2011

MACHINE LEARNING WITH INCOMPLETE INFORMATION

Yaling Zheng
University of Nebraska-Lincoln

MACHINE LEARNING WITH INCOMPLETE INFORMATION

by

Yaling Zheng

A DISSERTATION

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Doctor of Philosophy

Major: Computer Science

Under the Supervision of Professor Stephen D. Scott

Lincoln, Nebraska

December, 2011

Department of Computer Science & Engineering, University of Nebraska-Lincoln,
Technical Report, TR-UNL-CSE-2011-0013

MACHINE LEARNING WITH INCOMPLETE INFORMATION

Yaling Zheng, Ph.D.

University of Nebraska, 2011

Adviser: Stephen D. Scott

Machine learning algorithms detects patterns, regularities, and rules from the training data and adjust program actions accordingly. For example, when a learner (a computer program) sees a set of patient cases (patient records) with corresponding diagnoses, it can predict the presence of a disease for future patients. A somewhat unrealistic assumption in typical machine learning applications is that data is freely available. In my dissertation, I will present our research efforts to mitigate this assumption in the areas of active machine learning and budgeted machine learning.

In the area of active machine learning under the setting the labels of the instances have to be purchased, it is often assumed that there exists a perfect labeler labeling the chosen instances in the active machine learning setting. However it is possible that the labeler is not perfect, or it is possible there exists multiple noisy labelers with different known costs and different unknown accuracies, such as the Amazon Mechanical Turk. I will present our algorithms and experimental results of active learning from multiple noisy labelers with varied costs, which are based on ranking the labelers according to their estimated accuracies and costs. The experimental results show that our algorithms outperform those algorithms in the literature.

In the area of budgeted machine learning under the setting that the class label of every instance is known while the feature values of the instances have to be purchased at a cost, subject to an overall budget, the challenge to the learner is to decide which attributes of which instances will provide the best model from which to learn. I will present our budgeted

learning algorithms of naïve Bayes. Most of our algorithms perform well compared to existing algorithms in the literature. I will also present our algorithms for this budgeted learning of Bayesian network, which is a generalization of naïve Bayes. Experimental results show that some of our algorithms outperform those algorithms in the literature.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to many people who have supported and encouraged me throught my study at UNL. First I would like to thank my advisor and mentor, Dr. Stephen D. Scott, for his encouragement, patience, and meticulous guidance. Your enthusiasm, diligence, positive attitude, and confidence for research and teaching set a good example for me in every aspect of life. I also greatly appreciate the discussions, suggestions, and help from my colleagues Kun Deng, Michael Gubbels, Cate Anderson, Chris Bourke and many others. I would like to acknowledge my committee members Dr. Leen-Kiat Soh, Dr. Ashok Samal, Dr. Russell Greiner and Dr. Steve Dunbar for reading and providing valuable suggestions. Finally, I am grateful to my parents and my husband for their endless encouragement, love, and support.

Contents

Contents **v**

List of Figures **x**

List of Tables **xvii**

1 Introduction **1**

1.1 Introduction 1

2 Background **9**

2.1 Supervised Machine Learning 9

2.2 Examples of Classifiers 11

2.2.1 Naïve Bayes Classifier 11

2.2.1.1 Bayes' Theorem 11

2.2.2 Bayesian Network Classifier 13

2.2.2.1 Bayesian Networks 14

2.2.2.2 Bayesian Network Learning 17

2.2.2.3 Inference in Bayesian Networks 19

2.2.2.4 d-separation 23

2.2.2.5 Markov Blanket 25

2.3 Supervised Machine Learning with Incomplete Information 27

3 Active Learning from Multiple Labelers with Varied Costs 31

3.1 Introduction 31

3.2 Related Work 36

3.3 Our Algorithms 38

3.3.1 IEAdjCost 39

3.3.1.1 Notation 39

3.3.1.2 Selecting Instances 39

3.3.1.3 Estimating Labeler Accuracy 40

3.3.1.4 Combined Accuracy and Adjusted Cost 43

3.3.1.5 Choosing the Final Set of Labelers 45

3.3.1.6 IEAdjCost 49

3.3.2 wIEAdjCost 49

3.3.2.1 Notation 51

3.3.2.2 Procedure LabelersbywAdjCost 52

3.3.2.3 Computing O_r 53

3.3.2.4 Algorithm wIEAdjCost 55

3.4 Experimental Results 55

3.4.1 Experimental Results on UCI Data Sets 55

3.4.1.1 Uniform Accuracies 58

3.4.1.2 Setting λ 62

3.4.1.3 Low Accuracy Labelers 65

3.4.1.4 Conclusions on UCI Data Sets 67

3.4.2 Experiments on AMT data sets 71

3.5 Conclusions & Future Work 73

4 New Algorithms for Budgeted Learning **77**

4.1 Introduction 77

4.2 Background and Related Work 78

4.2.1 Biased Robin 83

4.2.2 Single-Feature Lookahead 83

4.2.3 The Multi-Armed Bandit Problem 85

4.2.3.1 Exp3 Algorithm 86

4.2.3.2 FPL Algorithm 86

4.2.3.3 Other Results 87

4.2.3.4 Discussion 88

4.3 Our Algorithms 88

4.3.1 Exp3-Based Algorithms (Exp3C and Exp3CR) 89

4.3.2 Follow the Expected Leader (FEL) 91

4.3.3 Variance-Based Biased Robin Algorithms 92

4.3.4 Instance (Row) Selection Heuristics 93

4.3.4.1 Entropy as Row Selection Criterion (EN, en) 93

4.3.4.2 Error Correction as Row Selection Criterion (EC, ec) 94

4.4 Experimental Results 94

4.4.1 Summary Statistics 97

4.4.2 Comparing Attribute (Column) Selectors and Instance (Row) Selectors 100

4.4.3 Comparisons via Algorithms Using Wilcoxon Tests 104

4.5 Conclusions and Future Work 110

5 Budgeted Learning of Bayesian Networks (BLBN) **112**

5.1 Introduction 112

5.2 Related Work 116

5.3 Budgeted Learning of Bayesian Networks 117
5.4 Enhancing Existing Budgeted Learning Algorithms 118
5.4.1 Random (a.k.a. random) 118
5.4.2 Round Robin (a.k.a. RR or rr) 119
5.4.3 Biased Robin (a.k.a. BR or br) 119
5.4.4 Single Feature Look-ahead (a.k.a. SFL or sfl) 120
5.4.5 Randomized Single Feature Look-ahead (a.k.a. RSFL or rsfl) 121
5.4.6 Generalized Single Feature Look-ahead (a.k.a. GSFL or gsfl) 123
5.4.7 Generalized Random Single Feature Look-ahead (a.k.a. GRSFL or grsfl) 124
5.5 Our Algorithms 125
5.5.1 Maximization of Expected Relative Probability Gain (a.k.a. MERPG or merpg) 126
5.5.2 MERPGDSEP (a.k.a. dsep) 127
5.5.3 MERPGDSEPW1 (a.k.a. dsepw1) and MERPGDSEPW2 (a.k.a. dsepw2) 130
5.5.4 Filtering with the Markov blanket 131
5.6 Experimental Results 132
5.6.1 Experimental Results on Animals 137
5.6.2 Experimental Results on Car Diagnosis 2 144
5.6.3 Experimental Results on Chest Clinic 150
5.6.4 Experimental Results on Poya Ganga 157
5.6.5 Experimental Results on ALARM 163
5.6.6 Discussion 168
5.7 Conclusions & Future Work 174

6 Conclusion & Future Work **176**

A The structures of Two Large Bayesian Networks **180**

Bibliography **183**

List of Figures

2.1 A classical Bayesian network ChestClinic from Norsys Corporation Net Library (2011). 16

2.2 A naïve Bayes network for Chest Clinic. 16

2.3 The top figure shows an example of computing expected counts based on incomplete data and current model w. The bottom figure shows an example of iterative process of expectation maximization. 18

2.4 A Bayesian network is shown in (a), the prior marginal probabilities of the variables in the network are shown in (b), and the posterior probabilities of the variables after A is instantiated for a_1 are shown in (c). Each variable has only two values; so the probability of only one is shown in (a). The figures of (a) and (b) come from Neapolitan (2004). 20

2.5 The first two examples of node A and the label becomes d-separated when node V is instantiated. For the third example, node A and the label becomes d-separated when node V and its descendants C, D, E, F, H are NOT instantiated. If any of nodes in $\{V, C, D, E, F, H\}$ is instantiated, A and the label node become connected. 26

2.6 In a Bayesian network, the Markov blanket of node A includes its parents, children and the other parents of all of its children. 27

2.7 An example of active learning: scene labeling problem from Amazon Mechanic Turk website (Amazon Mechanical Turk, 2005). 29

3.1 An example of the relationship among O_ℓ, O_r, O_g, and O_f in Phase 1. The relationship of them are as follows: $O_f \subseteq O_g$ and $O_l \cap O_g = \emptyset$. 43

3.2 Cost required for each algorithm to achieve specific classification accuracies on UCI data sets kr-vs-kp, mushroom, car, and splice. Fifty labelers were available, each with a cost randomly chosen from $\{1, \ldots, 30\}$ and accuracy randomly chosen from $(1/v, 1]$. Values of parameters were $R = 0.99$, $\delta = 0.2$, and $\epsilon = 0.8$. 61

3.3 Cost required for each algorithm to achieve specific classification accuracies on UCI data sets nursery and spambase. Fifty labelers were available, each with a cost randomly chosen from $\{1, \ldots, 30\}$ and accuracy randomly chosen from $(1/v, 1]$. Values of parameters were $R = 0.99$, $\delta = 0.2$, and $\epsilon = 0.8$. 62

3.4 Total cost required for IEAdjCost (with $\lambda \in \{0.02, 0.25, 0.5, 0.75, 1\}$ to achieve specific classification accuracies on the UCI data set splice. Fifty labelers were available, 1 with accuracy 0.85 and cost 100, 49 labelers with accuracy 0.8 and cost 1. Values of parameters were $R = 0.99$, $\delta = 0.2$, and $\epsilon = 0.8$. 64

3.5 Total cost required for wIEAdjCost (with $\lambda \in \{0.02, 0.25, 0.5, 0.75, 1\}$ to achieve specific classification accuracies on the UCI data set splice. Fifty labelers were available, 1 with accuracy 0.85 and cost 100, 49 labelers with accuracy 0.8 and cost 1. Values of parameters were $R = 0.99$, $\delta = 0.2$, and $\epsilon = 0.8$. 64

3.6 Cost required for each algorithm to achieve specific classification accuracies on UCI data sets kr-vs-kp, mushroom, car, and splice. One hundred labelers were available. Each labeler has an accuracy of 0.65 and a cost randomly chosen from $\{1, \ldots, 30\}$. Values of parameters were $R = 0.99$, $\delta = 0.2$, and $\epsilon = 0.8$. 66

3.7 Cost required for each algorithm to achieve specific classification accuracies on UCI data sets nursery and spambase. One hundred labelers were available. Each labeler has an accuracy of 0.65 and a cost randomly chosen from $\{1, \ldots, 30\}$. Values of parameters were $R = 0.99$, $\delta = 0.2$, and $\epsilon = 0.8$. 67

4.1 Learning curves for each data set under the Error Correction (EC) row selector. (a) Breast-cancer; (b) colic-nominalized; (c) mushroom; (d) kr-vs-kp; (e) vote; (f) zoo. 98

4.2 Learning curves for each data set under the Error Correction (EC) row selector. (a) Breast-cancer; (b) colic-nominalized; (c) mushroom; (d) kr-vs-kp; (e) vote; (f) zoo. 99

5.1 A classical Bayesian network ChestClinic from Norsys Corporation Net Library (2011). The label node is "Tuberculosis or Cancer". For one patient, when all feature values are unknown, no nodes are d-separated with the label node. If we instantiate node "Lung Cancer", feature nodes "Smoking" and "Bronchitis" are d-separted with the label node. 129

5.2 Network Animals from Norsys Corporation Net Library (2011). "Animal" is the label node. The values for the nodes are as follows. "Animal" has 5 states: *monkey*, *penguim*, *platypus*, *robin*, and *turtle*. "Has Shell" has 2 states: *true* and *false*. "Bears Young" has 2 states: *live* and *eggs*. "Class" has 3 states: *bird*, *mammal*, and *reptile*. "Environment" has 3 states: *air*, *land*, and *water*. "Warm Blooded" has 2 states: *true* or *false*. "Body Covering" has 3 states: *fur*, *feathers*, and *scales*. 133

5.3 Network Car Diagnosis 2 from Norsys Corporation Net Library (2011). "Car Starts" is the label node. The values for the nodes are as follows. "Alternator" has 2 states: *okay* and *faulty*. "Charing System", has 2 states: *okay* and *faulty*. "Distributer" has 2 states: *okay* and *faulty*. "Main Fuse" has 2 states: *okay* and *blown*. "Battery Age" has 3 states: *new*, *old*, and *very old*. "Battery Voltage" has 3 states: *strong*, *weak*, and *dead*. "Voltage at Plug" has 3 states: *strong*, *weak*, and *none*. "Starter Motor" has 2 states: *okay* and *faulty*. "Spark Plugs" has 3 states: *okay*, *too wide*, and *fouled*. "Spark Timing" has 3 states: *good*, *bad*, and *very bad*. "Starter System" has 2 states: *okay*, and *faulty*. "Headlights" has 3 states: *bright*, *dim*, and *off*. "Spark Quality" has 3 states: *good*, *bad*, and *very bad*. "Fuel System" has 2 states: *okay* and *faulty*. "Car Cranks" has 2 states: *true* and *false*. "Air Filter" has 2 states: *clean* and *dirty*. "Air System" has 2 states: *okay* and *faulty*. "Car Starts" has 2 states: *true* and *false*. 134

5.4 Partial Network of Poya Ganga (a.k.a Water Resource Management). A complete work is shown in Figure A.2. "Rice Yield" is the class label. The values for the nodes are as follows. "Illegal Extractions" has 2 states: *yes* and *no*. "Offtake Condition" has 2 states: *good* and *poor*. "Irrigable Area" has 2 states: *current* and *down 25pc*. "Farmer Control (Inputs)" has 2 states: *yes* and *no*. "Surface Storage" has 2 states: *high* and *low*. "River Extraction (Paddy)" has 3 states: *up 10pc*, *current*, and *down 20pc*. "Input Availability" has 2 states: *when needed* and *not*. "Input Quality" has 2 states: *good* and *poor*. "Paddy Demand Met?" has 2 states: *yes* and *no*. "Rice Yield" has 2 states: *high* and *low*. 135

5.5 Partial Network of ALARM from Norsys Corporation Net Library (2011). A complete network is shown in Figure A.1. "Breathing Pressure" is the label node. The values for the nodes are as follows. "MinVolSet" has 3 states: *low*, *normal*, and *high*. "Disconnection" has 2 states: *true* and *false*. "Vent Machine' has 4 states: *zero*, *low*, *normal*, and *high*. "Vent Tube" has 4 states: *zero*, *low*, *normal*, and *high*. "Pulmary Embolus" has 3 states: *true* and *false*. "Intubation" has 3 states: *normal*, *esophageal*, and *one sided*. "Kinked Tube" has 2 states: *true* and *false*. "Shunt" has 2 states: *normal* and *high*. "Vent Lung" has 4 states: *zero*, *low*, *normal*, and *high*. "Breathing Pressure" has 4 states: *zero*, *low*, *normal*, and *high*. 136

5.6 Comparing all the algorithms on Bayesian network, on Bayesian network with Markov blanket filter, and on naïve Bayes on Animals from Norsys Net Library (2011). 139

5.7 Comparing each algorithm (rr, br, MERPG, dsep, dsepw1, dsepw) on Bayesian network with Markov blanket filter, on Bayesian network, and on naïve Bayes on Animals from Norsys Net Library (2011). 140

5.8 Comparing each algorithm (sfl, rsfl, gsfl, grsfl, Baseline and ranom) on Bayesian network with Markov blanket filter, on Bayesian network, and on naïve Bayes on Animals from Norsys Net Library (2011). 141

5.9 Comparing all the algorithms on Bayesian network, on Bayesian network with a Markov blanket filter, and on naïve Bayes for Car Diagnosis 2 from Norsys Net Library (2011). 146

5.10 Comparing rr, br, merpg, dsep, dsepw1, and dsepw2 on Bayesian network with Markov blanket filter, on Bayesian network, and on naïve Bayes on Car Diagnosis 2 from Norsys Net Library (2011). 147

5.11 Comparing Baseline, random, and sfl on Bayesian network with Markov blanket filter, on Bayesian network, and on naïve Bayes on Car Diagnosis 2 from Norsys Net Library (2011). 149
5.12 Comparing all the algorithms of Bayesian network, of Bayesian network with Markov blanket filter, and of naïve Bayes on Chest Clinic from Norsys Net Library (2011). 152
5.13 Comparing each algorithm (rr, br, merpg, dsep, dsepw1, dsepw2) on Bayesian network with Markov blanket filter, on Bayesian network, and on naïve Bayes on on Chest Clinic from Norsys Net Library (2011). 153
5.14 Comparing each algorithm (sfl, rsfl, gsfl, grsfl, Baseline, and random) on Bayesian network with Markov blanket filter, on Bayesian network with Markov blanket filter, on Bayesian network, and on naïve Bayes on Chest Clinic from Norsys Net Library (2011). 154
5.15 Comparing all algorithms on Bayesian network, on Bayesian network with Markov blanket filter, and on naïve Bayes on Poya Ganga from Norsys Net Library (2011). 159
5.16 Comparing rr, br, merpg, dsep, dsepw1, and dsepw2 on Bayesian network, on Bayesian network with Markov blanket filter, and on naïve Bayes on Poya Ganga from Norsys Net Library (2011). 160
5.17 Comparing Baseline, random and sfl on naïve Bayes, Bayesian network, and Bayesian network with Markov blanket filter on Poya Ganga from Norsys Net Library (2011). 162
5.18 Comparing all the algorithms on Bayesian network, on Bayesian network with Markov blanket, and on naïve Bayes on ALARM from Norsys Net Library (2011). 165
5.19 Comparing rr, br, merpg, dsep, dsepw1, and dsepw2 on Bayesian network with Markov blanket filter, on Bayesian network, and on naïve Bayes on ALARM from Norsys Net Library (2011). 166

5.20 Comparing Baseline, random and sfl on Bayesian network with Markov blanket filter, on Bayesian network, and on naïve Bayes on ALARM from Norsys Net Library (2011). 167

A.1 A complete structure of the Bayesian network of ALARM. The figure is excerpted from Norsys Net Library (Norsys Software Corp., 2011). 181

A.2 A complete structure of the Bayesian network of Water Resource Management in Poya Ganga. The figure is excerpted from Norsys Net Library (Norsys Software Corp., 2011). 182

List of Tables

2.1 An example of budgeted learning: A project was allocated $2 million to develop a diagnosis classifier for patient cancer subtypes. 30

3.1 UCI Data Sets. 58

3.2 The accuracies in Section 3.4.1.1 for 50 labelers whose accuracy were randomly chosen from $(1/v, 1]$ where v is the number of class values. 59

3.3 Classification accuracy after spending a limited budget , using 50 labelers, each with a cost from $\{1, \ldots, 30\}$ and accuracy from $(1/v, 1]$. $R = 0.99$, $\delta = 0.2$ and $\epsilon = 0.8$. **Boldface** indicates a statistically significant advantage (at a 95% confidence level) of wIEAdjCost ($\lambda = 0.02$) over all other algorithms. *Italics* indicates a statistically significant advantage (at a 95% confidence level) of wIEAdjCost ($\lambda = 0.25$), wIEAdjCost ($\lambda = 0.5$), and IEAdjCost ($\lambda = 0.02$) over all other algorithms. 63

3.4 Classification accuracy after spending a limited budget, using one hundred labelers, each with an accuracy 0.65 and a cost from $\{1, \ldots, 30\}$ and accuracy from $(1/v, 1]$. $R = 0.99$, $\delta = 0.2$ and $\epsilon = 0.8$. **Boldface** indicates a statistically significant advantage (at a 95% confidence level) of wIEAdjCostwith specific parameters over all other algorithms. 67

3.5 Savings in cost and run time of wIEAdjCost over IEAdjCost in reaching the baseline accuracy on six UCI data sets. Fifty labelers were available. Each labeler has an accuracy randomly chosen from $(1/v, 1]$ and a cost randomly chosen from $\{1, \ldots, 30\}$. 69

3.6 Savings in cost and run time of wIEAdjCost over IEAdjCost in reaching the baseline accuracy on six UCI data sets. One hundred labelers were available. Each labeler has an accuracy of 0.65 and a cost randomly chosen from $\{1, \ldots, 30\}$. 70

3.7 The size and the labeler accuracies for the AMT data sets. 72

3.8 Training costs and test accuracies on RTE. The savings are the cost savings of IEAdjCost over IEThresh, and wIEAdjCost over IEAdjCost. 74

3.9 Training costs and test accuracies on TEMP. The savings are the cost savings of IEAdjCost over IEThresh, and wIEAdjCost over IEAdjCost. 75

4.1 Data set information. 95

4.2 Algorithm abbreviations, full names, and short descriptions. 96

4.3 Target budget and data utilization rates for algorithms with the EC row selector. Total budget was $B = 100$. 100

4.4 Target budget and data utilization rates for algorithms with the EN row selector. Total budget was $B = 100$. 101

4.5 Target budget and data utilization rates for algorithms with the UR row selector. Total budget was $B = 100$. 101

4.6 Mean accuracies of all algorithms using the EC row selector at the minimum target budget. 102

4.7 Mean accuracies of all algorithms using the EN row selector at the minimum target budget. 103

4.8 Mean accuracies of all algorithms using the UR row selector at the minimum target budget. 103
4.9 Mean accuracies of all algorithms using the EC, EN, and UR column selectors at the minimum target budget. 104
4.10 Mean accuracies of all algorithms using the EC, EN, and UR column selectors at the minimum target budget. 104
4.11 The Wilcoxon-based comparison of two algorithms at the minimum target budget of all the algorithms listed in the table for data sets breast-cancer, colic-Nominalized, kr-vs-kp, mushroom, vote, and zoo. "++0−−+" means that the left side algorithm compared to the top side algorithm, is significantly better, significantly better, no significant difference, significantly worse, significantly worse, and significantly better at $p < 0.05$ level for the 6 data sets. 105
4.12 The Wilcoxon-based comparison of two algorithms at the minimum target budget of all the algorithms listed in the table for data sets breast-cancer, colic-Nominalized, kr-vs-kp, mushroom, vote, and zoo. 105
4.13 The Wilcoxon-based comparison of two algorithms at the minimum target budget of all the algorithms listed in the table for data sets breast-cancer, colic-Nominalized, kr-vs-kp, mushroom, vote, and zoo. 106
4.14 The Wilcoxon-based comparison of two algorithms at the minimum target budget of all the algorithms listed in the table for data sets breast-cancer, colic-Nominalized, kr-vs-kp, mushroom, vote, and zoo. 106
4.15 The Wilcoxon-based comparison of two algorithms at the minimum target budget of all the algorithms listed in the table for data sets breast-cancer, colic-Nominalized, kr-vs-kp, mushroom, vote, and zoo. 106

4.16 The Wilcoxon-based comparison of two algorithms at the minimum target budget of all the algorithms listed in the table for data sets breast-cancer, colic-Nominalized, kr-vs-kp, mushroom, vote, and zoo. 106
4.17 The Wilcoxon-based comparison of two algorithms at the minimum target budget of all the algorithms listed in the table for data sets breast-cancer, colic-Nominalized, kr-vs-kp, mushroom, vote, and zoo. 107
4.18 The Wilcoxon-based comparison of two algorithms at the minimum target budget of all the algorithms listed in the table for data sets breast-cancer, colic-Nominalized, kr-vs-kp, mushroom, vote, and zoo. 107
4.19 The Wilcoxon-based comparison of two algorithms at the minimum target budget of all the algorithms listed in the table for data sets breast-cancer, colic-Nominalized, kr-vs-kp, mushroom, vote, and zoo. 107
4.20 The Wilcoxon-based comparison of two algorithms at the minimum target budget of all the algorithms listed in the table for data sets breast-cancer, colic-Nominalized, kr-vs-kp, mushroom, vote, and zoo. 107
4.21 The Wilcoxon-based comparison of two algorithms at the minimum target budget of all the algorithms listed in the table for data sets breast-cancer, colic-Nominalized, kr-vs-kp, mushroom, vote, and zoo. 107
4.22 The Wilcoxon-based comparison of two algorithms at the minimum target budget of all the algorithms listed in the table for data sets breast-cancer, colic-Nominalized, kr-vs-kp, mushroom, vote, and zoo. 108

5.1 Networks used in our BLBN experiments. 137

5.2 Result of Wilcoxon signed rank test between the left side algorithm and the top side algorithm. "+" ("−") indicates that the left side algorithm reaches a significantly higher (lower) classification accuracy than the top side algorithm at budget 100 for data set **Animals** at a confidence level $p < 0.05$. "0" means the left side algorithm and the top algorithm have no significant difference of the reached classification accuracies at the given budget 100 for data set Animals. . 142

5.3 Mean running time over 10 runs of each algorithm on Animal (7 nodes, 6 edges, 5000 instances). Budget = 100. 143

5.4 Result of Wilcoxon signed rank test between the left side algorithm and the top side algorithm. "+" ("−") indicates that the left side algorithm reaches a significantly higher (lower) classification accuracy than the top side algorithm at budget 100 for data set **Car Diagnosis 2** at a confidence level $p < 0.05$. "0" means the left side algorithm and the top algorithm have no significant difference of the reached classification accuracies at the given budget 100 for data set Car Diagnosis 2. 148

5.5 Mean running time over 10 runs of each algorithm on Car Diagnosis 2 (18 nodes, 20 edges, 5000 instances). Budget=100. 149

5.6 Result of Wilcoxon signed rank test between the left side algorithm and the top side algorithm. "+" ("−") indicates that the left side algorithm reaches a significantly higher (lower) classification accuracy than the top side algorithm at budget 40 for data set **Chest Clinic** at a confidence level $p < 0.05$. "0" means the left side algorithm and the top algorithm have no significant difference of the reached classification accuracies at the given budget 40 for data set Chest Clinic. 155

5.7 Mean running time over 10 runs of each algorithm on Chest Clinic (8 nodes, 8 edges, 5000 instances). Budget = 30. 156

5.8 Result of Wilcoxon signed rank test between the left side algorithm and the top side algorithm. "+" ("−") indicates that the left side algorithm reaches a significantly higher (lower) classification accuracy than the top side algorithm at budget 30 for data set **Poya Ganga** at a confidence level $p < 0.05$. "0" means the left side algorithm and the top algorithm have no significant difference of the reached classification accuracies at the given budget 30 for data set Poya Ganga. 161

5.9 Mean running time over 10 runs of algorithms on data set Poya Ganga (60 nodes, 65 edges, 1000 instances). Budget=40. 162

5.10 Result of Wilcoxon signed rank test between the left side algorithm and the top side algorithm. "+" ("−") indicates that the left side algorithm reaches a significantly higher (lower) classification accuracy than the top side algorithm at budget 100 for data set **ALARM** at a confidence level $p < 0.05$. "0" means the left side algorithm and the top algorithm have no significant difference of the reached classification accuracies at the given budget 100 for data set ALARM. . 164

5.11 Mean running time over 10 runs of algorithms on data set ALARM (37 nodes, 46 edges, 1000 instances). Budget=100. 167

5.12 Result of Wilcoxon signed rank test between the left side algorithm and the top side algorithm. "++0–" ("−") indicates that the left side algorithm reaches reaches a significantly higher classification accuracy than, a significantly higher classification accuracy than, has no significant difference with, a significantly lower classification accuracy than, a significantly lower classification accuracy than the top side algorithm for data sets **Animals, Car Diagnosis 2, Chest Clinic, Poya Ganga,** and **ALARM** at budgets 100, 100, 40, 30,and 100 at a confidence level $p < 0.05$. 169

5.13 Does an algorithm learning **Bayesian networks** get significant improvement over the algorithm learning naive Bayes? "+" indicates a significant improvement, "−" indicates a significant deterioration, "0" indicates no significant change. 170

5.14 Does an algorithm learning Bayesian networks on a **Markov blanket** filter get significant improvement over the algorithm learning Bayesian networks? "+" indicates a significant improvement, "−" indicates a significant deterioration, "0" indicates no significant change. 170

5.15 Does **MERPGDSEP** get significant improvement over MERPG? "+" indicates a significant improvement, "−" indicates a significant deterioration, "0" indicates no significant change. 170

5.16 Does **MERPGDSEPW1** get significant improvement over MERPG? "+" indicates a significant improvement, "−" indicates a significant deterioration, "0" indicates no significant change. 171

5.17 Does **MERPGDSEPW2** get significant improvement over MERPG? "+" indicates a significant improvement, "−" indicates a significant deterioration, "0" indicates no significant change. 171

Chapter 1

Introduction

1.1 Introduction

Machine learning studies the design of computer algorithms that derive general patterns, regularities, and rules from the training data. Our work falls under the category of supervised machine learning, that is, the feature values and the labels of the training data are available for free or for a cost, and the goal of supervised machine learning is to derive a classifier or function or hypothesis based on the training data. The features of the training data can be regarded as a high dimensional *feature vector* in which each component of the vector describes some characteristic of the object. The labels of the training data is a vector of the *labels* of the training data. The training data consists of instances, and each instance has a corresponding set of feature values and a label value. After building a classifier, prediction can be made for a new instance with known feature values and unknown class value. For example, after we build a patient cancer subtype classifier, we can predict a new patient's cancer subtype based on the patient's descriptions (features).

The collection of the training data may be very time-consuming and costly. Sometimes, the features of the training data is easy to get while the labels of the training data need

to be purchased. To save cost, we want to select those instances that are most helpful for building a classifier. Take a scene labeling problem from Amazon Mechanical Turk website (Amazon Mechanical Turk, 2005) for example, the pictures of the scenes can be collected easily, however labeling each scene as natural or man-made scene is tedious work. If we pay an expert to label the instances, we want to carefully choose which scenes to be labeled. In other situations, the output of the training data is known however the feature (attribute) values of the training data are unknown. For example, in one study, a pool of patients with known cancer subtypes were available, as were various diagnosis tests could be performed, each with a diagnostic cost. In this case, we want to carefully choose which patient gets what kind of diagnostic tests.

There are two main types of supervised machine learning to automate the choice of (instance, label) pairs and/or (instance, feature) pairs to purchase. One is *active learning*, and the other is *budgeted learning*. The main difference is that there is no hard budget in active learning while there is a hard budget in budgeted learning.

Many results in *active learning* focus on choosing the instances for labeling and assume that the labeling is handled by a single, noise-free labeler (Tong and Koller, 2000; Roy and Mccallum, 2001; Luo et al., 2004; Neville and Kuwadekar, 2011). However, it is possible that there is some noise in the labeling (Guillory and Bilmes, 2011), and it is possible that there is no perfect labeler and that instead multiple, noisy labelers are available.

For example, when building a speech recognizer, the raw speech samples are easily obtained however labeling the samples is a tedious process in which a human must examine the speech signal and carefully segment it into phonemes. For this speech recognition, it is difficult to guarantee 100% accuracy from a human labeler, due to the wide variations in how we interpret speech signals. However, one may easily have access to multiple human labelers, each with a different cost.

We refer to this model as *active learning with multiple noisy labelers* where the goal is

to learn a hypothesis that generalizes well while spending as little as possible on queries to labelers. Sheng et al. (2008) show that when labelers provide noisy labels under the persistent noise model, one can still estimate well the true labels of instances by requesting labels on a single instance from multiple, independent labelers. The idea is that, if all labeler responses are independent from each other and all noise rates are $< 1/2$ (for binary classification), then a majority vote from an appropriate subset of the labelers can be very effective. (We extend this concept into part of our algorithm: the notion of *combined accuracy* of a group of labelers.) However, their work assumed that all the labelers have the same costs and noise rates. Donmez et al. (2009) relaxed this assumption and assumed that labelers can have different, unknown accuracies. They proposed a method, called IEThresh, for active learning with multiple noisy labelers with different accuracies. Donmez et al. assumed the cost for querying each labeler is the same therefore those labelers with higher estimated accuracies are chosen to label instances. However, it is reasonable that sometimes higher accuracy labelers will ask for higher pay while lower accuracy labelers will ask for lower pay. We propose algorithms to handle the setting of *active learning with multiple noisy labelers with varied costs*. The intuition behind our algorithms is that we "normalize" the accuracies of labelers with respect to their costs, allowing for direct comparison between labelers. Then, rather than simply identifying the most accurate labelers, our algorithms instead seek a subset of labelers that, when combined, achieve the desired level of accuracy for low cost. Our experiment results show that our algorithms outperform existing algorithms in the literature.

In contrast to the setting that the labels of the instances have to be purchased at a cost, there is a line of study for the setting that the labels of the instances are known but the feature values have to be purchased, subject to an overall budget. Several results in the name of "active feature acquisition" (Zheng and Padmanabhan, 2002; Melville et al., 2004; Melville et al.; Lomasky et al., 2007; Saar-Tsechansky et al., 2009) exist in under the active

learning setting where the labels of the instances are known but the feature values have to be purchased. The difference between active feature acquisition and budgeted learning is that budgeted learning usually has a hard budget set up front, while active feature acquisition aims to improve classifier accuracy at any intermediate investment. Another minor distinction is that, in some applications of active feature acquisition (Zheng and Padmanabhan, 2002; Lomasky et al., 2007), individual instance/feature values cannot be bought one at a time. Instead, all the missing attributes of an instance must be synthesized or obtained as a whole.

Existing budgeted learning results under the setting in which the labels of the instances are known but the feature values have to be purchased have been done by Madani et al. (2004) and Kapoor and Greiner (2005c). Madani et al. (2004) proposed Biased Robin as a novel approach to be used in this setting. Kapoor and Greiner (Kapoor and Greiner, 2005c) proposed RSFL to for this setting. Their experiment results show that BR and RSFL are state-of-the-art approaches for budgeted learning. We proposed several new approaches for budgeted learning in which the labels of the instances have to be purchased and a tight budget is given, including algorithms adapted from the "multi-arm bandit" model (Auer et al., 2002b; Kalai and Vempala, 2005) and several algorithms that incorporate Biased Robin and second order statistics. We also proposed several instance selectors that can be combined with the algorithms. Most of our proposed algorithms perform well with existing algorithms and the proposed instance selectors work well for some algorithms.

Many existing approaches to budgeted learning assume the complete independence of the features (attributes) of the data set. However, in reality, it is possible that the features are not independent of each other. One feature (attribute) of an instance can have a direct influence on another feature (attribute) of that instance. For example, whether a patient is smoking or not has a direct influence on the probability that the patient has lung cancer; also the presence or absence of lung cancer has a direct influence on the probability that the patient's X-ray is positive. The probability relationships of the features can be represented

by a Bayesian network, which can be expressed by a structure (an acyclic graph that models the Markov condition) and a distribution (conditional probability tables for the variables in the network). Tong and Koller (2001a) studied learning the distribution of Bayesian networks under the active learning setting in which the feature values of the instances are known but the labels of the instances have to be purchased. Their approaches use the (instance, feature) pair that minimizes the expected loss of the distribution of the Bayesian network. Recently, Li et al. (2010) studied budgeted distribution learning of Bayesian networks under the setting that both the labels and the features are unknown. The goal of their work is also to learn the parameters of the Bayesian network. Li et al. adopted the objective function for the choice of (instance, feature) pair or (instance, label) that was used in Tong and Koller (2001a) for their budgeted distribution learning.

We studied the budgeted learning of Bayesian networks in which the labels of the instances are known but the feature values have to be purchased. Therefore, besides taking advantage of the learned distribution to do probability inference, we also took advantage of the given labels to compute the choice of (instance, feature) pair that maximizes relative expected probability gain. We also took advantage of the structure of the Bayesian network itself, and proposed methods that choose those (instance, feature) pairs whose features are nearby features of the label node. Why? Because the instantiation of the Markov blanket nodes make the remaining nodes independent from the label node. Some of our proposed algorithms also make the choice of (instance, feature) pairs by considering the factor of the number of increased feature nodes that are independent from the label node for the instance. Our experiment results on 5 data sets from Norsys Net library (2011) show that most of most of our approaches outperform existing approaches.

The goal of this dissertation is to bring more insight into the above thee topics under the common theme of *machine learning with incomplete information*. When the features/labels have to be purchased and multiple noisy labelers exist with different costs and accuracies,

which labelers should be selected for labeling the features/labels? Can we have better algorithms for budgeted learning besides existing algorithms in the literature? Can we have better algorithms for budgeted learning of Bayesian network besides existing algorithms that can be adapted to Bayesian networks? We answer these 3 questions in this dissertation.

Chapter 2 introduces the framework of our work and provides some common background information related to our work, including naïve Bayes and Bayesian networks, learning the distribution of the Bayesian network, doing probability inference in Bayesian networks, identifying d-separation in Bayesian network, finding a Markov blanket of a node in Bayesian networks, and examples of our work.

In Chapter 3, we present results on active learning on multiple noisy labelers with varied costs. We examine two existing algorithms, naïve Repeated and IEThresh (Donmez et al. (2009)). We also present our own active learning algorithms designed to handle multiple noisy labelers with varied costs by ranking each labeler according to its *adjusted* cost. Our first algorithm, IEAdjCost, was consistently the top performer, i.e., the first algorithm to reach the baseline. Our second algorithm, wIEAdjCost, assigns different labelers different weights to estimate the ground truth of the true label. The labeler with higher accuracy has a higher weight. Also, in our second algorithm, we adapted our algorithms to handle data sets with multiple class values. Experiment results show that our algorithm IEAdjCost outperforms existing algorithms and another of our algorithm wIEAdjCost outperforms IEAdjCost. Preliminary results of this work appeared in Zheng et al. (2010), and more comprehensive results of this work were submitted to *Machine Learning*.

In Chapter 4, we present new algorithms for choosing which features of which instances to purchase in the budgeted learning model. We examine three existing algorithms, Random, BR (Madani et al., 2004), and RSFL (Kapoor and Greiner, 2005c). We also present our own budgeted learning algorithms. Several of our algorithms were based on results in the "multi-armed bandit" model, in which there are n slot machines and a player can play

for a fixed number of rounds. At each round, one must decide which single slot machine to play to maximize total reward over all rounds. Our first two algorithms were based on the algorithm Exp3 of Auer et al. (Auer et al., 2002b), and our third algorithm, FEL, was based on the "follow the perturbed leader" approach of Kalai and Vempala (2005). We also proposed three other algorithms, ABR2, WBR2, and RBR2, which incorporate second-order statistic into decision making for biased robin. In addition, we present new heuristics to decide which example to purchase after the attribute is selected, instead of picking an instance uniformly at random, which is typically done in existing budgeted learning algorithms. One proposed instance selector is "Error-Correction" (EC), which chooses the instance that are most wrongly predicted. Another proposed instance selector is "Entropy" (EN), which chooses the instance that maximizes the entropy. On the six data sets we tested, we found that EC row selector works well for Random, and EN row selector works well for RBR2. When comparing algorithms with same row selector, ABR2 was one of the top 2 algorithms we tested. When comparing all algorithms, ABR2 with any row selector performs well, especially with EN row selectors. Also performing well are WBR2 and Exp3C with EC row selector and FEL with uniform random row selector. Preliminary results of this work appeared in Deng et al. (2007), and more comprehensive results have been accepted by *Machine Learning* pending revisions.

In Chapter 5, we present our new algorithms and adaptations of existing algorithms to learn Bayesian networks under the setting that the labels of the instances are known however the attributes of the instances have to be purchased, subject to an overall budget. Existing budgeted learning or active learning algorithms of Bayesian network take advantage of the learned distribution of the Bayesian network and choose the (instance, feature) pair that minimizes the loss to be the new distribution after purchasing this (instance, feature) pair. We proposed algorithms that not only take advantage of the learned distribution of the Bayesian network, but also take advantage of the known labels, as well as the structure

of the Bayesian network. Our first proposed algorithm, Maximization of Expected Relative Probability Gain (MERPG), takes advantage of the known labels and the learned distribution for the choice of (instance, feature) pair that maximizes the relative increase of the expected probability of predicting this instance to be its true label. Let NumIncreaseDseps value of an (instance, feature) be the number of increased independent features from the label node for the instance. Our second proposed algorithm, MERPGDSEP, breaks ties of MERPG by choosing the (instance, feature) pair that maximizes the NumIncreaseDseps value. Our third and fourth algorithms make NumIncreaseDseps value a weighting factor combined with MERGP. We also propose to choose only those (instance, feature) pairs whose feature is a nearby (label node's parent, or children, or spouse) feature of the label node to purchase. We call this idea "Markov blanket filter". Our preliminary results on 5 data sets show that learning a Bayesian network outperforms learning a naïve Bayes classifier, MERPGDSEP improves unpon MERPG, and Markov blanket filter does help.

Although the ideas and solutions in Chapters 3, 4 and 5 are quite different, are presented together in the umbrella of "machine learning of incomplete information". All these approaches assume the existence of efficient underlying base learners that can deal relative small amounts of complete information. All these topics are related to choosing which (instance, feature) pair or (instance, label) pair to purchase. Also, there are some common techniques to solve these problems. For example, our entropy based instance selection in budgeted learning was based on a popular active learning heuristic. The algorithms of budgeted learning used in Chapter 4 can be adapted to algorithms that can be used for the setting in Chapter 5.

In Chapter 6, we summarize our work and present directions of future work.

Chapter 2

Background

This chapter reviews the basic ideas of machine learning, including the naïve Bayes classifier, Bayesian networks, machine learning with incomplete information, active learning, and budgeted learning.

2.1 Supervised Machine Learning

Machine learning studies the design of computer algorithms that derive general patterns, regularities, and rules from training data. Given labeled training data, for example, cancer patient descriptions with examination results, the machine learning algorithm generates a patient cancer subtype classifier. The built classifier can predict a cancer subtype for any new cancer patient description.

Machine learning algorithms have a wide range of applications. Besides medical diagnosis (Kononenko, 2001), they are also applicable to news article topic spotting (Jo et al., 2000), which is to categorize the news articles into different subjects; email spam filtering (Tretyakov, 2004), which is to identify a given email as a spam or non-spam; face detection in image analysis (Sung and Poggio, 1998), which is to identify the presence of face in

an image; and lots more.

Supervised machine learning is a learning model in which an algorithm infers a function from *supervised* training data. The training data consists of a set of training examples. In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal). A supervised learning algorithm analyzes the training data and produces an inferred function, which is called a classifier (if the output is discrete) or a regression function (if the output is continuous). The inferred function should predict the correct output value for any valid input object. This requires the learning algorithm to generalize from the training data to unseen situations in a "reasonable" way. The work in this dissertation falls under the category of supervised learning; specifically, supervised classification.

In a typical supervised classification problem, the learning algorithm is presented with a training sample $S = \{(x_1, y_1), \ldots, (x_m, y_m)\} \in (\mathcal{X} \times \mathcal{Y})^m$. Each x_i is called an example or *pattern* of the domain $\mathcal{X}$, and y_i is called the *label* of x_i. Each x_i is frequently represented as a vector $(a_{i_1}, a_{i_2}, \ldots, a_{i_k})$, with each component a_{i_j} describing the value of some attribute or feature of x_i. Thus x_i is also referred to as a *feature vector*, and $\mathcal{X}$ is called a *feature space*. The label space $\mathcal{Y}$ is a finite set of discrete values $\mathcal{Y} = \{b_1, b_2, \ldots, b_n\}$ that serve to categorize the patterns in $\mathcal{X}$. If a classification problem has only 2 possible label values, e.g. $\mathcal{Y} = \{\text{good}, \text{bad}\}$ or $\mathcal{Y} = \{+, -\}$, we refer to it as a *binary* problem, otherwise we say it's a *multi-class* problem. It's desirable sometimes to further assume that training examples are independent and identically distributed (IID) according to a certain probability distribution $\mathrm{P}(\cdot)$ over $\mathcal{X} \times \mathcal{Y}$. A learning algorithm A is trained on S, resulting in a hypothesis $h : \mathcal{X} \to \mathcal{Y}$, which can be used to predict labels on future examples drawn according to the distribution $\mathrm{P}(\mathcal{X})$.

2.2 Examples of Classifiers

In the following, we introduce two examples of classifiers used in this dissertation. One is the naïve Bayes classifier, and the other is the Bayesian network classifier. The naïve Bayes classifier is a special case of the Bayesian network classifier.

2.2.1 Naïve Bayes Classifier

A naïve Bayes classifier is a simple probabilistic classifier based on applying Bayes' theorem with strong (naïve) independence assumptions. In simple terms, a naïve Bayes classifier assumes that the value of a particular feature of a class is unrelated to the value of any other feature given the value of the class label. Even if these features depend on each other, a naïve Bayes classifier considers all of these properties to independently contribute to the probability that this sample belongs to a class. For example, a patient who smokes has a much higher probability to have lung cancer than a patient who does not. However, a naïve Bayes classifier that has smoking and lung cancer as features considers these two properties independently contribute to the probability that this patient has a "Dyspnea". Another example is news articles topic spotting. The class label can be "sports", "music", or "politics" or other categories. The attributes of this example is appearance of some words. A naive Bayes classifier assume that the words are independent of each other. However, some words frequently appear simultaneously, e.g. "basketball" and "score".

2.2.1.1 Bayes' Theorem

A naïve Bayes classifier is a simple probabilistic classifier based on Bayes' Theorem, which states the posterior probability is in proportion to the conditional and the prior probabilities of two events A, B:

$$\mathrm{P}(A \mid B) = \frac{\mathrm{P}(B \mid A)\,\mathrm{P}(A)}{\mathrm{P}(B)}\,. \tag{2.1}$$

If we let B be the event that an example x has attributes $(a_1, \ldots, a_k)$ and A be the event that the label of x is y, Bayes' theorem gives us a way to estimate the probability of an example's label given its attributes:

$$\mathrm{P}(y \mid a_1, \ldots, a_k) = \frac{\mathrm{P}(a_1, \ldots, a_k \mid y)\,\mathrm{P}(y)}{\mathrm{P}(x)}\,. \tag{2.2}$$

A fundamental assumption of a naïve Bayes classifier is that feature values are independent given the label of an example. For the purpose of classification, we would like to predict the class to be the most probable label y^* given an unlabeled example x. Since we assumed that all attributes of an example are independent given its label, we can estimate the probability of seeing attributes $(a_1, \ldots, a_k)$ given the label is y by taking a simple product:

$$\mathrm{P}(a_1, a_2, \ldots, a_k \mid y) = \prod_i \mathrm{P}(a_i \mid y)\ . \tag{2.3}$$

After substituting Equation 2.3 into Equation 2.2, we obtain the prediction rule of naïve Bayes:

$$y^* = \operatorname*{argmax}_{y \in \mathcal{Y}} \mathrm{P}(y) \prod_i \mathrm{P}(a_i \mid y)\quad . \tag{2.4}$$

In this equation, $\mathrm{P}(y)$ and $\mathrm{P}(a_i \mid y)$ can be estimated from a training set by simple frequency counting. Also note that we omitted $\mathrm{P}(x)$ in Equation 2.4, as being a constant for a given example x, it does not affect the outcome of prediction.

Depending on the precise nature of the probability model, naïve Bayes classifiers can be trained very efficiently in a supervised learning setting. In many practical applications, parameter estimation for naïve Bayes models uses the method of maximum likelihood. Naive

Bayes classifiers assume that the features (attributes) of the training data are independent from each other. This fundamental assumption in a naive Bayes classifier is often deemed rather strong and unlikely to satisfied in most applications. In spite of their naïve design and apparently over-simplified assumptions, naïve Bayes classifiers have worked quite well in many complex real-world situations, such as news articles topic spotting, spam filters and text classification.

In Zhang (2004), analysis of the Bayesian classification problem has shown that there are some theoretical reasons for the apparently unreasonable efficacy of naïve Bayes classifiers. Still, a comprehensive comparison with other classification methods in Caruana and Niculescu (2006) showed that Bayes classification is outperformed by more current approaches, such as boosted trees or random forests.

An advantage of the naïve Bayes classifier is that it only requires a small amount of training data to estimate the parameters (means and variances of the variables) necessary for classification. Therefore we can use naïve Bayes classifier as the base learner in budgeted learning in which few (instance, feature) pairs are purchased. Because independent variables are assumed, only the variances of the variables for each class need to be determined and not the entire covariance matrix. The disadvantage of the naïve Bayes is that it is not good for estimating probabilities (Neapolitan, 2004).

2.2.2 Bayesian Network Classifier

In this section, we first introduce Bayesian networks in Section 2.2.2.1, we then explain Bayesian network learning in Section 2.2.2.2. After that, we illustrate probability inference in Bayesian network in Section 2.2.2.3. and then look at two important notions in Bayesian network in the background of machine learning, d-separation in Section 2.2.2.4 and Markov Blanket in Section 2.2.2.5.

2.2.2.1 Bayesian Networks

Formally, a Bayesian network $N = (G, P)$ where $G = (V, E)$ for a set of variables $V = \{V_1, \ldots, V_m\}$ consists of (1) a network structure G that encodes a set of conditional independence assertions about variables in V and (2) a set P of local probability distributions associated with each variable. Together, these components define the joint probability distribution for V. The network structure G is a directed acyclic graph. The nodes in G are in one-to-one correspondence with the variables V. We use V_i to denote both the variable and its corresponding node, and Pa_i to denote the parents of node V_i in G as well as the variables corresponding to those parents. The lack of possible arcs in G encode conditional independences. Given structure G, the joint probability distribution for N is given by

$$P(V) = \prod_{i=1}^{m} P(V_i \mid \mathrm{Pa}_i) \tag{2.5}$$

The pair (G, P) encodes the joint distribution $P(V)$.

In a Bayesian network, a link from variable A (the parent) to variable B (the child) indicates that A causes B, that A partially causes or predisposes B, that B is an imperfect observation of A, that A and B are functionally related, and/or that A and B are statistically correlated.

A Bayesian network $N = (G, P)$ satisfies the Markov condition. That is, for each variable in V_i in V, $\{V_i\}$ is conditionally independent of all its non-descendants given its parents:

$$I_p(V_i, ND_{V_i} \mid \mathrm{Pa}_i) \tag{2.6}$$

where ND_{V_i} represents non-descendants of V_i. I_p means independence.

A classical example of the use of Bayesian networks is in the medical domain. Here each new patient typically corresponds to a new case, and the problem is to diagnose the

patient (i.e. find beliefs for the undetectable disease variables), or predict what is going to happen to the patient, or find an optimal prescription, given the values of observable variables (symptoms). A doctor may be the expert who defined the structure of the net, and provided the initial conditional probabilities, based on his medical training and experience with previous cases. Then the net probabilities may be fine-tuned by using statistics from previous cases, and from new cases as they arrive.

Figure 2.1 shows a classical Bayesian network Chest Clinic from Norsys net library (Norsys Software Corp., 2011). A naïve Bayesian network can be viewed as a Bayesian network with a simple structure that has the label node as the parent node of all the feature nodes and there are no other links. When the label node is instantiated, all the features nodes are independent with each other. Figure 2.2 shows a naïve Bayes network for Chest Clinic. Based on the Markov condition (a variable in a Bayesian network is conditionally dependent from its non-descendants given its parents), every feature node in Figure 2.2 is conditionally dependent from any other feature node given the label node. Therefore, Figure 2.2 is a naïve Bayes network for Chest Clinic.

In a Bayesian network, an arrow from one node to the other node means a casual relationship. Each node has an associated conditional probability table in which every line indicates the probability of this node to be a specified value given its parent nodes' values. In Figure 2.1, the associated conditional probability table (CPT) of feature "Bronchitis" tells us that if a patient does not smoke, the probability that this patient has bronchitis is 0.01; however if a patient smokes, the probability that this patient has bronchitis is 0.1. For node "Bronchitis", given its parent "Smoking", is conditionally independent of all its non-descendants, including the following nodes: "Visit to Asia", "Tuberculosis", "Lung Cancer", "Tuberculosis or Cancer", and "X-Ray Result".

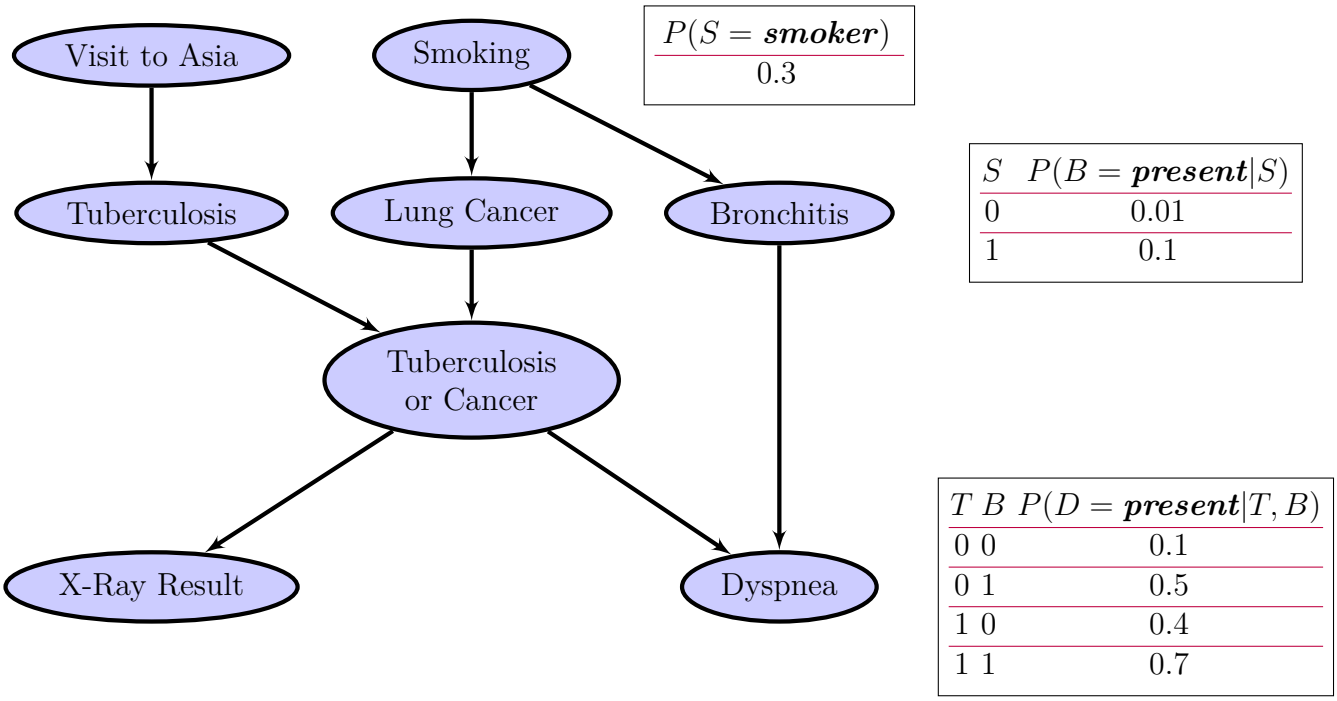


Figure 2.1: A classical Bayesian network ChestClinic from Norsys Corporation Net Library (2011).

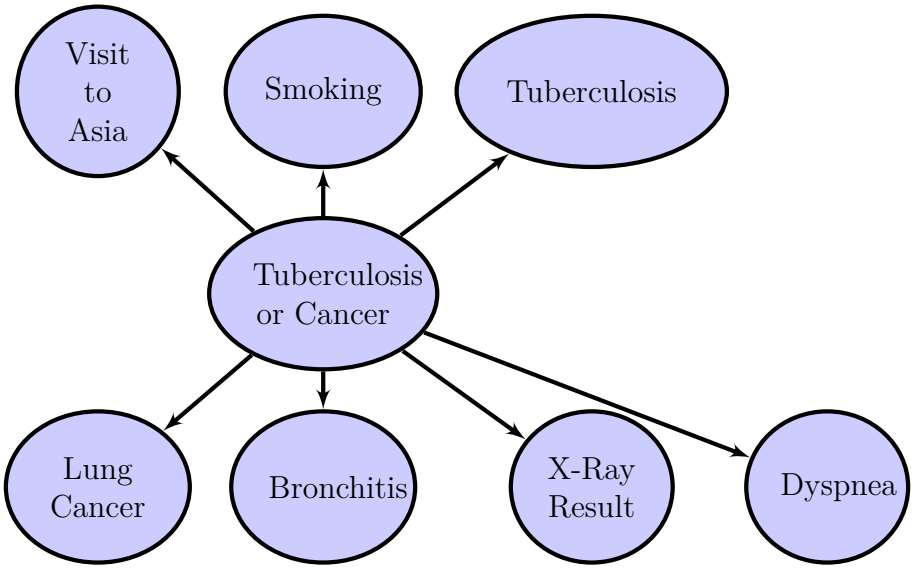


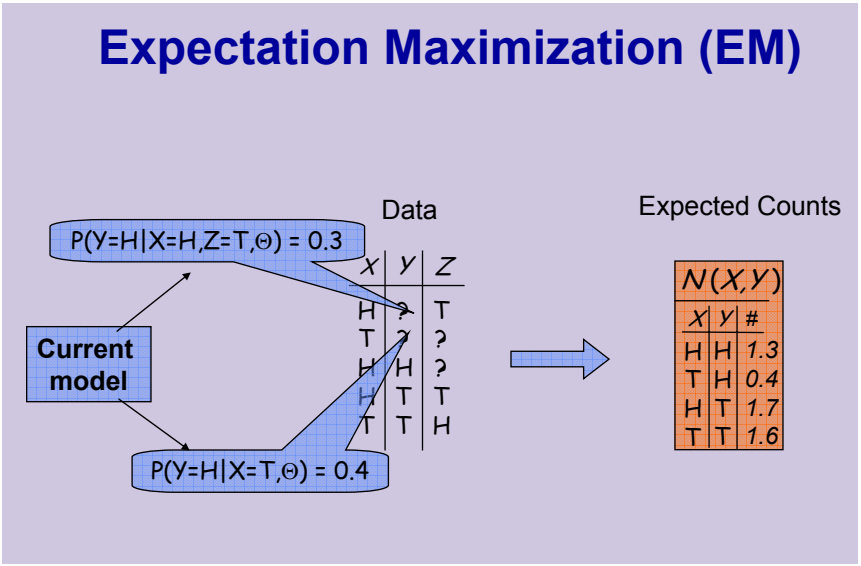
Figure 2.2: A naïve Bayes network for Chest Clinic.

2.2.2.2 Bayesian Network Learning

Bayesian Network Learning has been traditionally divided into two parts: structure learning and parameter learning. *Structure learning* determines the dependence and independence of sets of variables and suggests a direction of a causation, in other words, the placement of the links in the net. *Parameter learning* determines the conditional probability table (CPT) at each node, given the link structures, prior probabilities and the data. In our study, we assume the correct structures of the Bayesian networks are given, either from a domain expert or learned from previous data or a combination of two. Therefore, we only learn their parameters.

There are three main types of algorithms to learn CPTs: counting, expectation maximization (EM) and gradient descent (Korb and Nicholson, 2004; Russell and Norvig, 1995; Neapolitan, 2004). Of the three, "counting" is by far the fastest and simplest. It can be used whenever there is not much missing data or uncertain findings. It counts the number of occurrences of values given its parents values. The probability of a node to be a value given its parents will be the relative frequency of the occurrences of this value. However, in our studies, with relatively little information of the training data, we adopt an method called EM (a E-step and a M-step) which is an approach good for optimizing likelihood functions in the presence of incomplete data. The intuition for EM is as follows. If we have complete data, we will have true counts and we could estimate parameters. However, with missing data, counts are unknown. We "complete" counts using probabilistic inference based on the current parameter assignment. We then use complete counts as if they are real to re-estimate parameters. Figure 2.3 shows an example of iterative process of expectation maximization, and also shows an example of computing expected counts based on incomplete data and current model.

Let the current model to be w. Computation of expected counts in the E-step is as



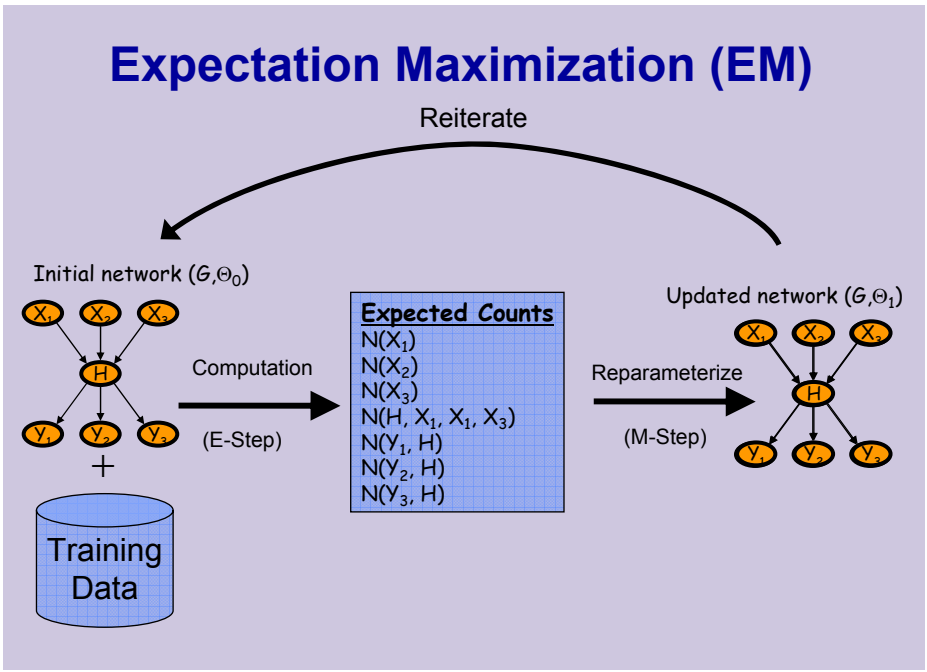


Figure 2.3: The top figure shows an example of computing expected counts based on incomplete data and current model w. The bottom figure shows an example of iterative process of expectation maximization.

follows:

$$\bar{N}(x_i, Pa_i) = \sum_m P(x_i, Pa_i | x_m, w) \tag{2.7}$$

Then in the M-step, we recompute the parameters of the Bayesian network based on the counts. Then, based on the updated parameters and the incomplete training data, we compute the expected counts again, and so on. The EM iterative process terminates until there is no obvious change of the parameters of the Bayesian network.

Since for our budgeted learning work, we only know a few (instance, feature) pairs by purchasing, there are much missing data, therefore "counting" is not a good algorithm for our setting. For the other two algorithms, generally speaking, EM learning is more robust (i.e. gives good result in wide variety of situations), but sometimes gradient descent is faster. We chose the EM algorithm to be used in our work because it is good at handling missing data and it is more robust than gradient descent. We used the EM algorithm implemented by Norsys Software Corporation (2011) in their Netica API.

2.2.2.3 Inference in Bayesian Networks

Given a Bayesian network, including its structure and its CPTs, we can do probability inference to derive the probability distribution over one or more attribute value given the value of one or more other attributes. For example, if we know a patient who a positive X-ray and is not smoking, we can compute the probability that he has lung cancer. In the following we show an example of probability inference.

Consider the Bayesian network in Figure 2.4 (a) (Neapolitan, 2004). The prior probabil-

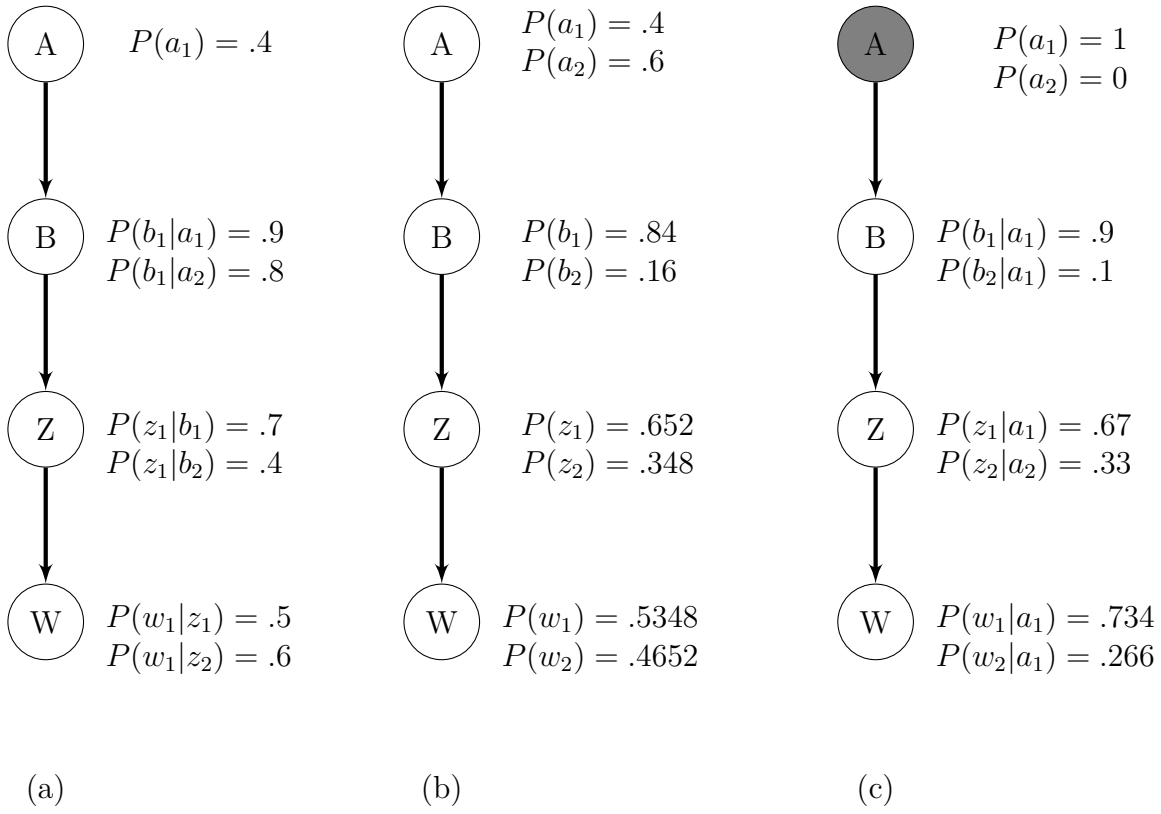


Figure 2.4: A Bayesian network is shown in (a), the prior marginal probabilities of the variables in the network are shown in (b), and the posterior probabilities of the variables after A is instantiated for a_1 are shown in (c). Each variable has only two values; so the probability of only one is shown in (a). The figures of (a) and (b) come from Neapolitan (2004).

ities of all variables can be computed as follows:

$$\begin{aligned} P(b_1) &= P(b_1|a_1)P(a_1) + P(b_1|a_2)P(a_2) \\ &= (.9)(.4) + (.8)(.6) \\ &= 0.84 \end{aligned}$$

$$\begin{aligned} P(z_1) &= P(z_1|b_1)P(b_1) + P(z_1|b_2)P(b_2) \\ &= (.7)(.84) + (.4)(.16) \\ &= 0.652 \end{aligned}$$

$$\begin{aligned} P(w_1) &= P(w_1|z_1)P(z_1) + P(w_1|z_2)P(z_2) \\ &= (.5)(.652) + (.6)(.348) \\ &= 0.5348 \end{aligned}$$

These probabilities are shown in Figure 3.2 (b). The computation for each variable required information determined for its parent. Therefore the inference can be regarded as a process of message passing in which each node passes to its child a message needed to compute child's probabilities.

Given a new instance whose value of node A is a_1, how do we compute the conditional probabilities of the remaining variables B, Z and W? By the Markov condition, we know that W is conditionally independent of B given its parent Z. Therefore, we can compute their conditional probabilities as follows:

$$
\begin{aligned}
P(b_1|a_1) &= 0.9 \\
P(z_1|a_1) &= P(z_1|b_1,a_1)P(b_1|a_1) + P(z_1|b_2,a_1)P(b_2|a_1) \\
&= P(z_1|b_1)P(b_1|a_1) + P(z_1|b_2)p(b_2|a_1) \\
&= (.7)(.9) + (.4)(.1) \\
&= 0.67
\end{aligned}
$$

$$
\begin{aligned}
P(w_1|a_1) &= P(w_1|z_1,a_1)P(z_1|a_1) + P(w_1|z_2,a_1)P(z_2|a_1) \\
&= P(w_1|z_1)P(z_1|a_1) + P(w_1|z_2)p(z_2|a_1) \\
&= (.8)(.67) + (.6)(.33) \\
&= 0.734
\end{aligned}
$$

The above example shows that when we know the value of A, how we do probability inference to find the probability of its descendants. What if we know the value of W as w_1, how do we do probability inference to find the probability of its ancestors? The solution is to use Bayes' theorem to compute the probability of Z.

$$P(z_1|w_1) = \frac{P(w_1|z_1)P(z_1)}{P(w_1)} \tag{2.8}$$

When implementing our algorithms, we used Netica API provided by Norsys Software Corp. (2011) to do probability inference. Netica uses message passing in a junction tree (or "join tree") of cliques (Lauritzen and Spiegelhalter, 1988; Cowell et al., 1999; Spiegelhalter

et al., 1993) for exact general probabilistic inference in a compiled Bayesian network.

In this process the Bayesian network is first "compiled" into a junction tree. You enter findings for one or more nodes of the original Bayesian network, and then when you want to know the resultant beliefs for some of the other nodes, belief updating is done by a message-passing algorithm operating on the underlying junction tree. It determines the resultant beliefs for each of the nodes of the original Bayesian network, which it attaches to the nodes so that you can retrieve them. You may then enter some more findings (to be added to the first), or remove some findings, and when you request the resultant beliefs, belief updating will be performed again to take the new findings into account.

The amount of memory required by the junction tree, and the speed of belief updating are approximately proportional to each other, and are determined by the elimination order used, which is a list of all the nodes in the net. Any order of the nodes will produce a successful compilation, but some do a much better job than others. As far as the complexity is concerned, inference in Bayesian Networks is NP-hard (Cooper, 1987).

2.2.2.4 d-separation

The "d" in d-separation stands for dependence. To explain d-separation, we need first explain the idea of a chain. Let A, Z, and B be nodes in an acyclic graph $G = (V, E)$. We call the set of edges connecting the nodes a chain. Given an edge, $V_1 \rightarrow V_2$, we say the **tail** of the edge is at V_1 and the **head** of the edge is at V_2 (Neapolitan, 2004).

- A chain $A \rightarrow Z \rightarrow B$ is a **head-to-tail meeting**, the edges meet head-to-tail at Z, and Z is a head-to-tail node on the chain.

- A chain $A \leftarrow Z \rightarrow B$ is a **tail-to-tail meeting**, the edges meet tail-to-tail at Z, and Z is a tail-to-tail node on the chain.

- A chain $A \rightarrow Z \leftarrow B$ is a **head-to-head meeting**, the edges meet head-to-head at Z, and Z is a head-to-head node on the chain.

The first is a directed chain from A to B through Z, the second is a pair of directed chains from Z to A and from Z to B, and the third is a pair of directed chains from A to Z and B to Z. If we interpret these chains causally, in the first case A is an indirect cause of B, in the second case Z is a common cause of A and B, and in the third case A and B have a common effect in Z.

In the following, we explain the situations in which a chain is blocked. Let $G = (V, E)$ be a directed acyclic graph, $V' \subseteq V$, and A, B be distinct nodes in $V - V'$. Then a chain ρ is **blocked** by V' if one of the following holds:

1. There is a node $Z \in V'$ on the chain ρ, and the edges incident to Z on ρ meet head-to-tail at Z.

2. There is a node $Z \in V'$ on the chain ρ, and the edges incident to Z on ρ meet tail-to-tail at Z.

3. There is a node Z, such that Z and all of Z's descendants are not in V', on the chain ρ, and the edges incident to Z on ρ meet head-to-head at Z.

Based on the above definitions, we now define d-separation as follows. Let $G = (V, E)$ be a directed acyclic graph, $V' \subseteq V$, A, B be distinct nodes in $V - V'$. A and B are **d-separated** by V' in G if every chain between A and B is blocked by V'.

Now, we know the definition of d-separation in a directed acyclic graph setting. Now let us take a look at the meaning of d-separation in a Bayesian network setting. Generally speaking, in a Bayesian network, two nodes are d-separated if they are independent of each other. In a Bayesian network, evidence may be transmitted through a chain, unless the state of the variable in the connection is known. For example, for chain $A \rightarrow Z \rightarrow B$ or

$A \leftarrow Z \rightarrow B$, when Z is given, A and B are d-separated. That is, when Z is instantiated it blocks the communication between A and B. For chain $A \rightarrow Z \leftarrow B$, it is completely different. Consider an example given by Pearl (1988) in which there are two independent causes of your car refusing to start: having no gas and having a dead battery.

$$\text{dead battery} \rightarrow \text{car won't start} \leftarrow \text{no gas} \tag{2.9}$$

Telling you that the battery is charged tells you nothing about whether there is gas, but telling you that the battery is charged after I have told you that the car won't start tells me that the gas tank must be empty. So independent causes are made dependent by conditioning on a common effect. Therefore, when the node "car won't start" is instantiated, the nodes "dead battery" and "no gas" become dependent with each other. That is, for the chain $A \rightarrow Z \leftarrow B$, when Z is instantiated, A and B become dependent; when Z is not instantiated, A and B are d-separated.

Figure 2.5 shows the examples in three cases, nodes A and the label node become d-separated from each other when V is instantiated in the first two cases and not instantiated in the third case. When node A becomes d-separated from the label node, instantiating it in the future will not affect the label node.

2.2.2.5 Markov Blanket

In a Bayesian network $N = (G, P)$ where $G = (V, E)$ is an acyclic graph and P is the joint probability distribution, and $V_i \in V$. Then a Markov blanket M_{V_i} of V_i is any set of variables such that V_i is conditionally independent of all the other variables given M_{V_i}). That is

$$I_P(\{V_i\}, V - (M_{V_i} \cup \{X\}) \mid M_{V_i}) \tag{2.10}$$

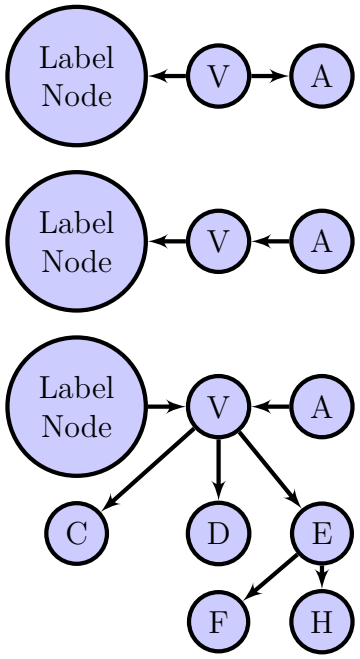


Figure 2.5: The first two examples of node A and the label becomes d-separated when node V is instantiated. For the third example, node A and the label becomes d-separated when node V and its descendants C, D, E, F, H are NOT instantiated. If any of nodes in $\{V, C, D, E, F, H\}$ is instantiated, A and the label node become connected.

where I_P means "conditionally independent over P". Suppose (S, P) satisfies the Markov condition (i.e. every variable is conditionally independent from its non-descendants given its parents). Then for each variable V_i, the set of all parents of V_i, children of V_i, and spouses (parents of children of V_i) is a Markov blanket of V_i.

The Markov blanket of a node contains all the variables that shield the node from the rest of the network. This means that the Markov blanket of a node is sufficient to predict the behavior of that node. The term was coined by Pearl (1988). In a Bayesian network, the values of the parents and children of a node evidently give information about that node; however, its children's parents also have to be included, because they can be used to explain away the node in question. Figure 2.6 shows a Markov blanket (the nodes except A in the blue background) for node A. We find that the Markov blanket for the label node is

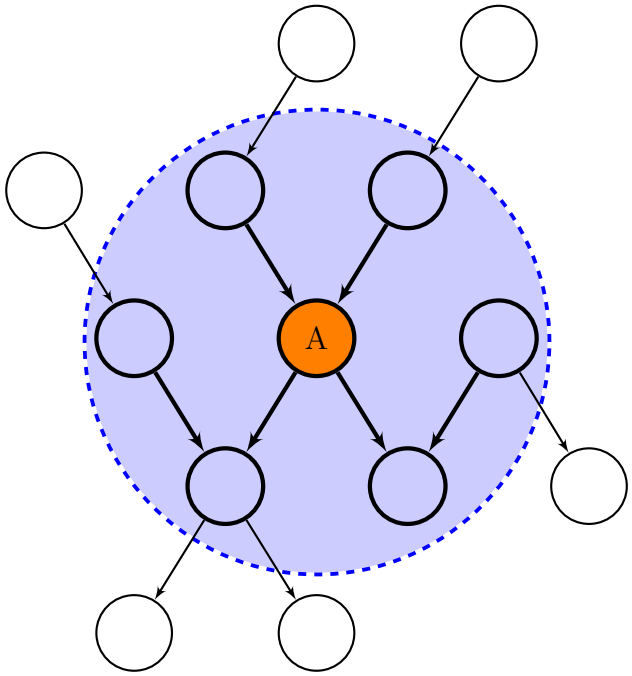


Figure 2.6: In a Bayesian network, the Markov blanket of node A includes its parents, children and the other parents of all of its children.

particularly useful because only the nodes in the Markov blanket affect the prediction of the class label directly, while the other nodes not in the Markov blanket affects the prediction of the class label via the nodes in the Markov blanket.

2.3 Supervised Machine Learning with Incomplete Information

In a supervised classification problem with incomplete information, as defined earlier in Section 2.1, we are given an incomplete training sample including the instance $x_1, \ldots, x_n$ and the labels for these instances $y_1, \ldots, y_n$. in which one or more attributes of x_i and/or y_i are unknown for $i \in \{1, \ldots, n\}$. In our study, we assume either values for whole vector Y are not given, or the values for the whole 2-dimensional X_{ij} are not given.

The Expectation-Maximization (EM) algorithm (Dempster et al., 1977) is a popular class of iterative algorithms for maximum likelihood estimation in problems with incomplete data. It works by filling in the missing values in the data using existing values by computing the expected values. The EM algorithm consists of two steps, the *Expectation* step, and the *Maximization* step. The *Expectation step* basically fills in the missing data. The parameters are estimated in the *Maximization step* after the missing data are filled. This leads to the next iteration.

Many machine learning algorithms, when encountering incomplete data, can combine with EM to handle the missing values. When the labels of the training data and/or the features of the training data are not given but are available for a cost, we need to carefully choose which (instance, label) pairs or (instance, feature) pairs to purchase and then use a machine learning algorithm that can handle missing value to learn a classifier or hypothesis.

Without a hard budget, we will need to carefully choose which instances to label, and/or which (instance, feature) pairs to purchase to learn a good classifier with as little cost as possible. We call this **active learning** (Tong and Koller, 2000; Roy and Mccallum, 2001; Luo et al., 2004; Neville and Kuwadekar, 2011). With a hard budget, we will also need to use the given hard budget to carefully choose which instances to label, and/or which (instance, feature) pairs to purchase so that we can learn as good classifier or hypothesis as possible. We call this **budgeted learning** (Lizotte et al., 2003; Kapoor and Greiner, 2005b).

Here we have two examples, one is active learning in which labels of the training data are available at a cost and the other is budgeted learning in which the features of the training data are available at a cost, subject to an overall cost. In Figure 2.7, we are given a set of pictures of scenes, however these scenes have not yet been marked as "natural" or "man-made". The labels of the scenes can be purchased from paid workers from Amazon Mechanical Turk website. The goal is to learn a scene classifier by using as few queries to labelers as possible. An active learning algorithm needs to choose which scenes to label so

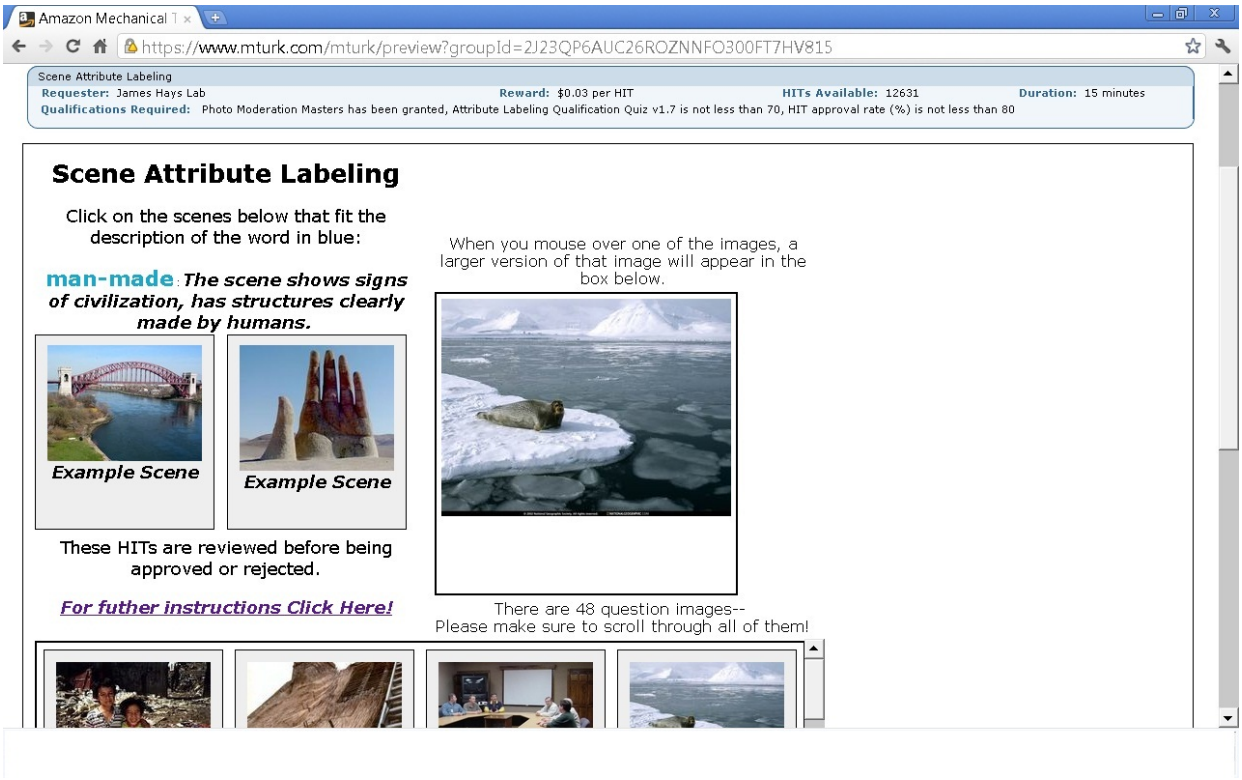


Figure 2.7: An example of active learning: scene labeling problem from Amazon Mechanic Turk website (Amazon Mechanical Turk, 2005).

that we can learn a classifier using as few queries as possible. To save cost, an active learning algorithm may choose those scenes that are hard to categorize to purchase, for example, the 5-finger scene in the middle of the pictures in Figure 2.7.

Table 2.1 suggest a project was allocated $2 million to develop a diagnosis classifier for patient cancer subtypes. In this study, a pool of patients with known cancer subtype were available, as were various diagnosis tests could be performed, each with a diagnostic cost. For this example, each test is expensive, and we have an overall budget of $2 million, so we have to carefully choose which patient do what kind of diagnosis test to learn a good cancer subtype classifier under the given $2 million budget. A budgeted learning algorithm may choose those tests that are most related to the patient cancer subtype for the patients.

In this dissertation, we study active learning under the setting that the labels of the

Table 2.1: An example of budgeted learning: A project was allocated $2 million to develop a diagnosis classifier for patient cancer subtypes.

Diagnosis Test 1	Diagnosis Test 2	Diagnosis 3	...	Cancer subtype
?	?	?	...	1
?	?	?	...	2
?	?	?	...	3
?	?	?	...	4
?	?	?	...	1
?	?	?	...	...

instances are not given and have to be purchased at a cost from multiple noisy labelers. We study budgeted learning under the setting that the features of the instances are not given and have to be purchased, subject to an overall limit. All the work in this dissertation falls under the umbrella within "supervised machine learning with incomplete information", more accurately, "supervised classification learning with incomplete information".

Chapter 3

Active Learning from Multiple Labelers with Varied Costs

3.1 Introduction

In active learning, it is assumed that unlabeled data is easy to obtain but labels are expensive. For example, when building a speech recognizer, it is easy to get raw speech samples, but labeling the samples is a tedious process in which a human must examine the speech signal and carefully segment it into phonemes. Therefore, only a subset of instances is chosen to be labeled. The goal is to learn a classifier by labeling as few instances as possible by actively selecting the instances to be labeled as learning proceeds. Many results in active learning focus on choosing the instances for labeling and assume that the labeling is handled by a single, noise-free labeler (Tong and Koller, 2000; Roy and Mccallum, 2001; Luo et al., 2004; Neville and Kuwadekar, 2011). However, it is possible that there is some noise in the labeling (Guillory and Bilmes, 2011), and it is possible that there is no perfect labeler and that instead multiple, noisy labelers are available. For example, consider the aforementioned speech recognition application. It is difficult to guarantee 100% accuracy from a human

labeler, due to the wide variations in how we interpret speech signals. However, one may easily have access to multiple human labelers, each with a different cost.

Another example of multiple noisy labelers is the online annotation tool Amazon Mechanical Turk (AMT) (Amazon Mechanical Turk, 2005). AMT requesters submit *human intelligence tasks* (HITs) to the site. A HIT is a task that is simple for humans but may be difficult to automate. Examples include: draw an outline around a human in a given picture, solve an optical character recognition task, etc. For their efforts, AMT workers are paid a fee, based on how many data items they label. Certainly, in the AMT model, one cannot guarantee noise-free labels from any human labelers, and in fact, depending on the task, noise rates may vary considerably. Finally, while requesters who submit the HITs set the pay rates, it is not conceptually difficult to extend the AMT idea to that of a marketplace in which the workers (labelers) would bid on jobs, using strong on-line ratings to justify higher costs. Such a marketplace would have variance both in cost and in quality of labelers.

We refer to this model as *active learning with multiple noisy labelers,* where the goal is to learn a hypothesis that generalizes well while spending as little as possible on queries to labelers. We will assume that the label given by labeler o_i for a particular instance will be the true label, corrupted by noise with probability $\eta_i < 1-1/v$, where v is the number of possible labels (i.e., each labeler performs strictly better than random guessing). We assume that the noise model is an i.i.d. process, where the independence is across labelers (one labeler's label of instance x is not influenced by the label issued by any other labeler for x) and across instances (a labeler's label for instance x is independent of the label it issued for instance x', which it saw earlier). Finally, we assume that each labeler's noise model is *persistent* in that if labeler o_i labels instance x as class y at some point, then it will always answer y as its label for x. Thus, there is no more information to be had by asking a labeler to label the same instance multiple times. Note that there is no hard budget in our problem definition. Instead, the goal in our model is to simply spend as little as possible while generalizing well.

Sheng et al. (2008) show that when labelers provide noisy labels under the persistent noise model, one can still estimate well the true labels of instances by requesting labels on a single instance from multiple, independent labelers. The idea is that, if all labeler responses are independent from each other and all noise rates are $< 1/2$ (for binary classification), then a majority vote from an appropriate subset of the labelers can be very effective. (We extend this concept into part of our algorithm: the notion of *combined accuracy* of a group of labelers.) However, they assume that all the labelers have the same costs and noise rates. Donmez et al. (2009) relaxed this assumption and assumed that labelers can have different, unknown accuracies. They proposed a method called IEThresh for active learning with multiple noisy labelers with different accuracies.

Briefly, IEThresh works as follows: First, the learning algorithm chooses an instance for labeling. Then it compares the relative performance of the labelers so far (via interval estimation learning (Kaelbling, 1993)) to choose what it believes to be the most accurate labelers to label the chosen instance. Next, it estimates the "ground truth" (true label) of the instance by a majority vote of these chosen labelers. At that point it updates its classifier based on the newly labeled training instance. Finally, it updates its estimate of each labeler's accuracy, and repeats the process.

Contrasting IEThresh to the naïve approach of simply having every labeler label every instance and then taking a majority vote (what is sometimes called "repeated labeling" in the literature and what we refer to as "Repeated" in our experimental section), the advantage of IEThresh is obvious. The execution of IEThresh excludes labelers believed to be inferior, which saves total cost over Repeated. However, while effective, IEThresh does not consider the possibility that one can further save labeling cost when many accurate labelers exist by further pruning the set of labelers used. Our new algorithms IEAdjCost and wIEAdjCost overcome this shortcoming.

The intuition behind our algorithms is that we "normalize" the accuracies of labelers

with respect to their costs, allowing for direct comparison between labelers. Then, rather than simply identifying the most accurate labelers, our algorithms instead seek a subset of labelers that, when combined, achieve the desired level of accuracy for low cost.

Our first algorithm, IEAdjCost, works in two phases. In Phase 1, it uses higher accuracy labelers to predict the ground truth by majority vote, and it stops estimating labeler accuracy when it has a sufficiently large set of labelers with good accuracy estimates. In Phase 2, a heuristic finds a subset of the labelers with good accuracy estimates that, when used together, has high accuracy and low cost. This subset is what is used from that point forward for labeling requests.

Thus, in Phase 1, there exist both exploration (estimating the accuracies of labelers) and exploitation (estimating ground truth by using only the subset of labelers believed to be the most accurate). In Phase 2, there is no exploration and IEAdjCost chooses a final set of labelers that, when combined, is believed to have high accuracy and low cost. (In Phase 2, we stop refining the accuracy estimates of the labelers since, by definition of the algorithm, we already have sufficiently good estimates for our purposes and further refinement would be an unnecessary expense.)

Another drawback of Donmez et al.'s IEThresh is that it ignores the possibility that labelers can have different costs. In reality, there could be high variance in the costs of available labelers, and the costs may not even correlate with accuracy (expensive labelers may not necessarily be highly accurate). But even in cases where cost correlates with accuracy, it is possible that multiple inexpensive labelers, when used together in a voting scheme, may have a greater accuracy than one highly expensive labeler. For these reasons, active learning with multiple labelers when both costs and accuracies vary, is a more complex problem than the case of identical costs, as addressed by IEThresh. Thus we introduce the notions of *adjusted cost* and *combined accuracy* that allow us to directly compare labelers with different costs and accuracies. Combined accuracy gives us the probability that a group

of independent labelers will make a labeling mistake when the label is found by a majority vote within that group. The adjusted cost of labeler o_j is the cost of o_j multiplied by the number of independent copies[1] of o_j needed to achieve a particular combined accuracy.

IEAdjCost was introduced in a preliminary version of this chapter (Zheng et al., 2010), in which only binary class labels are considered. Since then, we extended IEAdjCost to multiple (> 2) class labels and improved IEAdjCost to wIEAdjCost by giving each well-estimated labeler a weight. In wIEAdjCost, we heuristically assume that a set of well-estimated labelers reaches the required accuracy if the sum of their weights exceeds 1. We use this heuristic to improve the algorithm's efficiency. In addition, wIEAdjCost, when possible, uses the weights to filter the set of labelers used to estimate ground truth, which further reduces the total cost. These two improvements make wIEAdjCost outperform IEAdjCost.

Our experiments show that IEAdjCost saves cost when compared to IEThresh from the literature and the naïve algorithm Repeated. In addition, wIEAdjCost outperforms IEAdjCost by saving both time and cost. These new results are currently under review by the journal Machine Learning.

This rest of the chapter is organized as follows. Section 3.2 describes related work. Section 3.3 presents our new algorithms, including IEAdjCost in Section 3.3.1 and wIEAdjCost in Section 3.3.2. Our experiments are summarized in Section 3.4, comparing our algorithm IEAdjCost and wIEAdjCost to Repeated and IEThresh on six data sets from UCI (Frank and Asuncion, 2010) and two data sets from AMT (Amazon Mechanical Turk, 2005; Snow et al., 2008). We conclude in Section 3.5.

[1]This is purely hypothetical. In the persistent noise model, using the same labeler repeatedly on a single instance yields no new information.

3.2 Related Work

One of the primary objectives of engineering machine learning systems is to reduce the expense of human effort. Ironically, in many applications, significant amounts of human labor and time are still spent in preparing high-quality data in order to build such intelligent systems.

If we treat data preparation as an independent "post-processing" step after the data is initially collected, one way of utilizing incomplete, noisy or erroneous labeling sources without manual intervention is to infer the truth. The EM algorithm of Dempster et al. (1977) is a popular procedure for finding maximum likelihood estimates of parameters where the model depends on unobserved latent variables. Dawid and Skene (1979) proposed using EM to improve the quality of simple majority voting by experts in the biostatistics community. They showed that by using EM, one can better estimate the ground truth and therefore the accuracies of each expert. Smyth et al. (1995) took advantage of the EM algorithm for the application of a large-scale image analysis problem. They studied the problem of inferring volcano types based on labelings from multiple labelers. The EM algorithm was applied to estimate the conditional distribution of a volcano's type given noisy labels. This information was then used as the ground truth to learn a classifier.

Sheng et al. (2008) studied the problem of acquisition of labels from multiple imperfect labelers, and one of their conclusions was that repeated labeling (getting an instance's label once from each of several labelers) is often preferable to single labeling (getting an instance's label once from a single labeler), even when labels are not particularly cheap. In another study in the natural language processing domain, Snow et al. (2008) concluded that multiple cheap labelers can be preferable to a single expert. They also proposed a bias correction technique by checking the labelers' performance on the "gold standard" (true labels) in order to fine tune each labeler's relative weight. By doing experiments on five tasks from

the Amazon Mechanical Turk system (2005), they showed that significant improvements in annotation quality can be achieved at a fraction of the usual expense. In the medical domain, where an expert's performance is usually described by the trade-off between sensitivity and specificity, another study by Raykar et al. (2009) showed a significant advantage of combining the opinions of multiple experts when training a logistic regression classifier.

The goal of active learning from multiple noisy labelers with varied costs is to learn a classifier using as little cost as possible. Guillory and Bilmes (2011) studied the problem of simultaneously learning a model of user preferences (in the form of an objective function on subsets of items) and finding a set of items that best satisfy that preference model (minimizing or maximizing the objective function) in the presence of limited adversarial noise. An example of this is a movie recommendation problem in which the user's taste preferences must be actively learned through feedback from the user, while the system simultaneously searches for a collection of movies that maximizes the learned preference function. The goal, then, is to minimize the joint cost of both learning and covering.

Donmez and Garbonell (2008) provided a decision-theoretic framework to select labelers and instances in an active learning setting. However, their algorithms required at least one labeler to be 100% correct and able to answer all queries. They studied scenarios where (1) one labeler answers every query and is noise-free and the other is noise-free but declines to answer some queries; (2) both labelers answer all queries, one noise-free and one noisy; and (3) both labelers answer all queries without noise, but one labeler has a fixed cost per query and the other labeler's cost depends on the instance's class posterior probability.

Finally, Donmez et al. (2009) extended Donmez and Garbonell's work to remove the limitations that there are only two labelers and that one labeler must be perfect. They proposed a method called IEThresh that takes multiple noisy labelers and chooses those labelers with high accuracy to both save cost and improve accuracy. They studied active learning in the case of multiple noisy labelers with identical costs. Their method assumes

that each labeler's accuracy is strictly better than random guessing, which implies that majority voting will, in expectation, yield correct labels. They reduced the cost of labeling by filtering out labelers that are not as good as the others.

Donmez et al.'s work can be improved in two ways. First, after estimating the accuracies of the labelers, one does not need to select all of those that are highly accurate. Instead, one only needs to choose enough such labelers to achieve the desired rate of combined accuracy when using majority voting. Second, instead of assuming that all labelers have identical costs, labelers with different costs and accuracies should be directly compared, and the goal should be to choose a set of labelers whose combined accuracy is high but total cost is low. Our algorithm IEAdjCost makes both of these improvements. We then improve on IEAdjCost to get wIEAdjCost, which gives each well-estimated labeler a weight. In wIEAdjCost, we heuristically assume that a set of well-estimated labelers reaches the required accuracy if the sum of their weights exceeds 1. We use this heuristic to improve the algorithm's efficiency. In addition, wIEAdjCost, when possible, uses the weights to filter the set of labelers used to estimate ground truth, which further reduces the total cost. These two improvements make wIEAdjCost outperform IEAdjCost. IEAdjCost will avoid selecting labelers, even if they are accurate, if they are expensive and not needed to achieve the required combined accuracy. wIEAdjCost further improves IEAdjCost by simplifying the exponential-time computation of combined accuracy to a linear-time computation of total weights, and exploits well-estimated labelers to estimate the ground truth as early as possible.

3.3 Our Algorithms

In this section, we introduce our two algorithms, IEAdjCost in Section 3.3.1 and wIEAdjCost in Section 3.3.2.

3.3.1 IEAdjCost

In this section, we first introduce notation in Section 3.3.1.1, then summarize the principle of uncertainty sampling (Lewis and Gale, 1994a; Tong and Koller, 2000) used in IEAdjCost to choose instances to label in Section 3.3.1.2. In Section 3.3.1.3, we review interval estimation (1993), which is what we use to estimate the accuracies of the labelers, and we describe the various subsets of labelers that we use in our algorithm and their roles. Section 3.3.1.4 defines our concepts of combined accuracy and adjusted cost, which are used by our heuristic LabelersByAdjCost (Section 3.3.1.5) that finds a final set of labelers with high accuracy and low cost, to be used in Phase 2. Finally, Section 3.3.1.6 puts the pieces together into the full algorithm IEAdjCost.

3.3.1.1 Notation

We consider the problem of supervised classification for which a training set $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ is used to induce a classifier, where $\{x_1, \ldots, x_n\} = X$ is the set of instances and $\{y_1, \ldots, y_n\} = Y$ is the set of labels of those instances. We assume that the instances x_i are given, while the labels y_i are only obtained by purchasing them from one or more labelers from the set $O = \{o_1, \ldots, o_N\}$. (For convenience, we will partition X into U and L, where U is the set of instances for which a label has not yet been purchased, and L is the set of labeled instances.[2]) The cost of purchasing a label from labeler $o_i \in O$ is c_i and the label returned by o_i is correct with probability $a_i = 1 - \eta_i$ (labeler o_i's *accuracy*). We denote by $\hat{a}_i$ our algorithm's estimate of this accuracy.

3.3.1.2 Selecting Instances

In each iteration of our algorithm, the first step is to choose an unlabeled instance $x^* \in U$ for labeling. Our algorithm chooses the instance to label via uncertainty sampling (Lewis

[2]The actual label that goes to L is explained in Algorithms 3.3 and 3.6.

and Gale, 1994a; Tong and Koller, 2000), in which one chooses the unlabeled instance for which the current hypothesis is least certain in its classification. Specifically, if $\hat{P}(y \mid x)$ is the current hypothesis's estimate of the probability that instance x has label y, then the instance selected for labeling is the one that maximizes the entropy of the distribution over the labels[3]:

$$x^* = \operatorname*{argmin}_{x \in U} \sum_{y \in \mathcal{Y}} \left(\hat{P}(y \mid x) \ln(\hat{P}(y \mid x)) \right) \quad , \tag{3.1}$$

where $\mathcal{Y}$ is the set of possible labels.

3.3.1.3 Estimating Labeler Accuracy

Our algorithm IEAdjCost consists of two phases. Phase 1 is the exploration and exploitation phase and Phase 2 is the pure exploitation phase. In Phase 1, IEAdjCost purchases labels from many labelers (exploration) and uses the labels offered by a select subset of labelers to train the classifier (exploitation). In Phase 2, IEAdjCost selects a final subset of labelers to label training data.

To estimate the accuracies of the labelers, we adapt the procedure of interval estimation from Kaelbling (1993), which was also applied by Donmez et al. (2009). Our algorithm estimates the accuracy a_i of labeler o_i by asking o_i to label several instances from U and giving that labeler a *reward* of 1 for each instance that it labels correctly, and a reward of 0 otherwise. Then our estimate $\hat{a}_i$ of a_i is the sample mean of the rewards that o_i received. We also compute $\hat{s}_i$, the sample standard deviation. We then compute the upper interval

[3]The use of entropy for instance selection improved performance over the method used in the preliminary version of this Chapter (Zheng et al., 2010).

(UI_i) and lower interval (LI_i) of an ($\alpha/2$)-level confidence interval centered on $\hat{a}_i$:

$$\mathrm{UI}_i = \hat{a}_i + t_{\alpha/2}^{(n_i-1)} \frac{\hat{s}_i}{\sqrt{n_i}} \tag{3.2}$$

$$\mathrm{LI}_i = \hat{a}_i - t_{\alpha/2}^{(n_i-1)} \frac{\hat{s}_i}{\sqrt{n_i}} \ , \tag{3.3}$$

where n_i is the number of rewards (of value 0 or 1) given to labeler o_i (i.e., the number of values used to compute $\hat{a}_i$ and $\hat{s}_i$), and $t_{\alpha/2}^{n_i-1}$ is the critical value for the Student's t distribution with $n_i - 1$ degrees of freedom at the $\alpha/2$ confidence level. In our experiments, we set $\alpha = 0.05$. Since the true labels of instances are unavailable, we estimate the true labels by a majority vote of a select subset of the labelers, which we describe later. To avoid trivialities in the formulas, we initialize each labeler's set of rewards to be $\{0, 1\}$.

The first step of Phase 1 is for IEAdjCost to determine which subset of labelers will have their accuracy estimates refined. IEAdjCost uses a parameter λ (in our experiments, we tried $\lambda \in \{0.01, 0.25, 0.5, 0.75, 1\}$) that specifies the fraction of labelers from O that must be explored. Define

$$\mathrm{rank}(o_i) = 1 + |\{o_j \in O : \mathrm{UI}_j > \mathrm{UI}_i\}|$$

as 1 plus the number of labelers in O that have a larger upper interval as labeler o_i, i.e. it is o_i's rank in a partial ordering of the labelers based on upper interval.

Now we define the set of labelers whose accuracy estimates will be refined:

$$O_\ell = \{o_k \in O : \mathrm{rank}(o_k) \leq \lceil \lambda |O| \rceil\} - O_g \ , \tag{3.4}$$

where O_g is the set of labelers that have sufficiently good estimates of their accuracies (see below). Thus, we select for refinement the set of labelers whose upper intervals on their accuracy estimates rank relatively high, but do not already have sufficiently good estimates (in set O_g).

To estimate the accuracies of labelers in O_ℓ, we need an estimate of the correct label of the instance x^*. This is computed by taking a majority vote of a set of labelers, which we denote O_r:

$$O_r = \left\{ o_i : \mathrm{UI}_i \geq \epsilon \max_{o_j \in O} \mathrm{UI}_j \right\} \quad , \tag{3.5}$$

where $\epsilon \in [0, 1]$ is a parameter (in our experiments, we tried $\epsilon \in \{0.7, 0.8, 0.9\}$). The idea is adapted from Donmez et al. (2009), which chooses those labelers whose upper confidence interval are within a factor of ϵ of the maximum confidence interval (and therefore filtering out those inferior/low-accuracy labelers) to estimate the ground truth. Thus in each iteration in Phase 1, IEAdjCost requests labels of the chosen instance from the labelers in $O_\ell \cup O_r$. From the results of the labelings, we can update the rewards for each labeler $o_i \in O_\ell$, and then update UI_i and LI_i. (Since it does not cost any more, we also update UI and LI of labelers in $O_r - O_\ell$.)

Once a labeler's accuracy is well-estimated, we put it in

$$O_g = \{ o_i \in O : \mathrm{UI}_i - \mathrm{LI}_i \leq \delta \} \quad . \tag{3.6}$$

(In our experiments, we tried $\delta \in \{0.1, 0.2, 0.3\}$.) The intuition is to use δ to control how well the labelers are estimated. The smaller δ is, the more accurate the estimation. Phase 1 ends when $|O_g| \geq \lceil \lambda |O| \rceil$ and function LabelersByAdjCost(O_g) returns a non-empty set (Section 3.3.1.5).

Figure 3.1 shows an example of the relationship among O_ℓ, O_r, O_g, and O_f. To summarize, O_ℓ is the set of labelers whose accuracy estimates we are refining, O_r is the set of high-accuracy labelers that are used to predict the ground truth, O_g is the set of labelers whose accuracies are well-estimated, and O_f is the set of final chosen labelers (see Section 3.3.1.5).

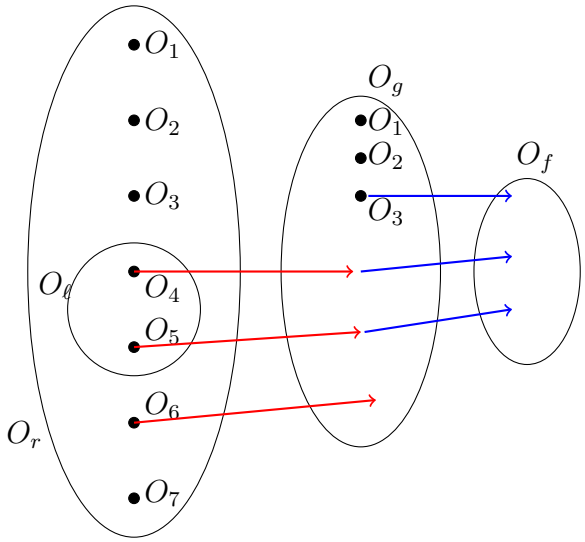


Figure 3.1: An example of the relationship among O_ℓ, O_r, O_g, and O_f in Phase 1. The relationship of them are as follows: $O_f \subseteq O_g$ and $O_l \cap O_g = \emptyset$.

3.3.1.4 Combined Accuracy and Adjusted Cost

Once O_g is sufficiently large, we switch to Phase 2, the pure exploitation phase. In this phase, a fixed set of labelers is chosen to label all future instances that are added to the labeled training set L. (We no longer refine accuracy estimates in Phase 2 because we already have sufficiently good estimates, and continuing to do so only incurs more cost.) Ideally, we want to choose labelers that together are highly accurate and individually are inexpensive to query. In our process to find such a set of labelers, we use quantities that we call *combined accuracy* and *adjusted cost*.

The notion of combined accuracy of a set O of labelers is based on the idea that, when taking a majority vote of the labelers in O, the label is estimated correctly. The combined accuracy of O is the probability that this occurs.

Without loss of generality, assume that the correct label of the current instance is 0. For a given instance, labeler i labels it correctly as 0 with probability a_i. Thus, labeler i will incorrectly predict one of the other classes from $\{1, 2, \ldots, v-1\}$ with a combined

probability of $1-a_i$. To simplify our calculations, we assume that this probability is uniformly distributed across all classes, so the probability of labeler i labeling an instance as any one of $\{1, 2, \ldots, v-1\}$ is $(1-a_i)/(v-1)$.

Let $\mathcal{P}(O)$ be the set of all partitions of the set of labelers $O = \{o_1, \ldots, o_N\}$ into exactly v subsets, some of which can be empty. Each such partition represents how labels are predicted by labelers from O. For partition $S \in \mathcal{P}(O)$, let S_i denote the ith set in partition S. Thus $S_i \subseteq O$ is the subset of labelers that predict label i on the current instance. Assuming that the correct label of the current instance is class 0, for each partition S, S_0 represents the subset of labelers that are correct. When computing combined accuracy, we are interested in the partitions in $\mathcal{P}(O)$ for which S_0 is at least as large as all other S_i, since these partitions are the ones in which a majority of labelers are correct. Now, to compute the combined accuracy of a set of labelers O, we simply need to compute the probability of all partitions S for which S_0 is the largest:

$$A(O) = \sum_{S \in \mathcal{P}(O): |S_0| \geq \max_{1 \leq i \leq v-1} |S_i|} \frac{1}{u_S} \prod_{j \in S_0} a_j \prod_{\ell \in \left(\bigcup_{i=1}^{v-1} S_i\right)} \left(\frac{1-a_\ell}{v-1}\right) \ ,$$

where $u_S = |\{i : |S_i| = |S_0|, i \in \{0, 1, \ldots, v-1\}\}|$ is the number of subsets $S_i \in S$ such that $|S_i| = |S_0|$, including S_0 itself. Dividing each product by u_S distributes partition S's accuracy uniformly across all sets S_i in case of a tie. Since we don't know the exact accuracies of the labelers, we substitute our estimates when computing combined accuracy:

$$\hat{A}(O) = \sum_{S \in \mathcal{P}(O): |S_0| \geq \max_{1 \leq i \leq v-1} |S_i|} \frac{1}{u_S} \prod_{j \in S_0} \hat{a}_j \prod_{\ell \in \left(\bigcup_{i=1}^{v-1} S_i\right)} \left(\frac{1-\hat{a}_\ell}{v-1}\right) \ . \tag{3.7}$$

In other words, for every partition of O in which the number of labelers $|S_0|$ predicting the correct class value is at least as large as the number of those labelers predicting any other class value, the products are the probability that the labelers in S_0 will correctly label

the instance. Summing over all such partitions, we get the probability that some dominant subset will correctly label the instance.

Using combined accuracy, we can now define the adjusted cost of a labeler, which quantifies its cost/accuracy tradeoff. Central to the concept of the adjusted cost of labeler o_i is its *multiplier* $\beta_i(R)$, which is the number of copies of o_i (in a hypothetical non-persistent noise model) needed to achieve a combined estimated accuracy of at least R. I.e., $\beta_i(R)$ is the minimum value of k such that[4]

$$\hat{A}(\{\overbrace{o_i, \ldots, o_i}^{k}\}) \geq R \ .$$

(Since we assume that R is fixed ahead of time as a parameter, for brevity's sake we will omit it from now on and just refer to o_i's multiplier as β_i.)

We can now define the adjusted cost of o_i in terms of its multiplier:

$$\text{AdjustedCost}(o_i) = c_i \beta_i \ . \tag{3.8}$$

3.3.1.5 Choosing the Final Set of Labelers

Given a set $O_g \subseteq O$ of distinct labelers whose accuracies are well-estimated, we want to choose a final subset $O_f \subseteq O_g$ to use in Phase 2. The subset we seek is one that achieves a minimum combined accuracy of R whose combined costs are minimized:

$$O_f = \underset{Q \subseteq O_g : \hat{A}(Q) \geq R}{\text{argmin}} \ C(Q) \ , \tag{3.9}$$

where $C(Q) = \sum_{o_j \in Q} c_j$ is the combined cost of Q.

To solve this optimization problem, we employ the heuristic LabelersByAdjCost, which first sorts the labelers in O_g in ascending order by their adjusted costs, breaking ties by their

[4]In Section 3.3.1.5, we describe how to compute β_i.

estimated accuracies. Then it adds the labelers one at a time to O_g in sorted order, stopping when the combined accuracy of the labelers in O_g exceeds the threshold R. Pseudocode for this routine is in Algorithm 3.1.

Algorithm 3.1 LabelersByAdjCost($\{o_1, \ldots, o_N\}$)

```
Compute adjusted cost for each labeler per Equation (3.8);
Sort the labelers in ascending order by their adjusted costs (breaking ties by their accuracies in descending order), yielding {o_{r_1}, ..., o_{r_N}} ;
for i from 1 to N do
   Let O' = {o_{r_1}, ..., o_{r_i}} ;
   Sort the labelers of O' by their accuracies in descending order (breaking ties by their costs in ascending order), yielding {o_{z_1}, ..., o_{z_i}} ;
   for j from 1 to i do
      if CombAccuReachesThresh({o_{z_1}, ..., o_{z_j}}, R) then
         return {o_{z_1}, ..., o_{z_j}} ;
      end if
   end for
end for
return ∅ ;
```

Algorithm 3.2 CombAccuReachesThresh($\{o_{z_1}, \ldots, o_{z_j}\}, R$)

```
m = |{o_{z_1}, ..., o_{z_j}}| ;
if m ≤ m_0 (where m_0 is a parameter) then
   Compute the combined accuracy Â of {o_{z_1}, ..., o_{z_j}} per Equation (3.7), stop early if Â ≥ R ;
   if Â ≥ R then
      return true ;
   else
      return false ;
   end if
else
   if m ≥ max_{o_i ∈ {o_{z_1}, ..., o_{z_j}}} β_i then
      return true ;
   else
      return false ;
   end if
end if
```

In Algorithm 3.1, we use combined accuracy to determine each labeler's multiplier for

computing its adjusted cost, and to compute the combined accuracy of candidate O_f sets. Below we describe methods we use to perform each task while mitigating the intractability of Equation (3.7).

There is a simple means to find the multiplier of a labeler with an accuracy estimate of $\hat{a}$ that leverages the fact that all the accuracies in the hypothetical collection are equal. Note that the vector $(|S_0|, \ldots, |S_{v-1}|)$ follows a multinomial distribution with parameters M (the number of labelers) and $p = (\hat{a}, (1-\hat{a})/(v-1), \ldots, (1-\hat{a})/(v-1))$. Let the random variables X_i indicate the number of times outcome number i was observed over the M trials,

$$P(X_0 = |S_0|, X_{v-1} = |S_{v-1}|) = \sum_{|S_0|+\cdots+|S_{v-1}|=N} \frac{M!}{|S_0|! \cdots |S_{v-1}|!} \hat{a}^{|S_0|} \left(\frac{1-\hat{a}}{v-1}\right)^{(M-|S_0|)} .$$

The probability that we choose class 0 (correct class value) as the dominant class value (when breaking ties uniformly at random) is

$$\sum_{\substack{|S_0| \geq \max_{1 \leq i \leq v-1} |S_i| \\ |S_0|+\cdots+|S_{v-1}|=M}} \frac{1}{u_S} \frac{M!}{|S_0|! \cdots |S_{v-1}|!} \hat{a}^{|S_0|} \left(\frac{1-\hat{a}}{v-1}\right)^{(M-|S_0|)} ,$$

where $u_S = |\{i : |S_i| = |S_0|, i \in \{0, 1, \ldots, v-1\}\}|$ is the number of subsets $S_i \in S$ such that $|S_i| = |S_0|$, including S_0 itself.

Since the left-hand side of the following expression is monotonic in M, we simply perform a binary search to find the smallest value of M (the number of labelers) such that the following expression holds:

$$\sum_{\substack{|S_0| \geq \max_{1 \leq i \leq v-1} |S_i| \\ |S_0|+\ldots+|S_{v-1}|=M}} \frac{1}{u_S} \frac{M!}{|S_0|! \ldots |S_{v-1}|!} \hat{a}^{|S_0|} \left(\frac{1-\hat{a}}{v-1}\right)^{(M-|S_0|)} \geq R .$$

The time complexity to enumerate all the possible partitions is exponential for data sets with multiclass labels. To mitigate this, in our experiments, we use a search tree with nodes

$\{|S_1|, \ldots, |S_{v-1}|\}$, where each node has a set of candidate values $\{0, \ldots, v-1\}$. We then break the symmetry of $\{|S_1|, \ldots, |S_{v-1}|\}$ (the values of $\{|S_1|, \ldots, |S_{v-1}|\}$ are interchangeable) to save time (Law et al., 2007).

Further, for fixed R, the multiplier for accuracy estimate $\hat{a}$ never changes, so we can in fact *precompute* the multipliers for each value of $\hat{a} \in \{0.51, 0.52, \ldots, 0.99, 1.00\}$. For example, if $R = 0.99$ and $v = 2$, then labelers with accuracies $0.8, 0.81, \ldots, 0.99$ will have multipliers $13, 11, 11, 9, 9, 9, 7, 7, 7, 7, 5, 5, 5, 5, 5, 3, 3, 3, 3, 1$.

When building the set O_f in Algorithm 3.1, iteration i of the loop of Lines 3–11 considers the i labelers in O_g that have the lowest adjusted costs. Call this set O'. Since we do not need to know O''s exact combined accuracy, but only whether it exceeds R, Line 7 of Algorithm 3.1 calls CombAccuReachesThresh (Algorithm 3.2) to determine if O' is acceptable to become O_f. For increased efficiency, calls to CombAccuReachesThresh are on sets that contain the most accurate labelers from O' (Lines 5–10), which will more quickly return a $true$ answer if one is possible[5].

CombAccuReachesThresh computes the combined accuracy using Equation (3.7) if the number of labelers $m \leq m_0$ (in our experiments we set m_0 to 20). The computation stops if the value exceeds R. If instead $m > m_0$, then to improve efficiency, we instead employ the heuristic that returns $true$ if m is at least as large as the largest multiplier in the set of labelers, and $false$ otherwise.

If multipliers are precomputed, then Algorithm 3.2 has time complexity $O(m + m_0 v^{m_0 - 1})$. In Algorithm 3.1, sorting the labelers in Line 2 takes $O(N \log N)$ time. In iteration i of the **for** loop, adding labeler o_{r_i} to the already sorted list in Line 5 takes $O(\log i)$ time, and iteration j of the inner **for** loop takes time $O(j + m_0 v^{m_0 - 1})$. Summing over both loops yields a time complexity of $O(N^3 + N^2 m_0 v^{m_0 - 1})$.

[5]Note that this efficiency optimization does not necessarily return the subset with smallest adjusted costs since it doesn't return a prefix of O', but we expect the set returned to still have low total cost, since what it returns is a subset of O', which consists of the i labelers with lowest adjusted costs.

3.3.1.6 IEAdjCost

Pseudocode for our main algorithm IEAdjCost is given in Algorithm 3.3. The parameter $\lambda \in (0, 1]$ used on Lines 5 and 12 specifies the minimum value of $|O_g|/|O|$, i.e. the minimum fraction of original labelers that need to have well-estimated accuracies. Thus $\lambda = 1/|O|$ means that we merely insist on at least one labeler in O to have an accuracy that we estimate well, where $\lambda = 1$ means that we insist that we have good estimates of the accuracies of all labelers.

In the experimental section we study the effect of varying λ. Will smaller values of λ will always yield superior results? That is not necessarily the case. Since λ determines what fraction of all the labelers are considered at each stage, the following guidelines apply to setting its value.

1. If one believes that there exists a labeler that is cheap and has high accuracy, then the smaller the λ value is, the more cost one can save on exploring. In our experiments, it is the case that some highly accurate labelers have low cost. As expected, low values of λ yielded high-accuracy classifiers at a lower cost than larger values of λ.

2. If one believes that the only highly accurate labelers are expensive, then increasing λ will allow our algorithm to consider cheaper alternatives, which when taken collectively will perform as well as the expensive labelers but for less cost.

3.3.2 wIEAdjCost

Despite the advantages that IEAdjCost has over IEThresh (which are corroborated by our experimental results), it has two drawbacks. One drawback is the exponential time complexity of computing combined accuracy of different labelers with different accuracies (recall that when all accuracies are the same, we can more efficiently compute combined accuracy

Algorithm 3.3 IEAdjCost

1: $O_g = \emptyset$;
2: **Phase 1:** Initialize rewards for each labeler to $\{0, 1\}$;
3: Compute the upper and lower confidence interval for each labeler per Equations (3.2) and (3.3) ;
4: Pick the most uncertain unlabeled instance x^* for labeling per Equation (3.1) ;
5: Build O_ℓ per Equation (3.4) and O_r per Equation (3.5) ;
6: Purchase labels for x^* from all labelers in $O_\ell \cup O_r$;
7: Estimate the ground truth $\bar{y}$ of x^* using unweighted majority voting of labelers in O_r ;
8: Update the labeled training data $L = L \cup (x^*, \bar{y})$ and train a new classifier with L ;
9: Update the rewards of all labelers in $O_\ell \cup O_r$;
10: Compute the upper and lower confidence interval for each labeler per Equations (3.2) and (3.3) ;
11: Compute the set O_g of well-estimated labelers per Equation (3.6) ;
12: **if** $|O_g| \geq \lceil \lambda |O| \rceil$ and $O_f = \text{LabelersByAdjCost}(O_g) \neq \emptyset$ **then**
13: GOTO Step 17 ;
14: **else**
15: GOTO Step 4 ;
16: **end if**
17: **Phase 2:** Use O_f to label future instances, add them to L, and train classifier until training complete ;

via binary search). The other drawback is that O_f is only used in Phase 2 until a specified fraction of labelers are well-estimated. IEAdjCost uses all labelers in O_r to estimate the ground truth, which could be excessive if there exists a smaller set of well-estimated labelers whose combined accuracy is sufficiently high. To overcome these two drawbacks, we propose a new algorithm wIEAdjCost. wIEAdjCost introduces the notion of the "weight" of a labeler, which encapsulates the idea of adjusted cost. wIEAdjCost gives higher weights to higher accuracy labelers and simplifies the procedure of judging whether a set of labelers reaches combined accuracy R by instead approximating when this happens by summing the total weights of the labelers. Specifically, rather than seeking a set of well-estimated labelers that reaches combined accuracy R, wIEAdjCost instead seeks a set of well-estimated labelers whose weights sum to at least 1. Further, wIEAdjCost, whenever possible, uses the set of well-estimated labelers O_f to estimate ground truth in Phase 1 if some subset of O_f has

combined accuracy at least R.

This section is organized as follows. We first define the concept of weight and related notation. Then in Section 3.3.2.2 we describe how we find a set of well-estimated labelers with high accuracy and low cost, to be used in both Phase 1 and Phase 2. Section 3.3.2.3 shows how wIEAdjCost computes O_r. Finally, Section 3.3.2.4 puts the pieces together into the full algorithm wIEAdjCost.

3.3.2.1 Notation

An important concept in our algorithm wIEAdjCost is that of a labeler's *weight*. We define the weight of o_i in terms of its multiplier $\beta_i \geq 1$ as:

$$\text{Weight}(o_i) = \frac{1}{\beta_i} \ . \tag{3.10}$$

The intuition behind the importance of weight is that a labeler o_i with weight $1/\beta_i$ has that much belief in its predicted class value. We also define weight of a set of labelers as the sum of the weights of the labelers:

$$\text{Weight}(\{o_1, \ldots, o_j\}) = \sum_{i=1}^{j} \text{Weight}(o_i) \ . \tag{3.11}$$

Given a specific instance x, we define SumWeights(y) of a class value $y \in \mathcal{Y} = \{0, 1, \ldots, v-1\}$ as:

$$\text{SumWeights}(y) = \sum_{o_i \text{ predicts class } y \text{ on instance } x} \text{Weight}(o_i) \ . \tag{3.12}$$

Note that

$$\sum_{o_i \in O} \text{Weight}(o_i) = \sum_{j=0}^{v-1} \text{SumWeights}(j) \ .$$

3.3.2.2 Procedure LabelersbywAdjCost

Both LabelersbyAdjCost and its weighted version Labelersby**w**AdjCost choose a set of well-estimated labelers for labeling a given instance. The order of how the sets of labelers are considered is computed similarly. The difference is that LabelersbyAdjCost chooses a set of labelers that will have equal votes for determining the final class label, while LabelersbywAdjCost chooses a set of labelers whose votes can be unequal (each with a weight) for determining the final class label. Therefore, another difference between these procedures is the way that they judge whether a set of well-estimated labelers qualify as the final chosen labelers, that is, reaches a combined accuracy R. Specifically, LabelersbyAdjCost judges whether a set of labelers reaches R via Algorithm 3.2, which directly computes the combined accuracy via Equation (3.7) if the number of labelers m is at most the parameter m_0, and otherwise conservatively bases its decision on how large m is when compared to the maximum of the labelers' multipliers β_i. Application of this heuristic is done for the sake of computational efficiency. In contrast, rather than seeking a set of well-estimated labelers that reaches combined accuracy R, it instead seeks a set of well-estimated labelers whose weights sum to at least 1.

Once some well-estimated labelers O' are chosen via procedure LabelersbywAdjCost, i.e.,

$$\sum_{\text{labeler } o_i \in O'} \text{Weight}(o_i) \geq 1 \ ,$$

we normalize the weights of these chosen labelers:

$$\text{Weight}'(o_i) = \text{Weight}(o_i) / \sum_{o_j \in O'} \text{Weight}(o_j) \ .$$

If $O' \neq \emptyset$ and at least $\lambda|O|$ labelers are well-estimated, wIEAdjCost switches to Phase 2 and O' is used in Phase 2 as O_f to estimate the ground truth of the chosen instance via a

Algorithm 3.4 LabelersbywAdjCost($\{o_1, \ldots, o_N\}$)

```
1: Compute adjusted cost for each labeler per Equation (3.8)
2: Compute weight for each labeler per Equation (3.10)
3: Sort the labelers in ascending order by their adjusted costs (breaking ties by their accu-
   racies in descending order), yielding {o_{r_1}, ..., o_{r_N}}.
4: for i from 1 to N do
5:    Let O' = {o_{r_1}, ..., o_{r_i}} ;
6:    if Weight({o_{r_1}, ..., o_{r_i}}) ≥ 1 then
7:       Sort the labelers of O' by their accuracies in descending order (breaking ties by their
         costs in ascending order), yielding {o_{z_1}, ..., o_{z_i}} ;
8:       for j from 1 to i do
9:          if Weight({o_{z_1}, ..., o_{z_j}}) ≥ 1 then
10:            return {o_{z_1}, ..., o_{z_j}} ;
11:         end if
12:      end for
13:   end if
14: end for
15: return ∅ ;
```

weighted vote. If $O' \neq \emptyset$ and less than $\lambda|O|$ labelers are well-estimated, wIEAdjCost remains in Phase 1 and O' is used in Phase 1 as O_r to estimate the ground truth of the chosen instance via a weighted vote. If $O' = \emptyset$, wIEAdjCost remains in Phase 1 and those labelers computed via Equation (3.5) are used to estimate the ground truth of the chosen instance via an unweighted (fair) vote.

3.3.2.3 Computing O_r

Just as in Section 3.3.1.3, we estimate a labeler's accuracy as in Equations (3.2) and (3.3) and compute O_ℓ as in Equation (3.4). However, IEAdjCost and wIEAdjCost differ in computing O_r. In IEAdjCost, O_r is always computed via Equation (3.5), i.e, those labelers whose upper confidence interval are within ϵ of the maximum upper confidence interval, and the chosen labelers have equal weight. wIEAdjCost improves this method by the following strategy: if procedure LabelersbywAdjCost returns a set of labelers O' that is not empty (i.e., there exists a set of well-estimated labelers whose total weight exceeds 1 and whose cost

is approximately minimized) and at the same time less than $\lambda|O|$ labelers are well-estimated, then wIEAdjCost remains in Phase 1 and it assigns O' to O_r and each labeler o_i in O' has a weight of $\text{Weight}'(o_i) = \text{Weight}(o_i)/\sum_{o_j \in O'} \text{Weight}(o_j)$. On the other hand, if O' is empty, then we set O_r by calling LabelersbyAdjCost (Algorithm 3.3), just as in IEAdjCost.

Different from LabelersbyAdjCost, which queries all chosen labelers (as in Line 6 of Algorithm 3.3, IEAdjCost), LabelersbywAdjCost queries the chosen labelers one by one (sorting the labelers first by their adjusted cost in ascending order, breaking ties by their estimated accuracies in descending order). After getting the current labeler's answer y_{current} and updating the value of SumWeights for y_{current}, the procedure LabelersbywAdjCost computes maxS as those class values whose SumWeights value are the highest among all class values. Then if

$$\text{SumWeights}(y) > \max_{y' \notin \text{maxS}} \text{SumWeights}(y') + \left(1 - \sum_{y''} \text{SumWeights}(y'')\right) \quad ,$$

where $y \in \text{maxS}, y' \notin \text{maxS}$, $y'' \in \mathcal{Y} = \{0, 1, \ldots, v-1\}$, and $(1 - \sum_{y''} \text{SumWeights}(y''))$ is the total weight of unqueried labelers, we stop querying the remaining labelers and use a randomly chosen value in maxS as the estimate of the ground truth. For example, if $v = 3$, SumWeights(0)=0.3, SumWeights(1)=0.2, and SumWeights(2)=0.2, and now we query one labeler which answers value 0 and this labeler's weight is 0.11, now we have SumWeights(0) = 0.41, SumWeights(1)=0.2, and SumWeights(2)=0.2. Since no matter how the remaining labelers answer, the dominant class value will be 0, we stop querying the remaining labelers and get an estimate of the ground truth as 0.

Algorithm 3.5 shows the procedure of querying the labelers one by one for instance x^*. First, given initial values of SumWeights for all class values, the procedure LabelersbywAdjCost judges whether there exists a dominant class value(s) (i.e., no matter how the given labelers label instance x^*, the dominant class value(s) will have the highest value of

SumWeights); if such class value(s) exist, the procedure returns this class value or a randomly selected value among those that are tied. If no dominant class value(s) exist(s), then the labelers are queried individually. Each time one labeler is queried, the procedure LabelersbywAdjCost updates the SumWeights value for the acquired class value from the labeler, and then checks whether there exists a dominant class value(s). If such class value(s) exist(s), the procedure returns this class value or a random class value among them.

3.3.2.4 Algorithm wIEAdjCost

Pseudocode for our main algorithm wIEAdjCost is given in Algorithm 3.6. The parameter $\lambda \in (0, 1]$ used on Lines 7 and 20 specifies the minimum value of $|O_g|/|O|$, i.e., the minimum fraction of original labelers that need to have well-estimated accuracies. Thus $\lambda = 1/|O|$ means that we merely insist on at least one labeler in O to have an accuracy that we estimate well, where $\lambda = 1$ means that we insist that we have good estimates of the accuracies of all labelers. The parameters in wIEAdjCost have the same meanings as in IEAdjCost.

3.4 Experimental Results

We tested IEAdjCost and wIEAdjCost on six data sets from the University of California-Irvine (UCI) repository (2010) and two data sets from Amazon Mechanical Turk (AMT) (Amazon Mechanical Turk, 2005; Snow et al., 2008). Section 3.4.1 shows the results of comparing IEAdjCost and wIEAdjCost with IEThresh and Repeated on the UCI data, and Section 3.4.2 presents results for the same algorithms on the AMT data.

3.4.1 Experimental Results on UCI Data Sets

Table 3.1 descibes the six UCI data sets we tested on: kr-vs-kp, mushroom, car, splice, nursery, and spambase. Data set spambase had continuous attributes, which we discretized

Algorithm 3.5 QueryLabelers(x^*,SumWeights, $O = \{o_1, \ldots, o_m\}$, weights for O)

v = the number of classes ;
allS $= \mathcal{Y} = \{0, \ldots, v-1\}$;
Compute the set of class values that have the highest SumWeights

$$\text{maxS} = \left\{ y : \text{SumWeights}(t) = \max_{y' \in \mathcal{Y}} \text{SumWeights}(y') \right\}$$

if The class values in maxS will have highest SumWeights no matter how the remaining labelers answer the query, i.e.,

$$\text{SumWeights}(y) > \max_{y' \in (\text{allS} - \text{maxS})} \text{SumWeights}(y') + \left(1 - \sum_{y''} \text{SumWeights}(y'') \right)$$

for $y \in$ maxS, $y' \neq y$, $y'' \in \mathcal{Y}$ **then**
  return a random element from maxS;
end if
$i = 1$;
while $i \leq m$ **do**
  Query labeler o_i and let y = answer from labeler i for labeling instance x^* ;
  Update SumWeights of class value y: SumWeights(y) = SumWeights(y)+weight(o_i) ;
  Compute the set of class values that have the highest SumWeights

$$\text{maxS} = \left\{ y : \text{SumWeights}(y) = \max_{y' \in \mathcal{Y}} \text{SumWeights}(y') \right\}$$

  if The class values in maxS will have highest SumWeights no matter how the remaining labelers answer the query, i.e.,

$$\text{SumWeights}(y) > \max_{y' \in (\text{allS} - \text{maxS})} \text{SumWeights}(y') + \left(1 - \sum_{y''} \text{SumWeights}(y'') \right)$$

  for $y \in$ maxS, $y' \neq y$, $y'' \in \mathcal{Y}$ **then**
    return a random element from maxS;
  end if
  $i = i + 1$;
end while

Algorithm 3.6 wIEAdjCost

1: $O_g = \emptyset$, $O_f = \emptyset$;
2: **Phase 1:** Initialize rewards for each labeler to $\{0, 1\}$;
3: Compute the upper and lower confidence interval for each labeler per Equations (3.2) and (3.3) ;
4: Pick the most uncertain unlabeled instance x^* for labeling per Equation (3.1) ;
5: Build O_ℓ per Equation (3.4) ;
6: **if** $O_f = \emptyset$ **then**
7: Compute O_r per Equation (3.5), each labeler in O_r is assigned a weight of $1/|O_r|$;
8: **else**
9: $O_r = O_f$;
10: Compute adjusted weight for each labeler $o_i \in O_r$ as $\text{Weight}'(o_i) = \text{Weight}(o_i)/\left(\sum_{o_j \in O_r} \text{Weight}(O_j)\right)$;
11: Assign each labeler $o_i \in O_r$ a new weight $\text{Weight}(o_i) = \text{Weight}'(o_i)$;
12: **end if**
13: Each labeler in $O_r - O_\ell$ is assigned a weight of 0 ;
14: Purchase labels for x^* from all labelers in O_ℓ, compute SumWeights(y) for each class $y \in \mathcal{Y}$;
15: $\bar{y}$ = QueryLabelers(x^*, SumWeights, $O_r - O_\ell$, weights for $O_r - O_\ell$) ;
16: Update the labeled training data $L = L \cup (x^*, \bar{y})$ and train a new classifier with L ;
17: Update the rewards of all labelers in $O_\ell \cup O_r$;
18: Compute the upper and lower confidence interval for each labeler per Equations (3.2) and (3.3) ;
19: Compute the set O_g of well-estimated labelers per Equation (3.6) ;
20: **if** $|O_g| \geq \lceil \lambda |O| \rceil$ and $O_f = \text{LabelersbywAdjCost}(O_g) \neq \emptyset$ **then**
21: GOTO Step 26 ;
22: **else**
23: GOTO Step 4 ;
24: **end if**
25: **Phase 2:** Compute weights for labelers in O_f ;
26: For any future instance x, $\bar{y}$ = QueryLabelers(x, SumWeights, o_f, weights for O_f) where SumWeights(y) = 0 for each $y \in \mathcal{Y}$, add $(x, \bar{y})$ to L, and train classifier until budget is exhausted.

using the package

`weka.filters.supervised.attribute.Discretize` from Weka (2009).

Table 3.1: UCI Data Sets.

Data set	Instances	Number of Attributes	Classes
kr-vs-kp	3196	35	2
mushroom	8124	22	2
car	1728	4	4
nursery	12960	8	3
splice	3190	62	5
spambase	4601	57	2

We repeated the following procedure 10 times for each data set. First, we partitioned the data set into 80% training data and 20% test data. Then as an initialization step, at the start of each run we pre-labeled one randomly selected training instance with its correct label, moved that labeled instance from U to L, and trained the initial naïve Bayes classifier on L. As more instances were added to L on Line 8 in Algorithm 3.3 or Lines 26 and 16 in Algorithm 3.6, the naïve Bayes classifier was updated.

Each time labeler o_i was asked to label an instance, it returned the true label from the data set with probability $1 - \eta_i$ and the incorrect label with probability η_i. We set the parameter $R = 0.99$ and ran each algorithm until 900 instances were purchased. This allowed us to witness the asymptotic behavior of each algorithm.

3.4.1.1 Uniform Accuracies

In our first set of experiments, we repeated the following process five times. We instantiated 50 labelers, the accuracies of which were randomly chosen from $(1/v, 1]$, where v is the number of classes of the data set, and the costs of which were randomly chosen from $\{1, \ldots, 30\}$. Table 3.2 shows the accuracies of the 50 labelers for experiments with each data set. (We

Table 3.2: The accuracies in Section 3.4.1.1 for 50 labelers whose accuracy were randomly chosen from $(1/v, 1]$ where v is the number of class values.

Datasets	Accuracies of the labelers
kr-vs-kp mushroom spambase	0.99, 0.99, 0.98, 0.98, 0.97, 0.97, 0.96, 0.96, 0.95, 0.93 0.92, 0.92, 0.92, 0.89, 0.88, 0.88, 0.88, 0.87, 0.85, 0.82, 0.81, 0.81, 0.8, 0.79, 0.78, 0.77, 0.76, 0.76, 0.76, 0.75, 0.74, 0.73, 0.71, 0.7, 0.68, 0.65, 0.64, 0.63, 0.63, 0.62, 0.6, 0.59, 0.59, 0.59, 0.58, 0.58, 0.57, 0.56, 0.52, 0.52
nursery	1, 1, 1, 0.97, 0.97, 0.97, 0.95, 0.95, 0.95, 0.95, 0.94, 0.92, 0.92, 0.92, 0.91, 0.9, 0.89, 0.88, 0.87, 0.85, 0.83, 0.83, 0.8, 0.8, 0.79, 0.78, 0.78, 0.76, 0.7, 0.66, 0.65, 0.61, 0.6, 0.57, 0.55, 0.55, 0.55, 0.53, 0.52, 0.5, 0.43, 0.43, 0.41, 0.41, 0.41, 0.41, 0.39, 0.39, 0.39, 0.36
car	1, 0.98, 0.97, 0.96, 0.95, 0.93, 0.89, 0.88, 0.87, 0.87, 0.83, 0.83, 0.77, 0.77, 0.76, 0.72, 0.72, 0.71, 0.7, 0.68, 0.66, 0.65, 0.64, 0.62, 0.6, 0.59, 0.58, 0.57, 0.54, 0.54, 0.54, 0.53, 0.53, 0.51, 0.48, 0.47, 0.41, 0.41, 0.41, 0.4, 0.38, 0.37, 0.34, 0.34, 0.34, 0.31, 0.31, 0.27, 0.27, 0.27
splice	0.99, 0.99, 0.96, 0.95, 0.92, 0.89, 0.88, 0.85, 0.84, 0.83, 0.76, 0.76, 0.73, 0.71, 0.7, 0.7, 0.68, 0.67, 0.67, 0.66, 0.64, 0.63, 0.63, 0.63, 0.61, 0.61, 0.61, 0.6, 0.58, 0.58, 0.57, 0.57, 0.54, 0.53, 0.53, 0.52, 0.52, 0.52, 0.5, 0.48, 0.46, 0.44, 0.44, 0.39, 0.38, 0.37, 0.32, 0.27, 0.25, 0.24

used the same set of labelers for kr-vs-kp, mushroom, and spambase since they are all binary class data sets.)

We tested IEAdjCost and wIEAdjCost with $\lambda \in \{0.02, 0.25, 0.5, 0.75, 1\}$, $\epsilon = 0.8$, and $\delta = 0.2$. We compared its results to those from IEThresh with $\epsilon = 0.8$ and Repeated on the six UCI data sets. For each collection of labelers, we ran 10 separate test/training combinations as described earlier. Thus we ended up with five sets of curves (one per set of 50 labelers), each an average of 10 runs of the algorithms.

Figures 3.2 and 3.3 show[6], for each data set, the cost required to reach a particular classification accuracy for each of the algorithms ("Baseline" is the classification accuracy of

[6]We got similar results for all five sets of curves from the five sets of labelers, so we just present one such set in the figure.

training a naïve Bayes classifier on the entire training set with all labels correctly specified).

From the figures, we can see that IEAdjCost consistently requires significantly less cost to achieve high accuracies than IEThresh and Repeated. Also, wIEAdjCost consistently requires significantly less cost to achieve high accuracies than IEAdjCost.

In results not shown, we also set $\epsilon = 0.7$ and $\epsilon = 0.9$, and found that the algorithms' performance was very similar. Again, the advantages of IEAdjCost over IEThresh and Repeated, as well as the advantages of wIEAdjCost over IEAdjCost still exist.

In other results not shown, we set $\delta = 0.1$ and $\delta = 0.3$. We found that increasing δ to 0.3 improved IEAdjCost and wIEAdjCost's performance for all values of λ, and decreasing it to 0.1 made them worse. Future work is to further tune δ to see how much more performance improvement we can achieve, perhaps dynamically choosing its value.

In conclusion, we found that when the fifty labeler costs and labeler accuracies are uniformly at random chosen from the ranges $\{1, \ldots, 30\}$ and $(1/v, 1]$ respectively, then IEAdjCost clearly required less cost to achieve high classification accuracies than IEThresh and Repeated, and wIEAdjCost clearly required less cost to achieve high classification accuracies than IEAdjCost. Table 3.5 shows the specific savings of the cost of wIEAdjCost over IEAdjCost, and also shows that for binary data sets kr-vs-kp, mushroom, and spambase, wIEAdjCost finds a smaller O_f and a cheaper total cost of O_f than IEAdjCost does.

Figures 3.2 and 3.3 show how much cost is incurred by each algorithm to achieve a certain accuracy. To instead consider how well each method can perform under a fixed budget, we noted the minimum cost across all algorithms required to achieve the baseline accuracy. (For each dataset, this was achieved by wIEAdjCost with $\lambda = 0.02$.) Table 3.3 reports this budget for each dataset, along with the accuracies of wIEAdjCost, IEAdjCost, IEThresh and Repeated, when limited to that budget.

As shown in Table 3.3, wIEAdjCost ($\lambda = 0.02$) significantly outperforms Repeated, IEThresh, IEAdjCost (with $\lambda \in \{0.02, 0.25, 0.5, 0.75, 1\}$), wIEAdjCost (with $\lambda \in \{0.25, 0.5, 0.75, 1\}$))

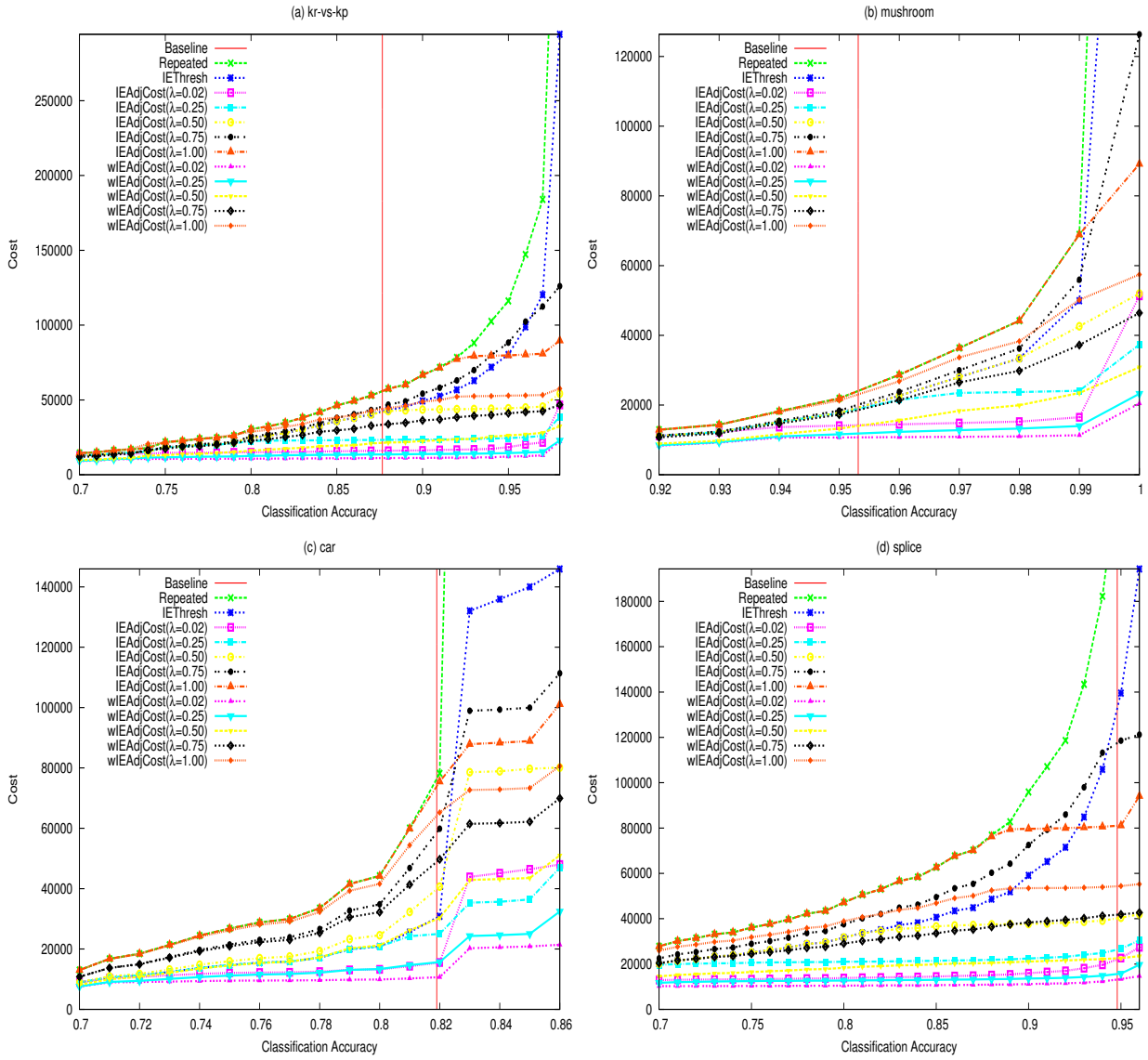


Figure 3.2: Cost required for each algorithm to achieve specific classification accuracies on UCI data sets kr-vs-kp, mushroom, car, and splice. Fifty labelers were available, each with a cost randomly chosen from $\{1, \ldots, 30\}$ and accuracy randomly chosen from $(1/v, 1]$. Values of parameters were $R = 0.99$, $\delta = 0.2$, and $\epsilon = 0.8$.

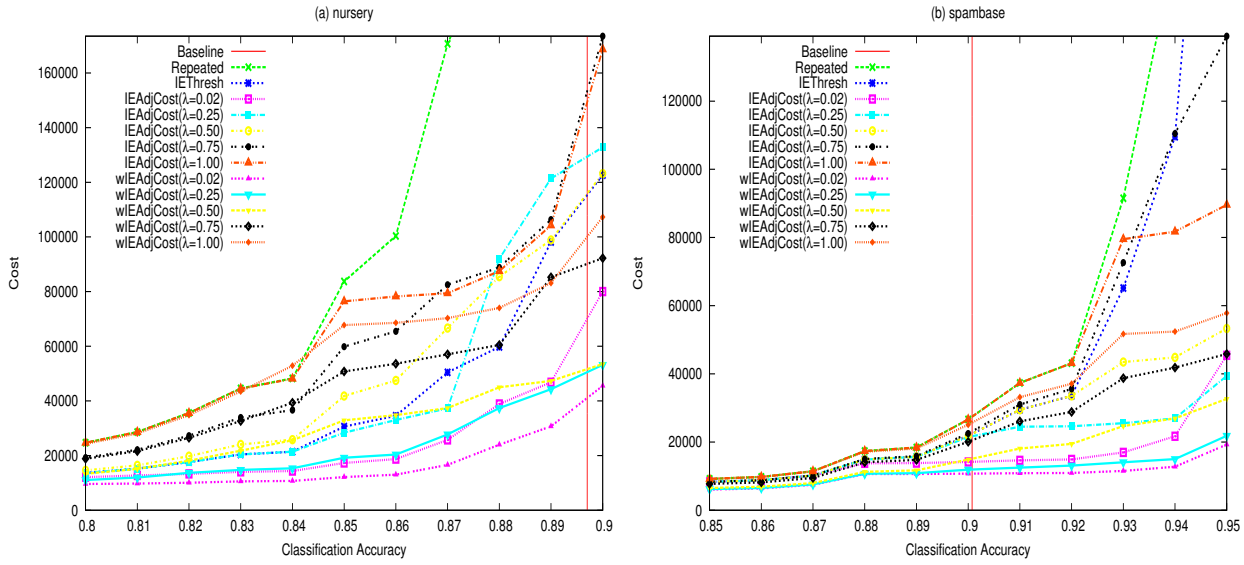


Figure 3.3: Cost required for each algorithm to achieve specific classification accuracies on UCI data sets nursery and spambase. Fifty labelers were available, each with a cost randomly chosen from $\{1, \ldots, 30\}$ and accuracy randomly chosen from $(1/v, 1]$. Values of parameters were $R = 0.99$, $\delta = 0.2$, and $\epsilon = 0.8$.

at a confidence level $p \leq 0.02$ for data sets kr-vs-kp, car, splice, and spambase.

wIEAdjCost ($\lambda = 0.25$) significantly outperforms Repeated, IEThresh, IEAdjCost (with $\lambda \in \{0.02, 0.25, 0.5, 0.75, 1\}$) (for dataset nursery, $\lambda = 0.02$ is excluded), wIEAdjCost (with $\lambda \in \{0.5, 0.75, 1\}$) (for dataset mushroom, $\lambda = 0.5$ is excluded) at a confidence level $p \leq 0.006$ for data sets splice and nursery.

3.4.1.2 Setting λ

Looking at Figures 3.2 and 3.3 and Table 3.3, one might assume that smaller values of λ for IEAdjCost and wIEAdjCost will always yield superior results. That is not necessarily the case. Since λ determines what fraction of all the labelers are considered at each stage, the following guidelines apply to setting its value.

1. If one believes that there exists a labeler that is cheap and has high accuracy, then

Table 3.3: Classification accuracy after spending a limited budget , using 50 labelers, each with a cost from $\{1, \ldots, 30\}$ and accuracy from $(1/v, 1]$. $R = 0.99$, $\delta = 0.2$ and $\epsilon = 0.8$. **Boldface** indicates a statistically significant advantage (at a 95% confidence level) of wIEAdjCost ($\lambda = 0.02$) over all other algorithms. *Italics* indicates a statistically significant advantage (at a 95% confidence level) of wIEAdjCost ($\lambda = 0.25$), wIEAdjCost ($\lambda = 0.5$), and IEAdjCost ($\lambda = 0.02$) over all other algorithms.

	Budget	Classification Accuracy											
		Repeated	IEThresh	IEAdjCost					wIEAdjCost				
				$\lambda = \frac{1}{50}$	$\lambda = \frac{1}{4}$	$\lambda = \frac{1}{2}$	$\lambda = \frac{3}{4}$	$\lambda = 1$	$\lambda = \frac{1}{50}$	$\lambda = \frac{1}{4}$	$\lambda = \frac{1}{2}$	$\lambda = \frac{3}{4}$	$\lambda = 1$
kr-vs-kp	11134	0.67	0.68	0.68	0.68	0.68	0.68	0.67	**0.88**	*0.73*	0.73	0.69	0.67
mushroom	10706	0.90	0.92	0.92	0.92	0.92	0.92	0.90	**0.96**	*0.94*	*0.94*	0.92	0.90
car	10667	0.70	0.71	0.72	0.71	0.71	0.70	0.70	**0.82**	*0.74*	0.72	0.70	0.70
splice	13351	0.60	0.64	0.75	0.64	0.64	0.63	0.60	**0.95**	*0.89*	0.67	0.63	0.60
nursery	45533	0.83	0.87	*0.89*	0.87	0.86	0.84	0.83	**0.90**	*0.89*	0.88	0.85	0.83
spambase	10734	0.87	0.87	0.87	0.87	0.87	0.87	0.87	**0.91**	*0.89*	0.88	0.87	0.87

the smaller λ is, the more cost one can save on exploring. In our experiments of Section 3.4.1.1, it is the case that some highly accurate labelers (Table 3.2) have low cost. As expected, low values of λ yielded high-accuracy classifiers at a lower cost than larger values of λ, as shown in Table 3.3.

2. If one believes that the only highly accurate labelers are expensive, then increasing λ will allow our algorithm to consider cheaper alternatives, which when taken collectively will perform as well as the expensive labelers but for less cost. Figure 3.4 shows an experiment in which one labeler has accuracy 0.8 and cost 100, and forty-nine labelers have accuracy 0.7 and cost 1 for algorithm IEAdjCost with different λ values. In this case, IEAdjCost with $\lambda = 0.25$ (instead of $\lambda = 0.02$) performs the best. Figure 3.5 shows an experiment in which one labeler has accuracy 0.8 and cost 100, and forty-nine labelers have accuracy 0.7 and cost 1 for algorithm wIEAdjCost with different λ values. Here, we see that wIEAdjCost with $\lambda = 0.25$ (instead of $\lambda = 0.02$) performs the best.

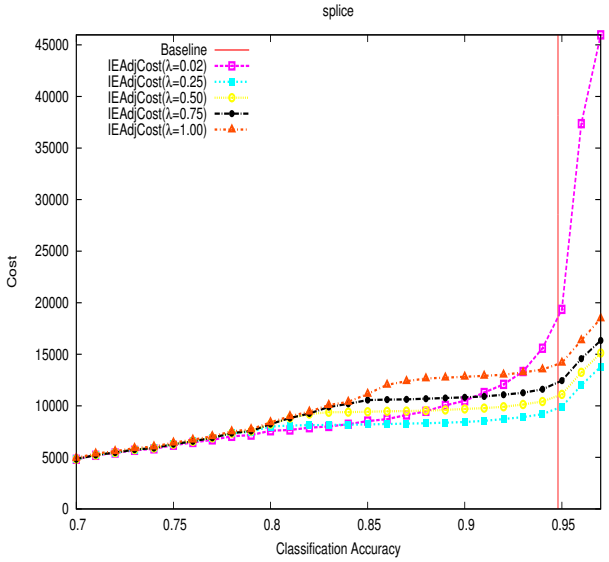


Figure 3.4: Total cost required for IEAdjCost (with $\lambda \in \{0.02, 0.25, 0.5, 0.75, 1\}$ to achieve specific classification accuracies on the UCI data set splice. Fifty labelers were available, 1 with accuracy 0.85 and cost 100, 49 labelers with accuracy 0.8 and cost 1. Values of parameters were $R = 0.99$, $\delta = 0.2$, and $\epsilon = 0.8$.

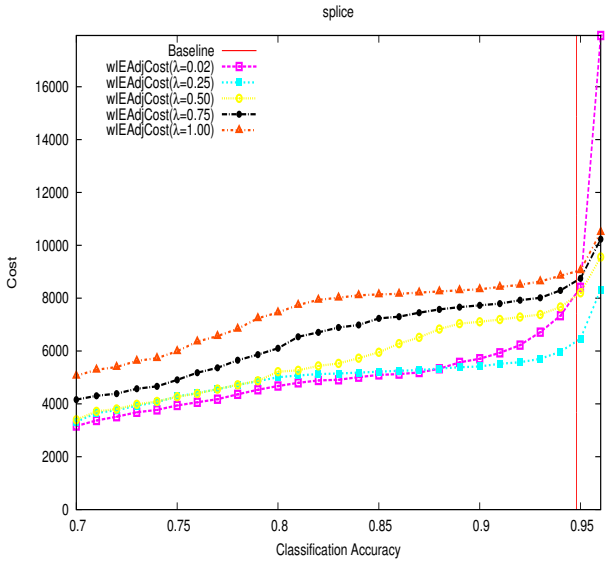


Figure 3.5: Total cost required for wIEAdjCost (with $\lambda \in \{0.02, 0.25, 0.5, 0.75, 1\}$ to achieve specific classification accuracies on the UCI data set splice. Fifty labelers were available, 1 with accuracy 0.85 and cost 100, 49 labelers with accuracy 0.8 and cost 1. Values of parameters were $R = 0.99$, $\delta = 0.2$, and $\epsilon = 0.8$.

3.4.1.3 Low Accuracy Labelers

In our experimental results so far, high-accuracy labelers were available, as shown in Table 3.2.

Procedures LabelersbyAdjCost and LabelersbywAdjCost return a set of labelers O_f whose size is small (because usually several high accuracy labelers can reach a combined accuracy of R). In case the labelers do not have high accuracies, the two procedures LabelersbyAdjCost and LabelersbywAdjCost return a large set of labelers (e.g., more than 20). In this case, we want to compare which procedure is more efficient, i.e., which can return a set of labelers in less time. We tested on the six UCI data sets, using one hundred labelers, each with an accuracy of 0.65. Figures 3.6 and 3.7 show the cost to reach a specified classification accuracy for algorithms Repeated, IEThresh, IEAdjCost and wIEAdjCost. From these figures, we can see that IEAdjCost saves cost compared to IEThresh and Repeated. wIEAdjCost further saves cost compared to IEAdjCost. To see how well each method can perform under a fixed budget, we noted the minimum cost across all algorithms required to achieve the baseline accuracy. (For data sets kr-vs-kp, mushroom, car, splice, nursery, and spambase, this was achieved by wIEAdjCost with $\lambda = 0.5$, $\lambda = 0.01$, $\lambda = 0.01$, $\lambda = 0.25$, $\lambda = 0.5$, and $\lambda = 0.01$, respectively.) Table 3.4 reports this number for the above data sets, along with wIEAdjCost, IEAdjCost, IEThresh and Repeated, the accuracy reached from spending only the budget used by wIEAdjCost (with $\lambda = 0.5$, $\lambda = 0.01$, $\lambda = 0.01$, $\lambda = 0.25$, $\lambda = 0.5$, and $\lambda = 0.01$, respectively) to reach its baseline.

As shown in Table 3.4, wIEAdjCost ($\lambda \in \{0.01, 0.25\}$ for mushroom and car and $\lambda = 0.01$ for splice) significantly outperforms other algorithms (except wIEAdjCost with $\lambda = 0.5$) at a confidence level $p \leq 0.01$ for mushroom, car, and splice.

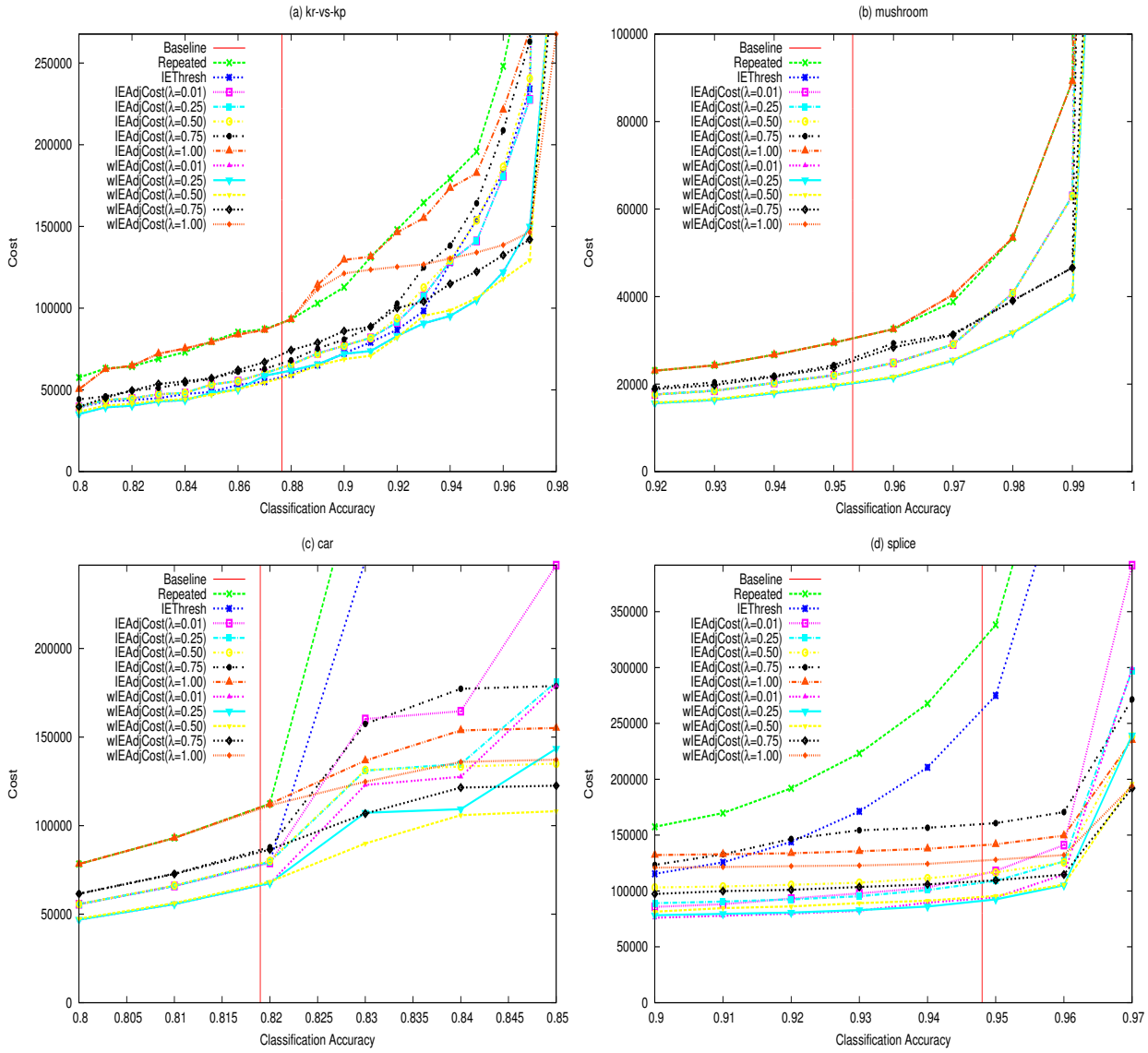


Figure 3.6: Cost required for each algorithm to achieve specific classification accuracies on UCI data sets kr-vs-kp, mushroom, car, and splice. One hundred labelers were available. Each labeler has an accuracy of 0.65 and a cost randomly chosen from $\{1, \ldots, 30\}$. Values of parameters were $R = 0.99$, $\delta = 0.2$, and $\epsilon = 0.8$.

Table 3.4: Classification accuracy after spending a limited budget, using one hundred labelers, each with an accuracy 0.65 and a cost from $\{1, \ldots, 30\}$ and accuracy from $(1/v, 1]$. $R = 0.99$, $\delta = 0.2$ and $\epsilon = 0.8$. **Boldface** indicates a statistically significant advantage (at a 95% confidence level) of wIEAdjCostwith specific parameters over all other algorithms.

	Budget	Classification Accuracy											
		Repeated	IEThresh	IEAdjCost					wIEAdjCost				
				$\lambda = \frac{1}{100}$	$\lambda = \frac{1}{4}$	$\lambda = \frac{1}{2}$	$\lambda = \frac{3}{4}$	$\lambda = 1$	$\lambda = \frac{1}{100}$	$\lambda = \frac{1}{4}$	$\lambda = \frac{1}{2}$	$\lambda = \frac{3}{4}$	$\lambda = 1$
kr-vs-kp	58730	0.803	0.858	0.866	0.866	0.867	0.858	0.808	0.868	0.868	**0.875**	0.852	0.808
mushroom	19576	0.901	0.940	0.940	0.940	0.940	0.928	0.901	**0.955**	**0.955**	0.952	0.932	0.901
car	67418	0.786	0.812	0.812	0.812	0.812	0.804	0.786	**0.820**	**0.820**	0.818	0.804	0.786
splice	92480	0.600	0.639	0.740	0.639	0.639	0.623	0.600	**0.952**	0.852	0.673	0.626	0.600
nursery	113919	0.853	0.861	0.873	0.880	0.875	0.859	0.853	0.888	**0.891**	**0.892**	0.883	0.851
spambase	29250	0.880	0.891	0.891	0.891	0.890	0.890	0.880	**0.894**	**0.894**	0.892	0.890	0.880

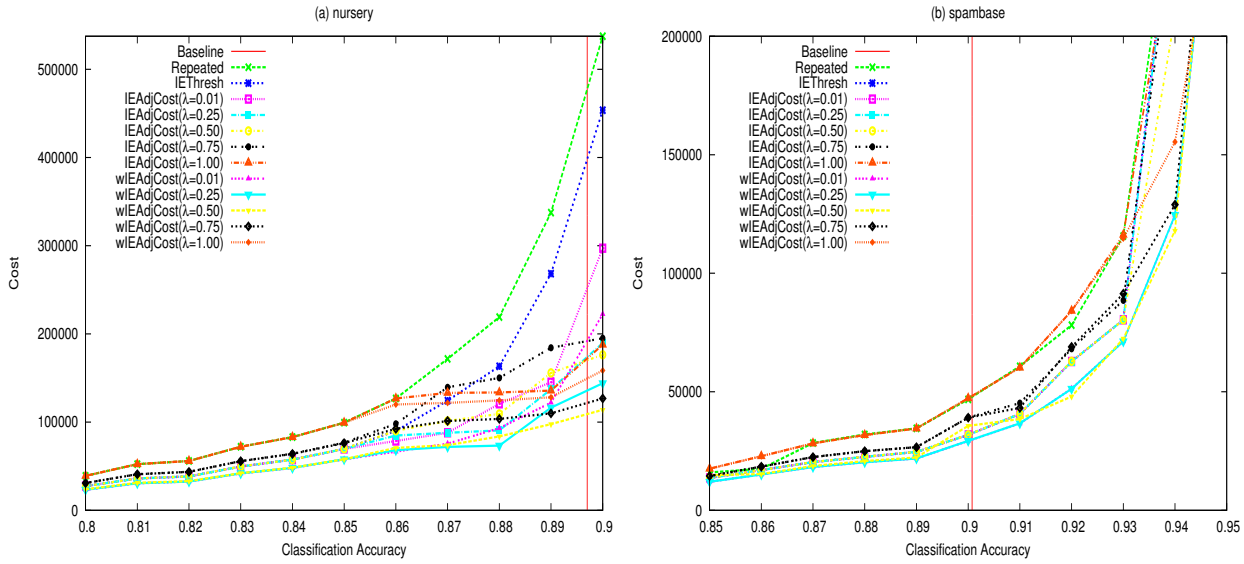


Figure 3.7: Cost required for each algorithm to achieve specific classification accuracies on UCI data sets nursery and spambase. One hundred labelers were available. Each labeler has an accuracy of 0.65 and a cost randomly chosen from $\{1, \ldots, 30\}$. Values of parameters were $R = 0.99$, $\delta = 0.2$, and $\epsilon = 0.8$.

3.4.1.4 Conclusions on UCI Data Sets

Besides Figures 3.2, 3.3, 3.6, and 3.7, which show the cost required for each algorithm to achieve specific classification accuracies on the six UCI data sets, we also present Tables 3.5 and 3.6, which show the savings of both cost and time when comparing wIEAdjCost with

IEAdjCost. Table 3.5 shows the savings in cost and run time of wIEAdjCost over IEAdjCost in reaching the baseline accuracy on the six UCI data sets where 50 labelers were available, each labeler has an accuracy randomly chosen from $(1/v, 1]$ (where v is the number of class values) and a cost randomly chosen from $\{1, \ldots, 30\}$. Table 3.6 shows the savings in cost and run time of wIEAdjCost over IEAdjCost in reaching the baseline accuracy on the six UCI data sets where one hundred labelers were available, each labeler has an accuracy of 0.65 and a cost randomly chosen from $\{1, \ldots, 30\}$. We have the following observations of the experimental results on the six UCI data sets.

For 50 labelers whose costs are randomly chosen from $\{1, \ldots, 30\}$ and whose accuracies are randomly chosen from $(1/v, 1]$: (1) All algorithms achieved the baseline accuracy eventually, but some required significantly more cost to do so[7]. (2) IEAdjCost clearly required less cost to achieve high classification accuracies than IEThresh and Repeated. (3) Varying ϵ did not seem to affect the results very much, though larger values of δ did improve our algorithms' performance somewhat. (4) For IEAdjCost and wIEAdjCost with the same parameter values, wIEAdjCost often finds a O_f with smaller combined cost than that of IEAdjCost. (5) wIEAdjCost always saves cost compared to IEAdjCost. (6) The computation of combined accuracy of only a few labelers is fast, so there is no obvious time savings of wIEAdjCost compared to IEAdjCost. (7) When using a limited budget (set as the minimum budget to reach the baseline accuracy across all algorithms), wIEAdjCost with $\lambda = 0.02$ always statistically significantly outperformed wIEAdjCost with $\lambda \in \{0.25, 0.5, 0.75, 1\}$, IEAdjCost with $\lambda \in \{0.02, 0.25, 0.5, 0.75, 1\}$, IEThresh, and Repeated.

For one hundred labelers whose costs are randomly chosen from $\{1, \ldots, 30\}$ and whose accuracies are 0.65: (1) The advantage of IEAdjCost over IEThresh and Repeated

[7]The number of instances labeled by each algorithm were roughly the same across all experiments; cost was the only thing that varied.

Table 3.5: Savings in cost and run time of wIEAdjCost over IEAdjCost in reaching the baseline accuracy on six UCI data sets. Fifty labelers were available. Each labeler has an accuracy randomly chosen from $(1/v, 1]$ and a cost randomly chosen from $\{1, \ldots, 30\}$.

dataset	λ	costs	saved cost	time (ms)	saved time	$\lvert O_f \rvert$	cost(O_f)
kr-vs-kp	0.02	15990.4 vs 11133.7	30%	6178 vs 6001	2%	4.0 vs 3.0	36.7 vs 24.2
kr-vs-kp	0.25	23315.9 vs 13573.4	41%	6036 vs 5824	3%	4.0 vs 3.4	18.5 vs 17.1
kr-vs-kp	0.5	42578.0 vs 21507.6	49%	5919 vs 5963	0%	2.4 vs 2.6	13.0 vs 8.7
kr-vs-kp	0.75	46844.5 vs 33775.9	27%	6044 vs 5985	0%	2.6 vs 2.6	12.1 vs 8.7
kr-vs-kp	1	57429.0 vs 44551.7	22%	6345 vs 6289	0%	2.8 vs 3.1	13.0 vs 10.2
mushroom	0.02	14097.5 vs 10705.9	24%	14187 vs 14573	−2%	4.0 vs 3.0	42.6 vs 25.4
mushroom	0.25	17692.9 vs 11595.2	34%	14347 vs 14222	0%	4.0 vs 3.6	16.1 vs 16.9
mushroom	0.5	17713.9 vs 13251.1	25%	14909 vs 14346	3%	2.4 vs 2.1	11.5 vs 6.6
mushroom	0.75	18385.6 vs 17238.4	6%	13610 vs 14487	−6%	2.8 vs 2.3	11.8 vs 8.4
mushroom	1	21906.6 vs 21304.8	2%	14279 vs 14077	1%	2.4 vs 2.3	12.2 vs 8.4
car	0.02	15538.5 vs 10667.1	31%	2582 vs 2421	6%	4.0 vs 3.0	56.3 vs 34.8
car	0.25	25011.9 vs 15658.1	37%	2616 vs 2489	4%	3.0 vs 1.3	19.9 vs 14.6
car	0.5	40676.6 vs 30299.0	25%	2555 vs 2529	1%	2.2 vs 1.3	19.7 vs 14.6
car	0.75	59868.0 vs 49705.1	16%	2526 vs 2645	−4%	2.2 vs 1.6	21.0 vs 15.9
car	1	75450.3 vs 65236.8	13%	2514 vs 2621	−4%	2.0 vs 1.6	22.0 vs 14.9
splice	0.02	22650.6 vs 13351.0	41%	25377 vs 25749	−1%	4.0 vs 3.0	37.4 vs 22.2
splice	0.25	26678.2 vs 15775.2	40%	25476 vs 26126	−2%	4.0 vs 3.3	25.0 vs 20.4
splice	0.5	40413.2 vs 22632.7	43%	26176 vs 25112	4%	3.0 vs 1.0	12.5 vs 4.8
splice	0.75	118571.7 vs 41877.4	64%	25389 vs 25627	0%	2.0 vs 1.0	8.8 vs 4.8
splice	1	81093.3 vs 54418.1	32%	25793 vs 26164	−1%	2.0 vs 1.0	8.6 vs 4.8
nursery	0.02	80023.1 vs 45532.8	43%	79833 vs 79983	0%	4.0 vs 3.0	69.8 vs 63.0
nursery	0.25	132807.0 vs 53211.5	59%	83168 vs 81102	2%	2.8 vs 3.5	34.5 vs 34.3
nursery	0.5	123182.8 vs 53499.4	56%	81263 vs 80889	0%	2.3 vs 4.0	33.7 vs 27.6
nursery	0.75	173465.8 vs 92235.7	46%	80135 vs 80372	0%	2.3 vs 4.2	33.7 vs 27.2
nursery	1	168655.8 vs 107306.0	36%	81634 vs 78581	3%	2.8 vs 4.2	31.8 vs 27.1
spambase	0.02	14181.0 vs 10734.4	24%	13389 vs 13473	0%	4.0 vs 3.0	36.0 vs 20.9
spambase	0.25	21486.2 vs 11894.8	44%	13391 vs 13450	0%	4.0 vs 3.0	17.4 vs 14.5
spambase	0.5	21839.8 vs 14868.3	31%	13750 vs 13351	2%	2.8 vs 2.8	12.4 vs 10.4
spambase	0.75	22512.4 vs 20074.8	10%	13340 vs 13427	0%	2.6 vs 2.6	11.1 vs 7.6
spambase	1	26636.3 vs 25059.3	5%	13947 vs 13595	2%	2.6 vs 3.4	12.8 vs 12.0

Table 3.6: Savings in cost and run time of wIEAdjCost over IEAdjCost in reaching the baseline accuracy on six UCI data sets. One hundred labelers were available. Each labeler has an accuracy of 0.65 and a cost randomly chosen from $\{1, \ldots, 30\}$.

dataset	λ	costs	saved cost	time (ms)	saved time	$\lvert O_f \rvert$	cost(O_f)
kr-vs-kp	0.01	65432.7 vs 61829.9	5%	80891 vs 6040	92%	67.0 vs 33.3	895.4 vs 450.3
kr-vs-kp	0.25	65432.7 vs 61829.9	5%	80592 vs 6175	92%	67.0 vs 33.3	895.4 vs 450.3
kr-vs-kp	0.5	65472.6 vs 58729.9	10%	66285 vs 6129	90%	70.0 vs 40.2	944.4 vs 450.5
kr-vs-kp	0.75	68113.2 vs 74311.0	−9%	41752 vs 6352	84%	64.0 vs 42.3	850.8 vs 371.0
kr-vs-kp	1	93203.2 vs 93145.3	0%	37127 vs 6199	83%	57.8 vs 36.0	766.6 vs 248.7
mushroom	0.01	22020.8 vs 19576.4	11%	81534 vs 13894	82%	66.0 vs 33.8	885.6 vs 440.5
mushroom	0.25	22020.8 vs 19578.4	11%	81482 vs 13955	82%	66.0 vs 33.6	885.6 vs 438.5
mushroom	0.5	22020.8 vs 19803.6	10%	67190 vs 14478	78%	65.0 vs 39.4	873.5 vs 433.1
mushroom	0.75	24338.6 vs 23789.1	2%	44507 vs 14052	68%	68.0 vs 45.1	931.2 vs 417.9
mushroom	1	29465.9 vs 29465.9	0%	79679 vs 14401	81%	65.6 vs 36.1	872.5 vs 252.5
car	0.01	78982.2 vs 67418.6	14%	4348 vs 2435	43%	11.3 vs 10.9	150.5 vs 131.1
car	0.25	79555.2 vs 67526.4	15%	8772 vs 2560	70%	12.5 vs 12.3	86.1 vs 83.9
car	0.5	80098.6 vs 68266.9	14%	34916 vs 2622	92%	13.5 vs 14.0	55.1 vs 57.0
car	0.75	87792.0 vs 86488.7	1%	102823 vs 2560	97%	14.5 vs 14.3	48.2 vs 48.6
car	1	112311.2 vs 111429.3	0%	131097 vs 2557	98%	14.3 vs 14.0	41.4 vs 39.3
splice	0.01	117993.6 vs 94308.6	20%	27555 vs 25651	6%	13.9 vs 13.6	181.2 vs 179.7
splice	0.25	109062.2 vs 92480.1	15%	32097 vs 25539	20%	15.0 vs 15.1	123.8 vs 126.1
splice	0.5	116665.9 vs 95621.9	18%	80259 vs 25919	67%	17.1 vs 18.2	80.4 vs 87.0
splice	0.75	160772.1 vs 109575.4	31%	310089 vs 26251	91%	18.7 vs 18.8	71.4 vs 72.3
splice	1	141752.5 vs 127978.6	9%	170918 vs 25984	84%	17.9 vs 17.5	61.0 vs 58.3
nursery	0.01	297127.1 vs 222440.9	25%	50283 vs 48650	3%	10.2 vs 9.7	128.0 vs 125.8
nursery	0.25	189707.1 vs 144072.0	24%	51556 vs 48342	6%	10.8 vs 10.8	61.5 vs 58.4
nursery	0.5	176304.0 vs 113918.6	35%	67001 vs 48393	27%	11.8 vs 12.3	44.8 vs 46.8
nursery	0.75	195067.6 vs 126802.4	34%	131937 vs 47724	63%	12.6 vs 12.1	38.0 vs 36.5
nursery	1	187686.3 vs 158518.9	15%	135842 vs 47848	64%	12.3 vs 12.1	33.0 vs 30.3
spambase	0.01	31775.6 vs 29249.8	7%	80143 vs 13840	82%	73.0 vs 36.6	973.7 vs 452.0
spambase	0.25	31775.6 vs 29281.1	7%	79297 vs 13607	82%	73.0 vs 36.6	973.7 vs 454.3
spambase	0.5	31832.0 vs 35648.2	−11%	74475 vs 14131	81%	73.0 vs 41.9	970.6 vs 455.2
spambase	0.75	39114.1 vs 39060.4	0%	48441 vs 13971	71%	71.0 vs 46.9	947.7 vs 433.0
spambase	1	47296.1 vs 47296.1	0%	52559 vs 13324	74%	61.2 vs 37.7	807.0 vs 264.5

is still obvious. (2) When needing many labelers to reach required accuracy, wIEAdjCost can save significant time compared to IEAdjCost. This is because wIEAdjCost adopts a much simpler (linear-time) method to compute combined weights while IEAdjCost adopts an exponential-time computation of combined accuracy for small sets of labelers (for no more than 20 labelers in our experiment). (3) For binary-class data sets (kr-vs-kp, mushroom, spambase) and IEAdjCost and wIEAdjCost with the same parameter values, IEAdjCost found a larger O_f than wIEAdjCost and and the total cost of the labelers in O_f found by IEAdjCost was bigger than by wIEAdjCost. This is because IEAdjCost adopts a conservative estimation of combined accuracy when the number of labelers exceeds a certain number (20 in our experiment). (4) wIEAdjCost almost always saves cost compared to IEAdjCost. Even for dataset kr-vs-kp ($\lambda = 0.75$) and dataset spambase ($\lambda = 0.5$), when reaching the baseline, IEAdjCost spends less, but after the baseline, eventually wIEAdjCost still needs less cost than IEAdjCost. (5) When using a limited budget (set as the minimum budget to reach the baseline accuracy across all algorithms), wIEAdjCost (with $\lambda = 0.01$ or 0.25 or 0.5) nearly always statistically significantly outperformed Repeated and IEThresh, IEAdjCost with $\lambda \in \{0.01, 0.25, 0.5, 0.75, 1\}$ and wIEAdjCost with some $\lambda \in \{0.01, 0.25, 0.5, 0.75, 1\}$.

3.4.2 Experiments on AMT data sets

We then tested our algorithms on two other data sets: RTE (Recognizing Textual Entailment) and TEMP (Temporal Event Recognition) (Snow et al., 2008) from Amazon Mechanical Turk (AMT). Each instance in RTE is a sentence-sentence pair and the annotators are asked to decide whether the second sentence can be inferred from the first, answering "true" or "false." The original RTE data set has 800 instances and 165 annotators. Each instance in TEMP is a short article including two events and the annotators need to judge which of the two events happens first. The original data set has 462 instances and 76 annotators. In

Table 3.7: The size and the labeler accuracies for the AMT data sets.

data	size	labeler accuracies
RTE	800	0.50, 0.51, 0.51, 0.53, 0.56, 0.58, 0.60, 0.65, 0.73, 0.78, 0.80, 0.80, 0.81, 0.82, 0.82, 0.83, 0.83, 0.83, 0.85, 0.85, 0.85, 0.85, 0.85, 0.85, 0.85, 0.88, 0.88, 0.89, 0.90, 0.90, 0.90, 0.91, 0.91, 0.92, 0.92, 0.93, 0.93, 0.93, 0.93, 0.95
TEMP	462	0.44, 0.44, 0.50, 0.54, 0.60, 0.70, 0.77, 0.77, 0.78, 0.80, 0.85, 0.89, 0.90, 0.90, 0.91, 0.92, 0.92, 0.92, 0.92, 0.93, 0.93, 0.93, 0.93, 0.93, 0.93, 0.94, 0.94, 0.95, 0.98, 0.98, 0.98

our experiments, we used all instances from each data set, but when deciding which labelers to use, we encountered the problem that for each data set, not every annotator labels every instance. Thus we selected as labelers only those labelers who labeled at least 30 instances, leaving 40 labelers for RTE and 31 for TEMP. For each such labeler o_i, we used the instances labeled by it to set[8] its accuracy a_i. Then when running the algorithms in our experiments, whenever labeler o_i is asked to label an instance x, we first check to see if o_i labeled x in the original data set. If this is the case, then in our experiments we have o_i return the label that corresponds to it in the original data. If instead o_i did not label x in the original data set, we then simulate o_i by having it return the correct label with probability a_i. Table 3.7 lists the number of instances and the accuracies of the labelers that remained after this preprocessing step. Note that one labeler for RTE and two labelers for TEMP ended up with accuracy < 0.5. We found that in our experiments, both IEAdjCost and wIEAdjCost was able to tolerate this violation of our assumptions, which is unsurprising since such a small fraction of labelers had accuracy < 0.5.

Unlike with the UCI data sets, we did not train classifiers for AMT data because they are represented as sentences as opposed to attribute vectors. Thus, rather than training a classifier, we adapted the approach used by Donmez et al. (2009): for IEAdjCost (wIEAdjCost),

[8]This value that we compute as its accuracy is of course just an estimate based on labeled data. But for the purposes of our simulation, we use this value as its "true" accuracy a_i.

we had each labeler in O_f make its individual prediction based on the procedure outlined above and then returned the "classifier's" prediction as a majority (weighted majority) vote of these labelers.

As done in Section 3.4.1, we ran 10 rounds of experiments. In each round we randomly chose 80% of the instances for training data, and used the remaining 20% as testing data. We ran our experiments with two cost models: one with unit cost for each labeler and one with costs randomly selected from $\{1, \ldots, 100\}$. For algorithm IEThresh, we set $\epsilon = 0.8$. For algorithm IEAdjCost and wIEAdjCost, we set $\epsilon = 0.8$, $\delta = 0.2$, $R = 0.99$, and $\lambda \in \{0.025, 0.25, 0.5, 0.75, 1\}$ for RTE and $\lambda \in \{0.03, 0.25, 0.5, 0.75, 1\}$ for TEMP.

Table 3.8 shows, for each algorithm, the cost spent on training and the accuracy on the test data for RTE. Table 3.9 shows, for each algorithm, the cost spent on training and the accuracy on the test data for TEMP. For both data sets, IEAdjCost and wIEAdjCost show a clear advantage over IEThresh: they save significant cost on labeling while maintaining high accuracy. Further, wIEAdjCost shows a clear advantage over IEAdjCost with the same value for λ. Generally speaking, wIEAdjCost outperformed IEAdjCost in most cases.

3.5 Conclusions & Future Work

We presented a new algorithm IEAdjCost for active learning from multiple labelers with unknown and varied accuracies and known, varied costs. IEAdjCost has two phases. Phase 1 is similar to IEThresh, which estimates the accuracies of the labelers, except that IEAdjCost reduces costs by reducing the number of labelers used to estimate the true labels and by stopping early when a sufficient number of labelers have sufficiently well-estimated accuracies. In Phase 2, IEAdjCost chooses a subset of the labelers for which a good accuracy estimate exists. When the labelers of this subset are used in a majority voting scheme, their combined accuracy is high and combined cost is low. To achieve this goal, we devel-

Table 3.8: Training costs and test accuracies on RTE. The savings are the cost savings of IEAdjCost over IEThresh, and wIEAdjCost over IEAdjCost.

When costs of labelers are 1.

Method	Cost (savings %)	Accuracy	$\lvert O_f \rvert$
Repeated	25600	1.00	
IEThresh	22072	1.00	
IEAdjCost ($\lambda = 0.025$)	2584 (88%)	0.987	3.0
IEAdjCost ($\lambda = 0.25$)	2725 (88%)	0.982	3.0
IEAdjCost ($\lambda = 0.50$)	3143 (86%)	0.986	3.1
IEAdjCost ($\lambda = 0.75$)	3572 (84%)	0.993	3.0
IEAdjCost ($\lambda = 1.00$)	5462 (75%)	0.995	3.0
wIEAdjCost ($\lambda = 0.025$)	1820 (30%)	0.986	3.0
wIEAdjCost ($\lambda = 0.25$)	1935 (29%)	0.981	3.0
wIEAdjCost ($\lambda = 0.50$)	2122 (32%)	0.983	3.1
wIEAdjCost ($\lambda = 0.75$)	2380 (33%)	0.988	3.2
wIEAdjCost ($\lambda = 1.00$)	3032 (44%)	0.989	3.4

When costs of labelers are randomly chosen from $\{1, \ldots, 100\}$.

Method	Cost (savings %)	Accuracy	$\lvert O_f \rvert$
Repeated	1340800	1.00	
IEThresh	1172742	1.00	
IEAdjCost ($\lambda = 0.025$)	95914 (92%)	0.984	3.0
IEAdjCost ($\lambda = 0.25$)	94002 (92%)	0.984	3.8
IEAdjCost ($\lambda = 0.50$)	107624 (91%)	0.989	4.5
IEAdjCost ($\lambda = 0.75$)	117106 (90%)	0.990	4.4
IEAdjCost ($\lambda = 1.00$)	222838 (81%)	0.993	4.5
wIEAdjCost ($\lambda = 0.025$)	58665 (39%)	0.982	3.0
wIEAdjCost ($\lambda = 0.25$)	70727 (25%)	0.984	3.8
wIEAdjCost ($\lambda = 0.50$)	78862 (27%)	0.986	4.2
wIEAdjCost ($\lambda = 0.75$)	75652 (35%)	0.988	4.2
wIEAdjCost ($\lambda = 1.00$)	100316 (55%)	0.989	5.0

Table 3.9: Training costs and test accuracies on TEMP. The savings are the cost savings of IEAdjCost over IEThresh, and wIEAdjCost over IEAdjCost.

When costs of labelers are 1.

Method	Cost (savings %)	Accuracy	$\lvert O_f \rvert$
Repeated	11470	1.00	
IEThresh	9674	1.00	
IEAdjCost ($\lambda = 0.03$)	1596 (84%)	0.989	3.0
IEAdjCost ($\lambda = 0.25$)	1681 (83%)	0.990	3.0
IEAdjCost ($\lambda = 0.50$)	1950 (80%)	0.993	3.1
IEAdjCost ($\lambda = 0.75$)	2308 (76%)	0.997	3.0
IEAdjCost ($\lambda = 1.00$)	3802(61%)	0.997	3.0
wIEAdjCost ($\lambda = 0.03$)	1067 (33%)	0.987	3.0
wIEAdjCost ($\lambda = 0.25$)	1183 (30%)	0.991	3.0
wIEAdjCost ($\lambda = 0.50$)	1306 (33%)	0.996	2.8
wIEAdjCost ($\lambda = 0.75$)	1379 (40%)	0.991	2.4
wIEAdjCost ($\lambda = 1.00$)	1973 (48%)	0.993	3.0

When costs of labelers are randomly chosen from $\{1, \ldots, 100\}$.

Method	Cost (savings %)	Accuracy	$\lvert O_f \rvert$
Repeated	625300	1.00	
IEThresh	496520	1.00	
IEAdjCost ($\lambda = 0.03$)	54148 (89%)	0.990	3.0
IEAdjCost ($\lambda = 0.25$)	56929 (89%)	0.991	3.0
IEAdjCost ($\lambda = 0.50$)	62215 (87%)	0.990	4.5
IEAdjCost ($\lambda = 0.75$)	81472 (84%)	0.991	4.8
IEAdjCost ($\lambda = 1.00$)	170540 (66%)	0.985	4.1
wIEAdjCost ($\lambda = 0.03$)	24111 (55%)	0.989	3.0
wIEAdjCost ($\lambda = 0.25$)	35500 (38%)	0.990	3.7
wIEAdjCost ($\lambda = 0.50$)	43835 (30%)	0.985	3.9
wIEAdjCost ($\lambda = 0.75$)	47116 (42%)	0.982	4.1
wIEAdjCost ($\lambda = 1.00$)	81552 (52%)	0.990	4.3

oped the notions of combined accuracy and adjusted cost to compare labelers with different costs and accuracies. Then our algorithm uses the heuristic LabelersByAdjCost to find the high-accuracy, low-cost subset that is used to label instances from that point forward. We then presented wIEAdjCost, which improves IEAdjCost by simplifying its computation of combined accuracy, and also exploits well-estimated labelers as soon as possible.

We tested our algorithms on six UCI data sets and data from Amazon Mechanical Turk, and showed that our new algorithms performed at least as well, and often better than, algorithm IEThresh from the literature and the naïve approach Repeated. Further, our even newer algorithm wIEAdjCost improves on our previous algorithm IEAdjCost by utilizing the notion of weight to significantly reduce final cost as well as time. This was especially true when there were many highly accurate labelers available, and/or when high combined accuracy was needed.

In future work, we will look at developing more sophisticated methods to learn when to switch from Phase 1 to Phase 2, perhaps by dynamically tuning δ and λ. We will also do a more rigorous Wilcoxon signed rank test instead of t-test because Wilcoxon signed rank test does not assume that the samples are normally distributed. We also plan to investigate if our ideas could be applied to solve the cost-sensitive feature acquisition problem (desJardins et al., 2010). Finally, it would be interesting to consider the possibility that labelers' accuracies could be affected by exposure to labeled and unlabeled instances, as what is seen to happen to human labelers (Zhu et al., 2007).

Chapter 4

New Algorithms for Budgeted Learning

4.1 Introduction

Some early results of our work appeared in our earlier work (Deng et al. (2007)), in which we presented new algorithms for choosing which attributes of which examples to purchase in the budgeted learning model. Several of our algorithms were based on results in the "multi-armed bandit" model. In this model, we have N slot machines that we may play for some fixed number of rounds. At each round, one must decide which single slot machine to play in order to maximize total reward over all rounds. Our first two algorithms were based on the algorithm Exp3 of Auer et al. (2002b) originally designed for the multi-armed bandit problem, and our third is based on the "follow the perturbed leader" approach of Kalai & Vempala (2005). In this chapter, we present the preliminary work as well as three new algorithms which are variations of Biased Robin from the literature (Kapoor and Greiner, 2005b) by incorporating second-order statistics into the decision-making process. We refer to our algorithms that use second-order statistics as Biased Robin 2 (BR2) series algorithms.

Our experimental results show that most of our proposed algorithm perform well compared to the existing algorithms.

Previous budgeted learning algorithms that use naïve Bayes as the base classifier focus only on selecting attributes to purchase and then choose uniformly at random the instance whose attribute is to be purchased. In other words, such algorithms consider one instance to be as good as any other (given the class label) for querying the chosen attribute. In this chapter, we experiment with strategies to select instances as well as attributes, choosing instances that are most wrongly predicted or that are least certain in the current model. (We note that in Melville et al. (2004) and Melville et al. (2008b), they proposed to use US (uniform sampling) and ES (Error Sampling) to consider partial instances instead of all of the instances. The sampling is done before choosing an (instance, feature) pair. This is different from our row selection. Our row selection policy is applied after a feature is chosen first.) Our results on these new instance selection methods are better than random row selection for some algorithms.

4.2 Background and Related Work

Budgeted learning is related to conventional machine learning techniques in that the learner is given a set D of labeled training data and it infers a classifier (hypothesis, or model) to label new examples. The key difference is that, in budgeted learning, there is an overall budget, the learner has to use this given budget to learn as good classifier/hypothesis as possible. There is a body of work in the data mining community by the name of "active feature acquisition" (Zheng and Padmanabhan, 2002; Melville et al., 2004; Melville et al.; Lomasky et al., 2007; Saar-Tsechansky et al., 2009), in which the features are purchased at a cost and the only difference is that their goal is to learn a hypothesis using at little cost as possible (i.e., no strict budget). For example, Saar-Tsechansky et al. (2009) proposed to use

Log Gain—the expected likelihood of the training set—as a smoother measure of goodness of an attribute purchase. This idea was somewhat similar to some of the measures (conditional entropy and GINI index) that we have used for several budgeted learning algorithms. They also consider for purchase only those instances that are misclassified by the current model instead of all of the instances to reduce the search space.

The difference between active feature acquisition and budgeted learning is that, budgeted learning usually has a hard budget set up front, while active feature acquisition does not have a hard budget. A minor distinction is that in some applications of active feature acquisition (Zheng and Padmanabhan, 2002; Lomasky et al., 2007), individual instance/feature values cannot be bought one at a time. Instead, all the missing attributes of an instance must be synthesized or obtained as a whole.

Budgeted learning falls under a general framework of problems that represent a trade-off between *exploration* and *exploitation* in an online decision process. In this framework, a learner has several actions it may choose from, each with some corresponding payoff. Initially, the learner starts with little or no knowledge and must spend some time gathering information about which attributes are most relevant, reflecting the need for exploration. At some point, the learner begins exploitation by purchasing more values for individual values of the attributes that are known to be informative, in an attempt to maximize its expected reward. In budgeted learning, the reward is the performance of the final classifier when the budget is exhausted. Clearly, the budgeted learner must purchase a variety of attributes in order to form a more complete model of the underlying problem domain and also figure out which attributes are more informative, thus showing the importance of exploration in budgeted learning. Exploration and exploitation can and often do operate at the same time. In other words, there may be no clean-cut boundaries between these two facets. For example, to explore more efficiently (which is crucial when the budget is limited), we often need to utilize/exploit what is already known to decide which attribute to purchase next. On the

other hand, the results of exploration are valuable for deciding which attributes should be purchased more often in order to build a good final classifier.

One of the original applications of budgeted learning was in the medical domain (Madani et al., 2004). In this application, the examples are patients, and the (known) class label of patient x is $y \in \{-1, +1\}$, indicating whether or not x responded to a particular treatment. The (initially unknown) attributes of x are the results of tests performed on tissue samples gathered from patient x, e.g., a blood test for the presence of a particular antibody. In this case, any attribute of any instance can be determined. However, each costs time and money, and there is a finite budget limiting the attributes that one can buy. Further, each attribute can cost a different amount of money, e.g. a blood test may cost less than a liver biopsy.

A second application of budgeted learning was in customer modeling. A company has significant data (attributes) on its own customers but may have the option to pay for other information on the same customers from other companies. Though they do not explicitly refer to it as "budgeted learning," Zheng and Padmanabhan (2002) discuss this problem as applied to learning models of customers to web services, where a customer's browsing history at a local site (e.g. Travelocity) is known, but the same customer's history at other sites (e.g. Expedia) is not, though by assumption it could be purchased from these competing sites. Zheng and Padmanabhan evaluated two heuristic approaches to this problem, both of which are based on the idea of how much additional information the unspecified attributes can provide. Their first algorithm (AVID) imputes, at each iteration, the values of the unspecified features based on the specified ones. It does this multiple times and then considers the feature with the highest variance to be the least certain and thus the best one to purchase. Their second algorithm (GODA) also imputes values of the unspecified features, but then it selects for purchase the unspecified feature that maximizes expected "goodness" based on a specified function (e.g. training error). In their work, they assume that all attributes' values are purchased at the same time rather than as individual (instance, feature) pairs.

In other work, Veeramachaneni et al. (2006) studied the problem of budgeted learning for the purpose of detecting irrelevant features rather than building a classifier. Specifically, let $\theta \in \Theta$ parameterize the probability distribution over $\mathcal{Z} \times \mathcal{X} \times \mathcal{J}$, where $\mathcal{Z}$ is the space of attributes that are always known to the learner, $\mathcal{X}$ is the space of attributes that can be queried by the learner, and $\mathcal{J}$ is the space of labels. Then their goal was to accurately learn some function $g(\theta)$ (e.g. the true classification error rate of a model) by querying as few unknown attributes as possible. Their algorithm purchased attributes that were expected to induce maximum change in the estimate of g.

Related to budgeted learning is the learning of "bounded active classifiers", in which one has fully specified training examples with their labels, but the final hypothesis h must pay for attributes of new examples when predicting a label, spending at most B_h. This, of course, can be combined with budgeted learning to the budgeted learning of bounded active classifiers. Such a learning algorithm has been termed a "budgeted bounded-active-classifier learner" (BBACL) (Kapoor and Greiner, 2005b,a; Greiner et al., 2002). For simplicity, in our work we focus on budgeted learning of unbounded classifiers, leaving the extension of our results to the BBACL problem as future work.

Most theoretical studies of budgeted learning are based on related problems within the exploration versus exploitation framework, such as the *coins problem* and the *multi-armed bandit problem* (see Section 4.2.3). In the coins problem, one is given N biased coins $\{c_1, \ldots, c_N\}$, where c_i's probability of heads is distributed according to some known prior R_i (though we say "coins," the problem can be generalized to non-binary outcomes). In each round, a learner selects a coin to flip at unit cost. After exhausting its budget, the learner must choose a single coin to flip *ad infinitum*, and its payoff is the number of heads that the coin yields. The goal is to choose the coin that has the highest probability of heads, and performance is measured by the *regret* incurred by the learner's choice, i.e. the expected amount that the learner could have done better by choosing the optimal coin. Madani et

al. (2004) showed that this problem can be modeled as one of sequential decision making under uncertainty, and as such can be solved via dynamic programming when the number of coins is taken to be constant. However, the complexity of such a dynamic programming solution is exponential, and in fact this problem is NP-hard under specific conditions. They argued that straightforward heuristics such as "Round Robin" (repeatedly cycle through each coin until the budget is exhausted) do not have any constant approximation guarantees. That is, for any constant ℓ there is a problem with minimum regret r^* such that the regret of Round Robin is $> \ell r^*$. Kapoor & Greiner (2005c) empirically evaluated common reinforcement learning techniques on this problem. Guha & Munagala (2007) were able to design a 4-approximation algorithm for the general coins problem using a linear programming rounding technique. Goel et al. (2009) presented an algorithm that also guranteed a constant factor approximation. Their algorithm was based on "ratio index" (analogous to the Gittins index) such that a single number can be computed for each arm and the arm with the highest index is chosen for experimentation.

Budgeted learning has recently been studied with alternative goals. For example, budgeted learning has been studied in the following context (Antos et al., 2008): sample as few times as possible to minimize the maximal variance of all the arms in a multi-armed bandit problem. In Li (2009), the goal is to minimize the expected risk of the parameters in a generative Bayesian network, with the risk chosen to be the expected KL divergence of the parameters from their expectations. Goel et al. (2006) studied optimization problems with inputs that are random variables, where the only available data are samples of the distribution.

In the following sections, we give detailed descriptions of algorithms on which we base our contributions. The following sections, we explain existing algorithms in the literature. Section 4.2.1 explains the Biased Robin (Lizotte et al., 2003). Section 4.2.2 introduces RSFL (Kapoor and Greiner, 2005b). Section 4.2.3 explains bandit-based algorithms (Auer

et al., 2002b; Kalai and Vempala, 2005).

4.2.1 Biased Robin

One of the simplest early budgeted learning algorithms was Biased Robin (BR). BR is similar to a Round Robin approach (where first attribute 1 is purchased once, then attribute 2, and so on to attribute N, then repeat), except that attribute i is repeatedly purchased until such purchases are no longer "successful," and then the algorithm moves on to attribute $i + 1$. Here, success is measured by a feedback function, examples of which are described later. Despite its simplicity, BR was a solid performer in training naïve Bayes classifiers (Lizotte et al., 2003; Kapoor and Greiner, 2005b).

4.2.2 Single-Feature Lookahead

Single Feature Lookahead (SFL) is a method introduced by Lizotte et al. (2003). Using the posterior class distribution of its naïve Bayes model, one can compute the expected loss of any sequence of actions. Given sufficient computational resources and the ability to compute expected loss, one can achieve Bayes optimality by considering all possible future actions contingent on all possible outcomes along the way (Wang et al., 2005). Because this is too computationally intensive, SFL and similar approaches restrict the space of future models they consider, by restricting the class of policies considered. In SFL, each (label, attribute) pair is associated with an expected loss incurred by spending the *entire* remaining budget on that pair. This expected loss is computed using the current posterior naïve Bayes model, which, given an allocation, gives the distribution over future models. Expected loss is computed over this distribution. The pair with the lowest loss is then purchased once. SFL introduces a lookahead into the state space S of all possible purchases without explicitly expanding the entire state space. At any point in the algorithm's execution, one has an

allocation α: a description of how many feature values have been purchased from instances with a certain class label. Specifically, α_{ijk} is a count of the number of times attribute i has been purchased from an instance with class label j with resulting attribute value k. Thus, SFL requires the possible attribute values to be discrete. Given a current allocation, SFL calculates the expected loss of performing all possible single-purchase actions (purchasing an attribute/label pair which results in a specific attribute value) and chooses the action that minimizes the expected loss of the resulting allocation. In a randomized version of SFL called RSFL (Kapoor and Greiner, 2005b), the next (label, attribute) pair to purchase is sampled from the Gibbs distribution induced by the SFL losses.

In SFL and RSFL, the expectation is over all belief states that can be reached using the given allocation:

$$\sum_{\alpha'} P(\alpha') \operatorname{Loss}(\alpha')$$

where the loss of a possible new state (represented by the new allocation α' after the purchase) is weighted by its probability of occurrence.

Several heuristics have been considered for the loss function, including the GINI index (Lizotte et al., 2003) and expected classification error (Kapoor and Greiner, 2005b). The GINI index is defined as

$$\sum_{j \in J} \sum_{\vec{x} \in X} P(\vec{x})\, P(j \mid \vec{x})\, (1 - P(j \mid \vec{x})) \tag{4.1}$$

where J is the set of all possible labels and $\vec{x}$ is a feature vector drawn from an instance space X. The expected classification error with respect to an attribute a_i is defined as

$$\sum_{k} P(a_i = k) \min_{j \in J} (1 - P(j \mid a_i = k)) \tag{4.2}$$

where the sum is taken over all possible attribute values k for attribute a_i.

In our experiments, we found that Randomized SFL and Biased Robin performed their

best when using conditional entropy (as used by Kapoor and Greiner (2005b)):

$$\mathsf{CE}(i,j) = -\sum_k P(a_i = k) \sum_j P(j \mid a_i = k) \log_2 \left(P(j \mid a_i = k)\right) \tag{4.3}$$

Conditional entropy measures the uncertainty of the class label given the value of an attribute.

In our experiments, we compared our algorithms against a randomized version of SFL called RSFL (Kapoor and Greiner, 2005b) because the straightforward version can degenerate into Biased Robin, exhausting its budget on the current best attribute (Kapoor and Greiner, 2005b). Such behavior was also observed in the context of the coins problem (Madani et al., 2004) and we experienced similar results in our experiments with pure SFL. In RSFL, the conditional entropy is used to define a Gibbs distribution for which we choose the ith attribute from an instance with class label j with probability

$$\frac{\exp\left(-\mathsf{CE}(i,j)\right)}{\sum_{i,j} \exp\left(-\mathsf{CE}(i,j)\right)}.$$

4.2.3 The Multi-Armed Bandit Problem

There are close connections between budgeted learning and the so-called "multi-armed bandit problem" first studied by Robbins (1952). The problem can be stated as follows (Gittins, 1979): there are N arms, each having an unknown success probability of emitting a unit reward. The success probabilities of the arms are assumed to be independent of each other. The objective is to pull arms sequentially so as to maximize the total reward. Many policies have been proposed for this problem under the independent-arm assumption (Lai and Robbins, 1985; Auer et al., 2002b). The key difference between budgeted learning and the multi-armed bandit problem is that in the latter, one tries to maximize the cumulative rewards over all pulls, whereas with budgeted learning, one simply wants to maximize the

accuracy of the resulting classifier or a "simple" reward in some sense.

In the following sections, we describe algorithm for multi-armed bandit problem that we use in our work. Section 4.2.3.1 presents the two algorithms adapted from bandit-based algorithms. Section 4.2.3.2 explains another adapted algorithms of bandit-based algorithm FEL. Section 4.2.3.3 shows other related work. Section 5.6.6 discusses about the related theoretical work that motivated our work.

4.2.3.1 Exp3 Algorithm

In the context of approaching budgeted learning as a multi-armed bandit problem, we apply results from Auer et al. (2002b). Their most basic algorithm (Exp3) maintains a weight w_i (initialized to 1) for each of the N arms. At each trial, it plays machine i with probability

$$P(i) = \frac{\gamma}{N} + (1-\gamma)\frac{w_i}{\sum_{n=1}^{N} w_n} \ ,$$

where γ is a parameter governing the mixture between the weight-based distribution (controlling exploitation) and the uniform distribution (allowing for exploration). After playing the chosen machine (call it i), the reward r is returned, which is used to update the weight w_i by multiplying it by $\exp(\gamma\, r/(P(i)N))$ (all other weights are unchanged).

Auer et al. proved that under appropriate conditions,[1] the expected total reward of Exp3 will not differ from the best possible by more than $2.63\sqrt{gN \ln N}$, where g is an upper bound on the total reward of the best sequence of choices.

4.2.3.2 FPL Algorithm

Another related work is the "follow the perturbed leader" (FPL) algorithm (Kalai and Vempala (2005)). Although originally designed as an online expert-based algorithm, is applicable

[1]These conditions are based only on the choice of parameters, not any statistical properties of the slot machines.

as an algorithm for the multi-armed bandit problem. The idea is to select the most informative arm by selecting the best arm thus far, hence "follow the leader." At each time step, a certain cost (or equivalently reward, as they are inversely related) is counted toward the arm selected. At time t, the cumulative cost of each arm can be calculated, and the arm that has incurred the least cost is chosen for the next pull. Without randomization, an adversary can easily trick such deterministic algorithms into wrong decisions. To address this problem, a random perturbation is added to the cost of each arm before making the decision, thus the name "follow the perturbed leader." Similar to the results of Auer et al. (2002b), it can be shown that, under appropriate conditions, the regret of the perturbed leader algorithm is small relative to the best sequence of choices. Let min-cost$_T$ be the total cost of the best single arm up to time T in hindsight. Then the expected cost of the perturbed leader can be bounded as

$$E[\text{cost}_{\text{FPL}}] \leq (1+\varepsilon)\,\text{min-cost}_T + \frac{\mathcal{O}(\log N)}{\epsilon}$$

where ε is a user specified parameter. The details of its adaptation to the budgeted learning setting are described in Section 4.3.2.

4.2.3.3 Other Results

Recently, Bubeck et al. (2008) pointed out an interesting link between simple (one-shot) and cumulative (overall) regret, which is the difference in reward of the algorithm in question and that of an optimal, omniscient algorithm. One of the surprising results is that an upper bound on the cumulative regret implies a lower bound on simple regret for *all* algorithms. In other words, the exploration-exploitation trade-offs are qualitatively different for these two types of regrets. In fact, according to Bubeck et al. (2008), one of the very successful algorithms (Auer et al., 2002a) for cumulative regret would perform worse than uniform random algorithm when the budget goes to infinity. However, in their simulation study,

this was not observed since in order for this to occur, the budget would have to be very large, to the point that the computed regrets would fall below that of the precision of the computer used to run the simulations. Their results do not seem to be directly transferable to budgeted machine learning due to their assumptions. Another important contribution is that they pointed out that exploration (allocation) strategies can be different from the decision (recommendation) strategies in the pure exploratory multi-armed bandit problems.

4.2.3.4 Discussion

The theoretical guarantees in the work of Auer et al. (2002b) give us good motivation for using similar approaches to the budgeted learning problem. Auer et al. make no assumptions about the underlying distribution of slot machines and their results still hold even when the rewards are dynamic and may depend on previous sequences of rewards and the player's random draws. Thus, we can plug in our choice of reward function for the slot machines (say, the negative conditional entropy of the class label with respect to an attribute) and their bounds automatically translate into guarantees in the budgeted learning context. However, these bounds only apply to the *cumulative* regret with respect to the best sequence of arm pulls. For instance, nothing is said by these bounds about the class label's uncertainty with respect to an attribute upon the last round of purchases. In other words, it remains an open question whether under some conditions, one can bound the resulting training error with respect to the best set of purchases or the best possible error rate.

4.3 Our Algorithms

For simplicity, all of our algorithms focus on a unit-cost model of budgeted learning, i.e. each attribute costs one monetary unit. Future work is to generalize our algorithms to handle nonuniform cost models. Sections 4.3.1 and 4.3.2 present our two algorithms based on Exp3

and one algorithm based on "follow the perturbed leader" for multi-armed bandit problems. Section 4.3.3 presents our algorithms based on Biased Robin and second order statistics. Section 4.3.4 presents our row selection policies.

4.3.1 Exp3-Based Algorithms (Exp3C and Exp3CR)

Our first two algorithms are based on the multi-armed bandit algorithm Exp3 of Auer et al. (2002b) described in Section 4.2.3.1. Our first algorithm (Exp3C, for "Exp3-Column") treats each of the N attributes (columns) as an arm. Each column has a weight that is initialized to 1. Each purchasing round, Exp3C chooses an attribute (column) of some instance to buy based on the weights. Specifically, Exp3C chooses column i with probability

$$P(i) = \frac{\gamma}{N} + (1-\gamma)\frac{w_i}{\sum_{n=1}^{N} w_n}$$

where γ is a parameter. After the purchase, we build a new naïve Bayes model on the training set and Exp3C gets as a reward the classification accuracy of the naïve Bayes model evaluated on the partially specified training set.[2] The Exp3C algorithm is presented as Algorithm 1.

After choosing a column to purchase, a row (instance) must also be selected. After choosing a column i, Exp3C selects a row uniformly at random from all rows that do not yet have column i filled in.

[2]We tried several reward functions as described before: GINI index, expected classification error, and conditional entropy. Classification error on the training set tended to work best for our algorithms.

Algorithm 4.1 Algorithm Exp3C.

Input : Parameter γ, budget B

Initialize weight vector $\vec{w} = (w_1, w_2, \ldots, w_N) = \vec{1}$;

for $t = 1, \ldots, B$ **do**

Choose attribute i with probability $P(i) = \frac{\gamma}{N} + (1-\gamma)w_i / \left(\sum_{n=1}^{N} w_n\right)$;

Select a row with attribute i missing according to some row selection strategy;

Observe the attribute value and update the classifier model;

Compute the reward r as a measure of improvement in the model's accuracy, which is equal to accuracy on the new training set minus accuracy on the previous set;

Update $w_i \leftarrow w_i \exp(\gamma\, r/(P(i)\, N))$, keeping other weights unchanged;

end for

In our second algorithm (Exp3CR, for "Exp3-Column-Row"), we define a distribution over the rows as well as the columns, i.e. we now have two weight vectors instead of one. After choosing a column according to its distribution (which is done the same way as in Exp3C), our algorithm then chooses a row according to the row distribution. Once reinforcement is received, both weight vectors are updated independently of each other. Thus we replace the naïve Bayes assumption with a product distribution over the (column, row) pairs.

As previously mentioned, by directly applying the regret bounds of Auer et al. (2002b), we easily bound the regret of Exp3C. Specifically, we see that the reward of our algorithms will be within a factor of $2.63\sqrt{gB \ln B}$ of that of the best sequence of attribute choices, where B is an upper bound on the total reward of this best sequence. It remains an open problem as to what kinds of bounds follow for Exp3CR.

4.3.2 Follow the Expected Leader (FEL)

Our third algorithm is a variation of the "follow the perturbed leader" (FPL) type algorithms due to Kalai & Vempala (2005). As with the previous two algorithms, we treat each attribute as an arm.

Let $x_i(t)$ be the cost of the ith attribute at time step t. At each time step t, FPL computes the sum of all costs of each arm, $c_i = \sum_{q=1}^{t-1} x_i(q)$ and adds a perturbation factor (or noise) ε_i generated uniformly at random from $[0, 1/\epsilon]$. FPL then choses to play the arm

$$\underset{1 \leq i \leq n}{\operatorname{argmin}} \{c_i + \varepsilon_i\} \ .$$

The framework for FPL assumes that we have access to the costs $x_i(t)$ for all arms at every time step (had they been chosen). However, this assumption is not reasonable in the context of budgeted learning. For this reason, our implementation is a slight variation of the standard FPL called FEL ("follow the *expected* leader") from Kalai & Vempala (2005). First, we assume that $x_i(t)$ is zero if the arm was not played (the attribute was not chosen as a purchase). Next, let $\#x_i(t)$ be the number of times attribute i was chosen up to time step t. Now, instead of cumulative cost, we use the average of the perturbed cost (over the trials that an attribute is actually purchased) as the selection criterion. The FEL algorithm is presented as Algorithm 2. Just as with Exp3C and Exp3CR, we measure the cost as the training error on the partially specified training set. Again, we considered other measures: GINI index, expected classification error, and conditional entropy. Classification error on the training set tend to work best for our algorithms.

Algorithm 4.2 Algorithm Follow the Expected Leader (FEL).

Input : Parameter ϵ, budget B

for $t = 1, \ldots, B$ **do**

Choose $\varepsilon_1, \ldots, \varepsilon_N$ independently and uniformly from $[0, 1/\epsilon]$;

Define $S = \{s_1, \ldots, s_N\}$ where $s_i = \Big(\big(\sum_{q=1}^{t-1} x_i(q) \big) + \varepsilon_i(t) \Big) \Big/ \Big(\#x_i(t-1) + 1 \Big)$;

Buy attribute $\text{argmin}_i \{s_1, \ldots, s_N\}$;

Update $\#x_i(t)$ accordingly;

end for

4.3.3 Variance-Based Biased Robin Algorithms

Recall that Biased Robin is similar to a Round Robin approach except that attribute i is repeatedly purchased until such purchases are no longer "successful," and then the algorithm moves on to attribute $i + 1$. In practice, however, making a decision solely based on the outcome of the last action can be problematic, e.g. due to the fluctuation (instability) of the learning process (Kapoor and Greiner, 2005b).

We tested three alternative methods based on second-order statistics to judge whether a further purchase of the same attribute is successful or not. Let the current change of the hypotheses be

$$\delta(t) = \frac{\sum_{m=1}^{M} \sum_{j=1}^{J} |P_t(j \mid x_m) - P_{t-1}(j \mid x_m)|}{M} \ ,$$

where $P_t(j \mid x_m)$ is the probability estimated by the trained classifier (after the tth purchase) that the instance m belongs to the jth class, $P_{t-1}(j \mid x_m)$ is the probability estimate made by the classifier built after the $(t-1)$th purchase, M is the number of instances, and J is the number of classes.

All the following heuristics are based on the intuition that no further exploration of an attribute should be continued if the induced hypothesis does not change much at all. For the

first method, a purchase is successful when $\delta(t) > \Delta_a$. For the second method, a purchase is successful when $\delta(t)/\delta(t-1) > \Delta_r$. For the third method, a purchase is successful when $\delta(t) > \mathrm{m}edian\{\delta(t-\Delta_w), \delta(t-\Delta_w+1), \ldots, \delta(t-1)\}$. In these algorithms, Δ_a, Δ_r, and Δ_w are parameters. The BR using the above three methods judging "successful" are called AbsoluteBR2 (ABR2), RelativeBR2 (RBR2), and WindowBR2 (WBR2), respectively.

4.3.4 Instance (Row) Selection Heuristics

Many budgeted learning algorithms (except Exp3CR) only select columns for purchase, implicitly assuming that given a column, any instance (or any instance with a given class label) is equally useful. Thus rows are selected uniformly at random from either the entire training set or from among instances of a particular class. However, it may not always be the case that two instances are equally informative given an attribute. Thus, we refine these algorithms by defining criteria for choosing specific instances from which to purchase an attribute. When using our row selection strategies, after the budgeted learner chooses an attribute and a class label,[3] the row (instance) chosen will be the one optimizing our selection criteria among those with the same class label and with the selected column yet unpurchased.

4.3.4.1 Entropy as Row Selection Criterion (EN, en)

Intuitively, given a selected column, one wants to find a row such that the (row, column) purchase gives the most information possible. Put another way, we want to choose an instance whose classification is least certain under the current model. That is, it is best to choose an example nearest to the current model's decision boundary. This technique has been very successful in active learning (Lewis and Gale, 1994b; Campbell et al., 2000; Schohn

[3]If a class label is not chosen at this stage, then the row selection strategy will randomly choose a class label.

and Cohn, 2000; Tong and Koller, 2001c). For naïve Bayes this means choosing the instance whose posterior class probability distribution is closest to uniform over the classes. I.e., we choose the instance x_m that maximizes the entropy of the posterior class distribution:

$$-\sum_{j=1}^{J} P_t(j \mid x_m) \log_2 P_t(j \mid x_m)$$

4.3.4.2 Error Correction as Row Selection Criterion (EC, ec)

The other row selection heuristic we considered is a greedy "error-correcting" approach. For each training instance m still with missing attributes, we calculate the predicted probability of its true class $P_t(\ell_m \mid x_m)$, where ℓ_m stands for the true class of instance m. We then pick $\text{argmin}_m \{P_t(\ell_m \mid x_m)\}$, the example with the smallest estimated probability in its true class. Intuitively, the selected example is more likely to be classified wrong by the classifier, so knowing more about this example should improve the performance, especially in the early stages of the training process. Melville et al. (2004) and Melville et al. (2008a) proposed to use US (uniform sampling) and ES (Error Sampling) to only consider partial instances to consider instead of all of the instances. The sampling is done before choosing an (instance, feature) pair. This is different from our row selection which is applied after a feature is chosen first.

4.4 Experimental Results

In this section we present our experimental results on several UCI data sets (Asuncion and Newman, 2009) (see Table 4.1). All of these data sets has a large number of attributes, which is good for testing for budgeted learning algorithms whose essence is to identify those attributes that are more helpful for building the hypothesis or classifier. In order to use naïve Bayes classifier, we chose only data sets that had nominal attributes or could easily be

Table 4.1: Data set information.

Data set	Num. of Instances	Num. of Attributes	Num. of Classes
`breast-cancer`	286	9	2
`colic`	368	22	2
`kr-vs-kp`	3196	35	2
`mushroom`	8124	22	2
`vote`	435	16	2
`zoo`	101	17	7

made nominal. For the few data sets with missing attributes, the mode was used to fill in that attribute value. All algorithms were written in Java within the Weka machine learning framework (Witten and Frank, 2005) and used its naïve Bayes (NB) as the base learner. We chose NB since it was used by related work (Lizotte et al., 2003), is quick to learn, and handles missing attribute values.

We partitioned each data set in 10 different ways. For each partitioning, we used 10-fold cross validation, so the results presented in this section are averages of 100 folds. We ran 10-fold CV on 10 different partitionings because we found that the performance of each algorithm is sensitive to the partitioning used, and repeating the process reduced the variance. In addition to our algorithms (Exp3C, Exp3CR, FEL and the BR2 variations) and those in the literature described earlier (RSFL and BR), as a control, we also considered a random shopper that uniformly at random selects an unpurchased (instance, attribute) pair. We tried various reward functions with each algorithm, and chose the reward function that worked best for each algorithm. For our algorithms, we used the classification accuracy on the partially specified training set as a reward function. For RSFL and BR, we used conditional entropy. When applicable, we ran each algorithm with uniform random selection of rows as well as with the entropy-based and error correction-based approaches of Section 4.3.4. For simplicity, we assumed uniform costs over all attributes (thus one purchase is simply one attribute of some instance).

Table 4.2: Algorithm abbreviations, full names, and short descriptions.

Algorithm Identifier	Full Name	Source
BR, br	Biased Robin	Section 4.2.1
ABR2, abr2	Absolute Biased Robin 2	Section 4.3.3
RBR2, rbr2	Relative Biased Robin 2	Section 4.3.3
WBR2, wbr2	Window Biased Robin 2	Section 4.3.3
Exp3C, exp3c	Exp3 Column	Algorithm 4.3.1
Exp3CR, exp3cr	Exp3 Column Row	Section 4.3.1
RSFL	Random Single Feature Look-ahead	Section 4.2.2
FEL	Follow the Expected Leader	Algorithm 4.3.2
Rand	Random	Randomly choose a feature and an instance
algoname.ec	aglorithm algname with Error-Correction row selector	Section 4.3.4.2
algoname.en	algorithm algname with Entropy row selector	Section 4.3.4.1
algoname.ur	algorithm algname with Uniform Random row selector	Section 4.3.4

For the algorithms with adjustable parameters, we report the best results among the tried results with different parameters. For Exp3C and Exp3CR, we tried $\gamma \in \{0.01, 0.05, 0.10, 0.15, 0.20, 0.25\}$, and chose $\gamma = 0.15$ for Exp3C and $\gamma = 0.20$ for Exp3CR. For FEL, we tried $\epsilon \in \{0.01, 0.05, 0.1, 0.2, 0.5\}$ and chose $\epsilon = 0.1$. For AbsoluteBR2, we chose $\Delta_a = 0.01$ from $\{0.00001, 0.0001, 0.001, 0.01, 0.1, 1\}$, for RelativeBR2, we chose $\Delta_r = 1$ from $\{0.01, 0.1, 1, 10, 100\}$, and for WindowBR2 we chose $\Delta_w = 2$ from $\{2, 3, 4, 5, 6, 7, 8, 9\}$.

To evaluate the overall behavior of each of the algorithms, we constructed learning curves that reflect the performance of a heuristic by its mean error over the 10×10 folds as more attributes are purchased. On each fold, each algorithm was run up to a budget of $B = 100$. Each purchase was unit cost.

In our experiment, more than 20 algorithms were evaluated for each data set. In Table 4.2 we list algorithm abbreviations, full names, and sources. For the sake of brevity, only learning curves using the EC-based row selection are presented (Figures 4.1 and 4.2), and we only sampled every fifth data point. To keep the plots uncluttered, in Figure 4.1 we plot results for Random, Biased Robin (BR), Absolute BR2 (ABR2), Window BR2 (WBR2) and Relative BR2 (RBR2). Since ABR2 is a strong performer in these curves, we include it in Figure 4.2 as

a reference against Random, Exp3 Column-Row (Exp3CR), Exp3 Column (Exp3C), Follow the Expected Leader (FEL), and Randomized Single-Feature Lookahead (RSFL). In the following sections, we present a detailed analysis based on summary statistics derived from all learning curves.

4.4.1 Summary Statistics

Ultimately, the goal in budgeted learning is to reduce the number of attributes one must purchase in order to effectively learn. To summarize and compare learning curves, we use summary statistics.

For the first such statistic, we define *target mean* be the mean of the error rates for the final 20% of the total budget achieved by the random shopper. Then we define the *target budget* of an algorithm over the 10×10 folds on a given data set as the minimum budget needed by an algorithm to be competitive with the random shopper. For a given algorithm A and trial t (i.e., the naïve Bayes model trained on the training set after t purchases by A), we compute the mean error rate of the last 5% of purchases for A. Then the target budget is the smallest t for which the target mean is achieved by A. We use a window size of 5% of the total budget to reduce the influence of outliers as the learning curves can have high variance early on. If an algorithm fails to achieve the target mean, its target budget is simply the entire budget B.

We also report an algorithm's *data utilization ratio*, which is the algorithm's target budget divided by the target budget of the random shopper. Thus, a lower data utilization ratio reflects that the algorithm was able to make more useful purchases overall while excluding large changes in performance as the budget is exhausted. This is especially informative because with larger budgets, each algorithm will naturally converge to the baseline, making distinctions between them meaningless as more purchases are made. These metrics are

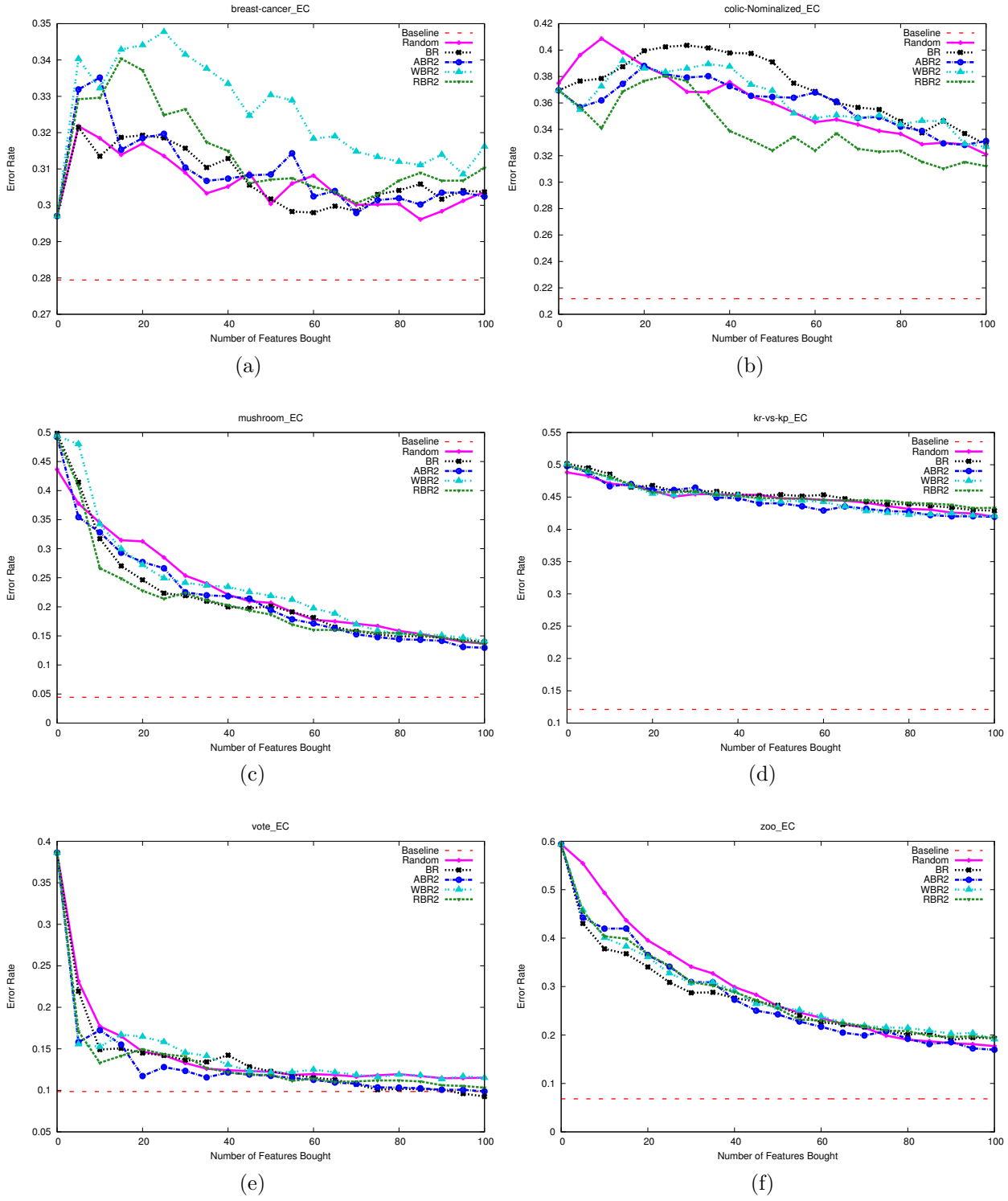


Figure 4.1: Learning curves for each data set under the Error Correction (EC) row selector. (a) Breast-cancer; (b) colic-nominalized; (c) mushroom; (d) kr-vs-kp; (e) vote; (f) zoo.

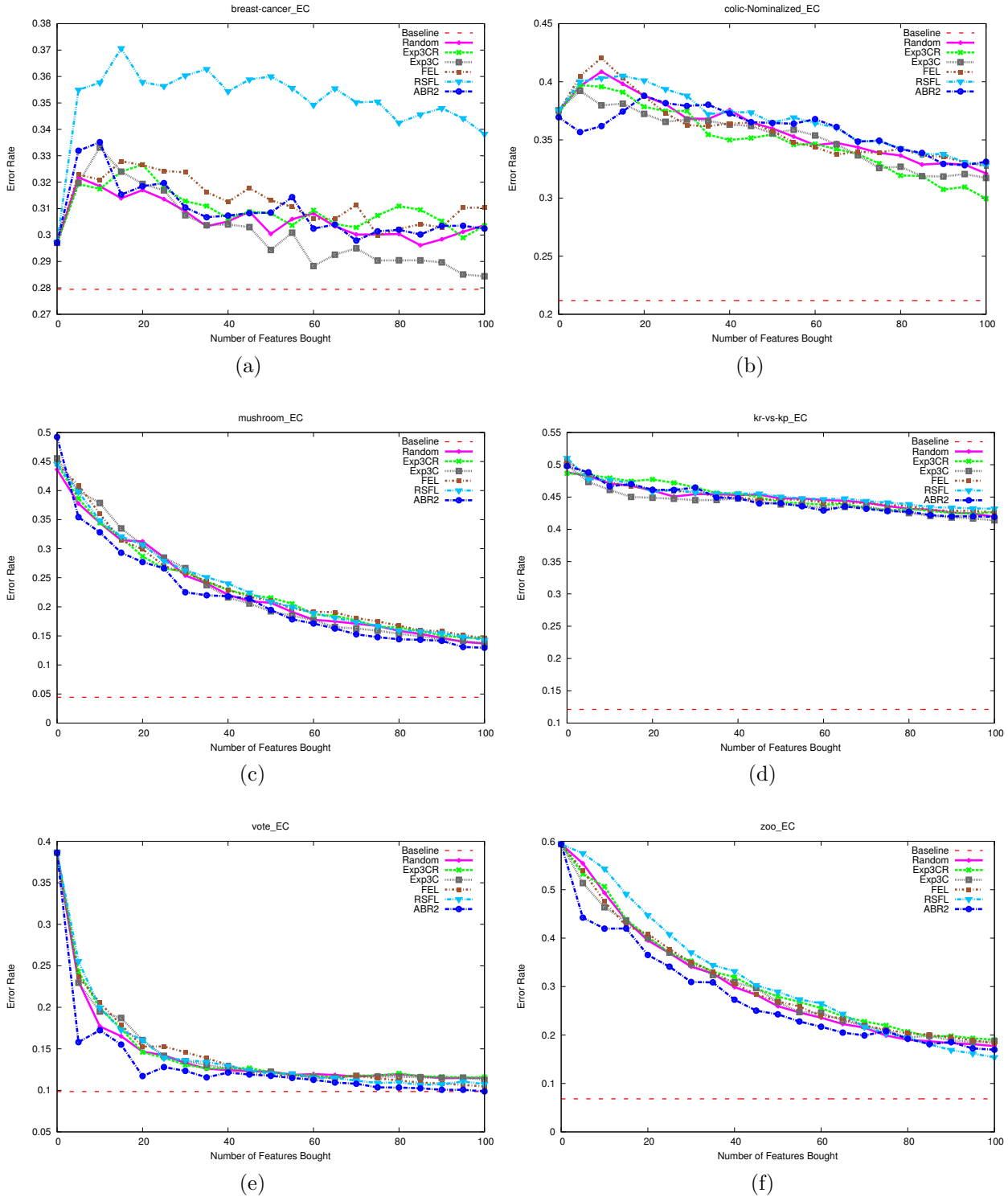


Figure 4.2: Learning curves for each data set under the Error Correction (EC) row selector. (a) Breast-cancer; (b) colic-nominalized; (c) mushroom; (d) kr-vs-kp; (e) vote; (f) zoo.

Table 4.3: Target budget and data utilization rates for algorithms with the EC row selector. Total budget was $B = 100$.

Dataset	exp3cr	exp3c.ec	fel.ec	rsfl.ec	random.ec	br.ec	abr2.ec	wbr2.ec	rbr2.ec
`breast-cancer`	42.0	35.0	100.0	100.0	50.0	46.0	35.0	100.0	60.0
	(0.61)	(0.51)	(1.45)	(1.45)	(0.72)	(0.67)	(0.51)	(1.45)	(0.87)
`colic-Nominalized`	78.0	83.0	78.0	100.0	89.0	100.0	93.0	98.0	51.0
	(0.89)	(0.94)	(0.89)	(1.14)	(1.01)	(1.14)	(1.06)	(1.11)	(0.58)
`kr-vs-kp`	90.0	80.0	90.0	100.0	91.0	100.0	82.0	74.0	100.0
	(0.96)	(0.85)	(0.96)	(1.06)	(0.97)	(1.06)	(0.87)	(0.79)	(1.06)
`mushroom`	86.0	77.0	73.0	83.0	82.0	70.0	71.0	77.0	65.0
	(0.91)	(0.82)	(0.78)	(0.88)	(0.87)	(0.74)	(0.76)	(0.82)	(0.69)
`vote`	100.0	100.0	100.0	71.0	100.0	65.0	63.0	100.0	57.0
	(1.14)	(1.14)	(1.14)	(0.81)	(1.14)	(0.74)	(0.72)	(1.14)	(0.65)
`zoo`	82.0	77.0	86.0	76.0	75.0	77.0	68.0	85.0	77.0
	(0.88)	(0.83)	(0.92)	(0.82)	(0.81)	(0.83)	(0.73)	(0.91)	(0.83)
Mean	79.67	75.33	87.83	88.33	81.16	76.33	68.67	89	**68.33**
Mean DUR	0.9	0.85	1.02	1.03	0.92	0.86	**0.77**	1.04	0.78
Median DUR	0.9	0.84	0.94	0.97	0.92	0.79	**0.74**	1.01	0.76

similar to those used by Abe et al. (1998), Melville et al. (2004), and Culver et al. (2006) in the context of active learning.

4.4.2 Comparing Attribute (Column) Selectors and Instance (Row) Selectors

In our first set of experiments, we held the row selector fixed and compared the budgeted learning algorithms (which we sometimes refer to as "attribute selectors" or "column selectors"). Tables 4.3–4.5 show the target budgets and data utilization rates for the algorithms for each row selector. In those tables, the suffix of each algorithm's name denotes which row selector was used: "ec" for Error-Correction, "en" for Entropy, and "ur" for Uniform Random. For example, abr2.ec is the AbsoluteBR2 algorithm run with the Error-Correction row selector.

When using EC as the row selector (Table 4.3), RBR2 performed the best in terms

Table 4.4: Target budget and data utilization rates for algorithms with the EN row selector. Total budget was $B = 100$.

Dataset	exp3cr	exp3c.en	fel.en	rsfl.en	random.en	br.en	abr2.en	wbr2.en	rbr2.en
`breast-cancer`	42.0	100.0	100.0	100.0	100.0	100.0	59.0	50.0	46.0
	(0.61)	(1.45)	(1.45)	(1.45)	(1.45)	(1.45)	(0.86)	(0.72)	(0.67)
`colic-Nominalized`	78.0	100.0	100.0	95.0	100.0	100.0	68.0	96.0	82.0
	(0.89)	(1.14)	(1.14)	(1.08)	(1.14)	(1.14)	(0.77)	(1.09)	(0.93)
`kr-vs-kp`	90.0	82.0	100.0	100.0	98.0	82.0	100.0	100.0	69.0
	(0.96)	(0.87)	(1.06)	(1.06)	(1.04)	(0.87)	(1.06)	(1.06)	(0.73)
`mushroom`	86.0	76.0	94.0	85.0	84.0	78.0	81.0	70.0	63.0
	(0.91)	(0.81)	(1)	(0.9)	(0.89)	(0.83)	(0.86)	(0.74)	(0.67)
`vote`	100.0	58.0	79.0	51.0	87.0	57.0	55.0	76.0	85.0
	(1.14)	(0.66)	(0.9)	(0.58)	(0.99)	(0.65)	(0.62)	(0.86)	(0.97)
`zoo`	82.0	79.0	73.0	86.0	83.0	73.0	69.0	81.0	89.0
	(0.88)	(0.85)	(0.78)	(0.92)	(0.89)	(0.78)	(0.74)	(0.87)	(0.96)
Mean	79.67	82.5	91	86.17	92	81.67	**72**	78.83	72.33
Mean DUR	0.9	0.96	1.06	1	1.07	0.95	**0.82**	0.89	**0.82**
Median DUR	0.9	0.86	1.03	0.99	1.02	0.85	**0.81**	0.87	0.83

Table 4.5: Target budget and data utilization rates for algorithms with the UR row selector. Total budget was $B = 100$.

Dataset	exp3cr	exp3c.ur	fel.ur	rsfl.ur	random.ur	br.ur	abr2.ur	wbr2.ur	rbr2.ur
`breast-cancer`	42.0	46.0	18.0	100.0	69.0	66.0	45.0	51.0	52.0
	(0.61)	(0.67)	(0.26)	(1.45)	(1)	(0.96)	(0.65)	(0.74)	(0.75)
`colic-Nominalized`	78.0	89.0	82.0	100.0	88.0	100.0	90.0	93.0	82.0
	(0.89)	(1.01)	(0.93)	(1.14)	(1)	(1.14)	(1.02)	(1.06)	(0.93)
`kr-vs-kp`	90.0	74.0	89.0	100.0	94.0	100.0	81.0	77.0	91.0
	(0.96)	(0.79)	(0.95)	(1.06)	(1)	(1.06)	(0.86)	(0.82)	(0.97)
`mushroom`	86.0	80.0	77.0	85.0	94.0	75.0	76.0	72.0	65.0
	(0.91)	(0.85)	(0.82)	(0.9)	(1)	(0.8)	(0.81)	(0.77)	(0.69)
`vote`	100.0	100.0	100.0	76.0	88.0	35.0	100.0	100.0	77.0
	(1.14)	(1.14)	(1.14)	(0.86)	(1)	(0.4)	(1.14)	(1.14)	(0.88)
`zoo`	82.0	72.0	77.0	80.0	93.0	83.0	71.0	82.0	96.0
	(0.88)	(0.77)	(0.83)	(0.86)	(1)	(0.89)	(0.76)	(0.88)	(1.03)
Mean	79.67	76.83	**73.83**	90.17	87.67	76.5	77.17	79.17	77.17
Mean DUR	0.9	0.87	**0.82**	1.05	1	0.87	0.87	0.9	0.88
Median DUR	0.9	**0.82**	0.88	0.98	1	0.92	0.84	0.85	0.9

Table 4.6: Mean accuracies of all algorithms using the EC row selector at the minimum target budget.

dataset	exp3c.ec	fel.ec	rsfl.ec	random.ec	br.ec	abr2.ec	wbr2.ec	rbr2.ec
breast-cancer	**0.698**	0.676	0.632	0.693	0.690	0.693	0.659	0.675
colic-Nominalized	0.644	0.652	0.635	0.640	0.609	0.635	0.631	**0.676**
kr-vs-kp	0.569	0.567	0.558	0.563	0.558	0.572	**0.575**	0.555
mushroom	0.832	0.831	0.817	0.820	0.834	0.837	0.809	**0.841**
vote	0.883	0.879	0.881	0.880	0.885	**0.888**	0.879	0.886
zoo	0.769	0.765	0.771	0.776	0.781	**0.790**	0.778	0.774

of mean target budget, and ABR2 performed the best for mean and median DUR. When using EN as the row selector (Table 4.4), ABR2 performed the best in terms of mean target budget and median DUR, and ABR2 and RBR2 performed the best in terms of mean DUR. In contrast to EC and EN, when UR was used, FEL performed the best in terms of mean target budget and mean DUR, and Exp3C performed the best in terms of median DUR.

Next, we computed the mean classification accuracy of each algorithm after spending budget b_d, where b_d is the minimum target budget on data set d of all algorithms that use the same row selector. For example, for the EC row selector on data set `vote`, we considered all algorithms' mean accuracies after spending a budget of $b_{vote} = 57.0$ (Table 4.3). These average accuracies are presented in Tables 4.6–4.8. Tables 4.6 and 4.7 show that with the EC or EN row selector, ABR2 and RBR2 each has two wins, which is the most for any algorithm. In Table 4.8, we see that for the UR row selector, FEL has the most wins (again, two).

Our later Wilcoxon signed rank test of all algorithms at the minimum target budget show that ABR2 is consistently one of the top 2 performers of the algorithms using same row selector.

In our second set of experiments, we held the column selector fixed and compared the row selectors. Referring back to Tables 4.3–4.5, we set each budget b_d as the minimum target budget on data set d of all algorithms that use the same column selector. For example,

Table 4.7: Mean accuracies of all algorithms using the EN row selector at the minimum target budget.

dataset	exp3c.en	fel.en	rsfl.en	random.en	br.en	abr2.en	wbr2.en	rbr2.en
breast-cancer	0.647	0.626	0.630	0.592	0.679	0.691	**0.694**	0.692
colic-Nominalized	0.639	0.630	0.633	0.617	0.624	**0.679**	0.652	0.664
kr-vs-kp	0.566	0.550	0.566	0.553	0.565	0.554	0.557	**0.575**
mushroom	0.821	0.811	0.814	0.817	0.817	0.819	0.823	**0.843**
vote	0.884	0.879	**0.890**	0.862	0.884	0.884	0.880	0.868
zoo	0.762	0.784	0.748	0.752	0.783	**0.792**	0.775	0.769

Table 4.8: Mean accuracies of all algorithms using the UR row selector at the minimum target budget.

dataset	exp3c.ur	fel.ur	rsfl.ur	random.ur	br.ur	abr2.ur	wbr2.ur	rbr2.ur
breast-cancer	0.682	**0.692**	0.660	0.670	0.674	0.667	0.677	0.681
colic-Nominalized Nominalized	0.666	**0.672**	0.669	0.657	0.656	0.667	0.656	0.671
kr-vs-kp	**0.575**	0.571	0.564	0.555	0.558	0.569	0.572	0.563
mushroom	0.820	0.827	0.817	0.809	0.826	0.827	0.824	**0.844**
vote	0.869	0.872	0.878	0.839	**0.889**	0.872	0.865	0.875
zoo	0.792	0.779	0.753	0.757	0.779	**0.794**	0.780	0.763

when considering the set of algorithms $\mathcal{A} = \{$exp3c.ec, exp3c.en, exp3c.ur$\}$, we have $b_{zoo} = \min\{77.0, 79.0, 72.0\} = 72.0$. Next, we computed the mean classification accuracy of each algorithm after spending budget b_d, and reported these values in Tables 4.9 and 4.10. From these two tables, we can see that Random and BR with EC row selector at the mininum target budget reaches a higher mean accuracy for more data sets than Random and BR with any other row selector. ABR2 with EN row selector at the mininum target budget reaches a higher mean accuracy for more data sets than ABR2 with any other row selector. The result is consistent with our following Wilcoxon signed rank test.

Table 4.9: Mean accuracies of all algorithms using the EC, EN, and UR column selectors at the minimum target budget.

dataset	exp3c.ec	exp3c.en	exp3c.ur	fel.ec	fel.en	fel.ur	rsfl.ec	rsfl.en	rsfl.ur	rand.ec	rand.en	rand.ur
breast-cancer	**0.698**	0.638	0.689	0.675	0.618	**0.692**	0.664	0.660	**0.668**	**0.698**	0.595	0.675
colic-Nominalized	**0.672**	0.647	0.667	**0.673**	0.630	0.668	0.669	**0.673**	0.670	**0.672**	0.632	0.668
kr-vs-kp	0.569	0.568	**0.575**	0.573	0.563	**0.576**	0.569	0.568	**0.570**	**0.574**	0.568	0.570
mushroom	0.841	**0.844**	0.837	**0.844**	0.819	0.837	**0.843**	0.841	0.842	**0.843**	0.842	0.833
vote	0.882	**0.887**	0.879	0.880	**0.887**	0.880	0.880	**0.890**	0.886	0.883	**0.888**	0.886
zoo	0.787	0.772	**0.790**	0.775	**0.796**	0.785	**0.797**	0.758	0.785	**0.795**	0.775	0.756

Table 4.10: Mean accuracies of all algorithms using the EC, EN, and UR column selectors at the minimum target budget.

dataset	br.ec	br.en	br.ur	abr2.ec	abr2.en	abr2.ur	wbr2.ec	wbr2.en	wbr2.ur	rbr2.ec	rbr2.en	rbr2.ur
breast-cancer	**0.694**	0.679	0.688	**0.693**	0.680	0.689	0.669	0.695	**0.699**	**0.694**	0.692	0.689
colic-Nominalized	**0.670**	0.641	0.666	0.652	**0.679**	0.663	0.661	0.669	**0.676**	**0.676**	0.649	0.655
kr-vs-kp	0.559	**0.573**	0.556	0.573	0.562	**0.579**	**0.575**	0.557	0.572	0.558	**0.575**	0.555
mushroom	**0.844**	0.832	0.838	**0.847**	0.835	0.835	0.826	**0.843**	0.841	0.840	**0.843**	0.839
vote	0.861	0.871	**0.889**	0.886	**0.887**	0.882	0.884	0.887	0.886	0.886	0.870	**0.880**
zoo	0.786	**0.790**	0.781	0.790	**0.791**	0.784	0.785	0.787	**0.795**	**0.791**	0.775	0.768

4.4.3 Comparisons via Algorithms Using Wilcoxon Tests

For a rigorous analysis, we did a Wilcoxon signed rank test (Wilcoxon, 1945) to compare every two algorithms of a group of algorithms at their minimum target budget. For a Wilcoxon signed rank test, we first have a hypothesis which is $H_0 : \theta = 0$ meaning that there is no difference for the reached classification accuracies of two algorithms at a given budget. Each algorithm has 100 achieved accuracies (10 iterations times 10 folds) at a given budget. Let A_i and B_i be the i-th achieved accuracy of the two algorithms, and let $Z_i = A_i - B_i$ for $i = 1, ..., 100$. A Wicoxon signed ranked test procedure is as follows: (1) Observations of $Z_i = 0$ are excluded. Let m be the reduced sample size. (2) Order the absolute values $|Z_1|, \ldots, |Z_m|$ in ascending sequence, and let the rank of each non-zero $|Z_i|$ be R_i (the smallest positive $|Z_i|$ gets the rank of 1, and a mean rank is assigned to tied scores). (3) Denote the positive Z_i values with $\varphi_i = I(Z_i > 0)$, where $I(.)$ is an indicator function: $\varphi_i = 1$ for $Z_i > 0$, otherwise $\varphi_i = 0$. (4) The Wilcoxon signed ranked statistic

Table 4.11: The Wilcoxon-based comparison of two algorithms at the minimum target budget of all the algorithms listed in the table for data sets breast-cancer, colic-Nominalized, kr-vs-kp, mushroom, vote, and zoo. "++0−−+" means that the left side algorithm compared to the top side algorithm, is significantly better, significantly better, no significant difference, significantly worse, significantly worse, and significantly better at $p < 0.05$ level for the 6 data sets.

	exp3cr	exp3c.ec	fel.ec	rsfl.ec	rand.ec	br.ec	abr2.ec	wbr2.ec	rbr2.ec	Wins	Losses
exp3cr	000000	+000−−	+000+−	+0+−−+	+000++	−000−−	−000−−	+000−−	+−00−+	12	15
exp3c.ec	−000++	000000	0000++	+00−−+	+−0−++	0+00−+	−000−+	0000−+	−−00−+	14	12
fel.ec	−000−+	0000−−	000000	+−00++	+000++	0000−−	−−00−−	+000−−	+0+0−+	11	14
rsfl.ec	−0−++−	−00++−	−+00−−	000000	0−00+−	−+00−−	−+00−−	−+−0−−	−000+−	10	22
rand.ec	−000−−	−+0+−−	−000−+	0+00−+	000000	−+00−−	−0++−+	−0+0−−	−−00−+	11	20
br.ec	+000++	0−00+−	0000++	+−00++	+−00++	000000	+0+0++	0−00+−	+000+−	19	7
abr2.ec	+000++	+000+−	++00++	+−00++	+0−−+−	00−0−−	000000	+000+−	0000++	18	9
wbr2.ec	−000++	0000+−	−000++	+−+0++	+0−0++	0+00−+	−000−+	000000	0−0+−+	17	10
rbr2.ec	−+00+−	++00+−	−0−0+−	+000−+	++00+−	−000−+	00−0−−	0+0−+−	000000	14	15

Table 4.12: The Wilcoxon-based comparison of two algorithms at the minimum target budget of all the algorithms listed in the table for data sets breast-cancer, colic-Nominalized, kr-vs-kp, mushroom, vote, and zoo.

	exp3cr	exp3c.en	fel.en	rsfl.en	rand.en	br.en	abr2.en	wbr2.en	rbr2.en	Wins	Losses
exp3cr	000000	++00−−	++00−−	0000−+	++00++	++00−−	+000−+	+000+−	+000−+	17	10
exp3c.en	−−00++	000000	0++0−−	0000++	+000++	0+00++	−−00+−	0000++	−−+0++	18	9
fel.en	−−00++	0−−00+	000000	0−000+	+000++	00−0++	−−0−0+	−000++	−−00−+	13	13
rsfl.en	0000+−	0000−−	0+000−	000000	0000++	0+0−+−	−0−0−+	−0−−+−	−000−−	8	16
rand.en	−−00−−	−000−−	−000−−	0000−−	000000	−0−0−+	−−00−−	−000−−	−000−−	1	25
br.en	−−00+−	0−00−−	00+0−−	−−0+−+	+0+0+−	000000	−−00+−	−000+−	−−+00+	11	19
abr2.en	−000++	+000−+	++0+0−	+0+0+−	++00++	++00−+	000000	−−00++	−0+0++	22	8
wbr2.en	−000−+	0000−−	+000−−	+0++−+	+000++	+000−+	++00−−	000000	−000−−	13	13
rbr2.en	−000+−	++−0−−	++00+−	+000++	+000++	++−00−	+0−0−−	+000++	000000	18	11

W_+ is defined as $W_+ = \sum_{i=1}^{n} \varphi_i R_i$. Define W_- similarly by summing ranks of the negative differences Z_i. (5) Calculate S as the smaller of these two rank sums: $S = \min(W_+, W_-)$. (6) Find the critical value for the given sample size m and the wanted confidence level. (7) Compare S to the critical value, and reject H_0 if S is less than or is equal to the critical value.

The results of comparison between algorithms with different row selectors are shown in Table 4.14−4.22. We first compare algorithms with different row selectors, and then we compare algorithms with different column selectors. Finally, we compare all the algorithms.

Table 4.13: The Wilcoxon-based comparison of two algorithms at the minimum target budget of all the algorithms listed in the table for data sets breast-cancer, colic-Nominalized, kr-vs-kp, mushroom, vote, and zoo.

	exp3cr	exp3c.ur	fel.ur	rsfl.ur	rand.ur	br.ur	abr2.ur	wbr2.ur	rbr2.ur	Wins	Losses
exp3cr	000000	+−00+−	0−−0−−	0−000−	+00+++	+000−−	−0+0−−	0000++	+−0+++	14	14
exp3c.ur	−+00−+	000000	0−00−+	−−00+−	++00++	−−00−+	−000−−	−−00++	+0000+	13	15
fel.ur	0++0++	0+00+−	000000	+0000−	0++0++	+000−+	−000−−	00+0++	0−00+−	17	8
rsfl.ur	0+000+	++00−+	−0000+	000000	+++00+	+++0−+	−000−−	++00++	+−++0+	22	7
rand.ur	−00−−−	−−00−−	0−−0−−	−−−00−	000000	+000−−	0000−−	−0000+	−−00−−	2	25
br.ur	−000++	++00+−	−000+−	−−−0+−	−000++	000000	−−−0+−	−00++−	−000+−	13	17
abr2.ur	+0−0++	+000++	+000++	+000++	0000++	+++0−+	000000	−+00++	0−000+	22	4
wbr2.ur	0000−−	++00−−	00−0−−	−−00−−	+0000−	+00−−+	+−00−−	000000	+−000+	8	18
rbr2.ur	−+0−−−	−0000−	0+00−+	−+−−−−	++00++	+000−+	0+000−	−+000−	000000	12	16

Table 4.14: The Wilcoxon-based comparison of two algorithms at the minimum target budget of all the algorithms listed in the table for data sets breast-cancer, colic-Nominalized, kr-vs-kp, mushroom, vote, and zoo.

	exp3c.ec	exp3c.en	exp3c.ur	Wins	Losses
exp3c.ec	000000	0000−+	00−0+−	2	3
exp3c.en	0000+−	000000	−000++	3	2
exp3c.ur	00+0−+	+000−−	000000	3	3

Table 4.15: The Wilcoxon-based comparison of two algorithms at the minimum target budget of all the algorithms listed in the table for data sets breast-cancer, colic-Nominalized, kr-vs-kp, mushroom, vote, and zoo.

	fel.ec	fel.en	fel.ur	Wins	Losses
fel.ec	000000	+++0−−	−+00+−	5	4
fel.en	−−−0++	000000	−−0−++	4	6
fel.ur	+−00−+	++0+−−	000000	5	4

Table 4.16: The Wilcoxon-based comparison of two algorithms at the minimum target budget of all the algorithms listed in the table for data sets breast-cancer, colic-Nominalized, kr-vs-kp, mushroom, vote, and zoo.

	rsfl.ec	rsfl.en	rsfl.ur	Wins	Losses
rsfl.ec	000000	0000−−	−+00+−	2	4
rsfl.en	0000++	000000	−−0−++	4	3
rsfl.ur	+−00−+	++0+−−	000000	5	4

Table 4.17: The Wilcoxon-based comparison of two algorithms at the minimum target budget of all the algorithms listed in the table for data sets breast-cancer, colic-Nominalized, kr-vs-kp, mushroom, vote, and zoo.

	rand.ec	rand.en	rand.ur	Wins	Losses
rand.ec	000000	++00−+	−+0+−+	6	3
rand.en	−−00+−	000000	−−00++	3	5
rand.ur	+−0−+−	++00−−	000000	4	5

Table 4.18: The Wilcoxon-based comparison of two algorithms at the minimum target budget of all the algorithms listed in the table for data sets breast-cancer, colic-Nominalized, kr-vs-kp, mushroom, vote, and zoo.

	br.ec	br.en	br.ur	Wins	Losses
br.ec	000000	−+−00+	+−000+	4	3
br.en	+−+00−	000000	0−00−+	3	4
br.ur	−+000−	0+00+−	000000	3	3

Table 4.19: The Wilcoxon-based comparison of two algorithms at the minimum target budget of all the algorithms listed in the table for data sets breast-cancer, colic-Nominalized, kr-vs-kp, mushroom, vote, and zoo.

	abr2.ec	abr2.en	abr2.ur	Wins	Losses
abr2.ec	000000	+000+−	+000+−	4	2
abr2.en	−000−+	000000	−000++	3	3
abr2.ur	−000−+	+000−−	000000	2	4

Table 4.20: The Wilcoxon-based comparison of two algorithms at the minimum target budget of all the algorithms listed in the table for data sets breast-cancer, colic-Nominalized, kr-vs-kp, mushroom, vote, and zoo.

	wbr2.ec	wbr2.en	wbr2.ur	Wins	Losses
wbr2.ec	000000	+000++	−000−+	4	2
wbr2.en	−000−−	000000	+000−+	2	4
wbr2.ur	+000+−	−000+−	000000	3	3

Table 4.21: The Wilcoxon-based comparison of two algorithms at the minimum target budget of all the algorithms listed in the table for data sets breast-cancer, colic-Nominalized, kr-vs-kp, mushroom, vote, and zoo.

	rbr2.ec	rbr2.en	rbr2.ur	Wins	Losses
rbr2.ec	000000	−000−0	−+00++	3	3
rbr2.en	+000++	000000	++00++	7	0
rbr2.ur	+−00−−	−−00−−	000000	1	7

Table 4.22: The Wilcoxon-based comparison of two algorithms at the minimum target budget of all the algorithms listed in the table for data sets breast-cancer, colic-Nominalized, kr-vs-kp, mushroom, vote, and zoo.

	(1)exp3cr	(2)exp3c.ec	(3)fel.ec	(4)rsfl.ec	(5)rand.ec	(6)br.ec	(7)abr2.ec	(8)wbr2.ec	(9)rbr2.ec	(10)exp3c.en	(11)fel.en	(12)rsfl.en	(13)rand.en	(14)br.en	(15)abr2.en	(16)wbr2.en	(17)rbr2.en	(18)exp3c.ur	(19)fel.ur	(20)rsfl.ur	(21)rand.ur	(22)br.ur	(23)abr2.ur	(24)wbr2.ur	(25)rbr2.ur	Wins	Losses
exp3cr	000 000	+00 0+−	+−0 0+−	0−0 −++	+00 0++	0+0 00−	−00 00−	−00 +0−	+00 0++	0+0 0−−	++0 0+−	000 00+	+00 0++	++0 00−	−00 0−−	+00 0+−	+00 0−+	+00 0+−	0−0 0−−	0−0 00−	+−0 +++	++0 0−+	−0+ 0−−	000 0++	+00 0+−	42	31
exp3c.ec	−00 0−+	000 000	+−0 0++	+−0 0−+	+00 0++	0+0 0−+	−+0 0−+	−+0 00+	++0 0−+	+−− 0−+	000 0−+	−+0 0−+	+00 00+	−+0 0−+	−−0 0−+	−00 0++	0−0 0++	−00 0++	−0− 0−+	+0− 0−+	0−+ 0++	0++ 0−+	000 0−+	+00 0++	−00 0++	50	32
fel.ec	−+0 0−+	−+0 0−−	000 000	+−0 0++	+00 0+−	++0 0−−	0−0 0−−	−00 0+−	−00 0++	+00 0−−	+++ 0−−	000 0++	++0 0++	−+0 −0−	−00 0−−	−+0 0+−	−0+ 00−	−+0 0+−	−+0 0+−	−00 0−−	−−0 0++	−00 0−+	−−0 0−−	−0+ 0+−	−+0 00−	37	48
rsfl.ec	0+0 +−−	−+0 0+−	−+0 0−−	000 000	−+− 0+−	−+0 00−	−+0 00−	−+− 0+−	−00 +0−	−+0 +−−	0+0 00−	−+− +−−	++0 0+−	−+0 0+−	−+0 0−−	−00 0+−	−+0 00−	−+− 0+−	−+0 0−−	−+0 0+−	−+0 0+−	−+0 0−−	−+0 0+−	−+0 0+−	−+0 0−−	38	56
rand.ec	−00 0−−	−00 0−−	−00 0−+	+−+ 0−+	000 000	−00 00−	−00 +−+	−0+ 0−−	0−+ 0−+	000 0−−	+00 0−−	00+ 0−−	+0+ 00+	−00 0−−	−+0 +−−	−00 0−−	0−+ −−−	−+0 0−−	−−0 0−−	−00 0−−	−−0 0++	−−+ 0−+	000 00+	−00 00−	−0+ 0−−	24	58
br.ec	0−0 00+	0−0 0+−	−−0 0++	+−0 00+	+00 00+	000 000	−−0 00+	−−0 00−	0−0 00+	+00 0−+	+−0 00−	+−0 00+	+00 0++	0+− 00+	−−0 00+	+−0 00−	−−0 00−	+−0 0+−	0−0 00+	0−0 00+	0−0 0++	−00 00+	−00 0+−	−00 0++	−−0 0+−	33	36
abr2.ec	+00 00+	+−0 0+−	0+0 0++	+−0 00+	+00 −+−	++0 00−	000 000	0−0 0+−	+0+ 0++	+00 00−	+00 00−	0−0 ++−	+0+ 0+−	++0 00+	0−0 0−−	+00 0−−	+00 0−+	+0− 0+−	+00 0+−	+00 0++	+00 −++	++0 0−+	+−0 00−	+−0 0+−	+−0 0++	50	29
wbr2.ec	+00 −0+	+−0 00−	+00 0−+	+−+ 0−+	+0− 0++	++0 00+	0+0 0−+	000 000	+00 0−+	++0 0−+	+−0 0−−	+00 0−+	+++ 00+	++0 −−+	+00 0−−	+00 0−+	+00 0−+	++− 0++	0−− 0−+	+0− 0++	+0+ 0++	+00 0−+	+00 0−+	000 0++	+−0 0−−	55	30
rbr2.ec	−00 0−−	−−0 0+−	+00 0−−	+00 −0+	0+− 0+−	0+0 00−	−0− 0−−	−00 0+−	000 000	−+0 0−−	+00 00−	000 0−−	++0 0++	−+0 0−−	−0− −−−	−00 0−−	−00 0−−	000 0+−	00− 00−	−+− 0+−	−−0 00+	−+0 0−−	+00 00−	−00 0+−	−+0 0+−	24	54
exp3c.en	0−0 0++	−−+ 0+−	−00 0++	+−0 −++	000 0++	−00 0+−	−00 00+	−−0 0+−	+−0 0++	000 000	+0+ 00−	0−0 00+	+−0 0++	−00 0−+	−−0 0+−	000 0++	−−+ 0++	000 0++	−−0 0+−	+0+ 0++	+−0 0++	000 00+	−00 0+−	−−0 0++	+−0 0++	49	32
fel.en	−−0 0−+	000 0+−	−−− 0++	0−0 00+	−00 0++	−+0 00+	−00 00−	−+0 0++	−00 00+	−0− 00+	000 000	−00 0++	+00 0++	−0− 0++	−00 −0+	−00 0++	−00 0++	−−− 0++	−00 0−+	−−0 −++	−00 00+	−00 0−+	−0− 0−+	−00 0++	−00 00−	36	40
rsfl.en	000 00−	+−0 0+−	000 0−−	+−+ −++	00− 0++	−+0 00−	−+0 −−+	−00 0+−	000 0++	0+0 00−	+00 0−−	000 000	+00 0++	−00 −+−	+0− 00−	−0− −++	−−0 0−−	−0− 0+−	−0− 0+−	−00 0−+	+00 0++	0+0 +−−	0+0 0+−	−00 0+−	−−0 0++	35	46
rand.en	−00 0−−	−00 00−	−−0 0−−	−−0 0−+	−0− 00−	−00 0−−	−0− 0−+	−−− 00−	−−0 0−−	−+0 0−−	−00 0−−	−00 0−−	000 000	−0− 0−−	−−0 0−−	−00 0−−	−00 0−−	−00 0−−	−00 0−−	−+0 0−−	−−0 0−−	−00 0−+	−0− 0−−	−−0 00−	−−0 0−−	5	78
br.en	−−0 00+	+−0 0+−	+−0 +0+	+−0 0−+	+00 0++	0−+ 00−	−−0 00−	−−0 ++−	+−0 00+	+00 0+−	+0+ 0−−	+00 +−+	+0+ 0++	000 000	0−0 00−	000 00−	−−+ 00−	−−0 0+−	−−0 00−	+−0 0+−	+00 0++	+−0 0−+	−−0 0++	0−0 0++	+−0 +++	43	41
abr2.en	+00 0++	++0 0+−	+00 0++	+−0 0++	+−0 −++	++0 00−	0+0 0++	−00 0++	+0+ +++	++0 0−+	+00 +0−	−0+ 00+	++0 0++	0+0 00+	000 000	++0 0++	+0+ 0−+	+00 0++	+00 +++	0+0 0++	+0+ 0++	+++ 00+	0−0 0++	+00 0++	000 0++	72	11
wbr2.en	−00 0−+	+00 0−−	+−0 0−+	+00 0−+	+00 0++	−+0 00+	−00 0++	−00 0+−	+00 0++	000 0−−	+00 0−−	+0+ +−−	+00 0++	000 00+	−−0 0−−	000 000	−00 0−−	0+0 0+−	−00 0−+	−−0 0++	−0+ +++	+00 0−−	000 0−−	−−0 0++	+−0 00+	39	37
rbr2.en	−00 0+−	0+0 0−−	+0− 00+	+−0 00+	00− +++	++0 00+	−00 0+−	−00 0+−	+00 0++	++− 0−−	+00 0−−	++0 0++	+00 0++	++− 00−	−0− 0+−	+00 0++	000 000	0+0 0+−	−00 00+	−−− 00+	−−0 0++	++0 0−+	−00 00−	−+− 0++	++0 0++	49	33
exp3c.ur	−00 0−+	+00 0−−	+−0 0−+	+−+ 0−+	+−0 0++	−+0 0−+	−00 0−+	−−+ 0−−	000 0−+	000 −−−	+++ 0−−	+0+ 0−+	+00 0++	++0 0−+	−00 0−−	0−0 0−+	000 0−+	000 000	−−0 0−+	+0+ 0++	−−+ 0−+	−00 0−−	+00 0++	−00 0++	+00 00+	42	41
fel.ur	0+0 0++	++0 0+−	+−0 0−+	+−0 0++	++0 0++	0+0 00−	−00 0−+	00+ 0+−	00+ 00+	+++ 0−−	+00 0+−	+0+ 0−+	+00 0++	++0 00+	−00 −−−	+00 0+−	+00 00−	0+0 0+−	000 000	+−0 00+	0−+ 0++	+00 +−+	0−0 +−−	0+0 0++	54	27
rsfl.ur	0+0 00−	−0+ 0+−	+00 0++	+−0 0−+	+00 0++	0+0 00−	−00 +−−	−0+ 0−−	+−+ 0−+	−0− 0−−	++0 +−−	+00 0−+	+00 0++	+−0 0−+	−+0 0−−	0−0 0−−	++0 00−	+++ 0−−	++0 00−	−00 000	000 00+	+−+ 0−+	+++ 0−−	−00 0++	++0 +0+	+00	48	39
rand.ur	−+0 −−−	−+− 0−−	++0 0−−	+−0 0−+	0+0 0−−	0+0 0−−	−00 +−−	−0− 0−−	++0 00−	−+0 0−−	+00 0−−	−00 0−−	++0 0++	−00 0−−	−0− 0++	+0− 0−−	++0 −−−	−0− 0−−	0+− 0−−	−+− 00−	000 000	++0 0−−	000 0−−	−00 00−	−00 0−−	24	64
br.ur	−−0 0+−	0−− 0+−	+00 0+−	+−0 0++	++− 0+−	+00 00−	−−0 0+−	−00 0+−	+−0 0++	000 00−	+00 0+−	0−0 −++	+00 0+−	−+0 0+−	−−− 0+−	−00 0++	−−0 0+−	++− 0+−	−00 −+−	−−− 0+−	−−0 0++	000 000	−00 0+−	−−0 0+−	−−0 0+−	38	51
abr2.ur	+0− 0++	+00 0+−	++0 0++	+−0 0−+	000 0++	+00 0−+	−+0 00+	−00 0++	−00 00+	+00 0−+	+0+ 0+−	0−0 0−+	+0+ 0++	++0 0−+	0+0 0−−	000 0++	+00 00+	+00 0++	+−− −++	+−0 0++	000 0++	+00 0−+	000 000	−00 0++	000 00+	54	21
wbr2.ur	000 0−−	−00 0−−	+0− 0−+	+−0 0−+	+00 0+−	+00 0−−	−+0 0−+	0−0 0−−	+−0 0−+	++0 0−−	+00 0−−	+00 0−+	++0 00+	0+0 0−−	−00 0−−	++0 0−−	+−+ 0−−	+00 0−−	0+0 0−−	−−0 0−−	+00 00+	++0 0−+	+00 0−−	000 000	+−0 00−	33	47
rbr2.ur	−00 0−+	+00 0−−	+−0 00+	+−0 00+	+0− 00+	++0 0−+	−+0 0−−	−+0 0+−	+−0 0−+	−+0 0−−	+00 00+	++0 0−+	++0 0++	−+0 −−−	−00 0−−	−+0 00−	−−0 0−−	−00 00−	0−0 0−+	−00 −−−	+00 0++	++0 0−+	000 00−	−+0 00+	000 000	37	43

In our first set of experiments, we held the row selector fixed and compared the budgeted learning algorithms (which we sometimes refer to as "attribute selectors" or "column selectors"). Tables 4.11–4.13 show the results of our Wilcoxon-based comparison between each algorithm on the left side of the table to each algorithm on the top of the table. When the left side algorithm is significantly better (i.e. reaches a significantly higher classification accuracy) than the top side algorithm at $p < 0.05$ level for a data set, a "+" sign is shown there; when the top side algorithm is significantly better than the left side algorithm at $p < 0.05$ level for a data set, a "−" sign is shown there; when the left side algorithm and the top side algorithm are not significantly different, a "0" is shown there. Thus , "++0−−+" means the first algorithm compared to the second algorithm, is significantly better, significantly better, no significant difference, significantly worse, significantly worse, and significantly better at $p < 0.05$ level for the 6 data sets.

In Table 4.11, when all algorithms use the EC row selector (except Exp3CR), BR.ec has the largest number of wins and the smallest number of losses, followed by ABR2.ec which has the second largest number of wins and the second smallest number of losses, therefore BR and ABR2 outperform other algorithms with EC row selector. In Table 4.12, when all algorithms use the EN row selector (except Exp3CR), ABR2.en has the largest number of wins and the smallest number of losses, followed by Exp3C.en which has the second largest number of wins and the second smallest number of losses. Therefore, ABR2 and Exp3C outperform other algorithm with EN row selector. In Table 4.13, when all algorithms use the UR row selector (except Exp3CR), ABR2.ur has the largest number of wins and the smallest number of losses, followed by RSFL.ur which has the second largest number of wins and the second smallest number of losses. Therefore, ABR2 and RSFL are the best algorithms which outperform other algorithms with the UR row selector.

In our second set of experiments, we held the column selector fixed and compared the row selectors. Tables 4.14–4.21 show the results of our Wilcoxon-based comparison between

each algorithm on the left side of the table to each algorithm on the top side of the table. From these tables, we can see that when comparing algorithms with same column selectors at their mininum target budget, EC stands out for Random. EC has an advantage over other row selectors for BR, ABR2, and WBR2. EN stands out for RBR2. EN has an advatnage over other row selectors for Exp3C. The result that EC stands out for Random and BR via Wilcoxon test is consistent with the previous result of the mean accuracies of the algorithms at their minimum targe budget.

In our third experiment, we compare all 25 algorithms. Table 5.12 shows the results of our Wilcoxon-based comparison between each algorithm on the left side of the table to each algorithm on the top side of the table. From this table, we can see that ABR2.en has the largest number of wins and the smallest number of losses. Therefore, ABR2 with EN row selector outperforms all other algorithms at the minimum target budget of all algorithms in the table. ABR2 with other row selectors also perfrom well at the minimum target budget. Also performing well are WBR2 with EC row selector, Exp3C with EC row selector, and FEL with UR row selector.

In conclusion, BR and ABR2 stand out for all algorithms with EC row selectors; ABR2 and Exp3C stands out for all algorithms with EN row selectors; ABR2 and RSFL stands out for all algorithms with UR row selectors. We can conclude that the EC row selector stands out for Random. ABR2 with all row selectors, WBR2 and Exp3C with EC row selector, and FEL with UR row selector perform well at the minimum target budget of all the algorithms.

4.5 Conclusions and Future Work

We presented new algorithms for the budgeted learning problem (choosing which attribute to purchase at each step), many showing improvement over the state of the art, for example ABR2, WBR2, Exp3C, and FEL. We also described variations on how to select the row

(instance) to purchase an attribute of, specifically, selecting the row with most uncertainty or with the row with the maximum entropy in the current model. For the same algorithm using different row selectors, EC stands out for Random and EN stands out for RBR2. For different algorithms using the same row selector, BR and ABR2 stands out for all algorithms with EC row selectors; ABR2 and Exp3C stands out for all algorithms with EN row selectors; ABR2 and RSFL stands out for all algorithms with UR row selectors. When comparing all algorithms, ABR2 with all row selectors, WBR2 and Exp3C with EC row selector, and FEL with UR row selector perform well.

There are several other directions for future work. First, while there are some theoretical results for the coins problem, there are no learning-theoretic results (e.g. PAC-style results) for budgeted learning problems. Our use of Auer et al.'s multi-armed bandit algorithms to this problem (Section 4.3.1) may ultimately yield such a result. However, in order to get a PAC-style bound for our algorithm, one needs to relate regret bounds or one shot bounds such as those in Bubeck (2008) to generalization error bounds of the final model.

Second, future work includes extensions to the basic budgeted learning model in the context of bandit-based algorithms. In particular, the budgeted learning of bounded active classifiers (BACs) (Kapoor and Greiner, 2005b,a). Third, treating class label and attribute pair as a bandit by separating the rewards for different class labels. Finally, exploring different base learners other than naïve Bayes (such as support vector machines) is another direction for future research.

Chapter 5

Budgeted Learning of Bayesian Networks (BLBN)

5.1 Introduction

We continue our study of budgeted learning under the setting that the labels of the training data are given, however the features are available at a cost, subject to an overall budget. Many budgeted learning algorithms assume an overly simplistic probabilistic data model, such as naïve Bayes, which assumes that the features of the training data are independent from each other. However, it is possible that the features of the training data are correlated with each other. What if we know the dependencies of the features? Can we learn a more accurate classifier? Can we exploit the dependencies of the features to choose (instance, feature) pairs to purchase so that we can learn a better classifier/hypothesis? To answer these questions, we propose algorithms that exploit the information among the dependencies of the features and class label to choose (instance, feature) pairs to purchase.

When the features of the training data have dependencies, we use a Bayesian network to represent the dependencies of the features and class label. A Bayesian network is an acyclic

graph with a joint probability distribution satisfying the Markov condition, that is, every variable in this graph is conditionally independent of the set of all its non-descendants given the set of all its parents. Usually a link from node A (the parent) to node B (the child) indicates that A causes B, that A partially causes or predisposes B, that B is an imperfect observation of A, that A and B are functionally related, or that A and B are statistically correlated.

Currently there is no paper related to budgeted learning of Bayesian networks under the setting that the labels of the instances are known and the feature values have to be purchased. Tong and Koller (2001a) studied estimating parameters of the Byesian networks in active learning setting that the labels of the instances are unknown. Li et al. (2010) studied learning the parameters for Bayesian networks under the budgeted learning setting that both the labels and the features of the training data have to be purchased at a cost. We will talk about the differences between our work and their work in Section 5.2.

When the structure of a Bayesian network of the features and class label is given, either from an expert or from the learning of previous cases or a combination of these two, the improved probabilistic model might improve the accuracies of the objective functions of existing budgeted learning algorithms. One of the results of this chapter is the adaption of the existing algorithms that work on naïve Bayes to Bayesian networks. Those algorithms work well with naïve Bayes and we want to know whether adapting them to a Bayesian network will learn a more accurate classifier. For those algorithms with an objective function, we will check whether the change of probabilistic model from naïve Bayes to Bayesian network and the possible improved accuracy of the objective function that is based on the learned classifier will help the choice of (instance, feature) pairs[1].

Besides the result of adapting existing algorithm to Bayesian networks to determine if

[1]Note that the question is distinct from the question of whether accuracy will improve simply due to the change of classifier from naïve Bayes to Bayesian network.

improved probabilistic models help them in their purchase choices, the other main result of this chapter are new algorithms that exploit the improved representational power of Bayesian networks over naïve Bayes classifiers, afforded by the relationship among features encoded in its structure to choose (instance, feature) pairs to purchase, and take advantage of the known label. Our new algorithm, MERPG (Maximization of expected relative probability gain), at each purchase, chooses the (instance, feature) pair if purchasing it will cause the maximum relative expected improved accuracy of this instance among all the unpurchased (instance, feature) pairs; when there are several (instance, feature) pairs with same maximum expected improved accuracy, MERPG randomly chooses one. The motivation of MERPG is as follows. We want to know whether the choice of an (instance, feature) pair is better than another (instance, feature) pair. If we find that the purchase of one (instance, feature) pair and do the probability inference to find the new probability of the label node to be the true label has a much bigger improvement than other purchases, then we believe this purchase is better than other purchases.

Another new algorithm, MERPGDSEP, at each purchase, for the (instance, feature) pairs with the same maximum expected improved accuracy among all the (instance, feature) pairs, breaks the ties by choosing the (instance, feature) pair that leads to the maximum increase of the number of the unpurchased features d-separated from the label node. The motivation of MERPGDSEP is as follows. For any unpurchased feature of an instance that is d-separated from the label node, purchasing it in the future will not affect the label node, that is, purchasing this (instance, feature) will not change the probability of the label node to be its true label for this instance. We believe that the increased number of d-separations is a good criterion to measure the influence of this purchase. Breaking ties of MERPG by the increased number of d-separations is one application of this motivation. Is the metric of the increased number of d-separations powerful enough to be a weighting factor or a log weighting factor combined with MERPG? To test this, we also proposed another two

algorithms, MERPGDSEPW1 and MERPGDSEPW2, which, at each purchase, choose the (instance, feature) pair that maximizes the weighted expected improved accuracy, where the weighting is based on the increase of d-separation from the label node.

Note that the naïve Bayes classifier is a special case of Bayesian network in which all feature nodes are independent from each other given the class label and there is an edge from the label node to each feature. Therefore, all our proposed algorithms can be applied to the naïve Bayes classifier as well.

In our experiments, existing algorithms adapted to Bayesian networks sometimes outperform existing algorithms learning naïve Bayes. Further, our algorithms MERPG, MERPGDSEP, MERPGDSEPW1, and MERPGDSEPW2 often outperformed existing algorithms from the literature adapted to Bayesian networks.

Instead of choosing all features of instances to purchase, we believe that some features have greater influence than other features. When a Bayesian network has a large number of nodes, although the probability of the label node of one instance can be affected by instantiating a node distant to the label node, the nearby nodes of the label node have greater influence than those more distant nodes to the label node. Therefore, what if we purchase only those features of some instances that are near the label node? To answer this question, we ran all our algorithms on a subset of the network's nodes that are a Markov blanket (MB) of the label node, which consists of the label node's parents, children, and spouses (parents of its children). Algorithms running on a MB filter execute as follows: at each purchase, they only choose one (instance, feature) pair from the unpurchased (instance, feature) pairs whose features fall in the Markov blanket of the label node. Our experimental results show that algorithms running on the filter perform better than those that are not. The algorithms run on the filter also take significantly less time than those that are not.

The remaining sections are organized as follows. Section 5.2 describes related work. Section 5.4 presents existing budgeted learning algorithms and how we adapt them to run

with Bayesian networks. Section 5.5 presents our proposed algorithms for BLBN. Section 5.6 shows our experimental results on 5 data sets obtained from Norsys Net Library. Finally, Section 5.7 concludes our work of BLBN and presents the directions of our future work of BLBN.

5.2 Related Work

Learning Bayesian networks under the active learning setting has been done in Tong and Koller (2001a) for estimating parameters of the Bayesian network and in Tong and Koller (2001b) for structure learning of Bayesian networks. More details of parameter learning and structure learning of Bayesian networks can be found in Neapolitan (2004). In our work, we assume that we know the structure of the Bayesian network, which is similar to that of Tong and Koller (2001a). Tong and Koller decide on their next query given the current distribution, by choosing an instance label to purchase such that the risk of purchasing this instance is the lowest. The objective function for choosing the label of an instance to purchase is based on the learned distribution. Recall that the goal of active learning of the parameters of the Bayesian network is to learn a good generative distribution (i.e., a distribution that closes to its true distribution) of the Bayesian network using as little cost as possible. In budgeted learning, we are given a hard budget and the goal of budgeted learning is to use this given hard budget to learn as good distribution of the Bayesian network as it can. Most recent work of budgeted learning assumes that the features of the training data are independent from each other. In reality, it is possible that they are correlated with each other. Recently, Li et al. (2010) studied budgeted distribution learning under the setting that both the labels and the features are unknown. The goal of their work is also to learn the parameters of the Bayesian network. In Li et al.'s work, they also adopted the objective function that was used in Tong and Koller (2001a) for their budgeted learning to choose an (instance, feature)

or (instance, label) to purchase. Therefore, Li et al. and Tong and Koller choose the next (instance, feature) or (instance, label) to purchase based only on the current distribution and expected distribution if purchasing one (instance, feature) or (instance, label). In our work, our focus is different, since we already know the labels of the instances, we choose an (instance, feature) pair that maximizes the expected relative probability gain of predicting the instance as its correct label. Therefore, our approach is different from Tong and Koller (2001a) in that our objective function is based on the known labels of the instances. We also propose algorithms that are not only based on the relative probability gain but also consider the increase of the number of d-separated nodes from the label node when purchasing an (instance, feature). That is, our algorithms also incorporate d-separatedness of the Bayesian network. In addition, we propose choosing (instance, feature) pairs from a subset of unpurchased (instance, feature) pairs whose features fall in the Markov blanket of the label node to purchase for algorithms. The structure of a Bayesian network decides the Markov blanket of the label node. Therefore, our algorithms not only take advantage of the distribution of the Bayesian network, but also take advantage of the known labels, structure of the Bayesian network, and the dynamically changing d-separation of the Bayesian network.

5.3 Budgeted Learning of Bayesian Networks

For budgeted learning of Bayesian network (G, P), we are given a set of instances $X = \{x_1, \ldots, x_n\}$ whose labels are given as $Y = \{y_1, \ldots, y_n\}$ and whose feature values are unknown and are available at a cost c, subject to an overall budget b. In our study, we assume the cost is uniform, that is, purchasing any instance any feature incurs the same cost. We represent the features as $V = \{v_1, \ldots, v_m\}$ where m is the number of features. We are also given the correct structure G of a Bayesian network representing the relationships of the

features and the class label. Bayesian networks may be constructed using expert knowledge provided by some person, by an automatic learning process that examines many previous cases, or by a combination of the two. We set the initial counts in the conditional probability tables to be 1. The goal of budgeted learning of Bayesian networks is to use the given budget to learn as good a Bayesian network as it can. Our goals in this chapter are twofold: enhance existing budgeted learning algorithms with Bayesian networks and come up with new budgeted learning algorithms that exploit Bayesian networks.

5.4 Enhancing Existing Budgeted Learning Algorithms

We now describe existing budgeted learning algorithms and describe how they can enhanced by using a more precise probabilistic model via the use of the Bayesian networks. Section 5.4.1 introduces the Random algorithm. Section 5.4.2 introduces the Round Robin algorithm. Section 5.4.3 introduces the Biased Robin algorithm. Sections 5.4.4–5.4.7 describe the Single Feature Look-ahead (SFL) related algorithms, including SFL, Randomized SFL (RSFL), Generalized SFL (GSFL), and Generalized Randomized SFL (GRSFL), and how we extend them to use Bayesian networks.

5.4.1 Random (a.k.a. random)

While the budget is not yet used up, for each purchase, Random chooses an (instance, feature) pair uniformly at random from all the unpurchased (instance, feature) pairs. This algorithm cannot be adapted to take advantage of Bayesian network for the choice of (instance, feature) pairs. Algorithm 5.1 shows this algorithm.

Algorithm 5.1 Random(b, c) where b is the given budget and c is the unit cost of purchasing an (instance, feature) pair

while $b > 0$ **do**
  Choose $(x_{\text{chosen}}, V_{\text{chosen}})$ u.a.r. from the unpurchased (instance, feature) pairs;
  Purchase $(x_{\text{chosen}}, V_{\text{chosen}})$;
  $b = b - c$;
  Update the probability tables of the Bayesian Network;
end while

5.4.2 Round Robin (a.k.a. RR or rr)

While the budget is not yet used up, for each purchase, RR chooses an (instance, feature) pair in which the instance is randomly chosen and the feature is the next feature of a list of features. Algorithm 5.2 presents this method. This algorithm cannot be adapted to take advantage of Bayesian network for the choice of (instance, feature) pairs.

Algorithm 5.2 RR(b, c) where b is the given budget and c is the unit cost of purchasing an (instance, feature) pair

Assign V_{chosen} as a random feature of all the features $\{V_1, \ldots, V_m\}$.
while $b > 0$ **do**
  $V_{\text{chosen}} = V_{(\text{chosen}+1) \bmod m}$;
  // Guarantee that at least the chosen feature/node has unpurchased instances;
  while there are no unpurchased instances of the feature V_{chosen} **do**
    chosen = (chosen + 1) mod m;
  end while
  x_{chosen} is a uniformly at random (u.a.r) instance from the unpurchased instances of feature V_{chosen};
  Purchase $(x_{\text{chosen}}, V_{\text{chosen}})$;
  $b = b - c$;
  Update the probability tables of the Bayesian Network;
end while

5.4.3 Biased Robin (a.k.a. BR or br)

While the budget is not yet used up, for each purchase, BR chooses an (instance, feature) as follows: we first compute the log loss, which is the minus of the summation of log probability

of predicting the correct label for each instance,

$$\text{LogLoss}(w) = -\sum_{k=1}^{n} \log(P(y_k \mid x_k; w)) \ , \tag{5.1}$$

where w is a vector of the current parameters of the Bayesian network, x_i are the ith instance, and y_i is the label of the instance x_i. If the current log loss (the log loss of the current learned distribution) is bigger than or equal to the previous log loss (the log loss of the previous learned distribution before purchasing the current (instance, feature) pair), we choose the previously chosen feature, otherwise we choose the next feature. The instance is randomly chosen from a uniform distribution. The computation of LogLoss is based on the distribution of the learned Bayesian network, therefore BR takes advantage of the Bayesian network for the choice of the (instance, feature) pairs.

Algorithm 5.3 presents our adaption of this algorithm to use Bayesian networks.

5.4.4 Single Feature Look-ahead (a.k.a. SFL or sfl)

While the budget is not yet used up, for each purchase, SFL chooses an instance and a feature as follows: it first finds the (instance, feature) pair X_{ij} whose SFL value is the minimum, then chooses this feature; and the instance is chosen uniformly at random from the unpurchased instances of the chosen feature.

SFL value is computed as follows:

$$\text{SFL-value}(X_{i,j}) = \sum_{\text{val} \in \text{Values}(X_{i,j})} P(X_{i,j} = \text{val}) \text{LogLoss}(w') \ , \tag{5.2}$$

where LogLoss is defined in Equation 5.1, w' is vector of the posterior parameters of the current Bayesian network after assigning the (instance, feature) pair (i, j) $X_{i,j} = \text{val}$. The computation of SFL-value is based on the distribution of the learned Bayesian network,

Algorithm 5.3 BR(b, c) where b is the given budget and c is the unit cost of purchasing an (instance, feature) pair

```
1: Assign V_chosen as a random feature of all the features {V_1, ..., V_m};
2: Compute current-log-loss using Equation 5.1;
3: previous-log-loss = current-log-loss;
4: while b > 0 do
5:    Compute current-log-loss using Equation 5.1;
6:    if current-log-loss ≥ previous-log-loss then
7:       chosen = (chosen + 1)  mod m;
8:       // Guarantee that the chosen feature/node has unpurchased instances;
9:       if there is no unpurchased (instance, feature) then
10:         return;
11:         while there is no unpurchased instances of the feature V_chosen do
12:            chosen = (chosen + 1)  mod m;
13:         end while
14:      end if
15:   end if
16:   x_chosen is a u.a.r. instance from the unpurchased instances of feature V_chosen;
17:   Purchase (x_chosen, V_chosen);
18:   b = b − c;
19:   Update the probability tables of the Bayesian Network;
20:   previous-log-loss = current-log-loss;
21: end while
```

therefore SFL takes advantage of the Bayesian network for the choice of the (instance, feature) pairs.

Algorithm 5.4 presents our adaption of this algorithm to use Bayesian networks.

5.4.5 Randomized Single Feature Look-ahead (a.k.a. RSFL or rsfl)

While the budget is not yet used up, for each purchase, RSFL chooses an instance and a feature as follows: it first computes the (instance, feature) pairs with the K smallest SFL values, and then picks a (instance, feature) from the pairs according to a Boltzmann distribution (Kapoor and Greiner, 2005b), defined below. Then we choose the feature in

Algorithm 5.4 SFL(b, c) where b is the given budget and c is the unit cost of purchasing an (instance, feature) pair

```
1: while b > 0 do
2:    largest-sfl-value = the smallest number a computer can store;
3:    chosen-instance-feature = (−1, −1);
4:    for unpurchased (feature, instance) pair X_{i,j} do
5:       Compute SFL-value(X_{i,j}) using Equation 5.2;
6:       if SFL-value(X_{i,j}) > largest-sfl-value then
7:          chosen-instance-feature = (i, j);
8:          largest-sfl-value = SFL-value(X_{i,j});
9:       end if
10:   end for
11:   V_chosen = the feature in the chosen-instance-feature ;
12:   x_chosen is chosen u.a.r. from instances whose feature values of V_chosen have not yet
      been purchased;
13:   Purchase (x_chosen, V_chosen);
14:   b = b − c;
15:   Update the probability tables of the Bayesian Network;
16: end while
```

the selected (instance, feature) and the instance is chosen uniformly at random from the unpurchased instances of the chosen feature. The computation of SFL-value is based on the distribution of the learned Bayesian network; therefore, RSFL takes advantage of the Bayesian network for the choice of the (instance, feature) pairs.

To choose a (instance, feature) pair $X_{i,j}$ from all unpurchased (instance, feature) pairs according to a Boltzmann distribution, Kapoor and Greiner use a Boltzmann distribution:

$$\exp\left(\frac{-\text{SFL-value}(X_{i,j})}{\tau}\right) \Big/ \sum_{X_{i,j} \in C} \exp\left(\frac{-\text{SFL-value}(X_{i,j})}{\tau}\right) \ , \tag{5.3}$$

where C is the set of unpurchased (instance, feature) pairs, τ is a parameter that we set to 1 in our experiment, and SFL-value is defined in Equation 5.2.

Algorithm 5.5 presents our adaption of this algorithm to use Bayesian networks. In our experiments, we set K to 10.

Algorithm 5.5 RSFL(b, c, K) where b is the given budget c is the unit cost of purchasing an (instance, feature) pair, and K is an integer parameter

while $b > 0$ **do**
  largest-sfl-values = a set of K smallest numbers a computer can store;
  chosen-instance-features = $\emptyset$;
  for unpurchased (feature, instance) pair $X_{i,j}$ **do**
    Compute SFL-value($X_{i,j}$) using Equation 5.2;
    if SFL-value($X_{i,j}$) > the smallest value in largest-sfl-value **then**
      put SFL-value($X_{i,j}$) into largest-sfl-values;
      put $X_{i,j}$ into chosen-instance-features;
      remove the (instance, feature) pair from largest-sfl-values whose SFL-value is the smallest from the chosen-instance-features;
      remove the smallest SFL-value from largest-sfl-values;
    end if
  end for
  (x, V_{chosen}) is chosen according to Equation 5.3 from chosen-instance-features;
  x_{chosen} is chosen u.a.r. from the instances whose feature value V_{chosen} has not yet been purchased;
  Purchase $(x_{\text{chosen}}, V_{\text{chosen}})$;
  $b = b - c$;
  Update the probability tables of the Bayesian Network;
end while

5.4.6 Generalized Single Feature Look-ahead (a.k.a. GSFL or gsfl)

While the budget is not yet used up, for each purchase, GSFL chooses the (instance, feature) pair with the minimum SFL value. The computation of SFL-value is based on the distribution of the learned Bayesian network, therefore GSFL takes advantage of the Bayesian network for the choice of the (instance, feature) pairs.

Algorithm 5.6 presents our adaption of this algorithm to use Bayesian networks. Note that the only difference of GSFL and SFL is that at each purchase, after we choose a (instance, feature) with the minimum SFL value, GSFL purchases this (instance, feature) while SFL purchases a instance which is chosen uniformly at random from unpurchased instances for this feature. SFL assumes the equivalence of the the chosen feature's unpurchased in-

stances while GSFL does not.

Algorithm 5.6 GSFL(b, c) where b is the given budget and c is the unit cost of purchasing an (instance, feature) pair

```
1: while b > 0 do
2:    largest-sfl-value = the smallest number that a computer can store;
3:    chosen-instance-feature = (−1, −1);
4:    for unpurchased (feature, instance) pair X_{i,j} do
5:       Compute SFL-value(X_{i,j}) using Equation 5.2;
6:       if SFL-value(X_{i,j}) > largest-sfl-value then
7:          chosen-instance-feature = (i, j);
8:          largest-sfl-value = SFL-value(X_{i,j});
9:       end if
10:   end for
11:   Purchase chosen-instance-feature;
12:   b = b − c;
13:   Update the probability tables of the Bayesian Network;
14: end while
```

5.4.7 Generalized Random Single Feature Look-ahead (a.k.a. GRSFL or grsfl)

While the budget is not yet used up, each purchase GRSFL chooses the (instance, feature) pair according to a Boltzmann distribution from the (instance, feature) pairs with the K smallest SFL values. The computation of SFL-value is based on the distribution of the learned Bayesian network, therefore GRSFL takes advantage of the Bayesian network for the choice of the (instance, feature) pairs.

Algorithm 5.7 presents our adaption of this algorithm to use Bayesian networks. In our experiments, we set K to 10. Note that the only difference of GRSFL and RSFL is that at each purchase, after we choose a (instance, feature) at random according to a Boltzmann distribution from the (instance, feature) pairs with smallest K SFL values, GRSFL purchases this (instance, feature) while RSFL purchases a random unpurchased instance for this fea-

ture. I.e., RSFL assumes the equivalence of the chosen feature's unpurchased instances while GRSFL does not.

Algorithm 5.7 GRSFL(b, c, K) where b is the given budget c is the unit cost of purchasing an (instance, feature) pair, and K is a integer parameter

```
1: while b > 0 do
2:    largest-sfl-values = a set of K smallest numbers that a computer can store;
3:    chosen-instance-features = ∅;
4:    for unpurchased (feature, instance) pair X_{i,j} do
5:       Compute SFL-value(X_{i,j}) using Equation 5.2;
6:       if SFL-value(X_{i,j}) > the smallest value in largest-sfl-value then
7:          put SFL-value(X_{i,j}) into largest-sfl-values;
8:          put X_{i,j} into chosen-instance-features;
9:          remove the (instance, feature) pair from largest-sfl-values whose SFL-value is the
            smallest from the chosen-instance-features;
10:         remove a smallest SFL-value from the largest-sfl-values;
11:      end if
12:   end for
13:   (x_chosen, V_chosen) is a randomly chosen (instance, feature) according to Equation 5.3
      from the chosen-instance-features;
14:   Purchase (x_chosen, V_chosen);
15:   b = b − c;
16:   Update the probability tables of the Bayesian Network;
17: end while
```

5.5 Our Algorithms

In this section, we present our algorithms which take advantage of Bayesian network in different ways. Our first algorithm, Maximization of Expected Relative Probability Gain (MERPG), chooses at each purchase the (instance, feature) pair that maximizes relative increase of the expected accuracy of the instance label's true label. MERPGDSEP, at each purchase, chooses the (instance, feature) by using the same method as MERPG except that it breaks ties by choosing the (instance, feature) pair whose instantiation maximizes the number of features d-separated from the label node for the instance. MERPGDSEPW1

and MERPGDSEPW2, at each purchase, combine a weighting factor with the relative increase of expected accuracy of the instance's true label. Sections 5.5.1, 5.5.2, and 5.5.3 present the MERPG related algorithms, including MERPG (merpg), MERPGDSEP (dsep), MERPGDSEPW1 (dsepw1), and MERPGDSEPW2 (dsepw2). Section 5.5.4 describes how we use the Markov blanket of the label node as a filter for each algorithm.

5.5.1 Maximization of Expected Relative Probability Gain (a.k.a. MERPG or merpg)

Briefly speaking, MERPG chooses an (instance, feature) pair (i, j) that maximizes the relative expected improved accuracy. When there are multiple $X_{i,j}$s with the same maximum relative improved expected accuracy, a (instance, feature) pair is chosen uniformly at random from them. The intuition behind this idea is that the (instance, feature) pair with the maximum relative improved accuracy of the instance is most meaningful to the instance. For example, when the expected accuracy for one instance increases from 0.9 to 0.95, compared to another expected accuracy for another instance increases from 0.7 to 0.75, although both have same improved expected accuracy of 0.05, the latter has a bigger relative improved expected accuracy and is more meaningful.

The formula to compute the relative improved expected accuracy of purchasing $X_{i,j}$ is as follows:

$$\mathrm{ERPG}(X_{i,j}) = \frac{\left(\sum_{q \in \mathrm{vals}(V_j)} P(X_{i,j} = q) P(y_i \mid x_i, X_{i,j} = q)\right) - P(y_i \mid x_i)}{P(y_i \mid x_i)} \ , \tag{5.4}$$

which we call the ERPG value for $X_{i,j}$. In Equation 5.4, $\mathrm{vals}(V_j)$ are the possible values of feature j, y_i represents the label of instance x_i, $P(y_i \mid x_i)$ represents the probability of predicting x_i as its true label y_i, and $P(y_i \mid x_i, X_{i,j} = q)$ represents the probability of predicting x_i as its true label y_i after $X_{i,j}$ is instantiated to value q.

MERPG then chooses the (instance, feature) pair $X_{i,j}$ that maximizes the relative improved expected accuracy:

$$\underset{X_{i,j} \in C}{\operatorname{argmax}} \operatorname{ERPG}(X_{i,j}) \ , \tag{5.5}$$

where C is the set of unspecified (unpurchased) (instance, feature) pairs. $X_{i,j}$ represents (instance, feature) pair (i, j). When there are multiple unpurchased (instance, feature) pairs that have the maximum MERPG values, one pair is chosen uniformly at random from them. Algorithm 5.8 presents the algorithm MERPG.

Algorithm 5.8 MERPG(b, c) where b is the given budget and c is the cost of purchasing an (instance, feature) pair

```
1: while b > 0 do
2:    Choose X_{i,j} which is computed using Equation 5.5;
3:    Purchase (instance, node) X_{i,j};
4:    b = b − c;
5:    Update the probability tables of the Bayesian Network;
6: end while
```

5.5.2 MERPGDSEP (a.k.a. dsep)

Algorithm 5.9 shows the algorithm MERPGDSEP which breaks ties of MERPG by choosing the (instance, feature) pair whose instantiation leads to the maximum increase of d-separations of the nodes from the label node.

Recall that the motivation of this method is that instantiating those nodes that are d-separated from the label node will not affect the label node. Therefore, the greater the increase of d-separation of nodes from the label node when instantiating a feature for one instance, the bigger the influence of this instantiation. Therefore, we propose MERPGDSEP which uses the increase of d-separation as a tie breaker for MERPG. MERPGDSEP uses the following function NumDsep, which computes the number of nodes that are d-separated

from the label node for a given instance x_i and Bayesian network N:

$$\text{NumDsep}(x_i, N) =$$

$$\left| \{j \neq \ell : V_j \text{ is d-separated from the label node in } N \ V_\ell \text{ and not purchased for instance } x_i\} \right| \ .$$

The method to judge whether a node is d-separated from the label node or not is described in Section 2.2.2.4.

Figure 5.1 shows a classical Bayesian network Chest Clinic from Norsys Corporation Net Library (2011). The label node is "Tuberculosis or Cancer". Let us call the network as N. For one patient, when all feature values are unknown, which nodes are d-separated from the label node? The answer is none. For "Bronchitis", although in the path of "Tuberculosis or Cancer" $\rightarrow$ "Dyspnea" $\leftarrow$ "Bronchitis", "Bronchitis" is blocked by "Dyspnea" (refer to the third example in Figure 2.5), it is connected to the label node via "Tuberculosis or Cancer" $\leftarrow$ "Lung Cancer" $\leftarrow$ "Smoking" $\rightarrow$ "Bronchitis" (refer to the first and second example in Figure 2.5), therefore 'Bronchitis" is connected to the label node. Therefore, let x_i be the patient, NumDsep(x_i with no feature values purchased, N) = 0.

If we instantiate node "Lung Cancer", which nodes will be d-separated from the label node? Feature nodes "Smoking" and "Bronchitis" will be d-separted from the label node. Feature nodes "Smoking" and "Bronchitis" are d-separated from the label node because the instantiation of "Lung Cancer" blocks the connection to these two nodes to the label node via "Lung Cancer", and in the path of "Tuberculosis or Cancer" $\rightarrow$ "Dyspnea" $\leftarrow$ "Bronchitis" $\leftarrow$ "Smoking", these two nodes are blocked by "Dyspnea". Therefore, after we instantiate node "Lung Cancer", NumDsep(x_i in which"Lung Cancer" is instantiated, N) = 2.

We define the increase of d-separations of the nodes from the label node in Bayesian

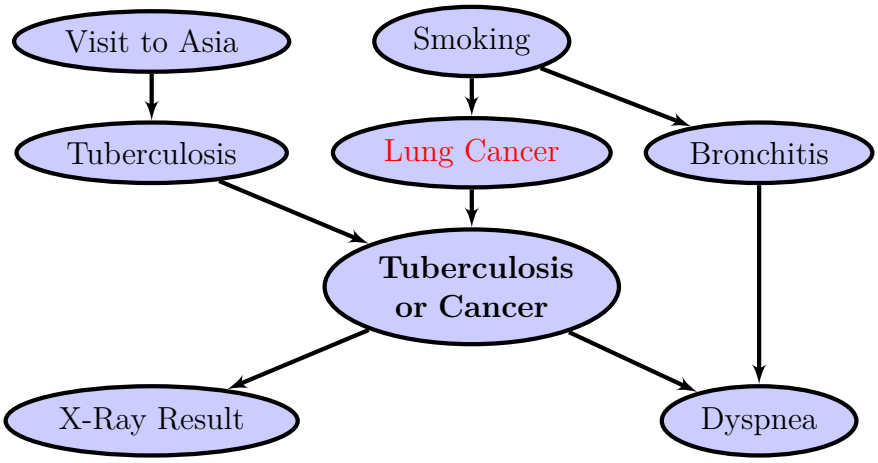


Figure 5.1: A classical Bayesian network ChestClinic from Norsys Corporation Net Library (2011). The label node is "Tuberculosis or Cancer". For one patient, when all feature values are unknown, no nodes are d-separated with the label node. If we instantiate node "Lung Cancer", feature nodes "Smoking" and "Bronchitis" are d-separted with the label node.

network N after instantiating an (instance, feature) pair $X_{i,j}$ as follows:

$$\text{NumIncreaseDseps}(X_{i,j}, N) = \text{NumDsep}(x_i \text{ with } X_{i,j} \text{ instantiated}, N) - \text{NumDsep}(x_i, N) \ . \tag{5.6}$$

For MERPGDSEP, for each purchase, when there are multiple (instance, feature) pairs with the maximum MERPG values, MERPGDSEP chooses one (instance, feature) pair as follows:

$$\underset{X_{i,j} \in C}{\text{argmax}} \Big(\text{NumIncreaseDseps}(X_{i,j}, N)\Big) \tag{5.7}$$

where C is the set of (instance, feature) pairs with the maximum ERPG values.

Algorithm 5.9 MERPGDSEP(b, c) where b is the given budget and c is the unit cost of purchasing an (instance, feature) pair

1: **while** $b > 0$ **do**
2: Compute C' as the set of all the (instance, feature) pairs whose MERPG values are maximum;
3: Choose $X_{i,j}$ via Equation 5.7;
4: Purchase (instance, node) $X_{i,j}$;
5: $b = b - c$;
6: Update the probability tables of the Bayesian Network;
7: **end while**

5.5.3 MERPGDSEPW1 (a.k.a. dsepw1) and MERPGDSEPW2 (a.k.a. dsepw2)

Recall that in Section 5.5.2, we use NumIncreaseDseps as a tie breaker for MERPG. The motivation is that the bigger NumIncreaseDseps is after instantiating an (instance, feature) pair, the bigger the influence of instantiating this (instance, feature) pair. Because of this same motivation, we propose MERPGDSEPW1 and MERPGDSEPW2 which use a function related to the NumIncreaseDseps value as a weighting factor combined with MERPG.

We want the bigger increase of number of d-separations has a larger weighting factor, therefore we proposed algorithms MERPGDSEPW1 (Algorithm 5.10) and MERPGDSEPW2 (Algorithm 5.11), which using combineds with MERPG with a weighting factor in a linear and logarithmetic way. In Algorithm 5.10, we compute FACT$[i, j]$ as follows: when NumIncreaseDseps is positive for an instance x_i when considering $X_{i,j}$, FACT$[i, j]$ = 1 + NumIncreasedDseps, which is bigger than 1; when NumIncreaseDseps is negative for an instance x_i when purchasing $X_{i,j}$, FACT$[i, j] = 1/(1 - \text{NumIncreasedDseps})$, which is smaller than 1. MERPGDSEPW1 chooses an (instance, feature) pair $X_{i,j}$ as follows:

$$\underset{X_{i,j} \in C}{\text{argmax}} \left(\text{FACT}[i,j] * \frac{\left(\sum_{q \in \text{vals}(X_{i,j})} P(X_{i,j} = q) P(y_i \mid x_i, X_{i,j} = q) \right) - P(y_i \mid x_i)}{P(y_i \mid x_i)} \right) \quad (5.8)$$

In Algorithm 5.11, we compute LOGFACT$[i, j]$ as follows: when NumIncreaseDseps is positive for an instance x_i when considering $X_{i,j}$, LOGFACT$[i, j] = \ln(e+\text{NumIncreasedDseps})$ which is bigger than 1; when the NumIncreaseDseps is negative for an instance x_i when considering $X_{i,j}$, LOGFACT$[i, j] = 1/\ln(e - \text{NumIncreasedDseps})$, which is smaller than 1. MERPGDSEPW2 chooses an (instance, feature) pair $X_{i,j}$ as follows:

$$\underset{X_{i,j} \in C}{\operatorname{argmax}} \left(\text{LOGFACT}[i, j] * \frac{\left(\sum_{q \in \text{vals}(X_{i,j})} P(X_{i,j} = q) P(y_i \mid x_i, X_{i,j} = q) \right) - P(y_i \mid x_i)}{P(y_i \mid x_i)} \right) \tag{5.9}$$

Algorithm 5.10 MERPGDSEPW1(b, c) where b is the given budget and c is the unit cost of purchasing an (instance, feature) pair

```
1: while b > 0 do
2:    // Compute the weighting factor of every unpurchased instance X_{i,j}
3:    for each unpurchased X_{i,j} do
4:       Compute NumIncreaseDseps(X_{i,j}) via Equation 5.6;
5:       if NumIncreasedDseps(X_{i,j}) ≥ 0 then
6:          FACT[i, j] = 1 + NumIncreaseDseps(X_{i,j});
7:       else
8:          FACT[i, j] = 1 / (1 − NumIncreaseDseps(X_{i,j}));
9:       end if
10:   end for
11:   Compute X_{i,j} via Equation 5.8;
12:   Purchase (instance, node) X_{i,j};
13:   b = b − c;
14:   Update the probability tables of the Bayesian Network;
15: end while
```

5.5.4 Filtering with the Markov blanket

The Markov blanket of a node in a Bayesian network is explained in Section 2.2.2.5. In Bayesian networks, a node's parent, children, and spouses (the children's other parent nodes) form a Markov blanket for this node. The Markov blanket of a node is sufficient to predict the behavior of that node.

Algorithm 5.11 MERPGDSEPW2(b, c) where b is the given budget and c is the unit cost of purchasing an (instance, feature) pair

```
1: while b > 0 do
2:    // Compute the log weighting factor of every unpurchased instance X_{i,j}
3:    for each unpurchased X_{i,j} do
4:       Compute NumIncreaseDseps(X_{i,j}) via Equation 5.6;
5:       if NumIncreasedDseps(X_{i,j}) ≥ 0 then
6:          LOGFACT[i, j] = ln(e + NumIncreaseDseps(X_{i,j}));
7:       else
8:          LOGFACT[i, j] = 1/ ln(e − NumIncreaseDseps(X_{i,j}));
9:       end if
10:   end for
11:   Compute X_{i,j} via Equation 5.9;
12:   Purchase (instance, node) X_{i,j};
13:   b = b − c;
14:   Update the probability tables of the Bayesian Network;
15: end while
```

Will it be better to choose (instance, feature) pairs from those unpurchased pairs whose features fall in the Markov blanket instead of from all unpurchased pairs? To answer this question, we ran all our algorithms on a Markov blanket of the label node, and compared our algorithms running on this filter with our original algorithms that select purchases from all features. We want to see whether this filter saves running times of the algorithms and improves the algorithm to reach a lower classification error.

Note that when we combine the Markov blanket with an algorithm "algo", we refer to this algorithm as "MBalgo" in our experiments. Therefore we have MBrandom, MBrr, MBbr, MBsfl, MBrsfl, MBgsfl, MBgrsfl, MBMERPG, MBdsep, MBdsepw1, and MBdsepw2.

5.6 Experimental Results

In this section, we show our experimental results on 5 Bayesian networks, Chest Clinic, Animals, Car Diagnosis 2, ALARM and Poya Ganga, from Norsys Net Library (`http://www.norsys.com/netlibrary/index.htm`). The Bayesian networks of these 5 data sets

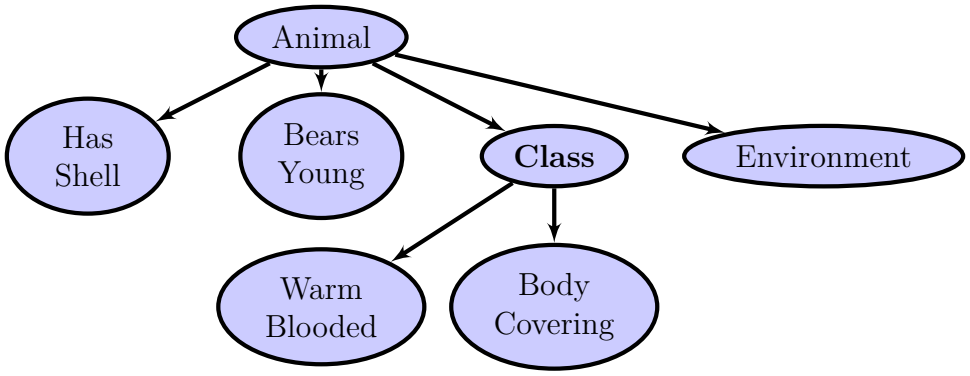


Figure 5.2: Network Animals from Norsys Corporation Net Library (2011). "Animal" is the label node. The values for the nodes are as follows. "Animal" has 5 states: *monkey*, *penguim*, *platypus*, *robin*, and *turtle*. "Has Shell" has 2 states: *true* and *false*. "Bears Young" has 2 states: *live* and *eggs*. "Class" has 3 states: *bird*, *mammal*, and *reptile*. "Environment" has 3 states: *air*, *land*, and *water*. "Warm Blooded" has 2 states: *true* or *false*. "Body Covering" has 3 states: *fur*, *feathers*, and *scales*.

are available from this website. Figure 5.2 shows the network of Animals. Figure 5.3 presents the network of Car Diagnosis 2. The Bayesian network of Chest Clinic is shown in Figure 5.1. Figure 5.4 presents a partial network of Poya Ganga (a complete network is shown in Figure A.2). Figure 5.5 shows a partial network of ALARM (a complete network is shown in Figure A.1).

Using the Netica package (Norsys Software Corp., 2011), we generated data based on these five Bayesian networks. In our experiments, we used the correct structure of each Bayesian network and initialized all counts of all conditional probability distribution tables to be 1.

To evaluate the overall behavior of each of the algorithms, we used 10-fold cross validation and constructed learning curves that reflect the performance of a heuristic by its mean error over the 10 folds as more attributes are purchased. In each fold, the algorithms have 90% of all the data as the training data and the remaining 10% of the remaining data as the test data. We assume each (instance, feature) is available at a cost of 1. For each learning

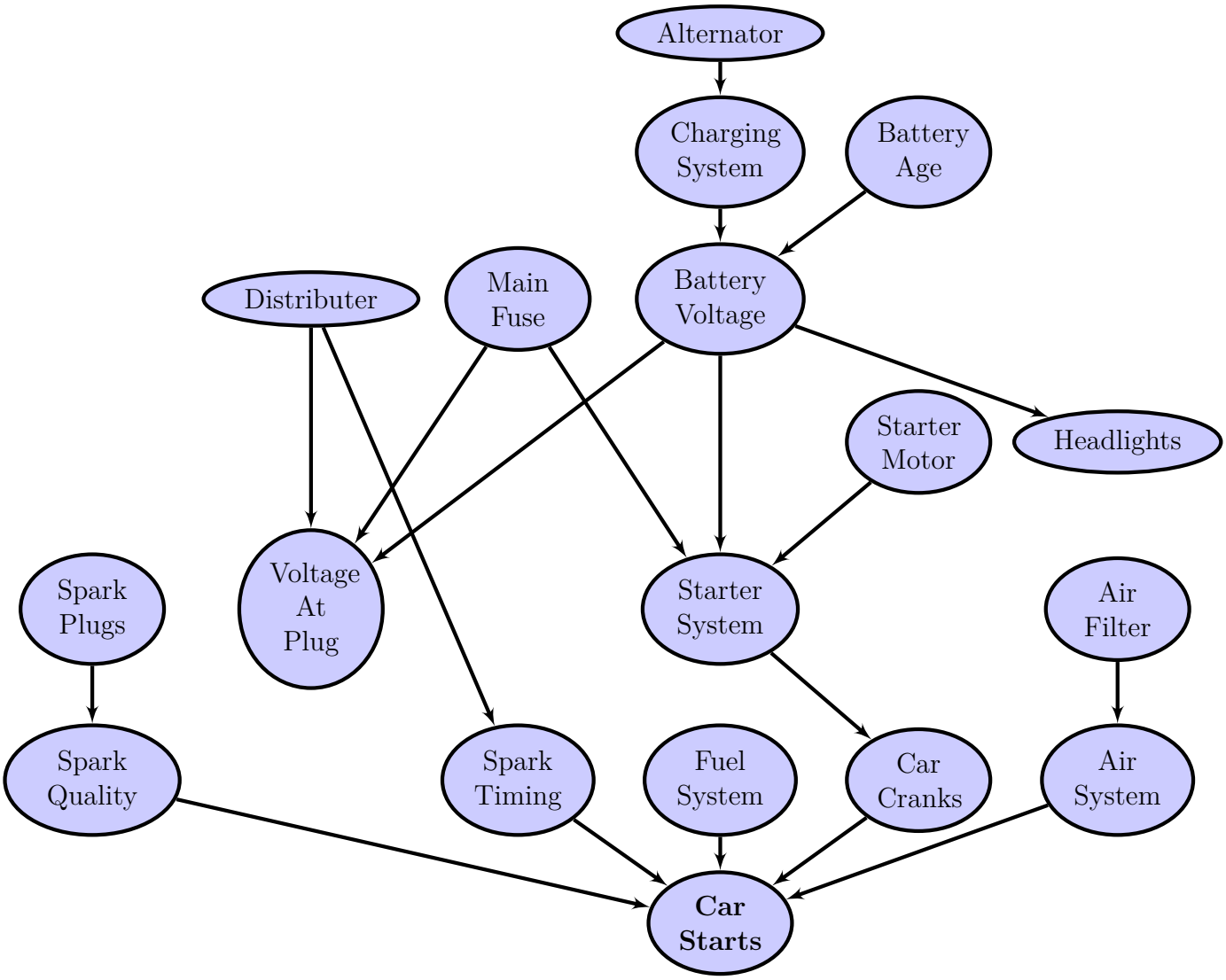


Figure 5.3: Network Car Diagnosis 2 from Norsys Corporation Net Library (2011). "Car Starts" is the label node. The values for the nodes are as follows. "Alternator" has 2 states: *okay* and *faulty*. "Charing System", has 2 states: *okay* and *faulty*. "Distributer" has 2 states: *okay* and *faulty*. "Main Fuse" has 2 states: *okay* and *blown*. "Battery Age" has 3 states: *new*, *old*, and *very old*. "Battery Voltage" has 3 states: *strong*, *weak*, and *dead*. "Voltage at Plug" has 3 states: *strong*, *weak*, and *none*. "Starter Motor" has 2 states: *okay* and *faulty*. "Spark Plugs" has 3 states: *okay*, *too wide*, and *fouled*. "Spark Timing" has 3 states: *good*, *bad*, and *very bad*. "Starter System" has 2 states: *okay*, and *faulty*. "Headlights" has 3 states: *bright*, *dim*, and *off*. "Spark Quality" has 3 states: *good*, *bad*, and *very bad*. "Fuel System" has 2 states: *okay* and *faulty*. "Car Cranks" has 2 states: *true* and *false*. "Air Filter" has 2 states: *clean* and *dirty*. "Air System" has 2 states: *okay* and *faulty*. "Car Starts" has 2 states: *true* and *false*.

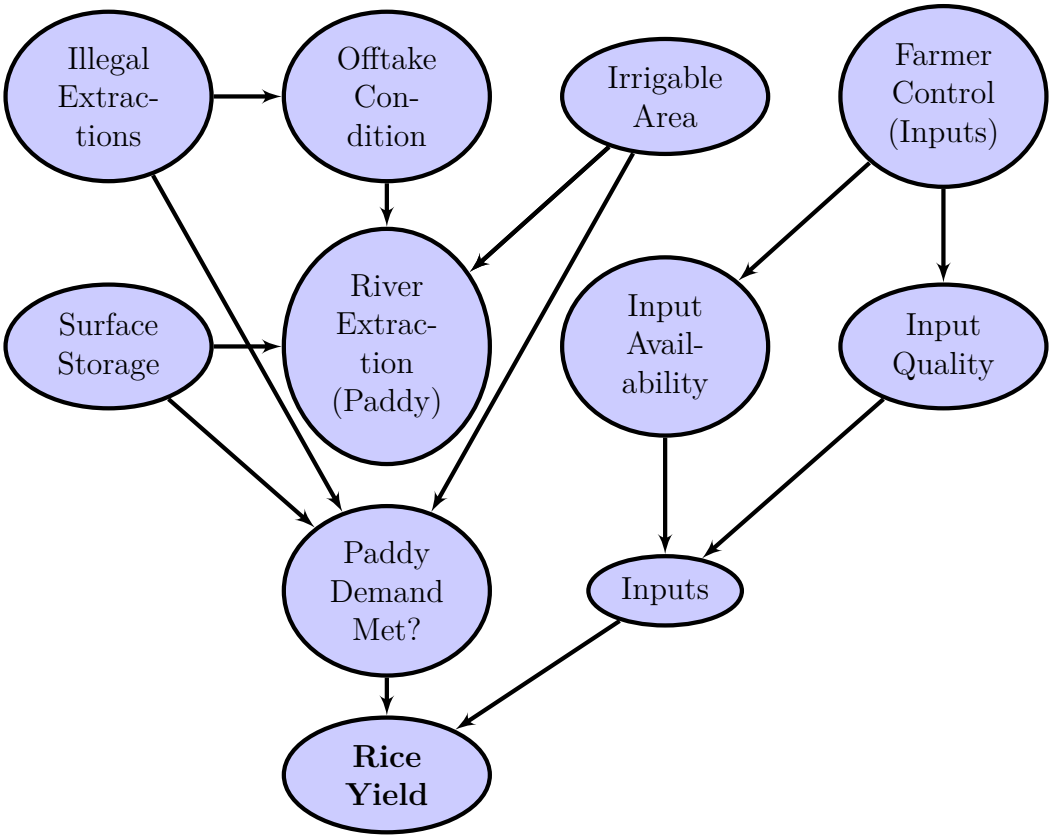


Figure 5.4: Partial Network of Poya Ganga (a.k.a Water Resource Management). A complete work is shown in Figure A.2. "Rice Yield" is the class label. The values for the nodes are as follows. "Illegal Extractions" has 2 states: *yes* and *no*. "Offtake Condition" has 2 states: *good* and *poor*. "Irrigable Area" has 2 states: *current* and *down 25pc*. "Farmer Control (Inputs)" has 2 states: *yes* and *no*. "Surface Storage" has 2 states: *high* and *low*. "River Extraction (Paddy)" has 3 states: *up 10pc*, *current*, and *down 20pc*. "Input Availability" has 2 states: *when needed* and *not*. "Input Quality" has 2 states: *good* and *poor*. "Paddy Demand Met?" has 2 states: *yes* and *no*. "Rice Yield" has 2 states: *high* and *low*.

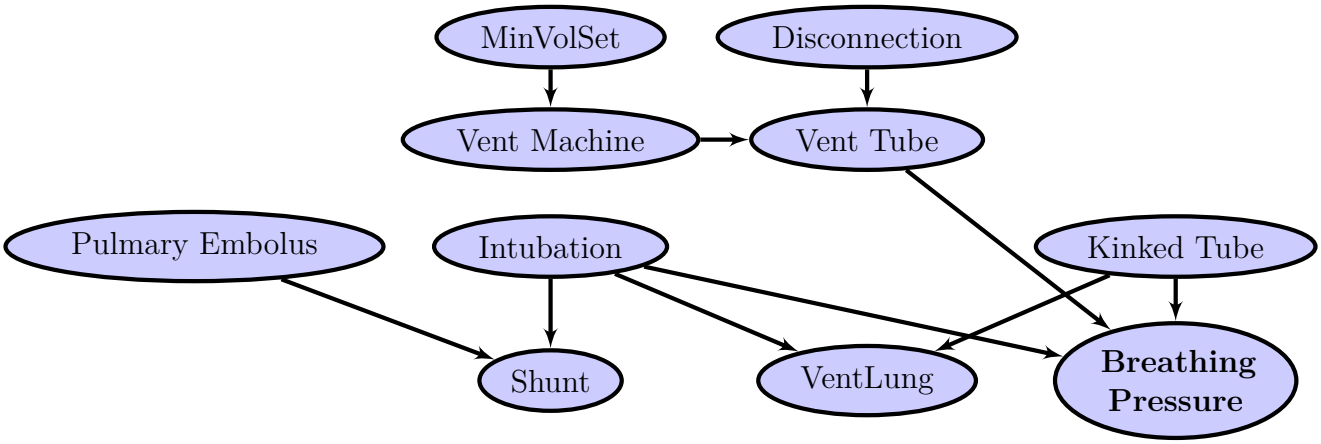


Figure 5.5: Partial Network of ALARM from Norsys Corporation Net Library (2011). A complete network is shown in Figure A.1. "Breathing Pressure" is the label node. The values for the nodes are as follows. "MinVolSet" has 3 states: *low*, *normal*, and *high*. "Disconnection" has 2 states: *true* and *false*. "Vent Machine' has 4 states: *zero*, *low*, *normal*, and *high*. "Vent Tube" has 4 states: *zero*, *low*, *normal*, and *high*. "Pulmary Embolus" has 3 states: *true* and *false*. "Intubation" has 3 states: *normal*, *esophageal*, and *one sided*. "Kinked Tube" has 2 states: *true* and *false*. "Shunt" has 2 states: *normal* and *high*. "Vent Lung" has 4 states: *zero*, *low*, *normal*, and *high*. "Breathing Pressure" has 4 states: *zero*, *low*, *normal*, and *high*.

problem, a budget is set as the cost needed for the best algorithm to approximately reach the optimum (the classification error of the learned Bayesian network when all the features and labels of the data set are given). Therefore, for Animals, Car Diagnosis 2, and ALARM, the budget is set to 100. For Poya Ganga, the budget is set to 30. For Chest Clinic, the budget is set to 40. The smaller budgets of Poya Ganga and Chest Clinic is because these two data sets quickly approach their baseline at budget 30 and 40.

Table 5.1 describes these 5 networks, including the number of nodes, the number of edges, and the number of instances of each data set. For smaller networks Animals and Car Diagnosis 2, and Chest Clinic we generated 5000 instances; while for larger networks ALARM and Poya Ganga, we generated 1000 instances because of our consideration for running efficiency.

Table 5.1: Networks used in our BLBN experiments.

DataSet	# Nodes of BN	# Edges of BN	# instances generated
Animals	7	6	5000
Car Diagnosis 2	18	20	5000
Chest Clinic	8	8	5000
Poya Ganga	60	65	1000
ALARM	37	46	1000

5.6.1 Experimental Results on Animals

Our first learning problem is Animals from Norsys Net Library (2011). It is a belief network with 7 nodes and 6 edges, and we generated 5000 instances based on the provided conditional probability tables. We ran rr, br, merpg (MERPG), dsep (MERPGDSEP), dsepw1 (MERPGDSEPW1), dsepw2 (MERPGDSEPW2), sfl, gsfl, rsfl, grsfl on both naïve Bayes (NB) and Bayesian network (BN) and BN with Markov blanket filter on Animals. Figure 5.6 shows the mean classification errors of the algorithms learning for data set Animals. Classifiers are Bayesian network on Markov blanket, Bayesian network, and naïve Bayes. Figures 5.7 and 5.8 show the mean classification errors of the same algorithm of Bayesian network, of Baysian network on a Markov blanket filter, and of naïve Bayes for data set Animals. Table 5.2 shows the result of a Wilcoxon signed rank test between every pair of algorithms after 100 purchases for data set Animals at a confidence level $p < 0.05$. We say one algorithm *algo1* wins (outperforms) another algorithm *algo2* if *algo1* reaches a significantly higher classification accuracy (i.e. a significantly lower classification error) than *algo2* at a confidence level $p < 0.05$. **MBdsepw1 (BN) and MBdsepw2 (BN)** have the maximum number of wins over other algorithms and have no losses other algorithms, therefore we conclude that MBdsepw1 and MBdsepw2 outperform other algorithms after 100 purchases for Animals. **Random (BN)** has no wins over other algorithms and has the maximum number of losses to other algorithms, therefore we conclude that Random (BN) is outperformed by other algorithms after 100 purchases for Animals.

From Table 5.2, we see that an algorithm using Bayesian network does not outperform the same algorithm using naïve Bayes. On the contrary, random on naïve Bayes even outperforms random on Bayesian network. This is because for Animals, the naïve Bayes is as accurate as Bayesian network because the baselines of naïve Bayes and of Bayesian network are the same, each having a classification error 0.1 (see the 5th subfigure in Figure 5.8). This means that there is no performance improvement to improved purchase decisions due to using the Bayesian network as a probabilistic model over naïve Bayes for Animals.

From the structure of Animals in Figure 5.2, the label node is "class" and it has 1 parent "Animal" and 2 children "Warm Blooded" and "Body Covering". Therefore, its Markov blanket includes 3 nodes: "Animal", "Warm Blooded", and "Body Covering". The reason that MBdsepw1 (BN) and MBdsepw2 (BN) perform the best is because node "Animal" is an important feature while these two algorithms choose instances of this feature to purchase more frequently because the large weighting factor compared to weighting factors when instantiating the other two nodes. (Instantiation of "Animal" makes NumIncreaseDseps equal to 3, which is far greater than the instantiation of "Warm Blooded" or "Body Covering" making NumIncreaseDseps equal to 0.)

Note that merpg (NB), dsep (NB), dsepw1 (NB), dsepw2 (NB), merpg (BN), dsep (BN), dsepw1 (BN), dsepw2 (BN), MBmerpg (BN), and MBdsep (BN) also perform well for Animals, each with at least 6 wins. random (NB), rr (NB), br (NB), sfl (NB), random (BN), rr (BN), br (BN), sfl (BN) MBrandom (BN), MBrr (BN), MBbr (BN), and MB (sfl) are big losers, each with at most 1 win and at least 4 losses.

For the mean running time of each algorithm on Animals, as shown in Table 5.3, we can see that SFL takes much longer time than MERPG series (MERPG, MERPGDSEP, MERPGDSEPW1, MERPGDSEPW2), which also takes longer time than Random, RR, and BR. MB*algo* takes significantly less time (up to 70%) than *algo*.

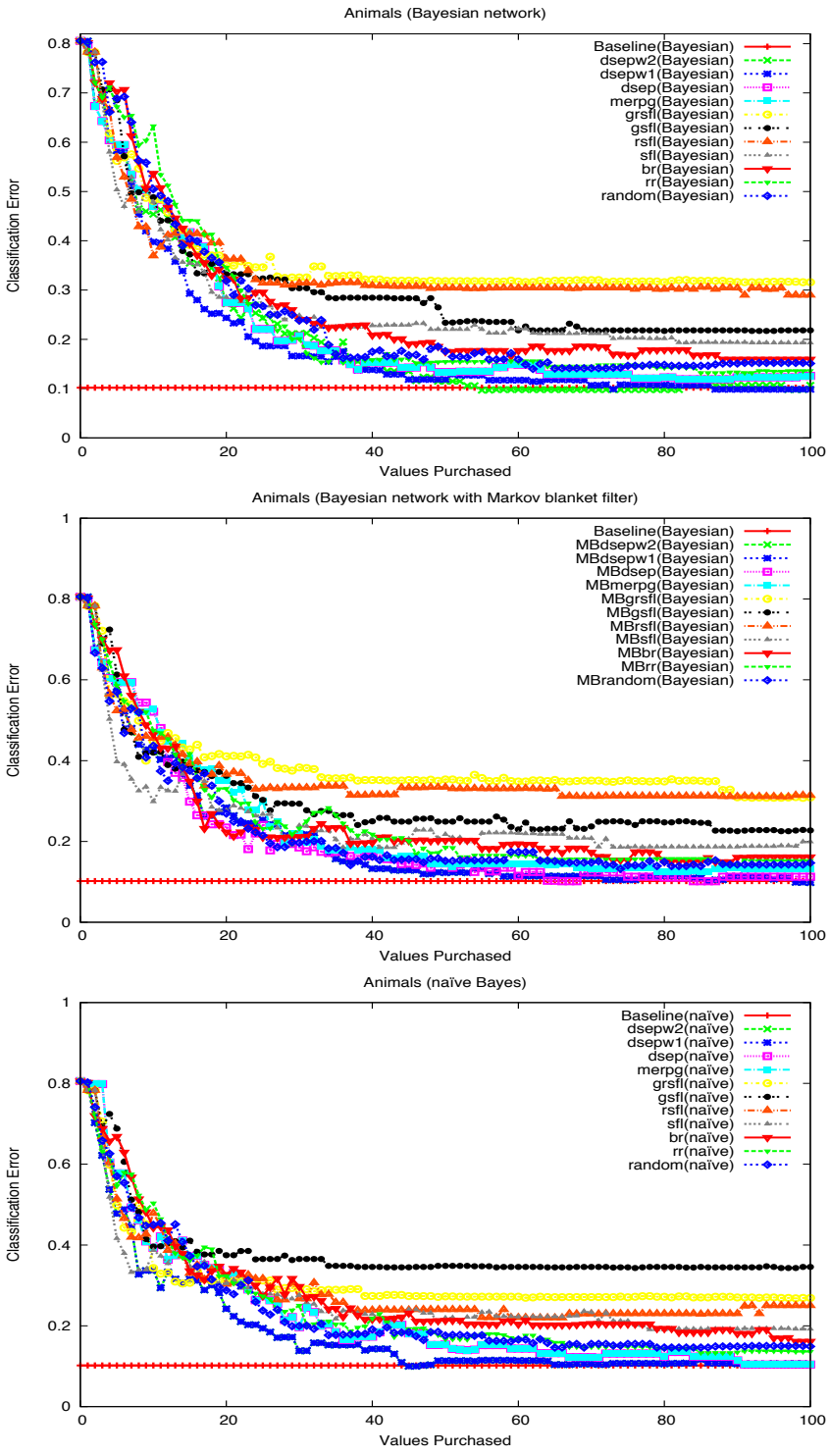


Figure 5.6: Comparing all the algorithms on Bayesian network, on Bayesian network with Markov blanket filter, and on naïve Bayes on Animals from Norsys Net Library (2011).

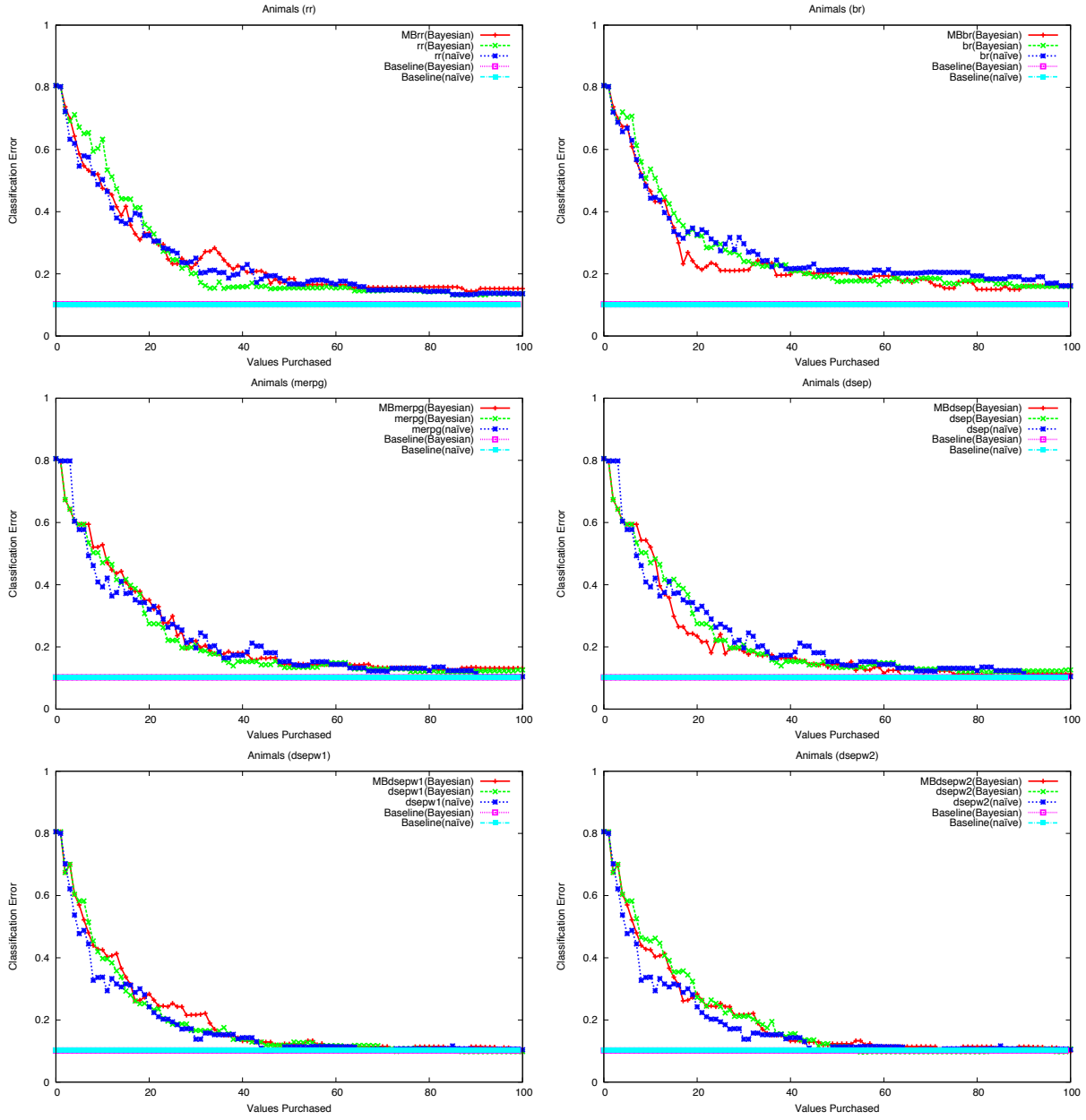


Figure 5.7: Comparing each algorithm (rr, br, MERPG, dsep, dsepw1, dsepw) on Bayesian network with Markov blanket filter, on Bayesian network, and on naïve Bayes on Animals from Norsys Net Library (2011).

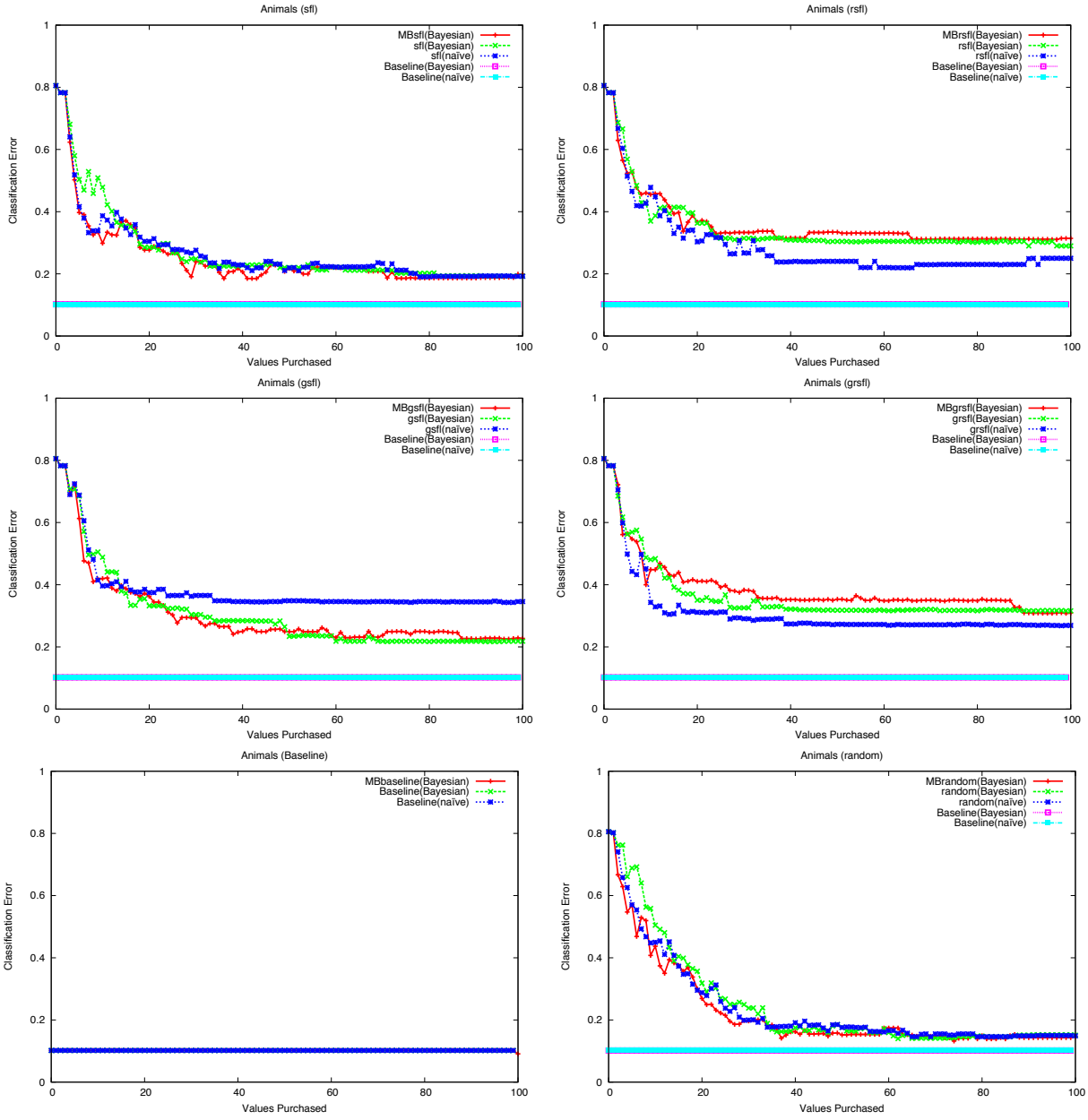


Figure 5.8: Comparing each algorithm (sfl, rsfl, gsfl, grsfl, Baseline and ranom) on Bayesian network with Markov blanket filter, on Bayesian network, and on naïve Bayes on Animals from Norsys Net Library (2011).

Table 5.2: Result of Wilcoxon signed rank test between the left side algorithm and the top side algorithm. "+" ("−") indicates that the left side algorithm reaches a significantly higher (lower) classification accuracy than the top side algorithm at budget 100 for data set **Animals** at a confidence level $p < 0.05$. "0" means the left side algorithm and the top algorithm have no significant difference of the reached classification accuracies at the given budget 100 for data set Animals.

	random (NB)	rr (NB)	br (NB)	merpg (NB)	dsep (NB)	dsepw1 (NB)	dsepw2 (NB)	sfl (NB)	random (BN)	rr (BN)	br (BN)	merpg (BN)	dsep (BN)	dsepw1 (BN)	dsepw2 (BN)	sfl (BN)	MBrandom (BN)	MBrr (BN)	MBbr (BN)	MBmerpg (BN)	MBdsep (BN)	MBdsepw1 (BN)	MBdsepw2 (BN)	MBsfl (BN)	Wins	Losses
random(NB)	0	0	0	−	−	−	−	0	+	0	0	−	−	−	−	0	0	0	0	−	−	−	−	0	1	12
rr(NB)	0	0	0	−	−	−	−	0	0	0	0	−	−	−	−	0	0	0	0	−	−	−	−	0	0	12
br(NB)	0	0	0	−	−	−	−	0	0	0	0	−	−	−	−	0	0	0	0	−	−	−	−	0	0	12
merpg(NB)	+	+	+	0	0	0	0	+	+	+	0	0	0	0	0	0	+	0	+	0	0	0	0	+	9	0
dsep(NB)	+	+	+	0	0	0	0	+	+	+	0	0	0	0	0	0	+	0	+	0	0	0	0	+	9	0
dsepw1(NB)	+	+	+	0	0	0	0	+	+	+	0	0	0	0	0	0	+	0	+	0	0	−	−	+	9	2
dsepw2(NB)	+	+	+	0	0	0	0	+	+	+	0	0	0	0	0	0	+	0	+	0	0	−	−	+	9	2
sfl(NB)	0	0	0	−	−	−	−	0	0	0	0	0	0	0	0	0	0	0	0	0	−	−	−	0	0	7
random(BN)	−	0	0	−	−	−	−	0	0	0	0	−	−	−	−	0	0	0	0	−	−	−	−	0	0	13
rr(BN)	0	0	0	−	−	−	−	0	0	0	0	−	−	−	−	0	0	0	0	−	−	−	−	0	0	12
br(BN)	0	0	0	0	0	0	0	0	0	0	0	0	0	−	−	0	0	0	0	0	0	−	−	0	0	4
merpg(BN)	+	+	+	0	0	0	0	0	+	+	0	0	0	0	0	0	+	0	+	0	0	−	−	0	7	2
dsep(BN)	+	+	+	0	0	0	0	0	+	+	0	0	0	0	0	0	+	0	+	0	0	−	−	0	7	2
dsepw1(BN)	+	+	+	0	0	0	0	0	+	+	+	0	0	0	0	0	+	+	+	0	0	0	0	+	10	0
dsepw2(BN)	+	+	+	0	0	0	0	0	+	+	+	0	0	0	0	0	+	+	+	0	0	0	0	0	9	0
sfl(BN)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
MBrandom(BN)	0	0	0	−	−	−	−	0	0	0	0	−	−	−	−	0	0	0	0	0	−	−	−	0	0	11
MBrr(BN)	0	0	0	0	0	0	0	0	0	0	0	0	0	−	−	0	0	0	0	0	0	−	−	0	0	4
MBbr(BN)	0	0	0	−	−	−	−	0	0	0	0	−	−	−	−	0	0	0	0	−	−	−	−	0	0	12
MBmerpg(BN)	+	+	+	0	0	0	0	0	+	+	0	0	0	0	0	0	0	0	+	0	0	0	0	0	6	0
MBdsep(BN)	+	+	+	0	0	0	0	+	+	+	0	0	0	0	0	0	+	0	+	0	0	0	0	+	9	0
MBdsepw1(BN)	+	+	+	0	0	+	+	+	+	+	+	+	+	0	0	0	+	+	+	0	0	0	0	+	15	0
MBdsepw2(BN)	+	+	+	0	0	+	+	+	+	+	+	+	+	0	0	0	+	+	+	0	0	0	0	+	15	0
MBsfl(BN)	0	0	0	−	−	−	−	0	0	0	0	0	0	−	0	0	0	0	0	0	−	−	−	0	0	8

Table 5.3: Mean running time over 10 runs of each algorithm on Animal (7 nodes, 6 edges, 5000 instances). Budget = 100.

Algorithm	of naïve Bayes	of Bayesian Network	of Bayesian Network on MB
random	3 seconds	5 seconds	3 seconds
rr	1 seconds	1 seconds	1 seconds
br	9 seconds	1 minutes	1 minutes
merpg	13 minutes	80 minutes	23 minutes
dsep	15 minutes	80 minutes	24 minutes
dsepw1	15 minutes	88 minutes	24 minutes
dsepw2	15 minutes	82 minutes	24 minutes
sfl	18.66 hours	22 hours	15.5 hours
rsfl	18.66 hours	22 hours	15.5 hours
gsfl	18.2 hours	19.3 hours	15 hours
grsfl	18.3 hours	19.5 hours	15 hours

5.6.2 Experimental Results on Car Diagnosis 2

Our second learning problem is Car Diagnosis 2 (a.k.a. CarDiagnosis2) from Norsys Net Library (2011). Figure 5.3 shows the Bayesian network is a belief network for diagnosing why a car won't start, based on spark plugs, headlights, main fuse, etc. It is a nontrivial belief network with 18 nodes and 20 edges, and we generated 5000 instances based on the provided conditional probability tables. We ran rr, br, merpg (MERPG), dsep (MERPGDSEP), dsepw1 (MERPGDSEPW1), merpg (MERPGDSEPW2), MBrr (which runs rr over Markov blanket of the label node), MBbr, MBMERPG, MBdsep, MBdsepw1, MBdsepw2, sfl and MBsf on Car Diagnosis 2. Figure 5.9 shows the mean classification errors of the algorithms learning the same of classifier for data set Car Diagnosis 2. Figures 5.10 and 5.11 show the mean classification errors of each algorithm of learning naïve Bayes, Bayesian network, and Bayesian network with Markov blanket filter on Car Diagnosis 2.

Table 5.4 shows the result of Wilcoxon signed rank test between every pair of algorithms after 100 purchases for data set Car Diagnosis 2 at a confidence level $p < 0.05$. **MBrandom (BN), MBbr (BN) and MBdsep (BN)** have the maximum number of wins over other algorithms and have no losses to other algorithms, therefore we conclude that MBrandom, MBbr, and MBdsep outperform other algorithms after 100 purchases for Car Diagnosis 2. merpg (BN) has the second maximum number of wins over other algorithms and has no losses to other algorithms, therefore it outperforms all algorithms except MBrandom (BN), MBbr (BN) and MBdsep (BN) after 100 purchases for Car Diagnosis 2. **sfl (NB) and MBsfl (BN)** have no wins over other algorithms and have the maximum number of losses to other algorithms, therefore we conclude that sfl (NB) MBsfl (BN) and are outperformed by other algorithms after 100 purchases for Car Diagnosis 2.

From Figure 5.3, we can see that the label node "Car Starts" has 5 parents: "Spark Quality", "Spark Timing", "Fuel System", "Car Cranks", and "Air System", has no children

or spouses. MBbr (BN) and MBdsep (BN) choose instances of those more informative parents "Spark Quality", "Spark Timing", and "Car Cranks" more frequently than instances of other parents "Fuel System" and "Air System". MBrandom (BN), theoretically, should perform as well as MBrr (BN). We believe the reason why MBrandom (BN) performs slightly better than MBrr (BN) is a mere coincidence.

For the Mean running time over 10 runs of each algorithm on Car Diagnosis 2, as shown in Table 5.5, we can see that SFL takes much longer time than MERPG series (MERPG, MERPGDSEP, MERPGDSEPW1, MERPGDSEPW2), which also takes longer time than Random, RR, and BR. MB*algo* takes significantly less time (up to 75%) than *algo*.

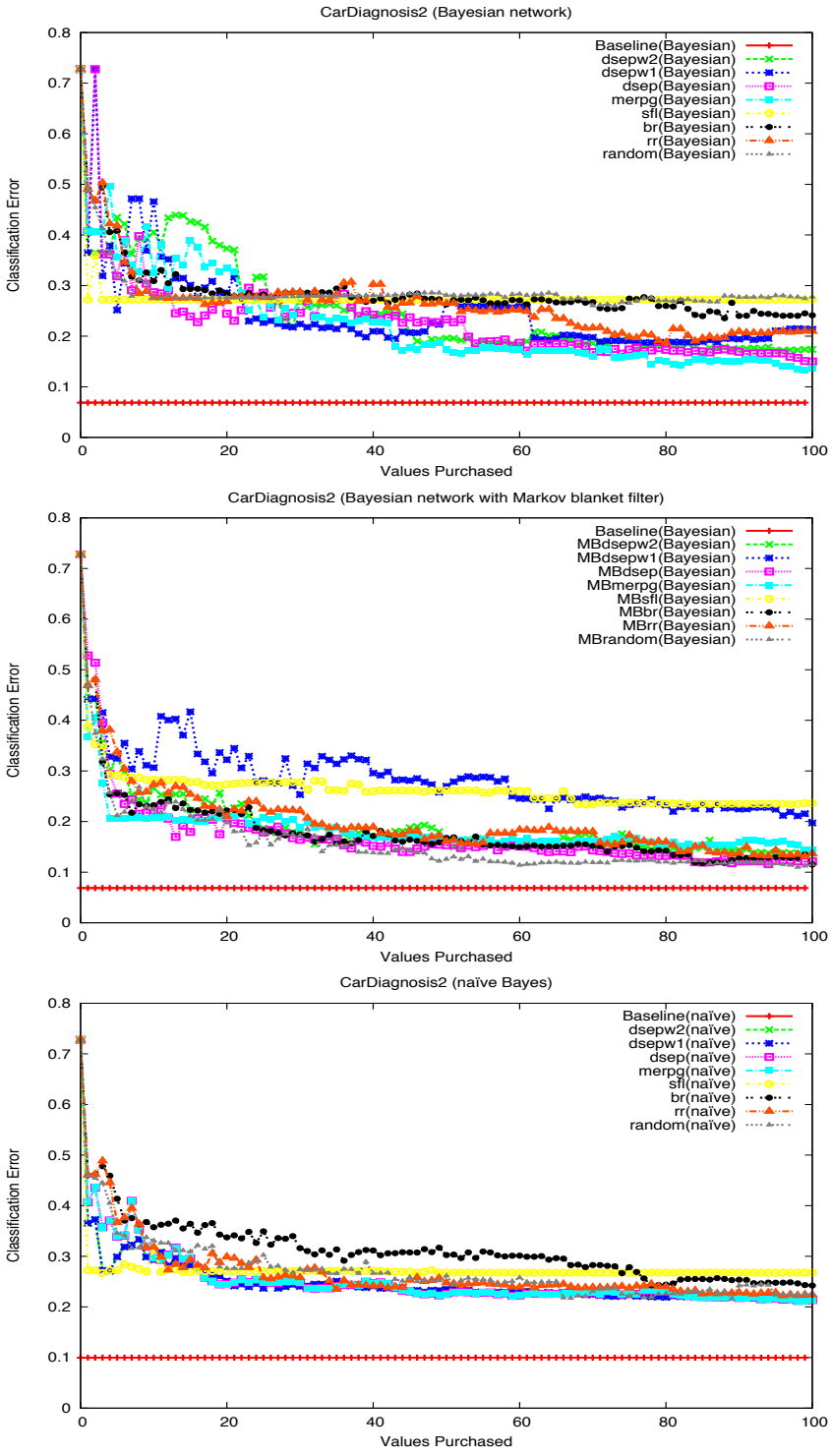


Figure 5.9: Comparing all the algorithms on Bayesian network, on Bayesian network with a Markov blanket filter, and on naïve Bayes for Car Diagnosis 2 from Norsys Net Library (2011).

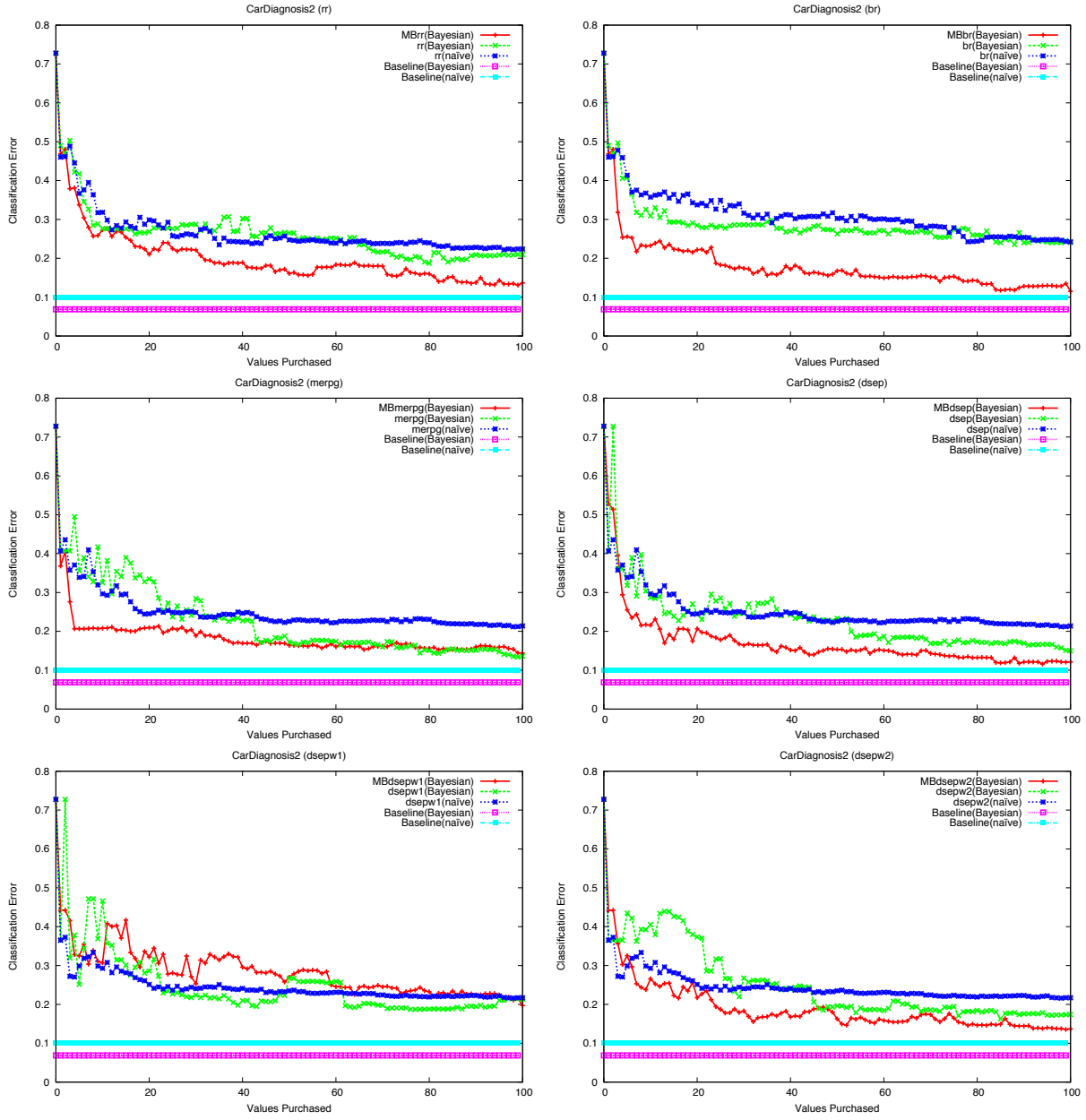


Figure 5.10: Comparing rr, br, merpg, dsep, dsepw1, and dsepw2 on Bayesian network with Markov blanket filter, on Bayesian network, and on naïve Bayes on Car Diagnosis 2 from Norsys Net Library (2011).

Table 5.4: Result of Wilcoxon signed rank test between the left side algorithm and the top side algorithm. "+" ("−") indicates that the left side algorithm reaches a significantly higher (lower) classification accuracy than the top side algorithm at budget 100 for data set **Car Diagnosis 2** at a confidence level $p < 0.05$. "0" means the left side algorithm and the top algorithm have no significant difference of the reached classification accuracies at the given budget 100 for data set Car Diagnosis 2.

	random (NB)	rr (NB)	br (NB)	merpg (NB)	dsep (NB)	dsepw1 (NB)	dsepw2 (NB)	sfl (NB)	random (BN)	rr (BN)	br (BN)	merpg (BN)	dsep (BN)	dsepw1 (BN)	dsepw2 (BN)	sfl (BN)	MBrandom (BN)	MBrr (BN)	MBbr (BN)	MBmerpg (BN)	MBdsep (BN)	MBdsepw1 (BN)	MBdsepw2 (BN)	MBsfl (BN)	Wins	Losses
random(NB)	0	0	0	0	0	0	0	+	0	0	0	−	0	0	0	0	−	−	−	−	−	0	−	+	2	7
rr(NB)	0	0	0	0	0	0	0	+	0	0	0	−	0	0	0	+	−	−	−	−	−	0	0	+	3	6
br(NB)	0	0	0	−	−	0	0	+	0	0	0	−	−	−	0	0	−	−	−	0	−	0	−	+	2	10
merpg(NB)	0	0	+	0	0	0	0	+	0	0	0	−	−	0	0	+	−	−	−	0	−	0	0	+	4	6
dsep(NB)	0	0	+	0	0	0	0	+	0	0	0	−	−	0	0	+	−	−	−	0	−	0	0	+	4	6
dsepw1(NB)	0	0	0	0	0	0	0	+	0	0	0	−	−	0	0	+	−	0	−	0	−	0	−	+	3	6
dsepw2(NB)	0	0	0	0	0	0	0	+	0	0	0	−	−	0	0	+	−	0	−	0	−	0	−	+	3	6
sfl(NB)	−	−	−	−	−	−	−	0	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	0	0	22
random(BN)	0	0	0	0	0	0	0	+	0	−	0	0	−	0	0	0	−	0	−	0	−	0	−	+	2	6
rr(BN)	0	0	0	0	0	0	0	+	+	0	0	0	0	0	0	0	0	−	−	0	−	0	0	+	3	3
br(BN)	0	0	0	0	0	0	0	+	0	0	0	−	0	0	0	0	−	0	−	−	−	0	0	+	2	5
merpg(BN)	+	+	+	+	+	+	+	+	0	0	+	0	0	0	0	+	0	0	0	0	0	0	0	+	11	0
dsep(BN)	0	0	+	+	+	+	+	+	+	0	0	0	0	0	0	+	−	0	0	0	0	0	0	+	9	1
dsepw1(BN)	0	0	+	0	0	0	0	+	0	0	0	0	0	0	0	+	−	−	−	−	−	0	0	+	4	5
dsepw2(BN)	0	0	0	0	0	0	0	+	0	0	0	0	0	0	0	+	0	0	0	0	0	0	0	+	3	0
sfl(BN)	0	−	0	−	−	−	−	+	0	0	0	−	−	−	−	0	−	−	−	−	−	0	−	+	2	15
MBrandom(BN)	+	+	+	+	+	+	+	+	+	0	+	0	+	+	0	+	0	0	0	0	0	0	0	+	14	0
MBrr(BN)	+	+	+	+	+	0	0	+	0	+	0	0	0	+	0	+	0	0	0	0	0	0	0	+	10	0
MBbr(BN)	+	+	+	+	+	+	+	+	+	+	+	0	0	+	0	+	0	0	0	0	0	0	0	+	14	0
MBmerpg(BN)	+	+	0	0	0	0	0	+	0	0	+	0	0	+	0	+	0	0	0	0	0	0	0	+	7	0
MBdsep(BN)	+	+	+	+	+	+	+	+	+	+	+	0	0	+	0	+	0	0	0	0	0	0	0	+	14	0
MBdsepw1(BN)	0	0	0	0	0	0	0	+	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	+	2	0
MBdsepw2(BN)	+	0	+	0	0	+	+	+	+	0	0	0	0	0	0	+	0	0	0	0	0	0	0	+	8	0
MBsfl(BN)	−	−	−	−	−	−	−	0	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	0	0	22

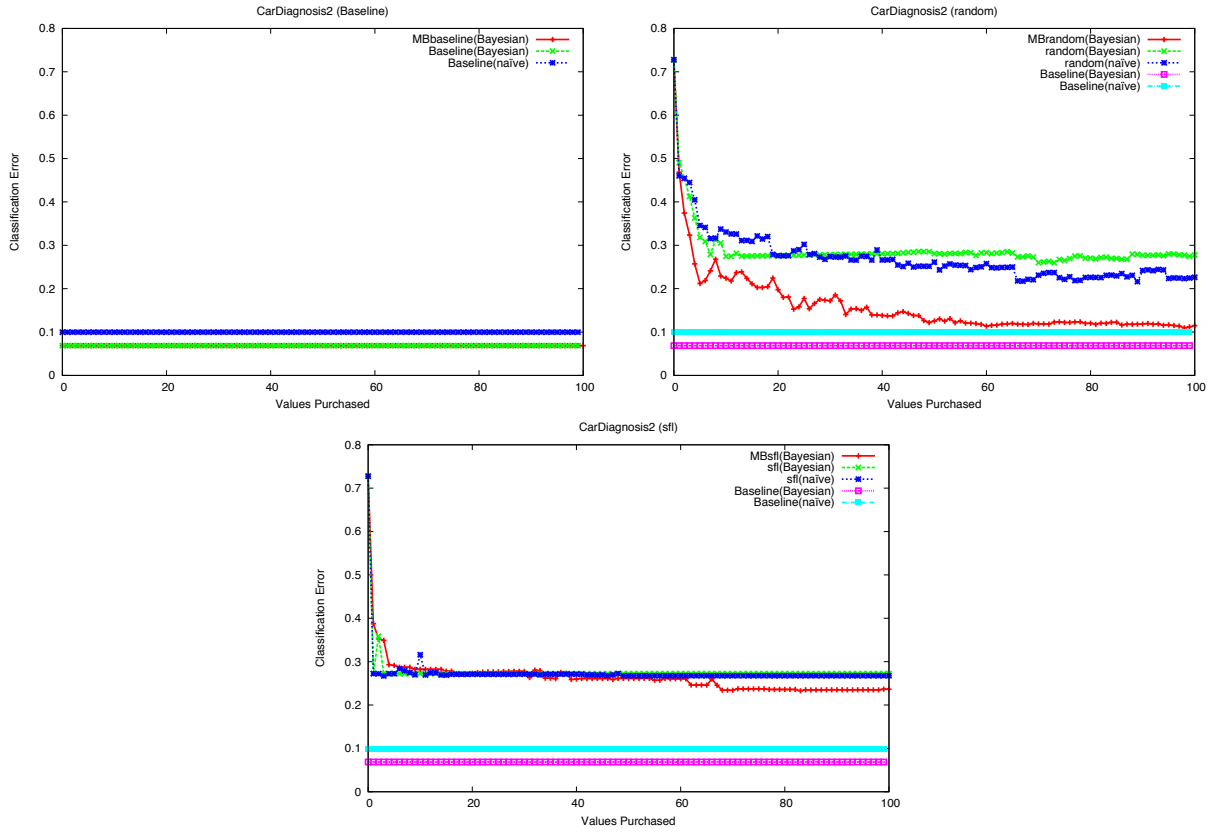


Figure 5.11: Comparing Baseline, random, and sfl on Bayesian network with Markov blanket filter, on Bayesian network, and on naïve Bayes on Car Diagnosis 2 from Norsys Net Library (2011).

Table 5.5: Mean running time over 10 runs of each algorithm on Car Diagnosis 2 (18 nodes, 20 edges, 5000 instances). Budget=100.

Algorithm	of naïve Bayes	of Bayesian Network	of Bayesian Network on MB
random	5 seconds	120 seconds	30 seconds
rr	1 second	150 seconds	35 seconds
br	3 seconds	60 seconds	50 seconds
merpg	50 minutes	80 minutes	23 minutes
dsep	54 minutes	80 minutes	24 minutes
dsepw1	54 minutes	88 minutes	24 minutes
dsepw2	54 minutes	82 minutes	24 minutes
sfl	129 hours	212 hours	74 hours

5.6.3 Experimental Results on Chest Clinic

Our third learning problem is Chest Clinic (a.k.a. ChestClinic) from Norsys Net Library (2011). This Bayes net is also known as "Asia", and is popular for introducing Bayes nets. It is a belief network with 8 nodes and 8 edges, and we generated 5000 instances based on the provided conditional probability tables. We ran rr, br, MERPG (MERPG), dsep (MERPGDSEP), dsepw1 (MERPGDSEPW1), dsepw2 (MERPGDSEPW2), sfl, gsfl, rsfl, grsfl on naïve Bayes, Bayesian network, and Bayesian network with Markov blanket filter on Chest Clinic.

Figure 5.12 shows the classfication errors of the algorithms learning same classifier. Classifiers are Bayesian network, Bayesian network with Markov blanket filter, and naïve Bayes on Chest Clinic. Figures 5.13 and 5.14 show the classification errors of each algorithm on Bayesian network with Markov blanket, Bayesian network, and naïve Bayes. on Chest Clinic.

Table 5.6 shows the result of Wilcoxon signed rank test between every pair of algorithms after 40 purchases for data set Chest Clinic at a confidence level $p < 0.05$. **MBrr (BN)** has the maximum number of wins over other algorithms and has no losses other algorithms, therefore we conclude that MBrr (BN) outperforms other algorithms after 40 purchases for Chest Clinic. **merpg (NB)** has no wins over other algorithms and has the maximum number of losses to other algorithms, therefore we conclude that merpg (NB) is outperformed by other algorithms after 40 purchases for Chest Clinic.

From Figure 5.1, we can see that Chest Clinic has 2 parents ("Tuberculosis" and "Lung Cancer"), 2 children ("X-Ray Result" and "Dyspnea"), and 1 spouse ("Bronchitis"). The remaining 2 nodes are "Smoking" and "Visit to Asia", which are the label node's two grandparents.

The reason that MBmerpg (BN) and MBdsep (BN) do not perform better than MBrandom (BN) and MBrr (BN) on Chest Clinic is that the difference of expected relative probability gain of choosing different features are slightly different while MBmerpg and MBdsep

chooses the (instance, feature) pair which is slightly better. Further investigation for the reason why for Chest Clinic, MBrr (BN) and MBrandom (BN) perform better than MBmerpg (BN) and MBdsep (BN) is future work.

For the Mean running time over 10 runs of each algorithm on Chest Clinic, as shown in Table 5.7, we can see that SFL takes much longer time than MERPG series (MERPG, MERPGDSEP, MERPGDSEPW1, MERPGDSEPW2), which also takes longer time than Random, RR, and BR. MB*algo* takes significantly less time (up to 95%) than *algo*.

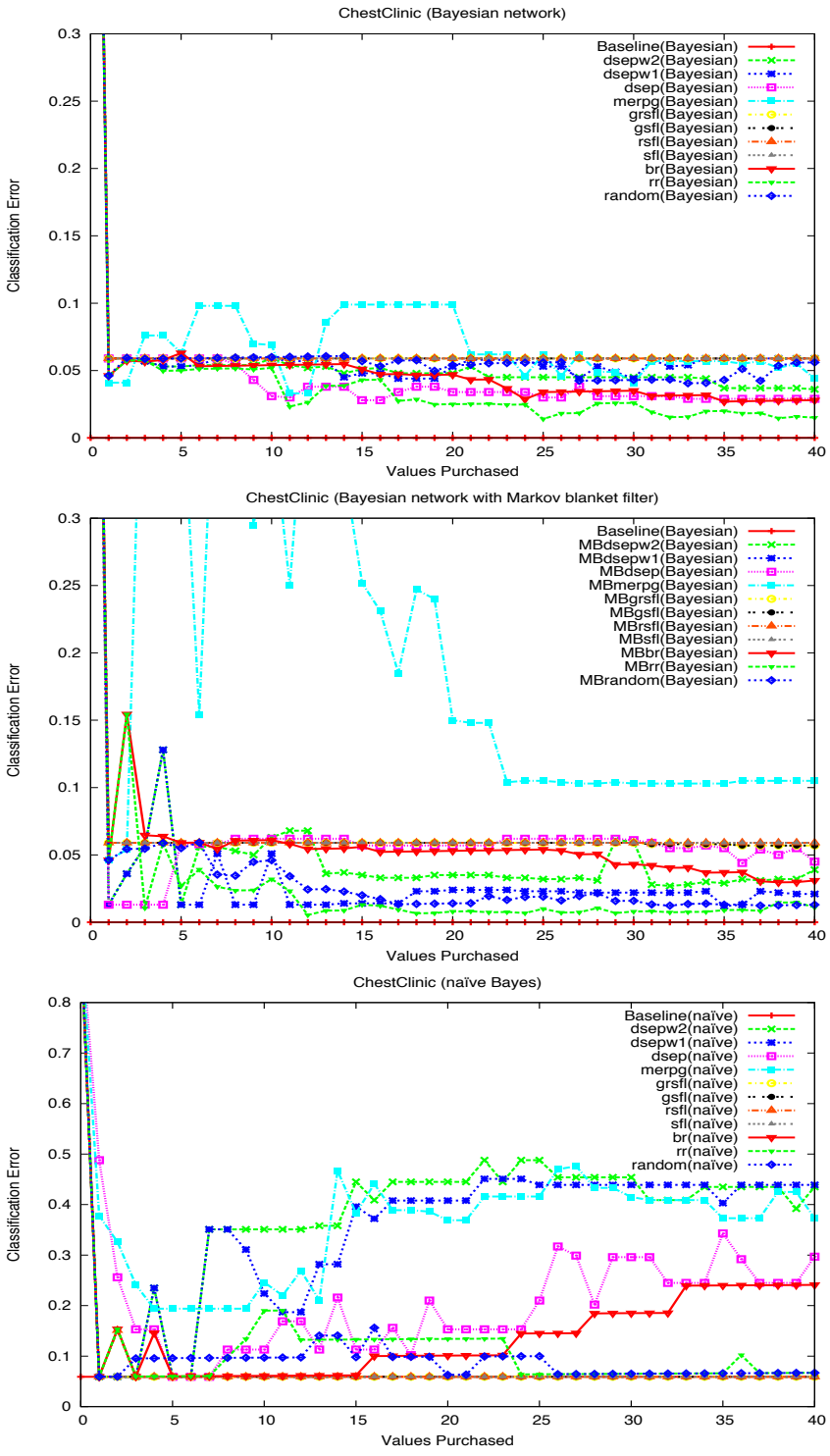


Figure 5.12: Comparing all the algorithms of Bayesian network, of Bayesian network with Markov blanket filter, and of naïve Bayes on Chest Clinic from Norsys Net Library (2011).

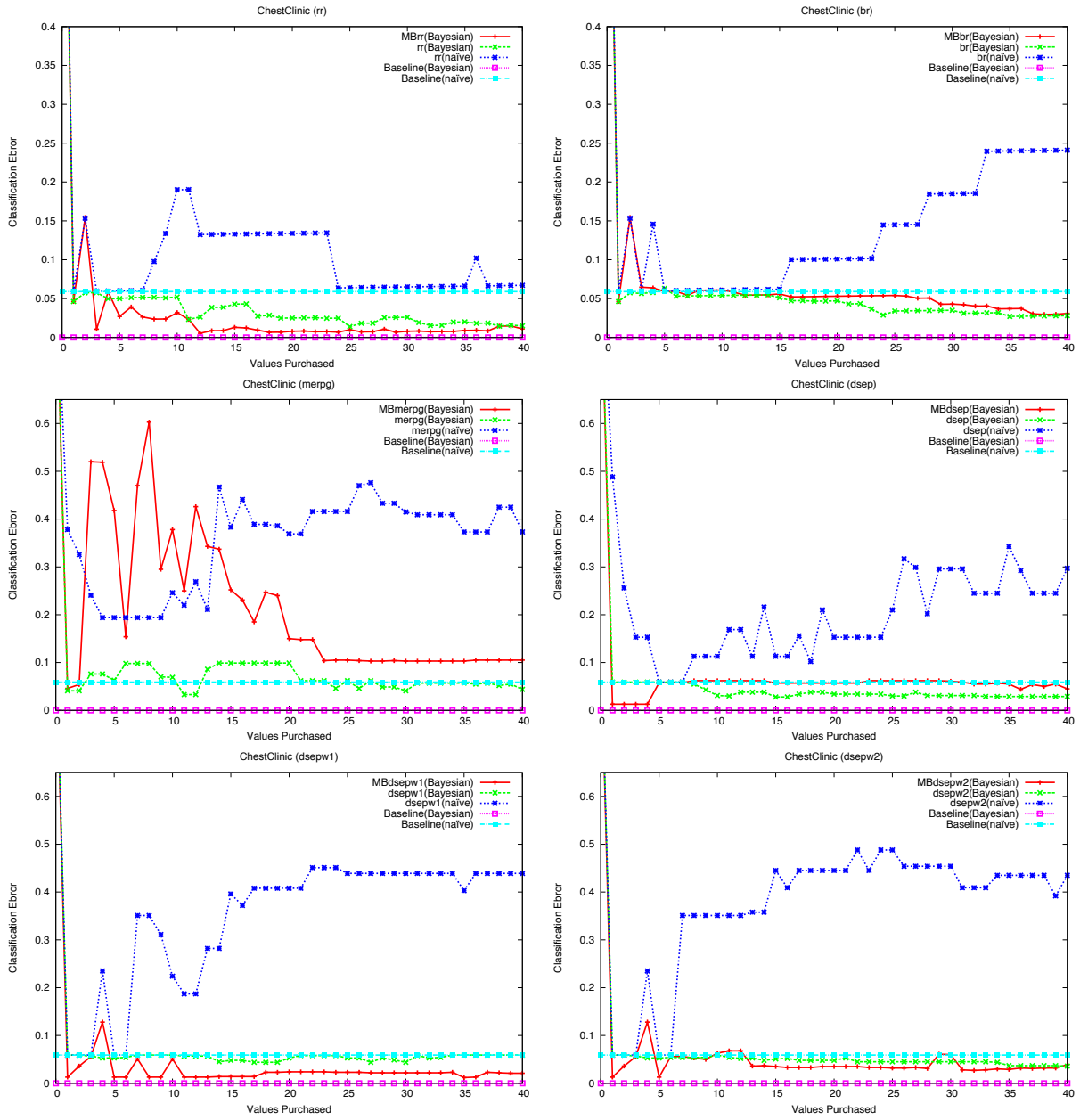


Figure 5.13: Comparing each algorithm (rr, br, merpg, dsep, dsepw1, dsepw2) on Bayesian network with Markov blanket filter, on Bayesian network, and on naïve Bayes on on Chest Clinic from Norsys Net Library (2011).

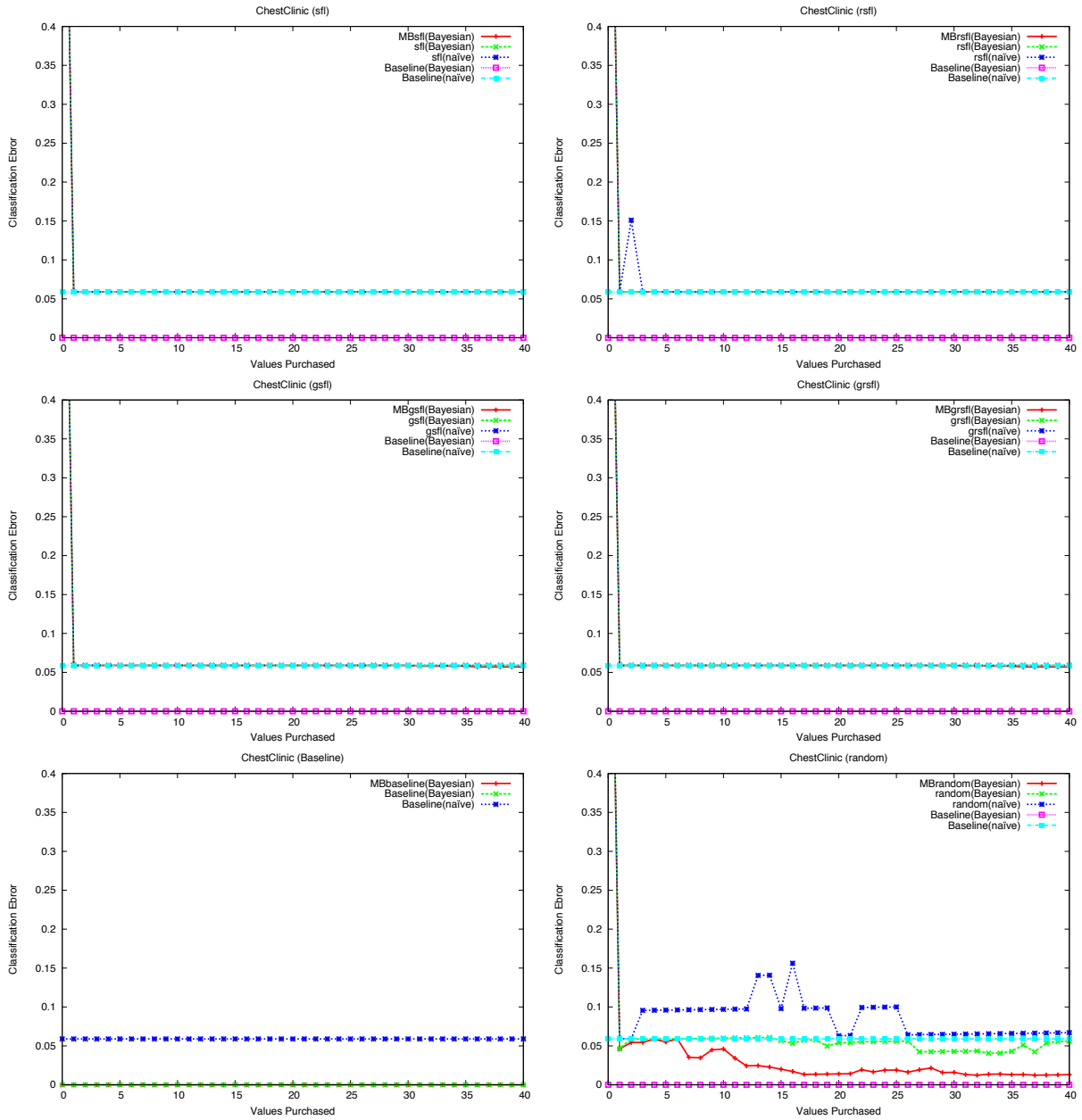


Figure 5.14: Comparing each algorithm (sfl, rsfl, gsfl, grsfl, Baseline, and random) on Bayesian network with Markov blanket filter, on Bayesian network with Markov blanket filter, on Bayesian network, and on naïve Bayes on Chest Clinic from Norsys Net Library (2011).

Table 5.6: Result of Wilcoxon signed rank test between the left side algorithm and the top side algorithm. "+" ("−") indicates that the left side algorithm reaches a significantly higher (lower) classification accuracy than the top side algorithm at budget 40 for data set **Chest Clinic** at a confidence level $p < 0.05$. "0" means the left side algorithm and the top algorithm have no significant difference of the reached classification accuracies at the given budget 40 for data set Chest Clinic.

	random (NB)	rr (NB)	br (NB)	merpg (NB)	dsep (NB)	dsepw1 (NB)	dsepw2 (NB)	sfl (NB)	random (BN)	rr (BN)	br (BN)	merpg (BN)	dsep (BN)	dsepw1 (BN)	dsepw2 (BN)	sfl (BN)	MBrandom (BN)	MBrr (BN)	MBbr (BN)	MBmerpg (BN)	MBdsep (BN)	MBdsepw1 (BN)	MBdsepw2 (BN)	MBsfl (BN)	Wins	Losses
random(NB)	0	0	+	+	0	0	0	−	−	−	−	0	−	0	0	−	−	−	−	0	0	0	0	−	2	10
rr(NB)	0	0	+	+	0	0	0	−	−	−	−	0	−	0	0	−	−	−	−	0	0	0	0	−	2	10
br(NB)	−	−	0	0	−	0	0	−	−	−	−	0	−	0	0	−	−	−	−	0	0	−	0	−	0	14
merpg(NB)	−	−	0	0	0	0	0	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	0	19
dsep(NB)	0	0	+	0	0	0	0	−	0	0	0	0	−	0	0	−	−	−	−	0	0	0	0	−	1	7
dsepw1(NB)	0	0	0	0	0	0	0	−	0	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	0	16
dsepw2(NB)	0	0	0	0	0	0	0	−	−	0	−	−	−	−	−	−	−	−	−	−	−	−	−	−	0	16
sfl(NB)	+	+	+	+	+	+	+	0	0	0	0	0	0	0	0	0	−	−	−	0	0	0	0	0	7	3
random(BN)	+	+	+	+	0	0	+	0	0	−	0	0	0	0	0	0	−	−	0	0	0	0	0	0	5	3
rr(BN)	+	+	+	+	0	+	0	0	+	0	0	0	0	0	0	0	0	0	+	0	0	0	0	0	7	0
br(BN)	+	+	+	+	0	+	+	0	0	0	0	0	0	0	0	0	−	−	0	0	0	0	0	0	6	2
merpg(BN)	0	0	0	+	0	+	+	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0
dsep(BN)	+	+	+	+	+	+	+	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0
dsepw1(BN)	0	0	0	+	0	+	+	0	0	0	0	0	0	0	0	−	0	−	−	0	0	0	0	0	3	3
dsepw2(BN)	0	0	0	+	0	+	+	0	0	0	0	0	0	+	0	0	0	−	0	0	0	0	0	0	4	1
sfl(BN)	+	+	+	+	+	+	+	0	0	0	0	0	0	0	0	0	−	−	−	0	0	0	0	0	7	3
MBrandom(BN)	+	+	+	+	+	+	+	+	+	0	0	0	0	+	0	+	0	0	+	0	0	0	0	+	13	0
MBrr(BN)	+	+	+	+	+	+	+	+	+	0	+	0	0	+	+	+	0	0	+	0	0	0	0	+	15	0
MBbr(BN)	+	+	+	+	+	+	+	+	0	−	0	0	0	0	0	+	−	−	0	0	0	0	0	+	10	3
MBmerpg(BN)	0	0	0	+	0	+	+	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0
MBdsep(BN)	0	0	0	+	0	+	+	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0
MBdsepw1(BN)	0	0	+	+	0	+	+	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0
MBdsepw2(BN)	0	0	0	+	0	+	+	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0
MBsfl(BN)	+	+	+	+	+	+	+	0	0	0	0	0	0	0	0	0	−	−	−	0	0	0	0	0	7	3

Table 5.7: Mean running time over 10 runs of each algorithm on Chest Clinic (8 nodes, 8 edges, 5000 instances). Budget = 30.

Algorithm	of naïve Bayes	of Bayesian Network	of Bayesian Network on MB
random	6 seconds	14 seconds	7 seconds
rr	3 seconds	10 seconds	8 seconds
br	25 seconds	100 seconds	5 second
merpg	15 minutes	15 minutes	10 minutes
dsep	15 minutes	17.5 minutes	12.5 minutes
dsepw1	15 minutes	17.5 minutes	12.5 minutes
dsepw2	15 minutes	17.5 minutes	12.5 minutes
sfl	11 hours	20 hours	15 hours
rsfl	10 hours	21 hours	15 hours
gsfl	10 hours	20 hours	15 hours
grsfl	10 hours	20 hours	17 hours

5.6.4 Experimental Results on Poya Ganga

The fourth learning problem is Poya Ganga (a.k.a. Poya_Ganga). The Bayesian network of Poya Ganga is the product of a hypothetical case study about managing water resources in the Poya Ganga. It is a belief network with 60 nodes and 65 edges. We generated 1000 instances based on the provided probability table. And we ran rr, br, merpg (MERPG), dsep (MERPGDSEP), dsepw1 (MERPGDSEPW1), dsepw2 (MERPGDSEPW2), MBrr, MBbr, MBMERPG, MBdsep, MBdsepw1, MBdsepw2, sfl, and MBsfl on Poya Ganga.

Figure 5.15 shows the classification errors of the algorithms learning same classifier. Classifiers are Bayesian network, Bayesian network with Markov blanket filter, and naïve Bayes.

Figures 5.16 and 5.17 show that the classification errors of an algorithm of Bayesian network with Markov blanket filter, of Bayesian network, and of naïve Bayes on Poya Ganga.

Table 5.8 shows the result of Wilcoxon signed rank test between every pair of algorithms after 30 purchases for data set Poya Ganga at a confidence level $p < 0.05$. **MBdsep (BN) and MBdsepw2 (BN)** have the maximum number of wins over other algorithms and have no losses other algorithms, therefore we conclude that MBdsep (BN) and MBdsepw2 (BN) outperform other algorithms after 30 purchases for Poya Ganga. **br (BN)** has no wins over other algorithms and has the maximum number of losses to other algorithms, therefore we conclude that br (BN) is outperformed by other algorithms after 30 purchases for Poya Ganga. From Table 5.8, we can see many pairs of algorithms have no significant difference. The reason why many pairs of algorithms have no significant difference is that Poya Ganga is a difficult learning problem whose baseline on naïve Bayes has an accuracy of 0.455 and whose baseline on Bayesian network is 0.545. Therefore, algorithms such as merpg, dsep, dsepw1, and dsepw2 cannot benefit from the structure of the Bayesian network and the improved probabilistic model because of the structure. Even random, rr, and br do not

perform well because of the the difficulty of classification for Poya Ganga whose Baseline is extremely low.

From Figure 5.4, we can see its label node "Rice Yield" has two parents "Paddy Demand Met?" and "Input". The reason that MBdsep (BN) and MBdsepw2 perform slightly better than other algorithms on Poya Ganga is that they identify the more important parent feature by the weighting factor.

For the Mean running time over 10 runs of each algorithm on Poya Ganga, as shown in Table 5.9, we can see that SFL takes much longer time than MERPG series (MERPG, MERPGDSEP, MERPGDSEPW1, MERPGDSEPW2), which also takes longer time than Random, RR, and BR. MB*algo* takes significantly less time (up to 94%) than *algo*.

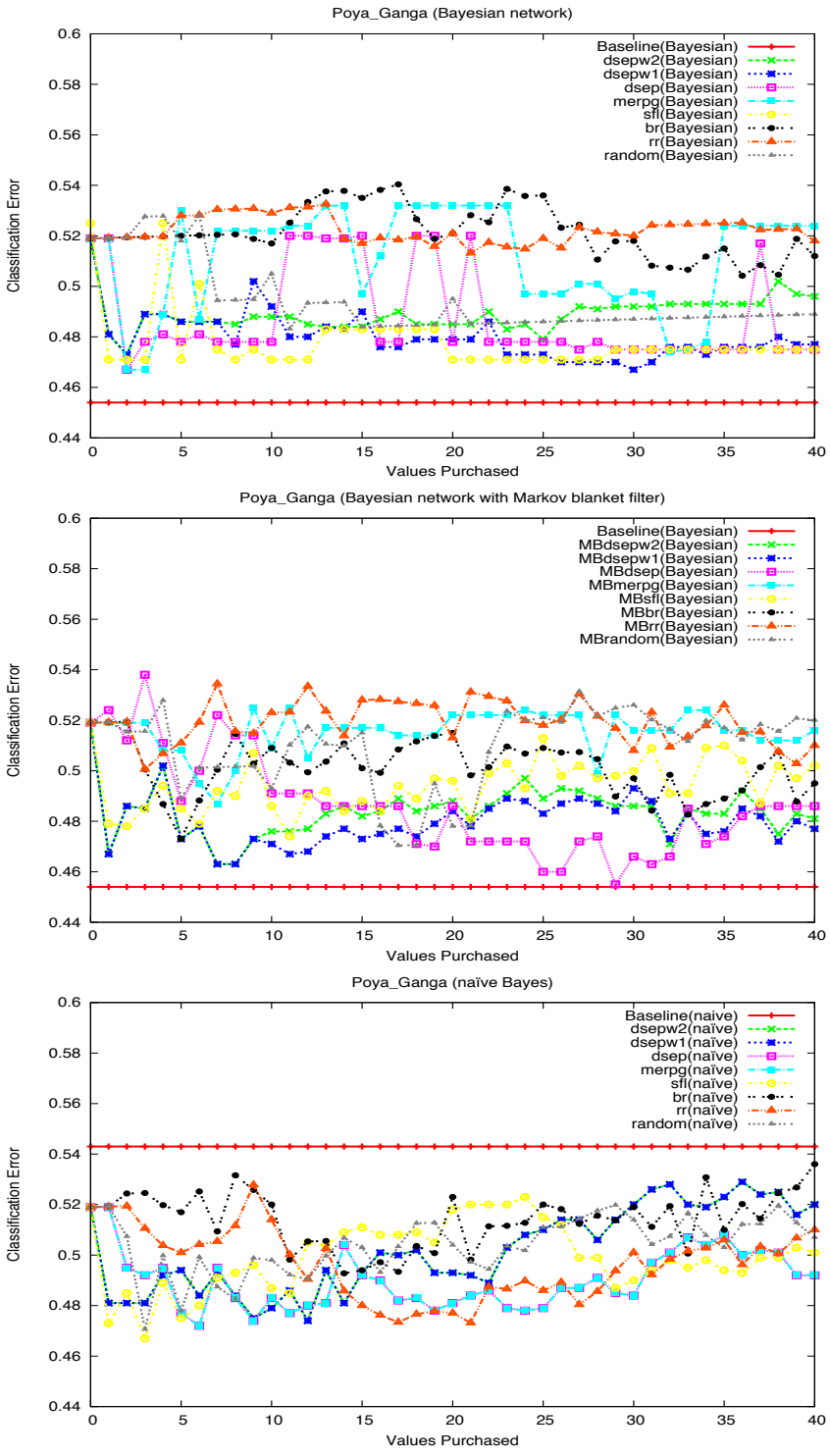


Figure 5.15: Comparing all algorithms on Bayesian network, on Bayesian network with Markov blanket filter, and on naïve Bayes on Poya Ganga from Norsys Net Library (2011).

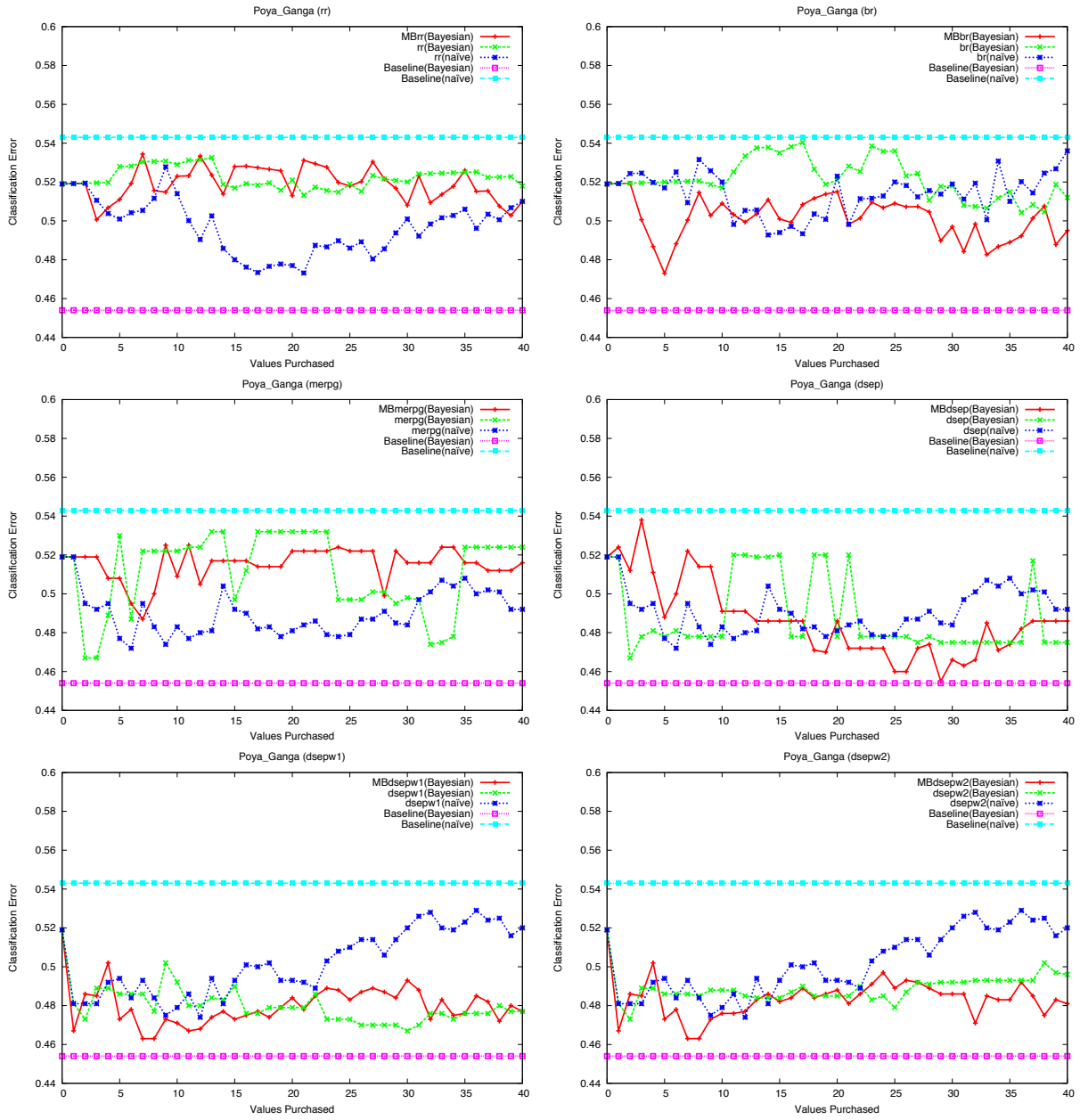


Figure 5.16: Comparing rr, br, merpg, dsep, dsepw1, and dsepw2 on Bayesian network, on Bayesian network with Markov blanket filter, and on naïve Bayes on Poya Ganga from Norsys Net Library (2011).

Table 5.8: Result of Wilcoxon signed rank test between the left side algorithm and the top side algorithm. "+" ("−") indicates that the left side algorithm reaches a significantly higher (lower) classification accuracy than the top side algorithm at budget 30 for data set **Poya Ganga** at a confidence level $p < 0.05$. "0" means the left side algorithm and the top algorithm have no significant difference of the reached classification accuracies at the given budget 30 for data set Poya Ganga.

	random (NB)	rr (NB)	br (NB)	merpg (NB)	dsep (NB)	dsepw1 (NB)	dsepw2 (NB)	sfl (NB)	random (BN)	rr (BN)	br (BN)	merpg (BN)	dsep (BN)	dsepw1 (BN)	dsepw2 (BN)	sfl (BN)	MBrandom (BN)	MBrr (BN)	MBbr (BN)	MBmerpg (BN)	MBdsep (BN)	MBdsepw1 (BN)	MBdsepw2 (BN)	MBsfl (BN)	Wins	Losses
random(NB)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
rr(NB)	0	0	0	0	0	+	+	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0
br(NB)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	−	0	−	0	0	2
merpg(NB)	0	0	0	0	0	0	0	0	0	0	+	0	0	−	0	0	0	0	0	0	0	0	0	0	1	1
dsep(NB)	0	0	0	0	0	0	0	0	0	0	+	0	0	−	0	0	0	0	0	0	0	0	0	0	1	1
dsepw1(NB)	0	−	0	0	0	0	0	0	0	0	0	−	0	0	0	−	0	0	0	0	0	0	−	0	0	4
dsepw2(NB)	0	−	0	0	0	0	0	0	0	0	0	−	0	0	0	−	0	0	0	0	0	0	−	0	0	4
sfl(NB)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
random(BN)	0	0	0	0	0	0	0	0	0	0	+	0	0	0	0	−	0	0	0	0	0	0	0	0	1	1
rr(BN)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
br(BN)	0	0	0	−	−	0	0	0	−	0	0	0	0	0	0	−	0	0	0	0	−	0	0	0	0	5
merpg(BN)	0	0	0	0	0	+	+	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0
dsep(BN)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
dsepw1(BN)	0	0	0	+	+	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0
dsepw2(BN)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
sfl(BN)	0	0	0	0	0	0	0	0	+	0	+	0	0	0	0	0	+	0	0	0	0	0	0	0	3	0
MBrandom(BN)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	−	0	0	0	0	−	0	0	0	0	2
MBrr(BN)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
MBbr(BN)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
MBmerpg(BN)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
MBdsep(BN)	0	0	+	0	0	0	0	0	0	0	+	0	0	0	0	0	+	0	0	0	0	0	0	+	4	0
MBdsepw1(BN)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	−	0	0	1
MBdsepw2(BN)	0	0	+	0	0	+	+	0	0	0	0	0	0	0	0	0	0	0	0	0	0	+	0	0	4	0
MBsfl(BN)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	−	0	0	0	0	1

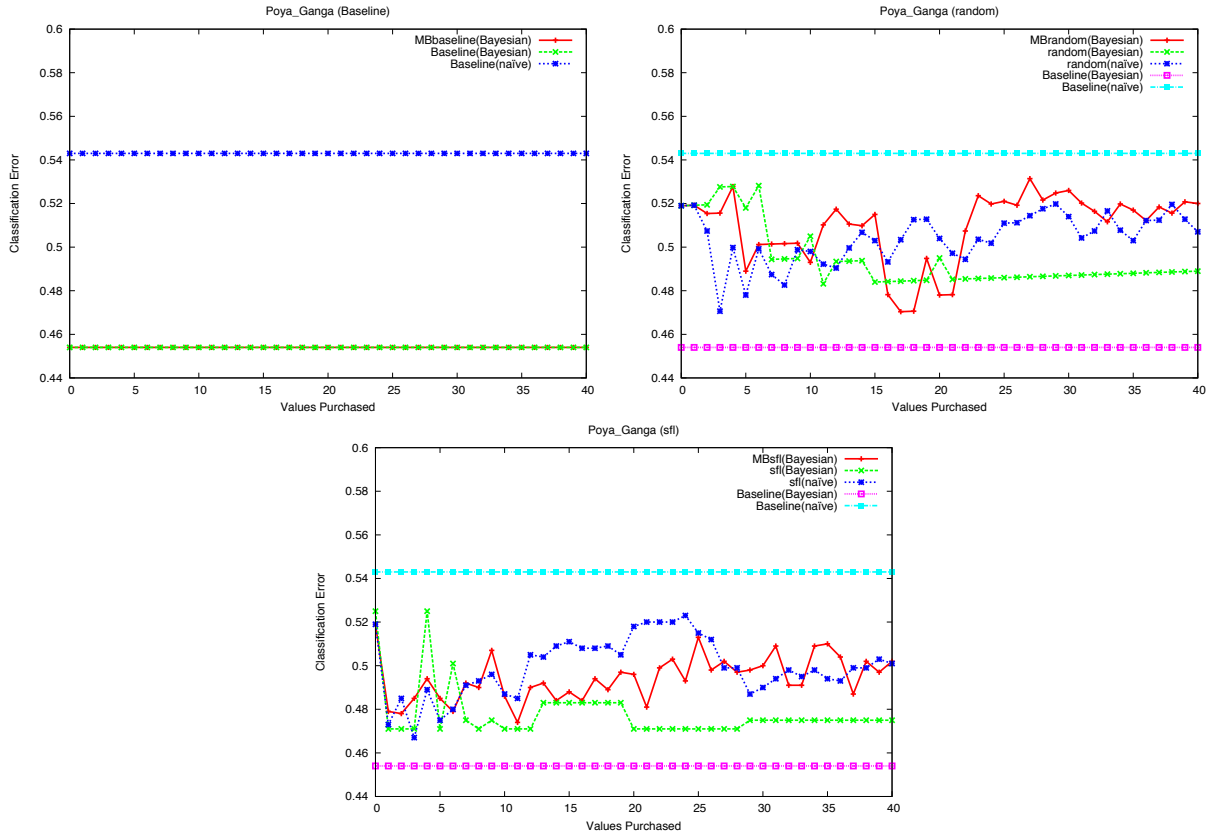


Figure 5.17: Comparing Baseline, random and sfl on naïve Bayes, Bayesian network, and Bayesian network with Markov blanket filter on Poya Ganga from Norsys Net Library (2011).

Table 5.9: Mean running time over 10 runs of algorithms on data set Poya Ganga (60 nodes, 65 edges, 1000 instances). Budget=40.

Algorithm	of naïve Bayes	of Bayesian Network	of Bayesian Network on MB
random	2 seconds	1200 seconds	210 seconds
rr	1 second	180 seconds	120 seconds
br	1 second	240 seconds	180 seconds
merpg	56 minutes	180 minutes	11 minutes
dsep	60 minutes	180 minutes	13 minutes
dsepw1	60 minutes	180 minutes	13 minutes
dsepw2	60 minutes	180 minutes	13 minutes
sfl	50 hours	79 hours	66 hours

5.6.5 Experimental Results on ALARM

The fifth learning problem is ALARM. Alarm stands for "A Logical Alarm Reduction Mechanism". This is a medical diagnostic system for patient monitoring. It is a nontrivial belief network with 37 nodes and 46 edges. We generated 1000 instances for ALARM. We ran rr, br, merpg (MERPG), dsep (MERPGDSEP), dsepw1 (MERPGDSEPW1), dsepw2 (MERPGDSEPW2), and sfl on a Bayesian network, Bayesian network with Markov blanket filter, and naïve Bayes on ALARM. Figure 5.18 shows the classification errors of the algorithms learning the same classifier on ALARM. Classifiers are Bayesian network with Markov blanket fiter, Bayesian network, and naïve Bayes. Figures 5.19 and 5.20 show the classification errors of an algorithm of Bayesian network with Markov blanket filter, of Bayesian network, and of naïve Bayes on ALARM.

Table 5.10 shows the result of Wilcoxon signed rank test between every two algorithms after 100 purchases for data set ALARM at a confidence level $p < 0.05$. **MBdsep (BN)** has the maximum number of wins over other algorithms and has no losses to other algorithms, therefore we conclude that MBdsep (BN) outperforms other algorithms after 100 purchases for ALARM. MBmerpg (BN) has the second maximum number of wins over other algorithms with only 1 loss to MBdsep (BN), therefore it is the second best algorithm. **sfl (NB)** has no wins over other algorithms and has the maximum number of losses to other algorithms, therefore we conclude that sfl (NB) is outperformed by other algorithms after 100 purchases for ALARM.

From Figure 5.5, we can see that the label node of ALARM "Press" has 3 parents "Intubation", "Vent Tube", and "Kinked Tube". MBmerpg (BN) identifies more informative parent nodes "Intubation" and "Vent Tube" and purchases instances of these two more frequently, and MBdsep further improve the result by breaking ties via NumIncreaseDseps.

For the Mean running time over 10 runs of each algorithm on Poya Ganga, as shown

Table 5.10: Result of Wilcoxon signed rank test between the left side algorithm and the top side algorithm. "+" ("−") indicates that the left side algorithm reaches a significantly higher (lower) classification accuracy than the top side algorithm at budget 100 for data set **ALARM** at a confidence level $p < 0.05$. "0" means the left side algorithm and the top algorithm have no significant difference of the reached classification accuracies at the given budget 100 for data set ALARM.

	random (NB)	rr (NB)	br (NB)	merpg (NB)	dsep (NB)	dsepw1 (NB)	dsepw2 (NB)	sfl (NB)	random (BN)	rr (BN)	br (BN)	merpg (BN)	dsep (BN)	dsepw1 (BN)	dsepw2 (BN)	sfl (BN)	MBrandom (BN)	MBrr (BN)	MBbr (BN)	MBmerpg (BN)	MBdsep (BN)	MBdsepw1 (BN)	MBdsepw2 (BN)	MBsfl (BN)	Wins	Losses
rr(NB)	0	0	+	+	+	+	+	+	0	0	0	0	−	−	−	+	−	0	−	−	−	−	−	0	7	9
br(NB)	0	−	0	+	+	+	+	+	−	0	0	−	−	−	−	0	−	0	−	−	−	−	−	0	5	12
merpg(NB)	−	−	−	0	0	0	0	+	−	0	−	0	0	−	0	−	−	−	−	−	−	0	−	0	1	13
dsep(NB)	−	−	−	0	0	0	0	+	−	0	−	0	0	−	0	−	−	−	−	−	−	0	−	0	1	13
dsepw1(NB)	−	−	−	0	0	0	0	+	−	−	−	0	0	−	0	−	−	−	−	−	−	−	−	0	1	15
dsepw2(NB)	−	−	−	0	0	0	0	+	−	−	−	0	0	−	0	−	−	−	−	−	−	−	−	0	1	15
sfl(NB)	−	−	−	−	−	−	−	0	−	−	−	−	−	−	−	−	−	−	−	−	−	−	−	0	0	22
random(BN)	0	0	+	+	+	+	+	+	0	0	0	−	−	−	−	+	−	−	−	−	−	−	−	0	7	11
rr(BN)	0	0	0	0	0	+	+	+	0	0	0	−	−	0	−	0	0	0	−	−	−	−	−	0	3	8
br(BN)	0	0	0	+	+	+	+	+	0	0	0	0	0	0	0	0	0	0	0	−	−	0	−	0	5	3
merpg(BN)	0	0	+	0	0	0	0	+	+	+	0	0	0	0	0	+	0	0	−	0	−	−	−	0	5	4
dsep(BN)	+	+	+	0	0	0	0	+	+	+	0	0	0	0	0	+	0	0	0	−	−	0	−	0	7	3
dsepw1(BN)	+	+	+	+	+	+	+	+	+	0	0	0	0	0	0	+	0	0	−	−	−	0	0	0	10	3
dsepw2(BN)	+	+	+	0	0	0	0	+	+	+	0	0	0	0	0	+	0	0	−	−	−	0	0	0	7	3
sfl(BN)	−	−	0	+	+	+	+	+	−	0	0	−	−	−	−	0	−	−	−	−	−	−	−	0	5	14
MBrandom(BN)	+	+	+	+	+	+	+	+	+	0	0	0	0	0	0	+	0	0	0	0	0	0	0	0	10	0
MBrr(BN)	0	0	0	+	+	+	+	+	+	0	0	0	0	0	0	+	0	0	0	−	−	0	−	0	7	3
MBbr(BN)	+	+	+	+	+	+	+	+	+	+	0	+	0	+	+	+	0	0	0	0	−	0	0	0	14	1
MBmerpg(BN)	+	+	+	+	+	+	+	+	+	+	+	0	+	+	+	+	0	+	0	0	−	0	+	0	17	1
MBdsep(BN)	+	+	+	+	+	+	+	+	+	+	+	+	0	+	+	+	0	+	+	+	0	0	+	0	20	0
MBdsepw1(BN)	+	+	+	0	0	+	+	+	+	+	0	+	0	0	0	+	0	0	0	0	0	0	0	0	10	0
MBdsepw2(BN)	+	+	+	+	+	+	+	+	+	+	+	0	0	0	0	+	0	0	0	−	−	0	0	0	12	2
MBsfl(BN)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

in Table 5.11, we can see that SFL takes much longer time than MERPG series (MERPG, MERPGDSEP, MERPGDSEPW1, MERPGDSEPW2), which also takes longer time than Random, RR, and BR. MB*algo* takes significantly less time (up to 58%) than *algo*.

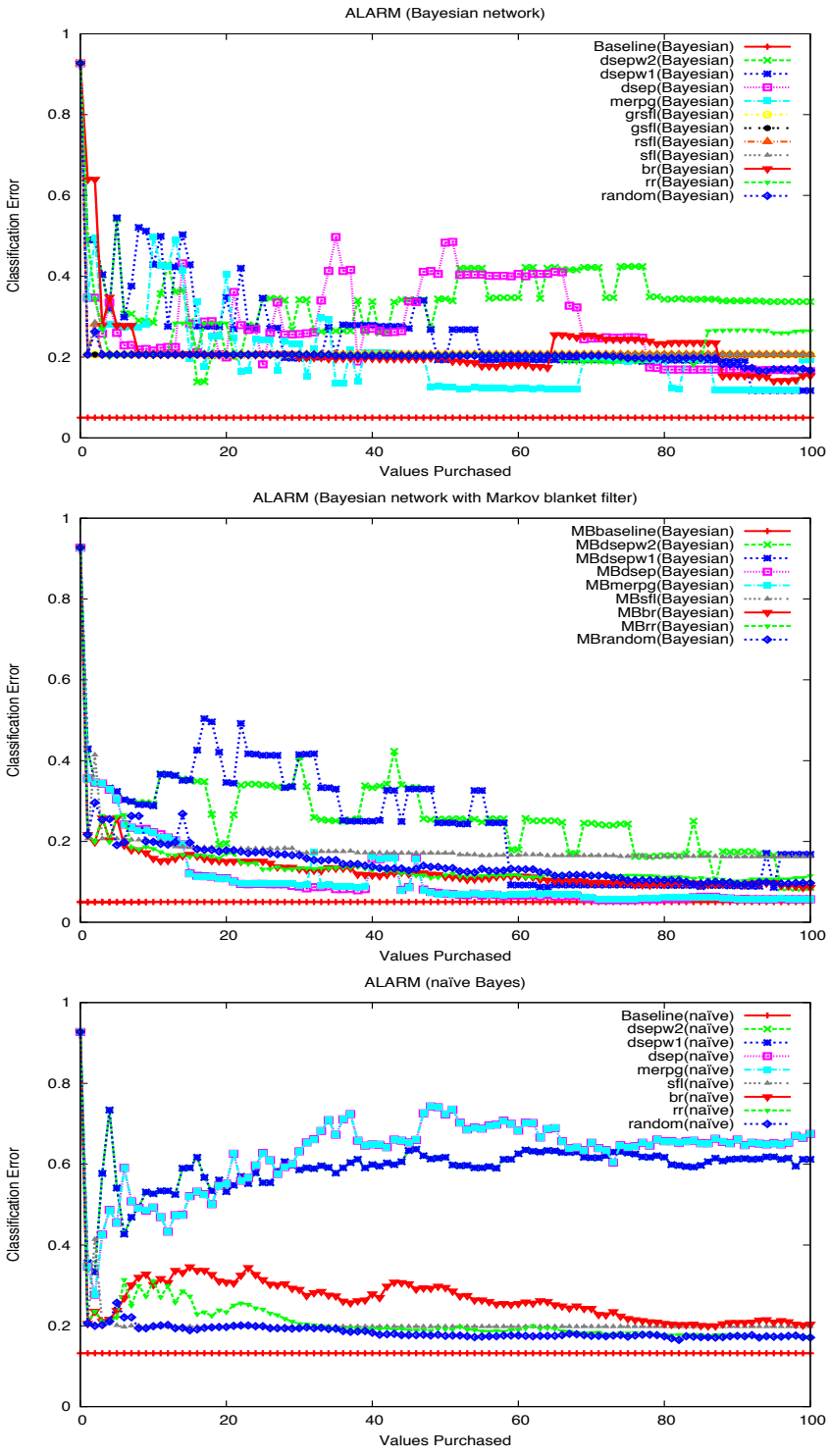


Figure 5.18: Comparing all the algorithms on Bayesian network, on Bayesian network with Markov blanket, and on naïve Bayes on ALARM from Norsys Net Library (2011).

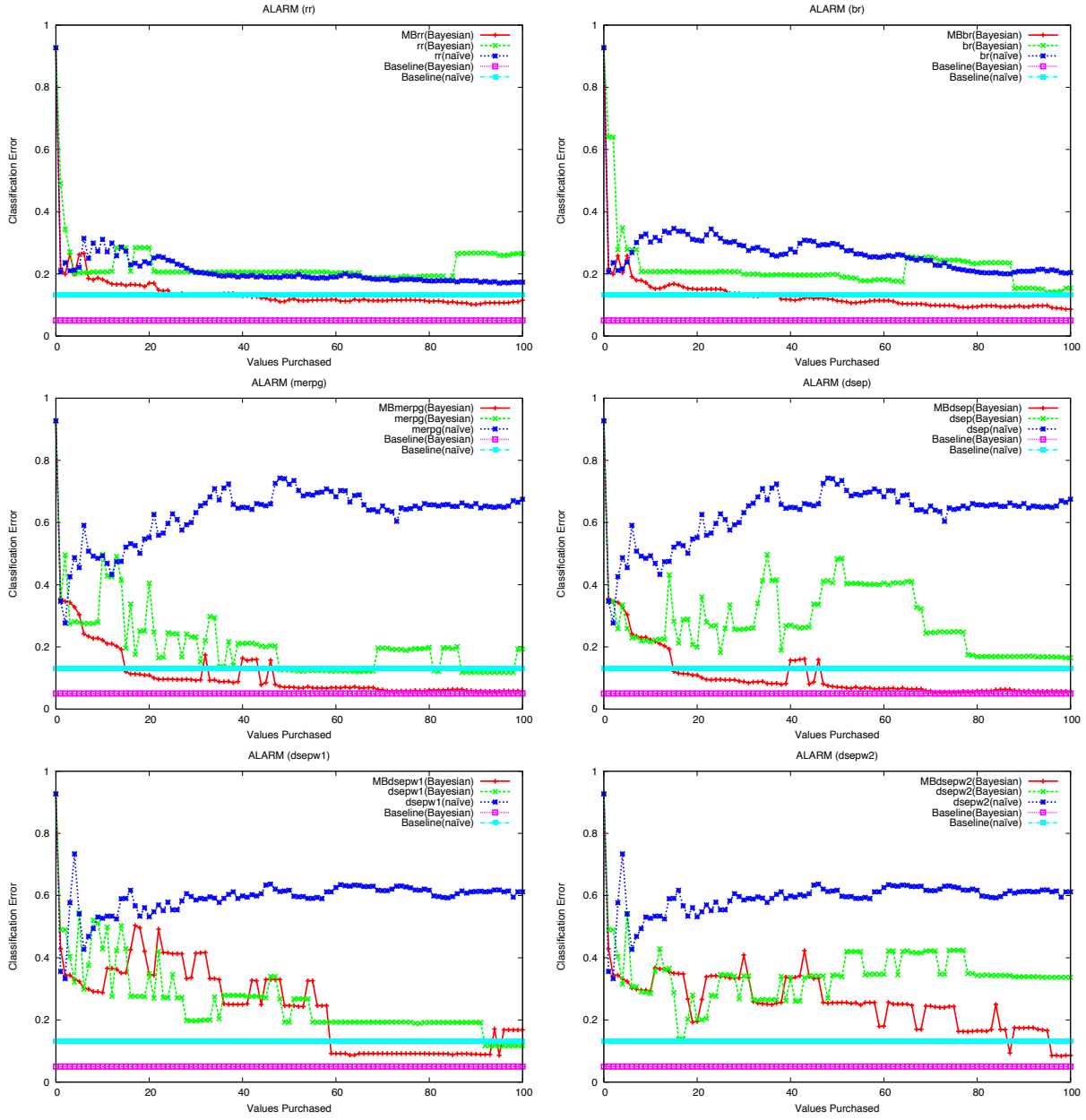


Figure 5.19: Comparing rr, br, merpg, dsep, dsepw1, and dsepw2 on Bayesian network with Markov blanket filter, on Bayesian network, and on naïve Bayes on ALARM from Norsys Net Library (2011).

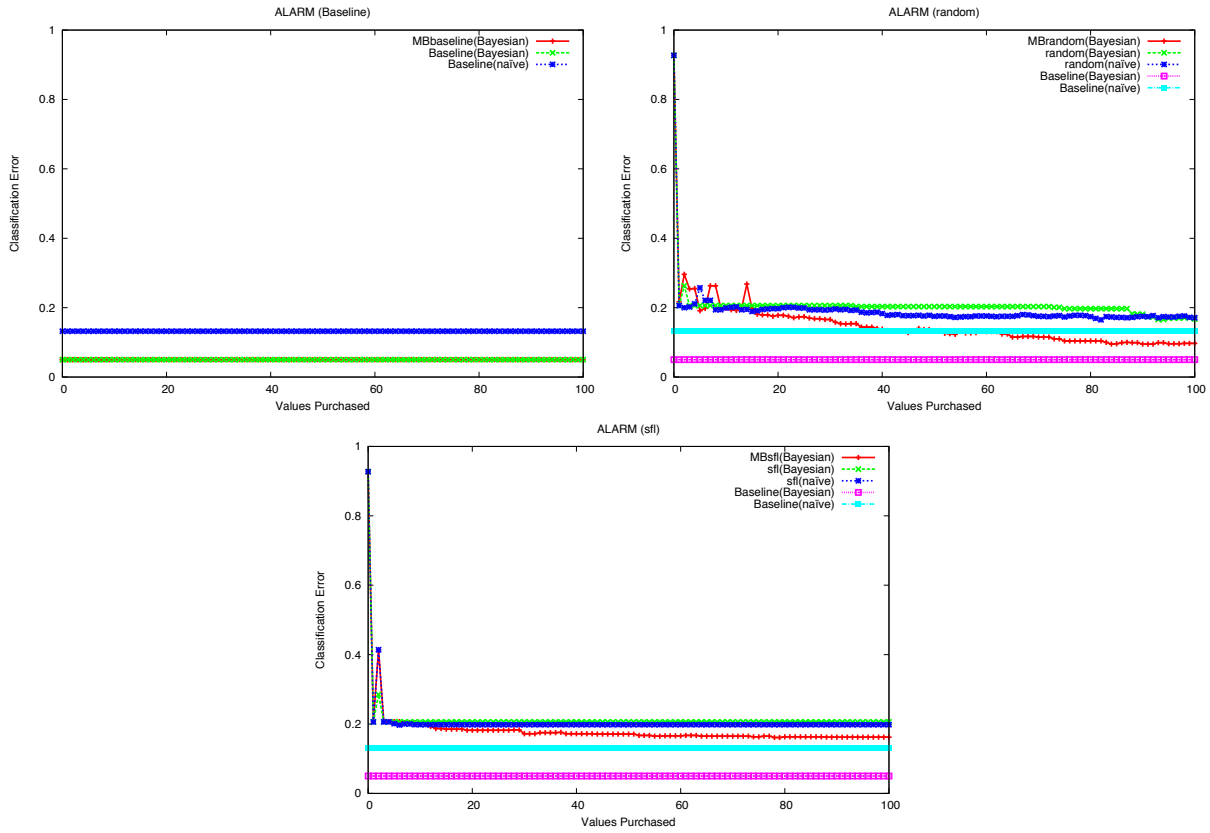


Figure 5.20: Comparing Baseline, random and sfl on Bayesian network with Markov blanket filter, on Bayesian network, and on naïve Bayes on ALARM from Norsys Net Library (2011).

Table 5.11: Mean running time over 10 runs of algorithms on data set ALARM (37 nodes, 46 edges, 1000 instances). Budget=100.

Algorithm	of naïve Bayes	of Bayesian Network	of Bayesian Network on MB
random	1 second	5 minutes	3 minutes
rr	1 second	5 minutes	3 minutes
br	1 second	240 seconds	180 seconds
merpg	22 minutes	60 minutes	24 minutes
dsep	24 minutes	60 minutes	25 minutes
dsepw1	24 minutes	60 minutes	25 minutes
dsepw2	24 minutes	60 minutes	25 minutes
sfl	10 hours	16 hours	10 hours

5.6.6 Discussion

In summary, which algorithm perform the best over the five data sets? To answer this question, we combine these 5 Wilcoxon-based results together and show the result in Table 5.12. From this table, we can see that MBdsep (BN) overall outperformed other algorithms and sfl (NB) is outperformed by other algorithms. The reason why MBdsep (BN) outperformed other algorithms is that it can identify more important features for instances to purchase. MBdsep (BN) is much better than all other algorithms, with MBdsepw2 (BN) second with only 84% as many wins.

In the following, we are going to answer several questions.

1. *Question*: **What is the conclusion on the Mean running time over 10 runs of the algorithms?**

   Answer: For Mean running time over 10 runs of the algorithms on each Bayesian network from Norsys Net Library (2011) with generated instances, SFL is most time-consuming, followed by MERPG related algorithms, and Random, rr, and br is least time-consuming. MB*algo* saves time compared to *algo* by up to 95% in our experiment.

2. *Question*: **What is the conclusion of baselines of the data sets whose instances are generated according to the Bayesian network from Norsys Net Library (2011)?**

   Answer: The baseline on Bayesian network with Markov blanket filter is the same as the baseline on Bayesian network. The reason is that the prediction of the label node can be decided by only the nodes in the Markov blanket. Except Animals whose baseline on naïve Bayes is the same as its baseline on Bayesian network, the baseline (classification error) on naïve Bayes is higher than the baseline on Bayesian network. Therefore, most Bayesian networks in our experiments are better classifiers than their

Table 5.12: Result of Wilcoxon signed rank test between the left side algorithm and the top side algorithm. "++0–" ("−") indicates that the left side algorithm reaches reaches a significantly higher classification accuracy than, a significantly higher classification accuracy than, has no significant difference with, a significantly lower classification accuracy than, a significantly lower classification accuracy than the top side algorithm for data sets **Animals, Car Diagnosis 2, Chest Clinic, Poya Ganga,** and **ALARM** at budgets 100, 100, 40, 30,and 100 at a confidence level $p < 0.05$.

	random (NB)	rr (NB)	br (NB)	merpg (NB)	dsep (NB)	dsepw1 (NB)	dsepw2 (NB)	sfl (NB)	random (BN)	rr (BN)	br (BN)	merpg (BN)	dsep (BN)	dsepw1 (BN)	dsepw2 (BN)	sfl (BN)	MBrandom (BN)	MBrr (BN)	MBbr (BN)	MBmerpg (BN)	MBdsep (BN)	MBdsepw1 (BN)	MBdsepw2 (BN)	MBsfl (BN)	Wins	Losses
random (NB)	000 00	000 00	00+ 00	−0+ 0+	−00 0+	−00 0+	−00 0+	0+− 0+	+0− 00	00− 00	00− 00	−−0 00	−0− 0−	−00 0−	−00 0−	00− 0+	0−− 0−	0−− 00	0−− 0−	−−0 0−	−−0 0−	−00 0−	−−0 0−	0+− 00	11	38
rr (NB)	000 00	000 00	00+ 0+	−0+ 0+	−00 0+	−00 ++	−00 ++	0+− 0+	00− 00	00− 00	00− 00	−−0 00	−0− 0−	−00 0−	−00 0−	0+− 0+	0−− 0−	0−− 00	0−− 0−	−−0 0−	−−0 0−	−00 0−	−00 0−	0+− 00	14	37
br (NB)	00− 00	00− 0−	000 00	−−0 0+	−−− 0+	−00 0+	−00 0+	0+− 0+	00− 0−	00− 00	00− 00	−−0 0−	−−− 0−	−−0 0−	−00 0−	00− 00	0−− 0−	0−− 00	0−− 0−	−00 0−	−−0 −−	−0− 0−	−−0 −−	0+− 00	7	50
merpg (NB)	+0− 0−	+0− 0−	++0 0−	000 00	000 00	000 00	000 00	++− 0+	+0− 0−	+0− 00	00− +−	0−− 00	0−− 00	00− −−	00− 00	0+− 0−	+−− 0−	0−− 0−	+−− 0−	00− 0−	0−− 0−	00− 00	00− 0−	++− 00	15	39
dsep (NB)	+00 0−	+00 0−	+++ 0−	000 00	000 00	000 00	000 00	++− 0+	+00 0−	+00 00	000 +−	0−0 00	0−− 00	000 −−	000 00	0+− 0−	+−− 0−	0−− 0−	+−− 0−	000 0−	0−0 0−	000 00	000 0−	++− 00	16	27
dsepw1 (NB)	+00 0−	+00 −−	+00 0−	000 00	000 00	000 00	000 00	++− 0+	+00 0−	+0− 0−	00− 0−	0−− −0	0−− 00	00− 0−	00− 00	0+− −−	+−− 0−	00− 0−	+−− 0−	00− 0−	0−− 0−	−0− 0−	−−− −−	++− 00	13	43
dsepw2 (NB)	+00 0−	+00 −−	+00 0−	000 00	000 00	000 00	000 00	++− 0+	+0− 0−	+00 0−	00− 0−	0−− −0	0−− 00	00− 0−	00− 00	0+− −−	+−− 0−	00− 0−	+−− 0−	00− 0−	0−− 0−	−0− 0−	−−− −−	++− 00	13	43
sfl (NB)	0−+ 0−	0−+ 0−	0−+ 0−	−−+ 0−	−−+ 0−	−−+ 0−	−−+ 0−	000 00	0−0 0−	0−0 0−	0−0 0−	0−0 0−	0−0 0−	0−0 0−	0−0 0−	0−0 0−	0−− 0−	0−− 0−	0−− 0−	0−0 0−	−−0 0−	−−0 0−	−−0 0−	000 00	7	54
random (BN)	−0+ 00	00+ 00	00+ 0+	−0+ 0+	−00 0+	−00 0+	−0+ 0+	0+0 0+	000 00	0−− 00	000 +0	−00 0−	−−0 0−	−00 0−	−00 0−	000 −+	0−− 0−	00− 0−	0−0 0−	−00 0−	−−0 0−	−00 0−	−−0 0−	0+0 00	15	34
rr (BN)	00+ 00	00+ 00	00+ 00	−0+ 00	−00 00	−0+ 0+	−00 0+	0+0 0+	0++ 00	000 00	000 00	−00 0−	−00 0−	−00 00	−00 0−	000 00	000 00	0−0 00	0−+ 0−	−00 0−	−−0 0−	−00 0−	−00 0−	0+0 00	13	23
br (BN)	00+ 00	00+ 00	00+ 00	00+ −+	000 −+	00+ 0+	00+ 0+	0+0 0+	000 −0	000 00	000 00	0−0 00	000 00	−00 00	−00 00	000 −0	0−− 00	00− 00	0−0 00	0−0 0−	0−0 −−	0−0 00	−00 0−	0+0 00	13	19
merpg (BN)	++0 00	++0 00	++0 0+	0++ 00	0+0 00	0++ +0	0++ +0	0+0 0+	+00 0+	+00 0+	0+0 00	000 00	000 00	000 00	000 00	0+0 0+	+00 00	000 00	+00 0−	000 00	000 0−	−00 0−	−00 0−	0+0 00	28	6
dsep (BN)	+0+ 0+	+0+ 0+	+++ 0+	0++ 00	0++ 00	0++ 00	0++ 00	0+0 0+	++0 0+	+00 0+	000 00	000 00	000 00	000 00	000 00	0+0 0+	+−0 00	000 00	+00 00	000 0−	000 0−	−00 00	−00 0−	0+0 00	30	6
dsepw1 (BN)	+00 0+	+00 0+	++0 0+	00+ ++	000 ++	00+ 0+	00+ 0+	0+0 0+	+00 0+	+00 00	+00 00	000 00	000 00	000 00	00− 00	0+0 0+	+−− 00	+−− 00	+−0 0−	0−0 0−	0−0 0−	000 00	000 00	++0 00	29	11
dsepw2 (BN)	+00 0+	+00 0+	+00 0+	00+ 00	000 00	00+ 00	00+ 00	0+0 0+	+00 0+	+00 0+	+00 00	000 00	000 00	00+ 00	000 00	0+0 0+	+00 00	+0− 00	+00 0−	000 0−	000 0−	000 00	000 00	0+0 00	23	4
sfl (BN)	00+ 0−	0−+ 0−	00+ 00	0−+ 0+	0−+ 0+	0−+ 0+	0−+ 0+	0+0 0+	000 +−	000 00	000 +0	0−0 0−	0−0 0−	0−0 0−	0−0 0−	000 00	0−− +−	0−− 0−	0−− 0−	0−0 0−	0−0 0−	000 0−	0−0 0−	0+0 00	17	32
MBrandom (BN)	0++ 0+	0++ 0+	0++ 0+	−++ 0+	−++ 0+	−++ 0+	−++ 0+	0++ 0+	0++ 0+	000 00	0+0 00	−00 00	−+0 00	−++ 00	−00 00	0++ −+	000 00	000 00	00+ 00	000 00	−00 −0	−00 00	−00 00	0++ 00	37	13
MBrr (BN)	0++ 00	0++ 00	0++ 00	0++ 0+	0++ 0+	00+ 0+	00+ 0+	0++ 0+	00+ 0+	0+0 00	00+ 00	000 00	000 00	−++ 00	−0+ 00	0++ 0+	000 00	000 00	00+ 00	000 0−	000 0−	−00 00	−00 0−	0++ 00	32	7
MBbr (BN)	0++ 0+	0++ 0+	0++ 0+	−++ 0+	−++ 0+	−++ 0+	−++ 0+	0++ 0+	0+0 0+	0+− 0+	0+0 00	−00 0+	−00 00	−+0 0+	−00 0+	0++ 0+	00− 00	00− 00	000 00	−00 00	−00 0−	−00 00	−00 00	0++ 00	38	16
MBmerpg (BN)	++0 0+	++0 0+	+00 0+	00+ 0+	000 0+	00+ 0+	00+ 0+	0+0 0+	+00 0+	+00 0+	0+0 0+	000 00	000 0+	0+0 0+	000 0+	0+0 0+	000 00	000 0+	+00 00	000 00	000 0−	000 00	000 0+	0+0 00	33	1
MBdsep (BN)	++0 0+	++0 0+	++0 ++	0++ 0+	0+0 0+	0++ 0+	0++ 0+	++0 0+	++0 0+	++0 0+	0+0 ++	000 0+	000 0+	0+0 0+	000 0+	0+0 0+	+00 +0	000 0+	+00 0+	000 0+	000 00	000 00	000 0+	++0 +0	50	0
MBdsepw1 (BN)	+00 0+	+00 0+	+0+ 0+	00+ 00	000 00	+0+ 0+	+0+ 0+	++0 0+	+00 0+	+00 0+	+00 00	+00 0+	+00 00	000 00	000 00	000 0+	+00 00	+00 00	+00 00	000 00	000 00	000 00	000 −0	++0 00	31	1
MBdsepw2 (BN)	++0 0+	+00 0+	++0 ++	00+ 0+	000 0+	+++ ++	+++ ++	++0 0+	++0 0+	+00 0+	+00 0+	+00 00	+00 00	000 00	000 00	0+0 0+	+00 00	+00 00	+00 00	000 0−	000 0−	000 +0	000 00	++0 00	42	2
MBsfl (BN)	0−+ 00	0−+ 00	0−+ 00	−−+ 00	−−+ 00	−−+ 00	−−+ 00	000 00	0−0 00	0−0 00	0−0 00	0−0 00	0−0 00	−−0 00	0−0 00	0−0 00	0−− 00	0−− 00	0−− 00	0−0 00	−−0 −0	−−0 00	−−0 00	000 00	7	34

Table 5.13: Does an algorithm learning **Bayesian networks** get significant improvement over the algorithm learning naive Bayes? "+" indicates a significant improvement, "−" indicates a significant deterioration, "0" indicates no significant change.

MB size/total size	Animals $b = 100$ 3/7	CarDiagnosis2 $b = 100$ 5/18	ChestClinic $b = 40$ 3/37	Poya_Ganga $b = 30$ 2/60	ALARM $b = 100$ 5/8
random	−	0	+	0	0
rr	0	0	+	0	0
br	0	0	+	0	0
merpg	0	+	+	0	0
dsep	0	+	+	0	0
dsepw1	0	0	+	0	+
dsepw2	0	0	+	0	0

Table 5.14: Does an algorithm learning Bayesian networks on a **Markov blanket** filter get significant improvement over the algorithm learning Bayesian networks? "+" indicates a significant improvement, "−" indicates a significant deterioration, "0" indicates no significant change.

MB size/total size	Animals $b = 100$ 3/7	CarDiagnosis2 $b = 100$ 5/18	ChestClinic $b = 40$ 3/37	Poya_Ganga $b = 30$ 2/60	ALARM $b = 100$ 5/8
random	0	+	+	0	+
rr	0	+	0	0	0
br	0	+	0	0	0
merpg	0	0	0	0	0
dsep	0	0	0	0	+
dsepw1	0	0	0	0	0
dsepw2	0	0	0	0	0
sfl	0	−	0	0	0

Table 5.15: Does **MERPGDSEP** get significant improvement over MERPG? "+" indicates a significant improvement, "−" indicates a significant deterioration, "0" indicates no significant change.

	Animals $b = 100$	CarDiagnosis2 $b = 100$	ChestClinic $b = 40$	Poya_Ganga $b = 30$	ALARM $b = 100$
of NB	0	0	0	0	0
of BN	0	0	0	0	0
on MB of BN	0	0	0	0	+

Table 5.16: Does **MERPGDSEPW1** get significant improvement over MERPG? "+" indicates a significant improvement, "−" indicates a significant deterioration, "0" indicates no significant change.

	Animals $b = 100$	CarDiagnosis2 $b = 100$	ChestClinic $b = 40$	Poya_Ganga $b = 30$	ALARM $b = 100$
of NB	0	0	0	0	0
of BN	0	0	0	0	0
on MB of BN	0	0	0	0	0

Table 5.17: Does **MERPGDSEPW2** get significant improvement over MERPG? "+" indicates a significant improvement, "−" indicates a significant deterioration, "0" indicates no significant change.

	Animals $b = 100$	CarDiagnosis2 $b = 100$	ChestClinic $b = 40$	Poya_Ganga $b = 30$	ALARM $b = 100$
of NB	0	0	0	0	0
of BN	0	0	0	0	0
on MB of BN	0	0	0	0	−

corresponding naïve Bayes classifiers.

3. *Question*: **Does an algorithm on Bayesian networks get significant improvement over the corresponding algorithm on naïve Bayes?**

   Answer: Yes. Please refer to Table 5.13. From Table 5.12, we can see that, overall, *algo* on naïve Bayes has more wins and fewer losses compared to *algo* on Bayesian network. From Tables 5.2, 5.4, 5.6, 5.8, and 5.10, we can see that for each data set, it is almost always true that *algo* on naïve Bayes has more wins and fewer losses compared to *algo* on Bayesian network. For Animals, this difference is not obvious because its naïve Bayes and Bayesian network are both good representations of the dependencies of the features of the training data. For Poya Ganga, *algo* on Bayesian network does not outperform *algo* on naïve Bayes because Poya Ganga is a difficult learning problem for classification and the provided structure of the Bayesian network, although better than naïve Bayes, is not good enough (only slightly better than random guessing) to

help the choice of (instance, feature) pairs.

4. *Question*: **Why is there a significant improvement of some *algo* on Bayesian network over *algo* on naïve Bayes? Is it because of the improved base learner of the Bayesian network, or because of the better choice of (instance, feature) pairs of improved objective functions related to the improved structure of the Bayesian network?**

   Answer: It is really hard to answer. Our answer is that the improvement may come from both factors. We believe that the improved base learner of Bayesian network makes the learned classifier more accurate. From Table 5.12, we can see that random (BN) and rr (BN) have more wins over random (NB) and rr (NB) and have fewer losses to random (NB) and rr (NB), however, since we did not control the experiments so that for each purchase, algorithms random (BN) and random (NB), or rr (BN) and rr (NB) choose same (instance, feature) pairs, we cannot reach the conclusion that the improved base learner makes the learned classifier more accurate. Also, we believe the improved objective functions related to the improved structure of the Bayesian network helped the choice of the (instance, feature) pairs, such as br, merpg, dsep, dsepw1 and dsepw2. However, for the same reason, we cannot reach the conclusion. We will further explore this question by doing controlled experiments that choose (instance, feature) pairs by *algo* on naïve Bayes (or Bayesian network), while learning different classifiers (naïve Bayes and Bayesian network); and choosing (instance, feature) pairs by *algo* on Bayesian network and naive Bayes, while learning the same classifier (naïve Bayes or Bayesian network) to identify where this improvement comes from.

5. *Question*: **Does an algorithm on Bayesian networks with Markov blanket filter get significant improvement over the algorithm on Bayesian networks without Markov blanket filter?**

Answer: Yes. Please refer to Table 5.14. For networks Car Diagnosis 2, Chest Clinic, and ALARM, some algorithms, e.g. Car Diagnosis 2 and Chest Clinic showed a significant improvement in performance when using the Markov blanket filter versus when not. There is only one minus in Table 5.14, so there really is no harm in using MB. From Table 5.12, we can see MB*algo* (BN) overall has more wins and fewer losses compared to *algo* (BN). From Tables 5.2, 5.4, 5.6, 5.8, and 5.10, we can see that for each data set, it is almost true that MB*algo* (BN) on has more wins and fewer losses compared to *algo* (BN). From Tables 5.3, 5.5, 5.7, 5.9, and 5.11, we can see MB*algo* always saves up to 95% time compared to *algo*. In summary, using Markov blanket filter can both saving time and improving performance.

6. *Question*: **Does MERPGDSEP get a significant improvement over MERPG?**

   Answer: Yes. From Table 5.15, we can see that MBdsep outperform MBmerpg for ALARM. From Table 5.12, we can see MBdsep (BN) versus MBmerpg (BN), dsep (BN) versus merpg (BN), and dsep (NB) versus merpg (NB) has an overall advantage.

7. *Question*: **Does MERPGDSEPW1 get a significant improvement over MERPG?**

   Answer: No. Please refer to Table 5.16.

8. *Question*: **Does MERPGDSEPW2 get a significant improvement over MERPG?**

   Answer: No. On the contrary, it gets significant worse over MERPG in 1 case. Please refer to Table 5.17.

9. *Question*: **Overall, which algorithm is best?**

   Answer: MBdsep, please refer to Table 5.12, Overall, MBdsep has the maximum number of wins to other algorithms and has no losses to other algorithms.

5.7 Conclusions & Future Work

Overall, we have the following results. (1) For Mean running time over 10 runs of the algorithms on each Bayesian network from Norsys Net Library (2011) with generated instances, SFL is most time-consuming, followed by MERPG related algorithms, and Random, rr, and br is least time-consuming. MB*algo* saves up to 95% time compared to *algo*. (2) The baseline on Bayesian network with Markov blanket filter is the same as the baseline on Bayesian network. Usually the baseline (classification error) on naïve Bayes is higher than the baseline on Bayesian network. (3) *algo* on Bayesian network has more wins and fewer losses compared to *algo* on naïve Bayes, therefore, we conclude that *algo* on naïve Bayes is outperformed by *algo* on Bayesian network. Whether the improvement comes from the improved structure of the Bayesian network, or the better choice of (instance, feature) pairs of improved objective functions related to the improved structure of the Bayesian network, we are not clear and further exploration is our future work. (4) *algo* on Bayesian network with Markov blanket filter get significant improvement over *algo* on Bayesian network without Markov blanket filter in terms of lower classification error rates and run time. (5) MERPGDSEP get significant improvement over MERPG. But MERPGDSEPW1 and MERPGDSEPW2 do not get significant improvement over MERPG. (6) Overall, MBdsep on Bayesian network has the maximum number of wins to other algorithms and has no losses to other algorithms, therefore we conclude that **MBdsep (BN)** is the best algorithm. SFL on naïve Bayes has the minimum number of wins to other algorithms and has the maximum losses to other algorithms, therefore we conclude that SFL (NB) is the worst algorithm.

For simplicity, we assume that the cost of each feature is the same. One direction of our future work is to consider budgeted learning under the setting that each feature has a varied cost. Another direction is to explore whether the improvement of algorithms on Bayesian network comes from the improved structure of Bayesian network or the improved objec-

tive functions because of the improved structure of Bayesian network, by doing controlled experiments that choosing (instance, feature) pairs by *algo* on naïve Bayes (or Bayesian network) while learning different classifiers (naïve Bayes and Bayesian network); and choosing (instance, feature) pairs by *algo* on Bayesian network and naive Bayes while learning same classifier (naïve Bayes or Bayesian network) to identify where this improvement comes from. The third direction of our future research is to figure in which condition which algorithms are more appropriate to be used.

Chapter 6

Conclusion & Future Work

For our proposed algorithms for active learning from multiple noisy labelers with varied costs, IEAdjCost and wIEAdjCost, they shift focus from instance selection in classical active learning to labeler selection and ground truth estimation. Compared with existing algorithms in the literature, naïve Repeated and IEThresh, these two algorithms consider both factors of labeler cost and accuracy, and they rank each labeler according to its adjusted cost: the lower adjusted cost a labeler has, the higher rank this labeler has. Also, for the ground truth estimation for instance, IEAdjCost used majority voting over the chosen labelers, while wIEAdjCost improved this by using weighted majority voting over the chosen labelers, that is, the higher estimated accuracy a labeler is, the higher weight it has.

We tested our algorithms and existing algorithms on six UCI data sets and data from Amazon Mechanical Turk, and showed that our new algorithms performed at least as well, and often better than, algorithm IEThresh from the literature and the naïve approach Repeated. Further, our even newer algorithm wIEAdjCost improves on our previous algorithm IEAdjCost by utilizing the notion of weight to significantly reduce final cost as well as time. This was especially true when there were many highly accurate labelers available, and/or when high combined accuracy was needed. In summary, IEAdjCost significantly outperforms

Repeated and IEAdjCost. wIEAdjCost significantly outperforms IEAdjCost by saving both time and cost.

These two algorithms are based on the assumption that the labelers are independent of each other; in reality, it is possible that human labelers are correlated. Another assumption is that one labeler's answers to individual instances are independent across instances; in reality, it is possible that a labeler's answers are correlated across several instances. Our future work will consider these conditions. We will look at developing more sophisticated methods to learn when to switch from Phase 1 to Phase 2, perhaps by dynamically tuning δ and λ. We also plan to investigate if our ideas could be applied to solve the cost-sensitive feature acquisition problem. Finally, it would be interesting to consider the possibility that labelers' accuracies could be affected by exposure to labeled and unlabeled instances, as what is seen to happen to human labelers.

For our proposed algorithms for budgeted learning, we presented several new algorithms for choosing which features of which example to purchase, based on the multi-armed bandit model. We also proposed several algorithms based on the second-order statistic. In line with these algorithms, we also described variations on how to select the row (instance) to purchase an attribute of, selecting the row with most uncertainty in the current model. We proposed two row selectors, entropy (EN) and "Error-Correction" (EC). EN row selector chooses an instance that maximizes the entropy of the posterior class distribution. EC row selector chooses an instance that is most wrongly predicted by current model. We tested our algorithms on 6 data sets from UCI machine learning repository. Our experiments show that EC stands out for Random and EN stands out for RBR2. Also these two row selectors have advantage of some algorithms over other row selectors. When comparing algorithms with same row selector, ABR2 is one of the top 2 algorithms that performs better than other algorithms with the same row selector. When comparing all algorithms, ABR2 with any row selector has an advantage over existing algorithms. Also performing well are WBR2 and

Exp3C with EC row selector and FEL with uniform random row selector.

Our work mostly lies in experiments and evaluations. Most of the recent theoretical work (Bubeck et al., 2008; Antos et al., 2008; Goel et al., 2009) aimed to tackle this problem through the study of budgeted bandit models using approximation analysis. This line of research and the technique developed can be extremely useful for understanding the hardness of budgeted machine learning, however we feel there seems to be a gap between these results and their applications in real world budgeted classification problems. The work of Li (2009) seemed to be one step closer. Maybe if the loss of the parameters is somehow linked to the loss of the prediction for a naïve Bayes classifier, then their results can be directly applied to budgeted classification problems as well. Part of our future work is developing some learning-theoretic results (e.g. PAC-style results) for the budgeted classification problem. In general, we are interested in identifying conditions where the querying complexity (i.e. the amount of attribute values needed) is strictly upper bounded by the sampling complexity in a budgeted PAC learning problem.

We also studied budgeted learning of Bayesian networks when the features of the training data are represented by a known correct Bayesian network. We proposed several algorithms that not only took advantage of the learned Bayesian network (just like Tong and Koller (2001a) and Li et al. (2010)), but also took advantage of the known labels as well as the structure of the Bayesian network itself (such as d-separation or considering a Markov blanet of the label node). Our experiments were done on 5 data sets from Norsys Net Library (Norsys Software Corp., 2011) and our conclusion based on our experimental results is as follows: (1) For mean running time of the algorithms on each Bayesian network from Norsys Net Library (2011) with generated instances, SFL is most time-consuming, followed by MERPG related algorithms, and Random, rr, and br are the least time-consuming. MB*algo* saves up to 95% time compared to *algo*. overUsually the baseline (classification error) on naïve Bayes is higher than the baseline on Bayesian network. (3) *algo* on Bayesian

network has more wins and fewer losses compared to *algo* on naïve Bayes, therefore, we conclude that *algo* on naïve Bayes is outperformed by *algo* on Bayesian network. Whether the improvement comes from the improved structure of the Bayesian network, or the better choice of (instance, feature) pairs of improved objective functions related to the improved structure of the Bayesian network, we are not clear and further exploration is our future work. (4) *algo* on Bayesian network with Markov blanket filter get significant improvement over *algo* on Bayesian network without Markov blanket filter in terms of lower classification error rates and run time. (5) MERPGDSEP get significant improvement over MERPG. But MERPGDSEPW1 and MERPGDSEPW2 does not get significant improvement over MERPG. (6) Overall, MBdsep on Bayesian network has the maximum number of wins to other algorithms and has no losses to other algorithms, therefore we conclude that **MBdsep (BN)** is the best algorithm. SFL on naïve Bayes has the minimum number of wins to other algorithms and has the maximum losses to other algorithms, therefore we conclude that SFL (NB) is the worst algorithm.

For simplicity, we assume that the cost of each feature is the same. One direction of our future work is to consider budgeted learning under the setting that each feature has a varied cost. Another direction is to explore whether the improvement of algorithms on Bayesian network comes from the improved structure of Bayesian network or the improved objective functions because of the improved structure of Bayesian network, by doing controlled experiments that choosing (instance, feature) pairs by *algo* on naïve Bayes (or Bayesian network) while learning different classifiers (naïve Bayes and Bayesian network); and choosing (instance, feature) pairs by *algo* on Bayesian network and naive Bayes while learning same classifier (naïve Bayes or Bayesian network) to identify where this improvement comes from. The third direction of our future research is to figure in which condition which algorithms are more appropriate to be used.

Appendix A

The structures of Two Large Bayesian Networks

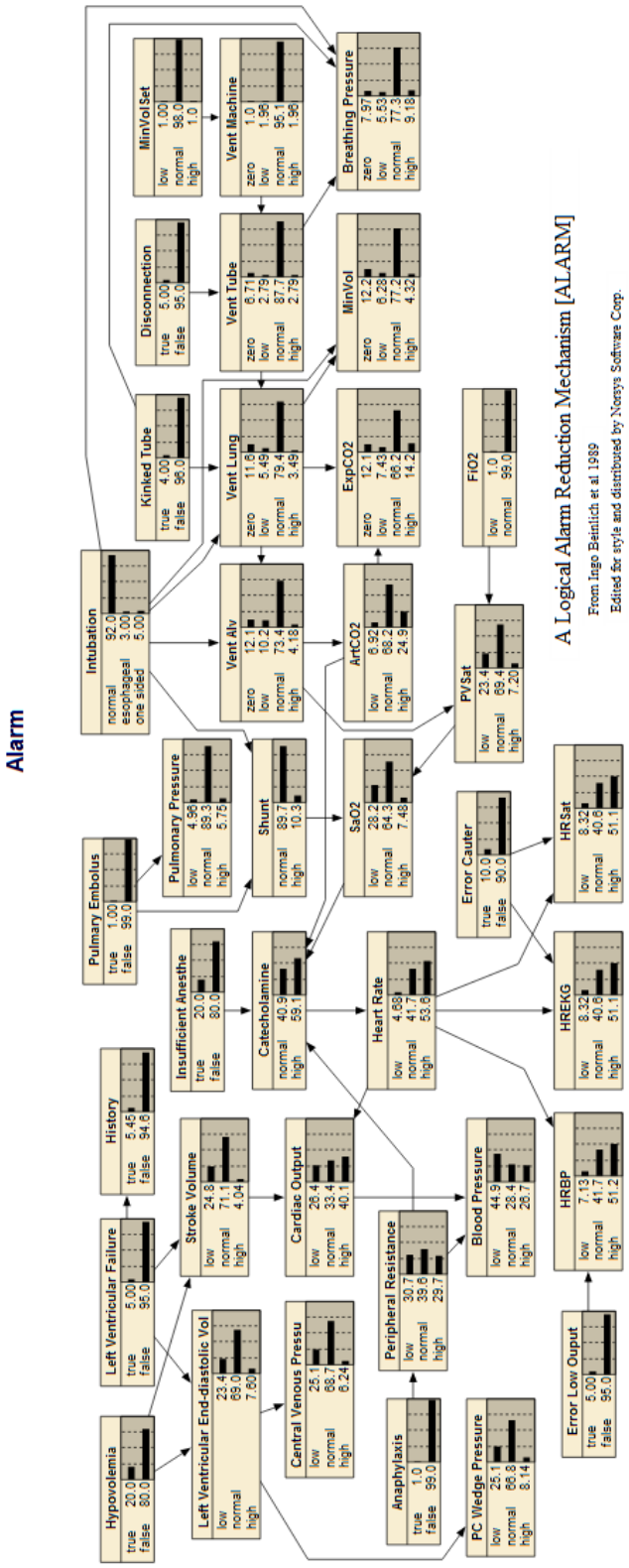


Figure A.1: A complete structure of the Bayesian network of ALARM. The figure is excerpted from Norsys Net Library (Norsys Software Corp., 2011).

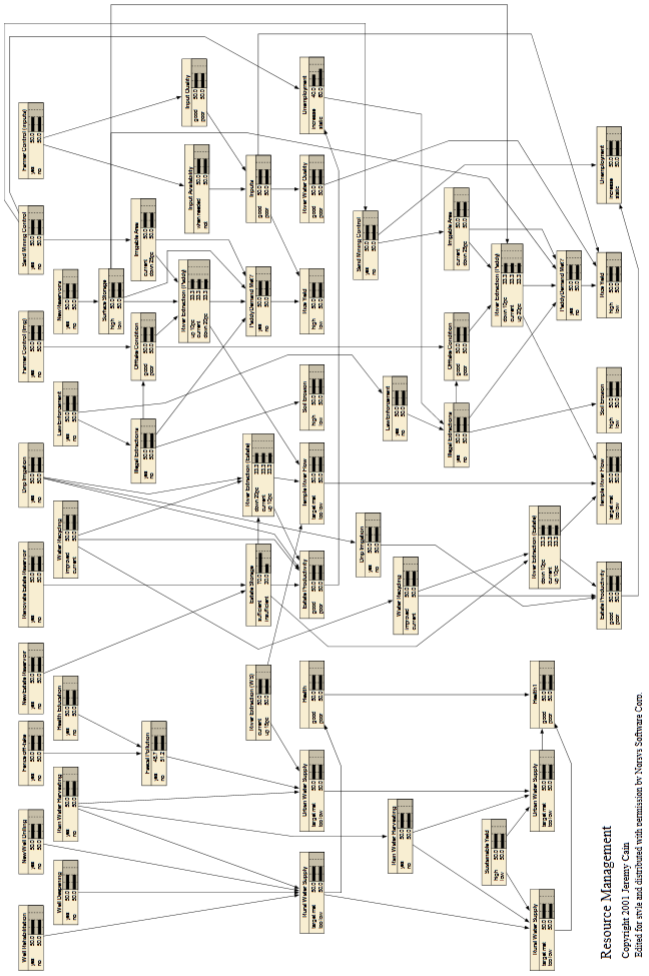


Figure A.2: A complete structure of the Bayesian network of Water Resource Management in Poya Ganga. The figure is excerpted from Norsys Net Library (Norsys Software Corp., 2011).

Bibliography

Naoki Abe and Hiroshi Mamitsuka. Query Learning Strategies Using Boosting and Bagging. In *Proceedings of the 15th International Conference on Machine Learning*, pages 1–10, 1998.

Amazon Mechanical Turk. http://mturk.com, 2005.

András Antos, Varun Grover, and Csaba Szepesvári. Active Learning in Multi-armed Bandits. In *Proceedings of the 19th international conference on Algorithmic Learning Theory*, pages 287–302, Berlin, Heidelberg, 2008. Springer-Verlag.

Arthur Asuncion and David Newman. UCI Machine Learning Repository, `http://www.ics.uci.edu/~mlearn/MLRepository.html`, 2009.

Peter Auer, Nicolò Cesa-Bianchi, Paul Fischer, and Lehrstuhl Informatik. Finite-time Analysis of the Multi-Armed Bandit Problem. *Machine learning*, 47(2):235–256, 2002a.

Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. The Non-stocastic Multi-armed Bandit Problem. *SIAM Journal on Computing*, 32(1):48–77, 2002b.

Sebastien Bubeck, Remi Munos, and Gilles Stoltz. Pure Exploration for Multi-Armed Bandit Problems, 2008.

Colin Campbell, Nello Cristianini, and Alex Smola. Query Learning with Large Margin Classifiers. In *Proceedings of the 17th International Conference on Machine Learning*, pages 111–118, 2000.

Rich Caruana and Alexandru Niculescu-Mizil. An Empirical Comparison of Supervised Learning Algorithms. In *Proceedings of the 23rd international conference on Machine learning*, pages 161–168, New York, NY, USA, 2006. ACM.

Gregory F. Cooper. Probabilistic Inference Using Belief Networks Is NP-Hard. Technical Report KSL-87-27, Knowledge Systems, AI Laboratory, 1987.

Robert G. Cowell, Dawid A. Philip, Steffen L. Lauritzen, and David J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer, 1999.

Matt Culver, Deng Kun, and Stephen Scott. Active Learning to Maximize Area Under the ROC Curve. In *Proceedings of the 6th IEEE International Conference on Data Mining*, pages 149–158, December 2006.

Philip Dawid, , and Allan Skene. Maximum Likelihood Estimation of Observer Error-Rates Using the EM Algorithm. *Journal of the Royal Statistical Society, Series C (Applied Statistics)*, 28(1):20–28, 1979. ISSN 00359254.

Arthur P. Dempster, Nan Laird, and Donald Bruce Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

Kun Deng, Chris Bourke, Stephen Scott, Julie Sunderman, and Yaling Zheng. Bandit-Based Algorithms for Budgeted Learning. In *Proceedings of the 7th IEEE International Conference on Data Mining*, pages 463–468, Washington, DC, USA, 2007. IEEE Computer Society.

Marie desJardins, James MacGlashan, and Kiri Wagstaff. Confidence-Based Feature Acquisition to Minimize Training and Test Costs. In *Proceedings of the SIAM Conference on Data Mining*, pages 514–524, 2010.

Pinar Donmez and Jamie Garbonell. Proactive Learning: Cost-Sensitive Active Learning with Multiple Incomplete Oracles. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, pages 619–628, 2008.

Pinar Donmez, Jaime G. Carbonell, and Jeff Schneider. Efficiently Learning the Accuracy of Labeling Sources for Selective Sampling. In *Proceedings of the 15th ACM SIGKDD international*, 2009.

Andrew J. Frank and Arthur Asuncion. UCI Machine Learning Repository, http://archive.ics.uci.edu/ml, 2010.

John C. Gittins. Bandit processes and dynamic allocation indices. *Journal of the Royal Statistical Society (Series B)*, 41:148–177, 1979.

Ashish Goel, Sudipto Guha, and Kamesh Munagala. Asking the Right Questions: Model-driven Optimization Using Probes. In *Proceedings of the 25th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 203–212, New York, NY, USA, 2006. ACM.

Ashish Goel, Sanjeev Khanna, and Brad Null. The Ratio Index for Budgeted Learning, with Applications. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 18–27, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.

Russell Greiner, Adam J. Grove, and Dan Roth. Learning Cost-sensitive Active Classifiers. *Artificial Intelligence*, 139(2):137–174, 2002.

Sudipto Guha and Kamesh Munagala. Approximation Algorithms for Budgeted Learning Problems. In *Proceedings of the 39th ACM Symposium on Theory of Computing*, pages 104–113, 2007.

Andrew Guillory and Jeff Bilmes. Simultaneous learning and covering with adversarial noise. In *Proceedings of the 28th International Conference on Machine Learning (to appear)*, 2011.

Mark Halland, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA Data Mining Software: An Update. *SIGKDD Explorations Newsletter*, 11:10–18, November 2009.

Taeho C. Jo, , Jerry H. Seo, and Hyeon Kim. Topic Spotting on News Articles with Topic Repository by Controlled Indexing. In *Proceedings of the 2nd International Conference on Intelligent Data Engineering and Automated Learning, Data Mining, Financial Engineering, and Intelligent Agents*, pages 386–391, London, UK, 2000. Springer-Verlag.

Leslie Pack Kaelbling. *Learning in Embedded Systems*. MIT Press, 1993.

Adam Kalai and Santosh Vempala. Efficient Algorithms for Online Decision Problems. *Journal of Computer and System Sciences*, 71:291–307, 2005.

Aloak Kapoor and Russell Greiner. Budgeted Learning of Bounded Active Classifiers. In *Proceedings of the ACM SIGKDD Workshop on Utility-Based Data Mining*, 2005a. Held in conjunction with the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2005).

Aloak Kapoor and Russell Greiner. Learning and Classifying under Hard Budgets. In *Proceedings of the 15th European Conference on Machine Learning*, pages 170–181, 2005b.

Aloak Kapoor and Russell Greiner. Reinforcement Learning for Active Model Selection. In *Proceedings of the 11th ACM SIGKDD Workshop on Utility-Based Data Mining*, pages 17–23, 2005c. Held in conjunction with the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2005).

Igor Kononenko. Machine Learning for Medical Diagnosis: History, State of the Art and Perspective. *Artificial Intelligence in Medicine*, 23:89–109, 2001.

Kevin B. Korb and Ann E. Nicholson. *Bayesian Artificial Intelligence*. Chapman & Hall, 2004.

Tze L. Lai and Herbert Robbins. Asymptotically Efficent Adaptive Allocation Rules. *Advances in Applied Mathematics*, 6:4–22, 1985.

Steffen L. Lauritzen and David J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. pages 157–194, 1988.

Yat-Chiu Law, Jimmy Lee, Toby Walsh, and Justin Yip. Breaking symmetry of interchangeable variables and values. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming*, pages 423–437, Berlin, Heidelberg, 2007. Springer-Verlag.

David D. Lewis and William A. Gale. A Sequential Algorithm for Training Text Classifiers. In *Proceedings of the 17th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 3–12. Springer-Verlag, 1994a.

David D. Lewis and William A. Gale. A Sequential Algorithm for Training Text Classifiers. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 3–12. Springer-Verlag New York, Inc., 1994b.

Liuyang Li. Budgeted Parameter Learning of Generative Bayesian Networks. Master's thesis, University of Alberta, 2009.

Liuyang Li, Barnabás Póczos, Csaba Szepesvári, and Russ Greiner. Budgeted Distribution Learning of Belief Net Parameters. In *Proceedings of 27th International Conference of Machine Learning*, 2010.

Daniel J. Lizotte, Omid Madani, and Russell Greiner. Budgeted Learning of Naïve-Bayes Classifiers. In *Proceedings of the 19th Annual Conference on Uncertainty in Artificial Intelligence*, pages 378–385, 2003.

Rachel Lomasky, Carla Brodley, Matthew J. Aernecke, David Walt, and Mark A. Friedl. Active Class Selection. In *Machine Learning: ECML 2007*, volume 4701 of *Lecture Notes in Computer Science*, pages 640–647. Springer Berlin, Heidelberg, 2007.

Tong Luo, Kurt Kramer, and Dmitry B. Goldgof. Active Learning to Recognize Multiple Types of Plankton. In *Journal of Machine Learning Research*, page 2005, 2004.

Omid Madani, Daniel Lizotte, and Russell Greiner. Active Model Selection. In *Proceedings of the 20th Conference on Uncertinty in Artificial Intelligence*, pages 357–365, July 2004.

Prem Melville and Raymond J. Mooney. Diverse Ensembles for Active Learning. In *Proceedings of the 21st International Conference on Machine learning*, pages 584–591, 2004.

Prem Melville, Maytal Saar-Tsechansky, Foster Provost, and Raymond Mooney. An Expected Utility Approach to Active Feature-value Acquisition.

Prem Melville, Maytal Saar-Tsechansky, Foster Provost, and Raymond Mooney. Active Feature-value Acquisition for Classifier Induction. In *Proceedings of the 4th IEEE International Conference on Data Mining*, pages 483–486, 2004.

Prem Melville, Saharon Rosset, and Richard D. Lawrence. Customer Targeting Models Using Actively-Selected Web Content. In *Proceeding of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 946–953, New York, NY, USA, 2008a. ACM.

Prem Melville, Saharon Rosset, and Richard D. Lawrence. Customer Targeting Models Using Actively-Selected Web Content. In *Proceeding of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 946–953, 2008b.

Richard E. Neapolitan. *Learning Bayesian Networks*. Prentice Hall, 2004.

Jennifer Neville and Ankit Kuwadekar. Relational Active Learning for Joint Collective Classification Models. In *Proceedings of the 28th International Conference on Machine Learning (to appear)*, 2011.

Norsys Software Corp. http://www.norsys.com/index.html, 2011.

J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.

Vikas C. Raykar, Shipeng Yu, Linda H. Zhao, Anna Jerebko, Charles Florin, Gerardo Hermosillo Valadez, Luca Bogoni, and Linda Moy. Supervised Learning from Multiple Experts: Whom to Trust When Everyone Lies a Bit. In *Proceedings of the 26th International Conference on Machine Learning*, pages 889–896, 2009.

Herbert Robbins. Some Aspects of the Sequential Design of Experiments. *Bulletin of the American Mathematical Society*, 55:527–535, 1952.

Nicholas Roy and Andrew Mccallum. Toward Optimal Active Learning through Sampling Estimation of Error Reduction. In *Proceedings of the 18th International Conf. on Machine Learning*, pages 441–448. Morgan Kaufmann, 2001.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995.

Maytal Saar-Tsechansky, Prem Melville, and Foster Provost. Active Feature-value Acquisition. *Management Science*, 55(4):664–684, 2009.

G. Schohn and D. Cohn. Less Is More: Active Learning with Support Vector Machines. In *Proceedings of the 17th International Conference on Machine Learning*, pages 839–846, 2000.

Victor S. Sheng, Foster Provost, and Panagiotis G. Ipeirotis. Get Another Label? Improving Data Quality and Data Mining Using Multiple, Noisy Labelers. In *Proceedings of the 14th SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 614–622, 2008.

Padhraic Smyth, Usama Fayyad, Michael Burl, Pietro Perona, and Pierre Baldi. Inferring Ground Truth from Subjective Labelling of Venus Images. In *Advances in Neural Information Processing Systems*, pages 1085–1092, 1995.

Rion Snow, Daniel Jurafsky, and Andrew Y. Ng. Cheap and Fast But is it Good? Evaluating Non-Expert Annotations for Natural Language Tasks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 254–263, 2008.

David J. Spiegelhalter, Dawid A. Philip, Steffen L. Lauritzen, and Robert G. Cowell. Bayesian analysis in expert systems. (3):219–283, 1993.

Kah-Kay Sung and Tomaso Poggio. Example-Based Learning for View-Based Human Face Detection. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 20:39–51, 1998.

Simon Tong and Daphne Koller. Support Vector Machine Active Learning with Applications to Text Classification. In *Journal of Machine Learning Research*, pages 999–1006, 2000.

Simon Tong and Daphne Koller. Active Learning for Parameter Estimation in Bayesian Networks. In *In NIPS*, pages 647–653, 2001a.

Simon Tong and Daphne Koller. Active Learning for Structure in Bayesian Networks. In *In International Joint Conference on Artificial Intelligence*, pages 863–869, 2001b.

Simon Tong and Daphne Koller. Support Vector Machine Active Learning with Applications to Text Classification. *Journal of Machine Learning Research*, 2(Nov):45–66, 2001c.

Konstantin Tretyakov. Machine Learning Techniques in Spam Filtering. Technical report, Institute of Computer Science, University of Tartu, 2004.

Sriharsha Veeramachaneni, Emanuele Olivetti, and Paolo Avesani. Active Sampling for Detecting Irrelevant Features. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 961–968, 2006.

Tao Wang, Daniel Lizotte, Michael Bowling, and Dale Schuurmans. Bayesian Sparse Sampling for On-line Reward Optimization. In *Proceedings of the 22nd International Conference of Machine Learning*, pages 956–963, 2005.

Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin 1*, 6: 80–83, 1945.

Ian Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco, 2nd edition, 2005.

Harry Zhang. The Optimality of naïve Bayes. In *Proceedings of the 17th International FLAIRS Conference*, 2004.

Yaling Zheng, Stephen Scott, and Kun Deng. Active Learning from Multiple Noisy Labelers with Varied Costs. *Proceedings of the 10th IEEE International Conference on Data Mining*, pages 639–648, 2010.

Zhiqiang Zheng and Balaji Padmanabhan. On Active Learning for Data Aquisition. In *Proceedings of the IEEE International Conference on Data Minding*, pages 562–570, 2002.

Xiaojin Zhu, Timothy Rogers, Ruichen Qian, and Chuck Kalish. Humans Perform Semi-Supervised Classification Too. In *Proceedings of the 22nd national conference on Artificial intelligence - Volume 1*, pages 864–869. AAAI Press, 2007.