Data Min Knowl Disc (2008) 17:253–282
DOI 10.1007/s10618-007-0082-x

Maximizing classifier utility when there are data acquisition and modeling costs

Gary M. Weiss · Ye Tian

Received: 2 December 2006 / Accepted: 2 August 2007 / Published online: 6 September 2007

Abstract Classification is a well-studied problem in data mining. Classification performance was originally gauged almost exclusively using predictive accuracy, but as work in the field progressed, more sophisticated measures of classifier utility that better represented the value of the induced knowledge were introduced. Nonetheless, most work still ignored the cost of acquiring training examples, even though this cost impacts the total utility of the data mining *process*. In this article we analyze the relationship between the number of acquired training examples and the utility of the data mining process and, given the necessary cost information, we determine the number of training examples that yields the optimum overall performance. We then extend this analysis to include the cost of model induction—measured in terms of the CPU time required to generate the model. While our cost model does not take into account all possible costs, our analysis provides some useful insights and a template for future analyses using more sophisticated cost models. Because our analysis is based on experiments that acquire the full set of training examples, it cannot directly be used to find a classifier with optimal or near-optimal total utility. To address this issue we introduce two progressive sampling strategies that are empirically shown to produce classifiers with near-optimal total utility.

Keywords Data mining · Machine learning · Induction · Decision trees · Utility-based data mining · Cost-sensitive learning · Active learning

Responsible editor: Geoff Webb.

G. M. Weiss (✉) · Y. Tian
Department of Computer and Information Science, Fordham University, Bronx, NY 10458, USA
e-mail: gweiss@cis.fordham.edu

1 Introduction

Classification is an important application area for data mining. Originally only simple measures like predictive accuracy were used to evaluate the utility of a classifier, but as the field advanced and more complex problems were addressed, more sophisticated performance measures were introduced—measures that more accurately reflect *how* the classifier will be used in its target environment. However, the quality of a classifier is still almost always measured exclusively by its performance on new examples, without considering the costs associated with acquiring the training data or the costs associated with generating the model. Recently, the topic of Utility-Based Data Mining has focused attention on the need to maximize the utility of the entire *data mining process* (Weiss et al. 2005; Zadrozny et al. 2006). The research in this article makes several contributions to Utility-Based Data Mining.

The first contribution of this article is that it fills a gap in the research on Utility-Based Data Mining by analyzing the impact on data mining of what Turney (2000) refers to as "the cost of cases," which is the cost associated with acquiring complete training examples. We find it surprising and notable that this cost has not been studied before because this cost occurs in many real-world situations. In particular, we have experienced this cost as data mining practitioners in several settings. In one case, customer data had to be acquired from an external vendor, which charged based on the amount of data purchased. In another instance the raw data was available for free, but at a level unsuitable for data mining. The process of aggregating the data to the appropriate level for data mining was extremely time consuming and computationally expensive, given that billions of records were involved. Thus, even though the raw data was available at no cost, there was a cost in generating useful data—and this cost could be reduced by generating fewer aggregated records.

It is also surprising that the cost of cases has not been studied since other classifier costs have been studied extensively. In particular, the cost of labeling examples (Lewis and Catlett 1994) and the cost of measuring features (Greiner et al. 2002; Veeramachaneni and Avesani 2003) have been studied in the context of active learning (Cohn et al. 1994), where one has a *choice* of what to label or to measure, while the costs associated with misclassification errors have been studied in the context of cost-sensitive learning (Elkan 2001). In this article we study the trade-off between the amount of training data used and overall classifier utility when each training example incurs a fixed cost (this is the "cost of cases"). We view this as a simple instance of active learning (a topic subsumed by Utility-Based Data Mining), where the only choice one has is the amount of training data to acquire.

One of the challenges of Utility-Based Data Mining is to maximize the utility of the data mining process when there are competing costs and benefits. This can be especially difficult when these costs and/or benefits occur at different stages in the data mining process and hence cannot be optimized simultaneously. For classification tasks the data mining process can be thought of has having three main stages: (1) data acquisition, (2) model induction, and (3) application of the

induced model to classify new data. We know of no prior data mining research that considers the utility values associated with all of these stages. However, the utility/cost model that we use in this article, described in Sect. 2, allows us to do just that. While our utility model is not completely realistic in that it does not consider all possible costs and/or benefits, it is nonetheless more complete than what is typically used in practice, which only considers the utility associated with applying the induced classifier to new data. Furthermore, we believe that the analysis we provide based on this utility model provides general insights into the data mining process and the trade-offs involved when data mining—and just as importantly, that our analysis can be adapted for more sophisticated utility models. We also expect that this work will stimulate more work in this area and lead to the analysis of increasingly sophisticated utility models. In summary, the second contribution of our research is that it addresses one of the central challenges of Utility-Based Data Mining by analyzing the trade-offs between decisions made at different stages of the data mining process and identifies, for a number of data sets, the decisions that lead to the *optimal-utility classifier*.

The empirical analysis of our utility model demonstrates how different decisions lead to classifiers with different total utility. While this enables us to identify the decisions that lead to the optimal-utility classifier, this is not an *actionable* strategy since the empirical analysis involves trying out a variety of decisions (such as the specific number of training examples to acquire) and once a decision is made the associated cost is immediately incurred. In order to develop an actionable strategy for identifying the optimal-utility classifier, we need to search the space of decisions more carefully. The third main contribution of our research is that we develop such a strategy, by using progressive sampling (Provost et al. 1999) to heuristically identify the number of training examples that maximizes the utility of the data mining process. Our results indicate that this heuristic method performs quite well. One should also be able to adapt this progressive sampling strategy to handle more sophisticated utility models. It is worth pointing out that this notion of a progressive sampling strategy fits nicely with the data mining paradigm (Fayyad et al. 1996), which is an iterative, incremental, process. Such a process is critical since only in this way can we hope to optimize decisions that are inter-related but are made at different stages in the data mining process. In fact, we would argue that this is not coincidental—the data mining process is iterative *because* of the need to refine a set of interrelated decisions. While the overhead associated with an iterative process may seem prohibitive, such overhead is almost always unavoidable when tackling complex, real-world, problems.

This article is organized as follows. In Sect. 2 we describe the utility/cost model. In Sect. 3 we describe our experiments and, in Sect. 4, we present our main experimental results. These results allow us to analyze the relationship between the factors in our utility model and overall classifier utility. In Sect. 5 we present two simple progressive sampling strategies that are shown to be effective at generating classifiers that are near-optimal in terms of the overall utility of the data mining process. Related work is discussed in Sect. 6 and Sect. 7

provides some concluding remarks and outlines possible future extensions to this work.

2 The utility/cost model

The total utility of a classifier, which incorporates the costs and benefits associated with the entire data mining process, can only be evaluated if the costs and benefits are enumerated and assigned specific weights. In this section we describe our utility model, the motivation for it, and its limitations.

The data mining process for a classification task can be partitioned into the three stages described in Sect. 1. The total utility of a classifier can be described conceptually as the sum of the utilities for these stages, as shown in Eq. (1).

$$\begin{aligned}\text{Total Utility} = {} & \text{Utility}_{\text{data-acquisition}} + \text{Utility}_{\text{model-induction}} \\ & + \text{Utility}_{\text{induced-classifier}} \end{aligned} \tag{1}$$

The utilities associated with data acquisition and model induction will always be non-positive and generally will be negative. We expect to derive benefits from acquiring the data and inducing the model, but these benefits will be realized in the third stage. This third component, the utility of the induced classifier, will generally have a positive utility. The Utility-Based Data Mining problem then is to maximize the total utility. Note that for Eq. 1 to be meaningful the terms must share the same units. We discuss this shortly.

The utility of the induced classifier could be measured by assigning a positive utility to each new example correctly classified and a negative utility to those incorrectly classified. However, most work in cost-sensitive learning assigns a cost to the incorrect classifications and no cost to the correct classifications, and we adopt this scheme. Thus our task is to minimize the total cost rather than to maximize total utility. This is reflected in Eq. (2). We assume that all costs are expressed in the same units, such as dollars.

$$\begin{aligned}\text{Total Cost} = {} & \text{Cost}_{\text{data-acquisition}} + \text{Cost}_{\text{model-induction}} \\ & + \text{Cost}_{\text{misclassification-errors}} \end{aligned} \tag{2}$$

There are numerous ways to measure the costs associated with these three stages of the data mining process. We make a specific set of assumptions. For the cost of data acquisition, we only consider the "cost of cases" described in Sect. 1. We do this because this cost has never been studied in detail. We measure the cost of model induction exclusively in terms of CPU time, although we recognize that this cost could be measured in other ways (memory used, elapsed time, hardware costs, etc.). Given that we are more concerned with showing how to trade off the costs at different stages of the data mining process than with the specific costs, we believe that this simplifying assumption is reasonable. Finally, the cost of misclassification errors is conceptually straightforward to compute, given a fixed cost for each error.

Equation 3 shows how total cost is actually calculated in our study. For each experiment we know the number of training examples, n, the CPU time required to build the classifier, *CPU*, and the estimated error rate of the classifier e, based on its performance on a test set. The data acquisition cost is simply the number of training examples, n, multiplied by the cost per training example, *Ctr*. The cost of model induction is the CPU time multiplied by *Ctime*, the cost per unit of CPU time. Computing the cost of misclassification errors is not quite as straightforward since this cost depends on the number of examples in the "score" data set, S, that are ultimately classified using the classifier. This score set is not the same as the test set, since the examples in the test set contain the correct classification and the sole purpose of the test set is to estimate the error rate of the classifier. In order to compute the number of errors that the classifier will make, we must multiply e by the size of S, denoted |S|, and then multiply this by the cost per error, *Cerr*, to get the cost of misclassification errors. Note that the three cost "factors", *Ctr*, *Ctime*, and *Cerr* must all convert the costs to the same units, such as the cost in dollars.

$$\text{Total Cost} = n \cdot Ctr + CPU \cdot Ctime + e \cdot |S| \cdot Cerr \tag{3}$$

Although we do not know the value of |S| for any of the data sets in this article, a domain expert should be able to estimate its value, although this may not always be a simple task. With specific domain knowledge we should also be able to estimate *Ctr*, *Cerr*, *and Ctime* and thus calculate the total cost. Unfortunately, for the data sets used in this study we do not have this information. Therefore we treat these values as variables and analyze the behavior for a wide range of values. The problem with this is that four variables make a thorough analysis difficult. However, we can eliminate one variable by arbitrarily assuming |S| is 100. This does not reduce the generality of our results because we can easily account for other values of |S| via a simple calculation. Namely, the cost of misclassification errors is proportional to the product |S|·*Cerr* so that if we find that |S| is 100,000 instead of 100, we can simply look at the experimental results for *Cerr*/1,000 rather than *Cerr*. In a sense we are measuring the cost of misclassification errors in terms of every 100 score examples and then adjusting for different score set sizes. We can simplify things further by only tracking the ratio of the three remaining cost factors. While the actual total cost will depend on the actual cost factors, the optimal training set size will only depend on the ratio of these costs. Note also that by specifying only the ratio of these costs, the units are irrelevant—as long as the three terms in Eq. (3) share the same units.

For our experiments that do not consider the cost of model induction, we simply report the cost ratio, *Ctr:Cerr*, where *Ctr* is typically 1 and $Cerr \geq 1$. Because we want to plot our results using numerical values, our figures report the relative cost, which is simply *Cerr*/*Ctr*. For example, if the cost ratio is 1:100 then the relative cost is 100. Note that in this case, from a utility perspective it is an even trade-off to purchase 100 training examples if it will reduce the number of errors by 1 (as noted this assumes |S| is 100). We can remove the condition on |S| by stating things in a slightly different manner: purchasing 100 training

examples leads to an even trade-off if it results in a 1% (1/100) reduction in error rate. When the cost of model induction is also considered, an additional variable, *Ctime,* is introduced and its value is also measured relative to the other two cost factors.

One potential issue with Eq. (3) is that if |S| is sufficiently large then the cost of misclassification errors will dominate and no analysis is required—just acquire as many training examples as possible and do not worry about the data acquisition or model induction costs. We do not believe that the cost of misclassification errors will always dominate the other costs. First, for some domains the cost of acquiring training data is very significant and once a certain amount of training data has been acquired, it may take tens or hundreds of thousands of additional training examples in order to improve accuracy by even a tenth of a percent (we observe this in Sect. 4 for several data sets). It is within that region that we expect our utility model and analysis to be most useful. In addition, |S| need not always be extremely large. As an example, consider the domain of game playing. If the goal is to learn something about an opponent so that one can design a game-playing strategy tailored to this opponent, the training data will usually be costly, in terms of time, or money if betting is involved. For example, if you want to learn something about an opponent in poker "you may play only 50 or 100 hands against a given opponent and want to quickly learn how to exploit them" (Hoehn et al. 2005). Finally, related work seems to support our intuition that costs associated with data acquisition and model induction are important. For example, the entire field of active learning is based on the assumption that error cost will *not* totally dominate the various data acquisition costs—if it did then active learning would be unnecessary. Similarly, the focus on scalable data mining algorithms would not be necessary if the cost of misclassification errors always dominated the cost of computation.

One concern is whether a practitioner will be able to accurately estimate the values of the three cost factors or the size of the score set. Fortunately the figures we generate in Sect. 4 show the relationship between total cost and these cost factors for a variety of values and this can aid a practitioner with incomplete knowledge by allowing him to evaluate any number of "what if" scenarios in order to help determine the optimal training set size. The figures may also show that the utility of the classifier is relatively insensitive to certain costs, which can also be helpful. The problem a practitioner faces here is actually quite similar to a problem often encountered in cost-sensitive learning, since specific misclassification cost information is often not known. In that situation a practitioner may get some guidance by viewing any one of a number of "performance curves" that encodes the performance of the classifier for a variety of different decisions. The most common of these are precision/recall curves (Van Rijsbergen 1979), lift curves (Berry and Linoff 2004), ROC curves (Provost and Fawcett 2001), and, more recently, cost curves (Drummond and Holte 2006). Thus the analysis and visualization techniques that we provide can aid a practitioner with incomplete domain knowledge.

Table 1 Description of data sets

Large data sets		Medium data sets		Small data sets	
Forest-covertype	581,012	Adult	21,281	Network1	3,577
Census-income	299,284	Coding	20,000	Kr-vs-kp	3,196
Protein	145,750	Blackjack	15,000	Move	3,029
Physics	50,000	Boa1	11,000	German	1,000

3 Description of experiments

The analyses in this article are derived from a common set of experiments. These experiments vary the size of the training set, generate a classifier from the training data, and then record the training set size, accuracy of the induced classifier, and the CPU time required to generate the classifier. These three measured quantities are later combined with specific cost information, as described in Sect. 2, to determine the total cost associated with a classifier and to evaluate the progressive sampling strategies described in Sect. 5. In this section we provide the details of our experimental methodology and describe the data sets employed in our study. The results of these experiments are provided in Sect. 4.

All of the experiments in this paper use C4.5 (Quinlan 1993), a popular decision tree learner that is a descendant of ID3. The twelve data sets analyzed in this article are described in Table 1. For each data set the total number of examples, for training and testing, are provided. The data sets are partitioned into three groups (small, medium, and large) to simplify the presentation of our results. The data sets were obtained from the following sources: the forest-covertype and census-income data sets were obtained from the UCI KDD Archive (Hettich and Bay 1999), the protein and physics data sets were obtained from the KDD Cup 2004 competition (Caruna et al. 2004), the adult, kr-vs-kp and german data sets were obtained from the UCI Machine Learning Repository (Newman et al. 1998), and the coding, blackjack, boa1, network1, and move data sets were obtained from researchers at AT&T (these data sets have been used in previous studies and are available from the author). The protein and physics data sets were utilized in a simpler manner than in the KDD-Cup competition, in that each record is treated as a single example and our learning task is to maximize predictive accuracy.

For all experiments, 25% of the available data is randomly selected and placed into the test set, while the remaining data is available for training. In order to determine the relationship between training set size, predictive accuracy, and the time required to build a model, a variety of training set sizes are generated and then used to build a classifier. Our basic sampling strategy is simple and incrementally builds larger and larger training sets using a constant increment amount. For each data set we generate 50 uniformly spaced training set sizes, using random sampling from the 75% of the data allocated for training. In addition to these uniformly spaced training set sizes, we also evaluate

the following five (small) training set sizes, since we expect the learning curves to exhibit dramatic changes when little data is available: 10, 50, 100, 500, and 1000. Other sampling schedules could have been employed, but as we will see in Sect. 4.1, this simple schedule is adequate for generating good learning curves.

In order to improve the quality and statistical significance of the results, multiple runs are employed and the reported accuracies and CPU times are based on the averages over these runs. Due to the large number of experiments and the computational resources required to run these experiments, fewer runs were executed for the large data sets. For the small and medium sized data sets 100 runs were executed, while 20 runs were executed for all of the large data sets except the forest-covertype data set, which used only 5 runs (in the next section we show that for the large data sets many runs are not necessary in order to generate reliable results). Throughout this article we place the most emphasis on the large data sets, because those are the most representative of the types of tasks we expect to encounter in practice—especially when data acquisition costs and model induction costs are an issue. Due to space considerations we focus our most detailed analyses on the forest-covertype data set, the largest data set in our study. However, summary results are often provided for all data sets.

There is one assumption in our experimental setup that deserves additional discussion. There is a maximum amount of training data available for each data set (i.e., 75% of the total data listed in Table 1). Given that one of the goals of this work is to identify the optimum training set size, this limit is an issue. Ideally we should be able to continue to acquire more and more data (at a cost). Because we cannot do this our experiments and analyses are limited in that they assume that there is a "maximum amount of potentially available training data." We do not believe that this is a critical issue given that some of our data sets are quite large and that, in practice, there often is a limit to the amount of data that can be purchased (e.g., there is a limit on the number of businesses that exist).

4 Experimental results and analysis

The analyses in this article require knowing the relationship between training set size and classifier accuracy, and hence in Sect. 4.1 we present the learning curves for all of the data sets employed in this study. We use these results in Sect. 4.2 to analyze the cost of cases, by looking at the relationship between training set size and total cost. In this section we also determine the training set size that yields optimal overall performance. This analysis is extended in Sect. 4.3, when we also consider the cost of model induction, measured in terms of CPU time. In Sect. 4.4 we provide a mathematical foundation for this work by showing how the optimal training set size can be derived from the learning curve.

4.1 The learning curves

The learning curves displayed in this section are generated using multiple runs, as described in Sect. 3. The learning curves for the large, medium, and small data sets are displayed in Figs. 1–3, respectively (the learning curves for the large data sets are plotted in separate sub-figures to improve their readability). The learning curves for the four large data sets all show a rapid increase in accuracy at the start, which then, as expected, diminishes as the training set size increases. None of the four learning curves in Fig. 1 have reached a plateau,

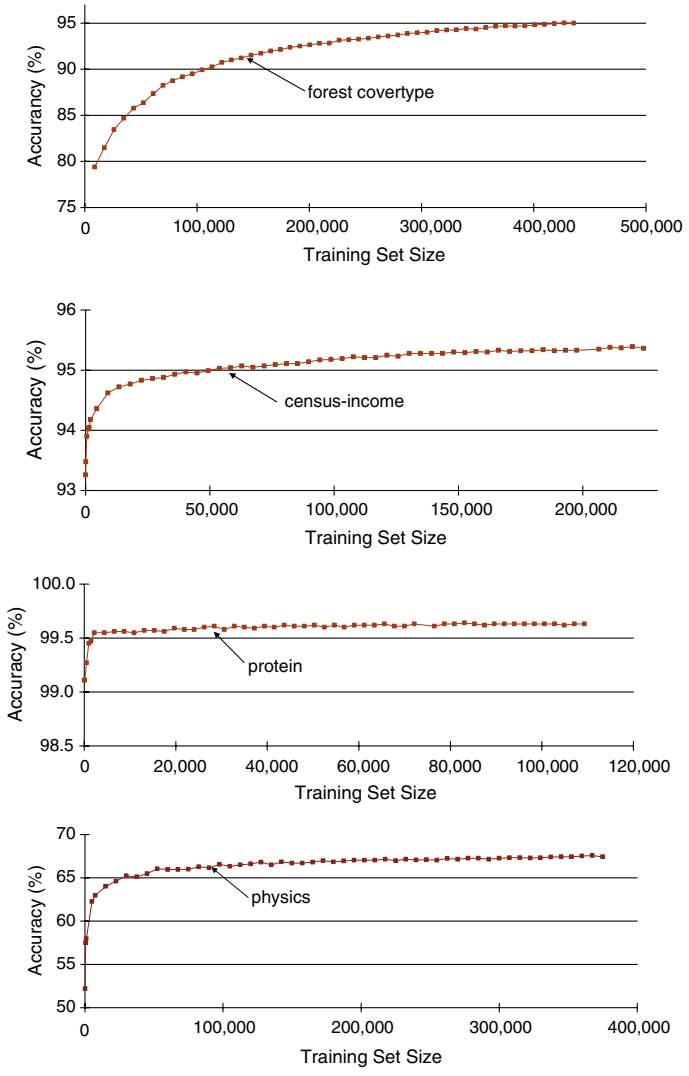

Fig. 1 Learning curves for large data sets

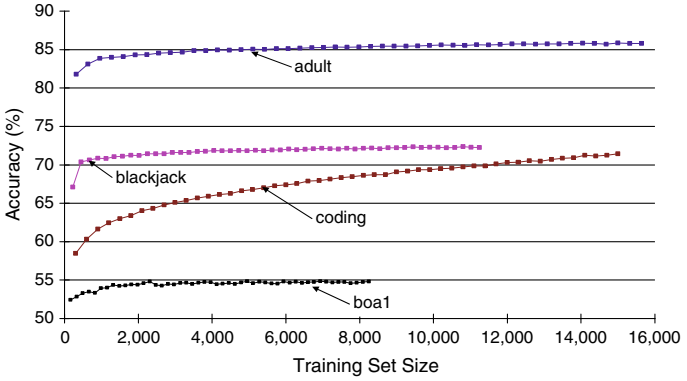

Fig. 2 Learning curves for medium-sized data sets

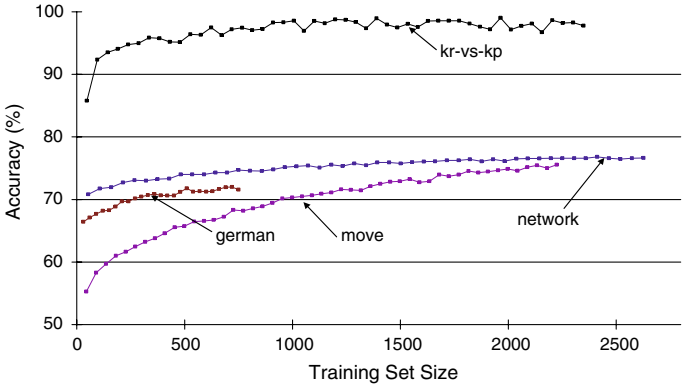

Fig. 3 Learning curves for small data sets

although the continuing improvement for the protein and physics data sets is only evident upon a careful examination of the underlying data. The fact that in some cases the improvement in accuracy is slight and may continue over tens of thousands of examples (e.g., for the protein data set) is noteworthy because in this situation the cost of acquiring training examples may very well prevent one from acquiring all of the potentially available training examples. The learning curves for the medium-sized data sets in Fig. 2 and the small-sized data sets in Fig. 3 are similar to ones for the large data sets, except that in one case, for boa1, it appears that the learning curve has reached a plateau.

We consider a learning curve to be "well behaved" if it is relatively smooth and monotonically non-decreasing. The learning curves for most of the data sets are relatively well behaved, although the large data sets, which have further spaced samples, tend to generate better behaved learning curves than the medium and small-sized data sets. We expect that the temporary decreases in accuracy in the learning curves are due to statistical variations in the performance of the learning algorithm, which would diminish if more runs were

used to generate the learning curves. Because the quality of the learning curves impacts the analysis in Sects. 4.2 and 4.3 as well as the progressive sampling strategy described in Sect. 5, we show, for a few data sets, how the number of runs impacts the behavior of the learning curves and how one might improve the behavior of the learning curves.

Figure 4 shows the impact of the number of runs on the learning curves for the census-income and coding data sets. Both of these clearly benefit from the use of more runs, which leads to smoother learning curves. As stated in Sect. 3 our analyses are based on 20 runs for the census-income data set and 100 runs for the coding data set. In the interest of space we do not show the analogous figures for the other data sets, but they show similar patterns.

The increased number of runs generally leads to more well-behaved learning curves because it increases the statistical significance of the results. To demonstrate this—and to show that we could generate better behaved learning curves if necessary—we took the results for the census-income data set using 20 runs and iteratively applied the Student *t*-test (Snedecor and Cochran 1989) between each data point and its successor. If the difference was not significant with 90% confidence then we eliminated the successor and repeated the *t*-test between the original point and the next successor. The results are displayed in Fig. 5. Note that even though the original curve was based on 20 averaged runs, the learning curve using the *t*-tests still leads to a better-behaved learning curve, with only

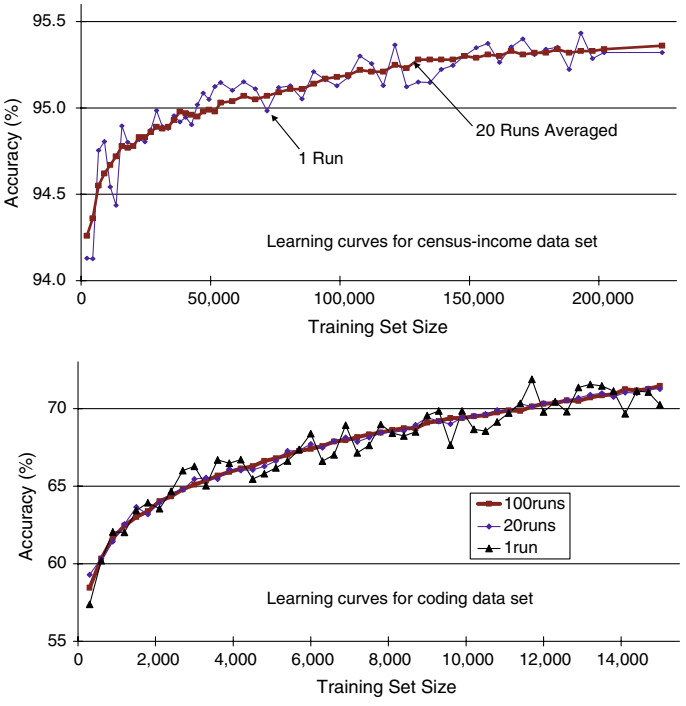

Fig. 4 Impact of number of runs on learning curves for two large data sets

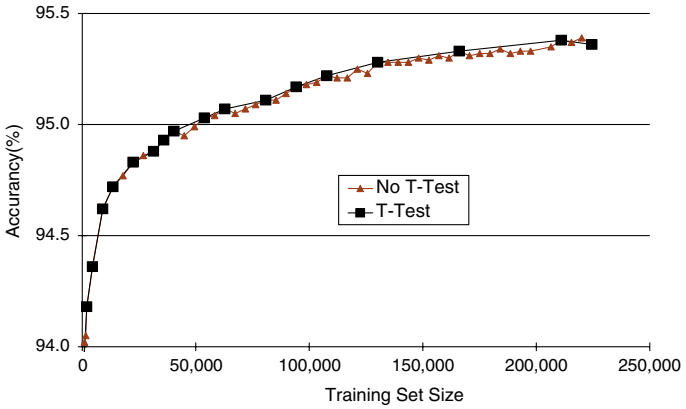

Fig. 5 Learning curve for census-income data set with and without *t*-test filtering

one point showing a decrease in accuracy (the last point). We believe that this method can be useful for improving learning curves and can especially improve the effectiveness of the progressive sampling strategy described in Sect. 5.

We consider the issue of how to best generate well-behaved learning curves a research question worthy of further study. The choice of learning algorithm, the number of runs, the distance between samples all will impact the behavior of the learning curve and a variety of statistical techniques could be use to help smooth these curves (including the technique described above). Because this is not the focus of our research, we leave this for future work and do not use this *t*-test "filtering" method in our analysis. However, our results indicate that such a method would lead to only modest improvements in our results.

4.2 Analysis of the cost of cases on classifier utility

In this section we analyze how the cost of training examples impacts the overall utility of a classifier. We use Eq. (3) to calculate total cost, but in this section ignore the second term, which concerns the cost of model induction (i.e., *Ctime* is set to 0). We begin with a detailed analysis of the forest-covertype data set and then provide summary results for the other data sets. Figure 6 shows the relationship between the total cost associated with the classifiers induced from the forest-covertype data set and the number of training examples. Each curve in Fig. 6 is labeled with a cost ratio (*Ctr:Cerr)*, which is required to compute the total cost. Note that we refer to the curves in Fig. 6 as utility curves because, as stated earlier, we view our work from the most general perspective, where cost is a form of utility, and because the term "cost curves" already has a specific meaning in the fields of machine learning and data mining (Drummond and Holte 2006).

The cost ratio in Fig. 6 that places the highest relative cost on the training examples is 2:1. In this case the curve is linear, indicating that the data

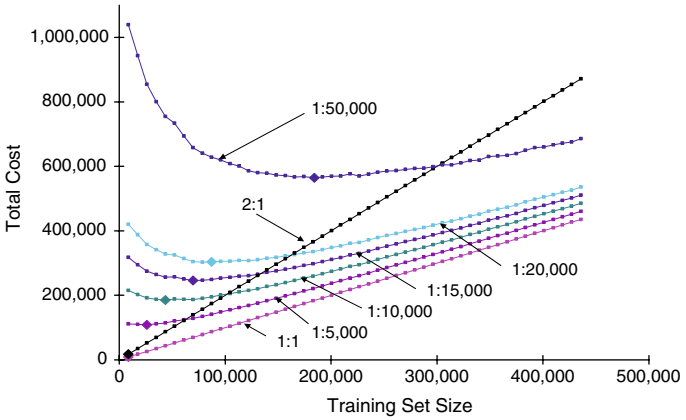

Fig. 6 Utility curves for forest-covertype data set

acquisition cost dominates the error cost (not surprisingly the 1:1 cost ratio also yields a linear curve with half the slope). As the cost ratio increases so that more emphasis is placed on the misclassification errors, the curve becomes non-linear and the minimum total cost (identified for each curve by the large diamond marker) no longer occurs at the minimum training set size, but rather shifts towards the larger training set sizes. At a cost ratio of 1:50,000 the lowest cost is achieved with 185,000 training examples.

One issue with Fig. 6 is that as the cost ratio becomes more skewed the total cost rises, which obscures some of the changes for the curves with lower total cost. To address this problem we normalize each curve by dividing the total cost by the maximum total cost associated with the curve. The resulting normalized utility curve for the forest-covertype data set is shown in Fig. 7. This method for representing the results also permits us to examine higher cost ratios and enables us to see that at a cost ratio of 1:1,000,000 the optimum strategy is to use all of the available training data. Figure 7, in conjunction with the learning curve for the forest-covertype data set in Fig. 1, shows that once the learning curve begins to flatten out, a great increase in the cost ratio is required in order for it to be profitable to acquire more training data. This is encouraging in that once we get past a certain point the optimal training set size is not overly sensitive to the exact value of the cost ratio; hence a good *estimate* of this ratio should be adequate. Figure 7 also makes it clear that using all of the potentially available training data is not a good strategy for most of the cost ratios analyzed.

The most critical information in Figs. 6 and 7 is the optimal training set size for each cost ratio. This information is summarized in Fig. 8, which plots, for each relative cost (*Cerr*/*Ctr*), the optimum training set size for the large, medium and small data sets. These optimal training set size curves can be used by a practitioner to determine the amount of training data to obtain even if the precise cost ratio is not known. Note that the optimal curve exhibits the full range of possible behaviors. At very low relative costs the best strategy is to acquire the minimum amount of data possible (our experiments start with more than

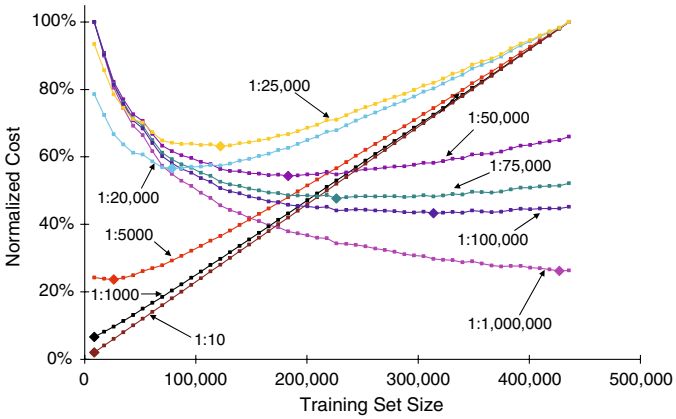

Fig. 7 Normalized utility curves for forest-covertype data set

zero examples) while at a high relative cost the best strategy is to acquire all available training data. Once the maximum amount of available training data is used, the curves are guaranteed to flatten out since the amount of training data used will be fixed as will be the performance of the induced classifier.

One issue concerning the optimal curves in Fig. 8 concerns the range of relative costs displayed on the *x*-axis. Are the relative costs toward the higher end of these ranges plausible? Would one ever want to acquire all potentially available training examples for the census-income and protein data sets when this is only optimal when the relative cost is greater than 800,000 and 2,000,000, respectively? We believe that these apparently very high cost ratios may realistically occur. First, since Eq. (3) assumes that the score set contains only 100 examples, the relative cost of 2,000,000 is equivalent to a cost ratio of 1:2,000 if the classifier will be used to classify 100,000 examples. In many situations the cost of an error may in fact be 2,000 times that of the cost of acquiring each training example—although in cases where the training data is expensive it may not be. Note that if more training examples were available for the large data sets and the rate of improvement for the learning curves continued to decrease, this would result in an even wider range of cost ratios for which the optimum strategy would not involve acquiring all potentially available training data.

4.3 The additional impact of model induction on classifier utility

The results in the previous section ignored the cost of generating the classifier. In this section we extend our analysis by including this cost, measured in terms of the CPU time required to generate the model (as discussed earlier any other costs associated with model induction are ignored). Therefore, in this section total cost is calculated using all of the terms in Eq. (3). Figure 9 shows the average CPU time required to build a classifier for each of the large data sets,

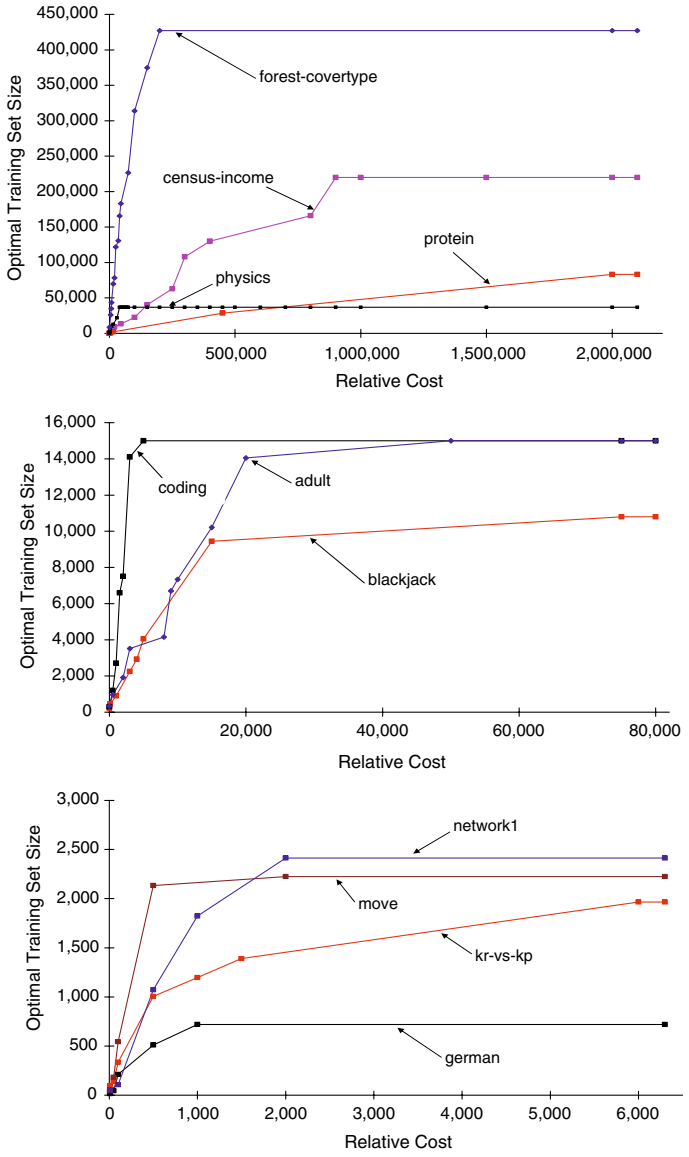

Fig. 8 Optimal training set sizes for large, medium, and small data sets

for varying training set sizes. Thus this figure shows the run-time complexity of C4.5.

Run-time complexity models, especially those that handle average-case rather than worst-case complexity, are not always available. However, previously reported empirical results for C4.5 (Provost et al. 1999) indicate that its run-time complexity varies between $O(n^{1.22})$ and $O(n^{1.38})$ and the results in

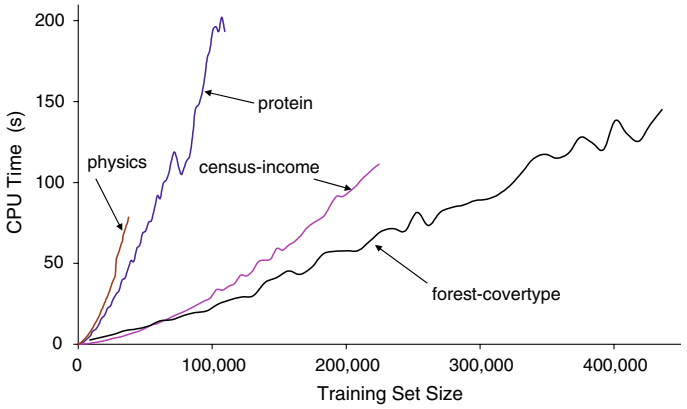

Fig. 9 Average CPU time to generate a single classifier

Fig. 9 are consistent with this. The actual complexity of a decision tree algorithm is not just based on the number of training examples, but also the complexity of the induced model and the pruning method used. However, because training set size is the only experimental parameter we analyze in this study, our primary interest is on how this impacts the CPU time required to induce the model. For more information on how other factors impact the time to induce a model, we refer the reader to Quinlan (1993), which provides a detailed description of C4.5 and its use of error-based pruning and to Breiman et al. (1983), which provides a valuable discussion of decision tree complexity and cost-complexity pruning. A comparison of decision tree pruning methods, including their computational complexity, is provided by Esposito et al. (1997), while Martin and Hirschberg (1996) provide a general discussion of the complexity of learning decision trees.

Returning to Fig. 9, one thing that is clear is that the CPU times are relatively modest in absolute terms, since all classifiers can be generated in under 4 min. However, given that the experiments for most of the large data sets are based on 20 runs, these times are actually more substantial—just over an hour for the protein classifiers. Furthermore, when progressive sampling is used to determine the appropriate training set size, the CPU times increase substantially, since the effective CPU time is the sum of the CPU times associated with each of the evaluated training set sizes. Whether the CPU times incur a significant cost for these data sets or not, we believe there are some situations where the time cost will be substantial and needs to be factored into total utility. As stated earlier, if this were not so, there would be less interest in the complexity of learning methods.

Figure 10 shows the impact of the CPU cost on the normalized utility curves for forest-covertype data set. Figure 10 corresponds to Fig. 7 except that in Fig. 10 the cost ratio is fixed at 1:100,000 and the curves are now labeled with the CPU cost factor instead of the cost ratio. Note that a CPU cost factor of 1,000 means that the number of CPU seconds is multiplied by 1,000 to obtain

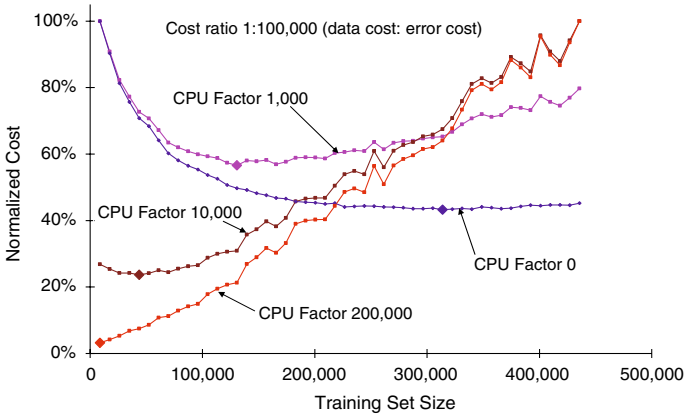

Fig. 10 Normalized utility curves (with CPU cost) for forest-covertype data set

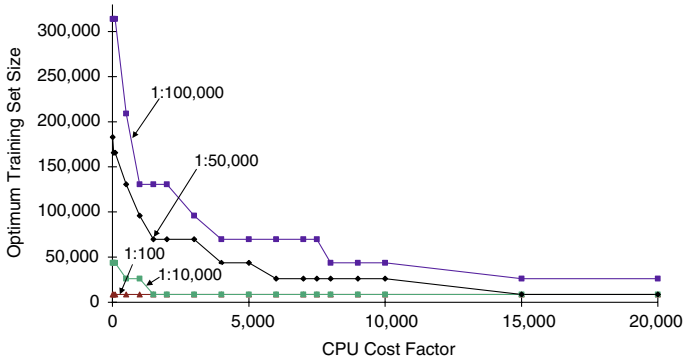

Fig. 11 The impact of CPU cost factor on forest-covertype optimum training size

the CPU cost. The curve in Fig. 10 corresponding to a CPU cost of 0 is identical to the curve in Fig. 7 labeled with the cost ratio of 1:100,000.

Figure 10 demonstrates that as the CPU cost factor increases, the optimum training set size (indicated by the enlarged diamond markers) moves toward smaller and smaller training set sizes. This is as expected since the CPU time required to build the model increases with training set size, as was shown in Fig. 9. Figures like this one show the sensitivity of the domain to CPU costs. Figure 11 shows the optimum training set sizes given a variety of different CPU cost factors and for a variety of different cost ratios (these label each curve). Figure 11 therefore shows the trade-offs involved between training data cost, error cost, and modeling (i.e., CPU time) costs. These modeling costs are also analyzed in Sect. 5 in the context of progressive sampling.

4.4 Relationship between learning curve and optimum training set size

One question that has intrigued us is the relationship between the learning curves for each data set, which were displayed in Figs. 1–3, and the corresponding optimum curves, which are displayed in Fig. 8. For example, one of the specific things we were interested in is how the shape of the learning curve impacts the shape of the optimal curve. In addition, we were interested in seeing if we could find an analytical relationship between these two curves, since this would provide more of a theoretical foundation for our work, and might also allow us to predict the optimal training set size.

Our approach is to mathematically describe the learning curve associated with a data set and then derive the optimal curve from it. In this section we start by assuming that we are given a function that describes the learning curve. From this, we show the step-by-step process of deriving the optimal curve. We then fit a function to the learning curve associated with the forest-covertype data set and then apply the same derivation process. We show that the derived optimal curve approximates the actual one (the differences are due to our not perfectly fitting the original learning curve). Note that because our focus is on the derivation process and not function approximation, we only try to fit a very simple function to the learning curves. We leave more sophisticated methods for future work. In this section we only consider the cost of cases and ignore the cost of model induction, although our analysis could be extended to handle this cost given a function that maps the number of training examples to the CPU time required to induce the model.

For our simple example, we assume that the learning curve is described by $f(x) = x/(x+1)$, where x represents the number of training examples and $f(x)$ the represents the accuracy of the induced classifier. The error rate of the classifier is then $1 - x/(x+1)$, which reduces to $1/(x+1)$. Assuming that the score set size |S| is 100 and the relative cost ratio is R (i.e., $Ctr = 1$ and $Cerr = R$), then Eq. (3) from Sect. 2 yields a total cost of $x + 100R/(x+1)$. This equation can be used to plot the utility curve, with the training set size x on the x-axis and total cost on the y-axis. Since we want to find the optimum training set size, which is the minimum of the utility curve, we take the first derivative of this equation and set it to 0. Thus we want to solve Eq. (4) below for training set size x, where R is a constant.

$$d(x + 100R/(x+1))/dx = 0 \tag{4}$$

Using the quotient rule for the second term and then solving for x, we get:

$$x = 100\sqrt{R} - 1 \tag{5}$$

We can then generate the optimal curve for the learning curve by plotting the relative cost ratio R on the x-axis and the optimal training set size x on the y-axis. Figure 12 shows the learning curve described by the equation $f(x) = x/(x+1)$ and the optimal curve derived from this learning curve, using Eq. (5).

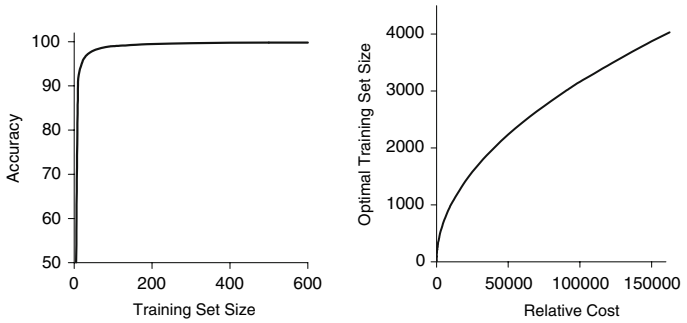

Fig. 12 Learning and optimal curves for $f(x) = x/(x+1)$

Figure 12 demonstrates that the optimal curve will be perfectly smooth and monotonically increasing if the learning curve is also smooth and monotonically increasing. However, the learning curve in Fig. 12 improves so rapidly that it is not representative of even moderately difficult learning problems. We approximate the learning curve for the forest-covertype data set by adapting the function $f(x) = x/(x+1)$. Since the observed accuracy of the forest-covertype learning curve begins at 78%, we add another term that ensures that the learning curve starts with this value. We then tried other values to replace the "1" in the denominator until we achieved a reasonably good fit with the actual learning curve. Equation (6) shows the function we use to approximate the forest-covertype learning curve.

$$f(x) = 0.78 + 0.22[x/(x + 120{,}000)] \tag{6}$$

Figure 13 shows the actual forest-covertype learning curve and the one generated by Eq. (6), while the empirically generated optimal curve for the forest-covertype data set is shown in Fig. 14 along with the one derived from Eq. (6) (we do not show the derivation but it follows the same steps as for the previous derivation). Note that the derived optimal curve will select a negative training set size for very low relative costs. Of course this is not possible (i.e., you cannot "sell" training examples) and in practice the curve should be set to 0 rather than being allowed to become negative.

This technique of finding an approximation of the actual learning curve is used here to gain a better understanding of the relationship between the learning and optimal curves. However, it is possible that this technique could be useful in practice. One could take a partially generated learning curve, fit a function to it, and then analytically find the optimum training set size for any relative cost. One could then "purchase" the optimal number of examples. This could be used as an alternative to the progressive sampling strategy described in the next section.

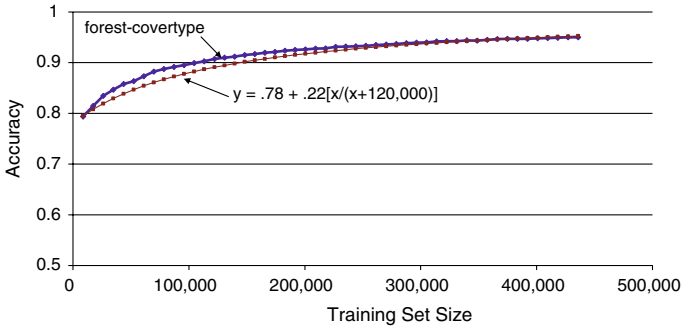

Fig. 13 Approximation of forest-covertype learning curve

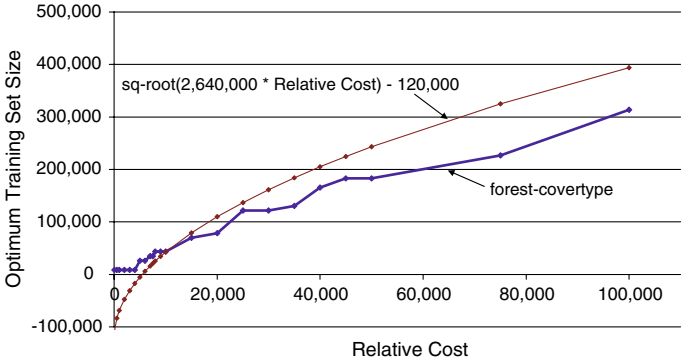

Fig. 14 Comparison of optimal curves: actual and derived

5 Progressive sampling

Section 4 demonstrated that one can improve total classifier utility by carefully selecting the training set size. However, given that we assume that payment for training data must be made when the data is acquired, to be of practical use a strategy must identify the number of training examples to acquire without acquiring more than this number of examples. One way to accomplish this is by using a progressive sampling strategy.

5.1 Progressive sampling methodology

The general outline of our progressive sampling strategy is simple. You begin with some initial amount of training data and then, iteratively, build a classifier, evaluate its performance and, based on those results, determine how much additional training data, if any, to acquire. In this article we consider relatively simple progressive sampling strategies. Our stopping strategy is quite simple: we stop obtaining training examples after the first observed increase in total cost. This guarantees that we will not achieve the optimum training set size since,

at minimum, there will be one better training set size (i.e., the one observed *before* the increase). If the accuracy of the learning curve is non-decreasing then this stopping condition will lead to a training set size that is close to optimal. While Fig. 1 demonstrates that our learning curves are not always non-decreasing, which can lead to "premature stopping," the results in this section will show that this has only a modest impact on our ability to find the optimal-utility classifier. We could also eliminate part of this problem of premature stopping by employing the *t*-test "filtering" method described in Sect. 4.1 to remove points on the learning curve that may not reflect statistically significant variations in classifier performance.

The next decision for a progressive sampling strategy is how much additional training data to acquire at each iteration. We evaluate two very simple, non-adaptive sampling schedules. Our first progressive sampling schedule utilizes the uniform sampling schedule described in Sect. 3.1. Our second progressive sampling strategy uses a geometric sampling schedule, where the training set size doubles each iteration. This geometric sampling scheme is motivated by previous work on progressive sampling, which shows that, given certain assumptions, this schedule is asymptotically optimal (Provost et al. 1999). Although these assumptions do not hold in our case due to the cost of training examples, the geometric sampling scheme nonetheless provides a valuable alternative to the "uniform" progressive sampling strategy. Note that as before, multiple runs are utilized to gain more reliable estimates of the accuracy for a given training set size.

Two other strategies are employed for comparison purposes. In order to determine how close the uniform and geometric progressive sampling strategies come to the optimum possible performance, we provide the results for the "optimal strategy" that always selects the optimum training set size, from the ones evaluated, using the data provided in Sect. 4. This strategy is not "fooled" by any temporary decreases in accuracy present in the learning curves. We also provide the results for a "straw man" strategy, which always uses all of the potentially available training data. The straw man strategy is used to quantify the benefits of considering the training data cost and the cost of model induction when building a classifier.

The remainder of this section follows the format of Sect. 4, except that in this section all of our results are based on the progressive sampling strategies. In Sect. 5.2 we consider the performance of the progressive sampling strategies when the cost of cases and error costs are considered and in Sect. 5.3 we extend this analysis to include the impact of the modeling costs, in terms of the CPU time required to build the classifier.

5.2 Progressive sampling strategy when there is a cost of cases

This section compares the results for the uniform and geometric progressive sampling strategies to the optimal and straw man strategies. In this section the cost of generating the model is not considered. Figure 15 presents the detailed results for the forest-covertype data set. We see that the straw man

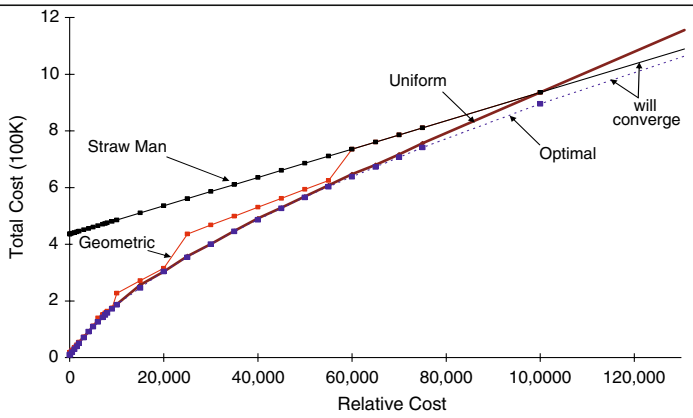

Fig. 15 Comparison of progressive sampling strategies for forest-covertype

strategy of using all of the training data independent of the relative cost initially leads to very poor results, which demonstrates the advantage of utilizing a progressive sampling strategy. The uniform strategy outperforms the geometric sampling strategy up until a relative cost of 100,000, because it samples more frequently that the geometric sampling strategy. Ultimately the uniform strategy performs worse than all other strategies because it stops prematurely, due to a temporary increase in total cost (the geometric sampling scheme is less susceptible to this problem since the distance between sample sizes is greater). At around a relative cost of 200,000 (not shown in Fig. 15) all strategies except for the uniform sampling strategy converge, since they properly determine that the optimum strategy is to acquire all potentially available training data.

Tables 2 and 3 compare the performance of the progressive sampling strategies for the large data sets (the names for the forest-covertype and census-income data set are abbreviated). The results in Table 2 indicate that the uniform sampling strategy generally is within 10% of the optimum performance. The only time this is consistently not the case is for very low relative costs ($\leq$1000), because then the best strategy is to acquire almost no training data—and in that situation the strategy of stopping *after* the first observed increase in training set size results in a relatively large penalty. The geometric sampling strategy generally performs worse, as can be seen by the last set of columns, since the values in those columns are generally positive. The cases where the geometric sampling strategy performs better is mainly due to the fact that the uniform sampling strategy stops prematurely, due to statistical variations in the performance of the learning algorithm. Table 3 provides a comparison with the straw man strategy, which always uses the maximum amount of potentially available training data. Consistent with the results presented in Fig. 15, we see that for low cost ratios the straw man strategy performs poorly. The straw man performs better as the relative cost increases and in quite a few cases outperforms the uniform strategy, which often stops prematurely due to temporary increases in total cost.

Table 2 Comparison of progressive sampling strategies for large data sets

Relative cost	Increase in total cost											
	Uniform vs. optimum				Geometric vs. optimum				Geometric vs. uniform			
	Covtype (%)	Income (%)	Protein (%)	Physics (%)	Covtype (%)	Income (%)	Protein (%)	Physics (%)	Covtype (%)	Income (%)	Protein (%)	Physics (%)
100	78.9	3.9	203.2	4.2	78.9	3.9	203.2	51.5	0.0	0.0	0.0	45.4
1,000	22.6	5.8	24.2	0.4	22.6	5.8	24.2	14.2	0.0	0.0	0.0	13.8
5,000	2.3	1.1	10.7	1.8	2.3	16.1	191.9	1.8	0.0	14.8	163.7	0.0
10,000	1.6	1.3	4.6	2.4	22.4	4.3	102.2	1.9	20.5	2.9	93.3	−0.6
25,000	1.2	1.4	9.8	4.0	23.2	19.3	47.0	1.1	21.8	17.7	33.9	−2.8
50,000	0.7	0.7	8.9	5.4	5.0	7.2	24.5	0.4	4.3	6.4	14.4	−4.7
75,000	2.1	0.2	6.1	6.1	9.4	3.5	16.2	0.4	7.2	3.3	9.5	−5.4
100,000	4.5	0.3	4.6	6.5	4.5	1.9	40.4	0.4	0.0	1.7	34.2	−5.7
200,000	16.4	0.9	2.4	7.0	1.0	2.7	15.2	0.4	−13.2	1.8	12.6	−6.2
250,000	5.6	0.9	1.9	7.1	1.0	1.5	10.1	0.4	−4.4	0.5	8.0	−6.3
500,000	8.8	3.2	2.7	7.3	0.8	2.2	12.4	0.4	−7.3	−1.0	9.5	−6.5
1,000,000	11.0	5.5	8.6	7.4	0.7	0.7	5.4	0.4	−9.2	−4.5	−2.9	−6.6

Table 3 Straw man versus progressive sampling strategies

Relative cost	Increase in total cost							
	Uniform vs. straw man				Geometric vs. straw man			
	Covtype (%)	Income (%)	Protein (%)	Physics (%)	Covtype (%)	Income (%)	Protein (%)	Physics (%)
100	2162.5	29810.5	18983.8	815.1	2162.5	29810.5	18983.8	529.4
1,000	1126.0	3182.3	8817.3	86.2	1126.0	3182.3	8817.3	63.6
5,000	313.4	692.5	2578.6	12.4	313.4	590.3	915.7	12.4
10,000	157.4	344.0	1562.0	3.0	113.7	331.5	759.7	3.6
25,000	56.6	134.0	703.8	−2.8	28.6	98.9	500.5	0.0
50,000	20.6	63.3	375.6	−4.7	15.6	53.5	315.7	0.0
75,000	7.2	39.6	259.5	−5.4	0.0	35.1	228.3	0.0
100,000	0.0	27.3	196.3	−5.7	0.0	25.2	120.9	0.0
200,000	−13.2	9.2	94.2	−6.2	0.0	7.2	72.6	0.0
250,000	−4.4	5.9	72.7	−6.3	0.0	5.3	59.9	0.0
500,000	−7.3	−1.0	28.3	−6.5	0.0	0.0	17.2	0.0
1,000,000	−9.2	−4.5	5.5	−6.6	0.0	0.0	8.6	0.0

In summary, we conclude that the uniform and geometric progressive sampling strategies are fairly successful at finding a near-optimal classifier and certainly outperform the strategy of always acquiring the maximum amount of potentially available training data. Improvements in the progressive sampling strategy are possible and these are discussed in Sect. 7.

5.3 Progressive sampling when the cost of model induction is considered

This section extends the analysis provided in the previous section by also considering the cost, in terms of CPU time, of inducing the classification model. In the interest of space we limit our analysis to the forest-covertype data set, although the same pattern of behavior is found for the other data sets. The results in Fig. 16 show the behavior of the three progressive sampling strategies given two different CPU cost factors.

The basic patterns we see are the same for both figures, although the curves intersect at different points. We begin by explaining the observed behavior. This explanation is supported by the data used to generate these figures, which includes the training set size selected for each relative cost value as well as the three cost components (cost of training data, cost of errors, and cost of building the model). The underlying data is not provided here due to space considerations.

Let us begin with the uniform and geometric progressive sampling strategies. The strategies overlap for relative costs below threshold T1. At this point the CPU cost dominates the error cost, so both strategies behave identically and minimize the CPU cost by acquiring the minimum amount of training data possible. In the interval from T1 to T4 the CPU cost no longer totally dominates the error cost, but is still a significant factor. In this interval the uniform sampling strategy outperforms the geometric sampling strategy, because it acquires data in smaller chunks, allowing it to more effectively trade off increased training set size—and the resulting increase in CPU time—with improved accuracy. However, this occurs only for low cost ratios because learning curves are most steep at the start. Finally, after point T4 the error cost component begins to dominate and the sampling strategies then need to acquire large amounts of training data in order to perform well. In this situation the geometric sampling strategy is better than the uniform strategy since it requires fewer samples and hence less cumulative CPU time to reach a similar training set size. Note that in both figures the uniform and geometric sampling strategies form a triangle, with the triangle extending from T1 to T4. The triangle spans a larger range of relative cost for the larger CPU cost factor (10,000 vs. 1,000), since then it takes longer for the error cost component to dominate.

Now let us turn to the straw man method, which uses all available training data regardless of the relative cost. This means that the number of errors and CPU time required to build the model are constant and therefore so are all of the terms in Eq. (3) for computing total cost. The observed slope for the straw man method is therefore completely due to the increasing penalty

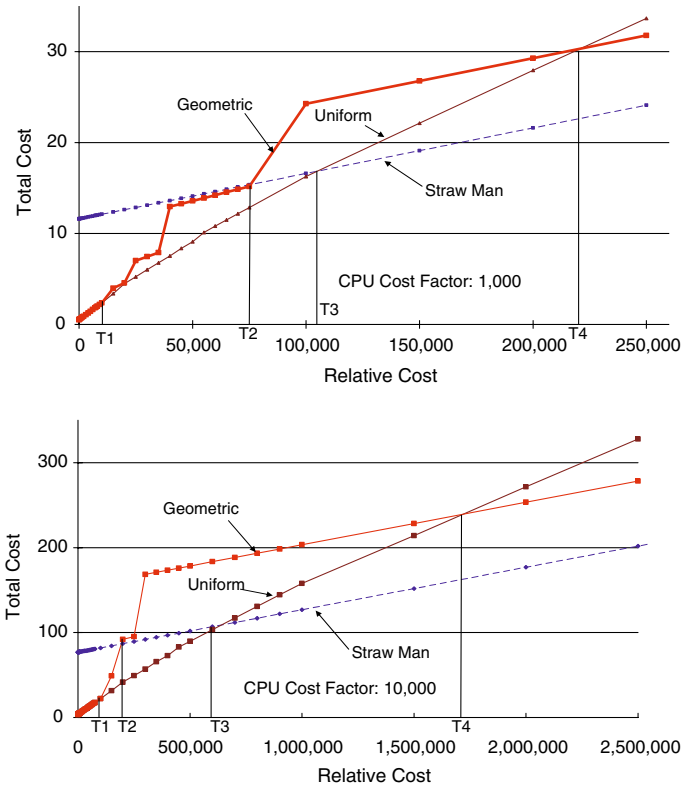

Fig. 16 Comparison of progressive sampling stratgies for different CPU cost factors

for errors that the increasing relative cost imposes. As one would expect, the relationship between relative cost and total cost is therefore linear. Both figures show that the straw man method does poorly at the beginning (due to its large CPU cost) but that it eventually outperforms the geometric and then uniform sampling strategies (at points T2 and T3, respectively). This occurs because the straw man strategy is much more efficient with respect to CPU time when all of the potentially available training data is acquired (i.e., it tries only one sample size); thus, once the cost of errors reaches a certain point the straw man strategy outperforms the other strategies.

We focus on the range of x-values in Fig. 16 because that is where the most interesting behavior occurs and where the various strategies intersect. If the relative costs are allowed to increase dramatically, then all three strategies will eventually intersect, if they acquire all available training data when the error cost dominates. However, this does not occur because, as described in Sect. 4.2, the uniform sampling strategy stops acquiring data prematurely.

This analysis demonstrates some of the patterns and behaviors that can occur when trying to optimize the sum of the training data cost, model induction cost, and error cost. In situations where all factors are significant, important choices

need to be made. Figure 16 and the analysis just provided clearly show under what circumstances these costs are important and that in many situations our progressive sampling strategies will outperform the strategy of always acquiring all possible training data.

6 Related work

In this section we discuss how our research relates to existing work. As mentioned earlier, the main connections are to work in progressive sampling and active learning. Work on progressive sampling has focused on efficiently finding the training set size where the learning curve reaches a plateau (Provost et al. 1999). The motivation for that work was to reduce the cost of computation by limiting the amount of training data, while still achieving the best predictive performance possible. Our research can be viewed as a generalization of that work, in that we consider the cost of the training data in addition to the cost of generating the model. The progressive sampling work focused on efficiently identifying convergence—the training set size beyond which the generalization accuracy ceases to increase. The corresponding task in our research is to find the minimum of the utility curve. While the work on progressive sampling showed that a geometric sampling schedule is asymptotically optimal, our results show that it often does not perform as well as a uniform sampling schedule. The observed difference is due to the fact that the training data cost is directly proportional to the training set size. However, if the time of computation were to dominate the data acquisition cost, then we would expect a similar result—the geometric sampling scheme to outperform the uniform sampling scheme. This is exactly what we observe in Sect. 4.3.

Our work can also be compared to work on active learning (Cohn et al. 1994). One main difference is that work on active learning focuses on addressing the cost of measuring features (Veeramachaneni and Avesani 2003) and labeling examples (Cohn et al. 1994), which Turney (2000) refers to as the "cost of tests" and the "costs of teacher," respectively, while we focus on the cost of acquiring complete examples (the "cost of cases"). In our setting the only choice involves *when* to stop acquiring training data, whereas in active learning one also has to decide *what* feature to measure or example to label next. That is, in most work on active learning the choices are typically ranked, thus providing an ordering, but no threshold for determining when to stop acquiring information. Other methods (Kapoor and Greiner 2005) assume a budget and stop the active learning once the budget is consumed. Given this general lack of a stopping criterion, the types of analyses we employ could easily be extended to decide when active learning should stop. The only difference would be that the experiments that generate the learning curve data would need to use an active learning method for selecting the training data. The closest match to our research from the active learning community involves work where the marginal utility of each example is estimated and this is used to determine how many examples to label (Melville et al. 2005).

Weiss and Provost (2003) also consider the cost of cases, but only in a limited way. The assumption in that work is that the cost of cases limits the amount of training data that can be procured. The decision to be made in that scenario is what class distribution should be generated for the training set in order to maximize classifier utility. That work also employed a progressive sampling strategy, although the goal was to identify the optimal class distribution for learning given a fixed training set size, whereas the goal in this article is to identify the optimal training set size.

7 Conclusions and future work

This article analyzed the impact that data acquisition costs and the cost of generating a classifier have on total classifier utility. We introduced a variety of charts to help visualize the relationship between these costs and total cost and also identified the optimal training set size for twelve data sets, for a variety of cost information. We then described and analyzed the performance of two simple progressive sampling strategies and showed that they perform substantially better than the strategy of acquiring all potentially available training data and that they also achieve near-optimal performance. This work also provides a good example of Utility-Based Data Mining, where utility factors from various stages of the data mining process are jointly considered in order to generate a classifier that maximizes the utility of the entire data mining process.

This research can help practitioners who are faced with a classification task where training data is acquired at a cost. It is not just the methods that we developed that are relevant, but the focus on the entire data mining process that we advocate. First, the practitioner needs to examine the entire classification process carefully and gather whatever cost information is available. Even if this cost information is not precise, it should be estimated. This includes all of the cost factors described in Sect. 2 as well as the expected size of the score set. The practitioner can then start to incrementally acquire training data and build and evaluate the total utility of the classification process at each iteration. This is essentially progressive sampling, even if it is not done using one of the sampling schedules we suggested. Rather than just generating a single utility curve, the practitioner can easily generate multiple curves based on different estimates of the cost factors. Note that this is trivial to do since the generation of each utility curve is quite simple and requires very little computation. If different (but reasonable) estimates of the cost factors yield the same decision (i.e., to stop or to continue acquiring data), then the next step is clear. If different estimates yield different decisions, then the practitioner must ultimately decide which estimate is most reasonable. However, by using this process the practitioner need only make this decision when it would lead to a different action.

The work described in this article can be extended in several ways. The most straightforward extension involves analyzing utility models that are more sophisticated that the one described in Sect. 2. We believe that work in this

direction would help address the ultimate goal of Utility-Based Data Mining, which is to maximize the utility of the entire data mining process.

Because the analyses are all driven by the learning curves, any method for improving the quality of the learning curves (i.e., smoothness, monotonicity) would improve the quality of our results, especially the effectiveness of the progressive sampling strategies. There are a number of possible methods for improving the learning curves. One could study the impact that different learning algorithms have on the learning curves in order to determine if some methods generate better behaved learning curves. Given that decision trees are known to be sensitive to small changes in data (Li and Belford 2002), this suggestion could lead to better learning curves. Also, one could try to use statistical methods to improve the quality of learning curves, as we did in Sect. 4.1.

Another possible extension involves the use of a more intelligent, adaptive, progressive sampling scheme. One strategy might be to reduce the "gap" between successive training set sizes as the slope of the utility curve approaches zero, so that one can get closer to the true optimum training set size, with minimal effort (the gaps would be kept large when the magnitude of the slope is large). Another approach would be to try to fit a function to a partial learning curve and then analytically determine the optimal training set size, as suggested by our work in Sect. 4.4. This seems like an area worthy of future research.

Finally, one could extend our utility curves to handle the case where there is uncertainty in the cost factors. A fairly straightforward method for doing this would be to allow the user to specify a range and distribution for each cost factor (i.e., it ranges between 10 and 50 and is uniformly distributed) and then consider this when generating the utility curve. It should be pointed out, however, that although misclassification cost has been studied extensively in the context of cost-sensitive learning, this technique, to our knowledge, has not been applied in that area.

References

Berry M, Linoff G (2004) Data mining techniques for marketing, sales, and customer relationship management. Wiley Publishing, Indianapolis, IN

Breiman L, Friedman JH, Olshen RA, Stone CJ (1983) Classification and regression trees. Wadsworth

Caruna R, Joachims T, Backstrom L (2004) KDD-CUP 2004: results and analysis. SIGKDD Explor 6(2):95–108

Cohn D, Atlas L, Ladner R (1994) Improving generalization with active learning. Mach Learn 15(2):201–221

Drummond C, Holte R (2006) Cost curves: an improved method for visualizing classifier performance. Mach Learn 65(1):95–130

Elkan C (2001) The foundations of cost-sensitive learning. In: Proceedings of the Seventeenth International Joint Conference on artificial intelligence, Seattle, WA, pp 973–978

Esposito F, Malerba D, Semeraro G (1997) A comparative analysis of methods for pruning decision trees. IEEE Trans Pattern Anal Mach Intell 19(5):476–491

Fayyad U, Piatetsky-Shapiro G, Smyth P (1996) From data mining to knowledge discovery in databases.. AI Mag 17:37–54

Greiner R, Grove A, Roth D (2002) Learning cost-sensitive active classifiers. Artif Intell 39: 137–174

Hettich S, Bay SD (1999) The UCI KDD archive [http://kdd.ics.uci.edu]. University of California, Dept. of Information and Computer Science, Irvine, CA

Hoehn B, Southey F, Holte R, Bulitko V (2005) Effective short-term opponent exploitation in simplified poker. In: Proceedings of the Twentieth National Conference on artificial intelligence, Pittsburgh, PA, pp 783–788

Kapoor A, Greiner R (2005) Learning and classifying under hard budgets. In: Proceedings of the Sixteenth European Conference on machine learning, Porto, Portugal, pp 170–181

Lewis D, Catlett J (1994) Heterogeneous uncertainty sampling for supervised learning. In: Proceedings of the Eleventh International Conference on machine learning, New Brunswick, NJ, pp 148–156

Li R, Belford G (2002) Instability of decision tree classification algorithms. In: Proceedings of the Eighth ACM SIGKDD International Conference on knowledge discovery and data mining, Edmonton, Canada, pp 570–575

Martin JK, Hirschberg DS (1996) On the complexity of learning decision trees. In: Proceedings of the fourth International Symposium on artificial intelligence and mathematics, Fort Lauderdale, Florida

Melville P, Saar-Tsechansky M, Provost F, Mooney R (2005) Economical active-feature value acquisition through expected utility estimation. In: Proceedings of the First International Workshop on Utility-Based Data Mining, Chicago, IL, pp 10–16

Newman DJ, Hettich S, Blake CL, Merz CJ (1998) UCI Repository of machine learning databases [http://www.ics.uci.edu/∼mlearn/MLRepository.html]. University of California, Department of Information and Computer Science, Irvine, CA

Provost F, Fawcett T (2001) Robust classification for imprecise environments. Mach Learn 42:203–231

Provost F, Jensen D, Oates T (1999) Efficient progressive sampling. In: Proceedings of the Fifth International Conference on knowledge discovery and data mining, San Diego, CA, pp 23–32

Quinlan JR (1993) C4.5: programs for machine learning. Morgan Kaufmann, San Mateo, CA

Snedecor GW, Cochran WG (1989) Statistical methods. Iowa State University Press, Ames, OH

Turney P (2000) Types of cost in inductive concept learning. In: Workshop on Cost-Sensitive Learning at the Seventeenth International Conference on machine learning, Stanford, CA

Van Rijsbergen CJ (1979) Information retrieval, 2nd edn. Butterworth, London

Veeramachaneni S, Avesani P (2003) Active sampling for feature selection. In: Proceedings of the Third IEEE International Conference on data mining, Melbourne, Florida, pp 665–668

Weiss GM, Provost F (2003) Learning when training data are costly: the effect of class distribution on tree induction. J Artif Intell Res 19:315–354

Weiss GM, Saar-Tsechansky M, Zadrozny B (2005) Report on UBDM-05: workshop on utility-based data mining. SIGKDD Explor 17(2):145–147

Zadrozny B, Weiss GM, Saar-Tsechasnky M (2006) UBDM-2006: utility-based data mining workshop report. SIGKDD Explor 8(2):98–101