometimes AO* performs badly. Nor is the "black sheep" in the greedy search family, and AO* is the "black sheep" in the systematic search family.

TABLE 5.45: Best and worst algorithms on each domain, according to the normal confidence intervals and BDeltaCost results. All 14 algorithms are compared.

domain	best algorithms	worst algorithms
bupa	ES-L	Nor
pima	VOI-L	Nor, AO*
heart	SP-L	AO*, Nor, VOI
b-can	MC-N-L, SP-L	Nor
spect	VOI-L	Nor

2. Question: How do greedy algorithms compare with systematic search ones?

   Answer: Depending on the problem, the greedy algorithms can be very competitive, sometimes even the best (e.g., VOI-L on pima and spect; MC-N-L on breast-cancer), but they can also be very bad (e.g., VOI-L, for large misclassification costs, on heart and breast-cancer, and MC-N-L on bupa).

   (a) Question: Is there a single best algorithm among the greedy ones?

   Answer: Most of the time, VOI-L is the best greedy algorithm. The next best choice is MC-N-L. Nor-L is inferior to both.

   (b) Question: Is there a single best algorithm among the systematic ones?

   Answer: There is no single best systematic algorithm. SP-L is the most robust across domains, but it is inferior to ES-L on bupa, and for smaller misclassification costs it is slightly worse than PPP-L on breast-cancer and spect. Most of the time, AO*-L, ES-L and PPP-L have similar performance, and they are slightly worse than SP-L (according to BDeltaCost). Of the systematic search algorithms, SP-L was never the worst.

3. Question: How does Laplace correction influence V_{test}?

   Answer: Laplace improves V_{test} of all algorithms, both greedy and systematic.

4. Question: How does AO* perform, and which regularizers help it the most?

   Answer: AO* by itself performs poorly, especially on pima, heart and spect. AO* is still significantly better than Nor (the worst algorithm, overall) on all domains and misclassification costs, except for pima and heart with medium misclassification costs.

   The regularizers significantly improve AO* performance. SP-L improves it the most. In general, combining Laplace with statistical pruning or early stopping helps AO* the most. Tables B.6 and B.7 in Appendix B list BDeltaCost results of AO* with each algorithm.

5. Question: Are there trends in performance as misclassification costs increase relative to test costs?

   Answer: The misclassification costs do influence the best and the worst algorithm on a domain, as seen on pima, heart and breast-cancer.

   Nor is particularly bad for small misclassification costs. This is probably due to the fact that Nor grows the same policy for all misclassification costs and this policy tests too many attributes. Nor-L, which grows the same policy as Nor, but then pessimistically prunes the policy using a Laplace correction, also grows the same policy irrespective of misclassification costs, but its pruned policy was better than Nor (see Table 5.8). On pima, heart and breast-cancer, for small misclassification costs, Nor-L is significantly better than Nor. Both algorithms performed well on large misclassification costs on heart and breast-cancer, but were bad on bupa, pima and spect.

In the Appendix B we organize the BDeltaCost results by comparing each Laplace-corrected algorithm with all the others, for each domain and misclassification cost level. For details on how each algorithm fares, see Tables B.8, B.9, B.10, B.11, B.12, B.13 and B.14 in the Appendix B. For example, Table B.13 shows that SP-L has many wins over AO*-L and at most one loss.

5.4 Discussion

This section answers the main question posed at the beginning of the chapter by recommending a robust CSL algorithm. It also discusses the effect of different parameters on the measurements introduced in Section 5.1.5.1.

5.4.1 An Overall Score for Algorithms (Chess Metric)

To get an overall idea of where each algorithm wins or fails and to see how robust each algorithm is across domains and misclassification costs, we have developed a measure we call the chess metric. This metric summarizes the BDeltaCost results in a single score for each algorithm.

For a given pair of algorithms, alg1 and alg2, and a domain D and misclassification cost MC, let $(wins, ties, losses)$ be the cumulative BDeltaCost results of alg1 over alg2, across all replicas. The score of alg1 on (D, MC) is computed by counting each win as one point, each tie as half a point, and each loss as zero points. Summing these points over all the other algorithms gives the chess score:

$$Score(alg1, D, MC) \overset{\text{def}}{=} \sum_{\text{alg2}\neq\text{alg1}} wins + 0.5 \times ties.$$

The score of alg1 on each domain D is obtained by summing across all MC levels:

$$Score(alg1, D) \overset{\text{def}}{=} \sum_{MC} Score(\text{alg1}, D, MC).$$

5.4.2 The Most Robust Algorithms

Table 5.46 displays scores for each algorithm and domain, for each MC level and also accumulated over all MCs. The first observation is that almost all algorithms are

TABLE 5.46: Chess score = $wins + 0.5 \times ties$. The score of each algorithm accumulates the wins and ties against all the other algorithms, for each domain, across all replicas. A win is counted as one point, a tie is counted as half a point. Scores for each separate MC in part are presented, as well as scores accumulated over all MCs.

alg1	bupa	pima	heart	b-can	spect
VOI-L	59	62.5	64.5	64	66
MC-N-L	53.5	61.5	64.5	58	60.5
Nor-L	39	48.5	43.5	46	34
AO*-L	59	62.5	64.5	64	66
ES-L	73	63	57.5	66	59
SP-L	70	62.5	64.5	58	66
PPP-L	66.5	59.5	61	64	68.5

(a) MC1

alg1	bupa	pima	heart	b-can	spect
VOI-L	59	55.5	55	61	65.5
MC-N-L	53.5	63.5	61.5	58.5	59.5
Nor-L	39	63.5	60.5	61	42
AO*-L	59	58.5	60.5	61	62.5
ES-L	73	58	58.5	60	60.5
SP-L	70	64.5	67.5	57.5	62.5
PPP-L	66.5	56.5	56.5	61	67.5

(b) MC2

alg1	bupa	pima	heart	b-can	spect
VOI-L	60	61.5	55.5	56.5	64.5
MC-N-L	48.5	63	61	65.5	60
Nor-L	47	61.5	62.5	58.5	42
AO*-L	59.5	54	58.5	62.5	62
ES-L	69	59.5	57.5	57.5	60.5
SP-L	67.5	65	67	58.5	63
PPP-L	68.5	55.5	58	61	68

(c) MC3

alg1	bupa	pima	heart	b-can	spect
VOI-L	61	69.5	56	46	66.5
MC-N-L	49.5	61.5	61.5	67.5	55.5
Nor-L	47.5	65.5	61.5	64	49.5
AO*-L	60.5	56.5	59.5	64.5	59.5
ES-L	70.5	60.5	59	53.5	63
SP-L	61.5	58.5	63	63.5	62.5
PPP-L	69.5	48	59.5	61	63.5

(d) MC4

alg1	bupa	pima	heart	b-can	spect
VOI-L	63.5	75	56	47.5	71.5
MC-N-L	47	57.5	61.5	68.5	64
Nor-L	54.5	50	61.5	62.5	40
AO*-L	59	54.5	59.5	55	60.5
ES-L	67.5	58	59	52.5	63.5
SP-L	62	61	63	70	62
PPP-L	66.5	64	59.5	64	58.5

(e) MC5

alg1	bupa	pima	heart	b-can	spect
VOI-L	302.5	324	287	275	334
MC-N-L	252	307	310	318	299.5
Nor-L	227	289	289.5	292	207.5
AO*-L	297	286	302.5	307	310.5
ES-L	353	299	291.5	289.5	306.5
SP-L	331	311.5	325	307.5	316
PPP-L	337.5	283.5	294.5	311	326

(f) Summed over all MCs.

TABLE 5.47: BDeltaCost wins, ties and losses of each algorithm on each domain, summed over all other algorithms, MCs and replicas.

alg1	bupa	pima	heart	b-can	spect
VOI-L	49, 507, 44	62, 524, 14	10, 554, 36	12, 526, 62	82, 504, 14
MC-N-L	6, 492, 102	38, 538, 24	30, 560, 10	51, 534, 15	43, 513, 44
Nor-L	10, 434, 156	34, 510, 56	19, 541, 40	39, 506, 55	0, 415, 185
AO*-L	42, 510, 48	25, 522, 53	35, 535, 30	43, 528, 29	47, 527, 26
ES-L	106, 494, 0	37, 524, 39	20, 543, 37	27, 525, 48	47, 519, 34
SP-L	74, 514, 12	42, 539, 19	60, 530, 10	40, 535, 25	52, 528, 20
PPP-L	77, 521, 2	16, 535, 49	23, 543, 34	40, 542, 18	72, 508, 20

performing well, with the exception of Nor-L. We can define a "relative zero" as the score that each algorithm would get if it was tied (according to BDeltaCost) with all the other algorithms on all replicas. If an algorithm's score is greater than the "relative zero", then the algorithm has more wins than losses. For each MC, the value of this "relative zero" is 60 (there are 6 other algorithms, and 20 replicas, for a total of 120 ties, which gives a score of 60 points). Summed over all 5 MCs, the value of "relative zero" is 300.

In Table 5.46 (f), the score of SP-L is the only one above 300 in all five domains. All other algorithms fall under 300 for at least one domain. In the tables for each MC, SP-L is under the relative zero value of 60 only on breast-cancer, MC ≤ 3, and pima, MC4. Table 5.46 (f) shows that the score of SP-L is greater than the score of AO*-L and Nor-L on every domain. It also dominates ES-L (except on bupa), MC-N-L (except on breast-cancer), and VOI-L (except on pima and spect; in fact, Table 5.46, (a), (b) and (c) show SP-L to be the best on pima, MC ≤ 3); SP-L lags behind PPP-L on bupa, breast-cancer and spect, but not by much. On pima, AO*-L, PPP-L, and

Nor-L perform badly. Nor-L is the only algorithm whose score is always less than the relative zero.

This suggests that there is no best overall algorithm but that SP-L performs well on every domain and is never the worst algorithm. This proves that systematic search, with proper regularizers, learns good policies on realistic CSL problems. VOI-L may do even better in some domains and misclassification costs, but it also does much worse in other domains and misclassification costs; in fact, VOI-L is best on pima and spect, and worst on heart and breast-cancer.

According to the chess metric, SP-L is best on heart, second best on pima (after VOI-L) and third best on bupa, breast-cancer and spect (on these domains, PPP-L is the second best algorithm). These findings are consistent with the graphs of average V_{test} across replicas and the 95% confidence intervals in the Results section (Figures 5.6, 5.7, 5.8, 5.9 and 5.10). In the Appendix B we discuss how SP-L compares with every other Laplace-corrected algorithm according to BDeltaCost (see discussion of Table B.13).

Another way of analyzing the data is to look at cumulative BDeltaCost results in Table 5.47. This displays the wins, ties and losses of each Laplace-corrected algorithm on each domain, summed over all the other variables (all the other algorithms, misclassification costs and replicas). Table 5.47 is the detailed view of each algorithm performance. Indeed, each entry in Table 5.46 (f) can be derived from the corresponding entry in Table 5.47 using the chess score (summing all the wins and half of the ties). For example, on heart SP-L has 60 wins and 530 ties overall, which produce a chess score of $60 + 0.5 \times 530 = 325$.

If each algorithm was always tied with all the others, each cell in Table 5.47 will record 600 ties (6 other algorithms, 5 MC, 20 replicas). Because BDeltaCost has a 5% error in missing a tie (that is, it wrongly declares a tie as either a win or a loss), that makes a cumulative error of $5\% \cdot 600 = 30$ in each table entry. Therefore wins over 30 will be considered significant, and also losses over 30. Wins (or losses) that are less than 30 could have been ties. All SP-L losses, being less than 30, could be ties, but

SP-L has more than 30 wins on each domain; in fact, SP-L is the only algorithm with more than 30 wins and less than 30 losses on each domain. This confirms again that that SP-L is the most robust algorithm.

One cannot help to notice though that Table 5.47 is dominated by ties. Nor-L is still the worst, but recall that it does not use misclassification costs at all during learning. The systematic search algorithms appear quite similar according to these cumulative BDeltaCost results. Inside the systematic search family, maybe it is not surprising that algorithms performance appears to be similar, since all of them share parts of the AND/OR search graph, despite the differences in regularizers. It is nevertheless surprising that the best greedy algorithms are competitive with the systematic search ones; one possible explanation is that the number of test examples is very small so BDeltaCost does not have enough data points to detect differences. The sizes of the test sets are as follows: 115 on bupa, 256 on pima, 100 on heart, 277 on breast-cancer and 89 on spect.

5.4.3 Comparing The Most Robust Algorithms Against the Best Algorithm on Each Domain

Table 5.48 lists the best and worst algorithms on each domain, according to the chess metric. Table 5.49 was built independently from the chess metric, by analyzing the normal confidence intervals and BDeltaCost tables in the Results section. Both tables compare only the Laplace-corrected algorithms, and they mostly agree on the best and worst algorithms.

We recommend SP-L as the safest choice of an algorithm. But if there are not enough resources (memory or CPU time) to run the systematic search algorithms, VOI-L becomes the second best choice. At the end of Appendix B, we included several detailed paired-graphs, at replica level, comparing the best algorithm on each domain with our top choices, SP-L and VOI-L.

We now summarize how SP-L and VOI-L compare with the best algorithm on each domain (see Tables 5.48 and 5.49). In general, BDeltaCost results of the pair (best

TABLE 5.48: Best and worst Laplace-corrected algorithms on each domain, according to the chess metric.

domain	best algorithm	worst algorithms
bupa	ES-L	Nor-L, MC-N-L
pima	VOI-L	PPP-L, AO*-L, Nor-L
heart	SP-L	VOI-L, Nor-L, ES-L
b-can	MC-N-L	VOI-L, ES-L, Nor-L
spect	VOI-L	Nor-L

TABLE 5.49: Best and worst Laplace-corrected algorithms on each domain, according to the normal confidence intervals and BDeltaCost results (Section 5.3).

domain	best algorithms	worst algorithms
bupa	ES-L	Nor-L, MC-N-L
pima	VOI-L	none (all others are similar, and good, except on MC5)
heart	SP-L	VOI-L, AO*-L
b-can	MC-N-L, SP-L	VOI-L, ES-L on larger MC
spect	VOI-L	Nor-L

algorithm on a domain, SP-L) agree with their paired-graphs of V_{test} for each replica. The same holds for the pair (best algorithm on a domain, VOI-L). These BDeltaCost and paired-graphs results are also consistent with the graphs of average V_{test} across replicas and the 95% confidence intervals (Figures 5.6, 5.7, 5.8, 5.9 and 5.10).

- On bupa, SP-L and VOI-L are quite bad, but so were all the other algorithms; ES-L is the definite winner here. In Table 5.47, ES-L has zero losses over all MC, replicas. This is consistent with Figure 5.6.

- On pima, SP-L is good on small misclassification costs, and it is worse on larger misclassification costs than VOI-L. VOI-L is best. This is consistent with Figure 5.7.

- On heart, SP-L is the best algorithm; VOI-L is consistently worse according to the V_{test} of the paired-graphs, but BDeltaCost finds them mostly tied. This is consistent with Figure 5.8.

- On breast-cancer, SP-L is worse on small misclassification costs than MC-N-L according to the V_{test} of the paired-graphs, but BDeltaCost finds them tied. VOI-L is better on small misclassification costs than MC-N-L, but BDeltaCost finds them tied except for one win of VOI-L. Both SP-L and VOI-L (and especially VOI-L) are worse than MC-N-L for larger misclassification costs. This is consistent with Figure 5.9.

- On spect, SP-L is worse on large misclassification costs than VOI-L (best on spect). This is consistent with Figure 5.10.

5.4.4 Summary of Discussion

This completes the main part of Chapter 5. Our experimental results showed that there is no overall best algorithm, but SP-L is robust across all domains, and is never the worst. Hence, we recommend SP-L as the method to apply in cost-sensitive learning problems. VOI-L could also be employed, though our experiments show that it has more variance, being the best algorithm on two domains and the worst on two domains. In terms of memory and CPU time used, VOI-L is very efficient, as shown next.

5.4.5 Insights Into the Algorithms' Performance

We discuss the effects of misclassification costs, algorithm family (greedy versus systematic) and Laplace correction over our measurements: V_{test}, Memory and CPU time, expected number of attributes and error rates. These measurements were introduced in Section 5.1.5.1. For each algorithm, the measurements are averaged over all replicas, on the test set.

5.4.5.1 V_{test}

Effect of misclassification costs

As misclassification costs increase while the test costs are kept constant, the systematic algorithms, which are based on AO*, may become worse, because tests appear cheaper compared with misclassification costs. With increasing misclassification costs, the AO* heuristic prunes less, and the systematic algorithms' search space increases in size. This in turn leads to increases in memory, CPU time, and potential for overfitting. Consequently, for systematic algorithms, larger misclassification costs compared to test costs can make the CSL problem harder (the algorithms performance is bad, they learn policies with larger V_{test}).

On the other hand, as misclassification costs increase and tests appear cheaper, policies may test more attributes. Hence, for the greedy algorithms the CSL problem can appear easier. We saw that on the largest misclassification costs, MC5, VOI-L and MC-N-L are the best algorithms or among the best. But on bupa, pima and spect, Nor, Nor-L and MC-N performed badly on MC5. So not all greedy algorithms are competitive on problems where test costs are cheap compared with misclassification costs.

Interestingly, VOI-L also performs well on smaller misclassification costs, and this can be because its built-in pruning mechanism prevents overfitting by pruning attributes whose value of information is negative.

Effect of algorithm family

In most cases, the V_{test} of the systematic algorithms was better than of the greedy algorithms. On three out of five domains, the best algorithm employed systematic search. But VOI-L, the most robust greedy algorithm, was the best algorithm in the other two domains. Another greedy algorithm, Nor, was consistently the worst.

Effect of Laplace regularization

Laplace correction improves V_{test}. For problems with large misclassification costs, like pima and heart, Laplace brings huge improvements in V_{test}.

5.4.5.2 Memory and CPU Time

Effect of misclassification costs

As misclassification costs increase, all the algorithms that use misclassification costs during learning (i.e., all but Nor and Nor-L) will use more Memory and CPU time, because tests become relatively cheaper and therefore the policies will test more attributes.

In the greedy algorithm family, Nor and Nor-L construct the same policy for all MC, since they do not use misclassification costs during learning. Therefore their memory is constant as MC increases. The same is true for the CPU time (with small variations due to the load on the machines at run time). The Memory used by MC-N and MC-N-L is also constant as MC increases, because their policies' storage does not depend on misclassification costs. The misclassification costs are only used in computing the best classification actions and in the post-pruning phase. Therefore they do not influence the choice of the attributes in the internal nodes of the policies, which are the same for all MCs.

Effect of algorithm family

Systematic algorithms use more memory and CPU time than the greedy ones, as expected (see Figures 5.11 and 5.12). Of the greedy algorithms, VOI and VOI-L use the least memory (Figure 5.11).

Effect of Laplace regularization

In general, Laplace correction increases memory used by all algorithms, but there are exceptions. For example VOI-L uses less memory than VOI on heart and breast-cancer. Laplace regularization also increases the amount of CPU time required, although there are some exceptions for VOI and AO* on spect and pima.

5.4.5.3 Expected Number of Attributes Tested (eatp)

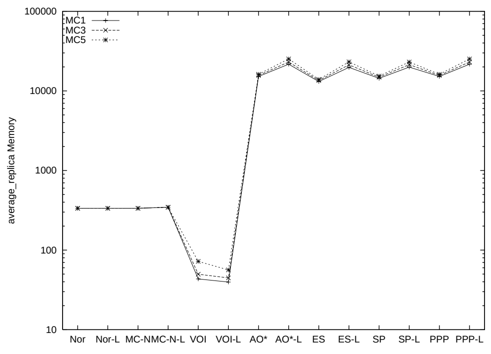
Effect of misclassification costs

As the misclassification costs increase, the policies usually have greater expected depth measured on the test set. That is, the expected number of attributes tested (eatp) increases with MC (Figure 5.13). In theory, eatp can still decrease as MC increases, if there is a very informative, but expensive, test. If such a test exists, it will only be tested for large misclassification costs, but then will be the only attribute tested, so eatp is equal to 1. We have not observed such a case, where eatp decreases as misclassification costs increase. On the domains we have analyzed, most of the policies test, on average, fewer than half the total number of attributes. Many policies are quite shallow.

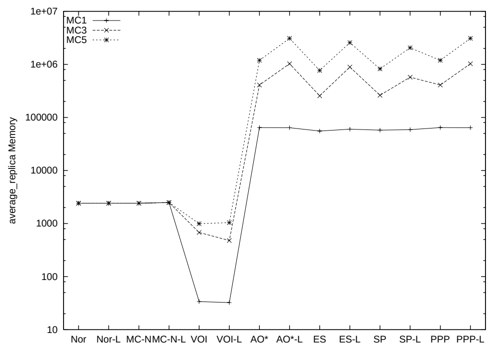
Effect of algorithm family

There is no clear pattern for how greedy versus systematic search influences eatp.

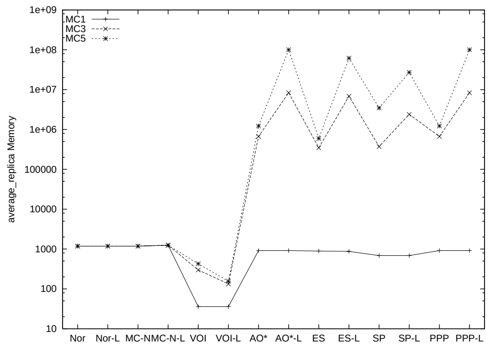
In the greedy search family, Nor and Nor-L policies do not depend on misclassification costs, so their eatp is constant. Nor's poor performance (its V_{test} is consistently the worst) is partly explained by the fact that its policy is inflexible to changes in misclassification costs. VOI-L's good performance on some domains may be explained by its small eatp. But VOI-L is the worst algorithm on heart. A rough comparison of its eatp with the eatp of the best algorithm there (SP-L) suggests that VOI-L stops testing attributes too early.



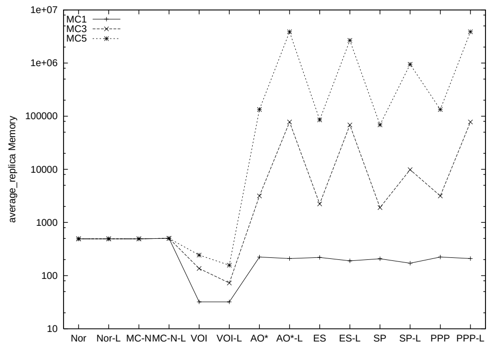
(a) Bupa



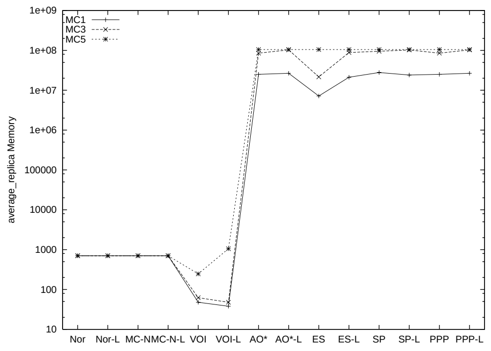
(b) Pima



(c) Heart

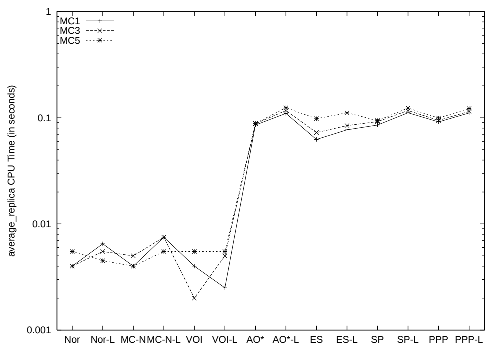


(d) Breast-cancer

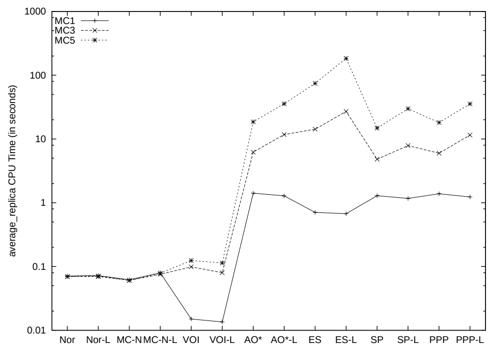


(e) Spect

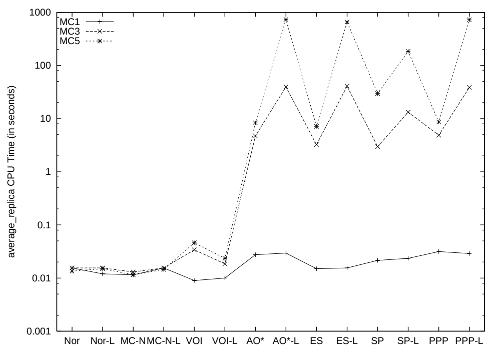
FIGURE 5.11: Graphs of Memory (measured in Bytes, and averaged over replicas), for all algorithms, as the misclassification costs increase. A memory limit of 100 MB was imposed for all algorithms. All algorithms were able to terminate before reaching this memory limit, except on spect, where the systematic algorithms reached the limit for larger misclassification costs.



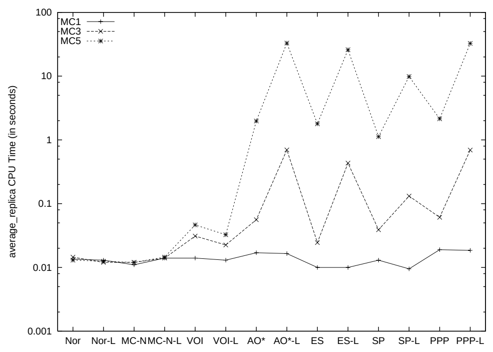
(a) Bupa



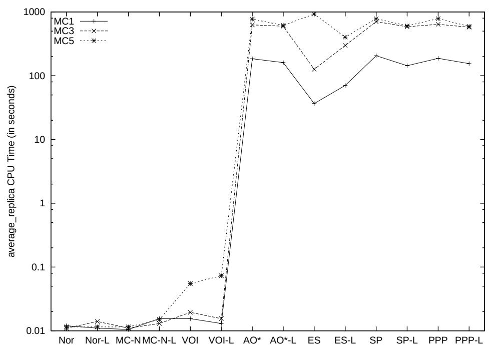
(b) Pima



(c) Heart



(d) Breast-cancer



(e) Spect

FIGURE 5.12: Graphs of CPU time (measured in seconds, and averaged over replicas), for all algorithms, as the misclassification costs increase.

In the systematic search family, AO* has the largest eatp. Since its V_{test} was bad, this large eatp is a sign of overfitting.

Pruning reduces eatp, as expected, and we see this in the eatp for Nor-L versus Nor, PPP versus AO*, and PPP-L versus AO*-L.

Effect of Laplace regularization

Laplace correction reduces eatp for all algorithms, all domains, and all MC levels. This may be one of the reasons why Laplace improves V_{test}.

5.4.5.4 Error Rate

Effect of misclassification costs

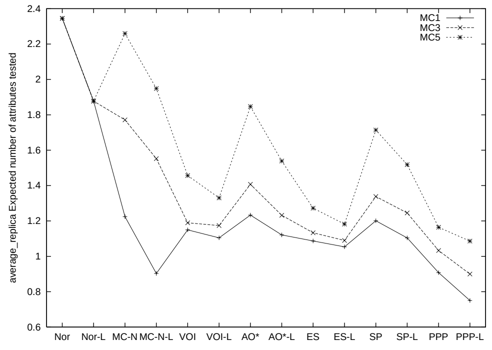
As the misclassification costs increase, we saw above that eatp increases. With more attributes tested, the learned policies could be more accurate in diagnosis, though that is not our objective. For all our algorithms that use misclassification costs during learning (i.e., all but Nor, Nor-L), we usually saw a decrease in the error rate as misclassification costs increase (Figure 5.14). There are some exceptions where error rate increased as misclassification costs increased: ES-L on bupa, larger misclassification costs; MC-N-L on heart, larger misclassification costs; and VOI-L on heart for all misclassification costs.

Effect of algorithm family

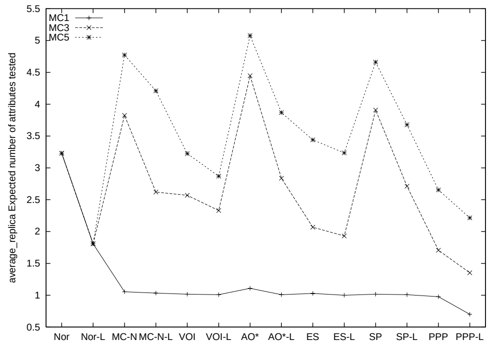
There is no clear pattern of how greedy versus systematic search influences the error rate.

Nor has a small error rate on all domains (after all, it uses InfoGain when growing its policy), but not the smallest one on any domain. The smallest (average) error rates found on each domain are the following:

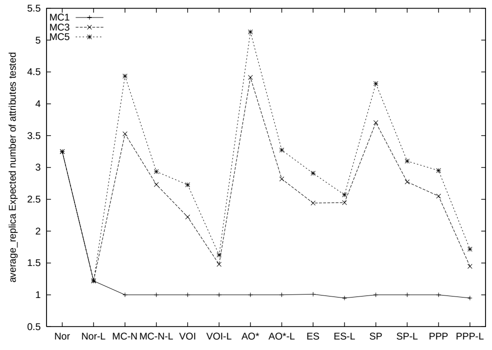
- on bupa, 39.65% achieved by SP-L for MC5,
- on pima, 24.88% achieved by VOI-L for MC5,



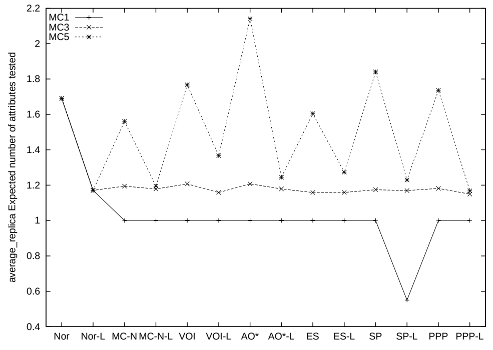
(a) Bupa, $N = 5$.



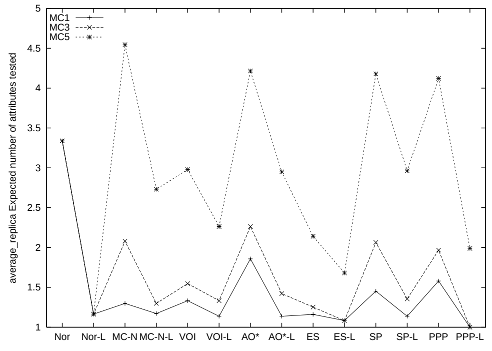
(b) Pima, $N = 8$.



(c) Heart, $N = 13$.



(d) Breast-cancer, $N = 9$.



(e) Spect, $N = 22$.

FIGURE 5.13: Graphs of expected number of attributes, abbreviated as eatp, averaged over replicas, for all algorithms, as the misclassification costs increase. eatp is measured on the test set. For each domain, we specify N, the number of attributes. Usually eatp increases as misclassification costs increase.

- on heart, 21.11% achieved by SP-L for MC3,
- on breast-cancer, 3.46%, achieved by MC-N for MC3, followed closely by MC-N-L and AO*, and
- on spect, 22.64% achieved by Nor-L for all misclassification costs.

These error rates are within the range of error rates reported for 0/1 loss algorithms on these domains. This may suggest using CSL algorithms with different misclassification costs as a way to obtain good, small error rates.

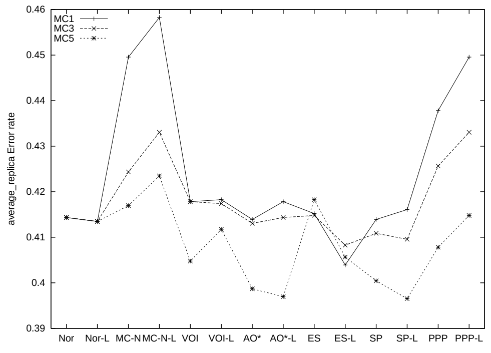
Effect of Laplace regularization

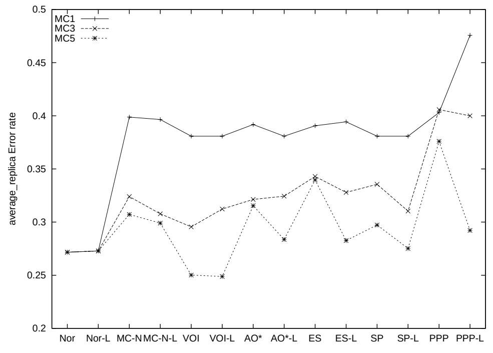
The effect of Laplace correction on the error rate is almost negligible; it can either increase it or decrease it. The largest effect is for spect.

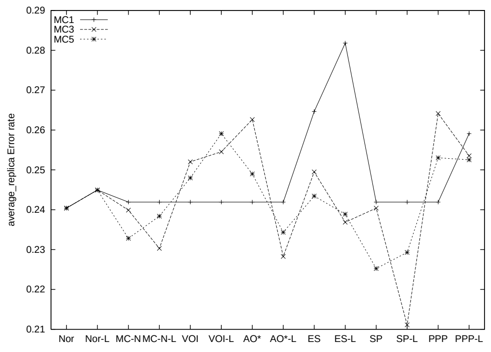
5.5 Summary

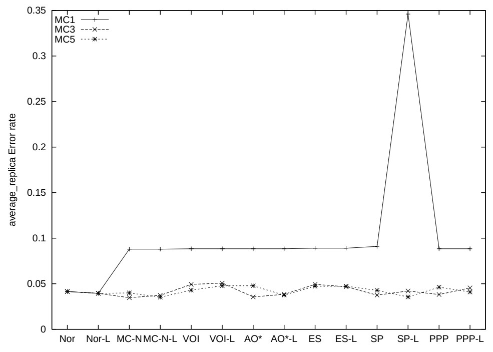
This chapter has evaluated the performance of the greedy and systematic search algorithms on several realistic problems involving medical diagnosis. It introduced a new method for defining realistic misclassification costs. The algorithms performance on test sets is evaluated at several different levels of abstraction, including: graphs of average V_{test}, and normal confidence intervals, across all replicas, for each algorithm; pairwise comparison of algorithms using the statistical test BDeltaCost; a BDeltaCost summary of each algorithm performance against all the others, using the chess metric. An interesting pattern is that Laplace correction improved the performance of all algorithms.

Different ways of interpreting the results support the main finding that, while there is no best overall algorithm across all domains and costs ranges, the systematic search algorithm AO* with Laplace corrections and Statistical Pruning was very robust across domains and never had the worst performance. The experimental results


(a) Bupa


(b) Pima


(c) Heart


(d) Breast-cancer

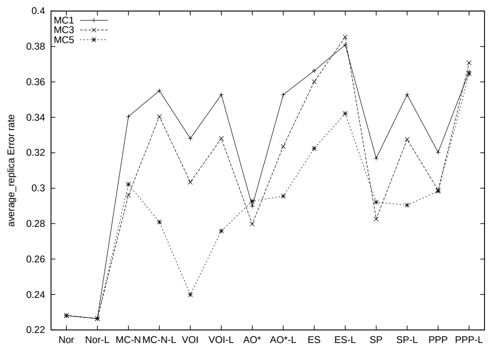

(e) Spect

FIGURE 5.14: Graphs of error rate (averaged over replicas), for all algorithms, as the misclassification costs increase. The error rate is measured on the test set. Usually error rates decrease as misclassification costs increase.

also show that AO* without regularization performs badly due to overfitting, and that regularizers are needed to improve the quality of its policy on the test data.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 Contributions of This Dissertation

The problem addressed in this dissertation is the learning of cost-sensitive diagnostic policies from a data set of labeled examples, paying attention to both test costs and misclassification costs. The tradeoff between these two types of costs is an important issue that machine learning has omitted so far, with the exception of Turney [71] and Greiner et al. [22].

1. We formulated this cost-sensitive learning problem as a Markov Decision Problem (MDP). The MDP notations enabled us to make concise and elegant proofs.

2. We showed how to apply the AO* algorithm to solve this MDP to find an optimal policy (on the training data). We also turned the AO* algorithm into an anytime algorithm by introducing the notion of a realistic policy, the best complete policy in the graph expanded so far.

3. In order to address the statistical question of how to reduce overfitting, we introduced several regularizers for the AO* algorithm: Laplace correction of probabilities, statistical pruning, pessimistic post-pruning, and early stopping.

   - Laplace correction improves the quality of the learned policy in almost all cases.
   - The statistical pruning heuristic prunes subtrees from the search space when the estimated cost of their solutions is statistically indistinguishable from the current realistic policy. When combined with Laplace correction

for probabilities, this regularizer turns systematic search based on AO* into a robust method.

- Pessimistic post-pruning is a technique for pruning the final diagnostic policy based on exaggerating the misclassification costs using the upper bound of a normal confidence interval.
- The early stopping regularizer sets aside part of the training data as a holdout set on which it evaluates the anytime realistic policy in order to find the best stopping point.

Experimental evidence in Chapter 5 shows that regularizers added to AO* reduce the risk of overfitting and produce policies that are statistically better. The statistical pruning combined with Laplace helps AO* the most. The AO* algorithm and its regularizers form the family of systematic search algorithms.

4. We also implemented greedy search algorithms.

   - A simple algorithm (InfoGainCost), similar to C4.5 but using test costs in the selection of attributes, has poor performance (when evaluated with both types of costs).
   - A better algorithm (MC+InfoGainCost) is obtained by using test costs in the selection of attributes, considering misclassification costs when choosing the classification actions (instead of classifying into the majority class), and post-pruning its policy based on both types of costs.
   - The best algorithm in the family of greedy search algorithms grows a policy using the one-step VOI (Value of Information) heuristic, which iteratively selects a test assuming it is the last one to perform before classifying. This heuristic incorporates both test costs and misclassification costs. The VOI policy also has a built-in pre-pruning step that works well in preventing overfitting by limiting the depth of the policy.

The Laplace regularizer added to MC+InfoGainCost and VOI further improves their performance.

5. We designed a method of computing interesting values for misclassification costs relative to test costs.

6. We compared algorithms from the systematic and greedy search families on realistic UCI problems, and we recommend AO* with both the Laplace and the statistical pruning regularizers (SP-L) as the cost-sensitive learning algorithm most likely to produce a good policy. However, if computational resources are an issue, the efficient and easy-to-implement one-step VOI policy with the Laplace regularizer often works well.

6.2 Future Work

This section is organized by chapter, as these questions are inspired by, or extend the work in each of the thesis chapters.

Chapter 2

- What CSL problems are polynomial-time approximable?

- A central methodological question in CSL is how to allocate the available data into training and test sets in order to get a good estimate of the quality of the learned policy [23].

- Extensions of our cost-sensitive learning framework:

  1. **Complex actions:** Our CSL framework assumes pure observational, non-interfering tests, that do not affect other tests nor do they change the examples' classes. In general, the actions can be complex. There could be treatment actions which may have side effects (on the disease, on other

measurements and treatments). Tests may also be repeated, and they may have delayed results. An important question is how to obtain training data for these complex tests.

2. **Missing attribute values:** When watching a doctor's policy, learning good policies is possible only if additional questions can be asked. This is related to the issue of off-policy learning in reinforcement learning [66].

3. **Continuous attributes:** In our studies, we discretized all continuous attribute values prior to learning. An important direction for future research would be to develop CSL methods that do not require pre-discretizing the attributes. It is straightforward to extend all of the greedy methods to discretize each attribute by dynamically choosing a threshold for each node of the decision tree. However, for the systematic search methods, this leads to a combinatorial explosion in the size of the AND/OR graph. Is there some way to integrate dynamic discretization with systematic search?

4. Other extensions are discussed in Section 2.5.

Chapter 3

- From the greedy algorithms proposed, one-step VOI is the easiest to analyze theoretically, and it also has the best performance. It would be interesting to theoretically characterize the CSL problems on which one-step VOI is close to the optimal policy π^* (either on the training data or on the true model, when known) and to compute theoretical bounds on the difference between them. This thesis shows that one-step VOI often learns good policies, compared to other algorithms. However, this comparative approach does not provide bounds on how far the VOI policies are from optimal.

- More work could be done in the design space of MC+InfoGainCost methods.

Chapter 4

- When trading-off between policy quality and limited memory, is it better to do depth-first, breadth-first or best-first (AO*) search?

- For what class of cost-sensitive problems is exhaustive search (or depth-first search) over the space of all trees feasible? Some branch-and-bound methods could be used to prune parts of the search space. A small training data set may be enough to guarantee termination of exhaustive search, though the overfitting issue remains.

- In our CSL framework, classification decisions are based on minimizing the expected misclassification costs, that is, $best_f = \arg\min_{f_k} \sum_y P(y|s) \cdot MC(f_k, y)$. We can learn a diagnostic policy by AO*, or by any of the CSL algorithms introduced in this thesis, then replace the classifications in the leaves with an ensemble of classifiers using the undiscretized attributes of the examples in the leaves, in order to compute the class probabilities $P(y|s)$ more accurately. Those ensembles should be sensitive to the misclassification costs (for example, bagged PETs [15], or cost-sensitive adaptations of Ada Boost [69]).

Chapter 5

- The CSL algorithms have obtained quite small error rates (Section 5.4.5.4), though this was a by-product. This suggests a new approach to learning and regularizing decision trees by applying AO* and the other CSL algorithms to supervised learning with 0/1 loss, setting all attribute costs to be equal to a parameter λ, to be tuned.

- Are there inadmissible heuristics that give better overall performance than the admissible heuristic that we employ? Are there ways of exploiting branch-and-bound methods to attain additional pruning?

- Another interesting direction for future work would be to first learn a probabilistic model from the data, then run the systematic search algorithms to construct diagnostic policies. Incorporating causal knowledge into the model may reduce the amount of training data needed.

6.3 Thesis Summary

There are many important problems where both test costs and misclassification costs matter. When learning cost-sensitive diagnostic policies, it is important to understand all costs involved and choose appropriate methods. Otherwise, ignoring costs that matter leads to policies that perform poorly when all costs are considered. This dissertation has shown that AO*-based systematic search learns good diagnostic policies from labeled examples when additional safeguards against overfitting are provided.

BIBLIOGRAPHY

[1] A. Barto, S. Bradtke, and S. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1):81–138, 1995.

[2] V. Bayer-Zubek and T. Dietterich. Pruning improves heuristic search for cost-sensitive learning. In *Proceedings of the Nineteenth International Conference of Machine Learning*, pages 27–35, Sydney, Australia, 2002. Morgan Kaufmann.

[3] H. Berliner. The B* tree search algorithm: a best-first proof procedure. *Artificial Intelligence*, 12:23–40, 1979.

[4] C.L. Blake and C.J. Merz. UCI repository of machine learning databases. http://www.ics.uci.edu/∼mlearn/MLRepository.html, 1998.

[5] B. Bonet and H. Geffner. Learning sorting and decision trees with POMDPs. In *Proceedings of the Fifteenth International Conference of Machine Learning*, pages 73–81. Morgan Kaufmann, San Francisco, CA, 1998.

[6] B. Bonet and H. Geffner. Faster heuristic search algorithms for planning with uncertainty and full feedback. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, San Francisco, 2003. Morgan Kaufmann.

[7] B. Bonet and H. Geffner. Labeled RTDP: Improving the convergence of real-time dynamic programming. In *Proceedings of ICAPS-03*, 2003.

[8] J. P. Bradford, C. Kunz, R. Kohavi, C. Brunk, and C. E. Brodley. Pruning decision trees with misclassification costs. In *European Conference on Machine Learning*, pages 131–136. Longer version from http://robotics.stanford.edu/users/ronnyk/ronnyk-bib.html, or as ECE TR 98-3, Purdue University, 1998.

[9] L. Breiman, J.H. Friedman, R. A. Olshen, and C.J. Stone. *Classification and Regression Trees.* Wadsworth, Monterey, California, 1984.

[10] P.P. Chakrabarti, S. Ghose, A. Acharya, and S.C. DeSarkar. Heuristic search in restricted memory. *Artificial Intelligence*, 41:197–221, 1989.

[11] P.P. Chakrabarti, S. Ghose, and S.C. DeSarkar. Admissibility of AO* when heuristics overestimate. *Artificial Intelligence*, 34:97–113, 1988.

[12] T. Das, A. Gosavi, S. Mahadevan, and N. Marchalleck. Solving semi-Markov decision problems using average reward reinforcement learning. *Management Science*, 45(4):560–574, 1999.

[13] T. G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1924, 1998.

[14] P. Domingos. MetaCost: A general method for making classifiers cost-sensitive. In *Knowledge Discovery and Data Mining*, pages 155–164, 1999.

[15] P. Domingos and F. Provost. Well-trained PETs: Improving probability estimation trees. CDER Working Paper #00-04-IS, Stern School of Business, New York University, New York, 2000.

[16] B. Efron and R. J. Tibshirani. *An introduction to the bootstrap.* New York: Chapman and Hall, 1993.

[17] T. Fawcett and F. Provost. Adaptive fraud detection. *Data Mining and Knowledge Discovery*, 1(3):1–28, 1997.

[18] T. Fountain, T.G. Dietterich, and B. Sudyka. Mining IC test data to optimize VLSI testing. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 18–25. ACM Press, 2000.

[19] J. H. Friedman and W. Stuetzle. Projection pursuit regression. *Journal of the American Statistics Association*, 76:817–823, 1981.

[20] J. Gama. A cost-sensitive iterative Bayes. In *Workshop on Cost-Sensitive Learning at ICML2000*, pages 7–13, Stanford University, California, 2000.

[21] D. Gordon and D. Perlis. Explicitly biased generalization. *Computational Intelligence*, 5(2):67–81, 1989.

[22] R. Greiner, A. J. Grove, and D. Roth. Learning cost-sensitive active classifiers. *Artificial Intelligence*, 139(2):137–174, 2002.

[23] I. Guyon, J. Makhoul, R. Schwartz, and V. Vapnik. What size test set gives good error rate estimates? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):52–64, 1998.

[24] E. Hansen. Solving POMDPs by searching in policy space. In *Proceedings of the Fourteenth International Conference on Uncertainty in Artificial Intelligence*, pages 211–219, San Francisco, 1998. Morgan Kaufmann.

[25] E. Hansen and S. Zilberstein. A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129(1–2):35–62, 2001.

[26] M. Hauskrecht. Value-function approximations for partially observable Markov decision processes. *Journal of Artificial Intelligence Research*, 12:33–94, 2000.

[27] D. Heckerman, J. Breese, and K. Rommelse. Troubleshooting under uncertainty. Technical report, MSR-TR-94-07, Microsoft Research, 1994.

[28] H. Hermans, J.D.F. Habbema, and A.T Van der Burgt. Cases of doubt in allocation problems, k populations. *Bulletin of the International Statistics Institute*, 45:523–529, 1974.

[29] L. Hyafil and R. L. Rivest. Constructing optimal binary decision trees is NP-Complete. *Information Processing Letters*, 5(1):15–17, 1976.

[30] N. Ikizler. Benefit maximizing classification using feature intervals. M.S. Thesis, Department of Computer Engineering, Bilkent University, Ankara, Turkey, 2002.

[31] P. Jimenez and C. Torras. An efficient algorithm for searching implicit AND/OR graphs with cycles. *Artificial Intelligence*, 124:1–30, 2000.

[32] U. Knoll, G. Nakhaeizadeh, and B. Tausend. Cost-sensitive pruning of decision trees. In *Proceedings of the Eighth European Conference of Machine Learning*, pages 383–386, Berlin, Germany, 1994. Springer-Verlag.

[33] D. E. Knuth and R. W. Moore. An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4):293–326, 1975.

[34] R. Korf. Real-time heuristic search. *Artificial Intelligence*, 42:189–211, 1990.

[35] M. Kukar and I. Kononenko. Cost-sensitive learning with neural networks. In *Proceedings of the Thirteenth European Conference Artificial Intelligence*, pages 445–449, Chichester, 1998. John Wiley & Sons, New York.

[36] V. Kumar and L. N. Kanal. A general branch and bound formulation for understanding and synthesizing AND/OR tree search procedures. *Artificial Intelligence*, 21:179–198, 1983.

[37] V. Kumar and L. N. Kanal. The CDP: A unifying formulation for heuristic search, dynamic programming, and branch-and-bound. In L. N. Kanal and V. Kumar, editors, *Search in Artificial Intelligence*, pages 1–27. Springer-Verlag, Berlin, 1988.

[38] V. Kumar and L. N. Kanal. A general branch-and-bound formulation for AND/OR graph and game tree search. In L. N. Kanal and V. Kumar, editors, *Search in Artificial Intelligence*, pages 91–130. Springer-Verlag, Berlin, 1988.

[39] J. B. Larsen and J. S. Dyer. Using extensive form analysis to solve partially observable Markov decision problems. Technical report, 90/91-3-1, University of Texas at Austin, 1990.

[40] Y. Lirov and O-C. Yue. Automated network troubleshooting knowledge acquisition. *Journal of Applied Intelligence*, 1:121–132, 1991.

[41] M. L. Littman, T. Nguyen, H. Hirsh, E. M. Fenson, and R. Howard. Cost-sensitive fault remediation for autonomic computing. In *Workshop on AI and Autonomic Computing: Developing a Research Agenda for Self-Managing Computer Systems*, 2003.

[42] A. Mahanti and A. Bagchi. AND/OR graph heuristic search methods. *Journal of the ACM*, 32(1):28–51, 1985.

[43] D. D. Margineantu. *Methods for Cost-sensitive Learning.* PhD thesis, Department of Computer Science, Oregon State University, Corvallis, 2001.

[44] D. D. Margineantu and T. G. Dietterich. Improved class probability estimates from decision tree models. In *Nonlinear Estimation and Classification; Lecture Notes in Statistics*, volume 171, pages 169–184, New York, 2002. Springer-Verlag.

[45] A. Martelli and U. Montanari. Additive AND/OR graphs. In *Proceedings of the Third International Joint Conference on Artificial Intelligence*, pages 1–11, 1973.

[46] A. Martelli and U. Montanari. Optimizing decision trees through heuristically guided search. *Communications of the ACM*, 21(12):1025–1039, 1978.

[47] T. M. Mitchell. *Machine Learning.* McGraw-Hill Companies, Inc., 1997.

[48] A. Moore and C. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning*, 13:103–130, 1993.

[49] D. S. Nau, V. Kumar, and L. N. Kanal. General branch and bound, and its relation to A* and AO*. *Artificial Intelligence*, 23(1):29–58, 1984.

[50] N. Nilsson. *Principles of Artificial Intelligence.* Tioga Publishing Co., Palo Alto, CA, 1980.

[51] S. W. Norton. Generating better decision trees. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 800–805, San Francisco, 1989. Morgan Kaufmann.

[52] M. Nunez. The use of background knowledge in decision tree induction. *Machine Learning*, 6(3):231–250, 1991.

[53] K. R. Pattipati and M. G. Alexandridis. Application of heuristic search and information theory to sequential fault diagnosis. *IEEE Transactions on Systems, Man and Cybernetics*, 20(4):872–887, 1990.

[54] M. Pazzani, C. Merz, P. Murphy, K. Ali, T. Hume, and C. Brunk. Reducing misclassification costs. In *Proceedings of the Eleventh International Conference of Machine Learning*, pages 217–225, New Brunswick, New Jersey, 1994. Morgan Kaufmann.

[55] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley Publishing Co., Massachusetts, 1984.

[56] E. Pednault, N. Abe, B. Zadrozny, et al. Sequential cost-sensitive decision-making with reinforcement learning. In *Proceedings of the Eighth International Conference on Knowledge Discovery and Data Mining*, pages 204–213, 2002.

[57] J. Peng and R. J. Williams. Efficient learning and planning within the Dyna framework. *Adaptive Behavior*, 1(4):437–454, 1993.

[58] F. J. Provost and T. Fawcett. Analysis and visualization of classifier performance: Comparison under imprecise class and cost distributions. In *Knowledge Discovery and Data Mining*, pages 43–48, 1997.

[59] F. J. Provost and T. Fawcett. Robust classification for imprecise environments. *Machine Learning*, 42(3):203–231, 2001.

[60] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons, New York, 1994.

[61] R. Qi. *Decision Graphs: Algorithms and Applications to Influence Diagram Evaluation and High-Level Path Planning Under Uncertainty.* PhD thesis, University of British Columbia, 1994.

[62] R. Qi and D. Poole. A new method for influence diagram evaluation. *Computational Intelligence*, 11:498–528, 1995.

[63] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, California, 1993.

[64] J. K. Satia and R. E. Lave. Markovian decision processes with probabilistic observation of states. *Management Science*, 20:1–13, 1973.

[65] G. C. Stockman. A minimax algorithm better than alpha-beta? *Artificial Intelligence*, 12:179–196, 1979.

[66] R. S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambrdige, Massachusetts, 1999.

[67] M. Tan. Cost-sensitive learning of classification knowledge and its applications in robotics. *Machine Learning*, 13(1):1–33, 1993.

[68] M. Tan and J. C. Schlimmer. Two case studies in cost-sensitive concept acquisition. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 854–860, Menlo Park, California, 1990. AAAI Press.

[69] K. M. Ting. A comparative study of cost-sensitive boosting algorithms. In *Proceedings of the Seventeenth International Conference of Machine Learning*, pages 983–990, San Francisco, CA, 2000. Morgan Kaufmann.

[70] P. Turney. Types of cost in inductive concept learning. In *Workshop on Cost-Sensitive Learning at ICML2000*, pages 15–21, Stanford University, California, 2000.

[71] P. D. Turney. Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm. *Journal of Artificial Intelligence Research*, 2:369–409, 1995.

[72] L. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[73] R. Washington. BI-POMDP: Bounded, incremental partially-observable Markov-model planning. In *Proceedings of the Fourth European Conference on Planning*, 1997.

[74] G. I. Webb. Cost-sensitive specialization. In *Proceedings of the 1996 Pacific Rim International Conference on Artificial Intelligence*, pages 23–34. Springer-Verlag, 1996.

[75] S. M. Weiss, R. S. Galen, and P. V. Tadepalli. Maximizing the predictive value of production rules. *Artificial Intelligence*, 45(1-2):47–71.

[76] B. Zadrozny. *Policy Mining: Learning decision policies from fixed sets of data.* PhD thesis, Department of Computer Science and Engineering, University of California, San Diego, 2003.

[77] B. Zadrozny and C. Elkan. Learning and making decisions when costs and probabilities are both unknown. In *Proceedings of the Seventh International Conference on Knowledge Discovery and Data Mining*, pages 204–213. ACM Press, 2001.

[78] A. Zuzek, A. Biasizzo, and F. Novak. Sequential diagnosis tool. *Microprocessors and microsystems*, 26:191–197, 2000.

APPENDICES

CHAPTER A

DETAILS ON OUR AO* IMPLEMENTATION

The following are detailed explanations related to our AO* implementation in Section 4.2.

1. The notion of a parent OR node s for OR node s' is with respect to an action a (AND node (s, a)), not to a policy (π^{opt} or π^{real}).

2. Any OR node has a single action marked as π^{opt}, π^{real}, but since the graph is a DAG, it may have more than one parent reaching it through their π^{opt}, π^{real}.

3. When we perform optimistic updates of the marked ancestors of a fringe OR node, it is useful to think of a tree (not a DAG!) rooted at the fringe OR node, whose links are reversed π^{opt} connectors (computed in previous iterations of the algorithm) in the entire graph expanded so far. So an OR node appears only once on the optimistic queue. In other words, an OR node has a single successor OR node through π^{opt}, so there can not be more than one path following π^{opt} reaching it backwards from the fringe node.

4. An OR node will appear at most once on the optimistic queue, per AO* iteration, but can appear more than once on the realistic queue (because we update the V^{real} of all ancestors of a fringe OR node, the reversed graph for the update of V^{real} is a DAG rooted at the fringe OR node).

5. From the fringe OR node, all its ancestors up to the root, along the branch of π^{opt} from the root to itself, may potentially be pushed onto the optimistic queue, along with other ancestors in the graph which reach the fringe node through their π^{opt}. It is possible that the optimistic updates will not reach the root node, if the last ancestors pushed onto the optimistic queue do not have their V^{opt} changed nor do they become solved OR nodes. In that case, these OR nodes will not push their marked parents onto the optimistic queue and the updates stop before reaching the root.

6. The root node may not appear on the optimistic queue, nor on the realistic queue. This happens when V^{opt}, respectively V^{real}, of the last ancestors on the queue did not change, so nothing else gets pushed onto the queue. For the optimistic values, it is easy to imagine such a situation when all attributes have the same costs and the generative model of the problem is symmetric. Example: a problem with three attributes, in state $s = \{x_1 = 0\}$ we have two unexpanded AND nodes (s, x_2) and (s, x_3) with equal Q values, which are cheaper than classifying: $Q_i^{opt}(s, x_2) = Q_i^{opt}(s, x_3) < C(s, best_f)$. We break ties in favor of the attribute with the smallest index, so $\pi^{opt}(s) = x_2$ and we expand AND node (s, x_2). Assume that its children OR nodes s' are not classified, that is in all of them $\pi^{opt}(s') = x_3$. Therefore the Q value of (s, x_2) increases after expansion, $Q_i^{opt}(s, x_2) < Q_{i+1}^{opt}(s, x_2)$, and because $Q_{i+1}^{opt}(s, x_3) = Q_i^{opt}(s, x_3) = Q_i^{opt}(s, x_2) < C(s, best_f)$, we have $V_i^{opt}(s) = V_{i+1}^{opt}(s)$, unchanged, and $\pi^{opt}(s) = x_3$, with (s, x_3) unexpanded. Since $V^{opt}(s)$ stays the same, and s did not become solved, nothing will be pushed onto the optimistic queue. For the realistic values, after expanding the fringe node's (s, a) and calculating $Q^{real}(s, a)$, this may be strictly greater than the old value $V^{real}(s)$. Therefore $V^{real}(s)$ will not change,

and no ancestors of the fringe node will be pushed onto the realistic queue.

7. $V^{opt}(s)$ may stay the same, but the node can become solved. Imagine this situation: in π^{opt}, there is an OR node s measuring attribute x_i, that only has two values. For one of the values, the resulting state is already classified. For the other value, the resulting state s' has only one unexpanded AND node (s', x_j), measuring the last of all attributes. Therefore $V^{opt}(s') = Q^{opt}(s', x_j) = Q^*(s', x_j)$, because x_j is the last attribute to be measured. Say we chose to expand AND node (s', x_j); then after expansion, its Q value stays the same, and since nothing changed, $V^{opt}(s')$ does not change either, but the OR node s' has become solved, and this needs to be propagated to its marked parents.

8. A solved OR node will not be pushed onto the optimistic queue, because a solved OR node's π^{opt} is a complete policy, and therefore it can not reach a fringe node with an unexpanded π^{opt}. Still, the fringe node that is first pushed onto the optimistic queue may become solved once it is popped off the queue.

9. We keep the "solved" notion as marking an OR node whose π^{opt} reaches "solved" OR nodes, instead of marking an OR node "solved" when $\pi^{opt}(s) = \pi^{real}(s)$, because we need this "solved" notion for our efficient optimistic updates. Note that if s is solved, its optimistic policy is a complete policy; after updating its realistic policy, it follows that $\pi^{opt}(s) = \pi^{real}(s)$.

10. A solved OR node may be pushed onto the realistic queue, since we push all the ancestors of the fringe node onto the queue (as long as there are changes in V^{real}).

11. We need to clarify more about the "solved" notion. An OR node will be marked as solved when its π^{opt} becomes a complete policy, therefore we will have $V^{opt}(s) = V^*(s)$ (the proof is similar to Theorem 4.2.7). We still need to update its V^{real} before having $V^{opt}(s) = V^{real}(s) = V^*(s)$ and $\pi^{opt}(s) = \pi^{real}(s)$. So it is possible that we pop an OR node from the realistic queue, and it is marked as solved, but its V^{real} was not yet updated. For example, say we expand AND node (s, a) of fringe node s, and all its children s' are classified. Assume that after expansion, this action still has the smallest Q value, therefore $V^{opt}(s) = Q^{opt}(s, a) = Q^*(s, a) = V^*(s)$. Because π^{opt} is a complete policy (in fact, it became π^*), the fringe node becomes solved. But we still need to compute $Q^{real}(s, a)$ and update its V^{real}. After expanding (s, a), $Q^{real}(s, a) = Q^{opt}(s, a) = V^*(s)$. Since $V^{real}(s) = \min_{a \in A'(s)} Q^{real}(s, a) \geq V^*(s)$, it follows that $V^{real}(s) = V^*(s)$.

12. For any OR node s popped from the optimistic queue, all the Q^{opt} values of its expanded AND nodes (s, a), $Q^{opt}(s, a)$, need to be updated, because it is possible that $V^{opt}(s')$ of some of their children s' may have changed (increased); $Q^{opt}(s, \pi_i^{opt}(s))$ may increase, and we need to have the up-to-date siblings' Q^{opt} in order to compute π_{i+1}^{opt}. Here is an example to illustrate this point: consider a problem with four attributes, where the graph expanded so far has $\pi^{opt}(s_0) = x_2, \pi^{opt}(\{x_1 = 0\}) = x_2, \pi^{opt}(\{x2 = 0\}) = x_1$, and $\pi^{opt}(\{x_1 = 0, x_2 = 0\}) = x_3$, and this last action is expanded. Assume $V^{opt}(\{x_1 = 0, x_2 = 0\})$ increases, so we will push its marked parents $\{x1 = 0\}$ and $\{x2 = 0\}$ onto the optimistic queue, and say their V^{opt} increases too; nevertheless, the root will be pushed only once onto the optimistic queue, as the marked parent of $\{x_2 = 0\}$, but it still needs to update its $Q^{opt}(s_0, x_1)$ because $V^{opt}(\{x1 = 0\})$ increased and x_1

may become $\pi_{opt}(s_0)$ if the new Q values in s_0 satisfy $Q^{opt}(s_0, x_1) < Q^{opt}(s_0, x_2)$.

13. Suggested improvement for the realistic updates: for any OR node s' we can push onto the realistic queue pairs containing a parent OR node s and the action a that relates the two OR nodes (note that action a is not necessarily $\pi_i^{real}(s)$, because a parent OR node s for OR node s' is defined with respect to an action a, not to a policy). Then we only recompute $Q_{i+1}^{real}(s, a)$ for $V_{i+1}^{real}(s)$, instead of updating all Q^{real} values of expanded AND nodes in state s. This improvement is correct because all ancestors of the fringe node get pushed onto the realistic queue (if there are changes in V^{real}).

14. During the bottom-up optimistic updates, for unexpanded AND nodes (s, a), we simply use the $Q^{opt}(s, a)$ stored in the AND nodes (these were computed at the creation of the AND nodes, based on h^{opt}). Unexpanded AND nodes are ignored during the realistic updates.

CHAPTER B

MORE INFORMATION ON THE EXPERIMENTAL STUDIES

This appendix contains additional information on Chapter 5. We refer the reader to Table 5.7 for a list of abbreviations.

B.1 Misclassification Costs Matrices for the UCI Domains

Tables B.1, B.2, B.3, B.4 and B.5 list the values for the five misclassification costs used in each domain. The method for defining the misclassification costs values was introduced in Section 5.1.2.

B.2 Comparing the Worst Algorithms in the Systematic and Greedy Search Families

In the systematic search family, AO* was the worst. In the greedy search family, Nor was the worst. We compared AO* with Nor. Based on our analysis of the average V_{test} over replicas (and the normal confidence intervals), and the BDeltaCost results, we found that:

- for smaller misclassification costs, AO* beats Norton.
- for larger misclassification costs, they become more alike.

If we Laplace-correct the probabilities during the pessimistic post-pruning of Nor, that helps a lot, sometimes Nor-L beating AO*, for example on pima; and also on

$$MC1 = \begin{pmatrix} 0 & 120.989 \\ 126 & 0 \end{pmatrix} \qquad MC2 = \begin{pmatrix} 0 & 121.949 \\ 127 & 0 \end{pmatrix}$$

$$MC3 = \begin{pmatrix} 0 & 154.597 \\ 161 & 0 \end{pmatrix} \qquad MC4 = \begin{pmatrix} 0 & 155.557 \\ 162 & 0 \end{pmatrix}$$

$$MC5 = \begin{pmatrix} 0 & 239.097 \\ 249 & 0 \end{pmatrix}$$

TABLE B.1: BUPA Liver Disorders (bupa), misclassification costs matrices. We constrain the ratio of misclassification costs of false negatives and false positives, $r = MC(fn)/MC(fp)$, to be equal to the ratio of class probabilities, $P(y=0)/P(y=1)$, computed on the entire data set. On bupa, $r = 0.96$.

$$MC1 = \begin{pmatrix} 0 & 13.0597 \\ 7 & 0 \end{pmatrix} \qquad MC2 = \begin{pmatrix} 0 & 52.2388 \\ 28 & 0 \end{pmatrix}$$

$$MC3 = \begin{pmatrix} 0 & 117.537 \\ 63 & 0 \end{pmatrix} \qquad MC4 = \begin{pmatrix} 0 & 259.328 \\ 139 & 0 \end{pmatrix}$$

$$MC5 = \begin{pmatrix} 0 & 10126.9 \\ 5428 & 0 \end{pmatrix}$$

TABLE B.2: Pima Indians Diabetes (pima), misclassification costs matrices. $r = MC(fn)/MC(fp) = 1.86$.

breast-cancer, MC5. These are cases where BDeltaCost results agree very nicely with the average V_{test} graphs. Still, for small misclassification costs, AO* beats Nor-L.

$$MC1 = \begin{pmatrix} 0 & 5.8394 \\ 5 & 0 \end{pmatrix} \qquad MC2 = \begin{pmatrix} 0 & 348.028 \\ 298 & 0 \end{pmatrix}$$

$$MC3 = \begin{pmatrix} 0 & 617.809 \\ 529 & 0 \end{pmatrix} \qquad MC4 = \begin{pmatrix} 0 & 1869.78 \\ 1601 & 0 \end{pmatrix}$$

$$MC5 = \begin{pmatrix} 0 & 1870.94 \\ 1602 & 0 \end{pmatrix}$$

TABLE B.3: Heart Disease (heart), misclassification costs matrices. r = MC(fn)/MC(fp) = 1.16.

$$MC1 = \begin{pmatrix} 0 & 3.71548 \\ 2 & 0 \end{pmatrix} \qquad MC2 = \begin{pmatrix} 0 & 5.57322 \\ 3 & 0 \end{pmatrix}$$

$$MC3 = \begin{pmatrix} 0 & 16.7197 \\ 9 & 0 \end{pmatrix} \qquad MC4 = \begin{pmatrix} 0 & 27.8661 \\ 15 & 0 \end{pmatrix}$$

$$MC5 = \begin{pmatrix} 0 & 122.611 \\ 66 & 0 \end{pmatrix}$$

TABLE B.4: Breast Cancer (breast-cancer), misclassification costs matrices. $r = MC(fn)/MC(fp) = 1.85$.

B.3 Comparing AO* with All the Other Algorithms using BDeltaCost

Tables B.6 and B.7 compare AO* with all the other algorithms (except with AO*-L, because this comparison appears in Table 5.9). The regularizers added to AO*

$$MC1 = \begin{pmatrix} 0 & 7.00472 \\ 27 & 0 \end{pmatrix} \qquad MC2 = \begin{pmatrix} 0 & 8.56132 \\ 33 & 0 \end{pmatrix}$$

$$MC3 = \begin{pmatrix} 0 & 8.82076 \\ 34 & 0 \end{pmatrix} \qquad MC4 = \begin{pmatrix} 0 & 14.7877 \\ 57 & 0 \end{pmatrix}$$

$$MC5 = \begin{pmatrix} 0 & 64.5991 \\ 249 & 0 \end{pmatrix}$$

TABLE B.5: SPECT (spect), misclassification costs matrices. r = MC(fn)/MC(fp) = 0.26.

improved its performance (Table B.7). As discussed above, AO* is mostly better than Norton. From the greedy algorithms, VOI and VOI-L have the most wins over AO*.

B.4 Results of Comparing Each Laplace-Corrected Algorithm with All the Other Laplace-corrected Algorithms, on Each Domain and Misclassification Cost Level (MC)

Tables B.8, B.9, B.10, B.11, B.12, B.13 and B.14 organize the BDeltaCost results by algorithm. Each Laplace-corrected algorithm is pairwise compared with all the other Laplace-corrected algorithms, for each domain and misclassification cost level.

Table B.13 for SP-L and every algorithm confirms that SP-L is a robust method. According to it, SP-L has a good number of wins over all methods, and it loses on at most 4 replicas on large misclassification costs, as follows: SP-L loses to VOI-L on pima and spect, MC4 and MC5 (on 3 and 4 replicas), to MC-N-L on breast-cancer on MC3 and MC4 (on 2 replicas), and to Nor-L on pima, MC4 and breast-cancer, MC5

alg1	alg2	domain	MC1	MC2	MC3	MC4	MC5
AO*	VOI	bupa	0, 17, 3	0, 17, 3	0, 16, 4	0, 16, 4	0, 15, 5
AO*	VOI	pima	0, 17, 3	0, 12, 8	0, 3, 17	0, 6, 14	0, 9, 11
AO*	VOI	heart	0, 20, 0	0, 18, 2	0, 16, 4	0, 20, 0	0, 20, 0
AO*	VOI	b-can	0, 20, 0	0, 20, 0	6, 14, 0	6, 13, 1	0, 15, 5
AO*	VOI	spect	0, 15, 5	0, 11, 9	0, 11, 9	0, 16, 4	0, 18, 2
AO*	VOI-L	bupa	0, 16, 4	0, 16, 4	0, 16, 4	0, 16, 4	0, 15, 5
AO*	VOI-L	pima	0, 17, 3	0, 15, 5	0, 3, 17	0, 5, 15	0, 5, 15
AO*	VOI-L	heart	0, 20, 0	0, 18, 2	0, 15, 5	0, 20, 0	0, 20, 0
AO*	VOI-L	b-can	0, 20, 0	0, 20, 0	2, 18, 0	2, 18, 0	0, 17, 3
AO*	VOI-L	spect	0, 14, 6	0, 10, 10	0, 9, 11	0, 15, 5	0, 16, 4
AO*	MC-N	bupa	4, 16, 0	4, 16, 0	3, 17, 0	4, 16, 0	5, 14, 1
AO*	MC-N	pima	0, 18, 2	0, 15, 5	0, 15, 5	0, 18, 2	0, 19, 1
AO*	MC-N	heart	0, 20, 0	0, 16, 4	0, 13, 7	0, 20, 0	0, 20, 0
AO*	MC-N	b-can	1, 19, 0	0, 20, 0	0, 17, 3	0, 17, 3	0, 14, 6
AO*	MC-N	spect	1, 15, 4	3, 13, 4	1, 14, 5	0, 20, 0	2, 17, 1
AO*	MC-N-L	bupa	2, 18, 0	2, 18, 0	1, 19, 0	1, 19, 0	4, 16, 0
AO*	MC-N-L	pima	0, 17, 3	0, 8, 12	0, 2, 18	0, 8, 12	0, 17, 3
AO*	MC-N-L	heart	0, 20, 0	0, 13, 7	0, 6, 14	0, 18, 2	0, 18, 2
AO*	MC-N-L	b-can	1, 19, 0	0, 20, 0	0, 15, 5	0, 15, 5	0, 8, 12
AO*	MC-N-L	spect	0, 14, 6	0, 11, 9	1, 9, 10	1, 15, 4	1, 17, 2
AO*	Nor	bupa	9, 11, 0	8, 12, 0	8, 12, 0	8, 12, 0	4, 15, 1
AO*	Nor	pima	19, 1, 0	14, 4, 2	0, 17, 3	0, 16, 4	0, 17, 3
AO*	Nor	heart	20, 0, 0	3, 15, 2	0, 14, 6	0, 20, 0	0, 20, 0
AO*	Nor	b-can	19, 1, 0	16, 4, 0	15, 4, 1	14, 5, 1	0, 14, 6
AO*	Nor	spect	16, 4, 0	11, 9, 0	10, 10, 0	1, 19, 0	2, 17, 1
AO*	Nor-L	bupa	7, 13, 0	7, 13, 0	3, 17, 0	3, 17, 0	1, 19, 0
AO*	Nor-L	pima	4, 16, 0	0, 10, 10	0, 2, 18	0, 7, 13	1, 16, 3
AO*	Nor-L	heart	6, 14, 0	0, 17, 3	0, 15, 5	0, 20, 0	0, 20, 0
AO*	Nor-L	b-can	7, 13, 0	0, 20, 0	2, 14, 4	2, 14, 4	0, 11, 9
AO*	Nor-L	spect	4, 16, 0	3, 17, 0	1, 19, 0	0, 20, 0	6, 13, 1

TABLE B.6: BDeltaCost of AO* paired with each greedy search algorithm, across all domains. Each table entry has (wins, ties, losses) of alg1 over alg2.

(on 2 replicas). SP-L loses to ES-L on bupa, MC4 and MC5 (on 3 and 2 replicas). SP-L has 2 losses to PPP-L (bupa and spect, large misclassification costs), and has only one loss to AO*-L (pima, heart and breast-cancer).

alg1	alg2	domain	MC1	MC2	MC3	MC4	MC5
AO*	ES	bupa	1, 14, 5	1, 14, 5	0, 16, 4	0, 16, 4	1, 19, 0
AO*	ES	pima	1, 17, 2	0, 12, 8	0, 9, 11	0, 13, 7	3, 16, 1
AO*	ES	heart	3, 17, 0	0, 15, 5	0, 11, 9	0, 20, 0	0, 20, 0
AO*	ES	b-can	0, 19, 1	0, 20, 0	4, 14, 2	4, 12, 4	0, 15, 5
AO*	ES	spect	0, 14, 6	0, 11, 9	0, 10, 10	0, 13, 7	2, 16, 2
AO*	ES-L	bupa	0, 13, 7	0, 13, 7	0, 15, 5	0, 16, 4	0, 19, 1
AO*	ES-L	pima	1, 17, 2	0, 12, 8	0, 6, 14	0, 11, 9	0, 15, 5
AO*	ES-L	heart	2, 18, 0	0, 15, 5	0, 8, 12	0, 19, 1	0, 19, 1
AO*	ES-L	b-can	0, 19, 1	0, 20, 0	3, 15, 2	2, 16, 2	0, 17, 3
AO*	ES-L	spect	0, 14, 6	0, 13, 7	0, 12, 8	0, 16, 4	0, 18, 2
AO*	SP	bupa	0, 15, 5	0, 15, 5	0, 18, 2	0, 18, 2	2, 13, 5
AO*	SP	pima	0, 17, 3	0, 14, 6	0, 15, 5	0, 17, 3	0, 16, 4
AO*	SP	heart	0, 20, 0	0, 13, 7	0, 11, 9	0, 16, 4	0, 16, 4
AO*	SP	b-can	1, 19, 0	0, 20, 0	1, 11, 8	1, 13, 6	0, 18, 2
AO*	SP	spect	0, 13, 7	1, 14, 5	1, 12, 7	0, 17, 3	0, 19, 1
AO*	SP-L	bupa	0, 15, 5	0, 15, 5	0, 17, 3	0, 17, 3	0, 14, 6
AO*	SP-L	pima	0, 17, 3	0, 8, 12	0, 2, 18	0, 10, 10	0, 12, 8
AO*	SP-L	heart	0, 20, 0	0, 9, 11	0, 5, 15	0, 18, 2	0, 18, 2
AO*	SP-L	b-can	1, 19, 0	0, 20, 0	2, 15, 3	1, 16, 3	0, 9, 11
AO*	SP-L	spect	0, 14, 6	0, 10, 10	0, 9, 11	0, 14, 6	1, 19, 0
AO*	PPP	bupa	0, 17, 3	0, 17, 3	0, 17, 3	0, 17, 3	0, 18, 2
AO*	PPP	pima	0, 16, 4	0, 16, 4	0, 11, 9	0, 16, 4	1, 15, 4
AO*	PPP	heart	0, 20, 0	0, 15, 5	0, 13, 7	0, 18, 2	0, 18, 2
AO*	PPP	b-can	0, 20, 0	0, 20, 0	1, 14, 5	1, 14, 5	0, 18, 2
AO*	PPP	spect	0, 18, 2	1, 17, 2	1, 13, 6	1, 18, 1	0, 18, 2
AO*	PPP-L	bupa	0, 16, 4	0, 16, 4	0, 16, 4	0, 16, 4	0, 19, 1
AO*	PPP-L	pima	0, 18, 2	0, 17, 3	0, 15, 5	0, 19, 1	0, 13, 7
AO*	PPP-L	heart	1, 19, 0	0, 17, 3	0, 15, 5	0, 20, 0	0, 20, 0
AO*	PPP-L	b-can	0, 20, 0	0, 20, 0	1, 19, 0	0, 19, 1	0, 12, 8
AO*	PPP-L	spect	0, 12, 8	0, 9, 11	0, 8, 12	0, 14, 6	1, 19, 0

TABLE B.7: BDeltaCost of AO* paired with each systematic search algorithm (except AO*-L), across all domains. Each table entry has (wins, ties, losses) of alg1 over alg2.

B.5 Paired-Graphs Comparing the Best Algorithm on Each Domain with Our Recommended Algorithms

We present several detailed graphs comparing the best algorithm on each domain with our top choices, SP-L and VOI-L (Figures B.1, B.2, B.3, B.4, B.5, B.6 and B.7).

alg1	alg2	domain	MC1	MC2	MC3	MC4	MC5
VOI-L	MC-N-L	bupa	2, 18, 0	2, 18, 0	3, 17, 0	3, 17, 0	5, 15, 0
VOI-L	MC-N-L	pima	0, 20, 0	0, 17, 3	0, 19, 1	2, 18, 0	5, 15, 0
VOI-L	MC-N-L	heart	0, 20, 0	0, 20, 0	0, 20, 0	0, 19, 1	0, 19, 1
VOI-L	MC-N-L	b-can	1, 19, 0	1, 19, 0	0, 17, 3	0, 14, 6	0, 14, 6
VOI-L	MC-N-L	spect	2, 18, 0	2, 18, 0	2, 18, 0	3, 17, 0	5, 12, 3
VOI-L	Nor-L	bupa	8, 12, 0	8, 12, 0	6, 14, 0	6, 14, 0	3, 17, 0
VOI-L	Nor-L	pima	4, 16, 0	0, 18, 2	1, 18, 1	1, 19, 0	6, 14, 0
VOI-L	Nor-L	heart	6, 14, 0	0, 17, 3	0, 15, 5	0, 17, 3	0, 17, 3
VOI-L	Nor-L	b-can	7, 13, 0	0, 20, 0	0, 18, 2	0, 16, 4	0, 14, 6
VOI-L	Nor-L	spect	9, 11, 0	5, 15, 0	5, 15, 0	5, 15, 0	10, 10, 0
VOI-L	AO*-L	bupa	0, 20, 0	0, 20, 0	0, 20, 0	0, 20, 0	2, 18, 0
VOI-L	AO*-L	pima	0, 20, 0	1, 17, 2	1, 19, 0	3, 17, 0	8, 12, 0
VOI-L	AO*-L	heart	0, 20, 0	0, 19, 1	0, 20, 0	0, 19, 1	0, 19, 1
VOI-L	AO*-L	b-can	0, 20, 0	0, 20, 0	0, 18, 2	0, 14, 6	0, 16, 4
VOI-L	AO*-L	spect	0, 20, 0	1, 19, 0	1, 19, 0	3, 16, 1	4, 15, 1
VOI-L	ES-L	bupa	0, 15, 5	0, 15, 5	0, 16, 4	0, 16, 4	0, 17, 3
VOI-L	ES-L	pima	1, 19, 0	1, 17, 2	2, 18, 0	3, 17, 0	5, 15, 0
VOI-L	ES-L	heart	2, 18, 0	0, 19, 1	1, 18, 1	0, 20, 0	0, 20, 0
VOI-L	ES-L	b-can	0, 19, 1	0, 20, 0	1, 18, 1	0, 17, 3	0, 19, 1
VOI-L	ES-L	spect	2, 18, 0	2, 18, 0	1, 19, 0	3, 16, 1	1, 19, 0
VOI-L	SP-L	bupa	0, 16, 4	0, 16, 4	0, 18, 2	0, 20, 0	1, 19, 0
VOI-L	SP-L	pima	0, 20, 0	0, 18, 2	0, 20, 0	3, 17, 0	4, 16, 0
VOI-L	SP-L	heart	0, 20, 0	0, 18, 2	0, 19, 1	0, 20, 0	0, 20, 0
VOI-L	SP-L	b-can	1, 19, 0	1, 19, 0	0, 20, 0	0, 14, 6	0, 15, 5
VOI-L	SP-L	spect	0, 20, 0	1, 19, 0	1, 19, 0	3, 15, 2	4, 15, 1
VOI-L	PPP-L	bupa	0, 17, 3	0, 17, 3	0, 17, 3	0, 17, 3	0, 19, 1
VOI-L	PPP-L	pima	0, 20, 0	0, 20, 0	1, 19, 0	7, 13, 0	3, 16, 1
VOI-L	PPP-L	heart	1, 19, 0	0, 17, 3	0, 17, 3	0, 17, 3	0, 17, 3
VOI-L	PPP-L	b-can	0, 20, 0	0, 20, 0	0, 20, 0	0, 17, 3	0, 17, 3
VOI-L	PPP-L	spect	0, 19, 1	1, 18, 1	0, 19, 1	2, 16, 2	4, 16, 0

TABLE B.8: BDeltaCost of VOI-L paired with each Laplace corrected algorithm, across all domains. Each table entry has (wins, ties, losses) of alg1 over alg2.

These paired-graphs of V_{test} first sort the V_{test} values for each replica in increasing order of V_{test} of alg1, and then display the V_{test} of alg2 for the corresponding replica. For alg1, we have selected the best algorithm on each domain according to the chess metric (see Table 5.48). The alg2 is either SP-L, or VOI-L. The two V_{test} values are

alg1	alg2	domain	MC1	MC2	MC3	MC4	MC5
MC-N-L	VOI-L	bupa	0, 18, 2	0, 18, 2	0, 17, 3	0, 17, 3	0, 15, 5
MC-N-L	VOI-L	pima	0, 20, 0	3, 17, 0	1, 19, 0	0, 18, 2	0, 15, 5
MC-N-L	VOI-L	heart	0, 20, 0	0, 20, 0	0, 20, 0	1, 19, 0	1, 19, 0
MC-N-L	VOI-L	b-can	0, 19, 1	0, 19, 1	3, 17, 0	6, 14, 0	6, 14, 0
MC-N-L	VOI-L	spect	0, 18, 2	0, 18, 2	0, 18, 2	0, 17, 3	3, 12, 5
MC-N-L	Nor-L	bupa	2, 16, 2	2, 16, 2	1, 17, 2	1, 17, 2	0, 18, 2
MC-N-L	Nor-L	pima	4, 16, 0	1, 19, 0	2, 16, 2	0, 18, 2	3, 16, 1
MC-N-L	Nor-L	heart	6, 14, 0	0, 20, 0	0, 20, 0	0, 20, 0	0, 20, 0
MC-N-L	Nor-L	b-can	0, 20, 0	0, 20, 0	2, 18, 0	2, 17, 1	3, 15, 2
MC-N-L	Nor-L	spect	9, 11, 0	5, 15, 0	5, 15, 0	3, 17, 0	8, 12, 0
MC-N-L	AO*-L	bupa	0, 18, 2	0, 18, 2	0, 17, 3	0, 17, 3	0, 15, 5
MC-N-L	AO*-L	pima	0, 20, 0	3, 16, 1	3, 17, 0	2, 18, 0	0, 19, 1
MC-N-L	AO*-L	heart	0, 20, 0	3, 16, 1	2, 18, 0	2, 17, 1	2, 17, 1
MC-N-L	AO*-L	b-can	0, 19, 1	0, 19, 1	1, 19, 0	1, 19, 0	6, 14, 0
MC-N-L	AO*-L	spect	0, 18, 2	0, 19, 1	0, 19, 1	1, 16, 3	2, 17, 1
MC-N-L	ES-L	bupa	0, 16, 4	0, 16, 4	0, 14, 6	0, 14, 6	0, 15, 5
MC-N-L	ES-L	pima	1, 17, 2	1, 18, 1	2, 16, 2	1, 18, 1	1, 19, 0
MC-N-L	ES-L	heart	2, 18, 0	2, 17, 1	2, 16, 2	0, 20, 0	0, 20, 0
MC-N-L	ES-L	b-can	0, 19, 1	1, 18, 1	2, 18, 0	4, 16, 0	4, 16, 0
MC-N-L	ES-L	spect	1, 18, 1	0, 20, 0	1, 19, 0	0, 18, 2	2, 17, 1
MC-N-L	SP-L	bupa	0, 17, 3	0, 17, 3	0, 15, 5	0, 17, 3	0, 14, 6
MC-N-L	SP-L	pima	0, 20, 0	0, 19, 1	2, 17, 1	1, 19, 0	0, 19, 1
MC-N-L	SP-L	heart	0, 20, 0	1, 17, 2	1, 17, 2	1, 19, 0	1, 19, 0
MC-N-L	SP-L	b-can	0, 20, 0	0, 20, 0	2, 18, 0	2, 17, 1	1, 18, 1
MC-N-L	SP-L	spect	0, 18, 2	0, 19, 1	0, 19, 1	1, 16, 3	1, 18, 1
MC-N-L	PPP-L	bupa	0, 18, 2	0, 18, 2	0, 15, 5	0, 15, 5	0, 17, 3
MC-N-L	PPP-L	pima	0, 20, 0	2, 18, 0	1, 19, 0	4, 16, 0	0, 19, 1
MC-N-L	PPP-L	heart	1, 19, 0	1, 19, 0	1, 19, 0	0, 20, 0	0, 20, 0
MC-N-L	PPP-L	b-can	0, 19, 1	0, 19, 1	1, 19, 0	2, 18, 0	2, 16, 2
MC-N-L	PPP-L	spect	0, 18, 2	0, 18, 2	0, 18, 2	1, 15, 4	0, 20, 0

TABLE B.9: BDeltaCost of MC-N-L paired with each Laplace corrected algorithm, across all domains. Each table entry has (wins, ties, losses) of alg1 over alg2.

connected by a vertical line in each replica where BDeltaCost could not reject the null hypothesis that they are tied. On each domain, as misclassification costs increase, the replicas on the x axis are not in the same order, because on each misclassification cost level MC the order of the replicas depends on the V_{test} of alg1.

alg1	alg2	domain	MC1	MC2	MC3	MC4	MC5
Nor-L	VOI-L	bupa	0, 12, 8	0, 12, 8	0, 14, 6	0, 14, 6	0, 17, 3
Nor-L	VOI-L	pima	0, 16, 4	2, 18, 0	1, 18, 1	0, 19, 1	0, 14, 6
Nor-L	VOI-L	heart	0, 14, 6	3, 17, 0	5, 15, 0	3, 17, 0	3, 17, 0
Nor-L	VOI-L	b-can	0, 13, 7	0, 20, 0	2, 18, 0	4, 16, 0	6, 14, 0
Nor-L	VOI-L	spect	0, 11, 9	0, 15, 5	0, 15, 5	0, 15, 5	0, 10, 10
Nor-L	MC-N-L	bupa	2, 16, 2	2, 16, 2	2, 17, 1	2, 17, 1	2, 18, 0
Nor-L	MC-N-L	pima	0, 16, 4	0, 19, 1	2, 16, 2	2, 18, 0	1, 16, 3
Nor-L	MC-N-L	heart	0, 14, 6	0, 20, 0	0, 20, 0	0, 20, 0	0, 20, 0
Nor-L	MC-N-L	b-can	0, 20, 0	0, 20, 0	0, 18, 2	1, 17, 2	2, 15, 3
Nor-L	MC-N-L	spect	0, 11, 9	0, 15, 5	0, 15, 5	0, 17, 3	0, 12, 8
Nor-L	AO*-L	bupa	0, 12, 8	0, 12, 8	0, 15, 5	0, 15, 5	0, 19, 1
Nor-L	AO*-L	pima	0, 16, 4	1, 19, 0	4, 15, 1	3, 17, 0	1, 16, 3
Nor-L	AO*-L	heart	0, 14, 6	0, 19, 1	0, 20, 0	0, 19, 1	0, 19, 1
Nor-L	AO*-L	b-can	0, 13, 7	0, 20, 0	0, 18, 2	2, 17, 1	4, 13, 3
Nor-L	AO*-L	spect	0, 11, 9	0, 15, 5	0, 15, 5	0, 17, 3	0, 15, 5
Nor-L	ES-L	bupa	0, 10, 10	0, 10, 10	0, 16, 4	0, 16, 4	0, 17, 3
Nor-L	ES-L	pima	0, 14, 6	4, 16, 0	1, 17, 2	1, 18, 1	0, 18, 2
Nor-L	ES-L	heart	1, 14, 5	0, 20, 0	1, 19, 0	0, 20, 0	0, 20, 0
Nor-L	ES-L	b-can	0, 14, 6	1, 19, 0	2, 17, 1	3, 16, 1	3, 15, 2
Nor-L	ES-L	spect	0, 13, 7	0, 13, 7	0, 13, 7	0, 17, 3	0, 13, 7
Nor-L	SP-L	bupa	0, 10, 10	0, 10, 10	0, 14, 6	0, 15, 5	0, 18, 2
Nor-L	SP-L	pima	0, 16, 4	0, 19, 1	0, 19, 1	2, 18, 0	0, 17, 3
Nor-L	SP-L	heart	0, 14, 6	0, 18, 2	0, 19, 1	0, 20, 0	0, 20, 0
Nor-L	SP-L	b-can	0, 19, 1	1, 19, 0	0, 19, 1	1, 19, 0	2, 15, 3
Nor-L	SP-L	spect	0, 11, 9	0, 15, 5	0, 15, 5	0, 17, 3	0, 14, 6
Nor-L	PPP-L	bupa	0, 14, 6	0, 14, 6	0, 14, 6	0, 14, 6	0, 16, 4
Nor-L	PPP-L	pima	0, 19, 1	2, 18, 0	2, 18, 0	5, 15, 0	0, 15, 5
Nor-L	PPP-L	heart	0, 15, 5	1, 19, 0	0, 20, 0	1, 19, 0	1, 19, 0
Nor-L	PPP-L	b-can	0, 13, 7	0, 20, 0	0, 19, 1	2, 17, 1	3, 13, 4
Nor-L	PPP-L	spect	0, 11, 9	0, 11, 9	0, 11, 9	0, 16, 4	0, 16, 4

TABLE B.10: BDeltaCost of Nor-L paired with each Laplace corrected algorithm, across all domains. Each table entry has (wins, ties, losses) of alg1 over alg2.

alg1	alg2	domain	MC1	MC2	MC3	MC4	MC5
AO*-L	VOI-L	bupa	0, 20, 0	0, 20, 0	0, 20, 0	0, 20, 0	0, 18, 2
AO*-L	VOI-L	pima	0, 20, 0	2, 17, 1	0, 19, 1	0, 17, 3	0, 12, 8
AO*-L	VOI-L	heart	0, 20, 0	1, 19, 0	0, 20, 0	1, 19, 0	1, 19, 0
AO*-L	VOI-L	b-can	0, 20, 0	0, 20, 0	2, 18, 0	6, 14, 0	4, 16, 0
AO*-L	VOI-L	spect	0, 20, 0	0, 19, 1	0, 19, 1	1, 16, 3	1, 15, 4
AO*-L	MC-N-L	bupa	2, 18, 0	2, 18, 0	3, 17, 0	3, 17, 0	5, 15, 0
AO*-L	MC-N-L	pima	0, 20, 0	1, 16, 3	0, 17, 3	0, 18, 2	1, 19, 0
AO*-L	MC-N-L	heart	0, 20, 0	1, 16, 3	0, 18, 2	1, 17, 2	1, 17, 2
AO*-L	MC-N-L	b-can	1, 19, 0	1, 19, 0	0, 19, 1	0, 19, 1	0, 14, 6
AO*-L	MC-N-L	spect	2, 18, 0	1, 19, 0	1, 19, 0	3, 16, 1	1, 17, 2
AO*-L	Nor-L	bupa	8, 12, 0	8, 12, 0	5, 15, 0	5, 15, 0	1, 19, 0
AO*-L	Nor-L	pima	4, 16, 0	0, 19, 1	1, 15, 4	0, 17, 3	3, 16, 1
AO*-L	Nor-L	heart	6, 14, 0	1, 19, 0	0, 20, 0	1, 19, 0	1, 19, 0
AO*-L	Nor-L	b-can	7, 13, 0	0, 20, 0	2, 18, 0	1, 17, 2	3, 13, 4
AO*-L	Nor-L	spect	9, 11, 0	5, 15, 0	5, 15, 0	3, 17, 0	5, 15, 0
AO*-L	ES-L	bupa	0, 15, 5	0, 15, 5	0, 16, 4	0, 16, 4	0, 18, 2
AO*-L	ES-L	pima	1, 19, 0	2, 17, 1	1, 18, 1	0, 18, 2	1, 17, 2
AO*-L	ES-L	heart	2, 18, 0	2, 17, 1	2, 17, 1	1, 19, 0	1, 19, 0
AO*-L	ES-L	b-can	0, 19, 1	0, 20, 0	1, 19, 0	4, 16, 0	4, 14, 2
AO*-L	ES-L	spect	2, 18, 0	1, 19, 0	1, 19, 0	1, 17, 2	1, 18, 1
AO*-L	SP-L	bupa	0, 16, 4	0, 16, 4	0, 18, 2	0, 20, 0	0, 19, 1
AO*-L	SP-L	pima	0, 20, 0	1, 15, 4	0, 15, 5	1, 17, 2	1, 15, 4
AO*-L	SP-L	heart	0, 20, 0	1, 16, 3	1, 15, 4	1, 13, 6	1, 13, 6
AO*-L	SP-L	b-can	1, 19, 0	1, 19, 0	1, 19, 0	1, 18, 1	0, 14, 6
AO*-L	SP-L	spect	0, 20, 0	0, 20, 0	0, 19, 1	0, 18, 2	0, 19, 1
AO*-L	PPP-L	bupa	0, 17, 3	0, 17, 3	0, 17, 3	0, 17, 3	0, 17, 3
AO*-L	PPP-L	pima	0, 20, 0	1, 19, 0	0, 20, 0	4, 16, 0	0, 18, 2
AO*-L	PPP-L	heart	1, 19, 0	2, 18, 0	1, 19, 0	2, 18, 0	2, 18, 0
AO*-L	PPP-L	b-can	0, 20, 0	0, 20, 0	0, 20, 0	1, 19, 0	2, 13, 5
AO*-L	PPP-L	spect	0, 19, 1	0, 19, 1	0, 19, 1	1, 17, 2	3, 15, 2

TABLE B.11: BDeltaCost of AO*-L paired with each Laplace corrected algorithm, across all domains. Each table entry has (wins, ties, losses) of alg1 over alg2.

alg1	alg2	domain	MC1	MC2	MC3	MC4	MC5
ES-L	VOI-L	bupa	5, 15, 0	5, 15, 0	4, 16, 0	4, 16, 0	3, 17, 0
ES-L	VOI-L	pima	0, 19, 1	2, 17, 1	0, 18, 2	0, 17, 3	0, 15, 5
ES-L	VOI-L	heart	0, 18, 2	1, 19, 0	1, 18, 1	0, 20, 0	0, 20, 0
ES-L	VOI-L	b-can	1, 19, 0	0, 20, 0	1, 18, 1	3, 17, 0	1, 19, 0
ES-L	VOI-L	spect	0, 18, 2	0, 18, 2	0, 19, 1	1, 16, 3	0, 19, 1
ES-L	MC-N-L	bupa	4, 16, 0	4, 16, 0	6, 14, 0	6, 14, 0	5, 15, 0
ES-L	MC-N-L	pima	2, 17, 1	1, 18, 1	2, 16, 2	1, 18, 1	0, 19, 1
ES-L	MC-N-L	heart	0, 18, 2	1, 17, 2	2, 16, 2	0, 20, 0	0, 20, 0
ES-L	MC-N-L	b-can	1, 19, 0	1, 18, 1	0, 18, 2	0, 16, 4	0, 16, 4
ES-L	MC-N-L	spect	1, 18, 1	0, 20, 0	0, 19, 1	2, 18, 0	1, 17, 2
ES-L	Nor-L	bupa	10, 10, 0	10, 10, 0	4, 16, 0	4, 16, 0	3, 17, 0
ES-L	Nor-L	pima	6, 14, 0	0, 16, 4	2, 17, 1	1, 18, 1	2, 18, 0
ES-L	Nor-L	heart	5, 14, 1	0, 20, 0	0, 19, 1	0, 20, 0	0, 20, 0
ES-L	Nor-L	b-can	6, 14, 0	0, 19, 1	1, 17, 2	1, 16, 3	2, 15, 3
ES-L	Nor-L	spect	7, 13, 0	7, 13, 0	7, 13, 0	3, 17, 0	7, 13, 0
ES-L	AO*-L	bupa	5, 15, 0	5, 15, 0	4, 16, 0	4, 16, 0	2, 18, 0
ES-L	AO*-L	pima	0, 19, 1	1, 17, 2	1, 18, 1	2, 18, 0	2, 17, 1
ES-L	AO*-L	heart	0, 18, 2	1, 17, 2	1, 17, 2	0, 19, 1	0, 19, 1
ES-L	AO*-L	b-can	1, 19, 0	0, 20, 0	0, 19, 1	0, 16, 4	2, 14, 4
ES-L	AO*-L	spect	0, 18, 2	0, 19, 1	0, 19, 1	2, 17, 1	1, 18, 1
ES-L	SP-L	bupa	1, 19, 0	1, 19, 0	0, 20, 0	3, 17, 0	2, 18, 0
ES-L	SP-L	pima	0, 19, 1	1, 18, 1	0, 18, 2	1, 19, 0	1, 18, 1
ES-L	SP-L	heart	0, 18, 2	0, 16, 4	0, 16, 4	0, 18, 2	0, 18, 2
ES-L	SP-L	b-can	2, 18, 0	1, 19, 0	1, 18, 1	1, 15, 4	0, 14, 6
ES-L	SP-L	spect	0, 18, 2	0, 19, 1	0, 19, 1	2, 17, 1	1, 18, 1
ES-L	PPP-L	bupa	1, 19, 0	1, 19, 0	0, 20, 0	0, 20, 0	0, 20, 0
ES-L	PPP-L	pima	3, 16, 1	0, 20, 0	3, 16, 1	2, 17, 1	1, 17, 2
ES-L	PPP-L	heart	1, 17, 2	2, 18, 0	1, 19, 0	2, 17, 1	2, 17, 1
ES-L	PPP-L	b-can	1, 19, 0	0, 20, 0	0, 19, 1	0, 17, 3	0, 17, 3
ES-L	PPP-L	spect	0, 17, 3	0, 18, 2	0, 18, 2	2, 17, 1	3, 16, 1

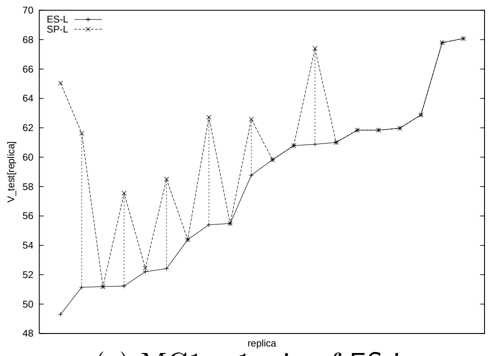
TABLE B.12: BDeltaCost of ES-L paired with each Laplace corrected algorithm, across all domains. Each table entry has (wins, ties, losses) of alg1 over alg2.

alg1	alg2	domain	MC1	MC2	MC3	MC4	MC5
SP-L	VOI-L	bupa	4, 16, 0	4, 16, 0	2, 18, 0	0, 20, 0	0, 19, 1
SP-L	VOI-L	pima	0, 20, 0	2, 18, 0	0, 20, 0	0, 17, 3	0, 16, 4
SP-L	VOI-L	heart	0, 20, 0	2, 18, 0	1, 19, 0	0, 20, 0	0, 20, 0
SP-L	VOI-L	b-can	0, 19, 1	0, 19, 1	0, 20, 0	6, 14, 0	5, 15, 0
SP-L	VOI-L	spect	0, 20, 0	0, 19, 1	0, 19, 1	2, 15, 3	1, 15, 4
SP-L	MC-N-L	bupa	3, 17, 0	3, 17, 0	5, 15, 0	3, 17, 0	6, 14, 0
SP-L	MC-N-L	pima	0, 20, 0	1, 19, 0	1, 17, 2	0, 19, 1	1, 19, 0
SP-L	MC-N-L	heart	0, 20, 0	2, 17, 1	2, 17, 1	0, 19, 1	0, 19, 1
SP-L	MC-N-L	b-can	0, 20, 0	0, 20, 0	0, 18, 2	1, 17, 2	1, 18, 1
SP-L	MC-N-L	spect	2, 18, 0	1, 19, 0	1, 19, 0	3, 16, 1	1, 18, 1
SP-L	Nor-L	bupa	10, 10, 0	10, 10, 0	6, 14, 0	5, 15, 0	2, 18, 0
SP-L	Nor-L	pima	4, 16, 0	1, 19, 0	1, 19, 0	0, 18, 2	3, 17, 0
SP-L	Nor-L	heart	6, 14, 0	2, 18, 0	1, 19, 0	0, 20, 0	0, 20, 0
SP-L	Nor-L	b-can	1, 19, 0	0, 19, 1	1, 19, 0	0, 19, 1	3, 15, 2
SP-L	Nor-L	spect	9, 11, 0	5, 15, 0	5, 15, 0	3, 17, 0	6, 14, 0
SP-L	AO*-L	bupa	4, 16, 0	4, 16, 0	2, 18, 0	0, 20, 0	1, 19, 0
SP-L	AO*-L	pima	0, 20, 0	4, 15, 1	5, 15, 0	2, 17, 1	4, 15, 1
SP-L	AO*-L	heart	0, 20, 0	3, 16, 1	4, 15, 1	6, 13, 1	6, 13, 1
SP-L	AO*-L	b-can	0, 19, 1	0, 19, 1	0, 19, 1	1, 18, 1	6, 14, 0
SP-L	AO*-L	spect	0, 20, 0	0, 20, 0	1, 19, 0	2, 18, 0	1, 19, 0
SP-L	ES-L	bupa	0, 19, 1	0, 19, 1	0, 20, 0	0, 17, 3	0, 18, 2
SP-L	ES-L	pima	1, 19, 0	1, 18, 1	2, 18, 0	0, 19, 1	1, 18, 1
SP-L	ES-L	heart	2, 18, 0	4, 16, 0	4, 16, 0	2, 18, 0	2, 18, 0
SP-L	ES-L	b-can	0, 18, 2	0, 19, 1	1, 18, 1	4, 15, 1	6, 14, 0
SP-L	ES-L	spect	2, 18, 0	1, 19, 0	1, 19, 0	1, 17, 2	1, 18, 1
SP-L	PPP-L	bupa	0, 20, 0	0, 20, 0	0, 20, 0	0, 18, 2	0, 18, 2
SP-L	PPP-L	pima	0, 20, 0	2, 18, 0	3, 17, 0	3, 17, 0	0, 19, 1
SP-L	PPP-L	heart	1, 19, 0	4, 16, 0	4, 16, 0	1, 18, 1	1, 18, 1
SP-L	PPP-L	b-can	0, 19, 1	0, 19, 1	0, 19, 1	1, 18, 1	3, 16, 1
SP-L	PPP-L	spect	0, 19, 1	0, 19, 1	0, 19, 1	1, 18, 1	2, 16, 2

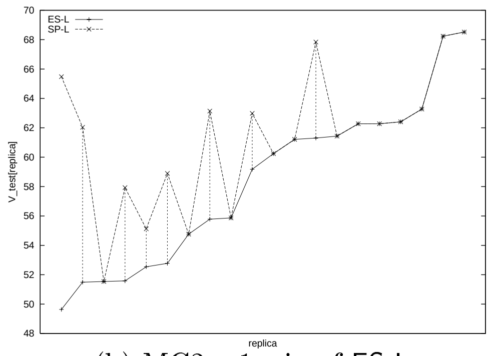
TABLE B.13: BDeltaCost of SP-L paired with each Laplace corrected algorithm, across all domains. Each table entry has (wins, ties, losses) of alg1 over alg2.

alg1	alg2	domain	MC1	MC2	MC3	MC4	MC5
PPP-L	VOI-L	bupa	3, 17, 0	3, 17, 0	3, 17, 0	3, 17, 0	1, 19, 0
PPP-L	VOI-L	pima	0, 20, 0	0, 20, 0	0, 19, 1	0, 13, 7	1, 16, 3
PPP-L	VOI-L	heart	0, 19, 1	3, 17, 0	3, 17, 0	3, 17, 0	3, 17, 0
PPP-L	VOI-L	b-can	0, 20, 0	0, 20, 0	0, 20, 0	3, 17, 0	3, 17, 0
PPP-L	VOI-L	spect	1, 19, 0	1, 18, 1	1, 19, 0	2, 16, 2	0, 16, 4
PPP-L	MC-N-L	bupa	2, 18, 0	2, 18, 0	5, 15, 0	5, 15, 0	3, 17, 0
PPP-L	MC-N-L	pima	0, 20, 0	0, 18, 2	0, 19, 1	0, 16, 4	1, 19, 0
PPP-L	MC-N-L	heart	0, 19, 1	0, 19, 1	0, 19, 1	0, 20, 0	0, 20, 0
PPP-L	MC-N-L	b-can	1, 19, 0	1, 19, 0	0, 19, 1	0, 18, 2	2, 16, 2
PPP-L	MC-N-L	spect	2, 18, 0	2, 18, 0	2, 18, 0	4, 15, 1	0, 20, 0
PPP-L	Nor-L	bupa	6, 14, 0	6, 14, 0	6, 14, 0	6, 14, 0	4, 16, 0
PPP-L	Nor-L	pima	1, 19, 0	0, 18, 2	0, 18, 2	0, 15, 5	5, 15, 0
PPP-L	Nor-L	heart	5, 15, 0	0, 19, 1	0, 20, 0	0, 19, 1	0, 19, 1
PPP-L	Nor-L	b-can	7, 13, 0	0, 20, 0	1, 19, 0	1, 17, 2	4, 13, 3
PPP-L	Nor-L	spect	9, 11, 0	9, 11, 0	9, 11, 0	4, 16, 0	4, 16, 0
PPP-L	AO*-L	bupa	3, 17, 0	3, 17, 0	3, 17, 0	3, 17, 0	3, 17, 0
PPP-L	AO*-L	pima	0, 20, 0	0, 19, 1	0, 20, 0	0, 16, 4	2, 18, 0
PPP-L	AO*-L	heart	0, 19, 1	0, 18, 2	0, 19, 1	0, 18, 2	0, 18, 2
PPP-L	AO*-L	b-can	0, 20, 0	0, 20, 0	0, 20, 0	0, 19, 1	5, 13, 2
PPP-L	AO*-L	spect	1, 19, 0	1, 19, 0	1, 19, 0	2, 17, 1	2, 15, 3
PPP-L	ES-L	bupa	0, 19, 1	0, 19, 1	0, 20, 0	0, 20, 0	0, 20, 0
PPP-L	ES-L	pima	1, 16, 3	0, 20, 0	1, 16, 3	1, 17, 2	2, 17, 1
PPP-L	ES-L	heart	2, 17, 1	0, 18, 2	0, 19, 1	1, 17, 2	1, 17, 2
PPP-L	ES-L	b-can	0, 19, 1	0, 20, 0	1, 19, 0	3, 17, 0	3, 17, 0
PPP-L	ES-L	spect	3, 17, 0	2, 18, 0	2, 18, 0	1, 17, 2	1, 16, 3
PPP-L	SP-L	bupa	0, 20, 0	0, 20, 0	0, 20, 0	2, 18, 0	2, 18, 0
PPP-L	SP-L	pima	0, 20, 0	0, 18, 2	0, 17, 3	0, 17, 3	1, 19, 0
PPP-L	SP-L	heart	0, 19, 1	0, 16, 4	0, 16, 4	1, 18, 1	1, 18, 1
PPP-L	SP-L	b-can	1, 19, 0	1, 19, 0	1, 19, 0	1, 18, 1	1, 16, 3
PPP-L	SP-L	spect	1, 19, 0	1, 19, 0	1, 19, 0	1, 18, 1	2, 16, 2

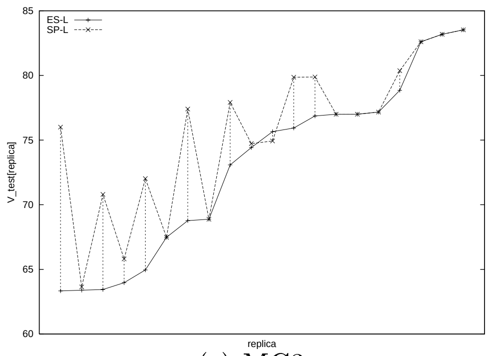
TABLE B.14: BDeltaCost of PPP-L paired with each Laplace corrected algorithm, across all domains. Each table entry has (wins, ties, losses) of alg1 over alg2.



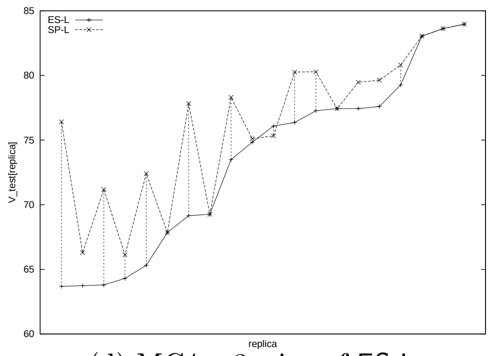
(a) MC1 – 1 win of ES-L



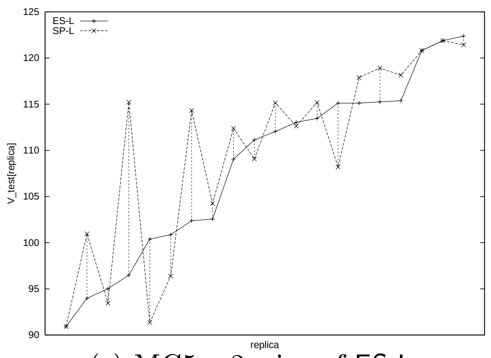
(b) MC2 – 1 win of ES-L



(c) MC3

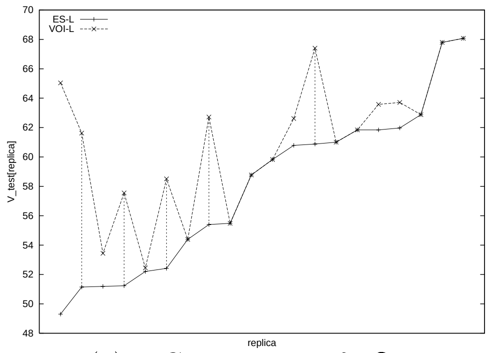


(d) MC4 – 3 wins of ES-L

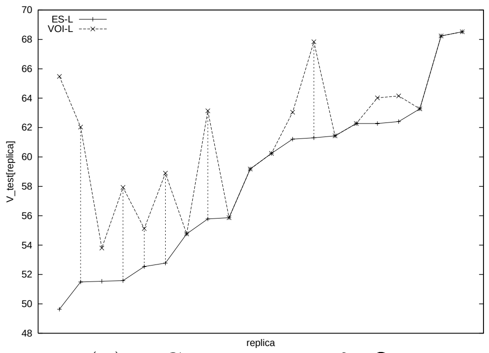


(e) MC5 – 2 wins of ES-L

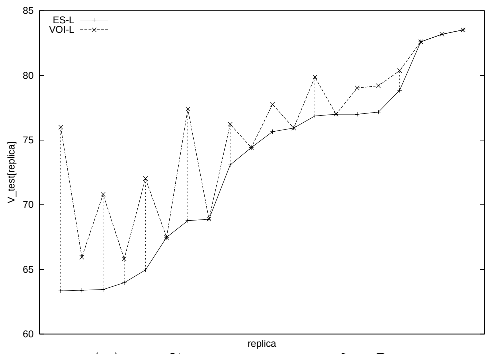
FIGURE B.1: Bupa, paired-graphs V_{test} of ES-L and SP-L for every replica, ordered by V_{test} of ES-L (best algorithm on bupa). Vertical lines connect V_{test} values that are tied according to BDeltaCost. Though mostly BDeltaCost-tied, V_{test} of ES-L is smaller than SP-L's.



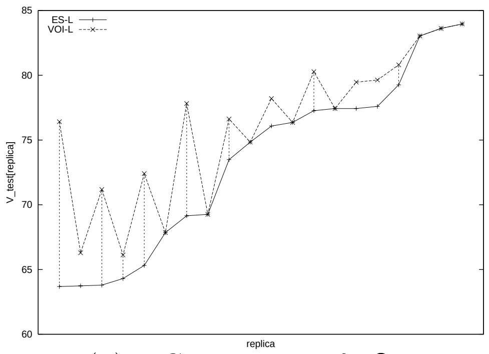
(a) MC1 – 5 wins of ES-L



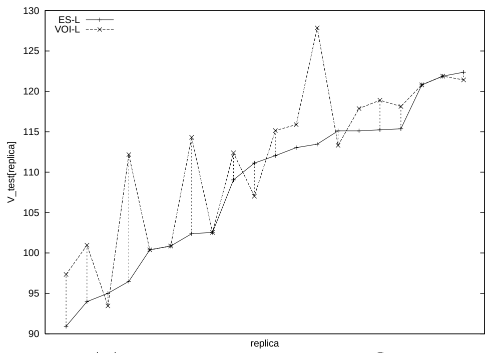
(b) MC2 – 5 wins of ES-L



(c) MC3 – 4 wins of ES-L

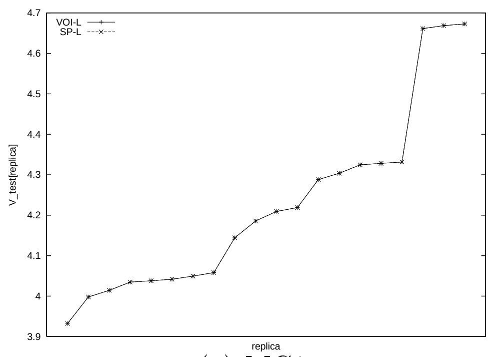


(d) MC4 – 4 wins of ES-L

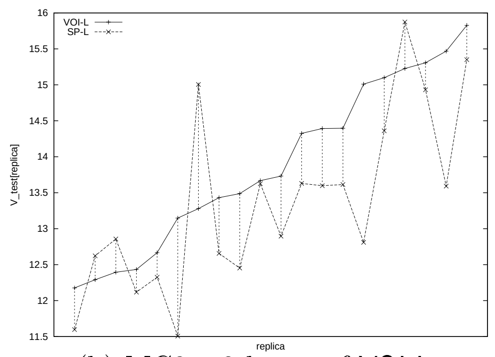


(e) MC5 – 3 wins of ES-L

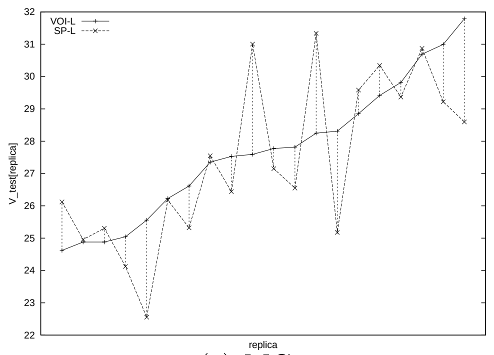
FIGURE B.2: Bupa, paired-graphs V_{test} of ES-L and VOI-L for every replica, ordered by V_{test} of ES-L (best algorithm on bupa). They confirm the superiority of ES-L over VOI-L. Vertical lines connect V_{test} values that are tied according to BDeltaCost. ES-L has wins over VOI-L and a smaller V_{test}.



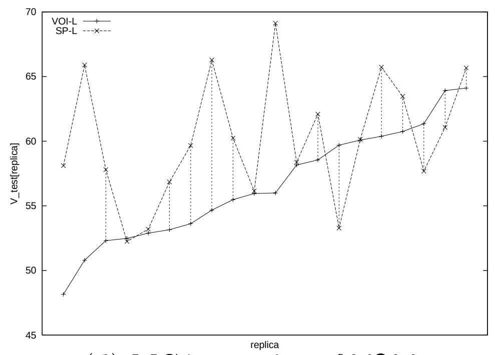
(a) MC1



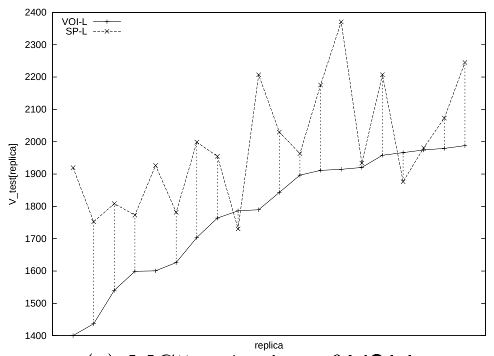
(b) MC2 – 2 losses of VOI-L



(c) MC3

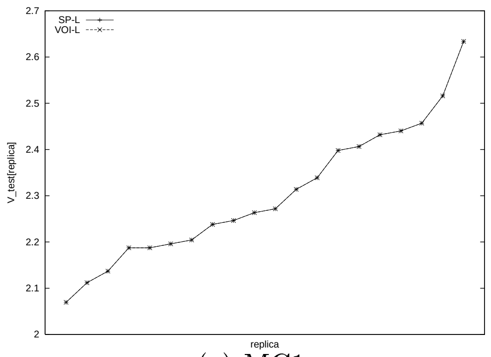


(d) MC4 – 3 wins of VOI-L

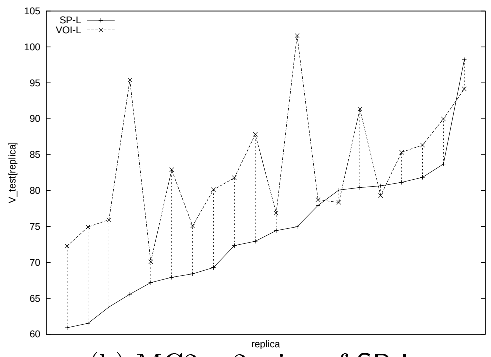


(e) MC5 – 4 wins of VOI-L

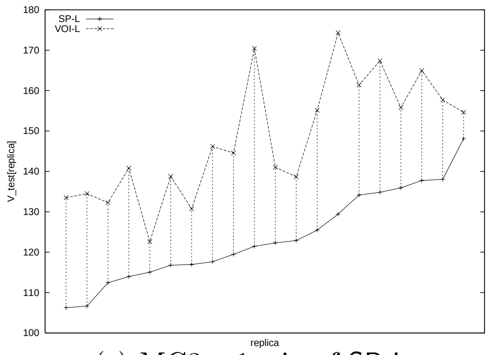
FIGURE B.3: Pima, paired-graphs V_{test} of VOI-L and SP-L for every replica, ordered by V_{test} of VOI-L (best algorithm on pima). For MC1, the two algorithms compute identical policies. Vertical lines connect V_{test} values that are tied according to BDeltaCost. For small misclassification costs, V_{test} of SP-L is mostly better than V_{test} of VOI-L, but this trend reverses for larger misclassification costs; this agrees with BDeltaCost, which picks several wins of VOI-L over SP-L.



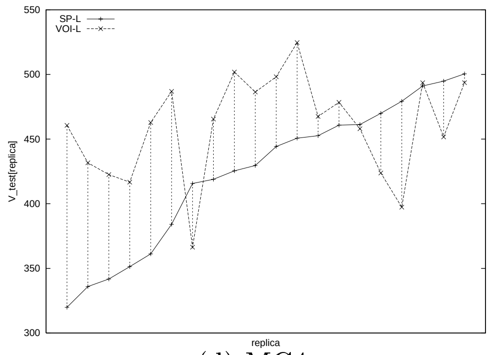
(a) MC1



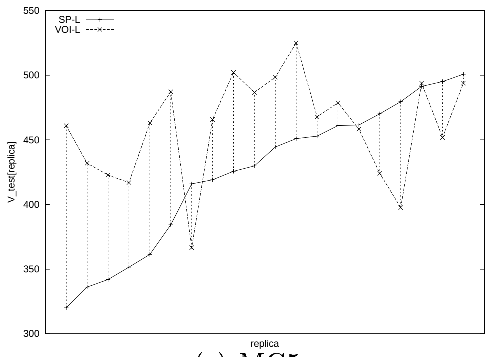
(b) MC2 – 2 wins of SP-L



(c) MC3 – 1 win of SP-L

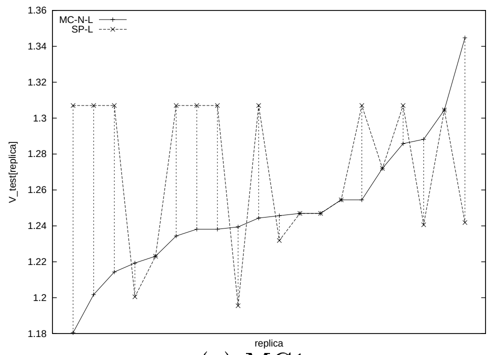


(d) MC4

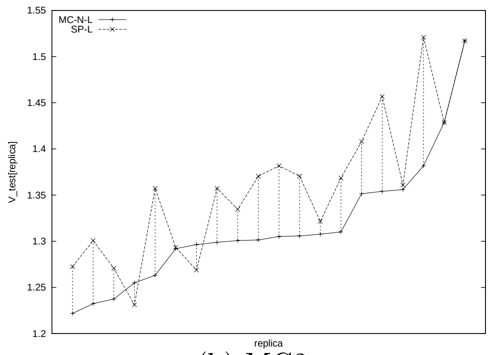


(e) MC5.

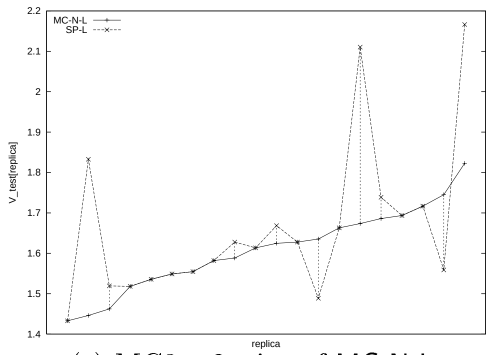
FIGURE B.4: Heart, paired-graphs V_{test} of SP-L and VOI-L for every replica, ordered by V_{test} of SP-L (best algorithm on heart). For MC1, the two algorithms compute identical policies. Vertical lines connect V_{test} values that are tied according to BDelta-Cost. V_{test} of SP-L is less than V_{test} of VOI-L, though they are mostly tied according to BDeltaCost.



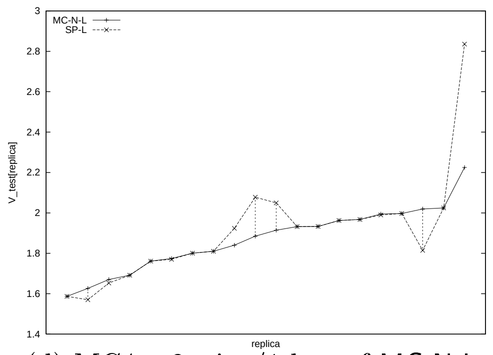
(a) MC1



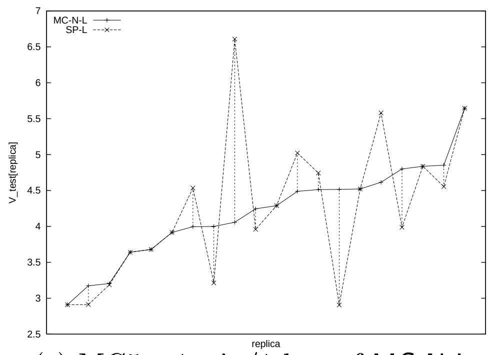
(b) MC2



(c) MC3 – 2 wins of MC-N-L

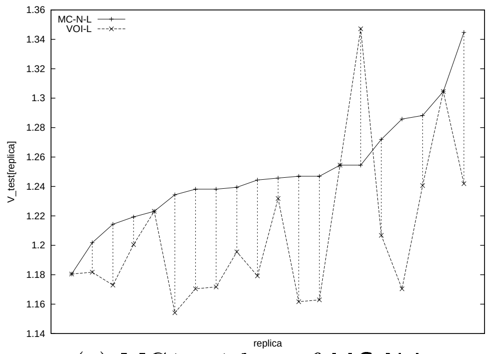


(d) MC4 – 2 wins/1 loss of MC-N-L

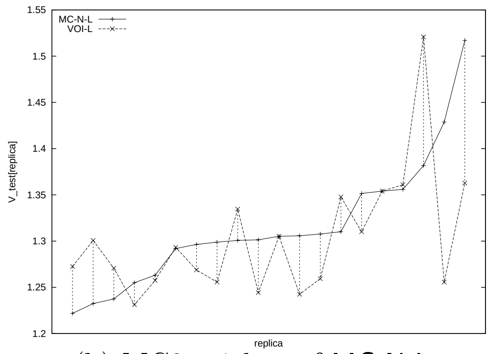


(e) MC5 – 1 win/1 loss of MC-N-L

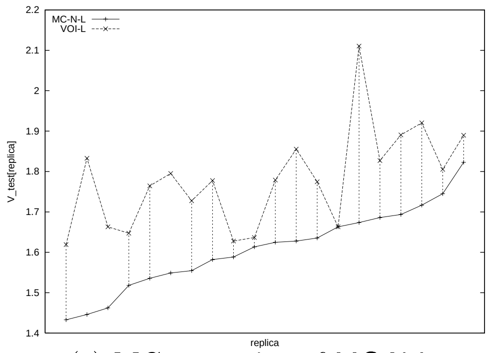
FIGURE B.5: Breast-cancer, paired-graphs V_{test} of MC-N-L and SP-L for every replica, ordered by V_{test} of MC-N-L (best algorithm on breast-cancer). Vertical lines connect V_{test} values that are tied according to BDeltaCost. The algorithms are mostly tied according to BDeltaCost. For small MC, V_{test} of MC-N-L is better, and remains mostly better for larger MCs, though the two graphs cross.



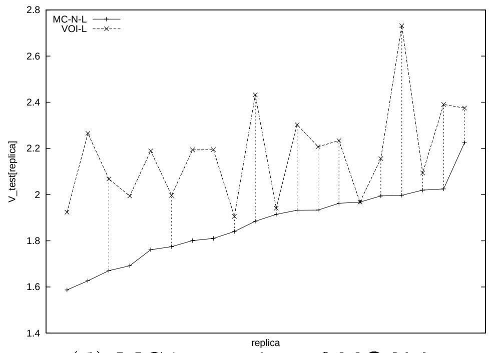
(a) MC1 – 1 loss of MC-N-L



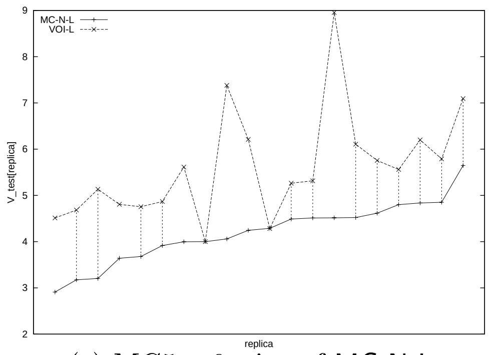
(b) MC2 – 1 loss of MC-N-L



(c) MC3 – 3 wins of MC-N-L

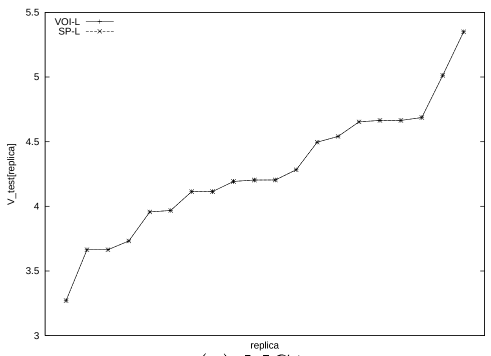


(d) MC4 – 6 wins of MC-N-L

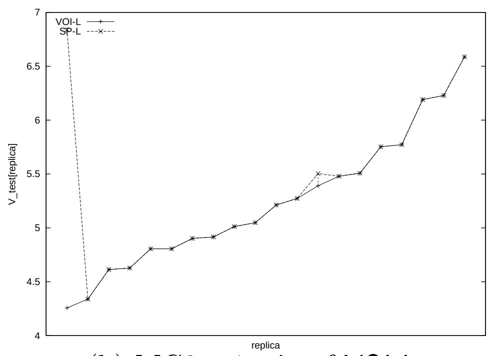


(e) MC5 – 6 wins of MC-N-L

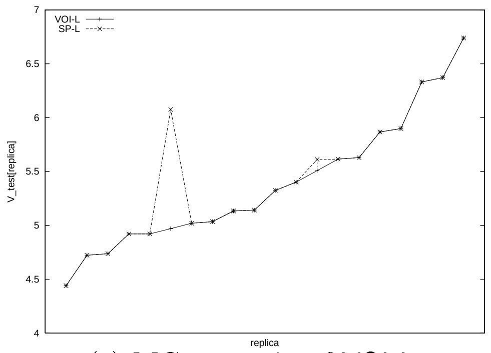
FIGURE B.6: Breast-cancer, paired-graphs V_{test} of MC-N-L and VOI-L for every replica, ordered by V_{test} of MC-N-L (best algorithm on breast-cancer). Vertical lines connect V_{test} values that are tied according to BDeltaCost. For larger MC, V_{test} of MC-N-L is better than V_{test} of VOI-L, which is confirmed by BDeltaCost; this trend reverses for smaller MCs, where V_{test} of MC-N-L is usually worse.



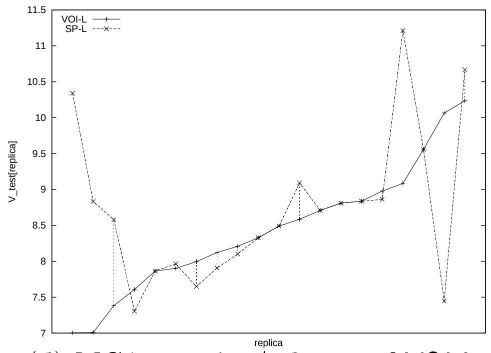
(a) MC1



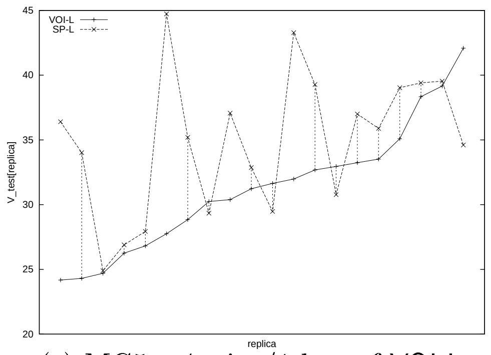
(b) MC2 – 1 win of VOI-L



(c) MC3 – 1 win of VOI-L



(d) MC4 – 3 wins/2 losses of VOI-L



(e) MC5 – 4 wins/1 loss of VOI-L

FIGURE B.7: Spect, paired-graphs V_{test} of VOI-L and SP-L for every replica, ordered by V_{test} of VOI-L (best algorithm on spect). For MC1, the two algorithms compute identical policies. Vertical lines connect V_{test} values that are tied according to BDeltaCost. For larger misclassification costs, the trend is that V_{test} of VOI-L is better than SP-L's, and BDeltaCost detects several wins of VOI-L.