2004

Multi-heuristic theory assessment with iterative selection

Kareem Ammar
West Virginia University

MULTI-HEURISTIC THEORY ASSESSMENT WITH ITERATIVE SELECTION

Kareem Ammar

Thesis submitted to the College of
Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Masters of Science
in
Electrical Engineering

Menzies, Tim J. Ph.D., Chair
Cukic, Bojan Ph.D.
Goseva-Popstojanova, Katerina D. Ph.D.

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2004

Keywords: Machine Learning, Data Mining, Software Metrics, ROC curves

WEST VIRGINA UNIVERSITY

ABSTRACT

MULTI-HEURISTIC THEORY
ASSESSMENT WITH ITERATIVE
SELECTION

by Kareem Ammar

Chairperson of the Supervisory Committee: Professor Tim Menzies
Department of Computer Science and Electrical Engineering

Modern day machine learning is not without its shortcomings. To start with, the heuristic accuracy, which is the standard assessment criteria for machine learning, is not always the best heuristic to gauge the performance of machine learners. Also machine learners many times produce theories that are unintelligible by people and must be assessed as automated classifiers through machines. Theses theories are either too large or not properly formatted for human interpretation. Furthermore, our studies have identified that most of the data sets we have encountered are satiated with worthless data that actually leads to the degradation of the accuracy of machine learners. Therefore, simpler learning is more optimal. This necessitates a simpler classifier that is not confused with highly correlated data. Lastly, existing machine learners are not sensitive to domains. That is, they are not tunable to search for theories that are most beneficial to specific domains.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

The author wishes to express sincere appreciation to Professor Menzies and Ammar for their unparalleled patience and support in the completion of this thesis and the author's Masters.

1 Introduction

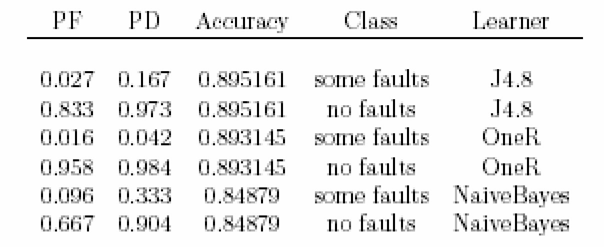
Current machine learning has developed a plethora of various algorithms that learn through diverse methods. In a few minutes anyone may download over thirty different public domain machine learning algorithms through the WEKA machine learning toolkit [Witten99]. Therefore, the question we focus on is not how the machine learners are executed or even which machine learner is best for a given data set, but what can be gained from learning through various algorithms. In summary, this thesis rejects classic singleton assessment statistics for evaluating a learnt theory. The classic singleton assessment for much of the modern learning algorithms is classification accuracy.

In this chapter we will discuss a few of the motivations behind this thesis and briefly introduce the general idea behind our approach. The following sections are a few of the current issues of modern machine learning. We briefly address each and explain how our own homegrown solution deals with these issues.

1.1 Accuracy Comparisons

The standard comparison of classifiers assists the selection of classifiers in a given domain. Therefore, in this classical approach, better comparisons would yield better classifiers. Fawcett proposes that the measure of accuracy does not perform well in domains with a greatly skewed, unknown class distribution or with uneven error costs [Fawcett98]. In other words, many real world classifiers would not benefit by comparison through accuracy.

For example, in the NASA software metric data set deemed CM1, standard machine learners achieve a classification accuracy of more than eighty five percent in predicting whether or not a given software module has faults as shown in Figure 1.1. The accuracy is the total accuracy reported by the learner through a validation technique known as ten way cross validation, which we define in Chapter 2. However, when you decompose the results on a class basis it becomes evident that this total accuracy is very misleading, as the probability of detecting faults is very low and the probability of false alarm when there is no fault is relatively high. As a result, accuracy is not a good indicator of the learner's future performance in this domain. If accuracy is not the best assessment for learning, then there must be other assessment criteria that is better or at least performs better in certain domains where accuracy is lacking. In the next section we will introduce different ways to assess learning.



PF	PD	Accuracy	Class	Learner
0.027	0.167	0.895161	some faults	J4.8
0.833	0.973	0.895161	no faults	J4.8
0.016	0.042	0.893145	some faults	OneR
0.958	0.984	0.893145	no faults	OneR
0.096	0.333	0.84879	some faults	NaiveBayes
0.667	0.904	0.84879	no faults	NaiveBayes

Figure 1.1 Leaner runs revealing misleading accuracies

1.2 Alternative Assessment Methods

Since it has been established that accuracy is not always the best assessment method, there is research in alternative methods to assess learning [Fawcette98]. One of the methods we employ throughout this thesis is receiver operator characteristic curves. These curves are created from machine learners' classifiers and reveal a level of performance of the classifier by comparing the continual trade offs between the statistical concepts of probability of detection and probability of false alarm. There are other statistical heuristics such as confidence and support which can also be used to assess learning algorithms.

Furthermore, we can also employ domain specific heuristics such as the effort it takes a test engineer to debug code by measuring the number of lines they must go through before finding a fault or the dollar costs of components on satellite with limited weight capacities. These heuristics would greatly enhance the significance and usability of the classifiers by tailoring the output of machine learners by what is most critical in a specific domain.

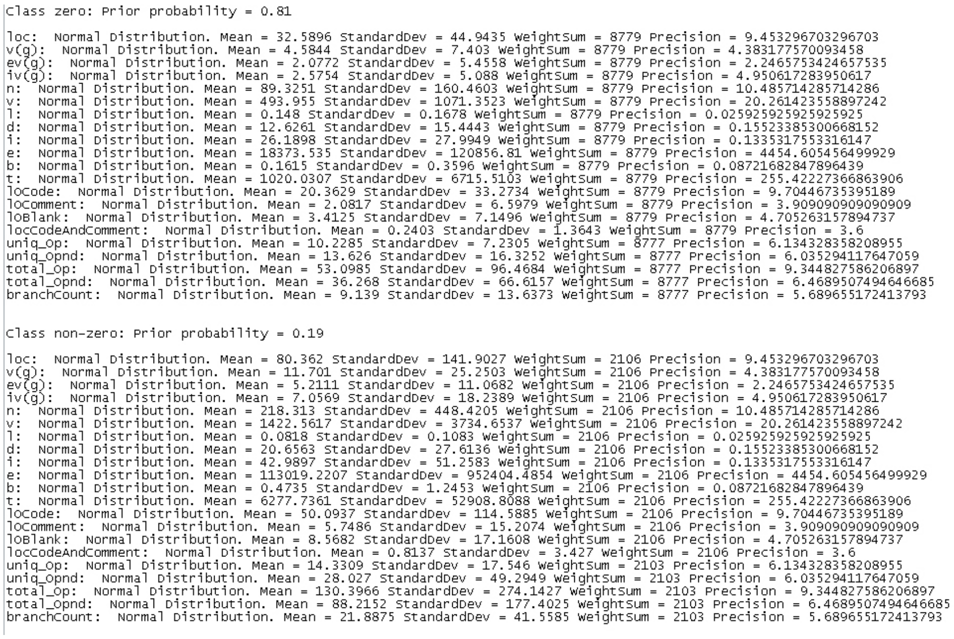
However, there is merit in the combinations of the above assessment criteria. The information obtained by the probability of false alarm is complimented by information obtained from the probability of detection, confidence, support, etc. These assessments can be even further enhanced by the production of classifiers that can be understood by humans and not tailored for automatic execution by machines, as we see in the next section.

1.3 Understandable Theories

Machine learning can vary greatly in the types and sizes of theories produced. Very simple classifiers such as Naïve Bayes produce a classifier with Bayesian values associated with attribute ranges that are not easily comprehended by people as seen in Figure 1.2.

In contrast, the popular machine learning algorithm C4.5 produces a decision tree, a conjunction of decision nodes and paths that is straightforward to follow if the size of the tree is reasonable. Figure 1.4 displays a small decision tree from a software metrics dataset. However, C4.5 often produces decision trees that are too large to fully realize, such as in Figure 1.3, which is produced from the same software metric data set but with all the features or attributes visible. Both these trees possess relatively the same classification accuracy produced by the machine learner J4.8.

We maintain that the smaller and more concise a theory, the easier it is to comprehend and thus should be chosen above a more complex theory if both theories possess similar merit. This premise is drawn from Ockham's Razor, a concept invented by the famed William of Ockham in the fourteenth century, which proposes that "entities should not be multiplied unnecessarily." In other words, the simpler the theory is, the better. In the next section we will discuss feature selection and how the simplicity of the initial space impacts learning.



```
Class zero: Prior probability = 0.81

loc:  Normal Distribution. Mean = 32.5896 StandardDev = 44.9435 WeightSum = 8779 Precision = 9.453296703296703
v(g):  Normal Distribution. Mean = 4.5844 StandardDev = 7.403 WeightSum = 8779 Precision = 4.383177570093458
ev(g):  Normal Distribution. Mean = 2.0772 StandardDev = 5.4558 WeightSum = 8779 Precision = 2.2465753424657535
iv(g):  Normal Distribution. Mean = 2.5754 StandardDev = 5.088 WeightSum = 8779 Precision = 4.950617283950617
n:  Normal Distribution. Mean = 89.3251 StandardDev = 160.4603 WeightSum = 8779 Precision = 10.485714285714286
v:  Normal Distribution. Mean = 493.955 StandardDev = 1071.3523 WeightSum = 8779 Precision = 20.261423558897242
l:  Normal Distribution. Mean = 0.148 StandardDev = 0.1678 WeightSum = 8779 Precision = 0.025925925925925925
d:  Normal Distribution. Mean = 12.6261 StandardDev = 15.4443 WeightSum = 8779 Precision = 0.15523385300668152
i:  Normal Distribution. Mean = 26.1898 StandardDev = 27.9949 WeightSum = 8779 Precision = 0.1335317553316147
e:  Normal Distribution. Mean = 18373.535 StandardDev = 120856.81 WeightSum = 8779 Precision = 4454.605456499929
b:  Normal Distribution. Mean = 0.1615 StandardDev = 0.3596 WeightSum = 8779 Precision = 0.08721682847896439
t:  Normal Distribution. Mean = 1020.0307 StandardDev = 6715.5103 WeightSum = 8779 Precision = 255.42227366863906
lOCode:  Normal Distribution. Mean = 20.3629 StandardDev = 33.2734 WeightSum = 8779 Precision = 9.70446735395189
lOComment:  Normal Distribution. Mean = 2.0817 StandardDev = 6.5979 WeightSum = 8779 Precision = 3.909090909090909
lOBlank:  Normal Distribution. Mean = 3.4125 StandardDev = 7.1496 WeightSum = 8779 Precision = 4.705263157894737
locCodeAndComment:  Normal Distribution. Mean = 0.2403 StandardDev = 1.3643 WeightSum = 8779 Precision = 3.6
uniq_Op:  Normal Distribution. Mean = 10.2285 StandardDev = 7.2305 WeightSum = 8777 Precision = 6.134328358208955
uniq_Opnd:  Normal Distribution. Mean = 13.626 StandardDev = 16.3252 WeightSum = 8777 Precision = 6.035294117647059
total_Op:  Normal Distribution. Mean = 53.0985 StandardDev = 96.4684 WeightSum = 8777 Precision = 9.344827586206897
total_Opnd:  Normal Distribution. Mean = 36.268 StandardDev = 66.6157 WeightSum = 8777 Precision = 6.4689507494646685
branchCount:  Normal Distribution. Mean = 9.139 StandardDev = 13.6373 WeightSum = 8777 Precision = 5.689655172413793


Class non-zero: Prior probability = 0.19

loc:  Normal Distribution. Mean = 80.362 StandardDev = 141.9027 WeightSum = 2106 Precision = 9.453296703296703
v(g):  Normal Distribution. Mean = 11.701 StandardDev = 25.2503 WeightSum = 2106 Precision = 4.383177570093458
ev(g):  Normal Distribution. Mean = 5.2111 StandardDev = 11.0682 WeightSum = 2106 Precision = 2.2465753424657535
iv(g):  Normal Distribution. Mean = 7.0569 StandardDev = 18.2389 WeightSum = 2106 Precision = 4.950617283950617
n:  Normal Distribution. Mean = 218.313 StandardDev = 448.4205 WeightSum = 2106 Precision = 10.485714285714286
v:  Normal Distribution. Mean = 1422.5617 StandardDev = 3734.6537 WeightSum = 2106 Precision = 20.261423558897242
l:  Normal Distribution. Mean = 0.0818 StandardDev = 0.1083 WeightSum = 2106 Precision = 0.025925925925925925
d:  Normal Distribution. Mean = 20.6563 StandardDev = 27.6136 WeightSum = 2106 Precision = 0.15523385300668152
i:  Normal Distribution. Mean = 42.9897 StandardDev = 51.2583 WeightSum = 2106 Precision = 0.1335317553316147
e:  Normal Distribution. Mean = 113019.2207 StandardDev = 952404.4854 WeightSum = 2106 Precision = 4454.605456499929
b:  Normal Distribution. Mean = 0.4735 StandardDev = 1.2453 WeightSum = 2106 Precision = 0.08721682847896439
t:  Normal Distribution. Mean = 6277.7361 StandardDev = 52908.8088 WeightSum = 2106 Precision = 255.42227366863906
lOCode:  Normal Distribution. Mean = 50.0937 StandardDev = 114.5885 WeightSum = 2106 Precision = 9.70446735395189
lOComment:  Normal Distribution. Mean = 5.7486 StandardDev = 15.2074 WeightSum = 2106 Precision = 3.909090909090909
lOBlank:  Normal Distribution. Mean = 8.5682 StandardDev = 17.1608 WeightSum = 2106 Precision = 4.705263157894737
locCodeAndComment:  Normal Distribution. Mean = 0.8137 StandardDev = 3.427 WeightSum = 2106 Precision = 3.6
uniq_Op:  Normal Distribution. Mean = 14.3309 StandardDev = 17.546 WeightSum = 2103 Precision = 6.134328358208955
uniq_Opnd:  Normal Distribution. Mean = 28.027 StandardDev = 49.2949 WeightSum = 2103 Precision = 6.035294117647059
total_Op:  Normal Distribution. Mean = 130.3966 StandardDev = 274.1427 WeightSum = 2103 Precision = 9.344827586206897
total_Opnd:  Normal Distribution. Mean = 88.2152 StandardDev = 177.4025 WeightSum = 2103 Precision = 6.4689507494646685
branchCount:  Normal Distribution. Mean = 21.8875 StandardDev = 41.5585 WeightSum = 2103 Precision = 5.689655172413793
```

Figure 1.2 Naïve Bayes classifier

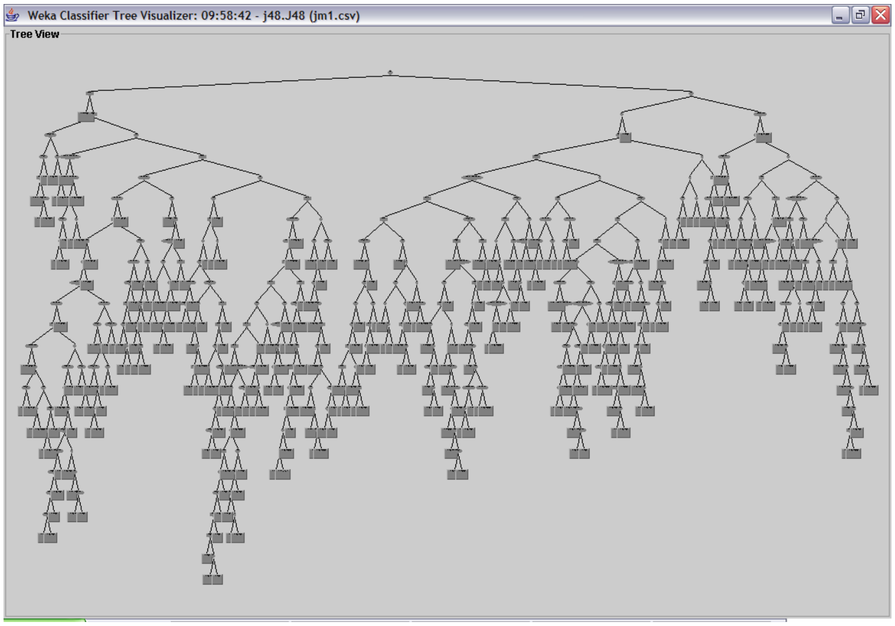

Weka Classifier Tree Visualizer: 09:58:42 - j48.J48 (jm1.csv)
Tree View

Figure 1.3 A large decision tree produced by C4.5

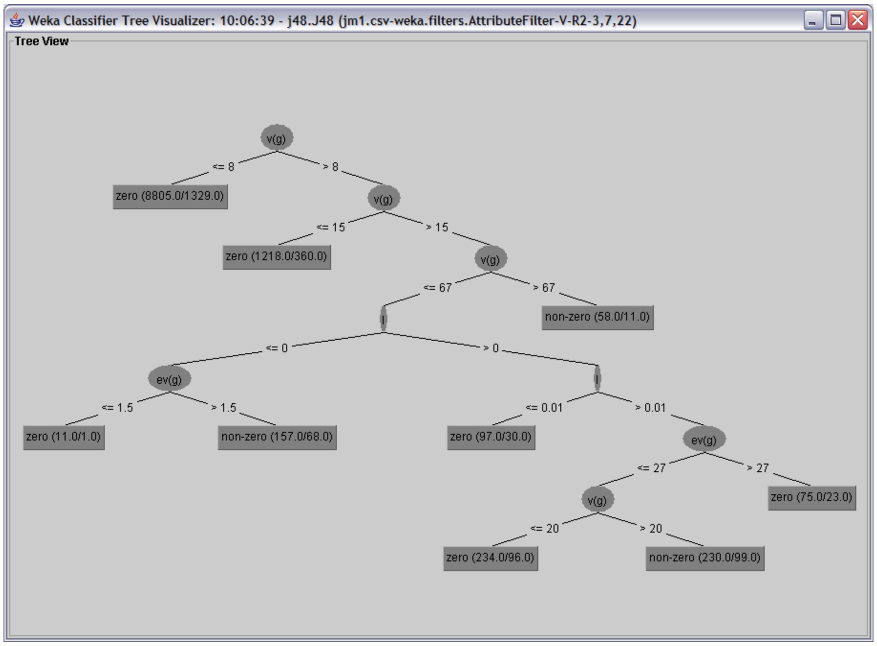


Figure 1.4 Small decision tree produced by J4.8

1.4 Feature Subset Selection

Feature subset selection is a method used in machine learning to cull worthless or redundant attributes from a data set. The difference between Figure 1.3 and Figure 1.4 is that a feature selection technique was applied to the data set before running a machine learner in Figure 1.4. Figure 1.3 is results from a learner using all the attributes from a data set. Surprisingly, the accuracy of the smaller classifier is slightly improved. As in other experiments [Hall98, Menzies03] there were also significant increases in accuracies of classifiers that used feature selection. However we do not only focus on accuracy. Instead, we observe the changes from a set of different heuristics with feature selection. We also focus on simpler learning algorithms that do not require the support of external feature selection techniques that become cumbersome to execute when several feature selection techniques are observed.

1.5 Contributions

In this section we show the contributions of this thesis.

- demonstrate a new learning technique that is inherently different from classical machine learners in that it iteratively assesses and builds detectors based on ROC curves and other heuristics.

- demonstrate that our learning technique is tunable to different domains

- show how well this learner performs on several different types of data.

- illustrate the effects of FSS techniques on many of heuristics we have observed.

- observe different types of detectors, conjunctive, disjunctive, composite and singleton and how they relate through the heuristics observed.

1.6 Organization

The organization of this thesis is as follows. First the above introduction established the motivation and direction of our methods. Next we disclose a wealth of pertinent background knowledge, discussing what already exists in machine learning and the variety of problems that we find with current fields of machine learning. Then we introduce our method, IQ, by briefly describing the algorithm. We then unfold all the fine detail and nuances associated with the algorithm such as the quick hull algorithm and iterative search. Afterward, we discuss our various case studies. These include studies with feature subset selection, standard accuracy heuristics, and studies with UC Irvine, NASA data sets, and emulation of treatment learning. We then discuss our results from the case studies and our future direction. Finally we conclude this thesis.

2 Literature Review

In this chapter we will discuss a variety of topics all pertinent to this thesis. First we look at ROC curves as they are the core of our new machine learning algorithm. Next we discuss standard machine learning, and the various shortcomings of several popular algorithms. Then we introduce feature subset selection and its impact on machine learning.

2.1 Receiver Operator Characteristic Curves

In this section we introduce ROC curves. Through ROC curves we assess our theories and theories of other learners. The first part of this section will explain the basics of ROC curves. We then explain how current machine learning uses ROC curves as oppose to our approach.

2.1.1 Classical Receiver Operator Characteristic Curves

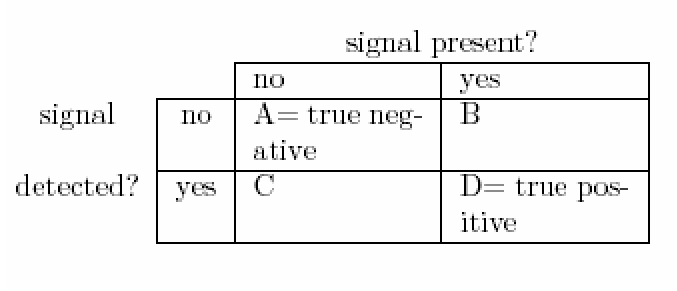
Formally, a defect *detector* seeks a *signal* that a software module is defect prone. Signal detection theory [Heeger98] offers receiver operator characteristic curves or ROC curves that are an analysis method for assessing different detectors. ROC curves are widely used in various fields including assessing different clinical computing systems [Adlassnig89] and assessing different machine learning methods [Fawcett98]. The central premise of ROC curves is that different detectors can be assessed via how often they correctly or incorrectly respond to the presence or absence of a signal.

2.1.2 Constructing Receiver Operator Characteristic Curves

To draw a ROC curve, the ROC sheet of Figure 2.1 must be first completed. If detector registers a signal, sometimes the signal is actually present (cell D) and sometimes it is absent (cell C). Alternatively, the detector may be silent when the signal is absent (cell A) or present (cell B). Figure 2.1 lets us define the accuracy, or

“Acc”, of a detector as the number of true negatives and true positives seen over all events:

$$\text{accuracy} = \text{Acc} = {}^{A + D}/_{A + B + C + D}$$



		signal present?	
		no	yes
signal	no	A= true negative	B
detected?	yes	C	D= true positive

Figure 2.1 A ROC sheet

If the detector registers a signal, there are two cases of interest. In one case, the detector has correctly recognized the signal. This probability of detection, or “PD”, is the ratio of detected signals, true positives, to all signals:

$$\text{probability detection} = \text{PD} = {}^{D}/_{B + D}$$

In the other case, the probability of a false alarm, or “PF”, is the ratio of detections when no signal was present to all non-signals:

$$\text{probability false alarm} = \text{PF} = {}^{C}/_{A + C}$$

Another heuristic we use is precision or in some texts known as confidence. Confidence is the ratio of truly detected signals to the all detected signals

$$\text{confidence} = \text{CONF} = {}^{D}/_{D + C}$$

The final heuristic we obtain from Figure 2.1 is support. Support is the ratio of truly detected signals to that of all data from that attribute. Support is usually less than 0.3 and normalized for our IQ algorithm. This process is described in Chapter 4.

$$\text{support} = {}^{D}/_{D + C + B + A}$$

In the ideal case, a detector has a high probability of detecting a genuine fault (PD) and a very low probability of false alarm (PF) in conjunction with a high confidence. This ideal case is very rare. Typically, engineers must trade-off between PF and PD. For example, a typical ROC curve is shown in Figure 2.2. Note that a high PD (indicated by the box along the left) is only achievable when PF (the box at the top of the graph) is also high.

The advantage of ROC curves is that they allow for cost-benefit trade-off between different detectors. Defect detectors that fall into the cost-adverse region shown in Figure 2.2 have low probabilities of false alarms. Such defect detectors are best when a budget is limited and the extra effort associated with chasing false alarms is unacceptable. On the other hand, detectors that fall into the risk-adverse region of Figure 2.2 have high probabilities of detecting a signal. However, due to the usual relationship between PD and PF, this high probability comes at the cost of a high false alarm rate. Hence, detectors that fall into this region are best for safety-critical systems where the cost of chasing false alarms is out-weighted by the cost of system failure.

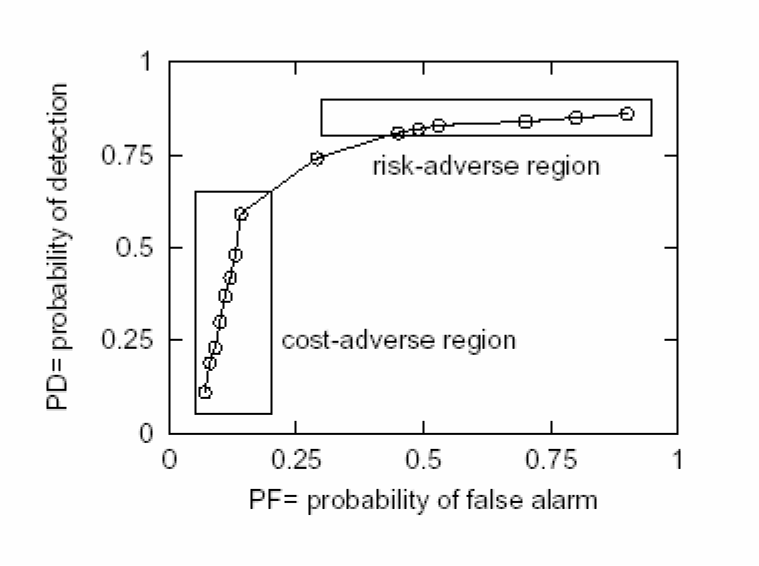


Figure 2.2 Regions of a typical ROC curve

2.1.3 *Receiver Operator Characteristic Curves in machine learning*

The ROC curves from machine learner models are created through assessing the probability of detection and false alarm of the learner. Witten and Frank utilize the values obtained from ten way cross validation for several points of the probability of false alarm and the correlating probability of detection to create a smoothed ROC curve. The more points taken the more veritable the curve becomes of the model.

ROC models traditionally are plotted in the same space for comparisons. In Figure 2.3 we show two different models *a* and *b*. Model *b* provides a higher true positive rate and lower false positive rate and would be optimal over model an if we are more concerned about not making mistakes. Conversely, in the case that we our concern is not about failures model *a* would be superior. There is also literature pertaining to the combination of both models [Fawcette98]. This hybrid classifier is a combination of the best of model *a* and model *b*. When the more true positives are desired Model *a* is utilized and when false positives need to be minimized model b is used. For our assessments we simply plot a single point and utilize the vertices in the creation of our curves to serve as a simple estimate of the model. However, we do not limit ourselves to merely a comparison of classifiers for ROC as in pervious literature [Fawcett98].

ROC curves are the current device employed in machine learning to assess classifiers under different criteria than accuracy. However, ROC curves can also be valuable in assisting in classifier construction in order to create a more flexible classifier that is not subjective solely to accuracy [Provost00]. By plotting defect detectors within ROC space we move from just basic accuracy to probability of detection and probability of failure [Menzies03]. The question arises as to why limit the space to just two measures. It's also possible to assess default detectors on bases of cost, effort, support, etc...

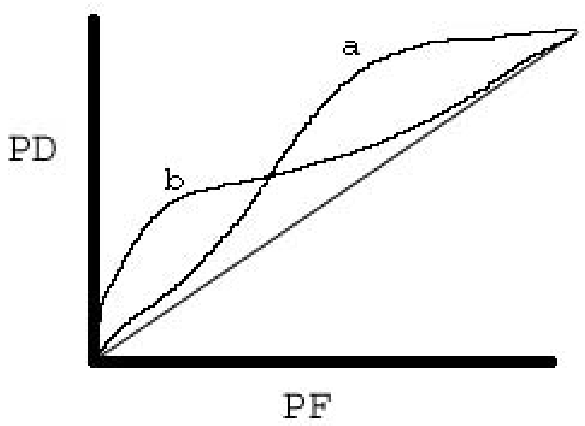


Figure 2.3 Two different classifiers plotted in ROC space

2.1.4 Receiver Operator Characteristic Convex Hull

The ROC convex hull is a combination of ROC analysis, decision analysis, and computational geometry. This device is what Fawcett refers to as the ROC Convex hull [Fawcett98]. The convex hull is used not only to compare classifiers on bases of probabilities of false alarm and of detection but also to obtain the best classifiers for projected future instances.

The first step in constructing the ROC convex hull is to create ROC curves for a set of given classifiers. Fawcette then constructs Iso-performance lines for each ROC curve in order to inject cost and class distribution information into the ROC space. However we do not focus on individual cost of failure or success of each detector. We utilize the ROC convex hull as an outer limit of detectors. These detectors are the most optimal for that specific region of ROC space. We provide further details in Chapter 3.

2.2 Machine Learning

Machine learning also known as data mining is a summarization technique that reduces large sets of examples to small understandable patterns using a range of techniques taken from statistics and artificial intelligence. It is commonly referred to as searching for pearls in the dust. The following are brief descriptions of common data mining concepts and algorithms and what we find to be problematic.

2.2.1 The Case Against Accuracy

Previously [Menzies02], like many others [Khoshgoftaar99, Selby88, Boetticher01], we have argued for certain detectors based on their classification accuracy, a term we define below. Based on new experiments, we now report that classification accuracy is a very poor selector for detectors since it misses many vital domain specific features and tends to ignore class distributions, which in many cases is extremely skewed. Therefore it becomes necessary to compare the performance of classifiers based on other criteria than accuracy. ROC curves are a tool used in the comparison of classifiers without regard to class distribution and error costs. Through preliminary studies, we identified several cases where standard machine learners reside in only select areas of ROC space ignoring potentially valuable regions that can maximize the probability of detection and minimize the probability of failure. In Figure 2.4 we observe three different machine learners running three different software metric data sets. Every data set is assessed twice, once for each class value.

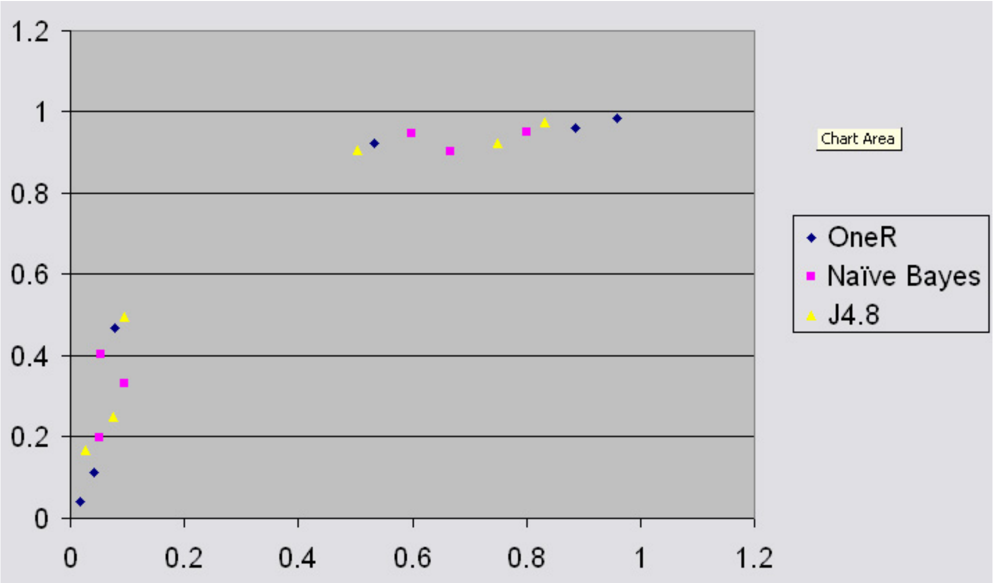


Figure 2.4 Results from three standard machine learners plotted in ROC space.

The same data sets are run through our IQ algorithm and produce hundreds of detectors that span the entire breadth of the ROC convex hull as seen in Figure 2.5

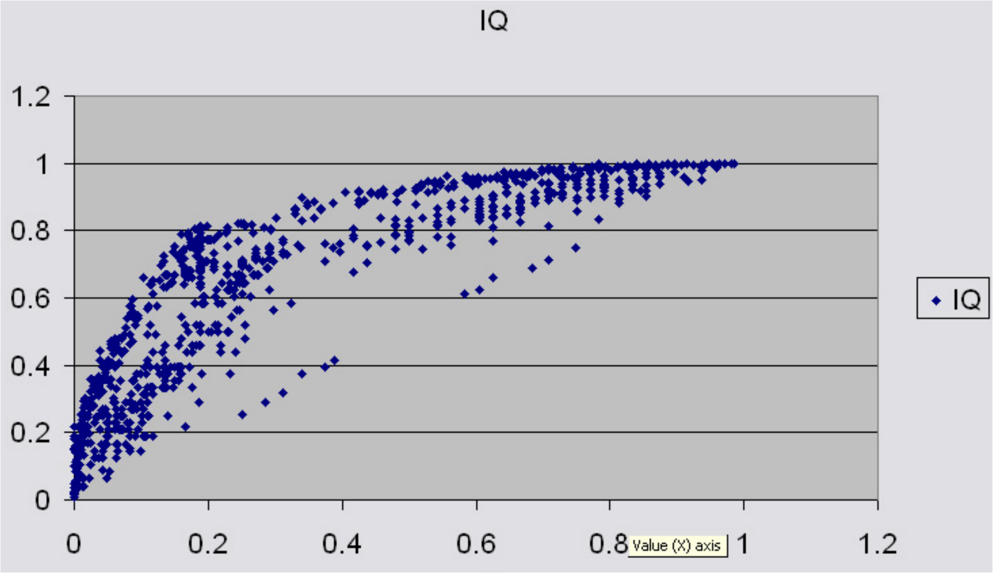


Figure 2.5 Results IQ plotted in ROC space

The difference in number of detectors is greatly varied between the two figures. IQ plots many more detectors distributed across more area in the space in Figure 2.5. From this we obtain more options in choosing a theory. Still, the ROC spaces in these graphs are only two dimensional and thus by assessing detectors only by those criteria, probability of detection and probability false alarm we are limiting our search across only two heuristics.

2.2.2 Decision Trees

Decision trees are a common form of machine learning. Figure 1.3 and Figure 1.4 both show different decision trees from the same data set. The difference lies in the space of attributes, which are culled creating radically different sized tree. All decision trees share commonalities. First each node usually represents an assessment for a one or more attributes. The number of child nodes correlates the number of values from the assessment, which usually include a binary split for continuous attributes or multiple splits if the attribute is discrete, but is not limited to such. Secondly, leaf or end nodes provide a classification. Decision trees start with a single node and usually grow to have many leaf nodes. The size of the tree depends heavily on the data set and the splitting criteria of the algorithm. Creation of decision tree splits can range from a simple divide-and-conquer algorithm to the more complicated C4.5 [Witten99] as we will notice next.

Decision tree creation is a recursive process. The first step is locating the attribute with the most information thus reducing the overall size of the tree. Decision tree learners recursively split creating new nodes by ranking attributes according to how much they decreases the diversity of the classes within the splits. This is essentially our information assessment for each attribute. Once an attribute in chosen as an initial node the corresponding children nodes are attributes with higher information controls the minimum number of instances for an additional split in the decision tree. C4.5 is a resilient machine learner and performs very well in a variety of domains see Figure 4.5 and Figure 4.7. However, we find that that in most cases the decision trees produced are excessively large and pose a problem when directly attempting to apply the learnt theory see Figure 5.1 for tree sizes of some tested data sets. In many cases the decision trees cannot be directly used and must be assessed as a classifier.

2.2.3 Rule Based Learning

Classification rules are another technique in use in data mining. In contrast to decision trees the covering algorithm used to create rule sets splits on basis of accuracy and not information gain. However, in both algorithms recursive splitting is necessary as illustrated in Figure 2.6. In Figure 2.6 we see a space with two class values (plus and minus) and two attributes dimensions (X and Y). Each instance is represented by either a plus or minus class value and is plotted in this dual attribute space. The splits for the nodes of the decision tree are created by assessing the information or accuracy through the frequency of class values within a certain space.

At X = a there is a clear division between plus and minus values. Also at Y = b there is a division since we designated the minimum number of instances valid for a split is more than two. This parameter is known as minobs in the Weka environment and plays a key role in the size of decision trees as well as number of classification rules since it directly limits the number of instances needed for the next rule or tree node. Different from decision trees, classification rules require an order of execution that is prioritized through a number of different methods.

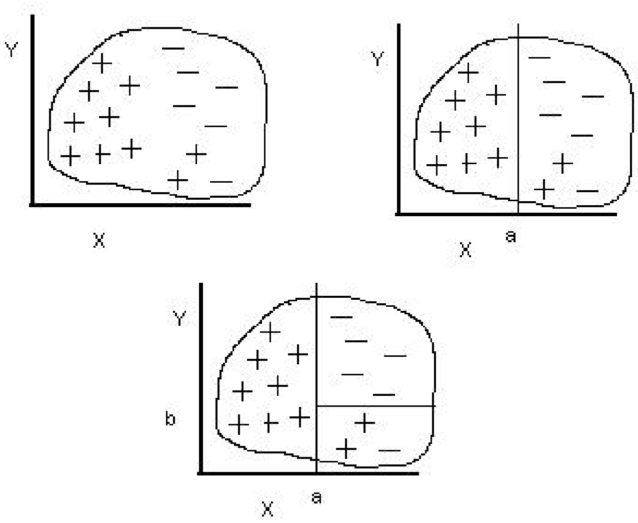


Figure 2.6 A simplified covering algorithm

2.2.4 Bayesian Learning

The Naïve Bayes classifier is also a well known and used classification method in machine learning [Witten99]. This learner uses prior probability theory to update probabilities through instances in the data set. Bayes rule is simply:

$$P(H\,|\,E)= {}^{P(E\,|H)\,P(H)}/_{P(E)}$$

Probabilistic methods for inductive learning such as Naïve Bayes have great advantages such as resistance to noise, strong statistical theoretical base and are naturally very flexible.

However, previous work on Naïve Bayesian classifiers proved that there are limitations. Initially, Naïve Bayes is labeled as naive because it lacks the foresight to perceive correlations between attributes. In other words, Naïve Bayes assumes that attributes are independent. This can greatly skew results if the attributes are highly correlated. Naïve Bayes classifiers are also limited to only learning classes that are separated by a single decision boundary [Witten99]. There are no composite attribute values that determine the class. But the most prevalent downside of Naïve Bayes is the lack of any comprehensible theory. Naïve Bayes produces a classifier as shown in Figure 1.2, but cannot easily be translated into a useable and concise theory. Instead it must be treated as a classifier and comparative data must be sent through it to obtain classifications.

2.2.5 OneR Learner

Simpler than any of the above techniques is the OneR machine learner [Holte93]. It creates a set of rules from a single attribute. First OneR selects an attribute then branches within the attribute to create a set of divisions based on class value. For each division it assigns the most frequent class and then computes the error rate. Finally, OneR simply chooses the attribute with the total least error rate. OneR concludes its execution with a classifier with single attribute with several possible divisions. One R is simple yet it not useful for data sets with several key features.

2.2.6 ROCKY Learner

ROCKY is a simple machine learning algorithm which utilizes components from ROC curves to evaluate its detectors. Given a set of numeric metrics:

$$\mathbf{attribute_1, attribute_2, ...attribute_n}$$

ROCKY exhaustively explores all singleton rules of the form:

attribute = threshold

Threshold is found as follows. Every numeric attribute is assumed to come from Gaussian distribution. Thresholds are then selected corresponding to equal areas under that distribution. For example, in one of the data sets we examine, the McCabe cyclomatic complexity v(g) had a mean of $\mu = 4.9$ and a standard deviation of $s = 11$. If this Gaussian is converted to a unit Gaussian (by subtracting the mean and dividing by the standard deviation), then standard Z-tables could are used to calculate a v(g) threshold value of 7.65. A key point is that ROCKY can only create detectors with a single attribute. This provides us with the inspiration to seek detectors with multiple attributes, and was the driving force for IQ. We focus more on the ROCKY algorithm in Chapter 3.

2.2.7 ROC and Rule learning

Fawcette also proposed a classifier design based of rule sets which are maximized from ROC curves [Fawcett01]. Through these rule sets, instance scores are assigned to instances revealing the probability of belonging to a certain class. ROC curves possess the ability to produce solid instance based scores. Although, these scores are not always accurate they are good enough in determining between positive and negative instances. The area under the ROC curve (AUC) is specifically utilized to optimize rule sets. Each rule carries with it, the number of true positives, false positives. These values when combined with the false negative and false positive costs allow a tentative look as to the class of an instance classified by a given rule. However, we find that Fawcett does not feedback ROC data in order to further improve the rule sets. As a result, the rule sets are merely assessed with ROC heuristics for possible classification and are not grown from ROC data.

2.2.8 Treatment Learning

A new data mining technique is the treatment learning technique developed by Menzies and Yu [Menzies02, Menzies03]. Treatment learning searches for a strong select statement that most changes the ratio of classes. The TAR2 treatment learner seeks ranges of features that select for preferred classes. A repeated empirical observation of TAR2 selects only a very small number of treatments. We will return to this premise in Chapter 5. A treatment is a constraint which, if applied to a data set, returns a subset of the data with different distributions of classes. Association rule learners such as TAR2 explore weighted learning in which some items are given a higher priority weighting those others. Such weights can focus the learning onto issues that are of particular interest to some audience and allows for limited tunings for different domains.

TAR2 uses confidence-based pruning. Without support-based pruning, association rule learners rely on confidence-based pruning to reject all rules that fall below a minimal threshold of adequate confidence. TAR2 uses an internal confidence1 heuristic to prune treatments [Menzies02].

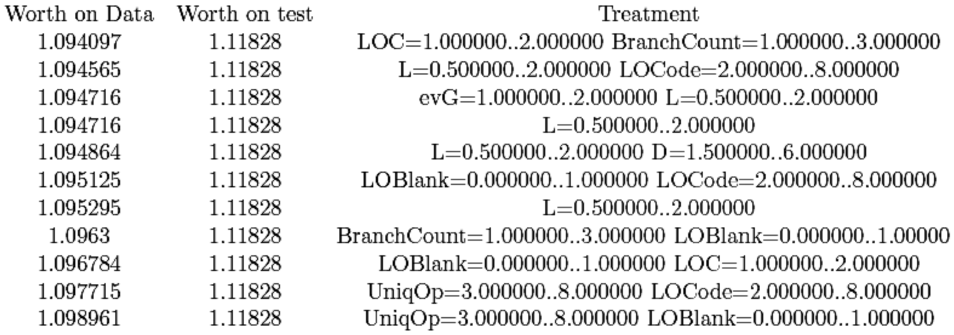
TAR3, the successor to TAR2, creates treatments in a random fashion to avoid a combinatorial explosion that impedes finding treatments with a higher number of attribute ranges. TAR3 more often chooses attribute ranges with a higher confidence1 heuristic, ultimately assembling a set of treatments with high confidence1 values.

However all current treatment learners are dependent upon a weights deemed worth which is the final assessment of the value of a treatment. Lift is the ratio of worth with the treatment to that of the worth of the baseline. We define worth and lift as follows:

$$\text{Worth} = \sum_{i=1}^{i=|\text{class}|} \text{Weight}(C_i) * P(C_i)$$

$$\text{L}\mathit{ift} = \text{Worth(Tre atment)/Wo rth(Baseline Distribution)}$$

Where Weight(C_i) is some incremental weight, which increases as you progress *i* to a best class. Also P(C_i) is the number occurrences of some criteria per class *i*. In the case of a baseline distribution in the lift computation, this criterion is the number of occurrence of each class and in the case of tre atments it's the number of occurrences of that treatment appeared with class *i*. This dependency upon worth and lift is our primary criticism of treatment learning. In Figure 2.7 we observe a set of treatments from a TAR3 run on metric data sets. The results given by TAR3 render theses treatments as indistinguishable. They are very different in terms of other heuristics such as probability of detection and confidence.



Worth on Data	Worth on test	Treatment
1.094097	1.11828	LOC=1.000000..2.000000 BranchCount=1.000000..3.000000
1.094565	1.11828	L=0.500000..2.000000 LOCode=2.000000..8.000000
1.094716	1.11828	evG=1.000000..2.000000 L=0.500000..2.000000
1.094716	1.11828	L=0.500000..2.000000
1.094864	1.11828	L=0.500000..2.000000 D=1.500000..6.000000
1.095125	1.11828	LOBlank=0.000000..1.000000 LOCode=2.000000..8.000000
1.095295	1.11828	L=0.500000..2.000000
1.0963	1.11828	BranchCount=1.000000..3.000000 LOBlank=0.000000..1.00000
1.096784	1.11828	LOBlank=0.000000..1.000000 LOC=1.000000..2.000000
1.097715	1.11828	UniqOp=3.000000..8.000000 LOCode=2.000000..8.000000
1.098961	1.11828	UniqOp=3.000000..8.000000 LOBlank=0.000000..1.000000

Figure 2.7 Worths from TAR3

2.2.9 Validation techniques

Cross validation is a machine learning technique used for validating the learnt model upon data unused in training. This validation technique is widely used and accepted. Cross validation involves dividing the data set through X number of folds. The learner is then executed on all folds save for one. The process is repeated for X times each execution leaves out a different fold. The end result produces X number of models all of with their accuracies, probability of detection, probability of false alarm, confidence and other assessment criteria are then averaged to produce heuristics used to describe the original model created from the entire data set. These heuristics are a are averaged and thus produce a more accurate assessment of the original model than merely assessing the original data set itself.

Ten-way cross validation is simply implies that the data set is divided into ten partitions. Ten is somewhat of a magic number and is the most common number of divisions. There is also the concern of how the data is divided. Stratified cross validation ensures that the divisions of the data set are random and contain relatively equal class distributions to insure that the learner does not receive any skewed data during a specific run of the cross validation process [Witten99].

2.3 Feature Subset Selection

Feature subset selection is a data mining concept which enables the removal of large portions of useless data from a dataset. This is significant because it allows learning algorithms to focus on important features within the data set.

In many past experiments involving feature selection, there are small degradations or in many cases an actual increase of accuracy of the learning algorithm once feature selection is applied [Hall03, Kohavi97, Witten99]. Kohavi & John [Kohvai97] perform studies with Naive Bayes classifiers. Their findings revealed that the accuracy of classifiers decreases slowly as meaningless attributes are added to an instance set. There is evidence that suggests at this occurrence is due to Naïve Bayes inherent assumption that all attributes are independent. Therefore, only relevant attributes are chosen. It is cited that the accuracy of classifiers can decrease significantly if closely correlated attributes the existing set are added. Witten & Frank report degradation in accuracy from one to five percent when relevant attribute is added to the data set. Gunnalan, Menzies, and others have argued that the success of feature selection lies in the concept of small backbones.

Systems with a small backbone possess a correspondingly small number of variants that dictate the control of the system. Feature selection simply locates the attributes that correlate closest to the backbones of the system. Moreover, there are several different types of feature selection techniques. Many of these techniques utilize heuristics to rank the attributes by order of impact while others merely return a subset of attributes. The following are a few feature subset selection techniques that are employed later in this thesis:

2.3.1 Principle Component Analysis

In the reliability engineering literature, principal components analysis [Dillon84] has been widely applied to resolve problems with structural code measurements [Munson90, Munson91]. PCA eliminates the problem of highly correlated measures by identifying the distinct orthogonal sources of variation and mapping the raw measurements onto a set of uncorrelated features that represent essentially the same information contained in the original measurements.

2.3.2 WRAPPER

PCA is a common FSS method used by statisticians. WRAPPER is a common FSS method used by data miners. In this method, a target learner is augmented with a pre-processor that used a heuristic search to grow subsets of the available features. At each step in the growth, the target learner is called to find the accuracy of the model learned from the current subset. Subset growth is stopped when the addition of new features did not improve the accuracy [Kohavi97].

2.3.3 Information Gain

This is a simple and fast method for feature ranking [Dumais98]. This method measures the split criteria of the class before and after observing a feature. The differences in the split criteria give a measure of the information gained because of that attribute [Quinlan92]. A final comparison of this measure is used in feature selection.

2.3.4 Relief

Relief is an instance based learning scheme [Kira92; Kononenko94]. It works by randomly sampling one instance within the data. It then locates the nearest neighbors for that instance from not only the same class but the opposite class as well. The values of the nearest neighbor features are then compared to that of the sampled instance and the feature scores are maintained and updated based on this. This process is specified for some user-specified M number of instances. Relief can handle noisy data and other data anomalies by averaging the values for K nearest neighbors of the same and opposite class for each instance [Kononenko94].

2.3.5 Correlation-bases Feature Selection

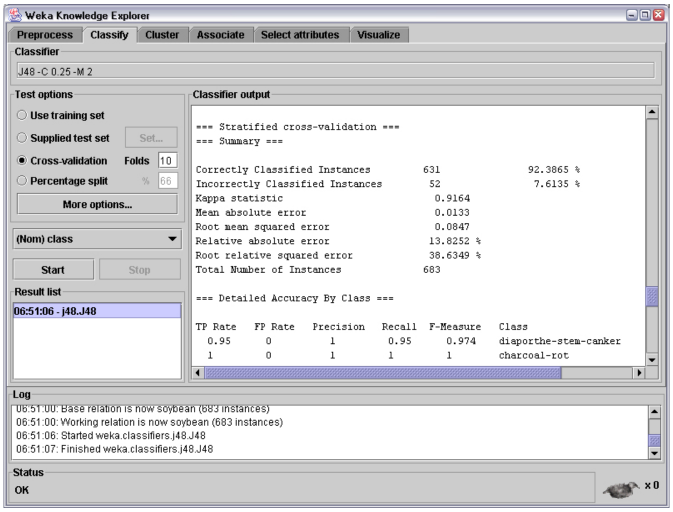
CFS uses subsets of features [Hall98]. This technique relies on a heuristic merit calculation that assigns high scores to subsets with features that are highly correlated with the class and poorly correlated with each other. Merit can find the redundant features since they will be highly correlated with the other features. It can also identify ignorable features since they will be poor predictors of any class. To do this CFS informs a heuristic search for key features via a correlation matrix.

2.3.6 Consistency-based Subset Evaluation

CBS is really a set of methods that use class consistency as an evaluation metric. The specific CBS studied by Hall and Holmes method finds the subset of features whose values divide the data into subsets with high class consistency [Almuallim91].

2.4 Weka Machine Learning Toolkit

Weka is an advanced state of the art machine learning toolkit. It comes bundled with a plethora of learners (currently over thirty) as well as feature selection techniques, data filters and more. Figure 2.8 shows Weka after it has run a data set through J4.8 (Weka's version of Quinlin's C4.5 algorithm). The toolkit is created entirely from JAVA enabling extensive portability on any platform with a JAVA virtual machine. Weka comes bundled with a GUI but also can be command line operated. We employ this toolkit throughout this thesis to obtain results from most of customary machine learning algorithms.



Classifier: J48 -C 0.25 -M 2

Test options: ○ Use training set ○ Supplied test set ● Cross-validation Folds 10 ○ Percentage split % 66

(Nom) class

Classifier output

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         631               92.3865 %
Incorrectly Classified Instances        52                7.6135 %
Kappa statistic                          0.9164
Mean absolute error                      0.0133
Root mean squared error                  0.0847
Relative absolute error                 13.8252 %
Root relative squared error             38.6349 %
Total Number of Instances              683

=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  0.95      0          1         0.95      0.974    diaporthe-stem-canker
  1         0          1         1         1        charcoal-rot
```

Log

```
06:51:00: Base relation is now soybean (683 instances)
06:51:00: Working relation is now soybean (683 instances)
06:51:06: Started weka.classifiers.j48.J48
06:51:07: Finished weka.classifiers.j48.J48
```

Status: OK

x 0

Figure 2.8 Weka toolkit running C4.5

2.5 Summary

In this chapter we a have seen a variety of drawbacks with traditional machine learning. Such drawbacks include incomprehensibly of Bayesian classifiers. Also Naive Bayes is not able to correlate attributes rages. We also observed the simplicity of One R and ROCKY that many times fails to yield useable theories from data with highly independent and principal attributes. Furthermore, we observed that one of the more prominent machine learners C4.5, which many times produces a decision tree that is far too large for human comprehension. Finally we noted that the TAR learners all depend on a single heuristic, lift, which in many cases produces a series of similar treatments but cannot further differentiate between them. The next Chapter introduces and describes in detail our solution to the above shortcomings, labeled IQ.

3 Method

This chapter is divided into several parts. The first explains briefly the IQ algorithm. Then we describe the merits of IQ over other learning schemas. All subsequent part divulge into explicit detail of the IQ algorithm describing all major subroutines.

3.1 Brief Overview

In this section we will discuss evolution of IQ and its relation to its predecessor ROCKY [Menzies03].

In order to create detectors with multiple attribute ranges ROCKY2 was formulated. It was a short lived learner designed as an extension to ROCKY in that it would create composite detectors. However, ROCKY2's performance exponentially degraded due to the combinatorial explosion when combining n-sized detectors. Therefore, a culling method was necessary and IQ was formulated.

IQ in essence learns the same way as ROCKY in that it produces many theories from a simple discretization of attribute ranges then continues in computing the probability of detection, probability of false alarm, and other heuristics. IQ also computes confidence (also known as precision in some literature), support, lift (worth from TAR), distance from (1,0) in ROC space, number of hulls the detector was located on, and stability of detector on cross validation. The specific method IQ uses to compute these values are discussed later. IQ is not limited to any single heuristic. Once all the computations for the heuristics are complete and normalized from zero to one, IQ plots all the detectors according to their heuristic values in n-dimensional space. Inferior detectors are then culled.

The culling of IQ theories occurs either with a beam search or with a geometrical n-dimensional convex hull algorithm under different culling rules. These rules are user specified. The most common culling rule we used for these algorithms is the difference of probability of detection and probability of false alarm. This allows the theories that most likely predict a class with the least amount of failing, the least amount of false alarms, and is applicable to most domains that were tested.

Moreover the theories that lie on the hull (or the outer most boundary of detectors in ROC space), or in the case of beam search detectors that are in the front of the detector array, are then used in subsequent combinations to produce composite detectors. The attribute ranges in the remaining detectors are then combined through subset combinations to produce a large set of detectors which are consequently culled using the same culling rules as the previous generation. This process is repeated until there are no new theories which are created that are better than any existing theory. The algorithm continues to execute until the hull area or detector array no longer increases in size.

IQ is essentially a best first search with iterative deepening in that its concurrent generations do not greatly increase domain space and there is no return to previous levels. A graphical example of the hull culling algorithm is seen though the following figures. A point on these Cartesian planes represents a detector with dimensions PD and PF, probability of detection and probability of false alarm respectfully. Figure 3.1 and Figure 3.2 reveal the culling power of quick hull for singleton IQ detectors. Figure 3.1 shows all detectors found via IQ generation1 without the quick hull culling. Figure 3.2 reveals hull detectors found via IQ generation1 with quick hull culling. Figure 3.3 and Figure 3.4 displays even more impressing culling for detectors of with two attribute ranges. Figure 3.3 reveals 32K of the 178K detectors found via IQ in generation 2, we were limited through Microsoft Excel. While Figure 3.4 shows the culled detectors from a convex hull formulated from the points from Figure 3.3.

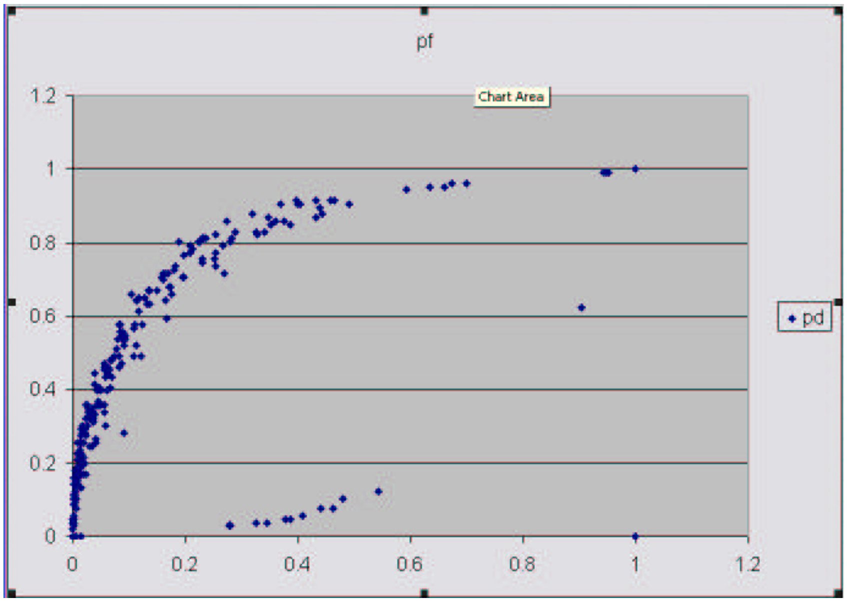


Figure 3.1 Detectors from IQ generation 1

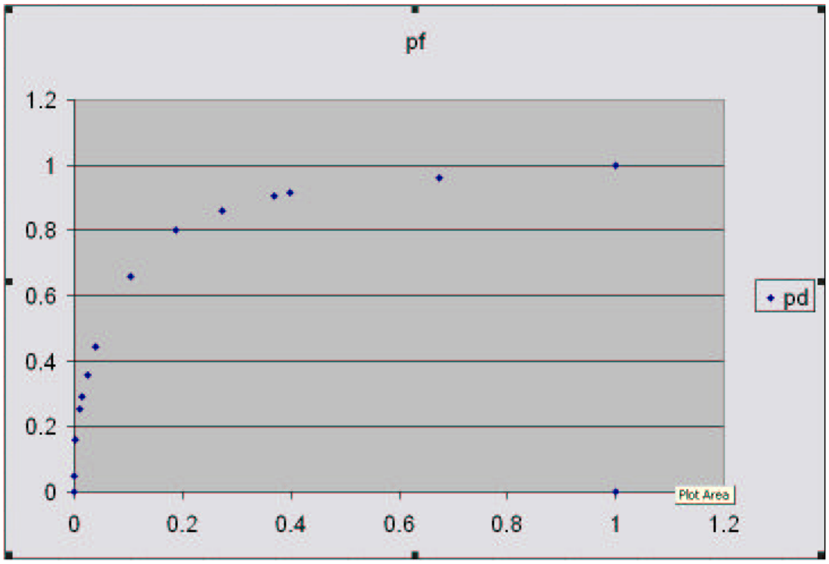


Figure 3.2 Culled space from IQ generation 1.

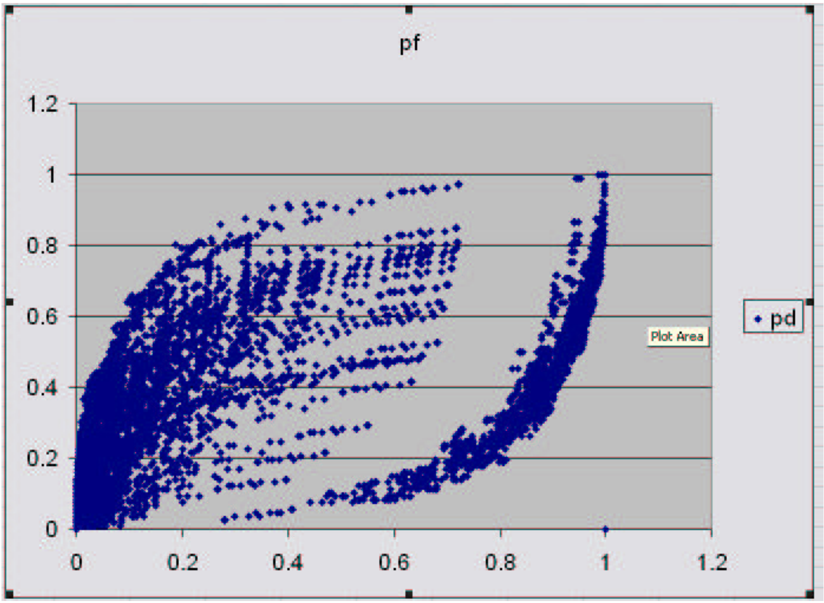


Figure 3.3 The Detectors from IQ generation 2

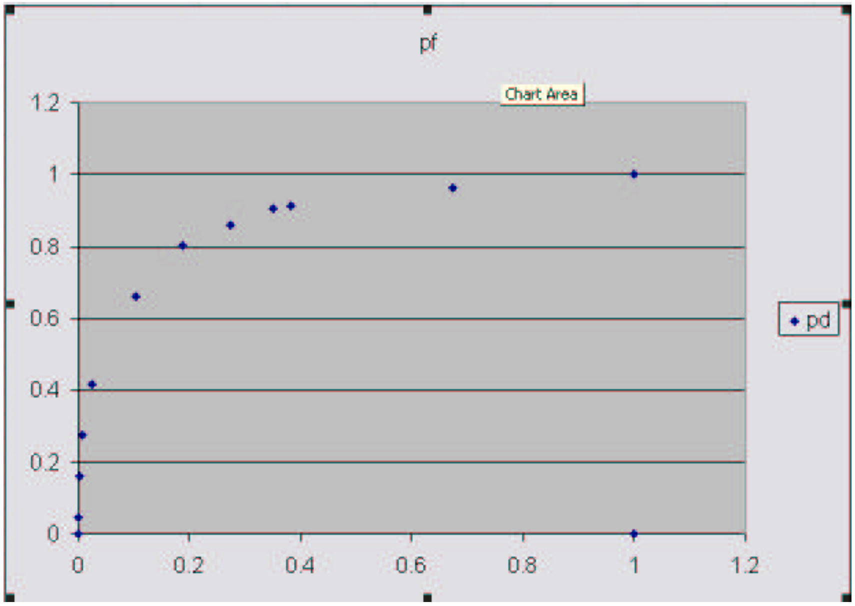


Figure 3.4 Culled space from IQ generation 2

3.2 Merits of IQ

The merits of IQ over other learning algorithms are three fold. First IQ produces an adjustable set of theories for each class value of a data set. Secondly, IQ is tunable based on culling rules that ultimately allows IQ to mimic treatment learning and find theories that are tailored to a specific domain. Finally, IQ is independent of any given heuristic.

IQ provides us with a set of theories or detectors. This set may contain theories with high or low probabilities of detection or false alarm or other user defined statistics. A new theory set is obtained for each iteration when IQ is executed. Many of these theories are not useful in terms of the domain especially if large combinations of attribute ranges have little worth. Based on a given sort criteria, IQ returns an adjustable amount of theories. These bags of theories are not assessed as a whole, but rather individual detectors are chosen.

One of the most evident advantages of IQ is its ability to be tuned to different heuristics. To emulate treatment learning for example, IQ is simply tuned to sort for lift, which is how TAR assesses its treatments. Theories with high lift are more favorable thus are not culled, ultimately leading to consecutive generations with high lifts. In terms of a more domain specific application such as software metrics, the difference probability of detection and the heuristic effort (percentage of lines of code) provides a meaningful heuristic gauge for test engineers to maximize fault detection while minimizing the number of lines of code read.

IQ also has the ability to add and remove heuristics dynamically. The IQ algorithm is independent of any single heuristics. Although, experimentation indicates that at least the probability of detection and probability of false alarm need to be present to ensure the collection of useful detectors, in the software metric domain. However, in theory, as long as there is not a linear relationship then as little as two heuristics are needed. IQ can function on as little as two heuristics, given that they are not linear in nature, and currently as many as ten simultaneous heuristic values. Linear relationships tends to breakdown the hull culling algorithms by providing too many detectors for subsequent generations.

3.3 Program Structure and Classes

This section focuses on the individual IQ classes and methods. We outline all the primary classes and some of the methods within the program structure.

3.3.1 Design Patterns

Many times in software engineering one part of a design needs to execute an action when another part needs to be updated but not informed of the details of the update. Consider the case of our learner. We don't want our hundreds or even thousands detectors to know about any of the IQ algorithms. But every time we alter or cull our set of detectors we want the detectors to reflect the changes without knowing the details of how the updated information was generated. We can address this situation with an observer pattern [Gamma95].

In this object oriented design pattern the subject contains a list of observers. This list is loaded by a method of subject and then updated by the subject. The updates are then reflected upon all the detectors. Our IQ leaner creates a vector or bag of detectors. These detectors are culled and inserted, but also altered. Alterations happen in the event of cross validation where our curve fitting algorithm averages all *x* data splits and then updates the detectors. In Figure 3.5 our Subject IQ obtains new heuristic values from our cross validation method. Then these values are notified to our detectors, which are in turn updated with the new heuristics.

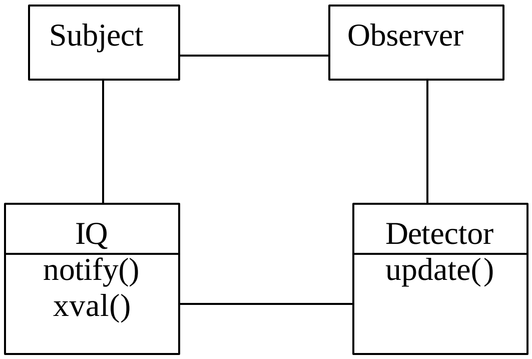


Figure 3.5 Observer pattern

3.3.2 Classes

In this section we will discuss the classes and their relations in our IQ learner. Figure 3.6 illustrates instantiations of IQ's primary classes and their relations. The main IQ class is responsible for instantiating sets of Attribute and Detector classes; as well as an instantiation of a ConvexHull class. The Attribute class instantiates a set of Range classes. And these Range instantiations are referenced instances the Detector class. The following is a list of classes and brief description of their functionality.

IQ: The IQ class is our primary class and is responsible or the IQ iterative cycle seen in Figure 3.10. This class handles all parameters, the command line interface, and output. It is also responsible for detector creation, subset generation for detectors, detector heuristic assessment, detector culling, and cross validation.

Attribute: The Attribute class is responsible for implementation of different discretizations. Attribute also creates a set of ranges which are instantiated according to the type of descretization ordered by the primary IQ class.

ConvexHull: The ConvexHull class inputs bi-polar Cartesian coordinates and returns the coordinates of points located on a hull using the Quickhull algorithm.

Detector: The Detector class holds all heuristic values of an individual detector. This class is essentially a vector class of different attribute ranges all instantiated by the primary IQ class.

Range: Range is an abstract class that must be inherited by a more specific class type. This inheritance we focus on in the next section.

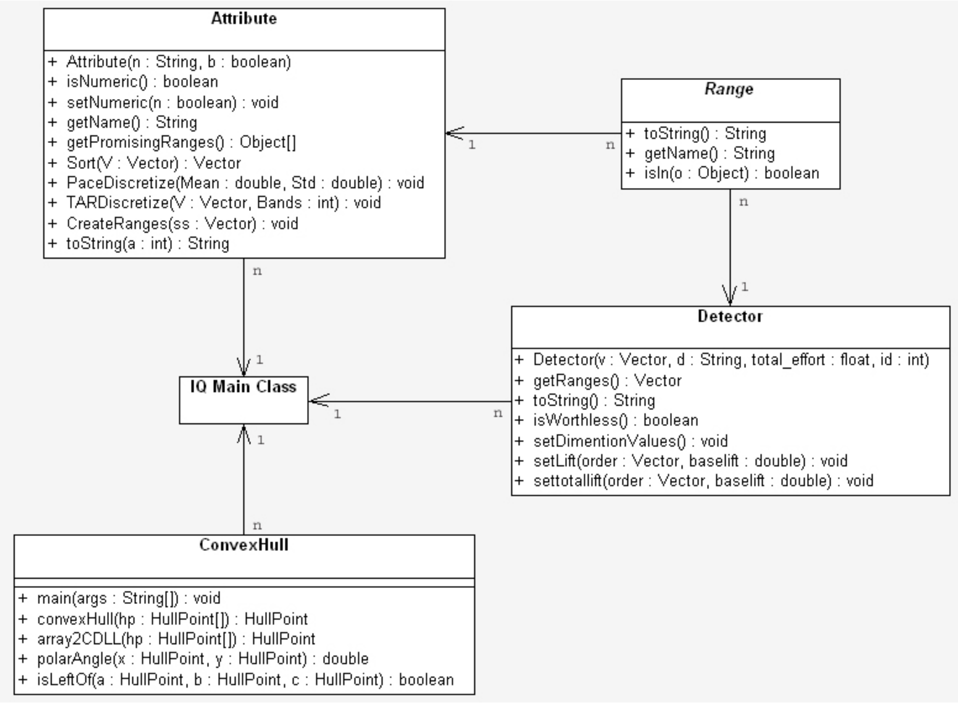

Attribute
+ Attribute(n : String, b : boolean)
+ isNumeric() : boolean
+ setNumeric(n : boolean) : void
+ getName() : String
+ getPromisingRanges() : Object[]
+ Sort(V : Vector) : Vector
+ PaceDiscretize(Mean : double, Std : double) : void
+ TARDiscretize(V : Vector, Bands : int) : void
+ CreateRanges(ss : Vector) : void
+ toString(a : int) : String
Range
+ toString() : String
+ getName() : String
+ isIn(o : Object) : boolean
1
n
n
1
n
1
Detector
+ Detector(v : Vector, d : String, total_effort : float, id : int)
+ getRanges() : Vector
+ toString() : String
+ isWorthless() : boolean
+ setDimentionValues() : void
+ setLift(order : Vector, baselift : double) : void
+ settotallift(order : Vector, baselift : double) : void
IQ Main Class
1
n
1
n
ConvexHull
+ main(args : String[]) : void
+ convexHull(hp : HullPoint[]) : HullPoint
+ array2CDLL(hp : HullPoint[]) : HullPoint
+ polarAngle(x : HullPoint, y : HullPoint) : double
+ isLeftOf(a : HullPoint, b : HullPoint, c : HullPoint) : boolean

Figure 3.6 Primary classes and aggregations

3.3.3 Attribute Inheritance

Every data set contains attributes and each attribute has a set of attribute ranges. How these ranges are represented is determined by or specific discretization policy. In Figure 3.7 we notice that the abstract Range is a generalization that cannot be directly instantiated, but must be instantiated as a more specific Range type associated with a type of discretization and attribute type. This is a form of polymorphism. That is the primary class IQ calls an abstract class type Range if it needs to check if a value exists within a specific range type under any discretization within any detector. This saves additional bottlenecks of needing to refer to each type of Range. The next sections will focus on the algorithms implemented in IQ.

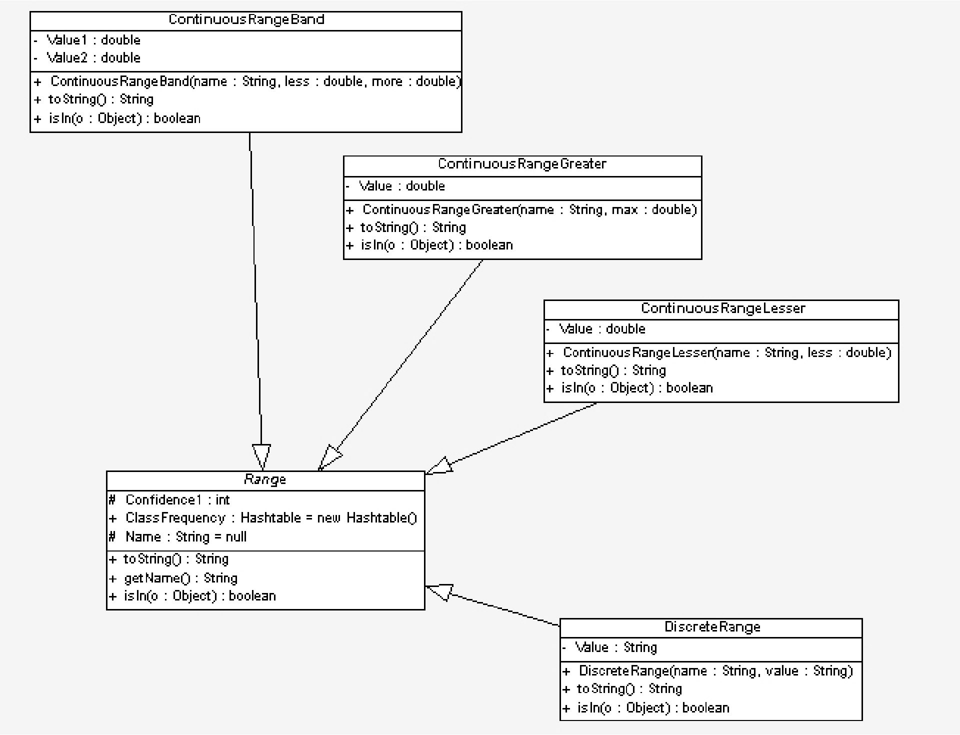

ContinuousRangeBand
- Value1 : double
- Value2 : double
+ ContinuousRangeBand(name : String, less : double, more : double)
+ toString() : String
+ isIn(o : Object) : boolean
ContinuousRangeGreater
- Value : double
+ ContinuousRangeGreater(name : String, max : double)
+ toString() : String
+ isIn(o : Object) : boolean
ContinuousRangeLesser
- Value : double
+ ContinuousRangeLesser(name : String, less : double)
+ toString() : String
+ isIn(o : Object) : boolean
Range
# Confidence1 : int
+ ClassFrequency : Hashtable = new Hashtable()
# Name : String = null
+ toString() : String
+ getName() : String
+ isIn(o : Object) : boolean
DiscreteRange
- Value : String
+ DiscreteRange(name : String, value : String)
+ toString() : String
+ isIn(o : Object) : boolean

Figure 3.7 Attribute range inheritance

3.4 Detailed Description

The following sections describe the details of IQ including imported and developed algorithms.

3.4.1 Discretization and Attribute Ranges

The first stage in IQ is the creation of the detectors, or generation zero. Since many data sets are not discrete but continuous real numbers, they must be discretized under a certain discretization policy. We have two policies, one that assumes a standard Gaussian distribution and discretizes on bases of standard deviation, and the second is a TAR emulation that disceretizes continuous attributes in terms of equal sized buckets. In the case of the standard deviation discretization, the following steps are executed from basic statistics.

- We first compute standard deviation from the square root of the variance.

- Once the standard deviation is calculated for every attribute range, each attribute is discretized on bases of Z values.

- For most datasets we utilize only ten positive Z values: 0, 0.11, 0.25, 0.38, 0.51, 0.67, 0.84, 1.04, 1.28, 1.64, 3.49. These Z values can also be negated if the data set has negative values.

This discretization assumes a discrete standard distribution, which is one of the most evident flaws in the IQ algorithm. The second discretization policy is even simpler:

- All continuous ranges are sorted.

- Then based on a parameter (*bands*), our default is four, the attributes are partitioned into ranges.

- In the case where a repeated series of values lie on the border of a partition, this discretization policy extends the partition to take into account this repeated border value.

Unlike the previous discretization, we have the formation of defined bands as opposed to a single bounded range.

Discreteization from standard distribution creates detectors like the following:

UniqueOperands >= 21.0 & UniqueOperators >= 0

Discreteization from bands creates detectors like the following:

UniqueOperands 21.0...105.0 & UniqueOperators 0.0...1.0

The next section describes the method for generating the internal heuristics used by IQ.

3.4.2 Heuristic Assessment

After the all attribute ranges are created individual heuristics need to be assessed. This section will describe how IQ computes each heuristic.

First, it is important to note that IQ operates with two different modes. The first is without ten way cross validation, and the second is with ten way cross validation. Theses different modes possess varying means of computing heuristics. For example, when computing effort in the sense of ten way cross validation, there is a localized total effort specific for each of ten way run which is maintained as a running computation of the percentage. While not operating under ten way cross validation, the total effort is simply the sum of all efforts. We will focus more on effort in Chapter 4.

The primary heuristics in IQ is the probability of detection the probability of false alarm, confidence and support are all computed from the Figure 2.1. The individual values from Figure 2.1 are obtained though a single pass through the data for each detector. This pass originally occurred once per iteration. However with indexing the data set through of storing indexes and the corresponding class value for every attribute range, we averted the necessity of passing through the entire data set for every iteration of IQ.

When detectors are formulated from our discretization in the first iteration, each detector is given the values from Figure 2.1. Once the *a, b, c,* and *d* values are stored in each detector object then the internal methods in the IQ class compute the values for probability of detection, probability of false alarm, confidence, and support.

Lastly, for the lift computation IQ assess a lift table for every theory, describing the frequency and weight each class. The values from this table are used to compute the lift heuristic found in Chapter 2.

3.4.3 Convex Hull

In order to limit the number of detectors from which we iterate through, we needed a series of culling rules. As mentioned previously ROC space has a designated optimal point at y=1 and x=0. In ROC space the combination of several different types of classifiers are proven more useful than any single classifier [Fawcett98]. This is due to different classifiers dominating different sections of ROC space. So by observing detectors that are not only close to the optimal ROC point but are also in a unique section of ROC space we include detectors that are optimal for each probability of detection and probability of false alarm values. A convex hull is defined as the smallest convex set containing all points in ROC space. Therefore in order to obtain a broad spectrum of detectors for combinations, all detectors on the convex hull are utilized in consecutive iterations.

In obtaining a convex hull, we are not concerned with detectors that lie on the underside of the hull because they are furthest from our optimal point. In Figure 3.5 and Figure 3.6 we observe two different hulls. The first is the convex hull of all detectors in ROC space while the second is the hull with the added points of (0,0) and (1,0) effectively eliminating the underside of the hull thus streamlining the number of detectors used in further iterations. All the culled points possess some value which is more optimal for that section of ROC space.

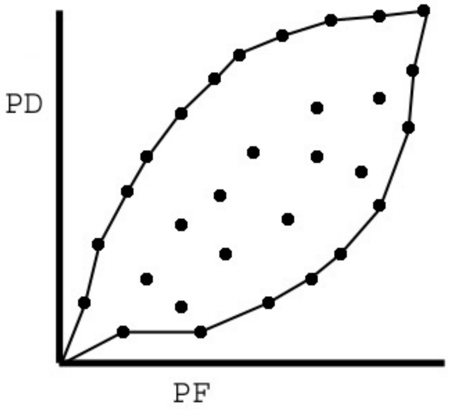


Figure 3.8 a complete convex hull surrounding all points

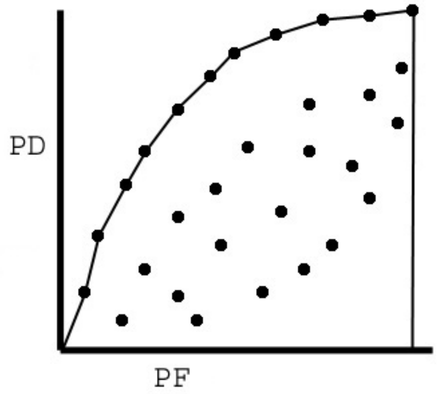


Figure 3.9 a convex hull surrounding all points and conjoining with the x-axis

3.4.4 Quick Hull

In order to quickly obtain the convex hull for IQ we employ the quick hull algorithm. Quick hull is fairly straight forward. Of n-dimensional space quick hull identifies the outer shell of points. The algorithm is as follows:

- For every facet and for each new point if the point is above the facet then assign the point to the hull.

- Additionally, for every facet with a point beyond select the furthest point couple with the facet and for all neighboring points if the point is higher then couple with the facet.

- Points associated with facets are then comprised into ridges.

- For each ridge create a new facet and couple with its neighboring points.

- As a result the set of facets comprise the outside hull.

In terms of points quick hull, starts out by obtaining two points with the furthest distance then adding them the hull. It then "draws" a line connecting these two extreme points. The point furthest from this line is then added to the hull and another line is drawn. This procedure is executed recursively until there are no longer any points remaining on the outside of any ridge.

Quick hull is an algorithm for computing the convex hull of group points in n dimensional space. However the execution time and complexity is reduced in two dimensions. Since we employ a multi dimensional hull for IQ we execute the quick hull algorithm once for every dimensional pairs. For example, if we have a three dimensional space consisting of probability of detection, probability of false alarm, and confidence then we take the convex hull of three different 2-d spaces:

- PF, PD Cartesian plane
- PF, Confidence Cartesian plane
- PD, Confidence Cartesian plane

It is also necessary to scale the dimensions so they all lie within zero to one range. Heuristics such as effort and support, discussed in Chapter 4, are normalized by IQ to lie between zero and one. Also, dimensions are inversely coupled in attempt to produce a hull with a maximum area. That is when probability of detection and confidence are combined the confidence on the y-axis is inverted to produce a hull with the probability of detection on the x-axis.

In the case where there is a linear type relationship and no hull is evident, our culling rules such as the maximum difference between probability of false alarm and probability of detection are used to further cull the space since a linear relation would yield too many detectors.

3.4.5 Beam Search

Beam search is a relatively basic computer science algorithm. Beam search evades the combinatorial explosion problem by expanding a set number of most promising nodes at each level. A heuristic is used to assess which nodes are most likely acceptable. This algorithm then ignores any nodes that are not close to our acceptable goal. Here we use the term node which pertains to an individual detector. Beam search is similar to breadth-first search in that the search progresses by level. However unlike breadth-first search beam search cannot move to a previous level.

Beam search is optimistic compared to other level searches such as best first search and breadth first search. The algorithm, which is slightly modified to obtain a bag of detectors and not just a single detector, for beam search is as follows:

- Form an initial detector queue [Q] consisting of singleton detectors
- Set counter M = 1
- Until [Q] is no longer growing loop
    - If [Q] is no longer growing do nothing
    - If [Q] is larger then and all subsets of size M of all attribute ranges located in [Q]
    - Quick Sort [Q] by heuristic.
    - Remove all but the first N nodes from [Q]
    - Increment M

The above algorithm is simple yet effective. However, when dealing with all combinations of size M the queue of size N is usually less than a fifty to avoid exceeding the memory allocated by the JAVA virtual machine. As an additional parameter to this algorithm we also include a separate initial queue size that allows for more diversity in the initial sampling of attribute ranges ultimately leading to a varied and improved end detector set. The next section will discuss these parameters in greater detail.

3.4.6 Parameters

Parameters for IQ are fed into beam search and Quick hull. Both culling methods share a common array containing all the theories in the current iteration. This array is controlled via two parameters.

The first parameter is the maximum number of theories the array can hold generation zero of IQ. This number is usually in magnitudes of a hundred and is used to allow IQ to sparingly cull generation zero permitting more attribute ranges that meet a tuning criterion to iterate to the next generation.

The second parameter is the maxim array size allowed in all generation is other than generation zero. This parameter is usually between ten and fifty. On data sets that contain many critical attribute ranges it is beneficial to increase this parameter to obtain more attribute ranges for composite theories.

Also, an internal sort criterion must also be provided in order to tune IQ. This tuning criterion is used to quick sort the detector array. This internal criteria may be the difference in probability of detection and probability of false alarm, accuracy etc.

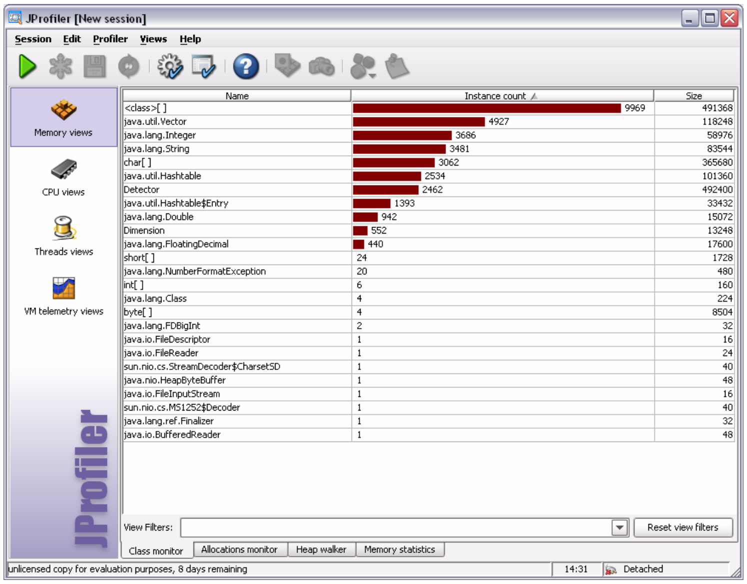
3.4.7 Cross Validation

Cross validation in IQ is not the same as standard machine learning cross validation. IQ identifies each attribute range or combination of attribute ranges as a separate model or theory. Since the model created by IQ is dependent upon the distribution of continuous attribute ranges, when the data set is divided we are faced with the possibility of different theories being created from the possibilities of varying distributions on continuous attribute ranges. This is a problem solely to IQ because IQ requires the same theory in order to assess the corresponding heuristics. If the theory changes even by a small percentage the heuristics values will most likely be altered as well. For example if v(g) <= 10 in one run and is altered due to a different sampling of the data set to v(g) <= 11 then the corresponding heuristics for that detector will not be comparable. Therefore the central IQ model is not recreated for each division in ten way cross validation. But each theory's heuristic values are created through the data division.. Therefore, in every iteration each of IQ's detectors assesses its heuristics through ten way cross validation by plotting ten different heuristic sets for the same detector and then averaging the heuristics.

3.4.8 Profiling Performance

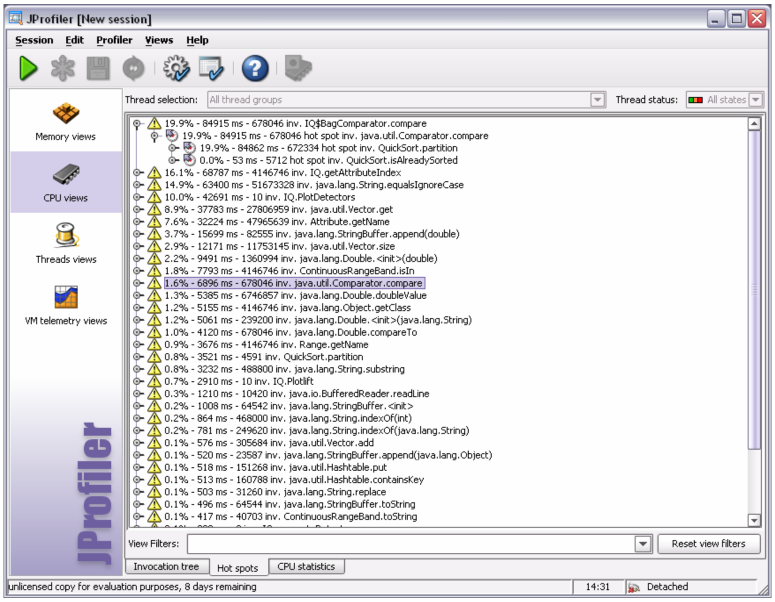
In this section we profile IQ running a NASA metric data set with 520 instances. The purpose is to assess possible performance bottlenecks in the algorithm. We employ a profiler called JProfiler to run a real-time assessment of memory and CPU usage. The first part we focus on memory usage. Based on Figure 3.8 we observed the most of the memory allocated is from Array and Vector class. These classes are standard JAVA classes that are dynamically allocated in IQ to produce attribute ranges and detectors.

In contrast to IQ, WEKA utilizes a specialized vector class called QuickVector that allows for faster searching and sorting. Future improvements to IQ may justify the employment of such classes. As we see in Figure 3.9, twenty percent of the CPU used when running IQ involved BagComparator which is an internal method for comparing detectors in a Vector construct. The next item in Figure 3.9 accounts for sixteen percent and is a function of indexing attribute ranges. These functions if optimized such as in WEKA may provide significant performance enchantments.



Name	Instance count	Size
<class>[ ]	9969	491368
java.util.Vector	4927	118248
java.lang.Integer	3686	58976
java.lang.String	3481	83544
char[ ]	3062	365680
java.util.Hashtable	2534	101360
Detector	2462	492400
java.util.Hashtable$Entry	1393	33432
java.lang.Double	942	15072
Dimension	552	13248
java.lang.FloatingDecimal	440	17600
short[ ]	24	1728
java.lang.NumberFormatException	20	480
int[ ]	6	160
java.lang.Class	4	224
byte[ ]	4	8504
java.lang.FDBigInt	2	32
java.io.FileDescriptor	1	16
java.io.FileReader	1	24
sun.nio.cs.StreamDecoder$CharsetSD	1	40
java.nio.HeapByteBuffer	1	48
java.io.FileInputStream	1	16
sun.nio.cs.MS1252$Decoder	1	40
java.lang.ref.Finalizer	1	32
java.io.BufferedReader	1	48

Figure 3.8 Memory Profile of IQ



```
19.9% - 84915 ms - 678046 inv. IQ$BagComparator.compare
  19.9% - 84915 ms - 678046 hot spot inv. java.util.Comparator.compare
    19.9% - 84862 ms - 672334 hot spot inv. QuickSort.partition
    0.0% - 53 ms - 5712 hot spot inv. QuickSort.isAlreadySorted
16.1% - 68787 ms - 4146746 inv. IQ.getAttributeIndex
14.9% - 63400 ms - 51673328 inv. java.lang.String.equalsIgnoreCase
10.0% - 42691 ms - 10 inv. IQ.PlotDetectors
8.9% - 37783 ms - 27806959 inv. java.util.Vector.get
7.6% - 32224 ms - 47965639 inv. Attribute.getName
3.7% - 15699 ms - 82555 inv. java.lang.StringBuffer.append(double)
2.9% - 12171 ms - 11753145 inv. java.util.Vector.size
2.2% - 9491 ms - 1360994 inv. java.lang.Double.<init>(double)
1.8% - 7793 ms - 4146746 inv. ContinuousRangeBand.isIn
1.6% - 6896 ms - 678046 inv. java.util.Comparator.compare
1.3% - 5385 ms - 6746857 inv. java.lang.Double.doubleValue
1.2% - 5155 ms - 4146746 inv. java.lang.Object.getClass
1.2% - 5061 ms - 239200 inv. java.lang.Double.<init>(java.lang.String)
1.0% - 4120 ms - 678046 inv. java.lang.Double.compareTo
0.9% - 3676 ms - 4146746 inv. Range.getName
0.8% - 3521 ms - 4591 inv. QuickSort.partition
0.8% - 3232 ms - 488800 inv. java.lang.String.substring
0.7% - 2910 ms - 10 inv. IQ.Plotlift
0.3% - 1210 ms - 10420 inv. java.io.BufferedReader.readLine
0.2% - 1008 ms - 64542 inv. java.lang.StringBuffer.<init>
0.2% - 864 ms - 468000 inv. java.lang.String.indexOf(int)
0.2% - 781 ms - 249620 inv. java.lang.String.indexOf(java.lang.String)
0.1% - 576 ms - 305684 inv. java.util.Vector.add
0.1% - 520 ms - 23587 inv. java.lang.StringBuffer.append(java.lang.Object)
0.1% - 518 ms - 151268 inv. java.util.Hashtable.put
0.1% - 513 ms - 160788 inv. java.util.Hashtable.containsKey
0.1% - 503 ms - 31260 inv. java.lang.String.replace
0.1% - 496 ms - 64544 inv. java.lang.StringBuffer.toString
0.1% - 417 ms - 40703 inv. ContinuousRangeBand.toString
```

Figure 3.9 Processor Profiling of IQ

3.4.9 Conclusion

Figure 3.10 best summarizes the iterative cycle of IQ. The detector creation composes the detectors from either descretiztion in generation zero or from attribute ranges decomposed by an earlier iteration. Once we have a set of detectors they are then populated with additional heuristic scores. Such heuristics include probability of false alarm, probability of detection, confidence, lift, support, and others. Then detectors are culled either by a hull culling algorithm such as quick hull or through a beam search which culls by a given heuristic. Finally chosen detectors are decomposed in order to recombine for further iterations. Or if no new detectors are created IQ terminates. The next section observes IQ in action against many popular learning algorithms employed upon varied data sets.

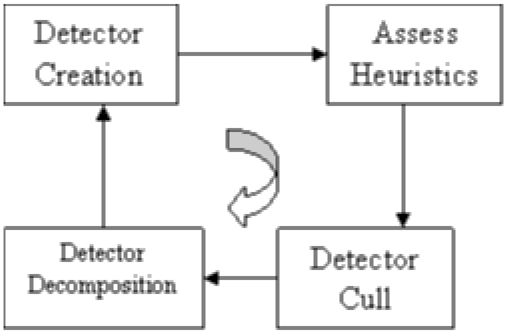


Figure 3.10 Iterative IQ cycle

4 Case Studies

This chapter is divided into several parts. Each section describes a different case study which our IQ algorithms is tested against other learning techniques and validations. As in all our studies we deal with detectors which are hand chosen. In the following section we show the criteria used for this process.

4.1 Choosing a Theory

Throughout all the case studies we generate theories from IQ utilizing different tunings. As discussed previously, IQ produces a bag of theories that are culled either via the quick hull algorithm or through a beam search. Each individual theory has its own assessment heuristics, such as probability of detection and probability of false alarm. Based on these heuristics we hand pick a best theory based on a few prioritized factors.

- The current tuning of IQ
- The difference of PD and PF
- The confidence of the theory
- The size of the theory

For example, two theories produced by IQ are shown in Figure 4.1. The first theory is obtained from a run of IQ when optimized for accuracy. The second theory listed in Figure 4.1 obtained from the same run would be a better detector based on the above criteria. Even though the second theory is composed of more attribute ranges and the confidence almost fifty percent less than the first theory, we chose it on basis of a profoundly better PD.

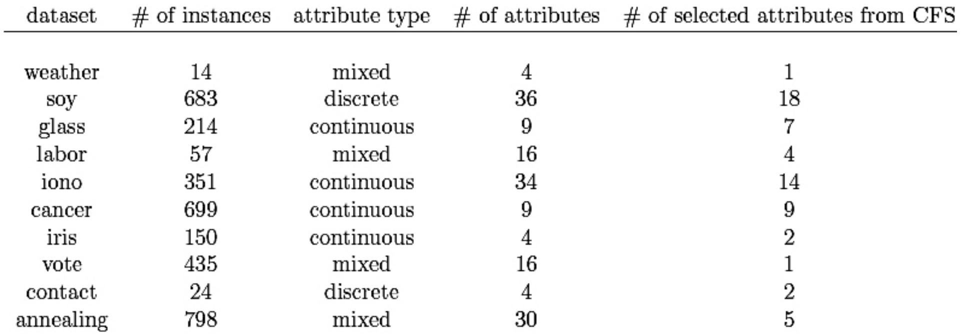
Detector	PD	PF	Confidence	Accuracy
Na>14.09 & Fe=0	**0.22**	**0.22**	**1**	**0.98**
Mg<3.05 & Na =13.50 & Ba<0.37	**1**	**0.034146**	**0.5625**	**0.96729**

Figure 4.1 Sample Detectors from a UC Irvine data set

4.2 Case Study 1: Comparison to machine learners with UC Irvine data

The purpose of this study is to evaluate the performance of IQ against industry standard machine learning algorithms using proven data sets that are diverse in number of attributes, number of instances and number of classes. This study involves running varied data sets obtained from the UC Irvine data mining repository through IQ. Also, the same data sets are run through three other Weka learners, J4.8, Naïve Bayes, and One R.

The data sets chosen also have inherent differences in complexity and correlation between attributes. For example, Hall's experiments reported a minimal decrease in attributes after feature subset selection techniques were employed on the UC Irvine data set soybean. As a result, it can be deduced that most of the attributes are important to classification. Figure 4.2 illustrates basic properties concerning the data sets used. Figure 4.2 also includes feature subset selection information from CFS. This is just to act as a quick indicator of the number of relevant attributes. In order to obtain an accurate description several feature subset selection methods should be applied to the data sets [Hall98].



dataset	# of instances	attribute type	# of attributes	# of selected attributes from CFS
weather	14	mixed	4	1
soy	683	discrete	36	18
glass	214	continuous	9	7
labor	57	mixed	16	4
iono	351	continuous	34	14
cancer	699	continuous	9	9
iris	150	continuous	4	2
vote	435	mixed	16	1
contact	24	discrete	4	2
annealing	798	mixed	30	5

Figure 4.2 UC Irvine data set properties

4.2.1 Method

The first step in data mining after obtaining data sets and machine learners is to format the data. Our data in this case study was formatted in to two types. First we formatted the data sets for the WEKA machine learners by placing the data in a csv file (comma separated values). The csv file contains rows containing data instances and columns containing data attributes while the first row in the file contains the attribute names. The other format is similar but instead of a heading row in the csv file, we include an additional file describing attribute names, types and additional parameters used for IQ.

Once the data sets are formatted they are run through the various learners. In the WEKA environment, the standard WEKA output contains a wealth of statistics pertaining to the performance of the model generated by the learner on a class basis. Useful statistics that we extracted are probability of detection, probability of false alarm, precision, accuracy, execution time. In terms of the WEKA learners, all learners were run under ten way cross validation using the default learner's parameters. Furthermore, all learner runs were conducted on the same Windows XP machine utilizing the same JAVA virtual machine.

Unlike other learners, IQ must be tuned to identify theories that best fit a desired heuristic. In this case study IQ is tuned for accuracy and the difference of probability of detection and probability of false alarm. Tuning is referred to the sort criteria of the detector list in IQ. In order to obtain strong detectors based on a variety of heuristic assessments we produced five tunings for this case study:

- beam search sorted by accuracy
- beam search sorted by the difference of pd and pf
- quick hull culled beam search sorted by accuracy
- quick hull culled beam search sorted by difference of pd and pf
- quick hull culled beam search sorted by difference of pd and pf with a reduced heuristics set

Finally, we have a designation for each learner depending on the parameters used.

- Slow : initial size of detector array is 200 and iterative samples are 50
- Mid: initial size of detector array: 100 and iterative samples are 20
- Fast: initial size of detector array: 50 and iterative samples are 10

The *fast* parameters produce very inferior detectors even through they ran very quickly. Consequently, all IQ tunings with *fast* parameters were omitted from this case study.

4.2.2 Information Gathering

Through the very nature of IQ we obtain a set of heuristics for every detector produced. Therefore in order to accurately compare IQ to other learning algorithms, a set of common statistical values need to be obtained. In this case, the learnt theory is assessed on basis of the probability of false alarm, the probability of detection, accuracy, and confidence. Furthermore, the size of the theory and the total learner execution time is also considered.

Since IQ produces theories on a class basis, other learning algorithms must also be compared in the same fashion. Figure 4.3 displays table heading used in gathering the results. Based on Figure 4.3 the table was populated with all learners run on all data sets for every class value. The conjunction between learner, data set, and class is a unique identifier for every instance. The end result is a fusion of all results for all learners.

An important caveat to note in this case study certain learners such as J4.8 and Naïve Bayes produce a classifier and is difficult to compare directly to a simple IQ detector. Therefore, the detector attribute from Figure 4.3 contains summary information in instances where the learner output cannot be compared directly to IQ. Such summary would be the number of nodes and tree levels describing a J4.8 decision tree. All the heuristics gathered from all the machine learner runs leads to a large amount of information that must be summarized for quick comparisons. The next section will explain how we summarize the data from this case study.

Detector	pd	pf	confidence	accuracy	pd - pf	class	learner	Data set	runtime

Figure 4.3 result table heading

4.2.3 Information Summary

Once the data is collected, comparisons between learners become problematic due to the amount data. In this case study there were over three hundred instances. To amend our problem we produced a summary of the space produced on basis of win/loss percentile of each learning schema per class and heuristic and a t-test to assess the differences values produced by the learners.

Figure 4.4 is the summary of results of this case study. Through this summary it becomes evident which learners lost or won in comparison to all other learners. In terms of probability of false alarm and runtime a win is measured by the least value. The format for the summary is as follows. Each heuristic is divided into two components, losses, and wins. Losses represent the percentage of losses compared to all other schemas, therefore a lower value is optimal. And wins are simply the percentage of wins compared to other learning schemas. The highest and lowest scores in each row are highlighted with green and red. For a loss, a higher scalar value is a shortfall signifying that a learner produced the worst heuristic value and therefore would be highlighted in red. The following are rules for highlighting:

- RED if the heuristic value in a win row is least
- RED if the heuristic value in a loss row highest
- GREEN if a heuristic value in a win row is highest
- GREEN if a heuristic value in a loss row is least

Furthermore, we added an adjustable error margin that was varied between one percent and five percent for each heuristic in determining a win or loss. This margin proved negligible on the final results, which indicates that results from the learners were either equivalent or are apart by a factor of more than fiver percent.

Also, this summary lacks information as to the range of heuristic values we are dealing with and how they compare to one another. If there is there is a win across a row in Figure 4.4 it may only be a detector with that heuristic as a win.

UC IRVINE
DATASETS

heuristics	Beam slow sorted by acc	Beam slow sorted by pd-pf	IQ 6d slow sort by acc	IQ 6d slow sorted by pd-pf	IQ 3d mid sorted by pd-pf	Naïve Bayes	OneR	J4.8
acc loss	2	8	3	8	15	44	66	36
acc win	35	27	33	26	21	13	6	25
conf loss	9	19	6	17	25	23	65	17
conf win	36	27	37	24	19	32	12	39
pd–pf loss	18	8	28	8	15	20	67	21
pd–pf win	22	31	16	29	23	32	10	35
pd loss	22	8	30	6	3	26	66	30
pd win	18	23	12	22	24	20	7	24
pf loss	11	23	9	21	32	29	37	31
pf win	40	29	40	27	21	33	31	36
runtime loss	79	67	77	65	49	1	0	14
runtime win	7	15	8	18	40	77	77	71

Figure 4.4 Results from Case Study 1

Results are in percentiles of detector's heuristic over all other detector heuristics in the for the same data set – class – learner instance

Therefore, the theories found are not necessarily useful or the best. For example, IQ beam sorted on accuracy produces the theory in data set soybean seen in Figure 4.5. From the Figure 4.5 the probability of detection of IQ is half of that of that of Naïve Bayes. Consequently, the classifier produced by Naïve Bayes when predicting the same class is superior in the probability of detection over IQ even though the accuracy of IQ is slightly higher, about six percent. This difference is greater than our five percent error margin constituting a win in terms of IQ and a loss in terms of Naïve Bayes. This phenomenon is repeated for other heuristics and thus compels us to employ t-tests on our result sets. We do t-tests to assess statistical similarities of means of paired populations across heuristic value ranges in order to give us some indication if we are dealing with analogous data or not. The next sections will reveal results from this case study.

IQ: Beam Search sorted by accuracy:

plant-growth = abnormal & seed-tmt = fungicide & area-damaged = scattered & Bacterial Blight germination = 80-89 & shriveling = absent → Bacterial Blight

Learner	**Pd**	**Pf**	**Confidence**	**Accuracy**	**Pd-Pf**
IQ: Beam	**0.5**	**0**	**1**	**0.983050823**	**0.5**
Naïve Bayes	**1**	**0.003**	**0.909**	**0.926794**	**0.997**

Figure 4.5 Sample Detectors from two learners

4.2.4 Analysis from Summary Space

The following section is an in depth examination of each heuristic and the how the learners compared from Figure 4.4. In Figure 4.4, a division separates the IQ tuning on the left from the standard WEKA learners on the right for easy comparisons.

4.2.4.1 Accuracy

For the accuracy heuristic we notice a definite win across most IQ tunings when compared to the standard machine learners. In this case study we tuned IQ twice, once with beam search and once with quick hull culling, for accuracy. This specifically means that IQ's internal detector array sorted by accuracy. IQ beam search when tuned for accuracy returned the most wins and least losses. In terms of Figure 4.4 there is an evident ten percent increase in theories with high accuracies over other learners and a thirty percent decrease in theories with inferior accuracies.

4.2.4.2 Confidence

IQ competed nicely with the other learners only to fall short by a few percent to J4.8 in terms of percentage of best confident theories. Also, IQ produced the fewest worst confidence values. This is significant because even though IQ did not always produce the most confident theory, it did, however, usually produce a confident theory in terms of the other learners.

Similar to accuracy, theories with a high confidence value are not always the best in terms of probability of detection and false alarm. A confident theory is a theory in which is has few exceptions in the data set. However, unlike accuracy the case study was conducted without a tuned instance of IQ for confidence.

4.2.4.3 Difference of PD and PF

IQ does not possess the best percentage of highest difference. In fact there is a difference of four percent between the J4.8 and IQ tuned for difference of pd and pf. In spite of this, IQ when tuned for difference of pd and pf still produces the least losses though, with an almost fifteen percent difference below J4.8.

In this case study, IQ was tuned to sort by the difference between the probability of detection and the probability of false alarm. The primary significance of this heuristic is to locate the largest difference of probability of detection and probability of false alarm. As cited earlier a large difference between probability of false alarm and probability of detection would directly contribute toward a better detector in terms of a higher detection rate and a lower false alarm rate.

4.2.4.4 Probability of Detection

The probability of detection is a much closer heuristic than the prior. J4.8 tied IQ for the best percentage of wins. Also, IQ produced far fewer detectors with small values probability of detection, a margin of almost thirty percent.

4.2.4.5 Probability of false alarm

In terms of probability of false alarm J4.8 tied IQ for the best percentage of wins. In fact IQ when tuned for accuracy performed moderately better than J4.8, with a margin of four percent.

4.2.4.6 Runtimes

In reference to run times IQ suffered a straight loss. Since both IQ and WEKA are both programmed in JAVA, it is fair to compare run times. However one must take into account that the learners from the Weka environment have been optimized for speed and includes custom classes such as quick vector to further shorten run times. Conversely, IQ is designed utilizing mainly standard JAVA libraries, to run in an adequate time frame, and is optimized to produce long lists of detectors.

4.2.5 *Student's T-test Procedure*

In order to obtain small sample inferences for the differences of means across our heuristic sets we employ a paired student's t-test. The procedure for the tests is as follows:

- Compute the differences of two population means: $\mu_d = (\mu_{set1} - \mu_{set1})$

- Assume a an initial null hypothesis that means are equal: $H_0 : \mu_d = 0$

- Use two tailed test with hypothesis that the population means can be greater or less then each other: $H_a : \mu_d$? 0 *or* $H_a : \mu_d < 0$ or $\mu_d > 0$

- Compute test statistic: $t = \overline{d} /(s_d \sqrt{n})$ where d = the mean of sample differences, n = number of paired differences, and s_d = differences of standard deviations from the two populations.

- We set the confidence coefficient (1-a)=95% OR a = 0.05

- We reject H_0 if the p-value < a, meaning that we have determined that there is a significant difference between the means of our two heuristic sets from different learners.

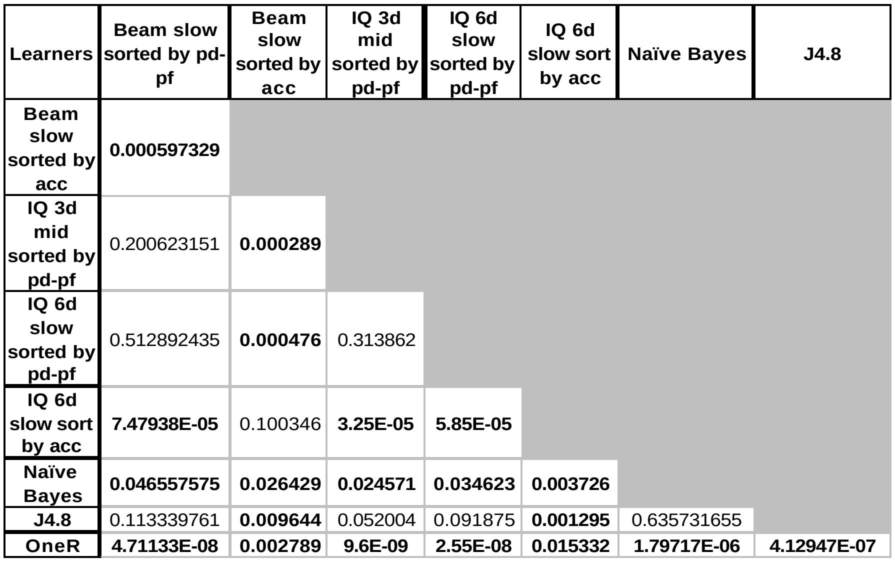
4.2.6 *Student's T-test Results*

The following figures are the results from the t-tests across all for all heuristics analyzed. A bold number represents a probability associated with a Student's paired t-Test, with a two-tailed distribution that was rejected by our hypothesis of equal means across population.

4.2.6.1 PD t-test Results

Figure 4.6 reveals results from our t-test for the heuristic probability of detection across all learners. For PD paired population means we conclude:

- OneR is not similar to any other learner
- There are similarities with in the same IQ sorting criterion
- Naïve Bayes is only similar to J4.8
- J4.8 is dissimilar to IQ tuned to accuracy



Learners	Beam slow sorted by pd-pf	Beam slow sorted by acc	IQ 3d mid sorted by pd-pf	IQ 6d slow sorted by pd-pf	IQ 6d slow sort by acc	Naïve Bayes	J4.8
Beam slow sorted by acc	**0.000597329**						
IQ 3d mid sorted by pd-pf	0.200623151	**0.000289**					
IQ 6d slow sorted by pd-pf	0.512892435	**0.000476**	0.313862				
IQ 6d slow sort by acc	**7.47938E-05**	0.100346	**3.25E-05**	**5.85E-05**			
Naïve Bayes	**0.046557575**	**0.026429**	**0.024571**	**0.034623**	**0.003726**		
J4.8	0.113339761	**0.009644**	0.052004	0.091875	**0.001295**	0.635731655	
OneR	**4.71133E-08**	**0.002789**	**9.6E-09**	**2.55E-08**	**0.015332**	**1.79717E-06**	**4.12947E-07**

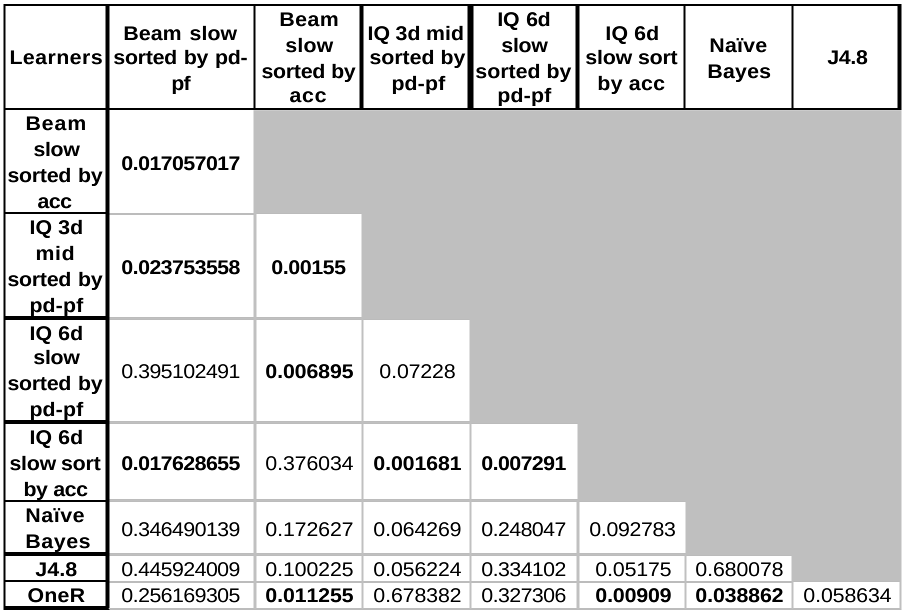
Figure 4.6 T-test results for PD with UC Irvine data

4.2.6.2 PF t-test Results

Figure 4.7 reveals results from our t-test for the heuristic probability of detection across all learners. For PF paired population means we conclude:

- OneR is dissimilar to IQ tuned for accuracy and Naïve Bayes
- There are no similarities within the same IQ sorting criterion
- Naïve Bayes is dissimilar only to OneR

- J4.8 is similar to IQ sorted by accuracy



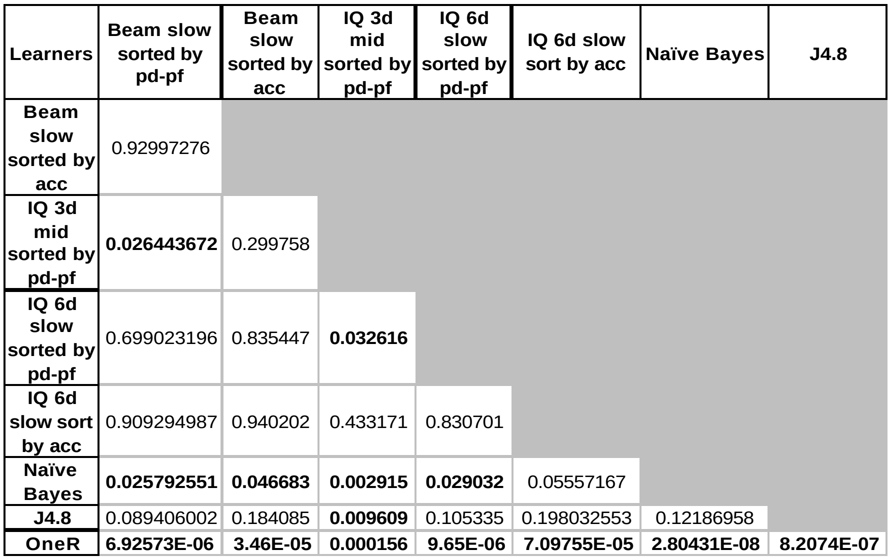
Learners	Beam slow sorted by pd-pf	Beam slow sorted by acc	IQ 3d mid sorted by pd-pf	IQ 6d slow sorted by pd-pf	IQ 6d slow sort by acc	Naïve Bayes	J4.8
Beam slow sorted by acc	**0.017057017**						
IQ 3d mid sorted by pd-pf	**0.023753558**	**0.00155**					
IQ 6d slow sorted by pd-pf	0.395102491	**0.006895**	0.07228				
IQ 6d slow sort by acc	**0.017628655**	0.376034	**0.001681**	**0.007291**			
Naïve Bayes	0.346490139	0.172627	0.064269	0.248047	0.092783		
J4.8	0.445924009	0.100225	0.056224	0.334102	0.05175	0.680078	
OneR	0.256169305	**0.011255**	0.678382	0.327306	**0.00909**	**0.038862**	0.058634

Figure 4.7 T-test results for PF with UC Irvine data

4.2.6.3 Confidence t-test Results

Figure 4.8 reveals results from our t-test for the heuristic probability of detection across all learners. For confidence paired population means we conclude:

- OneR is dissimilar to all
- IQ 3d is dissimilar to IQ and Beam sorted by PD-PF
- Naïve Bayes is dissimilar OneR and IQ sorted by accuracy
- J4.8 is dissimilar to IQ 3d and OneR



Learners	Beam slow sorted by pd-pf	Beam slow sorted by acc	IQ 3d mid sorted by pd-pf	IQ 6d slow sorted by pd-pf	IQ 6d slow sort by acc	Naïve Bayes	J4.8
Beam slow sorted by acc	0.92997276						
IQ 3d mid sorted by pd-pf	**0.026443672**	0.299758					
IQ 6d slow sorted by pd-pf	0.699023196	0.835447	**0.032616**				
IQ 6d slow sort by acc	0.909294987	0.940202	0.433171	0.830701			
Naïve Bayes	**0.025792551**	**0.046683**	**0.002915**	**0.029032**	0.05557167		
J4.8	0.089406002	0.184085	**0.009609**	0.105335	0.198032553	0.12186958	
OneR	**6.92573E-06**	**3.46E-05**	**0.000156**	**9.65E-06**	**7.09755E-05**	**2.80431E-08**	**8.2074E-07**

Figure 4.8 T-test results for confidence with UC Irvine data

4.2.6.4 Accuracy t-test Results

Figure 4.9 reveals results from our t-test for the heuristic probability of detection across all learners. For accuracy paired population means we conclude:

- OneR is dissimilar to all
- IQ and Beam sorted for accuracy are dissimilar to all but themselves
- Naïve Bayes is dissimilar to all except IQ 3d
- J4.8 is dissimilar to all

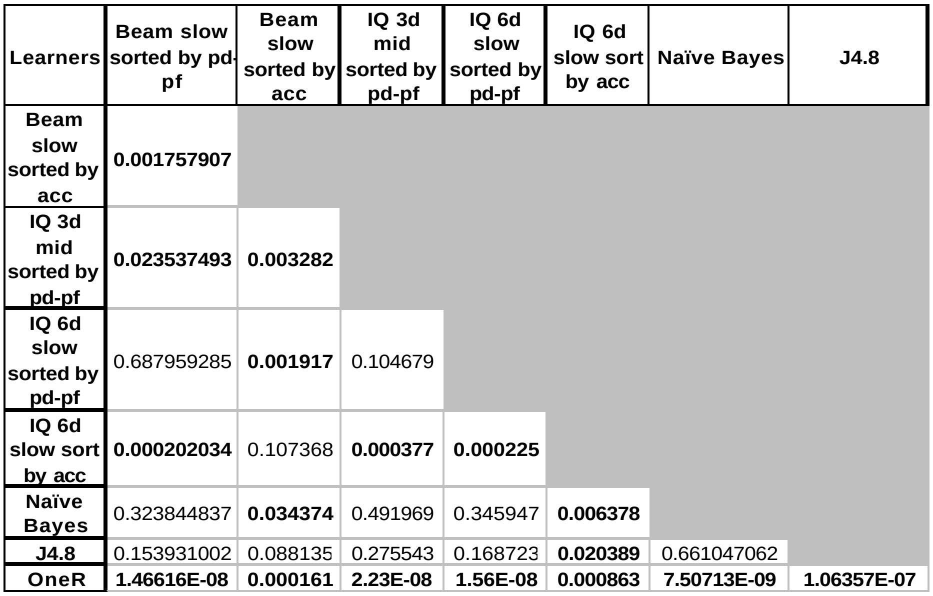
Learners	Beam slow sorted by pd-pf	Beam slow sorted by acc	IQ 3d mid sorted by pd-pf	IQ 6d slow sorted by pd-pf	IQ 6d slow sort by acc	Naïve Bayes	J4.8
Beam slow sorted by acc	0.012659933						
IQ 3d mid sorted by pd-pf	**0.029472667**	**0.002517**					
IQ 6d slow sorted by pd-pf	0.80862104	**0.01126**	0.060935				
IQ 6d slow sort by acc	**0.014741975**	0.524481	**0.002865**	**0.012997**			
Naïve Bayes	**0.022157928**	**4.36E-05**	0.148298	**0.026907**	**2.27712E-05**		
J4.8	**0.002816088**	**6.1E-05**	**0.011287**	**0.003461**	**4.07251E-05**	**0.032193**	
OneR	**2.7222E-11**	**1.31E-12**	**1.95E-10**	**3.76E-11**	**5.5245E-13**	**1.44E-10**	**1.54171E-07**

Figure 4.9 T-test results for accuracy with UC Irvine data

4.2.6.5 PD-PF t-test Results

Figure 4.10 reveals results from our t-test for the heuristic probability of detection across all learners. For PD-PF paired population means we conclude:

- OneR is dissimilar to all
- IQ and Beam sorted for accuracy are dissimilar to all but themselves
- Naïve Bayes is dissimilar to all except IQ and Beam sorted by accuracy
- J4.8 is dissimilar to OneR and IQ6d sorted by accuracy



Learners	Beam slow sorted by pd-pf	Beam slow sorted by acc	IQ 3d mid sorted by pd-pf	IQ 6d slow sorted by pd-pf	IQ 6d slow sort by acc	Naïve Bayes	J4.8
Beam slow sorted by acc	**0.001757907**						
IQ 3d mid sorted by pd-pf	**0.023537493**	**0.003282**					
IQ 6d slow sorted by pd-pf	0.687959285	**0.001917**	0.104679				
IQ 6d slow sort by acc	**0.000202034**	0.107368	**0.000377**	**0.000225**			
Naïve Bayes	0.323844837	**0.034374**	0.491969	0.345947	**0.006378**		
J4.8	0.153931002	0.088135	0.275543	0.168723	**0.020389**	0.661047062	
OneR	**1.46616E-08**	**0.000161**	**2.23E-08**	**1.56E-08**	**0.000863**	**7.50713E-09**	**1.06357E-07**

Figure 4.10 T-test results for PD-PF with UC Irvine data

4.2.7 Conclusion

To conclude this case study we produced Figure 4.11, which is a quick representation of Figure 4.4. In the last column we find the percent of runs that IQ performed better. We have three deficiencies, confidence, PD-PF, and runtimes. We also have tie in PD. And two marginal (ten percent or less) wins in accuracy and PF. Bases on information from Figure 4.4, Figure 4.11 and the t-test results we conclude the following.

- For accuracy, IQ (specifically tunings for accuracy) faired better than standard learners and possessed an overall significantly better mean.

- For confidence J4.8 faired better than OneR and Naïve Bayes learners but did not possess a significantly better mean than most IQ tunings (except for IQ 3d).

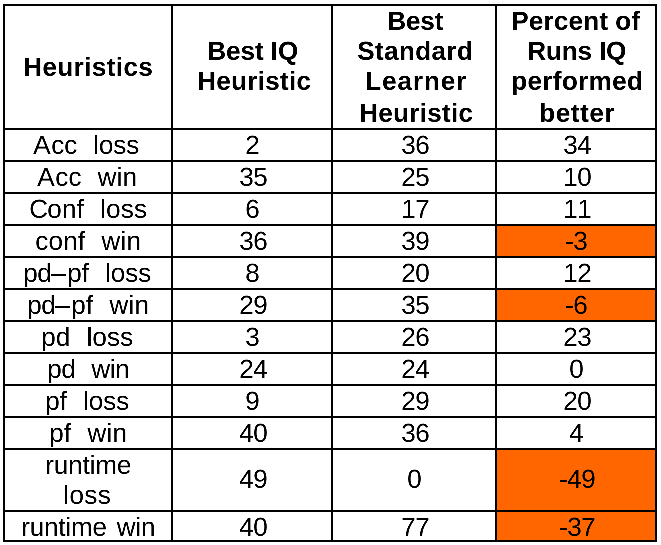
- For PD-PF, J4.8 faired better than OneR and Naïve Bayes learners but did not possess a significantly better mean than IQ tuned for PD-PF.

- For PD, J4.8 and IQ tuned for PD-PF faired better than OneR but did not possess a significantly better mean than Naïve Bayes.

- For PF, IQ tuned for accuracy faired better than OneR and Naïve Bayes learners but did not possess a significantly better mean than J4.8.

- And for runtimes, OneR performed the best while IQ was significantly worse.

The performance of One R desires special focus. In almost all heuristics One R performs the worst. As mentioned previously One R is a very simple learner that focuses on only a single attribute. Most of the UC Irvine attributes within the chosen data sets do not possess a high correlation thus depend on multiple attributes in order to produce a sound theory. In the next case study we focus on data sets with highly correlated data.



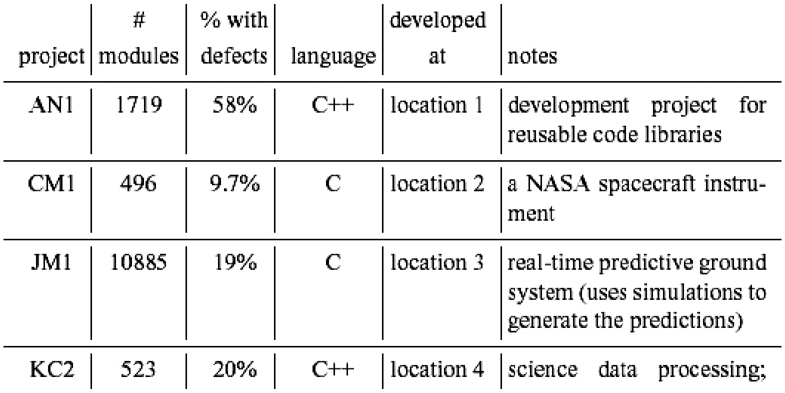
Heuristics	Best IQ Heuristic	Best Standard Learner Heuristic	Percent of Runs IQ performed better
Acc loss	2	36	34
Acc win	35	25	10
Conf loss	6	17	11
conf win	36	39	-3
pd–pf loss	8	20	12
pd–pf win	29	35	-6
pd loss	3	26	23
pd win	24	24	0
pf loss	9	29	20
pf win	40	36	4
runtime loss	49	0	-49
runtime win	40	77	-37

Figure 4.11 Win results for Case Study 1

4.3 Case Study 2: Comparison to machine learners using Metric data

This case study is very similar to the UC Irvine case study in that Weka learners are compared to various IQ tunings. The differences lie in the data sets, which are NASA software metrics that have several hundred to several thousand instances of highly correlated attributes. Every instance is a collection of Halstead and McCabe Metrics composed from a single method or function. The class values of these data sets indicate whether or not a module contained a software defect of some sort that was identified by a test engineer. Figure 4.6 describes the some basic statistics of the four metric data sets.

The purpose of this case study is to compare learners when they are employed upon data sets that possess high correlation of attributes. In the case studies performed in [Menzies03] it is revealed that most of these data sets can be reduced from twenty-one attributes to less than five when processed through various feature subset selection techniques. Subsequently, it is of interest to assess the performance of IQ and other learners upon such a flat and repetitive space presented in the software metric data for this case study as a contrast to the previous case study. Also, as our pervious case study the method and summary are consistent with this case study.



project	# modules	% with defects	language	developed at	notes
AN1	1719	58%	C++	location 1	development project for reusable code libraries
CM1	496	9.7%	C	location 2	a NASA spacecraft instrument
JM1	10885	19%	C	location 3	real-time predictive ground system (uses simulations to generate the predictions)
KC2	523	20%	C++	location 4	science data processing;

Figure 4.12 NASA Data set info

4.3.1 Method

The method in this case study is consistent with that of Section 4.2.1 except that we utilize NASA metric data sets instead of UC Irvine data.

4.3.2 Information Gathering

See Section 4.2.2

4.3.3 Information Summary

See Section 4.2.3

4.3.4 Analysis from Summary Space

The following section is an in depth examination of each heuristic and the how the learners compared.

4.3.4.1 Accuracy

Like the pervious case study, when IQ is tuned for accuracy the theories obtained contain a significantly higher accuracy than other learners. In terms of Figure 4.5 there is about a ten percent increase in theories with high accuracies over other learners and a ten percent decrease in theories with inferior accuracies.

4.3.4.2 Confidence

Unlike the pervious case study, IQ faired slightly better with confidence. J4.8 leads in confidence but is closely followed by IQ sorted by accuracy. And IQ leads in least cases of low confidence detectors by a threshold of ten percent.

4.3.4.3 Difference of PD and PF

In terms of the difference of probability of detection and probability of false alarm, IQ when tuned by this difference, achieved higher percentage wins and lower percentage losses over any other learner. Interestingly enough, IQ tuned for accuracy obtained the least percentage of wins. Furthermore, IQ had a large margin of over forty percent fewer detectors with a low value in this heuristic.

4.3.4.4 Probability of detection

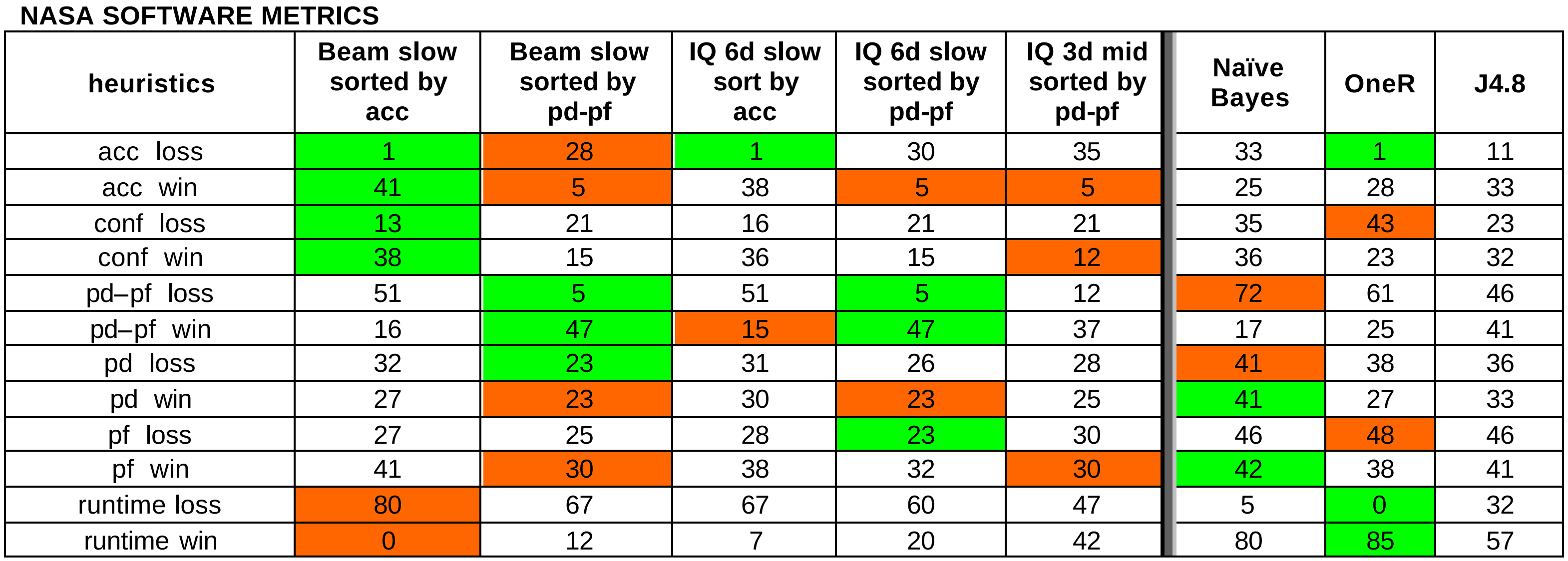
Naïve Bayes had the highest percentage wins in terms of the probability of detection in this case study. While IQ sorted by accuracy has the fewest detectors with high probability of detection.

4.3.4.5 Probability of false alarm

Like the probability of detection, Naïve Bayes had the highest percentage of wins with the heuristic of probability of false alarm. Naïve Bayes is ahead by a single percentage to IQ sorted on accuracy. IQ also has the least losses, by a margin of almost twenty percent.

4.3.4.6 Runtime

Again in this case study, IQ performed the worst in terms of runtime and OneR performed the best.



NASA SOFTWARE METRICS

heuristics	Beam slow sorted by acc	Beam slow sorted by pd-pf	IQ 6d slow sort by acc	IQ 6d slow sorted by pd-pf	IQ 3d mid sorted by pd-pf	Naïve Bayes	OneR	J4.8
acc loss	1	28	1	30	35	33	1	11
acc win	41	5	38	5	5	25	28	33
conf loss	13	21	16	21	21	35	43	23
conf win	38	15	36	15	12	36	23	32
pd–pf loss	51	5	51	5	12	72	61	46
pd–pf win	16	47	15	47	37	17	25	41
pd loss	32	23	31	26	28	41	38	36
pd win	27	23	30	23	25	41	27	33
pf loss	27	25	28	23	30	46	48	46
pf win	41	30	38	32	30	42	38	41
runtime loss	80	67	67	60	47	5	0	32
runtime win	0	12	7	20	42	80	85	57

Figure 4.13 Results from Case Study 2

Results are in percentiles of detector's heuristic over all other detector heuristics in the for the same data set – class – learner instance

4.3.5 Student's T-test Procedure

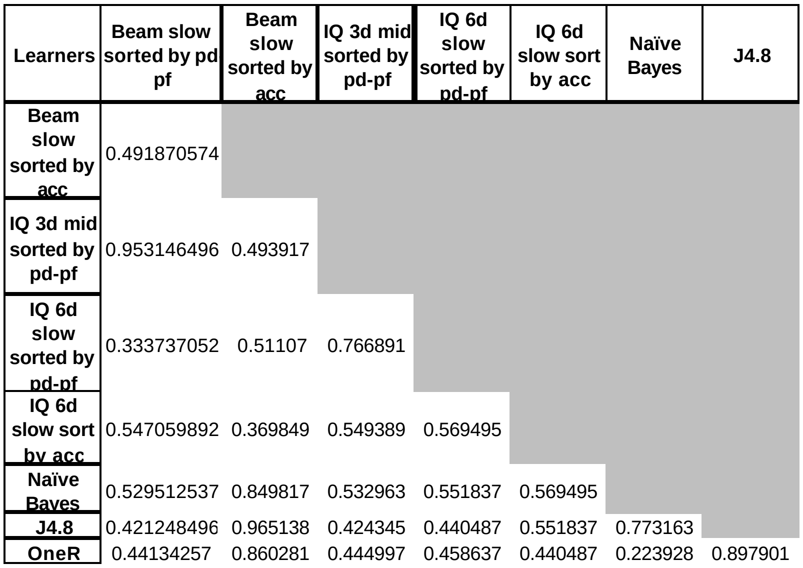
See Section 4.2.5

4.3.6 Student's T-test Results

The following figures are the results from the t-tests across all for all heuristics analyzed. A bold number represents a probability associated with a Student's paired t-Test, with a two-tailed distribution that was rejected by our hypothesis of equal means across population.

4.3.6.1 PD T-test Results

Figure 4.14 reveals results from our t-test for the heuristic probability of detection across all learners. For PD paired population means we cannot conclude any dissimilarity.

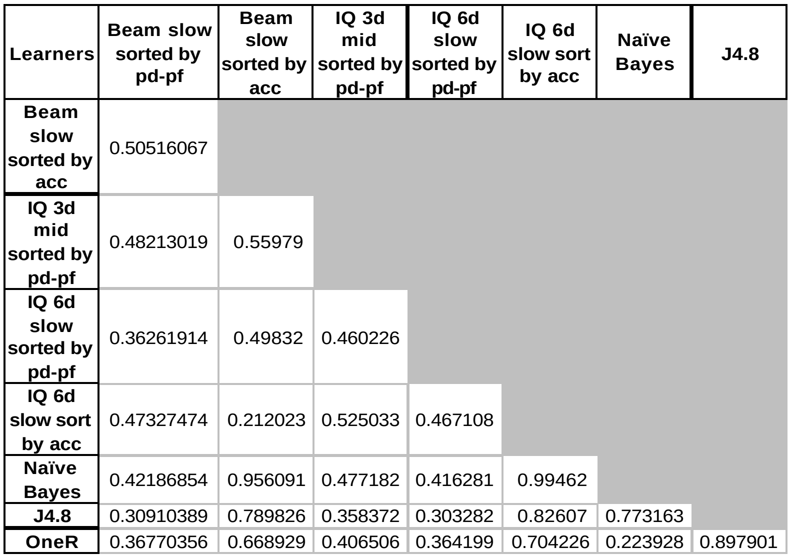


Learners	Beam slow sorted by pd pf	Beam slow sorted by acc	IQ 3d mid sorted by pd-pf	IQ 6d slow sorted by pd-pf	IQ 6d slow sort by acc	Naïve Bayes	J4.8
Beam slow sorted by acc	0.491870574						
IQ 3d mid sorted by pd-pf	0.953146496	0.493917					
IQ 6d slow sorted by pd-pf	0.333737052	0.51107	0.766891				
IQ 6d slow sort by acc	0.547059892	0.369849	0.549389	0.569495			
Naïve Bayes	0.529512537	0.849817	0.532963	0.551837	0.569495		
J4.8	0.421248496	0.965138	0.424345	0.440487	0.551837	0.773163	
OneR	0.44134257	0.860281	0.444997	0.458637	0.440487	0.223928	0.897901

Figure 4.14 T-test results for PD with NASA data

4.3.6.2 PF T-test Results

Figure 4.15 reveals results from our t-test for the heuristic probability of detection across all learners. For PF paired population means we cannot conclude any dissimilarity.

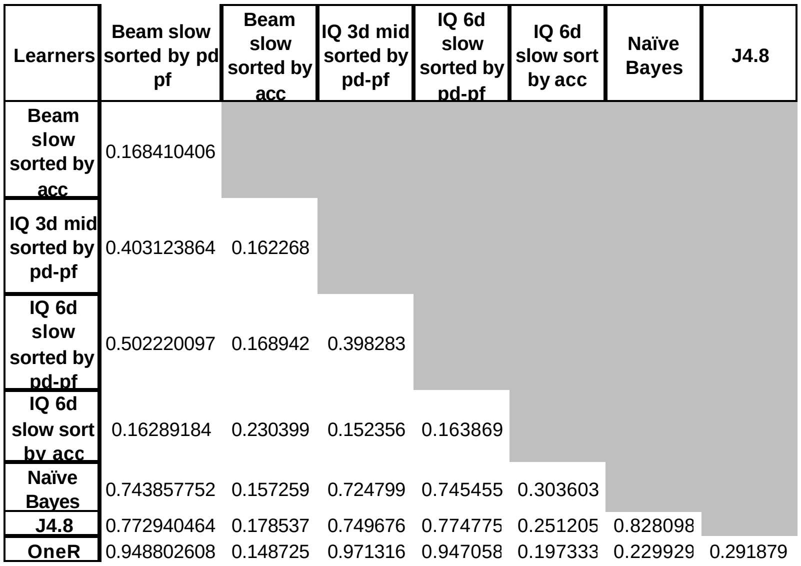


Learners	Beam slow sorted by pd-pf	Beam slow sorted by acc	IQ 3d mid sorted by pd-pf	IQ 6d slow sorted by pd-pf	IQ 6d slow sort by acc	Naïve Bayes	J4.8
Beam slow sorted by acc	0.50516067						
IQ 3d mid sorted by pd-pf	0.48213019	0.55979					
IQ 6d slow sorted by pd-pf	0.36261914	0.49832	0.460226				
IQ 6d slow sort by acc	0.47327474	0.212023	0.525033	0.467108			
Naïve Bayes	0.42186854	0.956091	0.477182	0.416281	0.99462		
J4.8	0.30910389	0.789826	0.358372	0.303282	0.82607	0.773163	
OneR	0.36770356	0.668929	0.406506	0.364199	0.704226	0.223928	0.897901

Figure 4.15 T-test results for PF with NASA data

4.3.6.3 Confidence T-test Results

Figure 4.16 reveals results from our t-test for the heuristic probability of detection across all learners. For confidence paired population means we cannot conclude any dissimilarity.



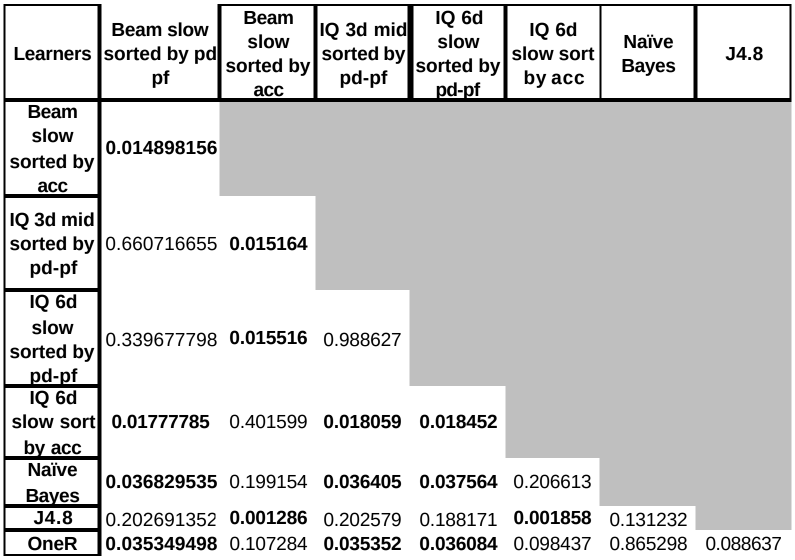
Learners	Beam slow sorted by pd pf	Beam slow sorted by acc	IQ 3d mid sorted by pd-pf	IQ 6d slow sorted by pd-pf	IQ 6d slow sort by acc	Naïve Bayes	J4.8
Beam slow sorted by acc	0.168410406						
IQ 3d mid sorted by pd-pf	0.403123864	0.162268					
IQ 6d slow sorted by pd-pf	0.502220097	0.168942	0.398283				
IQ 6d slow sort by acc	0.16289184	0.230399	0.152356	0.163869			
Naïve Bayes	0.743857752	0.157259	0.724799	0.745455	0.303603		
J4.8	0.772940464	0.178537	0.749676	0.774775	0.251205	0.828098	
OneR	0.948802608	0.148725	0.971316	0.947058	0.197333	0.229929	0.291879

Figure 4.16 T-test results for confidence with NASA data

4.3.6.4 Accuracy T-test Results

Figure 4.17 reveals results from our t-test for the heuristic probability of detection across all learners. For accuracy paired population means we conclude:

- OneR is not similar IQ tuned by PD-PF
- There are similarities within the same IQ sorting criterion
- Naïve Bayes is not similar IQ tuned by PD-PF
- J4.8 is dissimilar to IQ tuned to accuracy



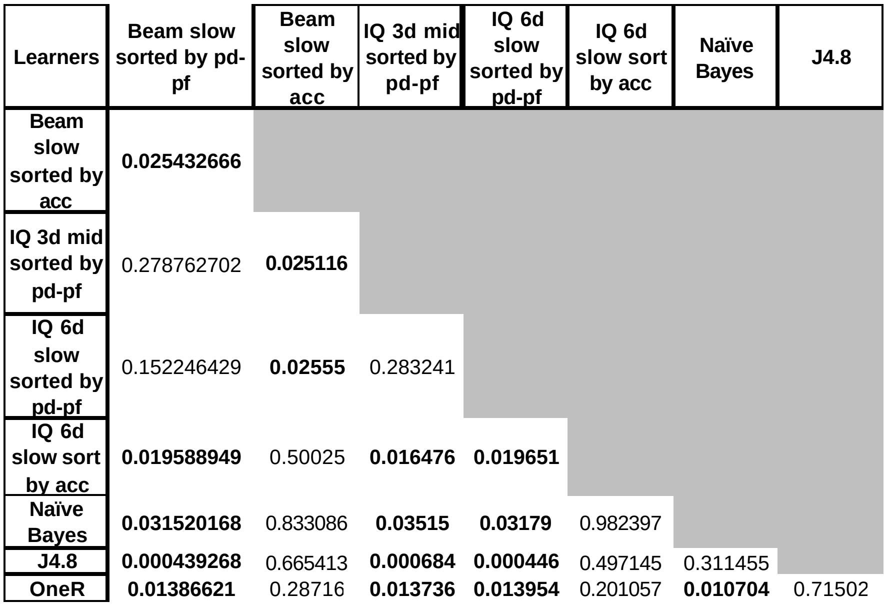
Learners	Beam slow sorted by pd pf	Beam slow sorted by acc	IQ 3d mid sorted by pd-pf	IQ 6d slow sorted by pd-pf	IQ 6d slow sort by acc	Naïve Bayes	J4.8
Beam slow sorted by acc	**0.014898156**						
IQ 3d mid sorted by pd-pf	0.660716655	**0.015164**					
IQ 6d slow sorted by pd-pf	0.339677798	**0.015516**	0.988627				
IQ 6d slow sort by acc	**0.01777785**	0.401599	**0.018059**	**0.018452**			
Naïve Bayes	**0.036829535**	0.199154	**0.036405**	**0.037564**	0.206613		
J4.8	0.202691352	**0.001286**	0.202579	0.188171	**0.001858**	0.131232	
OneR	**0.035349498**	0.107284	**0.035352**	**0.036084**	0.098437	0.865298	0.088637

Figure 4.17 T-test results for accuracy with NASA data

4.3.6.5 PD-PF T-test Results

Figure 4.18 reveals results from our t-test for the heuristic probability of detection across all learners. We for PD paired population means we conclude:

- OneR is not similar IQ tuned by PD-PF and Naïve Bayes
- There are similarities within the same IQ sorting criterion
- Naïve Bayes is not similar IQ tuned by PD-PF or OneR
- J4.8 is dissimilar to IQ tuned by PD-PF



Learners	Beam slow sorted by pd-pf	Beam slow sorted by acc	IQ 3d mid sorted by pd-pf	IQ 6d slow sorted by pd-pf	IQ 6d slow sort by acc	Naïve Bayes	J4.8
Beam slow sorted by acc	**0.025432666**						
IQ 3d mid sorted by pd-pf	0.278762702	**0.025116**					
IQ 6d slow sorted by pd-pf	0.152246429	**0.02555**	0.283241				
IQ 6d slow sort by acc	**0.019588949**	0.50025	**0.016476**	**0.019651**			
Naïve Bayes	**0.031520168**	0.833086	**0.03515**	**0.03179**	0.982397		
J4.8	**0.000439268**	0.665413	**0.000684**	**0.000446**	0.497145	0.311455	
OneR	**0.01386621**	0.28716	**0.013736**	**0.013954**	0.201057	**0.010704**	0.71502

Figure 4.18 T-test results for PD-PF with NASA data

4.3.7 Conclusion

To conclude this case study we produced Figure 4.19, which is a quick representation of Figure 4.13. In the last column we find the percent of runs that IQ performed better. We have three deficiencies, PD, PF, and runtimes. We also have tie in accuracy. And two marginal (ten percent or less) wins in confidence and PD. Based on information from Figure 4.13, 4.19 and the t-test results we conclude the following.

- For accuracy, IQ (specifically tunings for accuracy) faired better than J4.8 but did not possess an overall significantly better mean than Naïve Bayes or OneR.

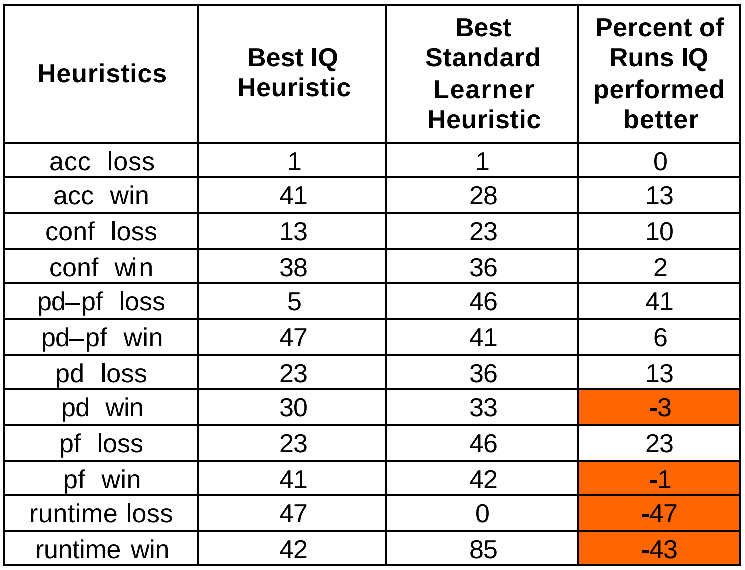
- For confidence, no learner had a significantly different mean than any other learner.

- For PD-PF, IQ sorted by PD-PF significantly better mean other learners.

- For PD, no learner had a significantly different mean than any other learner.

- For PF, no learner had a significantly different mean than any other learner.

- And for runtimes OneR performed the best while IQ was significantly worse.



Heuristics	Best IQ Heuristic	Best Standard Learner Heuristic	Percent of Runs IQ performed better
acc loss	1	1	0
acc win	41	28	13
conf loss	13	23	10
conf win	38	36	2
pd–pf loss	5	46	41
pd–pf win	47	41	6
pd loss	23	36	13
pd win	30	33	-3
pf loss	23	46	23
pf win	41	42	-1
runtime loss	47	0	-47
runtime win	42	85	-43

Figure 4.19 Win results in Case Study 2

4.4 Case Study 3: Comparison to ROCKY using NASA Metric data

This case study compares and contrasts the ROCKY algorithm to that of IQ. IQ's first iteration is ROCKY. Therefore, this study evaluates the merits of theories composing of merely one attribute to that of composite theories in the software metric domain. We also introduce a domain specific heuristic called effort, which we describe next.

4.4.1 Effort heuristic

Domains can vary greatly on the type of data sets that are extracted and the attributes which are deemed more critical through a business or operational perspective. In the case of software defect detection, our code metrics are not taken at their surface values. Metrics such as LOC can be translated into the effort that a test engineer must produce. In this domain effort is defined as

$$\text{Effort} = {}^{\text{(Total LOC found containing a detector)}}/_{\text{(Total LOC of the data set)}}$$

For example, if a detector found via IQ states that on a specified data set a detector with a relatively high pd of seventy percent and low probability of false alarm is coupled with a low effort of twenty percent then this can be translate into the test engineer finding seventy percent of the errors in a project by viewing only twenty percent of the code all with an error rate of thirty percent if this detector is employed as a test for defective code. There is also the argument that since code with potential defects is checked anyways then the probability of false alarm is negligible in this domain. For more information on effort and V&V see [Menzies03].

4.4.2 Method

The method for this case study is simplistic. We run IQ across three metric data sets, jm1, cm1, and kc2. An1 is omitted due to its lack of similar attributes and a clearly defined LOC attribute needed for the effort heuristic. IQ is tuned to sort by PD-PF and then PD-effort. The best three detectors are recorded under those sort criteria are recorded. In this case study we record the full detector taken from IQ for a non-zero fault class as seen in Figure 4.15.

4.4.3 Results

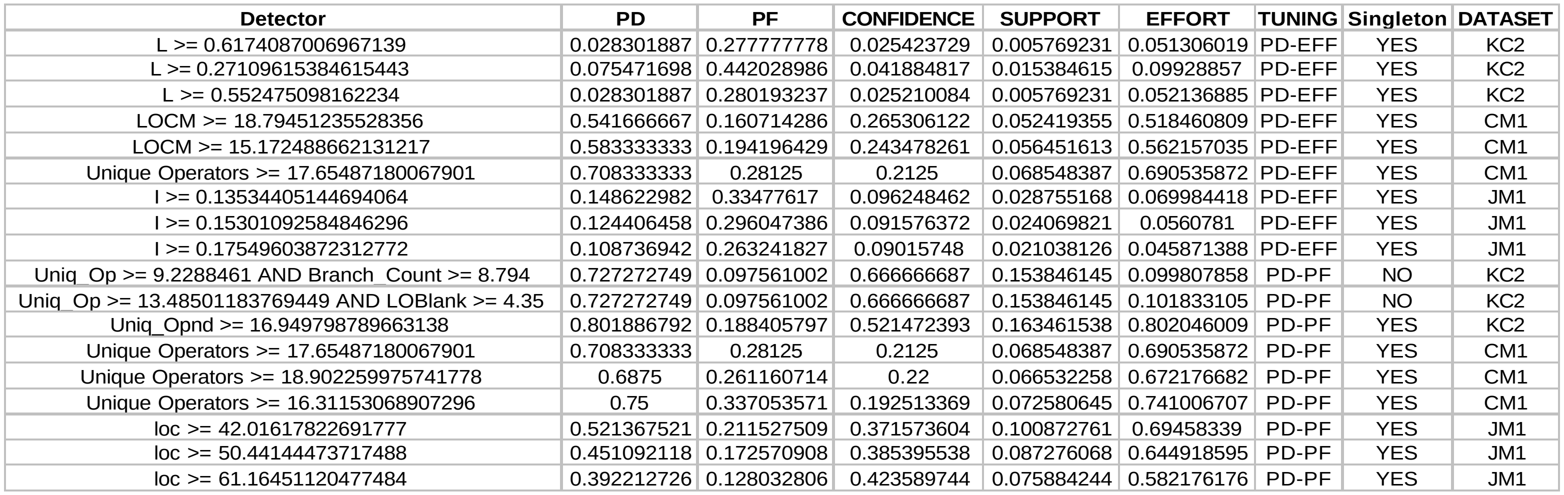
The results from this case study are as follows:

- The merit of composite conjunct theories in terms of difference between probability of detection, effort and the difference of probability of detection and probability of false alarm is outweighed by the singleton values produced via the basic ROCKY algorithm. That is the singleton theories produced during the first iteration of IQ are not usually bested, except for two theories in the KC2 data set, by any theories produced thereafter for the software metric domain. This reiterates our earlier hypothesis that the software metric data sets are a flat repetitive space easily summarized by one or two attributes.

- The difference between the probability of detection and the effort heuristic was never larger than ten percent.

- The confidence fluctuates between detectors, but is higher for tunings for PD-PF than it is with PD-effort.

- The information gained via this case study in terms of the software metrics domain out of the three data sets and two tunings is nothing ground breaking. All of the singleton metrics were from Halstead and not McCabes. However, McCabes metrics do appear in theories that are only a few percent lower than those in Figure 4.20, as do composite or multi range detectors.



Detector	PD	PF	CONFIDENCE	SUPPORT	EFFORT	TUNING	Singleton	DATASET
L >= 0.6174087006967139	0.028301887	0.277777778	0.025423729	0.005769231	0.051306019	PD-EFF	YES	KC2
L >= 0.27109615384615443	0.075471698	0.442028986	0.041884817	0.015384615	0.09928857	PD-EFF	YES	KC2
L >= 0.552475098162234	0.028301887	0.280193237	0.025210084	0.005769231	0.052136885	PD-EFF	YES	KC2
LOCM >= 18.79451235528356	0.541666667	0.160714286	0.265306122	0.052419355	0.518460809	PD-EFF	YES	CM1
LOCM >= 15.172488662131217	0.583333333	0.194196429	0.243478261	0.056451613	0.562157035	PD-EFF	YES	CM1
Unique Operators >= 17.65487180067901	0.708333333	0.28125	0.2125	0.068548387	0.690535872	PD-EFF	YES	CM1
I >= 0.13534405144694064	0.148622982	0.33477617	0.096248462	0.028755168	0.069984418	PD-EFF	YES	JM1
I >= 0.15301092584846296	0.124406458	0.296047386	0.091576372	0.024069821	0.0560781	PD-EFF	YES	JM1
I >= 0.17549603872312772	0.108736942	0.263241827	0.09015748	0.021038126	0.045871388	PD-EFF	YES	JM1
Uniq_Op >= 9.2288461 AND Branch_Count >= 8.794	0.727272749	0.097561002	0.666666687	0.153846145	0.099807858	PD-PF	NO	KC2
Uniq_Op >= 13.48501183769449 AND LOBlank >= 4.35	0.727272749	0.097561002	0.666666687	0.153846145	0.101833105	PD-PF	NO	KC2
Uniq_Opnd >= 16.949798789663138	0.801886792	0.188405797	0.521472393	0.163461538	0.802046009	PD-PF	YES	KC2
Unique Operators >= 17.65487180067901	0.708333333	0.28125	0.2125	0.068548387	0.690535872	PD-PF	YES	CM1
Unique Operators >= 18.902259975741778	0.6875	0.261160714	0.22	0.066532258	0.672176682	PD-PF	YES	CM1
Unique Operators >= 16.31153068907296	0.75	0.337053571	0.192513369	0.072580645	0.741006707	PD-PF	YES	CM1
loc >= 42.01617822691777	0.521367521	0.211527509	0.371573604	0.100872761	0.69458339	PD-PF	YES	JM1
loc >= 50.44144473717488	0.451092118	0.172570908	0.385395538	0.087276068	0.644918595	PD-PF	YES	JM1
loc >= 61.16451120477484	0.392212726	0.128032806	0.423589744	0.075884244	0.582176176	PD-PF	YES	JM1

Figure 4.20 Results from Case Study 3

4.4.4 Conclusion

Our conclusions for this case study are as follows:

- There are reoccurring attribute ranges across the data sets. Unique operators, unique operands, loc and L are seen repeatedly in Figure 20.
  - L and I seem helpful in minimizing the effort while maximizing the probability of detection.
  - Unique operators and unique operands seem to be useful in maximizing the probability of detection while minimizing the probability of false alarm.
  - Further, analysis is needed to assess any common thresholds that may exist for different metrics across data sets. We refer to this in our future works section.
- The difference between the probability of detection and the effort heuristic was never larger than ten percent. This implies that there is no theory with a high probability of detection that can be created by reading a considerably small amount of code.
- Only two detectors out of twenty four that were observed were composed of two attributes. As a result, there is not a great benefit to conjunct composite detectors in this domain.
- Furthermore, our earlier observation of triviality of composite conjunct detectors needs more information to solidity. That is, we need more data sets to accurately assert that conjunct theories in the software metric domain do not assist in fault detection.

4.5 Case Study 4: Comparison to Treatment learning

This case study focuses on IQ's ability to efficiently perform treatment learning. Treatment learning creates a set of theories that most drive the baseline class distribution towards a desired class goal [Menzies02]. In order to compare IQ to treatment learning, we compose a set of treatments from several of the previous data sets. However, since treatment learning identifies a best class and maintains a class order, it would be ideal to run the learner once for every class. This is not practical for data sets such as soybean, which contain eighteen unique class values and possesses no indication of a best or worst class value. Therefore, metric data sets such as soybean, glass, etc. are dropped from this study.

4.5.1 Method

In this case study we utilize the TAR3 treatment learner as well as IQ on all several data sets. The method is as follows:

- We first ran TAR3 for nine data sets with different class orders.
- Next we ran IQ tuned for lift with the same data sets and class order.
- In order to emulate treatment learning IQ must be tuned to optimize for lift as well as an altered discretization of using bands instead of standard deviation as in previous case studies.
- All class orders, i.e. best to worst, were matched in both IQ and TAR3 for every run.
- We then employed the same heuristic summary used in Case Studies 1 and 2.
- Finally we also perform a t-test on the difference in means of the lists from treatments created from IQ and TAR3.

As an additional caveat, the discretization method for continuous attributes differs slightly from TAR to IQ leading to a possible difference in lift. Also differences in lift will appear due to variations in cross validation between the algorithms and other slight algorithmic differences such as employment of a scalar penalty on worth.

4.5.2 Information Gathering

This section is similar to Section 4.2.2 in that we record heuristics for every run. Also, the only heuristics we observe are LIFT and learner runtimes Runs are done on a class order basis and not simply on class value.

4.5.3 Information Summary

Similar to Section 4.2.3

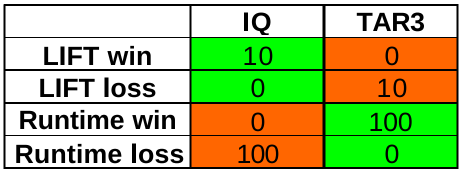
4.5.4 Analysis from Summary Space

The results Figure 4.9 from the almost forty treatments are consistent and are as follows.

- IQ always finds treatments with higher or comparable lift. Since TAR3 does not assess its treatments based on any of the discussed heuristics aside from lift, additional heuristics become negligible due to our focus on comparison to TAR3 and not on the merits of its treatments.

- The merit of these treatments when evaluated in terms of our previous assessment criteria is in question. Many of theses treatments with high lift lack a strong probability of detection or a low probability of false alarm. In some instances a theory could be found that so profoundly alters the base line that the lift is several magnitudes over that of the highest TAR3 lift yet is considered garbage in terms of all other heuristics, e.g. an extremely low confidence or probability detection.

- The size of treatments is also important and is also consistently comparable or bested by IQ and usually never exceeded three attribute ranges.



	IQ	TAR3
LIFT win	10	0
LIFT loss	0	10
Runtime win	0	100
Runtime loss	100	0

Figure 4.21 Results from Case Study 4

4.5.5 Student's t-test procedure

See Section 4.2.5.

4.5.6 Student's t-test results

The t-test results for the paired mean distribution between IQ and TAR for lefts of 40 treatments was that we were unable to reject our null hypothesis that the means are significantly equivalent or both distributions are from some similar population. There results were based on a p-value of 0.244605594. This probability produced we cannot rejected with a=0.05 or a confidence of 95%, implying that both populations have similar means.

4.5.7 Conclusion

Our goal in this study is to prove that IQ when tuned to lift can emulate the TAR3 and provide additional heuristics from which to assess treatments. Therefore, we compared treatments found from IQ that most resembled TAR3. Based on this case study we can conclude that IQ when tuned for lift and using a band discretization policy is able to effectively emulate treatment learning.

The added advantage to IQ is that it also assesses its treatments with other heuristics other than lift. This solves TAR3's problem of large sets of identical or near identical treatments based solely on the lift heuristic, because you have a pool of treatments that can be chosen on basis other than simply lift.

4.6 Case Study 5: Comparison to FSS techniques and simplified learning

This section we describe a case study conducted from FSS techniques on NASA data sets. The purpose of this study is to test our assumption that most of the metric data sets contain large portions of redundant and useless data and that most of the excess information leads to a degraded accuracy of classifiers, but employment of FSS techniques does not necessarily help other heuristics that we observe. We also intend to compare (in size and heuristic value) IQ with the best classifier obtained from the FSS chosen attributes.

4.6.1 Method

For this case study we employed the seven feature subset selection techniques seen Chapter 2. PCA, CBS, IG, RLF, WRAPPER, and 1R implementations supplied with the WEKA machine learning toolkit.

The first step was for each FSS method to generate candidate features which were then selected and assessed by WEKA learners. We also employed ten-way cross validation for most FSS techniques and for all learners. In the usual case, the WEKA environment offered options to conduct FSS via a ten-way cross validation. We disabled this option for WRAPPER since that was impractically slow, especially for the ten thousand records in the JM1 data set. As an additional note, the WRAPPER FSS was run once for each type of leaner. The reasoning behind this is that WRAPPER is specifically tailored to run for a specified classifier. For example, if we were “wrapping” learner “X” then we can only assess the WRAPPER’s output on learner “X”.

Secondly, the chosen features are then run though the machine learners, Naïve Bays and J4.8 within the WEKA. Assessing FSS via these two learners is quite standard in the FSS literature since these are widely used and understood learning systems [Kohavi97]. These learners were also run using ten-way cross validation. The accuracy, probability of detection, probability of false alarm, and confidence of the classifiers were then record accordingly from for a non-zero fault class in the software metric data sets and can be seen in Figures 4.10-13.

Moreover, we ran IQ and OneR separately without feature selection. Since OneR produces a theory consisting of a single attribute, employing FSS techniques is useless. The accuracy, probability of detection, probability of false alarm, and confidence were recorded in Figure 4.14 along with the best runs from J4.8 and Naïve Bayes from Figures 4.10-13.

4.6.2 Results

Figures 4.10 - 4.13 all show four different NASA metric data sets run through six feature selection techniques. Figure 4.14 shows that of IQ and OneR runs. The highest value per column per learner is in bold. The results are as follows:

4.6.2.1 J4.8 and Naïve Bayes

The values Figures 4.10 - 4.13 suggest several points for standard learners.

- There is a slight increase in probability of detection when FSS techniques are applied, and in one case, JM1, a decrease

- There is a slight decrease in probability of false alarm when FSS techniques are applied

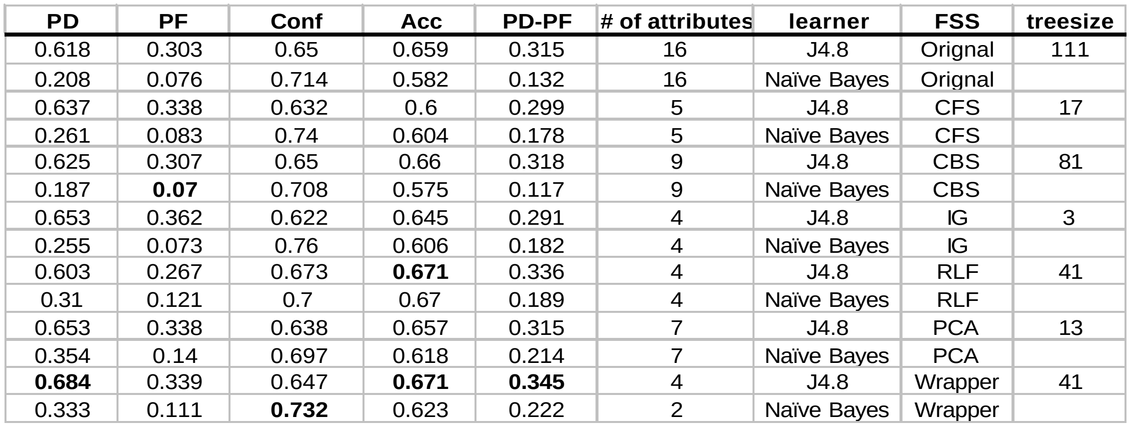
- There is a slight increase in confidence when FSS techniques are applied, and in one case, CM1, a decrease

- There is a consistent increase in accuracy when FSS techniques are applied

- There is a slight increase in PD - PF when FSS techniques are applied, and in one case, JM1, a decrease

- In the CM1 data set there was an evident breakdown of the J4.8 learning algorithm. In several feature sets, J4.8 was unable to stabilize and could not create a viable split across the candidate attributes selected by the several FSS techniques. Hence we notice a zero or negative difference in probability of detection and false alarm.

Next will observe how the improved coefficients compare with IQ.



PD	PF	Conf	Acc	PD-PF	# of attributes	learner	FSS	treesize
0.618	0.303	0.65	0.659	0.315	16	J4.8	Orignal	111
0.208	0.076	0.714	0.582	0.132	16	Naïve Bayes	Orignal	
0.637	0.338	0.632	0.6	0.299	5	J4.8	CFS	17
0.261	0.083	0.74	0.604	0.178	5	Naïve Bayes	CFS	
0.625	0.307	0.65	0.66	0.318	9	J4.8	CBS	81
0.187	**0.07**	0.708	0.575	0.117	9	Naïve Bayes	CBS	
0.653	0.362	0.622	0.645	0.291	4	J4.8	IG	3
0.255	0.073	0.76	0.606	0.182	4	Naïve Bayes	IG	
0.603	0.267	0.673	**0.671**	0.336	4	J4.8	RLF	41
0.31	0.121	0.7	0.67	0.189	4	Naïve Bayes	RLF	
0.653	0.338	0.638	0.657	0.315	7	J4.8	PCA	13
0.354	0.14	0.697	0.618	0.214	7	Naïve Bayes	PCA	
0.684	0.339	0.647	**0.671**	**0.345**	4	J4.8	Wrapper	41
0.333	0.111	**0.732**	0.623	0.222	2	Naïve Bayes	Wrapper	

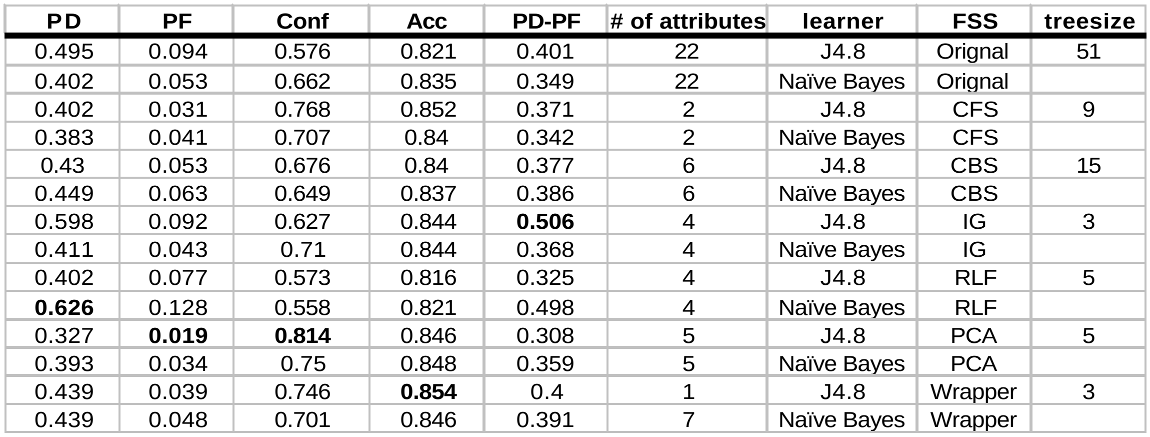
Figure 4.22 AN1 FSS runs

PD	PF	Conf	Acc	PD-PF	# of attributes	learner	FSS	treesize
0.167	0.027	**0.4**	0.895	0.14	24	J4.8	Orignal	33
0.333	0.096	0.271	0.849	**0.237**	24	Naïve Bayes	Orignal	
0	**0**	0	**0.903**	0	3	J4.8	CFS	1
0.208	0.042	0.345	0.885	0.166	3	Naïve Bayes	CFS	
0	**0**	0	**0.903**	0	2	J4.8	CBS	1
0.167	0.045	0.286	0.879	0.122	2	Naïve Bayes	CBS	
0.063	0.013	0.333	0.897	0.05	4	J4.8	IG	1
0.292	0.063	0.333	0.875	0.229	4	Naïve Bayes	IG	
0	**0**	0	**0.903**	0	4	J4.8	RLF	1
0.354	0.125	0.233	0.824	0.229	4	Naïve Bayes	RLF	
0	0.009	0	0.895	-0.009	7	J4.8	PCA	13
0.229	0.069	0.262	0.862	0.16	7	Naïve Bayes	PCA	
0	0.002	0	0.901	-0.002	4	J4.8	Wrapper	1
NA	NA	NA	NA	NA	NA	Naïve Bayes	Wrapper	

Figure 4.23 CM1 FSS runs

PD	PF	Conf	Acc	PD-PF	# of attributes	learner	FSS	treesize
0.25	0.076	0.441	0.793	**0.174**	22	J4.8	Orignal	677
0.199	0.051	0.485	0.804	0.148	22	Naïve Bayes	Orignal	
0.121	0.031	0.485	0.85	0.09	7	J4.8	CFS	63
0.219	0.055	0.489	0.804	0.164	7	Naïve Bayes	CFS	
0.248	0.074	0.445	0.794	**0.174**	19	J4.8	CBS	671
0.218	0.057	0.478	0.802	0.161	18	Naïve Bayes	CBS	
0.112	0.023	**0.543**	**0.809**	0.089	4	J4.8	IG	17
0.191	0.05	0.479	0.803	0.141	4	Naïve Bayes	IG	
0.076	**0.014**	0.563	**0.809**	0.062	4	J4.8	RLF	25
0.209	0.063	0.444	0.796	0.146	4	Naïve Bayes	RLF	
0.085	0.017	0.541	0.809	0.068	8	J4.8	PCA	17
0.199	0.06	0.443	0.796	0.139	8	Naïve Bayes	PCA	
0.148	0.032	0.523	**0.809**	0.116	4	J4.8	Wrapper	5
0.174	0.04	0.508	0.807	0.134	2	Naïve Bayes	Wrapper	

Figure 4.24 JM1 FSS Runs



PD	PF	Conf	Acc	PD-PF	# of attributes	learner	FSS	treesize
0.495	0.094	0.576	0.821	0.401	22	J4.8	Orignal	51
0.402	0.053	0.662	0.835	0.349	22	Naïve Bayes	Orignal	
0.402	0.031	0.768	0.852	0.371	2	J4.8	CFS	9
0.383	0.041	0.707	0.84	0.342	2	Naïve Bayes	CFS	
0.43	0.053	0.676	0.84	0.377	6	J4.8	CBS	15
0.449	0.063	0.649	0.837	0.386	6	Naïve Bayes	CBS	
0.598	0.092	0.627	0.844	**0.506**	4	J4.8	IG	3
0.411	0.043	0.71	0.844	0.368	4	Naïve Bayes	IG	
0.402	0.077	0.573	0.816	0.325	4	J4.8	RLF	5
0.626	0.128	0.558	0.821	0.498	4	Naïve Bayes	RLF	
0.327	**0.019**	**0.814**	0.846	0.308	5	J4.8	PCA	5
0.393	0.034	0.75	0.848	0.359	5	Naïve Bayes	PCA	
0.439	0.039	0.746	**0.854**	0.4	1	J4.8	Wrapper	3
0.439	0.048	0.701	0.846	0.391	7	Naïve Bayes	Wrapper	

Figure 4.25 KC2 FSS RUNS

4.6.2.2 IQ and WEKA learners

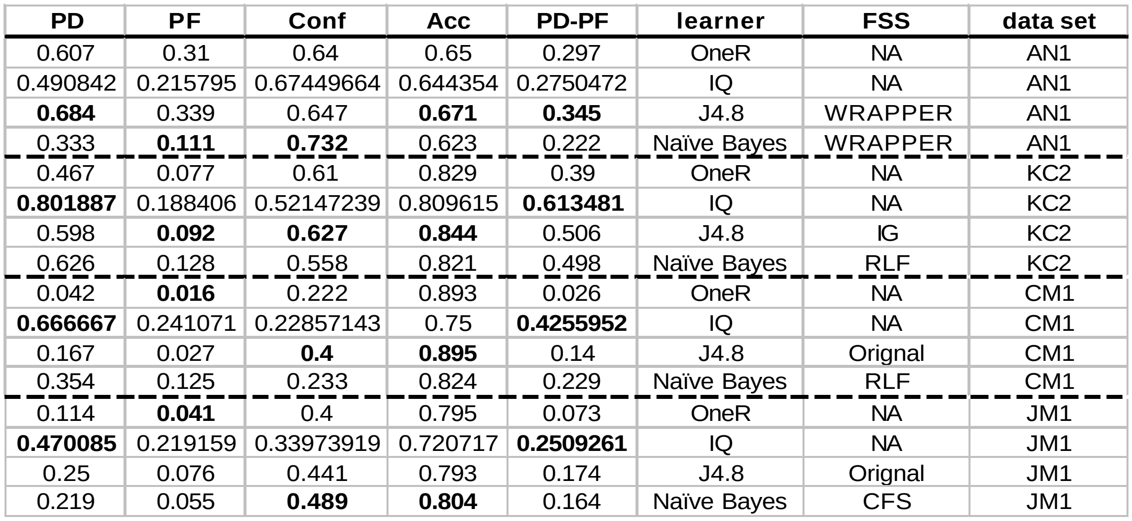
In Figure 4.14 we note that OneR runs and our Best IQ runs from Case Study 2 are paired against the best Naïve Bayes and J4.8 runs from obtain from Figure 4.10 - 4.13. The results are as follows:

- IQ significantly bests all learners by at least eight percent in the probability of detection and false alarm in all data sets except for AN1.

- There is no clear winner here by any significant margin for probability of false alarm. However IQ faired the worst usually loosing to a margin of at least ten percent

- There is no clear winner here by any significant margin for confidence.

- IQ significantly bests all learners by at least ten percent in the difference in probability of detection and false alarm in all data sets except for AN1.



PD	PF	Conf	Acc	PD-PF	learner	FSS	data set
0.607	0.31	0.64	0.65	0.297	OneR	NA	AN1
0.490842	0.215795	0.67449664	0.644354	0.2750472	IQ	NA	AN1
0.684	0.339	0.647	**0.671**	**0.345**	J4.8	WRAPPER	AN1
0.333	**0.111**	**0.732**	0.623	0.222	Naïve Bayes	WRAPPER	AN1
0.467	0.077	0.61	0.829	0.39	OneR	NA	KC2
0.801887	0.188406	0.52147239	0.809615	**0.613481**	IQ	NA	KC2
0.598	**0.092**	**0.627**	**0.844**	0.506	J4.8	IG	KC2
0.626	0.128	0.558	0.821	0.498	Naïve Bayes	RLF	KC2
0.042	**0.016**	0.222	0.893	0.026	OneR	NA	CM1
0.666667	0.241071	0.22857143	0.75	**0.4255952**	IQ	NA	CM1
0.167	0.027	**0.4**	**0.895**	0.14	J4.8	Orignal	CM1
0.354	0.125	0.233	0.824	0.229	Naïve Bayes	RLF	CM1
0.114	**0.041**	0.4	0.795	0.073	OneR	NA	JM1
0.470085	0.219159	0.33973919	0.720717	**0.2509261**	IQ	NA	JM1
0.25	0.076	0.441	0.793	0.174	J4.8	Orignal	JM1
0.219	0.055	**0.489**	**0.804**	0.164	Naïve Bayes	CFS	JM1

Figure 4.26 Results from Case Study 5

4.6.3 Conclusion

Our conclusions for this case study are as follows:

- First, the employment of FSS techniques does not show significant improvement across all heuristics tested. Only accuracy and probability showed consistent improvements.

- Second, in cases where there are increases in heuristics obtained from FSS selected attributes, do not generally improve all other heuristics. That is the classifier produced is not necessarily optimal under all or most conditions.

- Lastly, when compared with IQ, the best feature selected classifiers from Naïve Bayes and J4.8 did not produce better probability of detection or the difference of probability of detection and false alarm for three of the data sets for a non-zero fault class value.

4.7 Case Study 6: Comparison to Disjunctions in IQ

This case study explores the possibilities of a disjunctive IQ. That is, instead of unions or conjunctions between each attribute range in a detector we introduce disjunctions between attribute ranges. However this study is limited. It explores purely disjunctive detectors and does not incorporate detectors with both disjunction and conjunctions of attribute ranges.

The purpose of this case study was to assess if there was any benefit to disjunctive theories and not to assess the benefit of disjunctive theories over standard machine learner. In this study, we assess IQ against IQ disjunct for several of the standard data sets used in earlier case studies. Then we repeat Case Study 3 but for disjunct detectors.

4.7.1 Method

This section compares IQ and IQ disjunct. There are three parts to this case study.

The first part we compare the overall differences between both learners and are less interested to know precisely which theories from what data sets are best. The procedure is as follows:

- Run IQ disjunct tuned for PD-PF for all UC Irvine data sets.
- Compare results with IQ conjunct tuned for PD-PF from Case Study 1 with a summary table as in that case study.
- Run IQ disjunct tuned for PD-PF for all NASA data sets.
- Compare results with IQ conjunct tuned for PD-PF from Case Study 3 with a summary table as Case Study 1.

The second part we offer a t-test similar to that employed in earlier studies to assess any significant differences in heuristic means. We do this separately for NASA and UC Irvine data sets.

In the last part we focus on the types of theories we learned from the NASA metric data sets. The method is similar to Case Study 3 except we offer four runs per IQ tuning per data set.

4.7.2 Information Gathering

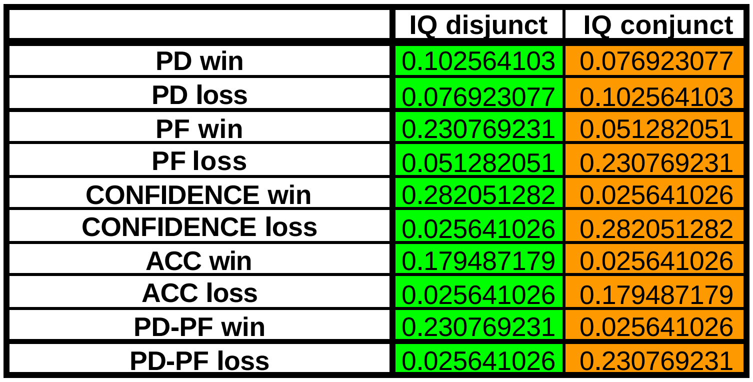
See Section 4.2.2 and 4.4.2

4.7.3 Information Summary

See Section 4.2.3

4.7.4 Analysis from Summary Space

Figure 4.27 are the results from the first half of the summary results. It details the summary information obtained from comparisons between IQ disjunct and conjuct for all UC Irvine data sets. The results are one-sided in favor of IQ disjunct.

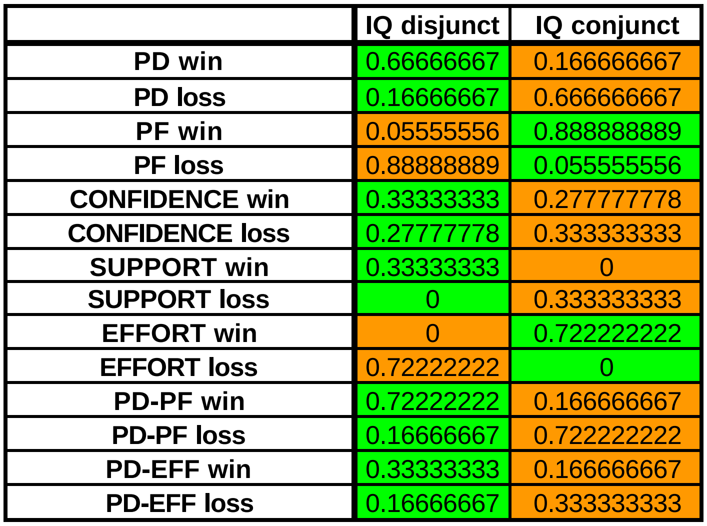


	IQ disjunct	IQ conjunct
PD win	0.102564103	0.076923077
PD loss	0.076923077	0.102564103
PF win	0.230769231	0.051282051
PF loss	0.051282051	0.230769231
CONFIDENCE win	0.282051282	0.025641026
CONFIDENCE loss	0.025641026	0.282051282
ACC win	0.179487179	0.025641026
ACC loss	0.025641026	0.179487179
PD-PF win	0.230769231	0.025641026
PD-PF loss	0.025641026	0.230769231

Figure 4.27 Learner Run Summary for UC Irvine data

Figure 4.28 are the results from the first half of the summary results. It details the summary information obtained from comparisons between IQ disjunct and conjuct for all UC Irvine data sets. The results are as follows:

- For PD, IQ disjunct had more wins and fewer losses.
- For PD, IQ conjunct had more wins and fewer losses.
- For confidence, IQ disjunct had more wins and fewer losses.
- For support, IQ disjunct had more wins and fewer losses.
- For effort, IQ disjunct had more wins and fewer losses.
- For PD-PF, IQ disjunct had more wins and fewer losses.
- For PD-PF, IQ disjunct had more wins and fewer losses.



	IQ disjunct	IQ conjunct
PD win	0.66666667	0.166666667
PD loss	0.16666667	0.666666667
PF win	0.05555556	0.888888889
PF loss	0.88888889	0.055555556
CONFIDENCE win	0.33333333	0.277777778
CONFIDENCE loss	0.27777778	0.333333333
SUPPORT win	0.33333333	0
SUPPORT loss	0	0.333333333
EFFORT win	0	0.722222222
EFFORT loss	0.72222222	0
PD-PF win	0.72222222	0.166666667
PD-PF loss	0.16666667	0.722222222
PD-EFF win	0.33333333	0.166666667
PD-EFF loss	0.16666667	0.333333333

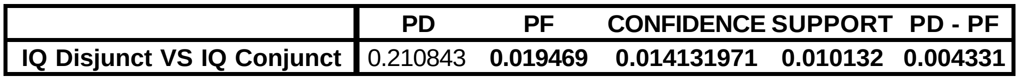
Figure 4.28 Learner Run Summary for NASA data

4.7.5 Students t-test Procedure

See Section 4.2.5

4.7.6 Students t-test Results

Figure 4.29 displays the results from the t-test for UC Irvine data. We can conclude that PF, confidence, support, and PD-PF all show significant differences in means.



	PD	PF	CONFIDENCE	SUPPORT	PD - PF
IQ Disjunct VS IQ Conjunct	0.210843	**0.019469**	**0.014131971**	**0.010132**	**0.004331**

Figure 4.29 t-test for IQ Disjunct and IQ Conjunct
means for UC Irvine

Figure 4.30 displays the results from the t-test for UC Irvine data. We can conclude that PD, PF, support, effort, and PD-PF all show significant differences in means.

	PD	PF	CONFIDENCE	SUPPORT	EFFORT	PD-PF	PD-EFF
IQ Disjunct vs IQ Conjunct	**0.00817451**	**0.0039206**	0.887684611	**0.03352056**	**0.0029136**	**0.0489462**	0.6192331

Figure 4.30 t-test for IQ Disjunct and IQ Conjunct
means for NASA

4.7.7 Result from NASA metrics assessment

Figure 4.31 shows several runs in two different IQ disjunctive tunings for NASA metric data sets with a non-zero fault class, while Figure 4.20 shows similar data but for a conjunctive IQ. We draw the following observations

- IQ disjunct possesses a much higher number of composite theories than IQ with conjunctions.

- “L” and “I” Halstead metrics are prevalent in more than one data set for IQ tuned for PD-EFF

- unique operand and unique operators metrics are prevalent in more than one data set for IQ tuned for PD-PFs

- LOC is prevalent in more than one data set when IQ is tuned for PD-PF

- The efforts produced with disjunct theories are marginally lower than conjunct theories.

Detector	PD	PF	CONFIDENCE	SUPPORT	EFFORT	TUNING	Singleton	DATASET
L >= 0.27109615384615443 OR I >= 28.42494230769226 OR IvG >= 5.676844638708482	0.91509434	0.690821256	0.253263708	0.186538462	0.883574804	PD-EFF	NO	KC2
L >= 0.27109615384615443 OR I >= 28.42494230769226 OR Branch_Count >= 8.794230769230769	0.933962264	0.712560386	0.251269036	0.190384615	0.903515605	PD-EFF	NO	KC2
I >= 28.42494230769226 OR ivG >= 4.547196256416347 OR L >= 0.3008573883411244	0.91509434	0.65942029	0.262162162	0.186538462	0.884665317	PD-EFF	NO	KC2
L >= 0.27109615384615443	0.075471698	0.442028986	0.041884817	0.015384615	0.049644285	PD-EFF	YES	KC2
LOCM >= 15.172488662131217 OR Unique Operators >= 18.902259975741778	0.8125	0.305803571	0.221590909	0.078629032	0.720547388	PD-EFF	NO	CM1
Unique Operators >= 18.902259975741778 OR LOCM >= 12.326612903225806	0.8125	0.323660714	0.211956522	0.078629032	0.731386762	PD-EFF	NO	CM1
LOCM >= 15.172488662131217 OR Unique Operators >= 17.65487180067901	0.8125	0.321428571	0.213114754	0.078629032	0.734164352	PD-EFF	NO	CM1
LOCM >= 18.79451235528356 OR Unique Operators >= 17.65487180067901	0.791666667	0.308035714	0.215909091	0.076612903	0.723595962	PD-EFF	NO	CM1
l >= 0.17549603872312772	0.108736942	0.263241827	0.09015748	0.021038126	0.045871388	PD-EFF	YES	JM1
l >= 0.15301092584846296	0.124406458	0.296047386	0.091576372	0.024069821	0.0560781	PD-EFF	YES	JM1
l >= 0.13534405144694064	0.148622982	0.33477617	0.096248462	0.028755168	0.069984418	PD-EFF	YES	JM1
l >= 0.19637507210674499	0.094966762	0.238182025	0.087298123	0.018373909	0.038520281	PD-EFF	YES	JM1
loc >= 42.01617822691777 OR lOCode >= 26.252209462563158 OR lOBlank >= 5.7219718907743715	0.61965812	0.29137715	0.337820347	0.119889757	0.76349619	PD-PF	NO	JM1
loc >= 42.01617822691777 OR lOCode >= 26.252209462563158 OR lOComment >= 4.98961821728556	0.613010446	0.284884383	0.340453586	0.118603583	0.757520409	PD-PF	NO	JM1
loc >= 42.01617822691777 OR lOCode >= 26.252209462563158 OR locCodeAndComment >= 1.973378653299577	0.594491928	0.26745643	0.347777778	0.115020671	0.743837117	PD-PF	NO	JM1
loc >= 42.01617822691777 OR lOCode >= 26.252209462563158 OR locCodeAndComment >= 1.6490313066891038	0.594491928	0.26745643	0.347777778	0.115020671	0.743837117	PD-PF	NO	JM1
Uniq_Opnd >= 16.949798789663138 OR LOBlank >= 6.657687475292603 OR evG >= 5.857382136459768	0.839622642	0.210144928	0.505681818	0.171153846	0.825517993	PD-PF	NO	KC2
Uniq_Opnd >= 16.949798789663138 OR LOBlank >= 6.657687475292603 OR ivG >= 5.676844638708482	0.839622642	0.210144928	0.505681818	0.171153846	0.825777639	PD-PF	NO	KC2
Uniq_Opnd >= 16.949798789663138 OR LOC >= 37.03269230769231 OR ivG >= 5.676844638708482	0.839622642	0.210144928	0.505681818	0.171153846	0.830814769	PD-PF	NO	KC2
Uniq_Opnd >= 16.949798789663138 OR LOComment >= 11.173827864530345 OR ivG >= 5.676844638708482	0.820754717	0.193236715	0.520958084	0.167307692	0.810458535	PD-PF	NO	KC2
Unique Operators >= 18.902259975741778 OR LOCM >= 15.172488662131217 OR LOC >= 40.46036177993663	0.833333333	0.316964286	0.21978022	0.080645161	0.739109816	PD-PF	NO	CM1
Unique Operators >= 18.902259975741778 OR LOCM >= 15.172488662131217 OR Unique Operands >= 42.86943205836745	0.833333333	0.319196429	0.218579235	0.080645161	0.738838832	PD-PF	NO	CM1
Unique Operators >= 18.902259975741778 OR LOCM >= 15.172488662131217	0.8125	0.305803571	0.221590909	0.078629032	0.720547388	PD-PF	NO	CM1
Unique Operators >= 18.902259975741778 OR LOCM >= 18.79451235528356	0.791666667	0.292410714	0.224852071	0.076612903	0.709978999	PD-PF	NO	CM1

Figure 4.31 Results from Case Study 6

4.7.8 Conclusions

Our conclusions are in three parts.

- **Part 1:** Based on the summary and t-tests of the runs produced from the UC Irvine data we conclude that IQ disjunct produced significantly better PF, confidence, support and PD-PF heuristics than IQ conjunct.

- **Part 2:** Based on the summary and t-tests of the runs produced from NASA metric data we conclude that IQ disjunct significantly found better PD, support and PD-PF heuristics then IQ conjunct. While IQ conjunct produced significantly lower PFs than IQ disjunct.

- **Part 3:** Based on the Figure 4.31 we conclude that there are significantly more composite theories produced with IQ disjunct than IQ conjunct..

4.8 Summary

In this chapter we observed six case studies conducted under a variety of different criteria comparing our IQ algorithm with itself other learners. The following is a list of each case study and its outcome.

- **Case Study 1**: In this case study we compared J4.8, Naïve Bayes, OneR, to two different IQ tunings, and two different IQ culling rules using ten data sets from the UC Irvine repository. The outcome of this case study was that for PD, PF, and accuracy IQ was significantly better than WEKA learners and runtimes were significantly worse.

- **Case Study 2**: In this case study we compared J4.8, Naïve Bayes, OneR, to two different IQ tunings, and two different IQ culling rules using ten metric data sets from the NASA. The outcome of this case study was that IQ had a significantly better PD-PF than WEKA learners.

- **Case Study 3**: In this case study we compared ROCKY to two different IQ tunings, and two different IQ culling rules using NASA metric data sets. The outcome of this case study was that there was no significant benefit to composite conjunct theories in the software metric domain.

- **Case Study 4**: In this case study we compared TAR3 to IQ tuned to lift using data sets from the UC Irvine repository and NASA. The outcome of this case study was that IQ could effectively emulate TAR3.

- **Case Study 5**: In this case study we compared J4.8, Naïve Bayes with FSS techniques employed to OneR and the best IQ runs using NASA data metrics. The outcome of this case study was that FSS techniques failed to improved all heuristics and IQ performed better in PD-PF in three data sets

- **Case Study 6**: In this case study we compared IQ disjunct to IQ conjunct using NASA and UC Irvine data. The outcome was IQ disjunct produced better heuristics save for PF's in the NASA data sets. It also produced more composite theories than IQ conjunct.

5 Discussion of Results

In this chapter we will discuss the results obtained from our case studies and tie them in with the problems we stated with current machine learning. First we will discuss our multiple heuristic assessments. Then we move on to theory size and simplicity. Afterwards we discuss feature selection. Finally we look at possible future directions.

5.1 Multiple Heuristic

Current classifier comparators revolve around maximizing accuracy ultimately ignoring concepts such as imbalanced error costs and greatly skewed class distributions, which are often found in real world data sets [Fawcett98]. Therefore, accuracy is not always the best assessment criteria for different classifiers. In this section we will take data from our case studies to support this claim and suggest that other heuristics are also beneficial.

5.1.1 Solution to accuracy instability

Figure 5.1 and Figure 5.2 consists of hundreds of instances of different data – class – learner detectors from the UC Irvine and NASA data sets respectfully. This is a compilation of all learners including all IQ tunings. All the data was sorted by the difference of probability of detection and false alarm in ascending order. Then this heuristic along with accuracy was plotted. The results are that there is no apparent stabilization of accuracy even the probability of detection reaches one and the probability of false alarm reaches zero. As mentioned previously, a sound theory is not dependent upon accuracy.

As a solution, we address this problem by allowing tuning for other heuristics, in order not to be limited to accuracy when it becomes instable in a domain. As illustrated in Case Study 1 and 2, we tuned for the different of PD and

PF. This would produce detectors that are essentially insensitive to accuracy as shown in figures Figure 5.1 and Figure 5.2.

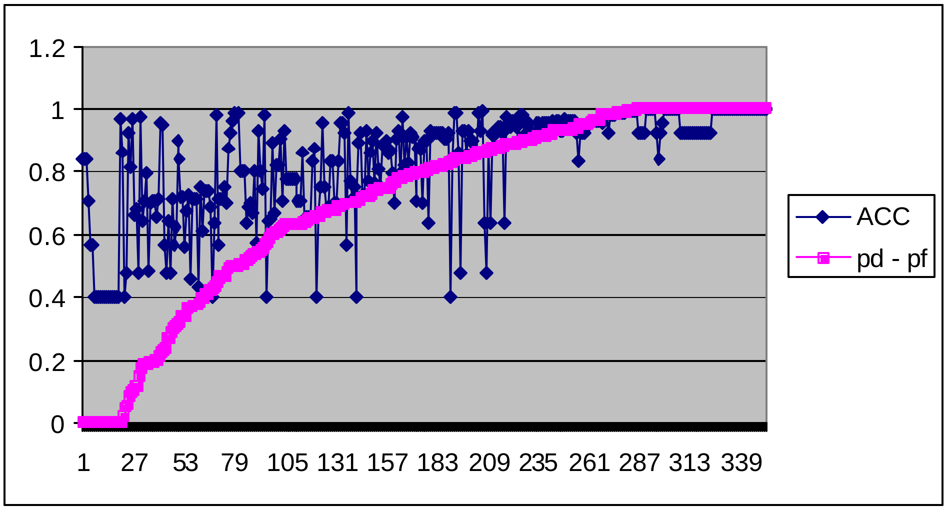


Figure 5.1 All learners results from UC Irvine

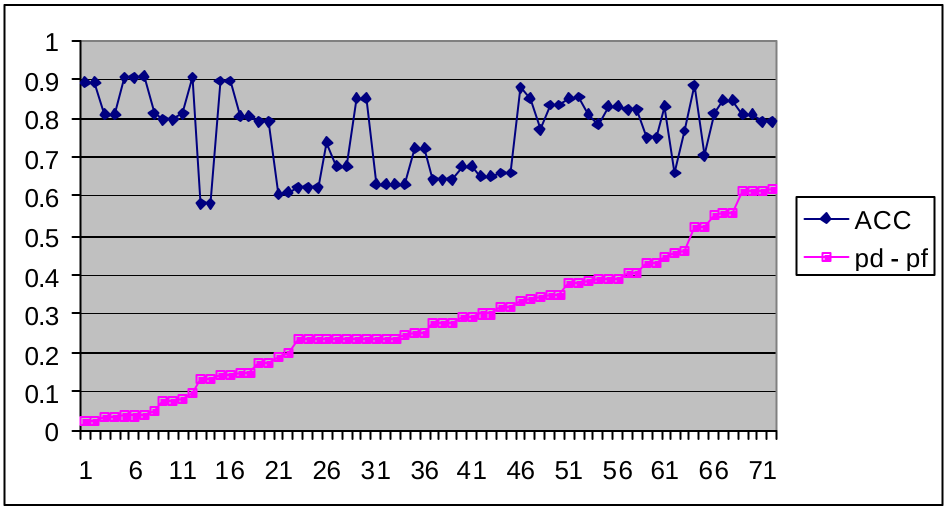


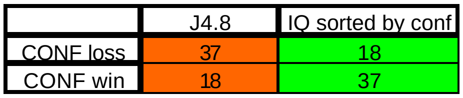
Figure 5.2 All learners from NASA data sets

5.1.2 Option Space

In real world applications the best detector found may not always be the most cost effective or feasible to implement. IQ accommodates multiple solutions by reporting a set of detectors with each run. We obtain an option space that is the most optimal in basic ROC space as with other heuristic comparisons. This

5.2 Heuristic comparisons across learners

For Case Study 1 and 2, IQ was placed against other learners for comparisons between heuristic values. We observe that there were few significant differences between learners based on a confidence of ninety five percent. However in Case Study 1 we significantly lost in confidence. The problem is that IQ is not specifically tuned for confidence therefore it preferred a higher PD-PF or ACC heuristic to that of confidence. However the question arises if we did tune for confidence could be compete with the J4.8 machine learner that produced a significantly better confidence. In Figure 5.3 we display a summary chart like that in Case Study 1 and 2 but for IQ tuned for confidence versus J4.8. The results are significant with a p-value from the t-test at 0.013 with 95% confidence. However we did not include a confidence tuning because there is no direct relationship with confidence or probability of detection or false alarm. In Figure 5.4 we display two detectors from the same data set that are radically different in terms of probability of detection and false alarm, yet the first detector with a perfect confidence is inferior to the second in terms of all other heuristics.



	J4.8	IQ sorted by conf
CONF loss	37	18
CONF win	18	37

Figure 5.3 J4.8 Vs IQ for confidence

Detector	PD	PF	CONF	SUPP
Mg >= 4.184636868945633	0.014286	0	1	0.004672897
Mg >= 3.4201607111642987 Al < 1.4449065420560752	0.8	0.215278	0.643678	0.261682243

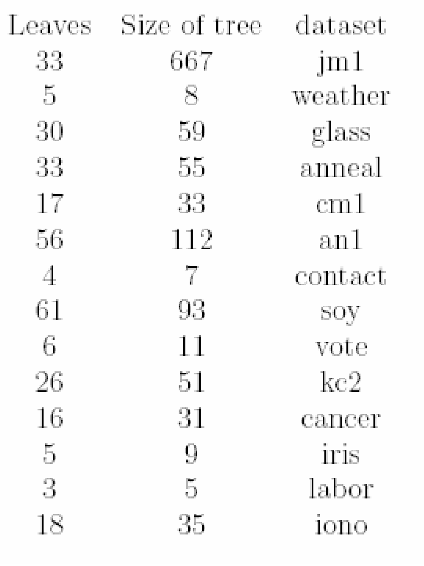
Figure 5.4 confidence and best detectors

5.3 Theory size and simplicity

In this section we will discuss the types of theories produced from different classifiers found throughout this thesis. We then compare and contrast our results. This section focuses on source data not culled by feature subset selection.

5.3.1 C4.5

For most data sets J4.8 produced large trees, which in many cases has hundreds of branches. Figure 5.5 reveals all branch and leaf counts produced by decision trees for the data sets in our case studies. These trees usually have more than fifty nodes and can have as many as 660 nodes for the jm1 data set.



Leaves	Size of tree	dataset
33	667	jm1
5	8	weather
30	59	glass
33	55	anneal
17	33	cm1
56	112	an1
4	7	contact
61	93	soy
6	11	vote
26	51	kc2
16	31	cancer
5	9	iris
3	5	labor
18	35	iono

Figure 5.5 J4.8 Tree Sizes

5.3.2 Naïve Bayes

Moreover, Naïve Bayes also produces a classifier of proportional size and to the number of attributes and classes of the data set. However Naïve Bayes is not a comprehensible classifier.

5.3.3 OneR

OneR produces theories from a single attribute. Most of the time there are few or no splits in this attribute producing theories that are composed of very ranges within the same attribute. OneR theories are comprehensible and similar to the theories produced by IQ.

5.3.4 IQ

IQ rarely produced detectors with more than five attributes. That is the iterative process of IQ could not locate better detectors after five or six cycles. In the case of NASA software metrics, IQ produced theories that were composed of a single attribute range. IQ disjunctive however produced theories with slightly higher number of attribute ranges but with out profound increases in heuristic vales.

5.4 Feature Selection

In this section we observe the results obtained in Case Study 5 and relate our results to simplified learning. In Case Study 5 we concluded that there was no general benefit to six FSS techniques when we focused on several different heuristics. We also compared the results to IQ. What was noticed was that IQ located several detectors with higher probability of detection and higher probability of false alarm when classifying a faulty module. This indicates that the portion of ROC space that IQ is focusing on is not observed by the traditional machine learners when employed through FSS techniques.

5.5 Future Work

5.5.1 More Heuristics

In addition to the heuristics used throughout this thesis we can employ any number of additional heuristics to our n-dimensional learner. Heuristics such as the area under the ROC curve, failure and success costs, etc. can bring added meaning and dimensionality to learnt theory.

5.5.2 IQ with disjunctions and conjunctions

In Chapter 4 we displayed results from a case study with IQ and IQ disjunct and found promising improvement upon IQ. However we have yet to explore combinations of disjunct and conjunct theories. We believe there is even additional gain in the combination of both, but the challenge lies in the combinatorial explosion that arise from both operator and operand when creating detectors of size n with two operators, *and* and *or.*

5.5.3 Streamlining IQ

One problem of IQ is that it is limited by parameters to shorten runtimes and avoid memory overruns. There are possibilities to enhance runtimes and minimize memory usage as seen in Chapter 3. By altering the performance of IQ we can explore higher dimensionalities and larger iterations, in order to find detectors that are currently out of reach. If we alter IQ and still cannot find any improvements across any data set with higher dimensionality or larger detectors, then it is a worthy note in support of Ockham's Razor.

5.5.4 Reasserting Learnt Theory

Since the software metric data sets share a common attributes, it is also of interest to assess the value of a theory learnt from one project to that of another. We found that although theories such as unique operands >= 16 are the best indicators for one data set they are not the best for another. However the thresholds by which a "good" theory compares across data sets is of great interest in that it validates that our theories are reusable within the same domain.

5.5.5 More Data

With only ten UC Irvine data sets and four NASA metric data sets we are unable to draw any concrete conclusions regarding the performance of IQ in regard to specific types of data. We also are unable to obtain certain series of metrics or metric thresholds that indicate if a code module is fault prone. If we obtain more data and conduct more tests then our results will be more conclusive.

6 Conclusion

In this section we will conclude by summarizing our case study results and showing the pros and cons of our techniques and methodologies.

We have demonstrated a new learning technique that is inherently different from classical machine learners in that it iteratively assesses and builds detectors based on ROC curves and other heuristics. We have also shown how well this learner performs on several different types of data. Also we have illustrated the effects of FSS techniques on many of the heuristics we have observed. Furthermore, we have observed different types of detectors, conjunctive, disjunctive, composite and singleton and how they relate through the heuristics observed.

To conclusion our findings in terms of comparisons to other machine learners, IQ did not perform significantly less than any other machine learner. When observing individual heuristics, IQ has the potential when tuned to that heuristic to significantly produce better results but this is not the rule. More testing is necessary as are tweaks in the IQ algorithm in order to generate more definitive results. Also, the data sets observed in Case Study 1 and Case Study 2 are inherently different, thus producing slightly different results.

In conclusion to our findings in terms of the types of detectors IQ produces, singleton theories in software metric domain are more prevalent, save for several disjunctive detectors. But the differences between conjunctive/disjunctive and composite/singleton detectors in this domain are slight at best. Conjunctive and disjunctive detectors are significantly different in this domain leading us to believe that the metric data sets are a flat repetitive space with few highly unique and key features.

In UC Irvine data sets, the composite detectors are much more critical in terms of key attributes for classes. Moreover, purely disjunctive theories are a superset of purely conjunctive theories given the same attribute ranges across detectors. This explains the slight increase in heuristic values of disjunctive theories in UC Irvine data but does not apply to NASA metric due to the high correlation of attributes when IQ iteratively selects and builds detectors.

Furthermore, the size and complexity of the theories produced by IQ are significantly more concise than most other learners. Naïve Bayes produces large classifiers that are not easy to decipher. J4.8 produces decision trees that are often large and cumbersome to follow. IQ produces theories which are comparable in size and complexity to the simple OneR algorithm.

We also conclude that IQ is more capable than other learners in observing detectors that lie in very different places in ROC space. This is due IQ's nature of producing a large set of detectors with optimized and varied heuristics that lie on a hull. It does not isolate its search to a certain regions of the search space, but instead obtains the outstanding detectors in all areas of the search space. Traditional machine learners internally optimize their classifiers on the basis of a single assessment, such as entropy or confidence, and are therefore limited in respect to those heuristics and the space produced from optimizing a specific heuristic. Additionally, IQ is not constricted by accuracy as are most traditional learners. Since in many domains accuracy has proven to be an inadequate heuristic on future performance, IQ is easily tailored to them.

While traditional machine learners produce a classifier that is valuable in predicting future classes, IQ can provide a large set of detectors that are easy to comprehend. This becomes advantageous in real world applications where the first or second theory is not always the optimal in terms of cost or feasibility. Through IQ's versatility in terms of tuning, domain specific heuristics, optional heuristics, and detector options, it is more adaptable to multiple domains.

[Adlassnig89] K. P. Adlassnig and W. Scheithauer. Performance evaluation of medical expert systems using roc curves. *Computers and Biomedical Research*, 22(4):297–313, 1989.

[Boetticher01] G. Boetticher. An assessment of metric contribution in the construction of a neural network-based sort estimator. In *Second International Workshop on Soft Computing Applied to Software Engineering*, Enschade, NL, 2001. Available from: http://nas.cl.uh.edu/boetticher/publications.html

[Dillon84] Dillon, W. and M. Goldstein: 1984, *Multivariate Analysis: Methods and Applications.* Wiley-Interscience.

[Dumais98] S. Dumais, J. Platt, D. Heckerman, and M. Sahami. Inductive learning algorithms and representations for text categorization. *In The International Conference on Information and Knowledge Management*, pages pp. 148–155, 1998.

[Fawcett01] Tom Fawcett. *Using Rule Sets to Maximize ROC Performance* Presented at the 2001 IEEE International Conference on Data Mining (ICDM-01)

[Gamma95] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: *Design Patterns.*1995 Addison-Wesley

[Hall98] Hall, M. A.: 1998, 'Correlation-based feature selection for machine learning'. Ph.D. thesis,

[Hall03] Hall, M. and G. Holmes: 2003, 'Benchmarking Attribute Selection Techniques for Discrete Class Data Mining'. *IEEE Transactions On Knowledge And Data Engineering* (to appear).

[Heeger98] D. Heeger. Signal detection theory, 1998. http://www.cns.nyu.edu/~david/ftp/handouts/sdt-advanced.pdf

[Holte93] R C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11:63, 1993.

[Khoshgoftaar99] T. M. Khoshgoftaar and E. B. Allen. *Model Software Quality with Classification Trees*. In H. Pham, editor, Recent Advances in Reliability and Quality Engineering. World Scientific, 1999.

[Kohavi97] Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

[Menzies02] Tim Menzies and Justin S. DiStefeno. Metrics that matter. *In 27th NASA SEL workshop on Software Engineering* (submitted), 2002.

[Menzies03] T. Menzies, J. Di Stefano, K. Ammar, K. McGill, P. Callis, R. Chapman, and Davis J. *When can we test less?* In Submitted to IEEE Metrics'03, 2003.

[Menzies03a] T. Menzies, K. Ammar, A. Nikora, J. Stefano. How Simple is Software Defect Detection?,2003.

[Munson90] Munson, J. C. and T. M. Khoshgoftaar: 1990, 'Regression Modeling of Software Quality'. *Information and Software Technology* 32(2), 105–114.

[Munson91] Munson, J. C. and T. M. Khoshgoftaar: 1991, 'The Use of Software Complexity Metrics in Software Reliability Modeling'. In: *Proceedings of the International Symposium on Software Reliability Engineering*, Austin, TX.

[Fawcett98] Foster Provost, Tom Fawcett, and Ron Kohavi. The case against accuracy estimation for comparing induction algorithms. *In Proc. 15th International Conf. on Machine Learning*, pages 445–453. ,Morgan Kaufmann, San Francisco, CA, 1998.

[Provost01] Foster Provost and Tom Fawcett. Robust Classification for Imprecise Environments, *Machine Learning Journal*, vol. 42, no. 3. March 2001. To appear

[Quinlan92] R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufman, 1992. ISBN: 1558602380.

[Rosen91] Kenneth H. Rosen, *Discrete Mathematics and Its Applications*, 2nd edition (NY: McGraw-Hill, 1991), pp. 284-286

[Shelby88] R.W. Selby and A.A. Porter. *Learning from examples: Generation and evaluation of decision trees for software resource analysis.* IEEE Trans. Software Eng., pages 1,743–1,757, December 1988.

[Witten99] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.* Morgan Kaufmann, 1999.