A Dynamic Approach to Reducing Dialog in On-Line Decision Guides

Michelle Doyle & Pádraig Cunningham

Department of Computer Science, Trinity College Dublin, Dublin 2, Ireland.
{Michelle.Doyle,Padraig.Cunningham}@cs.tcd.ie

Abstract. Online decision guides typically ask too many questions of the user, as they make no attempt to focus the questions. We describe some approaches to minimising the questions asked of a user in an online query situation. Questions are asked in an order that reflects their ability to narrow down the set of cases. Thus time to reach an answer is decreased. This has the dual benefit of taking some of the monotony out of online queries, and also of decreasing the amount of network request-response cycles. Most importantly, question order is decided at run time, and therefore adapts to the user. This approach is in the spirit of lazy learning with induction delayed to run-time, allowing adaptation to the emerging details of the situation. We evaluate a few different approaches to the question selection task, and compare the best approach (one based on ideas from retrieval in CBR) to a commercial online decision guide.

1 Introduction

Intelligent assistants on web sites, particularly e-commerce sites, are becoming increasingly common. Often these decision guides consist of a daunting list of questions that need to be answered in order to find the most suitable item[1]. Long-lived dialogs lead to monotony for the user, and do nothing to encourage user acceptance of such sites. Therefore an intelligent assistant which asks the *minimum* of questions in a dialog situation is desirable. This has the added benefit of minimising the number of network request-response cycles. A reduction in dialog length can be achieved through the production of *focused* questions.

Users often consult these kinds of systems with only a vague idea of their needs. Therefore a conventional one-shot retrieval strategy may not be ideal. Instead, a *multi-stage* retrieval process can be employed, in which the initial information is used to retrieve the first candidate set, and subsequent questions whittle down this subset until a manageable set of cases remains. At each step in the interaction with the user, we wish to ask them a question that will maximally discriminate between the candidate cases at that point. Thus the answers given by the user at each step influence the order of the questions. The incremental CBR technique (I-CBR) [3][12] employs such a retrieval strategy, and uses a simple information theoretic metric to find the feature that best discriminates between the current set of retrieved cases.

[1] See personalogic (http://www.personalogic.com) for example

However decision guides mostly operate on data without class labels, such as product descriptions. The information theoretic measure used in incremental CBR *requires* class labels in order to decide the discrimination power of a particular feature. Therefore we must either find another metric to use with this multi-stage retrieval technique, or apply a clustering algorithm to our unlabelled data in a pre-processing step, allowing us to use the information theoretic measure.

Associated with this problem of *single* feature selection is that of feature *subset* selection. Feature subset selection techniques attempt to reduce the number of attributes used to describe an input in order to improve predictive accuracy. This is an important area of research in machine learning, as irrelevant attributes in the training data can lead to a decrease in the effectiveness of learning algorithms. A variety of feature subset selection techniques for supervised learning exist (see [1] for a discussion of sequential feature selection algorithms and further references). However, the absence of class labels makes unsupervised feature subset selection a more difficult problem, and there are fewer techniques to choose from. We investigated some of these techniques, and chose one whose evaluation function could be applied to our particular task. This function's performance was then compared to that of I-CBR's metric after clustering the same data.

In the following sections we discuss the issues further, describe our approaches to the problem, and present some results. In section 2 we discuss the I-CBR technique and the clustering algorithm we used, while our attempts to find a metric which works on unlabelled data are discussed in section 3. Results from the application of the metrics to a variety of case-bases are presented in section 4, where we also compare our approach (in terms of number of questions asked) to an existing online decision guide. We discuss our findings and possible uses of the technique in section 5.

2 Incremental Case-Based Reasoning and Clustering

2.1 Incremental Case-Based Reasoning

Incremental Case-Based Reasoning (I-CBR) is an incremental case-retrieval technique based on information-theoretic analysis [3][12]. The technique is *incremental* in the sense that it does not require the entire target case description to be available at the start, but in fact builds it up by asking focused questions of the user. The ordering of these questions reflects their power to discriminate effectively between the set of candidate cases at each step.

As described in [3][12], the technique was originally developed for use in the domain of fault diagnosis, where it is difficult to gather a complete case description in advance. In these types of problems, there is potentially a large amount of information that could be used to aid the diagnosis, but not all of this information is necessarily needed to solve the problem. Moreover, some of this information is "expensive" - meaning it may be difficult or costly to acquire. Therefore the motivation was to design an incremental CBR system that would request expensive data only as needed, when it had been determined that it would contribute to the

diagnosis. A multi-stage retrieval process achieves this. The "free" (or readily available) features can be input to the initial (under-specified) query. This first pass retrieves a subset of the initial case-base, and subsequent refining questions reduce the size of the retrieved set even further, until a consistent set of cases remains. Evidently, this approach need not solely be used in domains with free and expensive features. In the case of a decision guide, it would be advantageous to minimise the questions asked, and this approach can be used to do just that. As questions are asked in order of their discrimination power, the set of retrieved cases is whittled down at a maximal rate, consequently only a subset of the features should be needed to reach an answer.

The information theoretic measure used to assess the discrimination power of a feature is similar to that used by ID3 [11] when deciding the feature ordering in a decision tree. It is based on the concept of *entropy* (or *information content*). One interpretation of entropy is that it specifies the *minimum number of bits of information* needed to encode the classification of an arbitrary case in the data set. This can be measured using (1).

$$I\left(|D_1|,\ldots,|D_d|\right) = -\sum_{i=1}^{d}\left(\frac{|D_i|}{|D_1|+\ldots+|D_d|}\bullet \log_2\left[\frac{|D_i|}{|D_1|+\ldots+|D_d|}\right]\right) \qquad \textbf{(1)}$$

(where d is the number of possible classes and $|D_i|$ is the number of cases of class D_i).

If all the cases are of the same class, then the number of bits of information needed to predict the class is 0. If there are multiple possible classes, a value greater than 0 will be produced, meaning *some* information will be needed to determine the class of any of the cases.

The actual measure used to discover the most discriminating feature is the expected *reduction* in entropy caused by knowing the value of a feature *f*. After partitioning the data set using *f*, we can calculate the entropy of each of these partitions. Each of these values is multiplied by the fraction of cases in that particular partition and these are summed to get the expected entropy after partitioning the cases using *f*.

$$E(f) = \sum_{i=1}^{n}\left(\frac{|D_1^i|+\ldots+|D_d^i|}{|D_1|+\ldots+|D_d|}\right)\bullet I\left(|D_1^i|,\ldots,|D_d^i|\right) \qquad \textbf{(2)}$$

(where n is the number of possible values for f, $|D^i_j|$ is the number of cases of class D_j with value i for f (i.e. the number of cases of class D_j in the i^{th} partition), and $I(|D^i_1|,\ldots,|D^i_d|)$ is the entropy of the i^{th} partition).

The reduction in entropy (or the *information gain*) is the difference between the entropy before the partition, and that after (see **3**). The lower the entropy (or information needed) after partitioning, the higher the gain. If, for example, each partition contains only cases of a particular class, then *f* is a highly predictive of the class of a case. This would be reflected in an entropy score of 0 for each of the partitions and a resulting maximal gain.

$$\textit{Information gain}(f) = I(|D_1|,...,|D_d|) - E(f) \quad \textbf{(3)}$$

After using this measure to decide the most discriminating feature for the current set of retrieved cases, the user is prompted for a value for this feature, the retrieved set is further reduced, and the most discriminating feature in relation to the new retrieved set is calculated. This continues until a manageable set of cases remains. In this way, the I-CBR approach differs from ID3, as no global decision tree is built; instead local, partial decision trees are built on the fly over a reduced set of cases.

Our implementation of I-CBR differs from the system described in [12]. We don't actually construct a local decision tree for retrieval; instead we use a spreading activation algorithm. The reason for this was to avoid the exact matching enforced by decision trees, and make use of the "fuzzy" matching allowed by this approach.

At the first pass stage of retrieval a score is given to each case depending on its similarity to the target along the specified features. This score is produced through a "spreading activation" process implemented using indexing structures. All those cases within a certain threshold of similarity are placed in the retrieved set. Fig. 1 shows this at work in a simple domain with only two features. The shaded area in Fig. 1(a) shows the retrieved set after the user provides the indicated value for feature f1. As each refining question is asked, the activation of each case in the retrieved set is updated according to their similarity to the evolving target description. The threshold of similarity is also updated, according to the total number of features entered. Those that are not within this threshold of similarity are excluded from the refined set of retrieved cases. Fig. 1(b) shows the refined set after determining the required value for feature f2. This process continues until the retrieved set is small enough. In the example shown, only feature f1 would have been asked if the goal were to retrieve 10 cases or less. Moreover, our system should also realise that f1 is more discriminating than f2, and therefore ask that feature first.

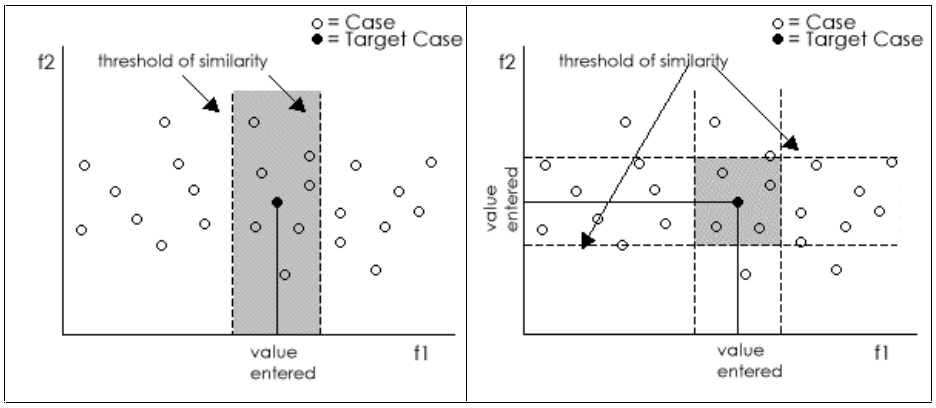

Fig. 1. (a) Shaded area is set of retrieved cases after user enters value for f1. (b) Shaded area is set of retrieved cases after user enters value for f2.

We also implemented extensions to the basic algorithm for calculating information gain to deal with continuous features and unknown values in the data (see [11]).

As we will show later, using the information gain metric is an extremely good way of reducing the number of features needed to reach a consistent set of retrieved cases. However, it will not work for cases without inherent "classes" or "outcomes", as this information is needed to decide the discrimination power of a given feature. Therefore we experimented with pre-clustering the unlabelled data and then using the approach described above. The clustering algorithm used in these experiments is described below.

2.2 Clustering - the *k*-Medoid Method

The clustering algorithm chosen was Kaufman and Rousseeuw's *k*-medoid algorithm [10]. Their approach was to attempt to find *k representative objects* (medoids) and to cluster the other objects according to their closeness to these. This has similarities to the well-known *k*-means algorithm with the important difference that the goal is to minimise the *average* dissimilarity of the representative object to all the other objects of the same cluster, whereas the *k*-means approach attempts to minimise the *average squared* difference. Therefore the *k*-medoid approach has the advantage of being less sensitive to outliers.

The optimal choice of *k* is a non-trivial problem and is crucial as a poor value can lead to unnatural clustering. Kaufman and Rousseeuw describe a method of graphically representing the cluster structure (a so-called *silhouette plot*), and suggest how a validity index, which this method computes, can be used to determine an appropriate choice of *k*. Using this approach *k* can be found through exhaustive search. We can perform the clustering algorithm (shown in Fig. 2) for different values of *k* from 2 to *M* (where *M* is less than the number of objects to be clustered), and calculate the validity index $\boldsymbol{S}(k)$ for each value of *k*. The value of *k* for which $\boldsymbol{S}(k)$ is highest is the best choice for *k*. This automatic determination of *k* means we can use the algorithm on data about whose structure we have no prior knowledge.

The method of calculating $\boldsymbol{S}(k)$ is straightforward. First, we must define the term $s(i)$ for an object *i* in the data set.

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \qquad (\mathbf{4})$$

$a(i)$ refers to the average dissimilarity of *i* to all other objects in its own cluster. Similarly, a measure $c(i)$ is defined which refers to the average dissimilarity of *i* to all the objects in any other cluster *c*. After calculating $c(i)$ for all clusters *c*, $b(i)$ is found as the minimum value of $c(i)$, and the cluster *b* is the corresponding cluster. In other words, *b* is the closest neighbour of object *i*, and $b(i)$ is the average dissimilarity of *i* to all the objects in its neighbour cluster. If the above formula yields a high value (close to 1), then the "within" dissimilarity $a(i)$ is much smaller than the "between" dissimilarity $b(i)$, and therefore *i* is well clustered. If however it is close to 0, then *i* lies between the two clusters. If $s(i)$ turns out to be negative, then *i* is on average closer to the objects in *b* than in *a*, and perhaps has been misclassified, although a very small negative value suggests an object lying at the periphery of both clusters. We can now define $\boldsymbol{S}(k)$ as the average of $s(i)$ over the entire data set. Therefore a high value of $\boldsymbol{S}(k)$ indicates "tight" clusters. The value of *k* for which $\boldsymbol{S}(k)$ is maximum thus produces the best cluster structure.

The first stage of the clustering algorithm (called BUILD), for a particular value of *k*, is shown in Fig. 2. This algorithm chooses the *k* medoids. Kaufman and Rousseeuw describe a second stage (called SWAP), that attempts to improve on the set of medoids chosen. However, this stage is quite computationally intensive, and in our experiments has not yielded better clusters. Since the medoids found in the first stage are independent of the order in which the objects are presented to the clustering system (which is a problem for some other clustering algorithms), we decided to stick with the medoids found at that stage. However, we did investigate a method of maximising the tightness of the clusters by examining the value of *s(i)* for each object *i* after clustering, moving objects with negative values into their neighbour clusters, and recalculating all *s(i)* after each cluster is changed, until no more negative values of *s(i)* exist. However, our experiments on data sets with known clusters have yielded similar or even slightly better results using the first stage of the clustering algorithm alone, despite the higher ***S(k)*** values resulting from our second stage.

```
Choose the most centrally located object as the
first medoid (as it has the smallest possible
sum of dissimilarities to all other objects).

 for 2 to k   // choose other k-1 medoids
     for i = 1 to N// N is number of objects
         if (notYetSelectedAsMedoid(i))
              for j = 1 to N
                   if (notYetSelectedAsMedoid(j))
                        M = closestMedoid(j)
                        Dij = dist(i,j)
                        Dj = dist(M,j)
                        Cij = max(Dj-Dij, 0)
                        gain(i) = gain(i) + Cij
                   endif
              endfor
              if (gain(i) > bestGain)
                   bestGain = gain(i)
                   bestObject = i
              endif
         endif
     endfor
     add bestObject to list of medoids
 endfor
```

Fig. 2. The first stage of the *k*-medoid algorithm

We have tested this clustering algorithm on a few data sets with known classes, and have had good results. For example, with the iris and thyroid[2] data sets, the accuracy scores of the clustering were 91% and 92% respectively.

[2] from the UCI ML Repository - http://www.ics.uci.edu/~mlearn/MLRepository.html

3 Unsupervised Feature Subset Selection Methods

Feature subset selection can be viewed as a search through the space of possible descriptions of the training data [7]. The goal, in supervised learning, is to improve predictive accuracy through the elimination of redundant or misleading features. Similarly, in unsupervised learning the goal is to improve the efficiency of clustering and the quality of the clusters produced (irrelevant features can obscure cluster structure). There has been much research in the area of feature subset selection for supervised learning tasks, but the absence of class labels in unsupervised learning makes it a more difficult problem. However there has been some work in the area in recent years. We investigated two recent approaches and a feature extraction technique from statistics. We compared them all in terms of the ease of using their evaluation functions with our incremental retrieval system.

3.1 Feature Selection in Conceptual Clustering

The first unsupervised feature subset selection algorithm we investigated was Devaney and Ram's [7]. This was implemented in the area of conceptual clustering, using Fisher's [9] COBWEB system as their underlying concept learner. They use category utility as their evaluation function, which is already used by COBWEB to guide the process of creating concepts. This function measures the increase in the number of feature values that can be predicted correctly given a set of concepts, over those which can be predicted without using any concepts.

Their approach was to generate a set of feature subsets (using either Forward Sequential Selection (FSS) or Backward Sequential Selection (BSS), run COBWEB on each subset, and then evaluate each resulting concept hierarchy using the category utility metric on the first partition[3]. BSS starts with the full feature set and removes the least useful feature at each stage until utility stops improving. FSS starts with an empty feature set and adds the feature providing the greatest improvement in utility at each stage. At each stage the algorithm checks how many feature values can be predicted correctly by the partition – i.e. if the value of each feature *f* can be predicted for most of the classes *C(k)* in the partition, then the features used to produce this partition were informative or relevant. The highest scoring feature subset, *B*, is retained, and the next larger (or smaller) subset is generated using *B* as a starting point. The process continues until no higher category utility score can be achieved.

Since we are interested in feature ranking, not subset selection, this approach cannot be directly applied to our task. We need to be able to rate the importance of a *single feature* rather than a subset. However running COBWEB on just one feature would simply result in a biased concept hierarchy, and the category utility metric wouldn't tell us much in that situation. A better approach would be to first run the feature subset selection algorithm offline to get the best subset (*B*). Then, for each feature *f*, we could calculate the category utility of the concept hierarchy generated using the feature set $B / \{f\}$. It seems plausible that the feature whose deletion causes

[3] Each *set of siblings* in the hierarchy is referred to as a *partition*. The first partition is the children of the root.

the greatest drop in category utility is the most important. This approach would leave us with an ordering of the features, which is exactly what we need.

However this theory has not been tested, as the evaluation function is computationally intensive; at each step, the concept structure must be reconstructed from scratch using the new feature set $B / \{f\}$, and the category utility metric computed. Devaney and Ram improved upon the time it takes to reconstruct a concept structure by using their own concept learner, AICC, instead of COBWEB. AICC can add features without having to rebuild the concept hierarchy from scratch, and shows large speedups. However even using AICC, this approach would require too much computation to be suitable for choosing the most predictive feature on the fly.

3.2 Principal Components Analysis and Variance

Principal Components Analysis (PCA) is a well-established statistical technique, which has been applied to many different tasks. Since it considers only the features and not the classes to which they belong, it can be used for unsupervised feature selection. PCA determines the linear transformation of a sample of points in *n*-dimensional space that exhibits the properties of the sample most clearly along the co-ordinate axes. These new axes are linear combinations of the original axes, found by calculating the eigenvectors of the original data's covariance matrix. The variance along each of these new axes is given by the corresponding eigenvalue in the covariance matrix. Along the new axes the sample variances are extremes, and those which show little or no spread (minimum variance) indicate interdependence in the data. Therefore setting a minimum bound on the amount of variance along each axis is a way of reducing dimensionality [2].

This method thus examines the variance along the new axes to determine those that are unimportant and therefore can be removed. Therefore we decided to determine if variance is an indicator of the relative importance of features. Using PCA, we could ascertain the new axis with the highest variance directly. However we are interested in determining the most important feature in the *original* feature space. Since the new axis with the highest variance is a linear combination of the original feature set, we simply need the feature that dominates this linear combination.

However, simply determining the variance of each of the features in the original data set would have much lower computational cost and would probably be just as valid. Therefore we investigated using a feature's variance in the current set of cases as its evaluation function. Intuitively, it makes sense that a feature with high variance will find fewer possible matches than one with low variance, and therefore would speed up the retrieval process (see Fig. 1). Although this approach lacks a strong theoretical justification, it is worth comparing to the more "rigorous" methods. We show some results of this comparison in section 4.

3.3 Entropy-based Feature Ranking

This method proposed by Dash and Liu [5,6] seems best suited to the query selection task. It uses an entropy measure to rank the features in order of importance - allowing us to access the most important feature directly. In addition, this entropy measure does not need class information to evaluate the features, unlike the information gain measure.

This approach uses a different definition of entropy that recognises that entropy is low for orderly configurations (where there are distinct clusters), but higher for disorderly or "chaotic" configurations. This is due to the fact that there are few orderly configurations compared to the number of disorderly ones – making the probability of a disorderly configuration so much higher. In other words, it is less likely that instances in the space of observations will be very close or very far (forming distinct clusters), and more likely that instances would be separated by the mean of all distances in the space. Therefore the entropy measure used gives a low score (close to 0.0) to very close or very distant pairs of instances, and a high score (close to 1.0) to those instances separated by a distance close to the mean distance. The total entropy is calculated over *all pairs* of cases. The formula (5) can be seen below, where S_{ij} stands for the similarity of cases *i* and *j*.

$$E = -\sum_{i=1}^{N}\sum_{j=1}^{N}\left(S_{ij} \times \log S_{ij} + \left(1 - S_{ij}\right) \times \log\left(1 - S_{ij}\right)\right) \quad \textbf{(5)}$$

The algorithm is straightforward, at each step, the entropy is calculated after removing each possible feature from the feature set. The feature that produces the greatest drop in entropy is selected, as removing it made the clusters more distinct (indicated by lower entropy). The converse is true for the most important feature. Using this knowledge, the features can be ranked from the one whose removal resulted in the highest entropy, to that which produced the lowest entropy figure. A further step to their algorithm uses this information to choose the optimal subset of the original features. The method has been shown to be successful in detecting irrelevant features, but cannot detect redundancy in features [6].

This method is directly applicable to the query selection task. Since the initial stage of the algorithm results in an *ordered* list of the features, we can simply implement this part of the algorithm and then query the user with the first feature in the list. Therefore this is the algorithm we chose to implement for our experiments. However, the computational cost of this evaluation function is fairly high. When testing each feature, we need to re-calculate the entropy after removing that feature from the feature set. This involves calculating the sum of the similarities of *every pair* of cases in the case-base, along all dimensions but the removed one. This method is too computationally expensive to generate refining questions at run time. In our implementation, we improved efficiency by calculating the *total* similarity (along all dimensions) of every pair of cases in the case-base in an initial start-up step. Then each time we wished to calculate the entropy of a particular feature, we only needed to calculate the similarity of each pair of cases along that particular dimension, and delete this value from the total similarity score for that pair of cases.

This modification did improve the efficiency of the measure, however the information gain metric is still clearly more efficient. More important, was discovering which method was most effective at reducing the number of query questions. Therefore we needed to compare the three metrics experimentally. The results of these experiments are discussed in the following section.

4 Experimental Comparisons

4.1 Initial Experiments on Data with Known Classes

We first compared the three metrics using data sets that actually contained class information. Our experiments followed the leave-one-out strategy. This strategy removes each case in turn from the case-base and uses it as a target case. Thus the incremental retriever was run *N* times for each data set (where *N* is the number of cases). This process was repeated for each of the three feature selection techniques; i.e., feature variance, Dash-Liu, Info-Gain + Clustering and a fourth time using random feature selection. The objective was to find the technique that asked the least questions.

The results for each data set and each technique were averaged over the *N* runs. In this first set of experiments, refining questions were generated until

1. All cases retrieved have the same classification - *or*
2. There are no more *discriminating* features left unspecified (i.e. all cases retrieved have the same values for the remaining features) - *or*
3. All features have been asked

The graph in Fig. 3 shows the results of experiments on five different data sets. The data sets are all taken from the UCI Machine Learning Repository. In each experiment with a particular data set, the average number of features asked was calculated, and this was then converted to a percentage of the total features in that data set. Since the information gain metric is the only one with access to class information, the comparison might seem unfair. Therefore we took the data sets, ran the clustering algorithm on them, and performed the information gain experiments using these clusters rather than the "correct" class information.

As can be seen in Fig. 3, the information gain metric, even in conjunction with clustering, outperforms the other metrics in terms of the number of features needed to perform retrieval, particularly with the soybean data set. It is also interesting that the simple approach of asking features in order of variance characteristics is at least as good, and sometimes better, than the (theoretically more sound) DashLiu metric. Therefore this would seem a good choice of measure for use with unlabelled data as it is the simplest metric and has the lowest computational cost. However we can only calculate the variance of interval-scaled and ordinal (ordered symbolic) data, and many data sets contain categorical/nominal data, therefore it is restricted in its

applications. In the Conclusions we discuss the possibility of using entropy as a measure of diversity for category variables that might overcome this problem.

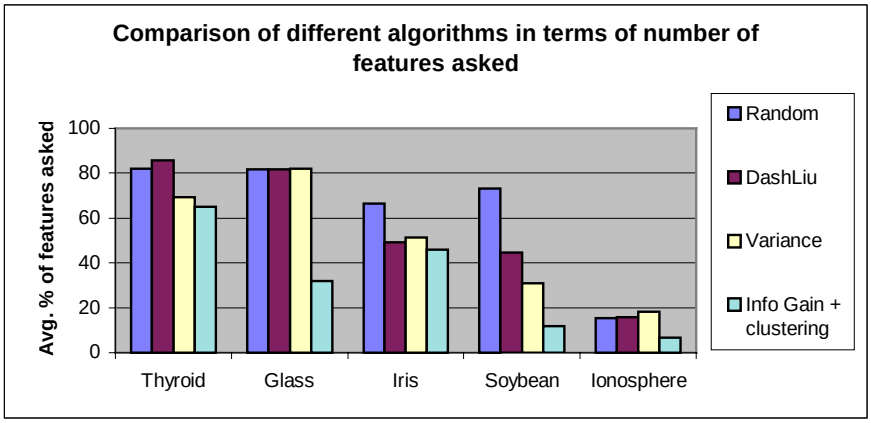

Fig. 3. Comparison of different metrics in terms of number of features asked

The reason for the poor performance of the DashLiu metric, particularly with the soybean and thyroid data sets, is not clear. Perhaps this can be explained by the fact that this measure has been shown to be unable to detect redundancy [6], but there is no definite evidence of redundant features in these data sets, so we cannot be sure. However, the particularly bad performance of random selection on the soybean data set would seem to suggest the existence of a lot of irrelevant - or perhaps redundant - features in that data set.

In general, all the metrics (except the DashLiu measure in one instance, and the variance in another) produce better results than asking the features in random order, which shows they all make a contribution to minimising inputs needed for retrieval. However it is clear that the information gain metric would be the best choice to assist an online decision guide, as it requires the fewest input features to reach an manageable set of cases.

4.2 Experiments with Unlabelled Data

The unsupervised data we experimented with was a case-base of laptop computer descriptions comprising over 600 cases. This data was a perfect example of the kind of data that would be used with an online decision guide. In fact, we compare the performance of our system (using the information gain metric) to that of an existing online laptop decision guide, on the personalogic site (www.personalogic.com). The personalogic system allows customers to impose constraints, which are then exploited by a constraint satisfaction engine in order to prune alternatives that do not satisfy them. Our goal was to prove that our system could retrieve a manageable set of cases with fewer questions. In other words, we wish to show that an adaptive intelligent approach to question order works better than a predetermined one.

However we first had to extend our algorithms for calculating information gain and clustering to deal with unknown values, as there were many of these in the laptop data set. There are a few different approaches advocated in the literature for dealing with missing values in data (see [11] and [8] for a discussion). You can ignore cases with missing data, but this can often leave you with too few cases to work with. Another approach is to set each missing value to the mean value for that feature. However it makes sense that only cases in the same cluster as the case missing a value should contribute to the mean value calculated. Therefore this method cannot be used before clustering data. The method we chose to implement (for the clustering algorithm) firstly ignores features with unknown values when calculating the distance between two cases. However this alone could lead to distance values close to 0 for two cases that between them have a lot of unknown values, and does not allow for the fact that they might be very distant along some of the unknown dimensions. Multiplying the distance value by the total number of features, and dividing it by the number of features with *known* values in both of these cases, makes the distance increase as the number of unknown values increases. It is worth noting that this method can only be used when all the features have the same weight, and so would have to be modified if it were to be used with features of differing weights.

A solution to the problem when calculating information gain was found in [11]. The approach chosen by Quinlan for use in the C4.5 algorithm is quite straightforward. Cases with unknown values are ignored when calculating the expected entropy after partitioning the cases using *f*, and then the *information gain* value for *f* (see formula (1)) is multiplied by the fraction of cases which have known values for that feature. This results in more realistic (lower) information gain values for features that have many unknowns.

Unknown values also pose a problem for retrieval. The method of retrieval used in our implementation gives cases in the case-base “scores” according to their closeness to the target case along all the dimensions input. If a case is missing a value along a particular dimension under consideration, it won’t get a score, and so will not make it into the "retrieved set" for the next stage, even if it is close to the target along other dimensions not yet considered. A way around this would be to replace unknowns with mean values as discussed earlier, or to use probabilities as in C4.5. This is a matter for further investigation and has not been addressed yet due to time constraints.

Next we had to decide on the best stopping criterion to use when generating our refining questions With a decision guide the user generally wishes you to “whittle down” the possibilities into a manageable set, therefore the number of retrieved cases would seem a sensible stopping criterion. Therefore we used this instead of those listed in section 4.1. We chose 20 as the cut-off figure in our experiments, but in an online situation the user could choose their own preferred maximum.

Similar experiments to those described in section 4.1 were performed on the laptop data. However since the data set contains nominal data we could not use the variance metric, and as it contains unknown values, we could not use the DashLiu metric either, as it does not allow for them. Therefore we simply compared the information gain metric to random generation of refining questions, to ensure the metric was producing useful feature orderings using the detected clusters. Fig. 4 shows the results of this comparison, and not surprisingly, the information gain metric produces better results than ordering the questions randomly. In fact, the feature ordering

determined by this metric needs, on average, only 4.66 (or 33%) of the 14 features available for retrieval. This result would suggest that this metric works well with the laptop data set.

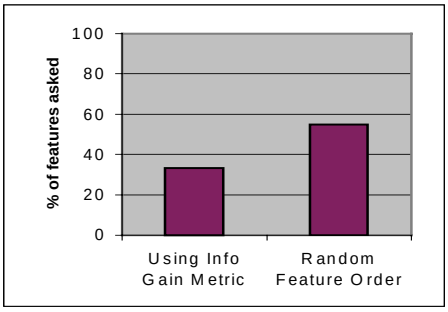

Fig. 4. Comparison of information gain method and random ordering of questions

In order to compare our system to the personalogic system, we chose 20 test cases at random from our case-base as hypothetical targets to guide our input to both systems. With each target case we input answers to the personalogic system in the order asked, stopping when the retrieved set had been cut down to 20 cases or less. The experiments with our system followed the same procedure, but were performed automatically rather than manually. The number of input features necessary to achieve this "manageable set" differed depending on the target case. The ordering of questions in the personalogic system did suit some of the cases used as target cases. In all cases bar one, our system performed as well or better.

In some of the experiments, our system stopped asking questions *before* it reached the threshold of 20 retrieved cases. This happened in two different situations. Firstly, if all of the retrieved cases are of the same class, then the *entropy* of the retrieved set is 0 (see (**1**)), meaning no more information is needed in order to know the classification of an arbitrary case in the set. Therefore it does not make sense to ask any more questions. The second situation occurs when all of the retrieved cases have the same values for each of the as-yet-unknown features. Therefore asking about these features will be of no use, and is reflected in an *information gain* value of 0 (see (**3**)) for each of the features. Therefore although we don't use these situations as explicit stopping criteria as in our initial experiments (section 4.1), the retrieval algorithm naturally stops early when they occur, as there is nothing to be gained from asking further questions (and no way of calculating the order in which to ask them). Therefore it is not always possible for the retriever to continue asking questions until the threshold amount of retrieved cases has been reached. In this situation the cases are returned in the ranked order indicated by the similarity measure described in section 2.1.

In fact, in the results shown in Fig. 5, the threshold of 20 was only reached in 7 out of the 20 experiments. We graph all the results however, and the 7 experiments in which the two systems can be more "directly" compared are marked. It can be seen that our system performs better in all of those experiments.

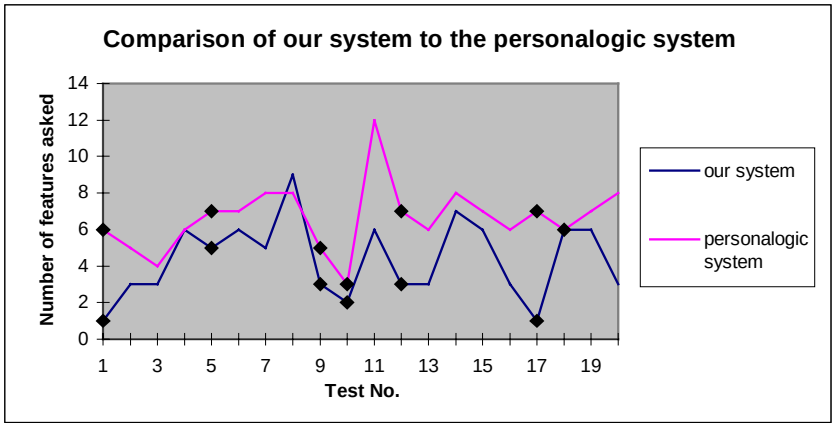

Fig. 5. Comparison of our system to the personalogic system in terms of number of features asked – marked points indicate tests in which our system retrieved 20 cases or less and therefore can be more directly comparable to the personalogic results.

5 Conclusions and Future Work

The above experiments show that the I-CBR based technique is indeed a powerful mechanism for reducing the number of features needed for retrieval. The fact that this metric requires the imposition of a cluster structure on unlabelled data has proved not to be a problem as it still out-performs the alternatives.

However the key advantage of the incremental approach, whichever metric we use for ordering the questions, is that it *adapts to the user* at run-time. The ordering determined relies entirely on the values entered by the user. In experiments to compare the number of features asked when using this adaptive approach, to those asked using the initial feature ordering calculated over the entire case-base, our dynamic approach not surprisingly performed better. With the laptop case-base, the dynamic approach asked on average 33% of the features, whereas the static approach required 51% of the features to reach an answer.

Therefore our system would be an ideal basis for an online decision guide. This could be a product selection site for a single company or a broker-type decision guide like the personalogic site. Since the goal of sites promoting e-commerce is to entice the user to buy, many are using intelligent guides to navigate the user through the products. However having to go through a lengthy interaction in order to find suitable product may turn many users off the idea. Therefore providing focus to the questioning, which leads to shorter interaction time, would make the whole process a lot more attractive to users.

Then there is the added benefit of reduced network request-response cycles. With the explosive growth of the Internet have come problems of increased server load and network latency. This means that systems that require interchange of data between

server and client over the network can be slow to unusable (at peak times). The work describe here is part of a larger project, which is investigating methods for developing distributed CBR systems that *minimise* the load on the servers and the network, in order to increase response time and usability. Shorter-lived queries mean reduced network traffic, and so this work is complimentary to the project as a whole.

5.1 Future Work

One criticism of this approach might be that this method of deciding which question to ask at each stage may result in an "unnatural" ordering of questions, since human intuition on the relative importance of the features is not taken into account. To address this, our interface allows the user to choose to input a value for whichever feature they wish, if they are not interested in the feature they are prompted for (see Figure 6). In the future we propose to perform some user trials to determine if the question ordering appears reasonable or counter intuitive.

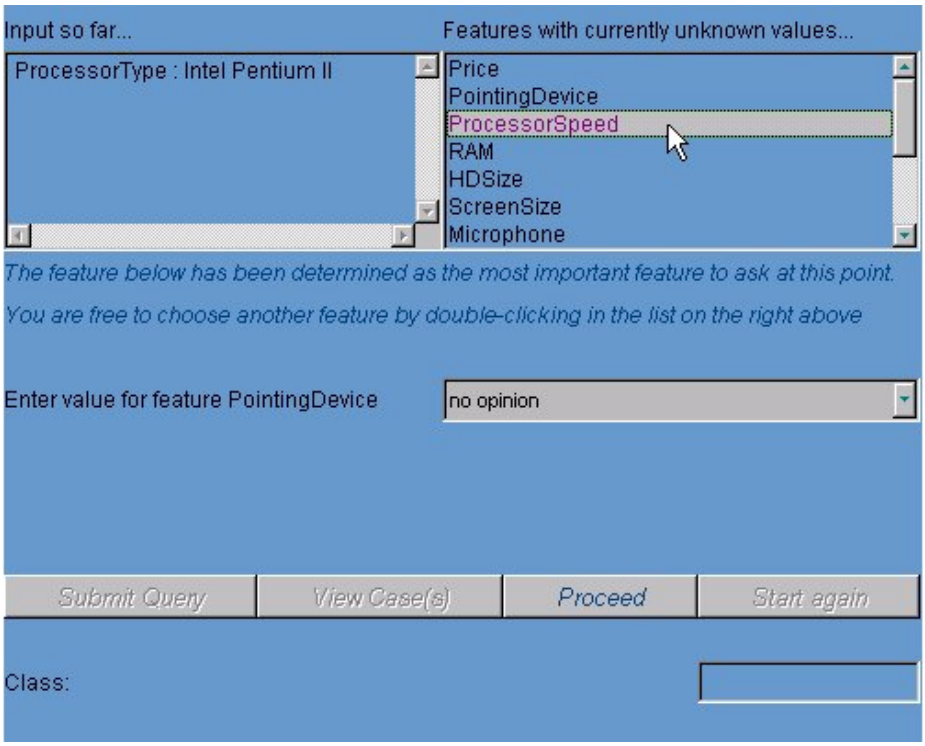

Fig. 6. The prototype interface to our laptop decision guide showing how the system dynamically selects the information to provide next but the user can override with their own choice.

Another issue that warrants exploration is the promising performance of the simple variance metric. So far we have not pursued this approach because of the problem of

measuring variance for category variables. However entropy is known to be a good measure of mixture for category variables [4] and we propose to evaluate an extension of the variance metric that incorporates entropy.

References

1. Aha David W., Bankert Richard L. (1995) *A Comparative Evaluation of Sequential Feature Selection Algorithms*, in proceedings of AI & Statistics Workshop 1995.
2. Bishop C.M. (1995) *Neural Networks for Pattern Recognition*, Oxford University Press, Oxford, 1995.
3. Cunningham P., Smyth B. (1994) *A Comparison of model-based and incremental case-based approaches to electronic fault diagnosis*, in proceedings of the Case-Based Reasoning Workshop, AAAI-1994.
4. Cunningham, P., Carney, J., (2000) *Diversity versus Quality in Classification Ensembles based on Feature Selection*, to be presented at European Conference on Machine Learning, Barcelona Spain, June 2000.
5. Dash M., Liu H., Yao J. (1997) *Dimensionality Reduction for Unsupervised Data*, in proceedings of IEEE International Conference on Tools with AI (TAI-97), pp. 532-539.
6. Dash M., Liu H. (1999) *Handling Large Unsupervised Data via Dimensionality Reduction*, in proceedings of SIGMOD Data Mining and Knowledge Discovery Workshop, (DMDK), Philadelphia, USA, May 1999.
7. Devaney M., Ram, A. (1997) *Efficient Feature Selection in Conceptual Clustering*, in proceedings of the 14th International Conference on Machine Learning, Nashville, TN, July 1997.
8. Everitt, B.S. (1993) Cluster Analysis, 3rd Ed., Edward Arnold.
9. Fisher, D.H. (1987) *Knowledge Acquisition via incremental conceptual clustering*, in Machine Learning, Vol. 2., pp.139-172.
10. Kaufman L., Rousseeuw P. (1990) Finding Groups in Data: An Introduction to Cluster Analysis, Wiley, New York.
11. Quinlan J.R. (1986) *Induction of Decision Trees*, in Machine Learning, Vol. 1, No. 1, pp. 81-106.
12. Smyth B., Cunningham P. (1994) *A Comparison of Incremental Case-Based Reasoning and Inductive Learning*, in proceedings of the 2nd European Workshop on Case-Based Reasoning, Chantilly, France, November 1994.