Chapter 2

On Induction

2.1 Introductory remarks

In concept learning, a learner is given a training set containing labeled instances, which are typically represented by a list of feature-value pairs. Its task is to inductively construct a hypothesis that accurately predicts the label of novel instances. Past studies have provided a number of successful algorithms such as ID3[67] or C4.5[68]. Unfortunately they degrade in performance, both in terms of prediction accuracy and of speed, when they are given data with many features that are not necessary for predicting the desired label. Because raw data in real-world applications usually contain a lot of irrelevant features, feature selection is crucial for successful application of machine learning algorithms to the real-world applications. That fact has boosted studies in the area[31, 1, 35, 72, 7, 36, 40]. The author will examine some of them later in this chapter.

In section 2.2, the author defines the problem dealt in this chapter, while giving some notational conventions. Section 2.3 presents a new feature subset selection algorithm called EBFS. Section 2.4 evaluates EBFS's performance through experiments. Section 2.5 discusses relationship between EBFS and some previous works, comparing EBFS's performance with other two feature subset selection algorithm. In section 2.6 theoretical background of EBFS is discussed. Finally in section 2.7 the author gives conclusions and remarks for future works.

2.2 Feature subset selection problem

First let us make some notational conventions. A particular learning algorithm is denoted by $\mathcal{M}$. The input to the algorithm is a multiset of

training instances and called a *training set*. Each instance $\vec{X}$ is a vector $(x_1, x_2, \ldots, x_m)$, where x_i denotes a value of the i-th feature X_i. X denotes a set of the features $\{X_1, X_2, \ldots, X_m\}$. The training instance is a tuple $\langle \vec{X}, y \rangle$ where y is the label, or output. X is called a *feature set* of the training instance $\langle \vec{X}, y \rangle$ or the instance $\vec{X}$.

T^X denotes a multiset of the training instances whose feature sets are X. For $W \subseteq X$, $T^{W,X}$ denotes a multiset of training instances whose feature set is W, which is obtained by *shrinking* each element of T^X. Namely, there are one-to-one correspondences between T^X and $T^{W,X}$ and for each element $\langle \vec{W}, y_1 \rangle$ in $T^{W,X}$ if the corresponding element of T^X is $\langle \vec{X}, y_2 \rangle$ then $y_1 = y_2$ and if $W_i = X_j$ then $w_i = x_j$.

The task of the learning algorithm is to induce a structure (e.g., a decision tree, a set of rules, a neural network) such that, given a new instance, it is possible to accurately predict the label of the instance. The structure that the algorithm $\mathcal{M}$ yields is denoted $S(\mathcal{M}, T^X)$, where T^X is given to $\mathcal{M}$ as a training set.

Intuitively the feature subset selection problem is defined as follows. Let $A = \{A_1, \ldots, A_m\}$ be a set of features potentially available to a learning algorithm M. Suppose there is a performance measure g for $S(\mathcal{M}, T^{X,A})$, by which $g(S_1) \leq g(S_2)$ means that S_2's performance is better than or equal to S_1's.

Let us use a term *test set* to denote a set of instances whose labels are known and are used to evaluate the algorithm's performance. The test set is represented by the same symbol as a training set because they share the equivalent structure. Let $gl(S, T)$ denote a structure S's performance which is measured using T as a test set.

The *feature subset selection problem* is to find a minimal subset A^* of A which achieves the best $g(S(\mathcal{M}, T^{A^*,A}))$.

From the definition we can say that an irrelevant feature is a feature such that performance of the learned structure is worsened by adding it to the original feature set.

There are several ways to define g. In this chapter we focus on the structure's predicting ability to unseen data and its relative simplicity.

2.3 New explanation-based method

2.3.1 The heuristic function H

First let we start from why irrelevant features can worsen performance of learned structure. The irrelevant features can worsen the prediction ability

of the structure for unseen data because they can help the induced structure overfit to the given training set. A feature subset selection algorithm should prevents the overfitting happening from the very beginning by removing those features which help the induction algorithm only for a certain training set.[1]

Then we need to estimate to possibility that each feature will cause the problem. Suppose our $\mathcal{M}$ is an ID3-like decision tree induction algorithm, which takes the divide-and-conquer approach, in each step a feature is chosen based on its property excluding its relationship to the other features, and does not perform any pruning.

When an induced decision tree is given a test instance, it will yield a label prediction and an *explanation path* for the instance, i.e., a decision path in the tree where the instance follows from a root node to a leaf node. If the prediction is correct then the explanation path is called a *successful path*, otherwise a *failed path*.

Let us think about frequency of feature appearing in failed paths. Because a failed path must contain an irrelevant feature[2], the feature's frequency of appearance in the failed paths, normalized by its frequency of appearance in the successful paths, is likely to reflect how irrelevant the feature is. This is a heuristic that the author presents here. Namely, for a training set T^X_{train} and feature X_i, the heuristic $H(X_i, T^X_{train})$ is defined as follows.

$H(X_i, T^X_{train}) \stackrel{def}{=} G - F$ such that

$$G \stackrel{def}{=} \begin{cases} 0 & \text{if } |TEST|_G = 0 \\ \frac{G_{X_i}}{|TEST|_G} & \text{otherwise} \end{cases}$$

$$F \stackrel{def}{=} \begin{cases} 0 & \text{if } |TEST|_F = 0 \\ \frac{F_{X_i}}{|TEST|_F} & \text{otherwise} \end{cases}$$

where G_{X_i} denotes the number of successful paths which include a node that tests X_i, $|TEST|_G$ denotes the number of elements in the test set whose labels are accurately predicted by the structure, F_{X_i} denotes the number of failed paths which include a node that tests X_i, and $|TEST|_F$ denotes the number of elements in the test set whose labels are not correctly predicted by the structure. A feature X_i with low $H(X_i, T^X_{train})$ can be expected to be irrelevant. Changing the test set by means of cross validation (CV), accumulation of $H(X_i, T^X_{train})$ can be more reliable.

[1] In the case of decision tree induction the overfit problem is usually solved by *pruning*, i.e., simplifying a decision tree after it is created. The author will show that the feature selection can be as effective as the pruning on the problem, with experimental results.

[2] Theoretical discussion will be shown later.

2.3.2 Greedy algorithm

The algorithm shown in figure 2.1 is based on *backward stepwise elimination* which is commonly seen in statistics literature. It starts from the whole set of features and greedily removes features which are most likely to be irrelevant in each step.

Algorithm 2.1 EBFS(X, T^X, $\mathcal{M}$, m)
1 $(e, h) \leftarrow$ EBFS_induction(X, T^X, $\mathcal{M}$, m)
2 $best \leftarrow$ EBFS_sub(X, T^X, $\mathcal{M}$, m, e, h)
3 return $best$

Algorithm 2.2 EBFS_sub(X, T^X, $\mathcal{M}$, m, e, h)
1 $h\_array[X] \leftarrow e$
2 $worst \leftarrow min_Y(h[Y])$
3 $rej \leftarrow \{X_i | h[X_i] = worst\}$
4 **if** $rej = X$ **then**
5 return argmax $h\_array[X]$
6 **else**
7 $N \leftarrow X - rej$
8 $(e', h') \leftarrow$ EBFS_induction(N, $T^{N,X}$, $\mathcal{M}$, m)
9 **if** $e' \ll e$ **then**
10 return argmax $h\_array[X]$
11 **else**
12 return EBFS_sub(N, $T^{N,X}$, $\mathcal{M}$, m, e', h')
13 **end**
14 **end**

Algorithm 2.3 EBFS_induction(X, T^X, $\mathcal{M}$, m)
1 **for** $X_i \in X$ **do**
2 $h[X_i] \leftarrow 0$
3 **end**
4 $E \leftarrow 0$
5 **for** $j = 1$ to m **do**
6 $T^X_{train} \leftarrow$ a training set for the j-th block of m-fold CV
7 $e \leftarrow gl(S(\mathcal{M}, T^X_{train}), T^X - T^X_{train})$
8 $E \leftarrow E + e$
9 **for** $X_i \in X$ **do**
10 $h[X_i] \leftarrow h[X_i] + H(X_i, T^X_{train})$
11 **end**
12 **end**
13 return (h, E)

Figure 2.1: Greedy algorithm based on $H(X_i, T^X_{train})$

Each time average performance of the remaining feature subset is estimated by CV. This CV is the same CV that is used to accumulate the value of H. The iteration continues until $X - \{X_i\}$ yields much worse result than X. At that time the algorithm returns a feature subset which yields the best result so far.

Because the heuristic is essentially based on *explanation* created by a decision tree, the author calls the feature subset selection method *explanation-based feature subset selection*, or *EBFS* for short.

2.4 Experimental results

In order to evaluate the explanation-based feature subset selection, the author has conducted experiments on several datasets. C4.5, which comes with Quinlan's book[68], is used as the induction algorithm. As for the CV to accumulate the heuristic H and evaluate a feature subset, 10-fold CV is used.

The algorithm is evaluated through CV. Note that this CV is purely used for evaluation of the algorithm, not a feature subset. To avoid confusion let us call the CV used for evaluation of the algorithm *outer*-CV[37].

The experiments were performed using 30 datasets, with 22 of them are taken from the UC-Irvine repository[45].

Two datasets are created by extending UCI datasets **Vote** and **Credit**. **Vote100** is created by adding 100 features, which randomly take a value from three candidates, to **Vote**. **Credit100** is created by the same procedure from **Credit**. **Vote100** and **Credit100** are intended to be tougher problems for feature subset selection algorithms.

The other six datasets are artificial and created by the author, to study effects of noisy, correlated and dependent features on the algorithm.

In each case data obtained from four algorithms are shown, i.e., C4.5 without the pruning, C4.5 with the pruning, the EBFS with C4.5 without the pruning, and the EBFS with C4.5 with the pruning. Each algorithm is denoted as **C4.5**, **C4.5P**, **EBFS** and **EBFSP** respectively. When the results are shown in a bar graph, the following legends are used.

C4.5 **EBFS**
C4.5P **EBFSP**

2.4.1 UCI and UCI-derived datasets

In this subsection, the author show results from the UCI datasets, **Credit100** and **Vote100**. For each dataset the author shows error rates on a test set

and complexity of induced decision trees in terms of the number of nodes they have.

Error rates

Figure 2.2 shows the result on the error rates.

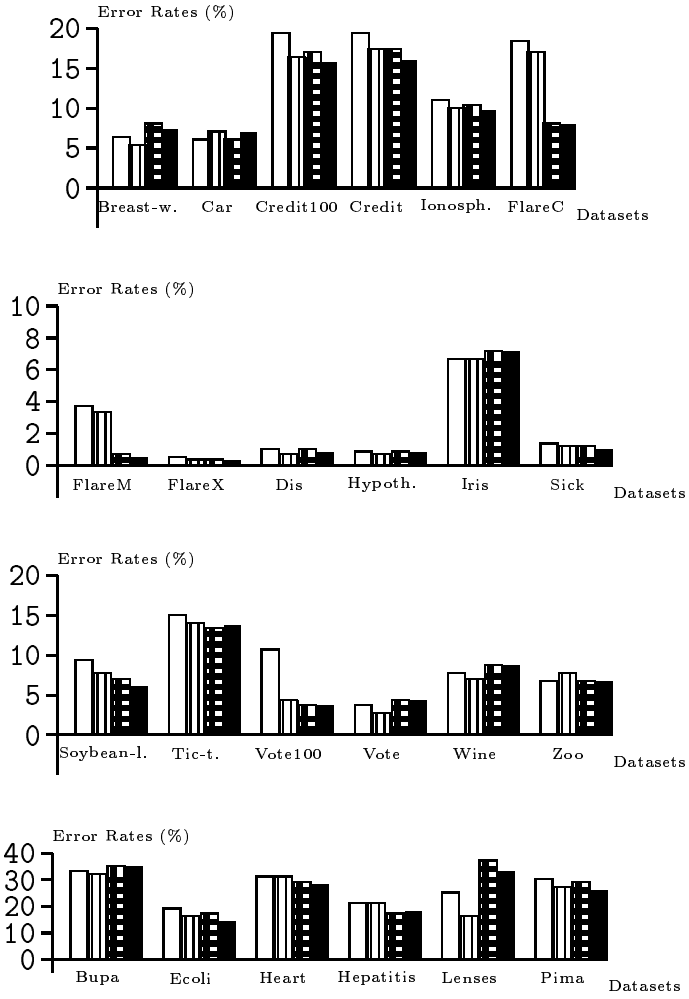

Figure 2.2: Error rates for UCI datasets

In most cases EBFS improved the error rate in a sense that **EBFS** and **EBFSP** are better than **C4.5** and **C4.5P** respectively. **EBFS** was outperformed by **C4.5** only in 6 cases out of 24 cases. For **EBFSP** that is 5 out of 24. Even in those cases the degradation is insignificant, with an exception of **Lenses** dataset. A possible explanation to this is the number of instances of

the dataset. **Lenses** has only 24 instances, significantly fewer than others. In that case one failed path can create too much impact to the heuristic H.

Although improvements are also insignificant in most cases, in cases like **Credit100**, **Credit**, **FlareC**, **FlareM** and **Soybean-l.** EBFS made a substantial contribution.

It is interesting to see which one of **C4.5P** and **EBFS** performed better, because the pruning is a usual choice to avoid overfitting in decision tree learning. Among 24 confrontations, **C4.5P** won 12 and **EBFS** won 10. We should not jump to any conclusion from here. However, the fact that **EBFSP** is generally better than **C4.5P** suggests EBFS's worth.

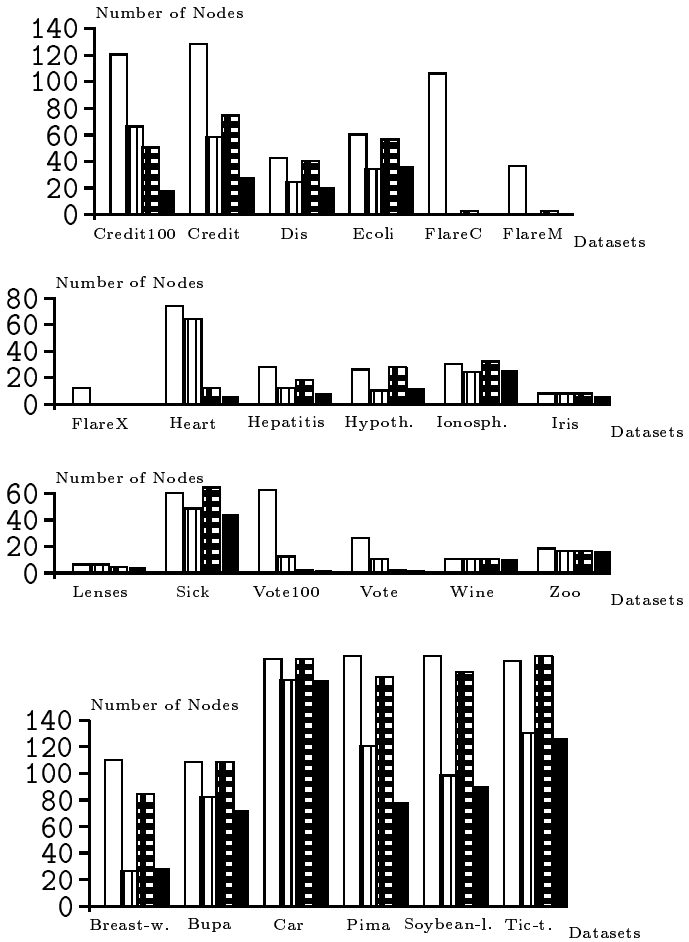

Figure 2.3: Number of nodes for UCI datasets

Size of the decision trees

Figure 2.3 shows the result on the size of the decision trees, in terms of the number of nodes of the tree. Like the case of error rates, **EBFS** and **EBFSP** resulted smaller trees than **C4.5** and **C4.5P** respectively. That is true even for the dataset where EBFS improved the accuracy greatly.

2.4.2 Artificial datasets

In this subsection, the author show results from the artificial datasets. All six dataset belong one family. The base dataset **Myd60** is created as the following.

- There are six features a, b, c, d, e, f. They are all three-valued, take one of $\{0, 1, 2\}$.
- The class is two-valued $(0, 1)$. The class is 1 if $(d = 0 \wedge e = 2) \vee (d = 2 \wedge e = 0)$ and otherwise 0.
- Value of e depends on a and b. $e = 1$ if $a = 1 \vee b = 1$, $e = 2$ if $(a = 0 \wedge b = 2) \vee (a = 2 \wedge b = 0)$ and otherwise $e = 0$.
- c is correlated to the class. Its value is equal to the class in 75% of the cases, otherwise random.
- f's value is randomly taken from $\{0, 1, 2\}$.
- 500 instances are created by choosing a, b and d randomly from $\{0, 1, 2\}$.

The other datasets, **Myd61**, **Myd62**, **Myd63**, **Myd64** and **Myd65** are derived from **Myd60** by adding noise to a feature e, whose noise levels are 10%, 20%, 30%, 40% and 50% respectively. The aim of this experiment is to see how EBFS handles dependent features (a, b, e), a correlated feature (c), an irrelevant feature (f) and a noisy feature (e). For each dataset the author shows error rates and the number of nodes. Furthermore, probabilities of removing unwanted features are shown as functions of the noise level.

Error rates

Figure 2.4 shows error rates on the artificial datasets.

We can see improvements by EBFS in **Myd62** to **Myd65**. However the improvements are very small. This can be explained as follows. In these experiments with the artificial datasets, plain C4.5 performed pretty well,

seldom choosing c, e, or f. That means few failed paths were available to EBFS in the backward stepwise elimination loop. Because the heuristic H's power essentially comes from the failed path, EBFS cannot make a good estimate for usefulness of each features.

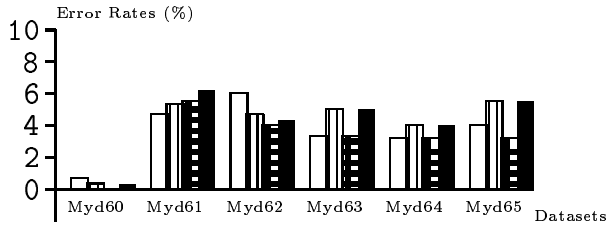

Figure 2.4: Error rates for artificial datasets

Size of the decision trees

Figure 2.5 shows the number of nodes of the decision trees on the artificial datasets. Again, we have to see almost the same trend as we saw in figure 2.4.

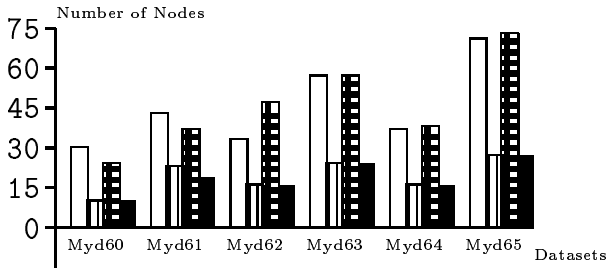

Figure 2.5: Number of nodes for artificial datasets

Probability of removing unwanted features

It is difficult to say anything about EBFS's ability in this domain from the previous two graphs. Figure 2.6 shows if EBFS removed unwanted features in each noise level. The y-axis shows probability of EBFS removing each unwanted feature, calculated from 10-fold CV.

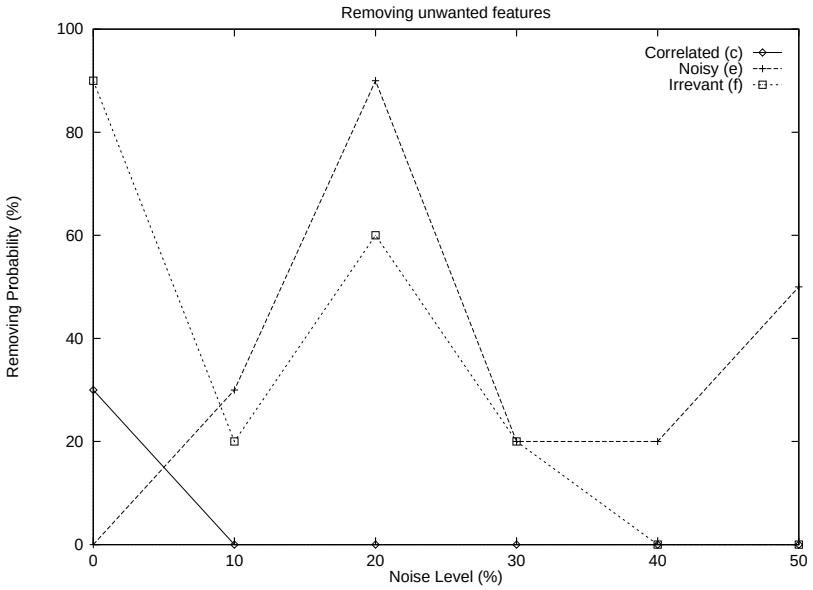

Figure 2.6: Probability of removing unwanted features

EBFS failed to remove the correlated feature, because C4.5 did not pick it in most cases. If C4.5 does not pick a feature, then there will be no failed path includes the feature, and EBFS will not regard it as bad. This can be said as shortcomings of EBFS. However, if an induction algorithm itself can avoid a bad feature, there are no need for any feature subset selection. At least from the viewpoint of accuracy, it does not hurt not to remove those features.

2.5 Related works

As the author mentioned in figure 2.1, there are a lot of studies in the area of feature subset selection. In the following sections the author reviews two existing studies on the feature subset selection problem. Then the author compares their performance with EBFS's. After that the author briefly discusses relationship between EBFS and explanation-based learning.

2.5.1 "Relief"

The Relief algorithm[35] assigns a weight to each feature, which is aimed to represent degree of relevance of the feature to the label. First Relief randomly picks an instance. The algorithm then searches a *near-hit* and a *near-miss*

of the picked instance, which means the nearest instance to the picked one whose label is the same and opposite respectively. Relief updates the weight of features according to the following motto: the more the difference in the feature between the picked one and near-miss, the more weight the feature gets; the more the difference in the feature between the picked one and near-hit, the less weight the feature gets. In this way Relief assigns high weights for features which have high power to discriminate an instance from its near-miss and to classify the instance and its near-hit to the same class. After enough iteration Relief returns a set of features which have weights larger than given threshold.

Because Relief's non-deterministic nature makes comparative study difficult, its deterministic version, ReliefD, which tries all the instances, is proposed[31]. The author uses ReliefD, instead of original Relief, in experiments shown later.

2.5.2 "Wrapper model"

John et al.[31] presented the *wrapper model*. According to them algorithms such as Relief are classified as the *filter model*, because the algorithm is used as a preprocess step to filter out irrelevant features independent of the learning algorithm. On the other hand, they used the learning algorithm itself to evaluate usefulness of candidate feature subsets.

Their basic algorithm is based on *backward stepwise elimination*, similar to one used in EBFS. In each step where n features are remaining, all feature subsets of size $n-1$ become a next candidate feature subset. In order to evaluate the candidate feature subset they used CV.

In their experiments they made a comparative study of the wrapper model, ReliefD and plain C4.5. As for accuracy to unseen data, the algorithm sometimes gave good results, but the effects of feature selection were generally insignificant. As for size of induced decision trees, the wrapper model gave the best results in reducing the size. However Relief sometimes yielded almost as good results as the wrapper model.

The wrapper model has an apparent shortcoming: it is computationally expensive. Using the backward algorithm the wrapper model with m-fold CV for an initial feature set with size n has to call the induction algorithm for $mn(n+1)/2$ times in the worst case. For large n, e.g., $n \geq 100$, it is prohibitive.

2.5.3 Experiments for comparison

In this subsection the author compares the three feature subset selection algorithm — ReliefD, the wrapper model and EBFS — each other and with C4.5. Two experiments were conducted. The first one employs 3-fold CV to evaluate the wrapper model and EBFS. Three UCI datasets along with **Vote100** and **Credit100** are used. 3-fold CV is chosen due to slowness of the wrapper model. The second one employs 10-fold CV to evaluate ReliefD and EBFS. Six UCI datasets along with **Vote100** and **Credit100** are used.

Again each algorithm is compared with C4.5, and both cases with/without pruning are studied. In addition to the algorithms shown in section 2.4, data produced by ReliefD with C4.5 without the pruning(**RD**), ReliefD with C4.5 with the pruning(**RDP**), the wrapper model with C4.5 without the pruning(**WM**) and the wrapper model with C4.5 with the pruning(**WMP**) are shown. Legends for them follow.

RD **WM**
RDP **WMP**

3-CV experiments

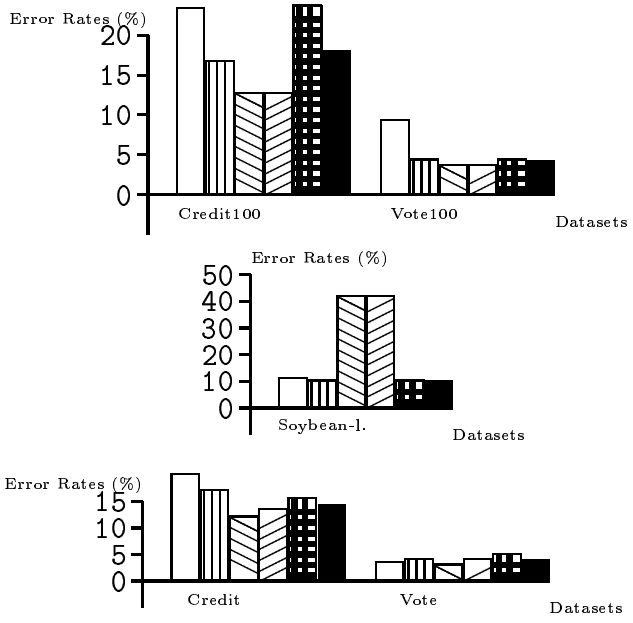

Figure 2.7: Error rates for 3-CV experiments

Error rates Figure 2.7 shows the error rates for the algorithms in the 3-CV experiment.

The wrapper model generally outperforms the other, with an exception of **Soybean-l.**. EBFS is almost as good as the wrapper model except in **Credit100**.

Size of the decision trees Figure 2.8 shows the number of nodes of the decision trees in the 3-CV experiment. The wrapper model is better than EBFS as a single method. Probably it was too aggressive to remove features for **Soybean-l.** and resulted poor accuracy.

However, combined with the pruning **EBFSP** is as good as, or sometimes outperforms, **WMP**. Regarding the speed of each algorithm, this is the huge advantage of EBFS over the wrapper model. As the author told in section 2.5.2, while the wrapper model with m-fold CV has to call the induction algorithm for $mn(n+1)/2$ times for a n-feature dataset in the worst case, EBFS needs only mn times. For example, the saving for **Vote100** whose $n = 116$ is more than 50 times.

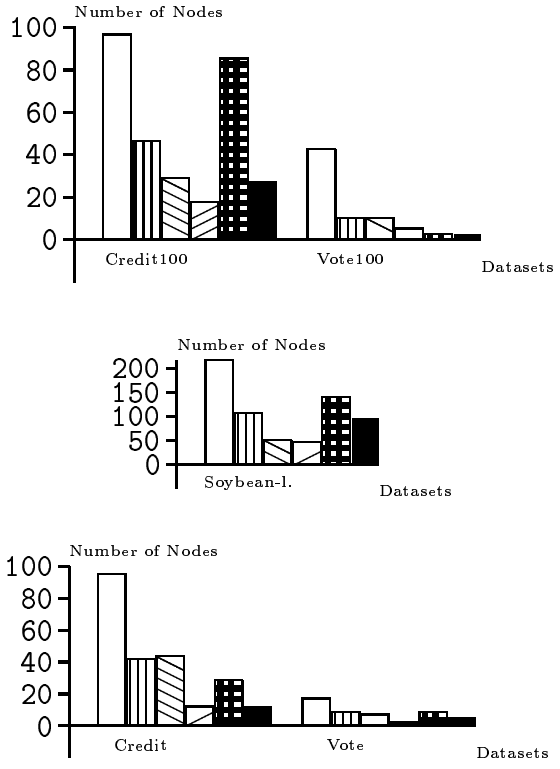

Figure 2.8: Number of nodes for 3-CV experiments

10-CV experiments

Error rates Figure 2.9 shows the error rates for the algorithms in the 10-CV experiment.

Here ReliefD and EBFS are performing similarly, with occasional bursts of ReliefD such as **Lenses** and **Tic-t.**. As the author showed earlier, EBFS also bursted for **Lenses**, but its degree is smaller than ReliefD's.

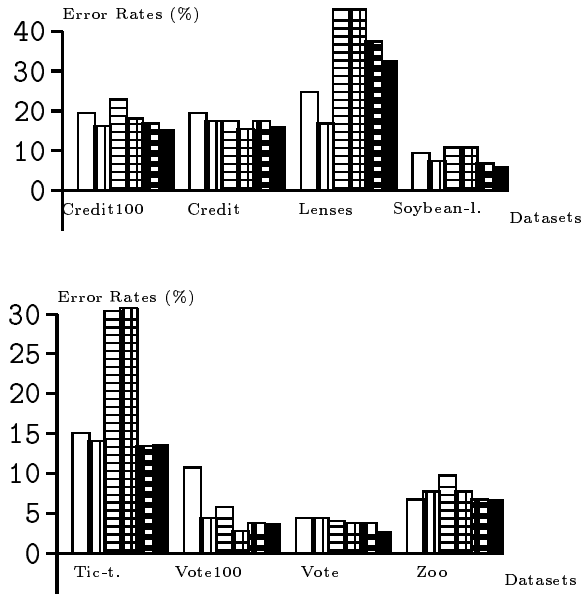

Figure 2.9: Error rates for 10-CV experiments

Size of decision trees Figure 2.10 shows the number of nodes of the decision trees in the 10-CV experiment. Now we can see that poor performance of ReliefD for **Tic-t.** was caused by too aggressive feature removal. As we saw in the 3-CV experiments, EBFS *alone* does not achieve a simple-enough tree. EBFS does milder removal and results a simple tree when the pruning is used.

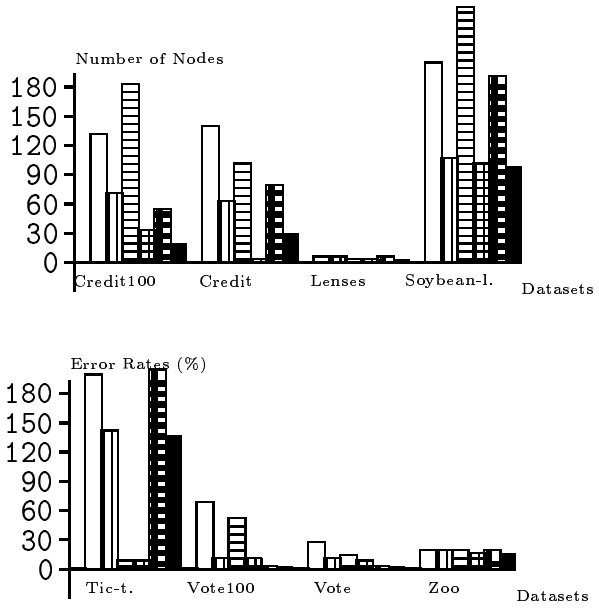

Figure 2.10: Number of nodes for 10-CV experiments

Size of the feature subset

In above experiments we saw that the wrapper model and ReliefD sometimes remove features very aggressively. Those behavior of feature subset selection algorithm can be directly seen by examining the size of the output of the feature subset selection algorithms, namely, the number of features given by EBFS/ReliefD/the wrapper model to C4.5.

Here three figures for each dataset are shown. Data corresponding to EBFS, ReliefD or the wrapper model show the size of output of each algorithm respectively. Data corresponding to C4.5 show the number of features in the original datasets. They are shown for reference.

3-CV experiments Figure 2.11 shows the size of the feature subset in the 3-CV experiment. Generally EBFS does relatively milder removal. Intuitively it removes harmful features only; useless but harmless features are left intact. An exception is **Vote100**, in which EBFS removed all but one features. Even in this extreme case, accuracy did not degrade (see figure 2.7).

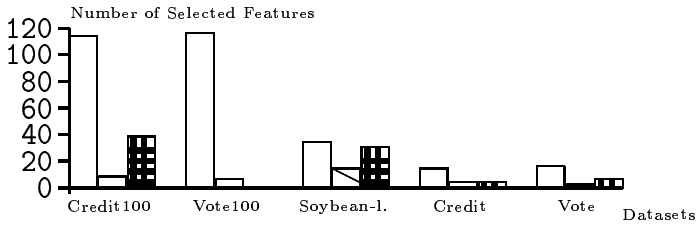

Figure 2.11: Number of selected features for 3-CV experiments

10-CV experiments Figure 2.12 shows the size of the feature subset in the 10-CV experiment. Again EBFS is less aggressive at removal than ReliefD, with exceptions of **Vote** and **Vote100**.

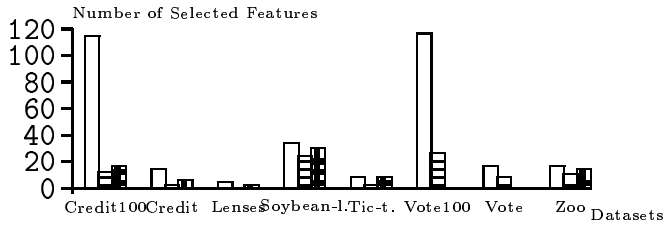

Figure 2.12: Number of selected features for 10-CV experiments

2.5.4 Explanation-based learning

Explanation-based learning (EBL) is a kind of speed-up learning. An EBL system is given a goal concept, a training example and domain knowledge. The system explains why the example is an instance of the goal concept using the knowledge. After the critical constraints in the explanation are determined, the explanation is generalized preserving the constraints. The result is a generalized recognition rule of the goal concept which may speed-up reasoning later.

If the domain knowledge is inconsistent, or perception limits are involved, it is possible to derive more than one plausible explanations for a training example. As Doyle[12] suggested, one way to cope with the problem is to conduct an experiment to gather an empirical justification which supports one but not the others.

EBFS's approach is similar in spirit to the EBL study. For a training set, more than one decision tree can be constructed. Testing the tree using a test set can be viewed as the experiment. When the experiment reveals that the tree classifies an instance incorrectly, we have a negative justification for the explanation. Each feature in the failed path is penalized so that the other

explanation will be offered for the instance next time, namely, the other tree, which less likely to use the penalized features, will be constructed.

2.6 Theoretical background

A study of feature subset selection is strongly related to definition of irrelevancy of feature. The definition specifies what kind of features should be removed by a feature subset selection algorithm.

In this chapter the author discusses theoretical background of EBFS, namely, the target irrelevancy of EBFS. First, there are several properties on a feature which are helpful to discuss the feature subset selection problem.

2.6.1 Properties of feature

Definition 2.6.1 *A feature is called* strongly relevant *iff it cannot be removed without loss of prediction accuracy. A feature is called* weakly relevant *iff it is not strongly relevant and it can sometimes contribute to prediction accuracy. A feature is* relevant *if it is either strongly or weakly relevant, and* irrelevant *otherwise[31]. Note that these concepts are defined without having a learning algorithm as a parameter.*[3]

Definition 2.6.2 $X_i \in X - W$ *is called* useful *for* $\mathcal{M}$ *using* W *if* $gl(S(\mathcal{M}, T^{W,X}), U^{W,X} - T^{W,X}) < gl(S(\mathcal{M}, T^X), U^X - T^X)$ *stands* on average over typical training sets $T^{W,X}$, *where* U^A *denotes a set of all instances whose labels are known.*

By using these properties, we can say that Relief approximates the relevant features and the wrapper model approximates useful features in a direct manner.

2.6.2 Why the wrapper model works

John et al. argued that the wrapper model works better than the filter models such as FOCUS[1] or Relief because the former selects the feature taking into account the biases of the induction algorithm which is completely ignored by the latter[31]. A similar argument is found in [7]. However this fails to explain the unexpected good results by Relief in [31].

The author claims that the wrapper model works better, not only because the model respects the biases of the induction algorithm, but also because

[3]More formal definitions are seen in [31].

the model partly solves an overfitting problem, which is not aimed to solve in the filter models.

The viewpoint shown above also explains Relief's good results in [31]. Relief gives the higher score to a feature that has the less difference of values between the picked instance and its near-hit. In a sense Relief examines the feature's predicting ability using the instance and its near-hit as a training set. Relief penalizes the feature if it has little difference of values between the picked instance and its near-miss. Here the near-miss is acting as a test set, ensuring that the feature useful to predict the label only for a certain training set is not rewarded too much. By iterating the process, Relief is performing *implicit* CV-like activity. However it largely differs from CV, because in Relief an instance can be chosen as the near-miss (i.e., test set) more than once, even if it is guaranteed that each instance is chosen as the *picked* instance only once.

This view of the advantage of the wrapper model directly leads the author to another property of feature: *L-usefulness*.

2.6.3 L-useful feature subset

General definition

A L-useful feature subset characterize a feature subset that is useful for a particular learning algorithm when it is given a particular training set and it is tested by a particular test set.

Definition 2.6.3 *Let W_i, X be a subset of A and $W_i = X - \{X_i\}$. Let T^X_{train} to be a proper subset of T^X and give it to the learning algorithm $\mathcal{M}$ to obtain $S(\mathcal{M}, T^X_{train})$. Then create $T^{W_i,X}_{train}$ according to T^X_{train}, give it to $\mathcal{M}$ to obtain $S(\mathcal{M}, T^{W_i,X}_{train})$.*

If $gl(S(\mathcal{M}, T^{W_i,X}_{train}), T^{W_i,X} - T^{W_i,X}_{train}) < gl(S(\mathcal{M}, T^X_{train}), T^X - T^X_{train})$ then the feature X_i is called L-useful *for $\mathcal{M}$ using W_i with respect to T^X_{train}.*

If $gl(S(\mathcal{M}, T^X_{train}), T^X - T^X_{train}) < gl(S(\mathcal{M}, T^{W_i,X}_{train}), T^{W_i,X} - T^{W_i,X}_{train})$ then the feature X_i is called L-harmful *for $\mathcal{M}$ using W_i with respect to T^X_{train}.*

For $\mathcal{M}, X, W_i$ and T^X_{train}, the feature X_i is called L-useless *if it is neither L-useful nor L-harmful.*

$gl(S(\mathcal{M}, T^X_{train}), T^X - T^X_{train}) - gl(S(\mathcal{M}, T^{W_i,X}_{train}), T^{W_i,X} - T^{W_i,X}_{train})$ is called X_i's L-usefulness *for $\mathcal{M}$ over W_i with respect to T^X_{train} and denoted by $LUF(X_i, W_i, \mathcal{M}, T^X_{train})$.*

A feature subset $X \subseteq A$ is called L-useful for $\mathcal{M}$ with respect to T^X_{train} if all $X_i \in X$ is L-useful for $\mathcal{M}$ using $X - \{X_i\}$ with respect to T^X_{train}.

L-usefulness of feature subset X is denoted $LUFS(X, \mathcal{M}, T^X_{train})$ and defined as $LUFS(X, \mathcal{M}, T^X_{train}) \stackrel{def}{=} \sum_{X_i \in X} LUF(X_i, X - \{X_i\}, \mathcal{M}, T^X_{train})$.

L-usefulness is a *localized* version of usefulness. Now the author can define his version of usefulness, called *U-usefulness*.

Definition 2.6.4 *Let proposition* $C(T_i^X, \mathcal{M})$ *denote that a training set* T_i^X *satisfies constraints that are required by* $\mathcal{M}$ *in order to create a reasonable structure.*[4] U-usefulness *of feature subset* X *for* $\mathcal{M}$ *with respect to* T^X *is denoted by* $UUF(X, \mathcal{M}, T^X)$ *and defined as* $UUF(X, \mathcal{M}, T^X) \stackrel{def}{=} \sum_{D(T_i^X)} LUFS(X, \mathcal{M}, T^X - T_i^X)$, *where* $D(T_i^X) \Leftrightarrow (T_i^X \subset T^X \wedge C(T^X - T_i^X, \mathcal{M}))$ *and* $\forall i, j (i \neq j \supset T_i^X \cap T_j^X = \phi)$.

The definition says that UUF is the sum of $LUFS$ over T^X's proper subset which satisfies C.

Obviously m-fold CV for large m can be used to estimate $UUF(X, \mathcal{M}, T^X)$. Such estimation of $UUF(X, \mathcal{M}, T^X)$ is done by the wrapper model. The wrapper model returns X with optimal estimated $UUF(X, \mathcal{M}, T^X)$.

2.6.4 L-usefulness and the heuristic function

The author has explained why the wrapper model is superior than the filter models and characterized a feature subset returned by it. Let us consider how to reduce the wrapper model's computationally expensiveness.

Suppose we are using the backward algorithm. When $|X| = k$ features remain as a candidate subset, the wrapper model calls the induction algorithm km times, because there are k possible next candidates and each candidate requires m runs of the induction algorithm to evaluate its U-usefulness by m-fold CV. If we can estimate $X_i \in X$'s L-usefulness just by constructing $S(\mathcal{M}, T_{train}^X)$, we can reduce km to m, which yields a drastic improvement from $mn(n+1)/2$ to mn in the worst case. The author made it possible by using the heuristic H. In EBFS, H is used to estimate relative L-usefulness of each remaining feature.

The following corollary serves as a base of the heuristic.

Let $PL(S, I)$ denote a label that is predicted by a structure S for an instance or a training instance I. $EP(S, I) = (N_0, \ldots, N_L)$ denotes an *explanation path* of the decision tree S for I.

Corollary 2.6.5 *Suppose* $X \subseteq A$ *is a set of feature. Let* X_i *be a L-harmful feature for a decision tree induction algorithm* $\mathcal{M}$ *using* $X - \{X_i\}$ *with respect*

[4] A typical example of the constraints is that the training set provides enough number of instances.

to T^X_{train}. *Let* $t \in T^X - T^X_{train}$ *be an instance in the test set whose label is* y. *Then there exists* t *such that*

$$PL(S(\mathcal{M}, T^X_{train}), t) \neq y$$
$$\wedge \quad PL(S(\mathcal{M}, T^{X-\{X_i\},X}_{train}), t) = y$$

and $EP(S(\mathcal{M}, T^X_{train}), t)$ *must include a node where* X_i *is tested.*

Proof Existence of t is trivial from the definition of L-harmful feature. For such t obviously $EP(S(\mathcal{M}, T^X_{train}), t) \neq EP(S(\mathcal{M}, T^{X-\{X_i\},X}_{train}), t)$. Assume that $EP(S(\mathcal{M}, T^X_{train}), t)$ does not include a node where X_i is tested. The first element of $EP(S(\mathcal{M}, T^X_{train}), t)$ does not test X_i because of the assumption. Then it must be equal to the first element of $EP(S(\mathcal{M}, T^{X-\{X_i\},X}_{train}), t)$ because at the root node the ID3-like algorithm must have selected the same feature as for $T^{X-\{X_i\},X}_{train}$ if it was not X_i. We can recursively repeat the same argument as we descends along $EP(S(\mathcal{M}, T^X_{train}), t)$ to show $EP(S(\mathcal{M}, T^X_{train}), t) = EP(S(\mathcal{M}, T^{X-\{X_i\},X}_{train}), t)$. This leads contradiction. ■

From the corollary we can say that a L-harmful feature is expected to appear more in failed paths than in successful paths, under some normalization on frequencies of appearance of features. This is what H is doing.

2.7 Discussions and concluding remarks

The author presented a heuristic H, which effectively estimates feature's expected irrelevance on a given training set, yet it is efficiently computed. Using H as an evaluation function for features the author constructed a greedy algorithm based on backward stepwise elimination. The algorithm, called EBFS, was tested on many datasets and proven to be effective. The comparative study revealed that EBFS is as much effective as the wrapper model on the problem and much more efficient than the wrapper model. Although EBFS is relatively less aggressive in removing features, its ability to result a simple decision tree is excellent when combined with the pruning.

A key concept behind H is L-usefulness, a new concept that characterize feature's relevancy to a given learning problem. The point is that the feature subset selection can contribute not only to simplify an induced structure but also to avoid overfitting. This view explained the fact that the wrapper model and Relief do a decent job on feature subset selection problems.

Future research should include investigation of search methods. As John et al.[31] pointed, the wrapper model consists of feature subset search and its evaluation, where the evaluation is done using an induction algorithm.

Other than the backward algorithm which has been focused in this chapter, they mentioned about the forward algorithm based on the forward stepwise selection and combination of the forward and the backward based on bidirectional search. Their experiments revealed that the forward algorithm often gives a better result than the backward algorithm. Because computing H is computationally much cheaper than the evaluation in the wrapper model, we should afford to employ complex search methods like best-first search or genetic algorithm[72] with EBFS. Combining EBFS with other filter method such as Relief should be also investigated. As we saw in section 2.4.2, EBFS can be unstable when little failed paths are available. In such case, using Relief instead of EBFS can be helpful.