Active Learning with Direct Query Construction

Charles X. Ling
Department of Computer Science
The University of Western Ontario
London, Ontario N6A 5B7, Canada
cling@csd.uwo.ca

Jun Du
Department of Computer Science
The University of Western Ontario
London, Ontario N6A 5B7, Canada
jdu42@csd.uwo.ca

ABSTRACT

Active learning may hold the key for solving the data scarcity problem in supervised learning, i.e., the lack of labeled data. Indeed, labeling data is a costly process, yet an active learner may request labels of only selected instances, thus reducing labeling work dramatically. Most previous works of active learning are, however, pool-based; that is, a pool of unlabelled examples is given and the learner can only select examples from the pool to query for their labels. This type of active learning has several weaknesses. In this paper we propose novel active learning algorithms that construct examples directly to query for labels. We study both a specific active learner based on the decision tree algorithm, and a general active learner that can work with any base learning algorithm. As there is no restriction on what examples to be queried, our methods are shown to often query fewer examples to reduce the predictive error quickly. This casts doubt on the usefulness of the pool in pool-based active learning. Nevertheless, our methods can be easily adapted to work with a given pool of unlabeled examples.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning—*induction, knowledge acquisition*

General Terms

Algorithms

Keywords

Active learning, classification, supervised learning

1. INTRODUCTION

Active learning is very important in classification tasks in machine learning and data mining, as it may hold the key

for solving the data scarcity problem, i.e., the lack of labeled data.[1] Indeed, labeling data is a very costly process. For example, in webpage (or image, movie, news articles, face) classification, it is crucial to have a set of correctly labeled examples for supervised learning, yet the labels are often given by human experts, and thus, it is a costly and time-consuming process. Active learning, on the other hand, is able to actively request labels of a small number of examples, and thus, reducing the labeling cost significantly. However, most previous work of active learning is "pool-based"; that is, a pool of unlabeled examples is given, and the learner can only select examples from the pool to query for their labels. (See a review of pool-based active learning in Section 2).

The pool-based active learning has several weaknesses. First of all, the computational time of most previous pool-based learning algorithms is high. This is because most pool-based methods must evaluate each example in the pool to see which one is most "uncertain" or "informative" (see Section 2). Sometimes new models for each additional example in the pool are built. If the pool is relatively large, the time for deciding which example in the pool to be selected is often quite significant. Second, as examples to be selected must be from the pool, they can be quite limited, especially if the pool is small, and thus, they may not be effective in reducing the error rate rapidly. Third, the pool of unlabeled examples themselves must be collected first, which can be a time-consuming process.

In this paper we propose novel Active learners with Direct Query construction (called ADQ for short) and query for their labels. This is also called "membership query" studied previously but mostly in theoretical setting [1]. More specifically, we first study a specific active learner based on the decision tree algorithm (called Tree-ADQ) to construct its queries. Then we propose a general active learner that can work with any base learning algorithm (called wrapper-ADQ, as it is like a wrapper enclosing any base learning algorithm). As there is no restriction on what examples to be queried, our ADQ algorithms are shown to often query fewer examples to reduce the predictive error quickly. Furthermore, our ADQ can also be easily adapted to work with a given pool of unlabeled examples. Our ADQ algorithms are also shown to be more time-efficient than the traditional

---

[1]Another promising research is semi-supervised learning, such as co-training.

pool-based methods.

Though our methods of direct query construction can be regarded as a special case of the pool-based method, when the pool is assigned to contain all of the rest of the examples not in the training set, such a method is often dreadfully inefficient. This is because with a large number of attributes, the size of all possible examples is exponentially large (to the number of attributes). For example, if the training set contains web pages on politics, then the constructed pool would include all other possible web pages (meaningful or meaningless ones), and this number is huge (if the total number of words in the page is bounded, the number of possible pages is finite but huge). Thus, the traditional pool-based methods would be extremely inefficient to choose which example to label. Our ADQ can construct examples directly to query, and does not need a pool of unlabeled examples. Another potential argument against our work is that the constructed examples may not be valid. For example, in learning handwritten digits, the learner may construct an image dissimilar to any digit. This can be easily handled. In binary classification, the label of such invalid examples can simply be labeled as negative. In multi-class cases, a new class, such as "invalid", can be created for labeling all invalid examples. However, sometimes the examples and feature values are different. For example, webpages are examples, but are transferred to feature values (a vector of word counts) for the learning algorithms. The active learner will only construct new feature values (vectors of word counts), which may be difficult to be labeled by human. We will study this issue in our future research.

The rest of the paper is organized as follows. Section 2 reviews previous works on active learning. Section 3 describes our tree-based active learner Tree-ADQ, and Section 4 describes our wrapper-based active learner Wrapper-ADQ. In both cases experimental comparisons are conducted to compare ADQs with the traditional pool-based methods. Section 6 presents conclusions and future work.

2. REVIEW OF PREVIOUS WORKS

The most popular type of active learning is called "pool-based" active learning. A pool of unlabeled examples is given, and the learner can choose which one(s) to label during learning. Many works have been published in recent years on pool-based active learning, including, for instance, [18, 20, 15, 16, 2, 9, 10].[2] In these previous works, each example in the pool is evaluated, sometimes extensively evaluated by building new learning models with each example added in the current training set (e.g., query by committee), to decide which example is most uncertain [15, 10], or most informative if the label is known [20]. As there is no restriction on what examples to query, our ADQ can often query fewer examples to reduce the error rate quickly. Also, pool-based methods are more time consuming, especially when the pool is relatively large, than our active learners with

[2]We have only included several typical works published in recent years. See [2] for a good review of active learning approaches.

direct query construction (ADQ). See Sections 3 and 4 for details.

Active learning with "membership queries" can construct examples and request labels; however, as far as we know, very few works have been published. Some are theoretical study in the PAC learning framework (e.g., [1]). [5] proposes a version-space search algorithm in neural networks for query construction but the algorithm works only for simple concepts. As we discussed in Section 1, although it can be regarded as pool-based learning when the pool contains all possible examples not in the labeled training set, such an approach is very inefficient. Last, in the "stream-based" active learning, the learner is given a stream of unlabeled examples, and the learner must decide, for each example in the stream, if or not to request its label ([6, 17, 11]). This approach can be regarded as an online version of the pool-based active learning.

Some recent works on active learning turn to a new direction of feature-value acquisition at cost during learning. In the work of ([8, 7]), a fixed budget is given, and the cost of feature value acquisition cannot exceed this hard budget.

3. TREE-ADQ

In this section we propose a specific active learning algorithm with direct query construction based on decision trees (thus, it is called Tree-ADQ). Though the basic idea can be modified and applied to other base learners, Tree-ADQ is applicable to the decision tree learning algorithm specifically; that is, a different method might be needed if naive Bayes is used as the base learner. In Section 4 we will describe a generic method that can be applied to any base learning algorithm.

3.1 The Algorithm

The general idea of Tree-ADQ is straightforward: it tries to find the most uncertain examples from the current decision tree (or other learned model), and request labels for those examples. The most uncertain examples are those whose predicted probability, either for the positive or negative examples, is most uncertain, or close to 50%. For example, a predicted positive example (or negative example) with 50% probability is most uncertain, and with 100% is most certain. Given a decision tree constructed from the labeled examples, we can find, for each leaf, its predicted label and its probability, and find the most uncertain leaves.

More specifically, Tree-ADQ consists of three general steps.

- Step 1: A decision tree is constructed using the currently available set of labeled examples.

- Step 2: The uncertainty of each leaf is determined, and examples are constructed in the most uncertain leaves.

- Step 3: Those most uncertain examples are queried, and after their labels are obtained, they are added into the labeled training set. Go to Step 1.

For each iteration, the predictive error rate on the (separate) test examples is monitored and used to plot the error curves (see Section 3.2).

Details of each step above are described below.

In Step 1, the standard decision learning algorithm C4.5 [14], implemented as j48 in Weka [19], is used to construct a pruned decision tree from the current labeled examples.

In Step 2, the uncertainty of each leaf in the tree is determined by its error rate upper bound. More specifically, the error rate of the each leaf is first calculated as the ratio of the incorrectly classified examples to the total number of examples in the leaf. However, the ratio itself is often not a good indicator of how uncertain the leaf is, especially when the leaf contains a small number of examples. For example, if a leaf contains only 5 examples and 1 of them belongs to a minority class, then 20% is not a reliable estimate of the true error rate of the leaf, as the number of examples (5) is too small. Statistically we can put an error bound on the ratio, and obtain a pessimistic estimation of the true error. We use the normal distribution to approximate the error rate distribution, and use 95% confidence to estimate the error bound. Thus, the error rate upper bound is calculated as follows to represent the pessimistic uncertainty of the leaf [12]:

$$error(h) + 1.96\sqrt{\frac{(error(h)(1 - error(h)))}{n}} \qquad (1)$$

where $error(h)$ is the error rate of each leaf, n is the total number of examples in the leaf, and the constant 1.96 reflects a 95% confidence interval in the normal distribution. Then, new examples can be constructed in the most uncertain leaves. To increase the variety of the newly constructed examples from different uncertain leaves, new examples are sampled from leaves with the sampling probability proportional to their error rate upper bounds. This way, more uncertain leaves have high probabilities to be sampled. When a leaf is sampled, a new example is constructed in this leaf. Its attribute values are determined as follows: for attributes appearing on the path from the root to the leaf, their values are determined by the attribute values on the path. Other attributes (not on the path) are assigned random (valid) attribute values. Clearly, the time complexity of the construction is only linear to the tree depth, and the number of attributes (thus, linear to the number of attributes). The Tree-ADQ is thus time-efficient.

In step 3, the constructed examples are queried to obtain their labels, and then included in the labeled training set for the next iteration.

As we mentioned in Section 1, Tree-ADQ can also be adapted easily to work with a pool of given unlabeled examples effectively (without evaluating each example in the pool as in the traditional pool-based approaches). We use the following simple strategy to modify Tree-ADQ when a pool is given. In each iteration, the most uncertain examples are still directly constructed following the Steps 1 and 2 of Tree-ADQ described above. Instead of querying the label of these examples (as the Step 3 in Tree-ADQ without the pool), we calculate the Euclidean distance between the constructed example and all examples in the pool, and choose the closest (or most similar) one in the pool to be queried.

	# Attributes	# Examples	Class dist.
breast-cancer	9	277	196/81
breast-w	9	699	458/241
colic	22	368	232/136
credit-a	15	690	307/383
credit-g	20	1000	700/300
diabetes	9	768	500/268
heart-statlog	13	270	150/120
hepatitis	19	155	32/123
sick	29	3772	3541/231
sonar	60	208	97/111
tic-tac-toe	10	958	332/626
vote	16	435	267/168

Table 1: Datasets used in the experiments

This way, examples to be queried are selected from the pool, and they are similar to the queries constructed directly by Tree-ADQ. We call such a method Tree-ADQ-p.

3.2 Experimental Comparisons

In this subsection we experimentally compare Tree-ADQ and Tree-ADQ-p with the traditional pool-based active learning algorithm. The traditional pool-based active learner selects the most uncertain examples (using the decision tree) from the pool to be labeled.

3.2.1 Datasets and Experimental Setup

We conduct experiments to compare Tree-ADQ algorithms with the traditional pool-based active learning using 12 real-world datasets from the UCI Machine Learning Repository [3]. These datasets have discrete attributes and binary class without missing values. Information on these datasets is tabulated in Table 1. Each dataset is randomly split into three (3) disjoint parts: 20% as the initial labeled training examples, 20% as the testing examples, and the rest (60%) as the "unlabeled" examples (examples given in the pool).

One problem when using the ADQ algorithms on the UCI datasets is how queries constructed by ADQ are labeled if such queries are not in the original dataset. In such cases the labels are unknown. We use the following approach to solve this problem. We first construct a pruned decision tree (using J48) based on the original, whole dataset, and designate this tree as an approximate target function. Then, this tree is used to answer queries constructed by Tree-ADQ (and other active learning algorithms with direct query construction proposed in this paper). Clearly, this tree is the closest we can get from a given dataset when the true target function is unknown.

Both Tree-ADQ, Tree-ADQ-p, and the traditional pool-based active learner (simply called pool) are implemented in Weka [19] with J48 (i.e., C4.5) as the decision tree algorithm. For traditional pool-based active learner, a decision tree is constructed based on the current labeled dataset. Then each example in the pool is evaluated by the tree, and the most pessimistic error rate (see Section 3.1) is calculated. The most uncertain examples are then selected to query for their

	pool	Tree-ADQ-p
Tree-ADQ	11/0/1	11/0/1
Tree-ADQ-p	2/6/4	

Table 2: Summary of the t-test on the average error rates.

labels, and added into the training set. For each iteration, 1, 5 or 10 examples are queried and added into the training set, depending on the size the dataset.

The experiment is repeated for 100 times for each dataset, and the average predictive error rate on the separate test sets and time of running different algorithms are plotted and recorded.

3.2.2 Experiment results

Figure 1 plots the curves for the predictive error rate on the test sets of the 12 UCI datasets for the three methods (Tree-ADQ, Tree-ADQ-p, pool) in comparison. From these figures, we can see clearly that for most datasets, Tree-ADQ has the lowest predictive error rates on the test sets. Only in one dataset ("tic-tac-toe"), Tree-ADQ performs slightly worse. To quantitatively compare the learning curves (which is often difficult if one curve does not dominate another), we measure the actual values of the error rates in 10 equal-distance points on the x-axis. The 10 error rates of one curve are compared with the 10 error rates of another curve using the two-tailed, paired t-test with a 95% confidence level. The results are summarized in Table 2. Each entry in the table, $w/t/l$, means that the algorithm in the corresponding row wins in w datasets, ties in t datasets, and loses in l datasets, compared to the algorithm in the corresponding column.

From the table, we can see that Tree-ADQ is better than pool in 11 datasets, ties in 0 dataset, and loses only in 1 dataset ("tic-tac-toe"). This indicates clearly that Tree-ADQ produces lower predictive errors on the test sets compared to the traditional pool-based active learner in most datasets. We can also see that Tree-ADQ is much better than Tree-ADQ-p (wins in 11 datasets, ties in 0, and loses in 1). Both of these results seem to indicate limitations of the pool. Tree-ADQ is free to choose whatever examples best for reducing the predictive error rates, and this is a major advantage of the ADQ algorithms. Section 5 further demonstrates that the pool is actually putting a harmful limitation on the pool-based active learning. We will show that the larger the pool, the better the pool-based active learners perform. This casts some doubt on the common assumption of the pool in the traditional pool-based active learning.

From Table 2, We can also compare Tree-ADQ-p with the traditional pool-based active learner (pool), and see that Tree-ADQ-p is better than pool in 2 datasets, ties in 6 datasets, and loses in 4 datasets. This indicates that Tree-ADQ-p is comparable to (or only slightly worse than) the traditional pool-based active learner. This is expected as both methods choose examples from the same pool.

	Tree-ADQ	Tree-ADQ-p	pool
Time(s)	0.38	0.51	0.55

Table 3: Running time on "sick" for active learning algorithms in comparison.

The computer time used for these active learners in comparison is reported in Table 3 on the largest dataset (the "sick" dataset which has most examples). The computer used is a PC with an Intel Core 2 Quad Q6600 (2.67 GHz) CPU and 4G memory, and the computer time is reported in seconds. From Table 3, we can see that Tree-ADQ is most efficient, and Tree-ADQ-p is similar but still faster than the traditional pool-based active learner (pool).

4. WRAPPER-ADQ

In the previous section we described a specific ADQ based on the decision tree algorithm. Tree-ADQ is very efficient in constructing examples to reduce the error rate quickly but the algorithm only works on decision trees. In this section we propose a generic ADQ that can work with any base learning algorithms that can produce the probability estimation for the prediction. As it can "wrap" around any base learning algorithm, we call it Wrapper-ADQ.

4.1 The Algorithm

"Wrapper-ADQ" uses the standard hill climbing algorithm to search for the most uncertain examples outside a learning algorithm to query their labels. More specifically, it starts with a randomly generated new example. Then every attribute value of this example is changed to a different value once a time, and those new examples (with one attribute value difference) are evaluated by the base learning algorithm (which returns its predicted label and its uncertainty). The one that is most uncertain is retained (i.e., greedy hill climbing) and the process repeats, until the example is not altered from the last iteration. The final unaltered example is queried for the label and then added into the training set. Note that, though the greedy hill climbing may only find the "locally" most uncertain example instead of the "globe" one, it still works well because it increases the diversity of new examples constructed.

The Wrapper-ADQ does rely on accurate and more discriminating probability estimates of the prediction. If we use a single decision tree as the base learner, its probability estimates are not fine enough, as examples in the same leaf are assigned the same probability. Thus, we use an ensemble of decision trees as the base learner here. We use the basic process of bagging, as it has been shown to produce the tree-based classifier with accurate probability estimates [13]. More specifically, 100 decision trees are first constructed from the current labeled examples as in the random forest [4]. That is, a bootstrapping new training set is drawn from the original training set (bagging), and a decision tree is grown on the new training set using random attribute selection (for all attributes with non-negative entropy reduction). This way, a variety of different trees can
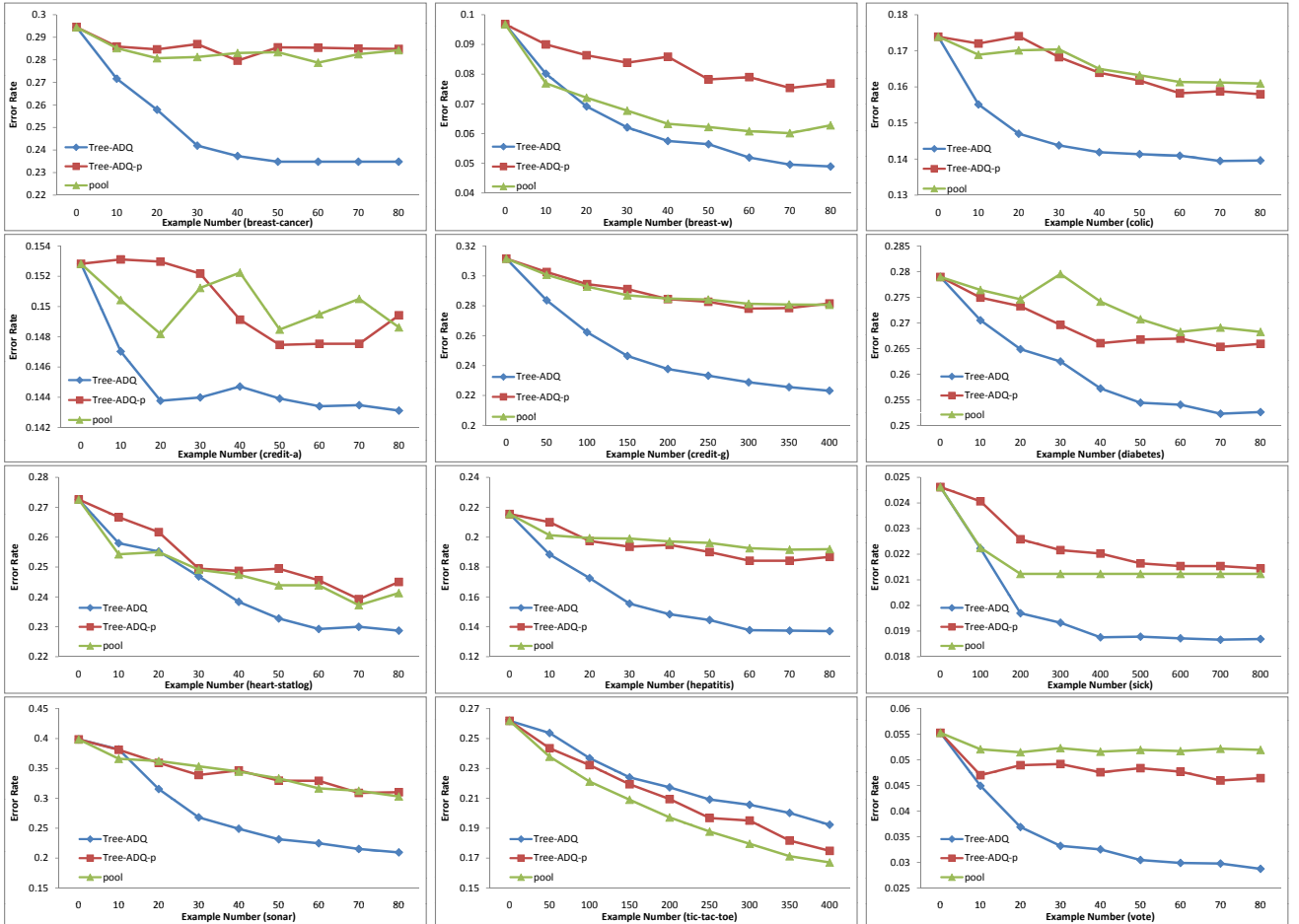
Figure 1: Comparing predictive error rates of Tree-ADQ, Tree-ADQ-p, and pool on the test sets. The lower the curve, the better.

be constructed. For each query evaluated in the hill climbing search, each tree's prediction (with its own uncertainty estimation; see Section 3.1) is combined to produce a final prediction and its uncertainty.

4.2 Experimental Comparisons

We use the same datasets and similar experiment setup to compare Wrapper-ADQ and pool-based active learning (called pool, which also uses an ensemble of 100 trees to choose the most uncertain examples from the pool; it is also called "query by committee" in active learning). Similarly, Wrapper-ADQ can be adapted easily to work with a pool of given labeled examples by first constructing the most uncertain examples, and then finding the one in the pool that is closest to them. Such a method is called Wrapper-ADQ-p. All of these three methods (Wrapper-ADQ, Wrapper-ADQ-p, pool) are compared and the results are reported in Figure 2 and Table 4 (with the same notation as in Table 2).

We can see that Wrapper-ADQ is better than pool in 10 datasets, ties in 2 datasets, and loses in 0 dataset. This indicates clearly that Wrapper-ADQ produces lower predictive errors on the test sets compared to the traditional pool-

	pool	Wrapper-ADQ-p
Wrapper-ADQ	10/2/0	11/1/0
Wrapper-ADQ-p	2/6/4	

Table 4: Summary of the t-test on the average error rates

	Wrapper-ADQ	Wrapper-ADQ-p	pool
Time(s)	95	97	108

Table 5: Running time on "sick" for active learning algorithms in comparison.

based active learner, which is still comparable to Wrapper-ADQ-p.

The computer time used for these wrapper-based active learning algorithms in comparison is reported in Table 5. We also draw the similar conclusion that Wrapper-ADQ is the most efficient, and Wrapper-ADQ-p still outperforms the traditional pool-based active learner (pool). The difference is not as large as the tree-based approaches (Table 3) because here, a lot of the time is spent on the hill-climbing search.
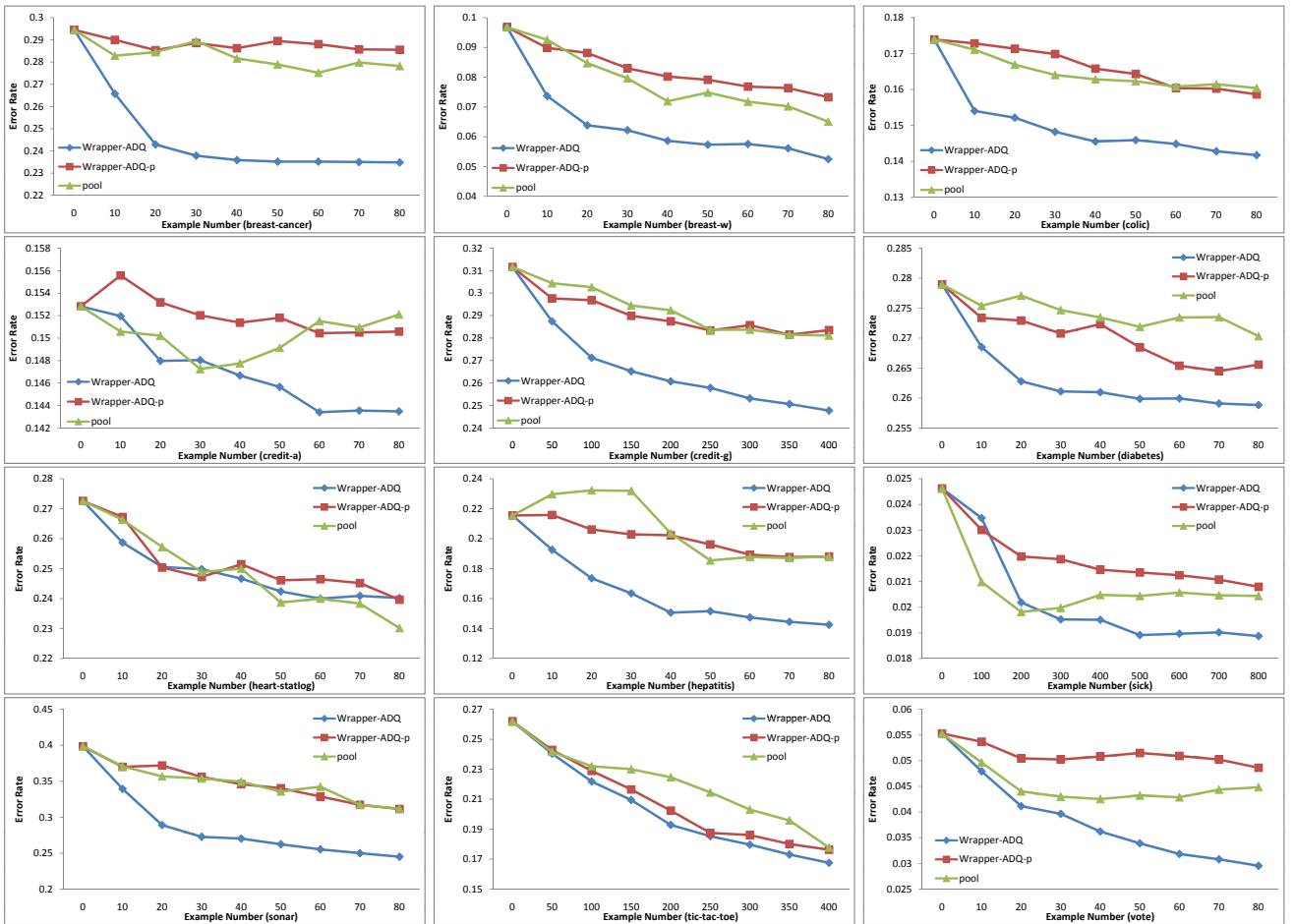
Figure 2: Comparing predictive error rates of Wrapper-ADQ, Wrapper-ADQ-p, and pool on the test sets. The lower the curve, the better.

	Wrapper-ADQ
Tree-ADQ	7/4/1

Table 6: Summary of the t-test on the average error rates

4.3 Comparing Tree-ADQ and Wrapper-ADQ

Table 6 (with the same notation as in Table 2) compares Tree-ADQ and Wrapper-ADQ on the 12 datasets used in our experiments. It shows that Tree-ADQ wins in 7 datasets, ties in 4 datasets, and loses in 1 dataset. Clearly, Tree-ADQ has a slight advantage over Wrapper-ADQ in terms of the error rate. However, Tree-ADQ is a specific active learner based on decision trees, while Wrapper-ADQ is generic; it can take any base learning algorithm as long as it returns probability estimates.

5. IS THE POOL REALLY NECESSARY?

Even though we have shown that the proposed active learning algorithms with direct query construction (ADQ) do not need a pool of given unlabeled examples to achieve lower or similar predictive errors, one might still argue that the pool can provide a useful "boundary" and distribution on what examples can be queried. We will show in this section that such a pool is unnecessary and even harmful to effective active learning.

More specifically, we will show that when the size of the pool incrementally increases, active learning algorithms, both ADQ with the pool and the traditional pool-based learner, work better. We will consider two cases. In the first case, the pools of various sizes consist of examples from the real data (i.e., a part of the UCI dataset). Thus, the distribution of the examples in the pools is the same as the training and test sets, and labels of examples in the pools are given by the original dataset. In the second case, the pools consist of examples artificially generated, and labels of the artificial examples are obtained from the decision tree built from the real data (as in Section 3.2.1).

5.1 Real Data Pools

We use the largest dataset "sick" in this experiment. The dataset is split into three disjoint subsets: 10% for the labeled training set, 10% for the test set, and 80% for the whole unlabeled set. The whole unlabeled set is larger in this

experiment as it has more real examples for us to manipulate different pools. We will construct four pools, or unlabeled sets, U_1, U_2, U_3 and U_4, provided to the active learners. More specifically, the whole unlabeled set is divided into four equal parts randomly. U_1 is equal to one quarter of the whole unlabeled set, U_2 is equal to U_1 plus another quarter of the whole unlabeled set, U_3 is U_2 plus another quarter, and U_4 is the whole unlabeled set. Thus, these unlabeled sets (pools) are all from the original real dataset with known labels. They increase in size, and $U_1 \subset U_2 \subset U_3 \subset U_4$.

Tree-ADQ with the pool (Tree-ADQ-p), Wrapper-ADQ with the pool (Wrapper-ADQ-p), and the traditional pool-based learning algorithm (pool) are applied with different unlabeled sets (U_1 to U_4). The results are shown in Figure 3. We can see that, overall, active learners with the large pools perform better. This is particularly evident in the traditional pool-based algorithm (pool). We can see that at the beginning of the query process, the algorithm with different pool sizes has a similar error rate. Then the difference starts to emerge. The algorithm with U_4, the largest pool, performs the best (the lowest error rate), with U_2 and U_3 performs similarly, and with U_1 performs the worst (the highest error rate). For Tree-ADQ-p and Wrapper-ADQ-p, the results are somewhat mixed but when more and more examples are queried, we can still see that algorithms with larger pools have slightly lower error rates in general. Similar results have been observed in other datasets as well.

5.2 Artificial Data Pools

In this experiment we use a smaller dataset "sonar". As "sonar" has the most attributes (60 attributes and 10 attribute values for each attribute), we can easily generate artificial examples not in the original dataset. This time, the original real dataset is split into three disjoint subsets: 20% for the labeled training set, 20% for the test set, and 60% for the initial pool of unlabeled set (called U_1). Artificial examples are randomly generated (with a uniform distribution on attribute values) and incrementally inserted into U_1, forming U_2, U_3, and U_4. The size of U_i is i times the size of U_1. Thus, U_1 is a part of the original dataset, but U_2 to U_4 include increasingly more artificial examples. We also have $U_1 \subset U_2 \subset U_3 \subset U_4$.

Similarly, Tree-ADQ with the pool (Tree-ADQ-p), Wrapper-ADQ with the pool (Wrapper-ADQ-p), and the traditional pool-based learning algorithm (pool) are applied with different unlabeled sets (U_1 to U_4). The results are shown in Figure 4. These results are somewhat mixed, but we can still see that algorithms with U_1, the smallest pool, perform the worst (the largest error rates). Algorithms with larger pools in general perform similarly and better than with U_1. Similar results have also been observed in other datasets.

These experiment results indicate that, indeed, the pool, especially when it is too small, limits the scope of examples to be queried for active learners. This casts doubt on the pool assumed in most previous work of active learning. This also further confirms the advantage of our ADQ algorithms (Tree-ADQ and Wrapper-ADQ) — they construct queries

directly without the restriction of the pool, and often reduce the predictive error rates more quickly, as shown in Sections 3 and 4.

6. CONCLUSIONS

Most previous active learning algorithms are pool-based. Our work indicates that the pool has several weaknesses. It may limit the range of examples to be queried for effective active learning. In our work, we eliminate the pool completely by allowing the active learners to directly construct queries and ask for labels. We design a specific Tree-ADQ algorithm based on decision trees, and a generic Wrapper-ADQ algorithm that can use any base learning algorithms. Our experiments show that our ADQ algorithms can construct queries that reduce the predictive error rates more quickly compared to the traditional pool-based algorithms. The ADQ algorithms are also more computationally efficient. Our algorithms can also be adapted easily to work with the pool if examples have to be selected from a given pool.

Co-training is another promising research for solving the problem of the lack of labeled data. In co-training, a set (pool) of unlabeled examples is given, and is used to improve supervised learning. In our future work, we will study the usefulness of the pool in co-training.

7. REFERENCES

[1] D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, April 1988.

[2] Y. Baram, R. El-Yaniv, and K. Luz. Online choice of active learning algorithms. *Journal of Machine Learning Research*, 5:255–291, 2004.

[3] C. Blake, E. Keogh, and C. J. Merz. Uci repository of machine learning databases. http://www.ics.uci.edu/~mlearn/MLRepository.html, 1998.

[4] L. Breiman. Random forests. *Machine Learning*, V45(1):5–32, October 2001.

[5] D. A. Cohn, L. Atlas, and R. E. Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, 1994.

[6] Y. Freund, S. H. Seung, E. Shamir, and N. Tishby. Selective sampling using the query by committee algorithm. *Machine Learning*, 28(2-3):133–168, 1997.

[7] R. Greiner, A. J. Grove, and D. Roth. Learning cost-sensitive active classifiers. *Artif. Intell.*, 139(2):137–174, August 2002.

[8] A. Kapoor and R. Greiner. Learning and classifying under hard budgets. pages 170–181. 2005.

[9] M. Lindenbaum, S. Markovitch, and D. Rusakov. Selective sampling for nearest neighbor classifiers. *Machine Learning*, 54(2):125–152, February 2004.

[10] D. D. Margineantu. Active cost-sensitive learning. In *the Nineteenth International Joint Conference on Artificial Intelligence*, Edinburgh, Scotland, 2005.
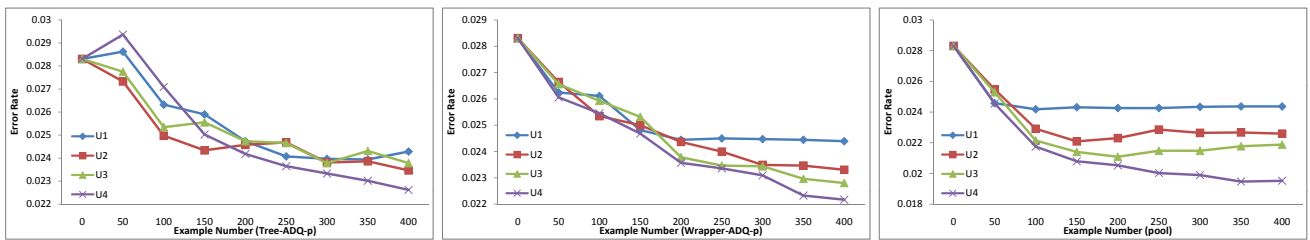
Figure 3: Comparing predictive error rates on "sick" with different real-data pools. The lower the curve, the better.
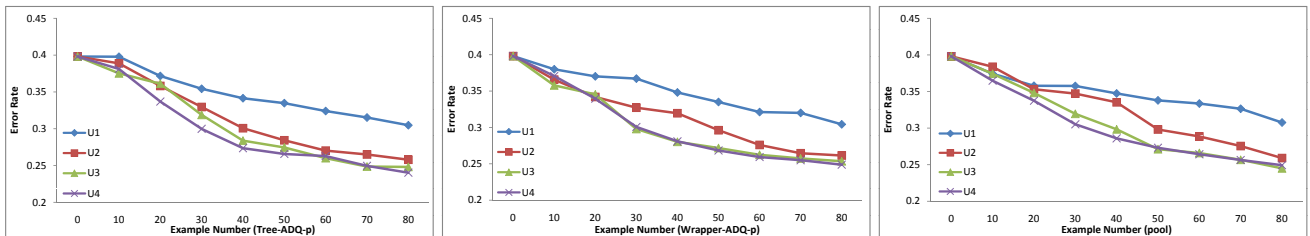

Figure 4: Comparing predictive error rates on "sonar" with different artificial pools. The lower the curve, the better.

[11] A. Mccallum and K. Nigam. Employing em and pool-based active learning for text classification. In *ICML '98: Proceedings of the Fifteenth International Conference on Machine Learning*, pages 350–358, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[12] T. M. Mitchell. *Machine Learning*. McGraw-Hill Science/Engineering/Math, March 1997.

[13] F. Provost and P. Domingos. Tree induction for probability-based ranking. *Machine Learning*, 52(3):199–215, September 2003.

[14] R. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., 1993.

[15] N. Roy and A. Mccallum. Toward optimal active learning through sampling estimation of error reduction. In *Proc. 18th International Conf. on Machine Learning*, pages 441–448. Morgan Kaufmann, San Francisco, CA, 2001.

[16] M. Saar-Tsechansky and F. Provost. Active sampling for class probability estimation and ranking. *Machine Learning*, 54(2):153–178, February 2004.

[17] H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *COLT '92: Proceedings of the fifth annual workshop on Computational learning theory*, pages 287–294, New York, NY, USA, 1992. ACM Press.

[18] S. Tong and D. Koller. Support vector machine active learning with applications to text classification. *Journal of Machine Learning Research*, 2:45–66, 2002.

[19] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, second edition, June 2005.

[20] T. Zhang and F. J. Oles. A probability analysis on the value of unlabeled data for classification problems. In *Proc. 17th International Conf. on Machine Learning*, pages 1191–1198, 2000.