Learning Tiny Theories

Tim Menzies*, Rajesh Gunnalan*, Kalaivani Appukutty*,
Amarnath Srinivasan*, Ying Hu†

*Lane Department of Computer Science, West Virginia University, USA
†Electrical and Computer Engineering, University of British Columbia, Canada
{gunnalan|avani|amarnath}@csee.wvu.edu,tim@menzies.us,huying_ca@yahoo.com

September 28, 2003

Abstract

Business users often prefer simpler, rather than complicated theories. In this paper we present SELECT: a new method for feature subset selection using the TAR2 "treatment learner". SELECT can be used as a pre-processor to other learners for identifying useful feature subsets. This approach finds smaller theories that other approaches, with little or no loss of classifier accuracy.

1. Introduction

Data mining summarizes data but some of those summaries may be too complex. Recently, we have been trying to explain C4.5's decision trees [17] to business users. After many failures, we concluded that busy business users don't want elaborate theories that describe all the details of their domain. Such busy users just want to *fewest* details that *most* influence their domain. "Just give me the bottom line", one of our users demanded gruffly.

In response to these demands from our users, we have been exploring methods for generating tiny theories. The drawback with generating tiny theories is that they can ignore important domain details. The *benefits* of decreasing theory size must be carefully balanced against the *cost* of decreasing theory accuracy. We will accept a slight accuracy reduction Δ in exchange for a much smaller theory.

This paper describes SELECT, a new method for generating a theory *new* that is"better" than *old* where "better" is defined as follows:

$$|new| \ll |old| \wedge \left(\frac{accuracy(old) - accuracy(new)}{accuracy(old)} < \Delta \right) \tag{1}$$

	# of features	accuracy
domain	$\frac{all-SELECTed}{all}$	$\frac{all-SELECTed}{all}$
Ionosphere	94.1%	0.9%
HorseColic	90.9%	▶ 4.45%
Diabetes	87.5%	2.28%
lymph	83.3%	2.75%
Anneal	81.6%	-2.66%
Segment	78.9%	0.51%
breast-c	77.8%	0.0%
credit-g	75.0%	-2.17%
vote	62.5%	-0.21%
Soybean	54.3%	-0.65%
average	78.6%	1.13%

Figure 1: Reduction in number of features/accuracy. Black triangles denote experiments that violate Equation 1. Negative accuracies mean accuracy increased using the SELECTed features.

According to our users, a 2% reduction is acceptable and a 3% reduction is not. Hence, for this study, we will assume that $\Delta < 3\%$.

SELECT is an extension to the TAR2 "treatment learner" [9,14]. To describe SELECT, this paper first describes TAR2. The use of TAR2 within SELECT is then described, followed by an comparison of SELECT to six other *feature subset selection* methods from the machine learning literature. Figure 1 shows some of the results from learning theories using *all* available features and just those $SELECTed$ by our method. The method is quite successful: on average, we could remove about $\frac{4}{5}ths$ of the features with an accuracy loss of only 1%.

Note that, in our results, we say that the size of a learnt theory is the number of features that it uses. While fewer features often generates smaller output theories, this is not necessarily always the case. For example, a decision tree with a single continuous feature can still have many nodes if that tree splits on multiple thresholds. However, the major benefit of measuring theory size in terms of number of used features is that this one measure can be applied to widely differing learning schemes. For example, there is no concept of "tree size" in the output of a Naive Bayes classifier. While we can't compare the simplicity of the learnt theory between Naive Bayes and C4.5, we can still compare the number of input features used by both schemes.

2. Treatment Learning with TAR2

We begin with a description of the TAR2 *treatment learner*. TAR2 seeks ranges of features that select for preferred classes. A repeated empirical observation of TAR2 selects only a very small number of treatments (see the experiments described in [15]). In the next section, we will experiment with using this property of TAR2 to find ignorable features.

TAR2 learns treatments and a *treatment* is constraint which, if applied to a data set, returns a subset of the data with a different distributions of classes. For

Items				*Criteria*
outlook	*temp(^oF)*	*humidity*	*windy?*	*class*
sunny	*85*	*86*	*false*	*none*
sunny	*80*	*90*	*true*	*none*
sunny	*72*	*95*	*false*	*none*
rain	*65*	*70*	*true*	*none*
rain	*71*	*96*	*true*	*none*
rain	*70*	*96*	*false*	*some*
rain	*68*	*80*	*false*	*some*
rain	*75*	*80*	*false*	*some*
sunny	*69*	*70*	*false*	*lots*
sunny	*75*	*70*	*true*	*lots*
overcast	*83*	*88*	*false*	*lots*
overcast	*64*	*65*	*true*	*lots*
overcast	*72*	*90*	*true*	*lots*
overcast	*81*	*75*	*false*	*lots*

Figure 2: A log of some golf-playing behavior.

example, consider the distribution of classes in the log of sporting activity seen in Figure 2. Before the treatment is applied, we play "lots" of golf in $\frac{6}{14}$ cases and "some" golf in $\frac{3}{14}$ cases. The treatment "outlook=overcast" is consistent only with the last four entries in the log and in all those cases, we play "lots" of golf. That is, if our users picked a vacation location with overcast weather, then TAR2 is predicted that we play "lots" of golf, all the time.

TAR2 learns treatments and the general form of a treatment is:

$$
\begin{array}{lll}
R_1 & if & Attr_1 = range_1 \wedge Attr_2 = range_2 \wedge ... \\
 & then & good = more \wedge bad = less \\
R_2 & if & Attr_1 = range_1 \wedge Attr_2 = range_2 \wedge ... \\
 & then & good = less \wedge bad = more
\end{array}
$$

where R_1 is the controller rule; R_2 is the monitor rule; *good* and *bad* are sets of classes that the agent likes and dislikes respectively; and *more* and *less* are the frequency of these classes, compared against the current situation, which we call the *baseline*. The nature of these output rules distinguishes TAR2 from many other learning strategies.

Association rule learning: Classifiers like C4.5 and CART learn rules with a single attribute pair on the right-hand side; e.g. *class= goodHouse*. Association rule learners like APRIORI [1] generate rules containing multiple attribute pairs on both the left-hand-side and the right-hand-side of the rules. That is, classifiers have a small number of pre-defined targets (the classes) while, for association rule learners, the target is less constrained.

General association rule learners like APRIORI input a set of D transactions of items I and return associations between items of the form $LHS \Rightarrow RHS$ where $LHS \subset I$ and $RHS \subset I$ and $LHS \cap RHS = \emptyset$. A common restriction with classifiers is that they assume the entire example set can fit into RAM. Learners like APRIORI are designed for data sets that need not reside in main memory. For example, Agrawal and Srikant report experiments with association rule learning using very large data sets with 10,000,000 examples and size 843MB [1]. However, just like Webb [19], TAR2 makes the "memory-is-cheap assumption"; i.e. TAR2 loads all it's examples into RAM.

Specialized association rule learners like CBA [13] and TAR2 impose restrictions on the right-hand-side. For example, TAR2's right-hand-sides show a prediction of the *change* in the class distribution if the constraint in the left-hand-side were applied. The CBA learner finds *class association rules*; i.e. association rules where the conclusion is restricted to one classification class feature. That is, CBA acts like a classifier, but can process larger datasets that (e.g.) C4.5. TAR2 restricts the right-hand-side features to just those containing criteria assessment.

Weighted-learning: Association rule learners such as MINWAL [4], TARZAN [16] and TAR2 explore *weighted learning* in which some items are given a higher priority weighting that others. Such weights can focus the learning onto issues that are of particular interest to some audience. For example TARZAN [16] swung through the decision trees generated by C4.5 [17] and 10-way cross-validation. TARZAN returned the smallest treatments that occurred in most of the ensemble that *increased* the percentage of branches leading to some preferred highly weighted classes and *decreased* the percentage of branches leading to lower weighted class. TAR2 was as experiment with applying TARZAN's tree pruning strategies directly to the C4.5 example sets. The resulting system is simpler, fast to execute, and does not require calling a learner such as C4.5 as a sub-routine.

Contrast sets: Instead of finding rules that describe the current situation, association rule learners like STUCCO [3] finds rules that differ meaningfully in their distribution across groups. For example, in STUCCO, an analyst could ask "what are the differences between people with Ph.D. and bachelor degrees?". TAR2's variant on the STUCCO strategy is to combine contrast sets with weighted classes with minimality. That is, TAR2 treatments can be viewed as the smallest possible contrast sets that distinguish situations with numerous highly-weighted classes from situations that contain more lowly-weighted classes.

Support-based pruning: In the terminology of APRIORI, an association rule has *support* s if $s\%$ of the D contains $X \wedge Y$; i.e. $s = \frac{|X \wedge Y|}{|D|}$ (where $|X \wedge Y|$ denotes the number of examples containing both X and Y). The *confidence* c of an association rule is the percent of transactions containing X which also contain Y; i.e. $c = \frac{|X \wedge Y|}{|X|}$.

Many association rule learners use *support-based pruning* i.e. when searching for rules with high confidence, sets of items $I_i, ... I_k$ are only be examined only if all its subsets are above some minimum support value. Support-based pruning is impossible in weighted association rule learning since with weighted items, it is not always true that subsets of *interesting* items (i.e. where the weights are high) are also interesting [4]. Another reason to reject support-based pruning is that it can force the learner to only miss features that apply to a small, but interesting subset of the examples [18].

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 *confidence1* pruning.

2.1. *Conf idence1 Pruning*

TAR2 targets the feature ranges that "nudge" a system away from undesired behavior and towards desired behavior. TAR2's score for each range is the *confidence1* measure. This value is high if a range occurs frequently in desired situations and infrequently in undesired situations. That is, if we were to impose this range as a constraint, then it would tend to"nudge" the system into better behavior.

To find confidence1, we assume that we can access $\$class$; i.e. some numeric value assigned to *class*. The class with the highest value is the *best* class. The *lesser* classes are the set of all classes, less the *best* class. Let $O[C]_{A.R}$ be the number of occurrences of some feature range in some class C; i.e.

$$O[C]_{A.R} = |A.R \wedge class = C \wedge D|$$

To generate confidence1, we compare the relative frequencies of an feature range in different classes. This comparison is weighted by the difference in the scores of the classes, and normalized by the total frequency count of the feature range; i.e.

$$\frac{\sum_{C \in lesser} ((\$best - \$C) * (O[best]_{A.R} - O[C]_{A.R}))}{|A.R \wedge D|}$$

2.2. *Example*

As an example of TAR2, suppose that users have scored the classes of Figure 2 as follows: "lots"=8, "some"=4, "none"=2; i.e. "lots" is the *best* class. The range *outlook=overcast* appears four, zero, and zero times when playing "lots", "some", and "none" golf (respectively). The confidence1 of *outlook=overcast* is therefore:

$$\frac{((8-2)*(4-0)) + ((8-4)*(4-0))}{4+0+0} = 10$$

Figure 3 shows the range of confidence1 seen in Figure 2. The confidence1 ranges shown in black are outstandingly high; i.e. these are the values may generate the best control treatments. TAR2 forms its treatments by exploring subsets of the ranges with outstandingly high confidence1 values.

TAR2's treatments are constraints which, if applied to the dataset, may reject certain examples. For example, the `controllerG` treatment of Figure 4 contains the constraint *outlook* = *overcast*. If we reject all items in the golf dataset that contradicts this constraint, then our golfers now play "lots", "some", and "none"

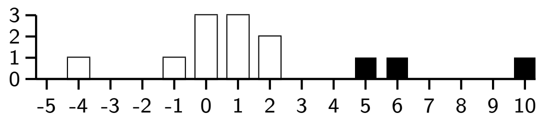

Figure 3: Frequency of conf idence1 generated from Figure 2. Assumes that numeric ranges have been divided into 3 *bands*. Outstandingly high conf idence1 values shown are in black. Y-axis is the number of ranges that have a particular confidence1 value.

```
controllerG if   outlook=overcast
then (230% more "lots" and no "some"
     and no "none").

monitorG if    90 <= humidity < 97
then (43% less "lots" and 5% less "some"
     and 167% more "none").
```

Figure 4: Control and monitor rules found from Figure 2. To control *outlook*, users could select a vacation location with overcast weather.

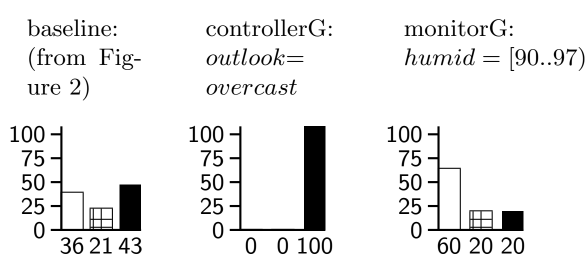

Figure 5: Percentage of classes seen in different situations. The left-hand-side histogram is a report of the class frequencies seen in Figure 2. The middle and right-hand-side histograms were generated by applying the treatments of Figure 4. KEY: none; some; lots.

```
input:  D          The examples.
        items      Attributes seen in the examples.
        best       The best combination of criteria.
        N          Desired size of LHS.
        promising  Threshold for a useful feature range.
        skew       Threshold for acceptable number of best entries in
                   treated.
        bands      Number of divisions within continuous ranges.

output: lhs        A conjunction of feature ranges
        rhs        a change in the class distributions
D1 ← discretize(D, bands)
temp ← baseline ← frequency(D1)
for attribute in items {
    for R in attribute.ranges {
        if   confidence1(attribute.R) ≥ promising
        then candidates ← candidates + attribute.R}}
for C ⊆ candidates where |C| = N {
    treated ← C ∧ D1
    result ← frequency(treated)
    if   result>temp and |best∧D1|/|best∧treated|>skew
    then   {lhs ← C
            rhs ← compare(baseline,result)
            temp ← result}}
if (lhs ≠ ∅ and rhs ≠ ∅) then return (lhs, rhs)
else return "no treatment"
```

Figure 6: The TAR2 algorithm.

golf in 100%, 0%, and 0% (respectively) of the constrained dataset (as shown in the middle histogram of Figure 5).

The monitor rule `monitorG` of Figure 4 was generated in a similar manner; but with the scoring system reversed; i.e. "lots"=2, "some"=4, "none"=8. In this case, "none" is the "*best*" class and TAR2 will find a treatment that selects for less golf behavior; i.e. $90 \leq humidity < 97$. After applying this constraint, the class distribution changes to the right-hand-side histogram of Figure 5.

2.3. *Inside TAR2*

TAR2 generates controller and monitor treatments. Monitors are generated using in same manner as generating controllers. However, before the monitor is generated, the scoring function for the criteria is reversed so TAR2 now seeks feature ranges that "nudge" a system into worse behavior. The rest of this section discusses how to generate controllers.

The TAR2 algorithm is shown in Figure 6. The `frequency` function counts the frequency of examples falling into different criteria. Using this function, a *baseline* class distribution is collected from D (this is used later to contrast different treatments) and copied to a *temp* variable (this is used to store the best distribution seen so far). The `compare` function compares two frequencies to generate reports like (e.g.) 43% less "lots" and 5% less "some" and 167% more "none". The `discretize` function divides the numeric ranges seen in the examples into *bands* number of groups. TAR2 was originally designed using a very simple discretization policy; i.e. TAR2 sorts the known values and divides into *bands*with (roughly) the same cardinality. It was anticipated that this policy would be too simplistic and would have to be improved. However, our empirical results (see below) were so encouraging that we were never motivated to do so.

Once a treatment is found, it is applied to the example set to create a *treated* example set; i.e. all the examples that don't contradict the proposed treatment (see line 8). A "good" treatment includes most of the examples that have the *best* criteria (e.g. in the golf example of Figure 2, *best*= playing "lots" of golf). The *skew*parameter is used at line 10 to reject "bad" treatments; i.e. those that don't contain enough of the *best* criteria. For example, at *skew*=5, at least 20% of the *best* criteria must appear in the treatment.

TAR2 explores subsets of the *ranges* found in a set of examples D (see line 7). Subset exploration is constrained to just the *ranges* with an outstandingly large confidence1 score (see line 5). Even with this restriction, there are still an exponential number of such subsets. Hence, to be practical, TAR2 must seek the *minimal* possible number of control actions and monitors. Accordingly, the user of TAR2 constrains its learning to rule conditions of size N, where N is small (see line 7). Often, effective treatments can be found using $N \leq 4$ which suggests that narrow funnels existed in the datasets used for our case studies.

2.4. *Comparisons*

This section compare how a treatment learner like TAR2 and a decision tree learner handles the same data set.

Figure 7 shows a tree learned by C4.5 [17]. This tree is generated from hundreds of examples of houses in the Boston area. Each branch of the tree tell us how we might recognize *high*, *mediumHigh*, *mediumLow* and *low* quality houses. TAR2, applied to same data set, learns the controller and monitor rules of Figure 8.

For the purposes of learning tiny theories, the important aspect of Figure 7 and Figure 8 is that they use only a subset of the 13 features available in the housing data set. That is, only *some* of the available features were useful when learning treatments or classifiers via entropy-based methods.

Note that for this housing example, TAR2 selected fewer features than C4.5: C4.5's learnt theory needed seven features while TAR2's treatments only needed four. It has often been observed that TAR2's theories use far fewer features than classifiers (see the experiments described in [15]). Perhaps TAR2 might be useful for finding core features that generate tiny classifiers? The SELECT algorithm tests this speculation.

3. SELECT

SELECT is a loop around TAR2. In each loop, a different class is declared to be the *best* class, and the features found in the resulting treatments are added to a set of SELECTed features. That is, the SELECTed set contains any feature that was useful for "nudging" towards any class.

```
lstat <= 11.66
|   rm <= 6.54
|   |   lstat <= 7.56 THEN  medhigh
|   |   lstat > 7.56
|   |   |   dis <= 3.9454
|   |   |   |   ptratio <= 17.6 THEN  medhigh
|   |   |   |   ptratio > 17.6
|   |   |   |   |   age <= 67.6 THEN  medhigh
|   |   |   |   |   age > 67.6 THEN  medlow
|   |   |   dis > 3.9454 THEN  medlow
|   rm > 6.54
|   |   rm <= 7.061
|   |   |   lstat <= 5.39 THEN  high
|   |   |   lstat > 5.39
|   |   |   |   nox <= 0.435 THEN  medhigh
|   |   |   |   nox > 0.435
|   |   |   |   |   ptratio <= 18.4 THEN  high
|   |   |   |   |   ptratio > 18.4 THEN  medhigh
|   |   rm > 7.061 THEN  high
lstat > 11.66
|   lstat <= 16.21
|   |   b <= 378.95
|   |   |   lstat <= 14.27 THEN  medlow
|   |   |   lstat > 14.27 THEN  low
|   |   b > 378.95 THEN  medlow
|   lstat > 16.21
|   |   nox <= 0.585
|   |   |   ptratio <= 20.9
|   |   |   |   b <= 392.92 THEN  low
|   |   |   |   b > 392.92 THEN  medlow
|   |   |   ptratio > 20.9 THEN  low
|   |   nox > 0.585 THEN  low
```

Features used in the decision tree:

$age =$	proportion of houses built prior to 1940
$b =$	information on racial mixture in the suburb
$dis =$	weighted distances to five employment centers
$lstat =$	living standard
$nox =$	nitric oxides concentration
$ptratio =$	parent-teacher ratio at local schools
$rm =$	number of rooms

Figure 7: A decision tree learned from the HOUSING database using WEKA's J4.8 algorithm [20] with the command line `J4.8 -C 0.25 -M 10`.

```
controllerH if   rm <= 6.6 AND ptratio <= 15.9
then ("high" increases by 334% and "medhi" decreases by 90%
      and no "medlo" and no "lo").

monitorH if    0.6   <= nox   < 1.9 AND
      17.16 <= lstat < 39
then (467% more "low" AND "medlow" decreases by 95.3%
     and no "medhi" and no "high").
```

Figure 8: Control and monitor rules found from the UCIrvine housing example. Percent changes are reported compared to the baseline class frequencies in original data set.

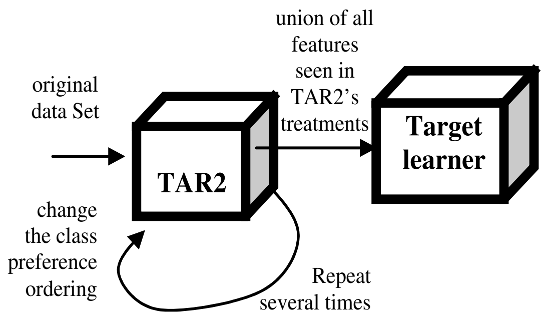

Figure 9: An illustration of SELECT

- Initialize the SELECTed features to nil.
- For each class in turn, declare it to be TAR2's "best" class. Then:
  - Set treatment size N to 1
  - Find the "best" treatment of size N via TAR2.
  - If the score of the best treatment is no better than that of the best treatment of size N-1, then
    - Add the features seen in the best treatment to SELECTed.
  - Else, N++ and loop.
- Collect the average accuracy seen in a 10-way cross validation of the target learner using
  - just the features seen in SELECTed
  - all features

Figure 10: SELECT algorithm.

After SELECT's loops through each class, some target learner is executed using all the features and the SELECTed features. Equation 1 is then applied to test if the theory learnt from the SELECTed theory is better than the theory learnt from all features. For more details on SELECT, see Figure 9 and Figure 10.

An important methodological point of SELECT is that TAR2 is not used in the 10-way to assess the SELECTed features. TAR2 is a novel machine learning algorithm that has yet to gain wide acceptance. Hence, SELECT relegates TAR2 to a pre-processor and uses commonly-used machine learners to assess the results.

Figure 1 was generated using SELECT and C4.5 as the target learner. As mentioned in the introduction, the SELECTed features were always few and usually satisfied Equation 1. But a complete assessment of SELECT requires a comparison of the Figure 1 results with other approaches. The rest of this paper describes such a comparison.

	number of attributes										
before:	10	13	15	180	22	8	25	36	6	6	6
after:	2	2	2	11	2	1	3	12	1	1	2
reduction:	80%	84%	87%	94%	90%	87%	88%	67%	83%	83%	67%
Δaccuracy:	0%	6%	5%	4%	2%	1%	0.5%	0%	-25%	6%	7%

Figure 11: Feature subset selection using a WRAPPER of a decision tree leaner. The Δ accuracy figure is the difference in the accuracies of the theories found by decision tree learner using the $before$ and $after$ features. From [11].

3.1. ***Feature Subset Selection***

SELECT belongs to a class of algorithms called *feature subset selection* (FSS) methods. This section reviews some FSS algorithms.

FSS is the process of identifying the most promising features in a given dataset. Datasets used in practical data mining applications have a large number of features. These data sets often contain several extraneous features which can reduce the efficiency of the learning algorithm. Feature subset selection helps us identify the important attributes and remove redundant ones. If only the most relevant features were to be selected and given to the learning algorithm they can produce smaller theories. This enhances the understanding of the dataset or domain under consideration. Dimensionality reduction also speeds up the learning process.

A repeated result in the FSS field is that ignoring features need not degrade classifier accuracy. How can ignoring information be useful? Kohavi & John [11] review studies with Naive Bayes classifiers. The accuracy of such classifiers decreases very slowly as irrelevant features are added to an instance set. However, the accuracy of the same classifiers can degrade sharply as the number of correlated features increase.

Another explanation for the success of ignoring features is offered by Witten & Frank [20]. They note that effective generalization requires numerous examples. Decision tree learners recursively split instances by ranking features according to how much they decrease the diversity of the classes in the split sets. As learning progresses, fewer and fewer instances are available to learn the next sub-tree. If the instances contain too many features of similar rank, then many splits are quickly generated. Hence, instances become sparser in the sub-trees, and effective generalization becomes harder.

The rest of this section describes several FSS methods.

3.1.1. *WRP: Wrapper Subset Evaluation*

In the WRAPPER 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.

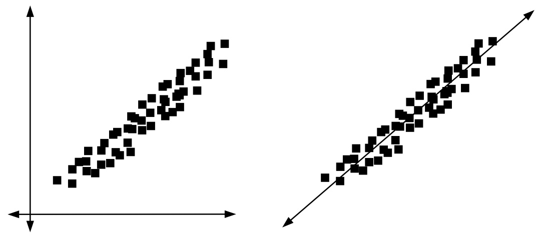

Figure 12: Transformation of axis.

Figure 11 shows some WRAPPER results from experiments by Kohavi and John [11]. In their experiments, 83% (on average) of the measures in a domain could be ignored with only a minimal loss of accuracy.

The advantage of the WRAPPER approach is that, if some target learner is already implemented, then the WRAPPER is simple to implement. The disadvantage of the wrapper method is that each step in the heuristic search requires another call to the target learner; i.e. it may be very slow.

For the results shown below, we will use a WRAPPER of two target learners: a decision tree learner (C4.5) and a Naive Bayes classifier.

3.1.2. *PCA: Principal Component Analysis*

Principal components analysis (PCA) [5] identifies 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. For example, the data shown in two dimensions of Figure 12 (left-hand-side) could be approximated in a single transformed dimension, (right-hand-side).

3.1.3. *IG: Information Gain Attribute Ranking*

This is a simple and fast method for feature ranking [6]. This method measures the split criteria of the class before and after observing a feature. The differences in the split criteria gives a measure of the information gained because of that feature [17]. A final comparison of this measure is used in feature selection.

3.1.4. *RLF: Relief*

Relief is an instance based learning scheme [10,12]. 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 [12]. For data sets with multiple classes, the nearest neighbors for each class that is different from the current sampled instance are selected and the contributions are determined by using the class probabilities of the class in the dataset.

3.1.5. *CFS:Correlation-based Feature Selection*

CFS uses subsets of features [8]. 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.

3.1.6. *CBS: 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 [2].

3.2. *Experiments*

SELECT was used to find a subset of these available features. Theories were learnt using either *all* or the *selected* features by the C4.5 decision tree learner or a Naive Bayes classifier. These two learners were deliberately selected to assess the utility of FSS on radically different learning schemes:

- Decision tree learners recursively split instances by ranking feature ranges according to how much they decreases the diversity of the classes in the split sets.
- Naive Bayes classifiers work in a very different manner. Statistics are collected on the distribution of feature ranges in different classes. Those statistics are used to estimates the probability that some new combination of features belongs to a certain class.

We used the implementation of C4.5 and Naive Bayes classifier found in WEKA: the Waikato Environment for Knowledge Analysis [20]. The WEKA is a free, JAVA-based, open source, GUI tool that provides a rich variety of machine learners, preprocessing tools, and visualization tools.

Our experiments were run on 10 datasets. These datasets, described in Figure 13, originally come from the UCI (University of California at Irvine) repository. These datasets had a wide range of nominal and numeric features. The size of these datasets varied from a few hundred to a few thousand instances.

The last column of Figure 14 shows the number of features found by SELECT. The middle columns come from FSS by Hall & Holmes [7]. In the Hall & Holmes

dataset	instances	numeric	nominal	classes
anneal	898	6	32	5
breast-c	286	0	9	2
credit-g	1000	7	13	2
diabetes	768	8	0	2
horsecolic	368	7	15	2
ionosphere	351	34	0	2
lymph	148	3	15	4
segment	2310	19	0	7
soybean	683	0	35	19
vote	435	0	16	2

Figure 13: Datasets used.

	original	ig	cfs	cbs	rlf	wrp	pc	select
Anneal	38	17	21	15	20	18	36	▶ 7
breast-c	9	4	4	7	7	4	4	▶ 2
credit-g	20	8	7	8	9	8	▶ 4	5
Diabetes	8	33	3	4	4	4	6	▶1
Horse colic	22	4	4	▶2	3	5	3	▶ 2
Ionosphere	34	12	7	9	9	7	10	▶ 2
lymph	18	6.8	5.3	4	4	6	9	▶ 3
Segment	19	16	12	9	13	9	16	▶ 4
Soybean	35	19	24	35	32	19	30	▶ 16
vote	16	12	10	▶ 6	11	9	11	▶ 6

Figure 14: Features selected using C4.5. Black triangles denote which FSS method found the smallest set of features.

experiments, WRP, PCA, IG, CBS, RLF, CFS and CBS were generate a sorting of the available features. For N set from 1 to the maximum number of features, the top N features were passed to some target learner (C4.5 or Naive Bayes). Hall & Holmes returned the N features that generated the maximum accuracy. Figure 14 is therefore a comparison of two FSS methods:

- SELECT vs
- Hall & Holmes using {WRP, PCA, IG, CBS, RLF, CFS} to rank features then C4.5 to assess the top N features.

Similarly, Figure 15 is a comparison between

- SELECT vs
- Hall & Holmes using the same FSS methods to rank features then Naive Bayes to assess them.

Note that the last column of Figure 14 and Figure 15 are the same since they both report the same results from SELECT.

Hall & Holmes repeated their FSS procedure repeated ten times, each time using a ten-way cross-validation on their FSS methods to rank the features, then passing the top N features to some target learner to assess their utility. Hence the middle columns of Figure 14 and Figure 15 are actually the average, rounded

	original	ig	cfs	cbs	rlf	wrp	pc	select
anneal	38	10	▶ 4	5	39	7	25	7
breast-c	9	4	7	6	5	3	3	▶ 2
credit-g	20	13	14	14	20	12	11	▶ 5
diabetes	8	3	4	4	6	3	4	▶ 1
horse colic	22	9	4	4	23	6	6	▶ 2
ionosphere	34	8	8	11	18	13	12	▶ 2
lymph	18	17	13	14	15	2	13	▶ 3
segment	19	11	11	5	15	8	9	▶ 4
soybean	35	31	31	33	36	26	21	▶ 16
vote	16	▶ 1	2	3	15	▶ 1	3	6

Figure 15: Features selected using Naive Bayes. Black triangles denote which FSS method found the smallest set of features.

number of features found to generate maximum accuracy in a ten-times ten-way-FSS followed by a ten-way-cross-val. Hall & Holmes argue that such a laborious method is required to compare different classes of FSS tools.

To our way of thinking, the analysis of Hall & Holmes is perhaps over-elaborate. An FSS can be viewed as a black-box preprocessor to a machine learner. This black-box method generates features (by any method), and the merits of those features are assessed via a single 10-way experiment with some target learner.

How can we reject the Hall & Holmes experimental method, yet still compare our results to theirs? Returning to the FSS-as-black-box metaphor, we argue that Hall & Holmes are delivering a set of features via some method and the *size* of that set can be compared to the *size* of the features found via SELECT. However, it would be an error to compare *accuracies* between SELECT and the Hall & Holmes study since the ten-times ten-way FSS sub-divides the available data into a smaller set that what is offered to SELECT. Hence, we will assess the accuracies of the theories learnt from the SELECTed attributes using our goal (Equation 1) and not via comparison with the accuracies seen in the Hall & Holmes study.

The key features of Figure 14 and Figure 15 is that SELECT found the smallest subset of any method studied here in 17 of the 20 experiments. For decision tree target learners, SELECT found the smallest subset in 9 of the 10 experiments. For Naive Bayes target learners, SELECT found the smallest subsets in 8 of the 10 experiments.

Hall & Holmes do not offer runtimes for their FSS methods. Hence, we can't compare the runtimes of SELECT with the other FSS results shown here. However, we have some evidence that SELECT will be a much faster than some FSS methods. Kohavi & John [11] report that their WRAPPER method can take up to hundreds or thousands of seconds to terminate. Total SELECT runtime for any of the domains studied here is much faster: i.e. always less than ten seconds.

Figure 16 show the average accuracies seen in 10-way cross validation using the two target learners using all or the features SELECTed by our methods. The features rejected by SELECT changed classification accuracy very little. In only

	C4.5			Naive Bayes		
	$c1$	$c2$		$n1$	$n2$	
	all	selected	$\frac{(c1-c2)}{c1}$	all	selected	$\frac{(n1-n2)}{n1}$
anneal	98.2	98.2	0.00%	86.6	84.3	2.66%
breast-c	75.2	75.2	0.00%	74.1	75.2	-1.48%
credit-g	73.9	72.3	2.17%	75.9	74.3	2.11%
diabetes	74.5	72.8	2.28%	76	74.6	1.84%
horsecolic	85.3	81.5	▶ 4.45%	78.8	79.6	-1.02%
ionosphere	88.6	87.8	0.90%	82.9	87.5	-5.55%
lymph	76.4	74.3	2.75%	81.8	77.7	▶ 5.01%
segment	97.1	96.6	0.51%	79.8	86.3	-8.15%
soybean	92.4	93	-0.65%	92.7	93	-0.32%
vote	95.9	96.1	-0.21%	90.1	94.9	-5.33%
		average:	1.22		average:	-1.02

Figure 16: Accuracies of theories using all or the SELECT-ed features. Black triangles denote cases where the SELECT-ed features generated a theory that violate our goal of Equation 1. Negative accuracies mean accuracy increased using the SELECTed features.

two cases out of twenty did SELECT violate our goal statement of Equation 1. The largest difference seen in Table 3 was the 5.01% loss seen for the *vote* domain and such a large difference was not the usual case. On average, the classification accuracies changed by around 1%. More specifically:

- A 1.22% average relative *decrease* for accuracy using C4.5 as the target learner.
- A 1.02% average relative *increase* for accuracy using Naive Bayes as the target learner.

In summary, of the FSS methods studied here, SELECT usually found the smallest feature subsets and those subsets usually resulted in an acceptable accuracies.

4. Conclusions

Theories generated by data miners aren't useful if users can't or won't read them. We work with business users that declines to read complex theories. These users wish to be shown the smallest possible "useful" theory.

In this paper, we have defined a theory *new* to be more "useful" than another theory *old* if *new* is uses far fewer features than *old*, and *new* is not "much less accurate" that *old*. Our user community asserts that an acceptable loss of accuracy is:

$$\frac{accuracy(old) - accuracy(new)}{accuracy(old)} < 3\%$$

Under that assumption, we have developed the SELECT feature subset selector method. SELECT uses the TAR2 treatment learner as a sub-routine. In comparisons with other feature subset selectors, we have shown that:

- The feature selected by SELECT were usually smaller that features selected by other FSS methods.

- Measured in terms of averages over a 10-way cross validation, the impact on accuracy was minimal and acceptable (where "acceptable" is defined as per Equation 1).
- SELECT runs faster than certain other leading FSS methods such as WRAPPER (but the evidence for this last conclusion is somewhat limited).

Future work would involve trying this approach on more datasets and with datasets have more number of features

Acknowledgements

This research was conducted at West Virginia University under NASA contract NCC2-0979 and NCC5-685. The work was sponsored by the NASA Office of Safety and Mission Assurance under the Software Assurance Research Program led by the NASA IV&V Facility. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government.

References

pt!

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Databases*, 1994. Available from `http://www.almaden.ibm.com/cs/people/ragrawal/papers/vldb94_rj.ps`.

[2] H. Almuallim and T. Dietterich. Learning with many irrelevant features. In *The Ninth National Conference on Artificial Intelligence*, pages pp. 547–552. AAAI Press, 1991.

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

[4] C. Cai, A. Fu, C. Cheng, and W. Kwong. Mining association rules with weighted items. In *Proceedings of International Database Engineering and Applications Symposium (IDEAS 98)*, August 1998. Available from `http://www.cse.cuhk.edu.hk/~kdd/assoc_rule/paper.pdf`.

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

[6] 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.

[7] M. Hall and G. Holmes. Benchmarking attribute selection techniques for discrete class data mining. *IEEE Transactions On Knowledge And Data Engineering (to appear)*, 2003.

[8] M. A. Hall. *Correlation-based feature selection for machine learning*. PhD thesis, Department of Computer Science, University of Waikato, Hamilton, New Zealand, 1998.

[9] Y. Hu. Treatment learning: Implementation and application, 2003. Masters Thesis, Department of Electrical Engineering, University of British Columbia.

[10] K. Kira and L. Rendell. A practical approach to feature selection. In *The Ninth International Conference on Machine Learning*, pages pp. 249–256. Morgan Kaufmann, 1992.

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

[12] I. Kononenko. Estimating attributes: Analysis and extensions of relief. In *The Seventh European Conference on Machine Learning*, pages pp. 171–182. Springer-Verlag, 1994.

[13] B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *KDD*, pages 80–86, Sept 1998. Available from `http://citeseer.nj.nec.com/liu98integrating.html`.

[14] T. Menzies, E. Chiang, M. Feather, Y. Hu, and J. Kiper. Condensing uncertainty via incremental treatment learning. In T. M. Khoshgoftaar, editor, *Software Engineering with Computational Intelligence*. Kluwer, 2003. Available from `http://menzies.us/pdf/02itar2.pdf`.

[15] T. Menzies and Y. Hu. Just enough learning (of association rules): The TAR2 treatment learner. In *Journal of Data and Knowledge Engineering (submitted)*, 2002. Available from `http://menzies.us/pdf/02tar2.pdf`.

[16] T. Menzies and E. Sinsel. Practical large scale what-if queries: Case studies with software risk assessment. In *Proceedings ASE 2000*, 2000. Available from `http://menzies.us/pdf/00ase.pdf`.

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

[18] K. Wang, Y. He, D. Cheung, and F. Chin. Mining confident rules without support

requirement. In *10th ACM International Conference on Informationand Knowledge Management (CIKM 2001), Atlanta*, 2001. Available from `http://www.cs.sfu.ca/~wangk/publications.html`.

[19] G. Webb. Efficient search for association rules. In *Proceeding of KDD-2000 Boston, MA*, 2000. Available from `http://citeseer.nj.nec.com/webb00efficient.html`.

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