Asking Generalized Queries to Domain Experts to Improve Learning

Jun Du, Charles X. Ling, *Senior Member, IEEE*

Abstract—With the assistance of a domain expert, active learning can often select or construct fewer examples to request their labels to build an accurate classifier. However, previous works of active learning can only generate and ask specific queries. In real-world applications, the domain experts (or oracles) are often more readily to answer "generalized queries" with don't-care attributes. The power of such generalized queries is that one generalized query is often equivalent to many specific ones. However, overly general queries are not good as answers from the domain experts (or oracles) can be highly uncertain, and this makes learning difficult. In this paper, we propose a novel active learning algorithm that asks good generalized queries. We then extend our algorithm to construct new, hierarchical features for both nominal and numeric attributes. We demonstrate experimentally that, our new method asks significantly fewer queries compared with the previous works of active learning, even when the initial labeled dataset is very small, and the oracle is inaccurate in class probability estimations. Our method can be readily deployed in real-world data mining tasks where obtaining labeled examples is costly.

This is an extension of the paper published in IEEE ICDM 2009 [16].

Keywords—active learning, domain expert, generalized query.

1 Introduction

In recent years, domain-driven data mining (D^3M) [10], [27], [38], [9] has received extensive attention in data mining. Unlike the traditional data-driven data mining, D^3M tends to discover actionable knowledge by tightly integrating the data mining methods with the domain-specific business processes. However, in most cases, the domain-specific actionable knowledge cannot be discovered without the support of domain knowledge, mainly provided by human experts. Thus, the human-machine-cooperated interactive knowledge discovery process is widely applied in real world applications [27]. Active learning, as a typical interactive learning paradigm, can naturally integrate the automated learning algorithm with the domain experts. Previous research in active learning shows, guided by the domain experts, data mining algorithms can often achieve significantly better performance than the traditional data-only learning.

Motivated by domain-driven data mining, in this paper, we attempt to maximize the utility of domain experts (oracles) in active learning process. Specifically, traditional active learning algorithms (see Section 2 for a detailed review) only assume that the domain expert (oracle) is capable of answering specific queries, with all attribute value provided. For example, if the task is to predict osteoarthritis based on a patient dataset with 30 attributes, the previous active learners could only ask the specific queries as: does this patient have osteoarthritis, if ID is 32765, name is Jane, age is 35, gender is female, weight is 85 kg, blood pressure is 160/90, temperature is 98F, no pain in the knees, no history of diabetes, and so on (for all 30 attributes). Many of these 30 attributes may not be relevant to osteoarthritis in this case. Not only could specific queries like this confuse the domain experts (oracles), but the answers returned are also specific: each label given is only for one specific query.

In real-world situations, the domain experts (oracles) are often more readily to answer generalized queries, such as "are people aged over 50 with knee pain likely to have osteoarthritis?" Here only two relevant attributes (age and type of pain) are mentioned, and the other 28 are don't-care. We have discussed with some experts in heart-disease diagnosis and used-car sale, and they regard this type of generalized queries intuitive and easy to comprehend. Thus, in this paper, we assume that the domain expert (oracle) is more powerful; it can answer generalized queries by returning probabilistic labels. Not only are such generalized queries more natural and relevant, answers from the oracle also provide much more information, as one generalized query is often equivalent to many specific queries. In this example, the answer for this query is for all people over 50 with knee pain. This allows the active learner to improve learning effectively and quickly.

The difficulty of the generalized queries is that the answers from the oracle can often be uncertain.[1] For example, the answer to the above generalized query

• *J. Du and C.X. Ling are with the Department of Computer Science, The University of Western Ontario, London, Ontario, Canada N6G 5B7. E-mail: {jdu42, cling}@csd.uwo.ca. Contact author: C.X. Ling.*

1. This is true even if we assume that answers for specific queries are always 100% certain. However, in some real-world applications, answers for specific queries may also be uncertain. We will study this issue in our future work.

can be "Yes with a 90% probability". An overly general query, such as "are people aged over 50 likely to have osteoarthritis?" (age only), might receive yes with only a 60% probability. Indeed, the experts in the heart-disease diagnosis and used-car sale also sometimes have to reply with low certainties in their answers. Highly uncertain answers can make learning difficult as they may introduce noise into the training data; or, they waste the effort of the domain experts (or oracles) if these answers are directly discarded.

In general, the more general a query is (with more don't-care attributes), the more powerful it is (representing more specific instances), but usually the more uncertain the answer is from the oracle. Our task is to design an active learner that attempts to ask generalized queries with highly certain answers from the oracle. The task is not trivial. As far as we know, no previous work of active learning can deal with such generalized queries. See Section 2 for details.

In this paper we assume that the oracle is capable of answering generalized queries, and we propose a novel active learning paradigm in which such generalized queries can be asked and answered. We design a new algorithm called AGQ, for Active learner with Generalized Queries. AGQ can construct generalized queries with don't-care attributes, for either the pool-based or the membership-query active learner. See Section 3 for details. However, AGQ can only generalize specific attribute values to don't-care. We then extend AGQ to AGQ$^+$, which can generalize specific attribute values to meaningful new features for both nominal and numeric attributes. For example, AGQ$^+$ can ask such queries as "are people aged between 50 and 65, with moderate or severe knee pain, likely to have osteoarthritis?" Here, age (a numeric attribute) is generalized to a range, and knee pain (a nominal attribute) is generalized to a subset of values. These newly constructed features can form hierarchical structures, and are often meaningful in real-world applications. See Section 4 for the detailed description of AGQ$^+$. Experiments on synthetic and real-world datasets show that AGQ and AGQ$^+$ ask significantly fewer queries compared with the traditional active leaner. See Sections 5 for details. In addition, AGQ$^+$ can also automatically produce subsets for nominal attributes and ranges for numeric attributes, which can be used in further learning. To the best of our knowledge, this is the first work proposing active learning with generalized queries, and showing that it is highly effective.

One might argue that it may be difficult for the oracle or human expert to provide accurate probabilities of the labels for the generalized queries. As we will show in Section 6.1, when the probabilities of labels are contaminated with low noise, AGQ still learns quite well. That is, AGQ is robust with estimated probabilities of the labels. In addition, Section 6.2 studies the difference between AGQ (AGQ$^+$) and active learning with feature selection. We will show that active learning with feature selection performs significantly worse than the proposed AGQ methods, due to the overly general queries it produces. In Section 6.3, we will discuss the behaviour of AGQ (AGQ$^+$) with only few initial labeled examples, and propose an additional heuristic to handle this issue.

2 Related Work

Involving domain experts into learning is a common and often necessary step in domain-driven data mining [10]. It has been shown that, in many real-world applications, domain experts can play an important role in the entire knowledge discovery and data mining process [2], [5]. Specifically, the learning algorithms guided by domain experts can achieve significantly better performance than the automated data-only learning. Thus, active learning has been intensively studied, due to its natural capability of integrating domain experts into the learning process.

Most previous works of active learning can be divided into two paradigms: the pool-based active learning and the membership query.[2] In the pool-based active learning, a pool of unlabeled examples is given, and the learner can only choose examples to label from the pool [21]. Briefly speaking, the pool-based active leaner first evaluates each example in the pool, to decide which one can maximumly improve the performance of the current model. Then the learner acquires its label from oracle to update the labeled training set and the learning model, and the process repeats. On the other hand, active learning with membership queries (or direct query construction) can construct examples (without the need of the pool) and request labels [4], [23]. Previous experiments show that both of these active learning methods reduce the number of labeled examples needed, compared with labeling examples randomly.

The essence of active learning lies in the "goodness" measurement of the unlabeled examples with respect to the current model. Many criteria have been proposed in the literatures. *Uncertainty sampling* [21] considers the most uncertain example as the most valuable one, and has been thoroughly studied and widely used in many previous researches [36], [32], [7], [22], [28]. *Query-by-committee* (QBC) [33] is a more theory-based approach, and considers the example minimizing the version space as optimal. [1] implements QBC by constructing committees from ensemble methods (bagging, boosting, etc.), thus essentially transforms it to a variant of uncertain sampling. Besides, other criteria, such as *variance reduction* [12], *Fisher information ratio* [39], and *estimated error reduction* [31], are also elaborately designed and well accepted in active learning research area. In this paper, the proposed AGQ algorithm can be integrated with any of the above criteria, and the most widely used *uncertain sampling* is chosen for illustration and empirical study in the rest of the paper.

2. Stream-based active learning [11] is considered as another paradigm in some literatures. In essence, it could be viewed as an online version of the pool-based active learning [7].

All previous works of active learning assume that the oracle could only answer specific queries, with all attribute values provided. To the best of our knowledge, our AGQ algorithm in this paper is the first work proposing active learning with generalized queries. Again, the main advantage of AGQ is that one generalized query is usually equivalent to many specific ones. Thus, the answer from the oracle is also for all of the specific queries.

Even though one generalized query is equivalent to multiple specific queries, our AGQ method is still quite different from batch-mode active learning [20], [18]. In batch-mode active learning, the learning model requests labels for a batch of examples (i.e., multiple specific queries) in each iteration, thus the oracle is required to provide multiple answers for all these queries (i.e., with multiple costs). On the other hand, in AGQ, the oracle answers only one generalized query in each iteration (i.e., with one cost). Thus, AGQ costs much less than the batch-mode active learning, for answering queries in the learning process.

[15] proposed active learning with feature labeling, which queries the label for one specific feature (for example, "$puck$" $\longrightarrow$ "$hockey$"), and is mainly used in natural language precessing. Although feature labeling is considered similar to the generalized query, our AGQ algorithm is significantly different in the following three aspects. First, instead of querying label for one specific feature, our AGQ could query the labels for multi-feature combinations (for example, "$puck$" + "ice" + "$player$" $\longrightarrow$ "$hockey$"). Thus, feature labeling is essentially a special case of our AGQ. In other words, our generalized query is a generic paradigm for both instance-based queries and feature-based queries. Second, AGQ always finds the most uncertain example (when integrated with uncertain sampling) and generalizes it to a query. Labeling such uncertain examples has been proved to be very effective in improving predictive accuracy (see Section 5.2 for details). On the other hand, feature labeling generally finds the most predictive (or most frequent) feature for querying, thus the answer from the oracle may not provide much new information to improve the model. Third, and most importantly, as feature labeling always queries label for only one feature, the answer from the oracle could be very uncertain. To deal with this problem, it is assumed in [15] that the oracle could "skip" the uncertain queries. But in fact, the oracle has "worked" on those queries, and the oracle's effort is wasted. On the other hand, AGQ makes a minimal generalization of a specific query, thus the answers from the oracle tend to be certain. Our experiments show that the average certainty of the replies is 90% (see Section 5.2 for details). In any case, every query of AGQ is counted, regardless of the certainty of the reply.

One major step of AGQ is to find irrelevant features and substitute them with the don't care (i.e. "$*$") (see Section 3.2). However, our algorithm is very different from, and much better than, combining feature selection and the traditional active learning. We will discuss this in detail in Section 6.2.

3 AGQ: Active Learning with Generalized Queries

Domain-driven data mining actively involves human experts in the learning process. Active learning naturally put human experts in the process. In this paper we propose a new active learning paradigm in which the learner can ask generalized queries, and we assume that the domain expert or oracle can answer such generalized queries. In this section, we will describe a novel active learning algorithm called AGQ (Active learning with Generalized Queries). AGQ can generalize attributes (nominal or numeric) with specific values to don't-care attributes. In Section 4, we extend AGQ to AGQ$^+$, which generalizes from specific attribute values to subsets of values for nominal attributes, or to ranges for numeric attributes. These newly constructed attributes can form meaningful hierarchies for further learning.

As most previous works of active learning are pool-based, and use uncertain sampling to choose the most valuable unlabeled examples, in this paper, we will also describe AGQ using uncertain sampling in pool-based paradigm. However, as our AGQ is a meta-learning method, it can be equally applied to the membership query active learning, or integrated with any other query strategy. We assume that examples are described by n nominal or numeric attributes $X_1, X_2, ..., X_n$ and the label Y of examples is binary, with values positive (1) and negative (0). The active learner is given an initial labeled training set R, and an unlabeled set U, from which the learner may choose examples to query for their labels from an oracle. A test set T is given but set aside to evaluate the accuracy of the learner during label acquisition.

The AGQ algorithm can be broken down into the following four major steps:

1) The first step is the same as in the previous pool-based active learning algorithms [36], [14], [28]. An initial learner L is built using the current labeled training dataset R. Then, L is used to predict each example in the pool U. The most uncertain example from the pool is chosen. (If the membership active learning is used, then the most uncertain example would be constructed in this step.)
   As an example, the specific example from the pool could be $[1, 0, 1, 1, 0, 1]$, with the predicted probability of 52% for the class 1 (and 48% for the class 0), according to the current model L. This is the most uncertain (the probability of the majority class is closest to 50%) among all examples in the pool.
2) AGQ then finds irrelevant attributes in the most uncertain example above, and substitute them with

"$*$" (representing don't-care values).[3]
For example, the generalized query based on the example $[1, 0, 1, 1, 0, 1]$ could be $[1, *, 1, *, 0, 1]$.

3) AGQ submits this generalized query to the oracle, which will return a label with a probability distribution.
For example, the oracle may return a probability of 0.9 for positive (and 0.1 for negative) for the generalized query $[1, *, 1, *, 0, 1]$.

4) AGQ will utilize the label and the probability distribution to update the training data, and iterate to Step 1 (to continue learning actively).
For example, from the generalized query $[1, *, 1, *, 0, 1]$ and the probability distribution for the class (0.9 for class 1 and 0.1 for class 0), four specific examples, $[1, 0, 1, 0, 0, 1]$, $[1, 0, 1, 1, 0, 1]$, $[1, 1, 1, 0, 0, 1]$, and $[1, 1, 1, 1, 0, 1]$, each with a probability label (0.9 for 1 and 0.1 for 0), could be added into the training set. This represents the power of generalized queries: each can represent effectively a set of specific queries. This would be useful if the probability of the majority class is high (close to 1). Otherwise, noise is introduced into the training set, and as we will show later, accuracy can even worsen. (We will study other strategies of utilizing the probabilistic labels in our future work.)

We will discuss each step in detail in the following subsections.

3.1 Finding the Most Uncertain Example

Similar to the previous works of the pool-based active learning, AGQ first builds a predictive model based on the current set of labeled examples, and uses it to predict each example in the pool. The most uncertain example from the pool, the one with the probability of the majority class closest to 50%, is chosen as the result of this first step.[4]

As the probability of the prediction is crucial in choosing the most uncertain example, we use an ensemble of decision trees in AGQ. Specifically, the bagging [8] of 100 j48 decision trees (implemented in Weka [37]) is used. The probability distribution of the prediction is estimated by the prediction of the 100 trees in the ensemble. Such an ensemble of many trees improves the probability estimation, compared with a single tree [29]. The standard decision tree algorithm is chosen because it tends to build small trees; this facilitates us to find irrelevant attributes in the next step.

3. Although feature selection [19], [25] can also discover irrelevant features, as we will show in Section 6.2, the AGQ method is significantly better than feature selection.

4. For highly imbalanced and cost-sensitive data, an optimal threshold for classification can be calculated [17], or found via cross-validation [34], and the example with the probability closest to the threshold is chosen as the most uncertain one. Thus, our algorithm can also deal with imbalanced and cost-sensitive data.

3.2 Constructing the Generalized Query

After finding the most uncertain (specific) example from the pool in the first step, AGQ needs to discover the irrelevant attributes (don't-care attributes).

If the set of m attributes are irrelevant, then the examples with any combination of their values would have the same prediction with similar probability estimation. The reverse may not be true, but it can be used as a heuristic to find the set of irrelevant attributes. However, there are $\binom{n}{m}$ subsets of m attributes (given a total of n attributes), and for each subset, 2^m value combinations (for binary attributes) must be tested. The task is clearly computationally expensive.

A heuristic, similar to the process of finding the largest itemsets in mining association rules [3], [24], is designed. More specifically, let D be the current don't-care attribute list, and let x_u be the current most uncertain example. We gradually expand D by adding more irrelevant attributes via greedy search, as follows. For each attribute X_i not currently in D, we generate a fixed number (100 in our experiments) of examples with randomly assigned values for attributes in D and X_i, all based on x_u. The number of examples is fixed to prevent combinatorial explosion of attribute values when D grows. The attribute value is randomly chosen according to the distribution of that attribute values in the original data set. This most accurately reflects the distribution of examples in the domain.[5] The attribute X_i with the smallest change in the probability distribution of all 100 examples is then regarded as irrelevant, and added into D if the smallest change is less than a pre-defined threshold. The process continues until D cannot be grown further. The generalized query is the one with don't-care (i.e., "$*$") for all attributes in D. This process is depicted with the pseudo code in Algorithm 1.

Clearly, this can generate most general queries (i.e., queries with most don't-care attributes) based on the current learning model. However, queries with too many don't-care attributes can be overly general, and labels from the oracle can be highly uncertain. Thus, we demand the threshold θ in Algorithm 1 to be a very small number (0.0001 in our case). This would allow AGQ to find the most general queries that, hopefully, also include all relevant attributes. Still, as the initial labeled training set can be very small, the current learning model can be inaccurate. Thus, AGQ may produce generalized queries with don't-care for relevant attributes (see Table 1 in Section 5.1). This will be especially true when the initial labeled training set is very small. In Section 6.3 we study AGQ when there are only two initial labeled examples.

3.3 Asking Generalized Queries to Oracle

In our work, we assume that the oracle can answer generalized queries with don't-care attributes just as

5. The same random sampling method is used in Sections 3.3 and 3.4.

Algorithm 1: find_don't-care_attributes

Input: M, the current learning model; x_u, the most uncertain example; θ, the predefined threshold.
Output: D, the don't-care attribute list.

p_u = probability of majority class for x_u (estimated by M);
$D = \emptyset$;
$noChange = true$;
repeat
  foreach $X_i \notin D$ **do**
    for $n = 1$ *to* 100 **do**
      begin // Generate x_n
        $x_n = x_u$;
        Randomly assign X_j for all $X_j \in D$;
        Randomly assign X_i;
      end
      p_n = probability of majority class for x_n (estimated by M);
    end
    $S_i = \sum_{n=1}^{100}(p_n - p_u)^2/100$;
  end
  Choose X_i with the smallest S_i;
  if $S_i < \theta$ **then**
    Add X_i in D;
  else
    $noChange = false$;
  end
until $noChange$ *is* $false$;

easily as specific queries (without don't-care attributes). We believe that in most real-world situations, human experts can easily answer such generalized queries with an estimated probability. As we will also show in Section 6.1, our AGQ performs well with a small error in probability estimation. Thus it is quite robust.

In Section 5.2 we will test AGQ on the UCI datasets [6], comparing it with the traditional pool-based active learner. An interesting question arises: as we do not know the target functions of the UCI datasets, nor do we have human oracles for them, how can such generalized queries be answered?

We design the following method to simulate human oracles to answer the generalized queries. We first train a model based on the original dataset to represent the target function. This is the best model we can get as it is built from the whole dataset. Specifically, we use the bagging of 100 j48 decision trees on the whole dataset to represent the target model. But still, this target model, as a black-box, cannot answer generalized queries directly. Since each generalized query effectively represents a set of specific queries, a set of such specific queries (in which the don't-care attributes are replaced with specific values sampled randomly) are generated. To avoid combinatorial explosion when the generalized query has too many don't-care attributes, the size of the set is fixed at 100. The target model then returns the predicted probability distribution of these 100 examples in the set.

One may argue that the generalized queries could be unrealistic thus hard to be answered by oracle (as in membership query). In the pool-based paradigm, AGQ chooses a specific example from the pool and generalizes it to a query. If the example is realistic, the generalized query is always realistic as well, so the oracle should be able to answer. For example, if the specific example is $[name = Jane, gentle = female, pregnant = yes, age = 30, \ldots]$, then the generalized query could be $[name = *, gentle = *, pregnant = yes, age = *, \ldots]$. Unrealistic generalized queries (such as $[name = *, gentle = male, pregnant = yes, age = *, \ldots]$) will never be constructed.

The next key step of AGQ is to utilize the generalized queries and their labels from the oracle to further improve learning.

3.4 Updating the Training Dataset

Given the probability distribution to the generalized query from the oracle, we need to utilize it to expand the training dataset and to build a better classifier. Again, because each generalized query effectively represents a set of specific queries, more than one specific example can be added into the original labeled training set. There are two issues to be resolved, however. One is how large the set of specific queries should be; the second is how to label those examples in the set.

The first question is relatively easy to answer. Again to avoid combinatorial explosion, a set with a fixed size (100 in our experiments) of specific examples is generated first, in which each don't-care attribute is replaced randomly by a specific value of that attribute. However, experiments (Section 5.2) indicate that the number of new examples added may influence adversely the distribution of the initial training set. If the initial training set is too small, then the new examples added may be overwhelming, and thus changing the distribution of examples in the training set. Thus, the number of examples added into the training set is the minimum of 100, half of the size of the initial training set, and the number of value combinations of all don't-care attributes.[6]

How should each specific query be labeled? As the oracle returns probability distribution of labels (such as 0.9 for positive, 0.1 for negative) for the generalized queries, specific examples can simply carry weighted labels if the learning model (bagging of 100 j48 trees here) can take weighted examples directly. Most learning algorithms (such as decision trees, naive Bayes, instance-based learning) can indeed take weighted examples naturally. Thus, in the above situation, every specific example carries a positive label with weight 0.9, and a negative label with weight 0.1.

Thus in AGQ, the labeled training set is usually increased by adding multiple labeled examples (with probability labels), rather than by adding just one labeled

6. Note that, as the generalized queries are constructed from the most uncertain examples, when updating the training set, these most uncertain examples are always added into the training set. Thus, AGQ always outperforms the traditional uncertain sampling method.

example in the traditional pool-based active learning. If examples added are mostly valid, and the probability of the majority class is near 1 (a highly certain label), the learning can be improved dramatically, as we will show in the experiments.

4 AGQ$^+$

In the previous section, we propose a novel algorithm AGQ that is composed of four steps to implement active learning with generalized queries. However, the AGQ algorithm in Section 3.2 can produce generalized queries with attributes that are either entirely irrelevant (i.e., generalized as "$*$"), or entirely specific (i.e., keeping the original specific value). That is, as long as the attribute is relevant, it could be represented by only one specific value in the generalized queries. This is clearly very restrictive. In most real-world applications, however, nominal attributes can form subsets (of values), and numeric attributes can form ranges. What we hope is that the active learner can automatically form such new, high-level features when it asks generalized queries.

For example, to predict "osteoarthritis", "knee pain" could be a relevant nominal attribute with values "none", "moderate" and "severe", and "age" could be another relevant attribute with numeric values. Then, in addition to generalizing the irrelevant attributes as "$*$", we may also generalize the relevant attributes to several nominal values (such as, "knee pain" being "moderate" or "severe") or a numeric interval (such as, "age" being $[50, 65]$). We can then construct generalized queries, such as "are people aged between 50 and 65, with moderate or severe knee pain, likely to have osteoarthritis?".

Not only are these generalized queries more natural and flexible to represent queries with difference degrees of generalization, the new features constructed can also form hierarchical structures, and can be meaningful for further learning. For example, if a subset of nominal attribute values or a range of a numeric attribute is repeatedly generated by the active learner, then they can be meaningful high-level concepts, to be used in future learning, or transfer learning [13], [30]. See Section 5.3 for more discussions. We call this extension "AGQ$^+$", due to its powerful generalization ability.

AGQ$^+$ has different strategies in the second step of AGQ in constructing the generalized queries; the other three steps (i.e., finding the most uncertain example, asking generalized queries to oracle, and updating the training dataset) are the same as AGQ (see Section 3). An additional fifth step is added. In this step, subsets of nominal attributes and ranges of numeric attributes generated by AGQ$^+$ are consolidated, and hierarchies may be formed. In this section, we will mainly describe how AGQ$^+$ generates subsets of nominal attributes and ranges of numeric attributes.

4.1 Nominal Attributes

For nominal attributes, we first still find all the *strong-irrelevant* attributes (i.e., the don't-care attributes) as in Section 3.2. Then, we check all the remaining attributes by greedy search to identify the *weak-irrelevant* attributes (i.e., the attributes that can be generalized with several, but not all, nominal values). The main idea is that the class probability of examples with combinations of weak-irrelevant attribute values should be the very similar from the current learning model. Our heuristic strategy is to use this property to discover which attributes should be weak-irrelevant with corresponding values. More specifically, we still denote by x_u and D the current most uncertain example and the strong-irrelevant attribute list (don't-care attribute list) respectively. We also denote by W the weak-irrelevant attribute list (with the corresponding attribute values). Given x_u found by the current learning model, D constructed by Algorithm 1, and an initially empty W, we gradually expand W with the weak-irrelevant attributes (and the corresponding attribute values), as follows. For each attribute X_i not currently in D and W, and for each of its attribute value $X_i = a_{ij}$, we generate a fixed number (100 in our experiments) of examples with randomly assigned values for the attributes in D and W, all based on x_u.[7] The current learning model then makes predictions on the class probabilities of these examples. If the model produces exactly the same class probabilities for all these examples and x_u, we add the current attribute X_i (and the corresponding attribute value a_{ij}) into W. Therefore, after checking all the rest attributes (together with the corresponding attribute values), we can identify all the weak-irrelevant attributes and include them into W. Furthermore, we can construct the generalized query, by substitute all attributes in D with $*$ and all attributes in W with their corresponding values in W. This process is depicted with the pseudo code in Algorithm 2.

4.2 Numeric Attributes

For numeric attributes, we apply a similar strategy to identifying weak-irrelevant attributes after obtaining all the strong-irrelevant ones. Roughly speaking, given a strong-irrelevant attribute list D, an initially empty weak-irrelevant attribute list W, and the current most uncertain example x_u, we gradually expand W with weak-irrelevant attributes (and their corresponding values). However, unlike nominal attributes, there are infinite valid values for numeric attributes, and we need find a numeric range (instead of several nominal values) for each weak-irrelevant attribute (such as, $[50, 65]$ for age). Thus, Algorithm 2 cannot be applied here. Instead, for each attribute X_i not currently in D and W, we construct a numeric range $[a_i - \delta, a_i + \delta]$ based on the current attribute value a_i and a pre-defined small number δ.[8] Then, we generate a fixed number (100 in our experiments) of examples with randomly assigned

7. For the attributes in D, we randomly assign any attribute values; whereas for the attributes in W, we only randomly assign the corresponding attribute values previously identified.

8. In our experiments, we set δ as 1/20 of entire valid attribute range.

Algorithm 2: find_weak-irrelevant_attributes (nominal)

Input: M, the current learning model; x_u, the most uncertain example; D, the strong-irrelevant attribute list (don't-care attribute list).
Output: W, the weak-irrelevant attribute list (nominal).

p_u = probability of majority class for x_u (estimated by M);
$W = \emptyset$;
foreach $X_i \notin (D \cup W)$ **do**
  foreach $X_i = a_{ij}$ **do**
    for $n = 1$ *to* 100 **do**
      begin // Generate x_n
        $x_n = x_u$;
        Randomly assign X_j for all $X_j \in D$;
        Randomly assign X_k (with available nominal values) for all $X_k \in W$;
      end
      p_n = probability of majority class for x_n (estimated by M);
    end
    $S_{ij} = \sum_{n=1}^{100}(p_n - p_u)$;
  end
  if $S_{ij} = 0$ **then**
    Add X_i (with a_{ij}) in W;
  end
end

Algorithm 3: find_weak-irrelevant_attributes (numeric)

Input: M, the current learning model; x_u, the most uncertain example; D, the strong-irrelevant attribute list (don't-care attribute list); δ, the pre-defined small number.
Output: W, the weak-irrelevant attribute list (numeric).

p_u = probability of majority class for x_u (estimated by M);
$W = \emptyset$;
foreach $X_i \notin (D \cup W)$ **do**
  $UB = LB = a_i$ (current attribute value);
  repeat
    $UB = a_i + \delta$;
    $LB = a_i - \delta$;
    for $n = 1$ *to* 100 **do**
      begin // Generate x_n
        $x_n = x_u$;
        Randomly assign X_j for all $X_j \in D$;
        Randomly assign X_k for all $X_k \in W$ (within available numeric range);
        Randomly assign X_i within $[LB, UB]$;
      end
      p_n = probability of majority class for x_n (estimated by M);
    end
    $S_{ij} = \sum_{n=1}^{100}(p_n - p_u)$;
  until $S_{ij} > 0$;
  Add X_i (with numeric range $[LB, UB]$) in W;
end

values for the attributes in D, W and the current attribute, all based on x_u.[9] Again, the current learning model makes predictions on the class probabilities of these examples. If the model produces exactly the same class probabilities for all of these examples and x_u, the numeric range will be again expanded by δ. Otherwise, it stops. Then, the current attribute (and its final numeric range) is included into W as a weak-irrelevant attribute. Therefore, after checking all the remaining attributes, we can identify all the weak-irrelevant attributes, and construct the generalized query by substitute all attributes in D with $*$ and all attributes in W with their corresponding numeric ranges. This process is depicted with the pseudo code in Algorithm 3.

In the next section, we will perform extensive experiments with AGQ and AGQ$^+$.

5 Experiments with Generalized Queries

In this section, we conduct experiments on a synthetic dataset and 14 UCI [6] datasets to compare AGQ with the previous active learning algorithm that asks specific queries. We then apply AGQ$^+$ on the same UCI datasets to see its advantages over AGQ.

5.1 AGQ on Synthetic Dataset

In this subsection, we use synthetic data to empirically study the performance of AGQ, compared with the traditional pool-based active learning with uncertain sampling.[10]

In addition, we also present the performance of the *optimal* AGQ, which represents the best performance that AGQ could possibly achieve. Specifically, for each generalized query, the optimal AGQ gradually specifies the original attribute values for the don't-care attributes, till the oracle provides a certain answer ($P(Y = 1|X) \geq 0.95$ or $P(Y = 0|X) \geq 0.95$ in our experiments). The training set is thereafter expanded according to this query and the answer. That is, the training set is *only* updated when the oracle returns highly certain labels (≥ 0.95). However, some extra queries may still be asked to the oracle when the answer is not highly certain, which makes optimal AGQ not realistic. Here, we simply do not count those extra queries, and only count the "effective" ones — those with certainty great than (or equal to) 0.95. Thus, it could reflect the fewest number of queries that AGQ can ask, which indicates the best performance AGQ can ever achieve.

We choose the target function as a decision tree with five relevant attributes, $X1$ - $X5$, and six leaves, $L1$ - $L6$, as in Figure 1. To simulate the real-world dataset, we add another five *irrelevant* attributes, $X6$ - $X10$, to

9. For the attributes in D, we randomly assign any attribute values; for the attributes in W, we randomly assign the values within the previously identified numeric range; whereas for the current attribute, we randomly assign value within $[a_i - \delta, a_i + \delta]$.

10. Note that, as we also use a bagging of 100 decision trees for the traditional pool-based active learning (as same as for AGQ), the most uncertain example can also be considered as the example with the maximum disagreement for the current committee (constructed by the current 100 decision trees). Thus, uncertain sampling in this case can also be regarded as an implementation of QBC.

generate the synthetic data. We assume that all these attributes are binary, so is the class label. Therefore, with 10 binary attributes, we can generate $2^{10} = 1024$ different examples, and label them with the target function. With this synthetic data, we know what the target function is and what the irrelevant attributes are. We can also directly use the target function as the oracle to answer the generalized queries.

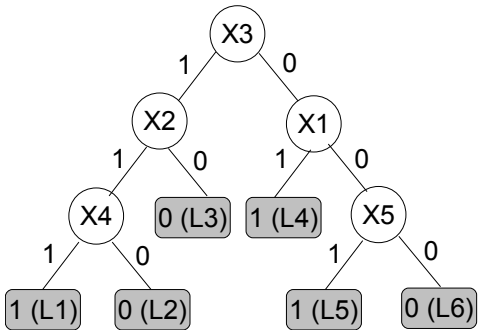

Fig. 1. Target tree used to generate synthetic data.

The experiment is repeated on the synthetic dataset 20 times. Each time, the whole dataset is randomly split into three disjoint subsets: the training set, the unlabeled set, and the test set. The training set and the test set are always 2% and 25% of the whole dataset respectively, and the rest is the unlabeled set.

Figure 2 plots the average error rates of the optimal AGQ ("AGQ-Opt" in short), AGQ and the traditional pool-based active learning ("Pool" in short). We can see clearly from Figure 2 that, AGQ's performance is quite close to the (unrealistic) optimal AGQ, and is much better than "Pool". This indicates that the strategies we designed for AGQ (Section 3) is quite effective — AGQ asks generalized queries with certain labels; that is, they are not overly general. Overly general queries would receive uncertain labels, and would negatively affect learning. This can happen especially when the initial training set is very small. See Section 6.3.

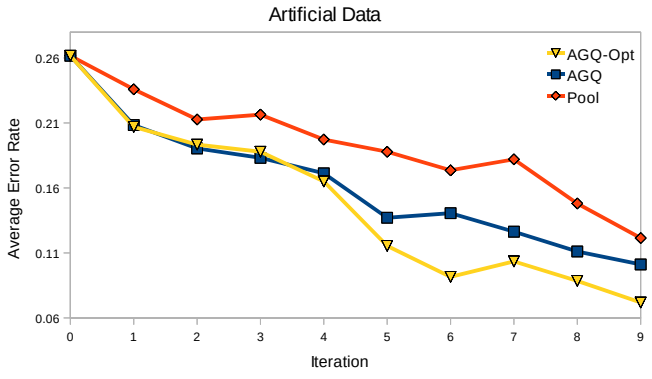

Fig. 2. Comparison of the average error rate among "AGQ-Opt", AGQ and "Pool" on the synthetic data.

To further compare AGQ and "Pool", we extract a typical series of queries from them during the active learning process. Table 1 tabulates these queries (Query in the table), as well as leaf(ves) in the target tree that these queries fall into (Classified by Leaf(ves)), ideal query according to the target tree (Ideal Query), answer from the oracle (Answer), number of specific examples generated to update the training set (No. of Examples), and error rate of the updated classifier (Error Rate). We can see from Table 1 that AGQ always constructs generalized queries with don't-care attributes while "Pool" can only choose the most specific queries. These generalized queries from AGQ may not be as general as the ideal queries (constructed directly from the target tree; see Figure 1), but they still contain most irrelevant attributes. Only one query (Query 2) is overly general (falling into two leaves), thus the answer to this query is highly uncertain (54%). However, such overly general queries rarely occur in AGQ learning. (Thus, the performance of AGQ is quite similar to the optimal AGQ, as we showed earlier.) In this case, answers for the other four queries from the oracle are highly certain (100%). Thus, AGQ can often include more examples with correct labels into the training set in each iteration, and obtain significantly lower error rates (compared with "Pool").

To summarize from the experiment on the synthetic data, AGQ can often identify correctly the irrelevant attributes and construct correctly the generalized queries with highly certain answers from the oracle. Thus the performance of the classifier is significantly improved when the corresponding multiple specific examples (with correct labels) are included into the training set. This yields the outstanding performance of AGQ (similar to the optimal AGQ) on the synthetic dataset, compared with the traditional pool-based active learning.

5.2 AGQ on UCI Datasets

In this subsection, we use 14 real-world datasets from the UCI Machine Learning Repository [6] to compare AGQ with the optimal AGQ and the pool-based active learning algorithm. All of these datasets have binary class and no missing values. Information on these datasets is tabulated in Table 2.

Each whole dataset (D) is first split randomly into three disjoint subsets: the training set (R), the unlabeled set (U), and the test set (T). The test set T is always 25% of D. To make sure that active learning can possibly show improvement when the unlabeled data are labeled and included into the training set, we choose a small training set for each dataset such that the "maximum reduction" of the error rate[11] is large enough (greater than 10%). The training sizes of the 14 UCI datasets range from 1/200 to 1/5 of the whole datasets, also listed

11. The "maximum reduction" of the error rate is the error rate on the initial training set R alone (without any benefit of the unlabeled examples) subtracts the error rate on R plus all the unlabeled data in U with correct labels. Thus, the "maximum reduction" reflects the upper bound on error reduction that active learning can achieve.

	AGQ	Pool
Query 1	[1, 1, 1, 0, *, *, *, *, *, *]	[1, 1, 1, 0, 1, 1, 1, 1, 0, 0]
Classified by Leaf(ves)	L2	L2
Ideal Query	[*, 1, 1, 0, *, *, *, *, *, *]	[*, 1, 1, 0, *, *, *, *, *, *]
Answer	0, 100%	0
No. of Examples	10	1
Error Rate	0.18	0.27
Query 2	[0, *, 0, 1, *, *, *, *, *, *]	[1, 0, 1, 1, 0, 0, 1, 0, 0, 1]
Classified by Leaf(ves)	L5, L6	L3
Ideal Query	-	[*, 0, 1, *, *, *, *, *, *, *]
Answer	0, 54%	0
No. of Examples	10	1
Error Rate	0.21	0.22
Query 3	[0, 1, 0, 1, 1, 0, 0, *, 1, *]	[1, 1, 1, 1, 0, 1, 1, 1, 0, 1]
Classified by Leaf(ves)	L5	L1
Ideal Query	[0, *, 0, *, 1, *, *, *, *, *]	[*, 1, 1, 1, *, *, *, *, *, *]
Answer	1, 100%	1
No. of Examples	8	1
Error Rate	0.16	0.26
Query 4	[0, 1, 0, 1, 0, 1, *, *, 0, *]	[1, 0, 1, 1, 0, 1, 0, 0, 1, 1]
Classified by Leaf(ves)	L6	L3
Ideal Query	[0, *, 0, *, 0, *, *, *, *, *]	[*, 0, 1, *, *, *, *, *, *, *]
Answer	0, 100%	0
No. of Examples	8	1
Error Rate	0.17	0.26
Query 5	[1, *, 0, *, 0, *, 1, *, *, *]	[1, 1, 1, 0, 0, 1, 0, 0, 1, 1]
Classified by Leaf(ves)	L4	L2
Ideal Query	[1, *, 0, *, *, *, *, *, *, *]	[*, 1, 1, 0, *, *, *, *, *, *]
Answer	1, 100%	0
No. of Examples	10	1
Error Rate	0.13	0.2

TABLE 1
Comparison of five consecutive queries between AGQ and "Pool" on synthetic data.

Dataset	Type of Attributes	No. of Attributes	No. of Examples	Class Distribution	Training Size
breast-cancer	nominal	9	277	196/81	1/5
breast-w	numeric	9	699	458/241	1/10
colic	nominal/numeric	22	368	232/136	1/5
credit-a	nominal/numeric	15	690	307/383	1/20
credit-g	nominal/numeric	20	1000	700/300	1/100
diabetes	numeric	8	768	500/268	1/10
heart-statlog	numeric	13	270	150/120	1/10
hepatitis	nominal/numeric	19	155	32/123	1/5
ionosphere	numeric	33	351	126/225	1/20
kr-vs-kp	nominal	36	3196	1669/1527	1/100
mushroom	nominal	22	8124	4208/3916	1/200
sonar	numeric	60	208	97/111	1/5
tic-tac-toe	nominal	9	958	332/626	1/10
vote	nominal	16	435	267/168	1/20

TABLE 2
The 14 UCI datasets used in the experiments.

in Table 2. The unlabeled set (U) is the whole dataset (D) taking away the test set (T) and the training set (R).

The experiment is repeated on each dataset 20 times (i.e., each dataset is randomly split 20 times), when comparing "AGQ-Opt", AGQ and "Pool". We stop training when the error rate of "Pool" is reduced by 3/4 of the "maximum reduction".

Figure 3 plots the average error rates of "AGQ-Opt", AGQ and "Pool" on a typical UCI datasets ("Hepatitis"), and the comparison on all the 14 datasets will be presented later. We can see from Figure 3 that, AGQ performs only slightly worse than "AGQ-Opt" but significantly better than "Pool", similar to the result on the synthetic dataset. This again clearly demonstrates the advantage of AGQ: AGQ performs almost as well as "AGQ-Opt", and significantly outperforms "Pool".

In addition, the t-test (the paired two-tailed t-test with a 95% confidence level) on the average error rates based on the 14 UCI datasets shows that, AGQ wins on 9, ties on 4, and loses on 1 dataset, compared with "Pool". This clearly indicates that, with the same number of queries (same number of iterations), the error rate of AGQ decreases much faster than "Pool".

To further analyse the performance of AGQ and "Pool", we extract some important statistics during the active learning process. They include the average num-

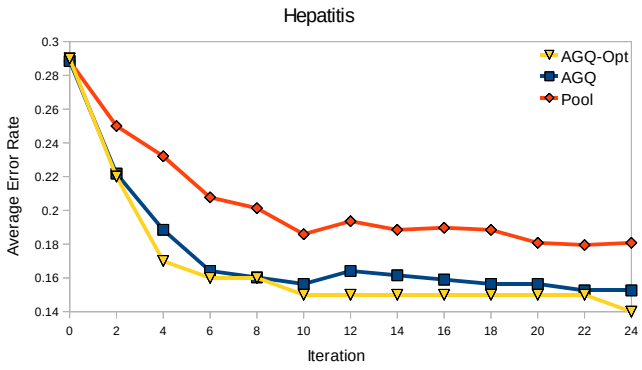

Fig. 3. Comparison of average error rate among "AGQ-Opt", AGQ, and "Pool" on "Hepatitis".

ber of don't-care attributes (and its percentage of the total attributes) in each query (Don't-care Attributes in the table), the average certainty of the oracle (Certainty of Oracle)[12], average number of specific examples generated to update the training set in each iteration (Number of Examples), the average number of iterations of AGQ and "Pool" when their error rates are reduced by 3/4 of the "maximum reduction" (Iteration of AGQ and Iteration of "Pool"), percentage of iteration reduction between AGQ and "Pool" (% of Iteration Reduction), and AGQ wins/ties/loses compared with "Pool" (AGQ w/t/l). Table 3 presents these statistics based on the 14 UCI datasets.

From Table 3 we can see that, on average, AGQ discovers 12.5 don't-care attributes, and includes 16.5 examples into the training sets in each iteration. Moreover, the certainty of the oracle for the constructed generalized queries is as high as 90.21% on average. This explains the good performance of AGQ: it can ask generalized queries, most with certain answers from the oracle. In the three datasets ("breast-w", "ionosphere" and "sonar") where AGQ ties with "Pool", we can notice that the certainties of the oracle are relatively low (87%, 86% and 73% respectively); this probably introduces more noise in the training sets, thus degrading the performance. In the dataset "tic-tac-toe" where AGQ also ties with "Pool", though the certainty of the oracle is high (100%), AGQ could only discover 0.07 don't-care attribute (on average), and include only 1.3 examples (on average) in each iteration. This is probably why AGQ is not much different from the traditional pool-based active learner. For the dataset "kr-vs-kp" where AGQ loses, the certainty of the oracle is relatively high (94%), and 39% of the attributes are discovered as don't-care in each query. So why does AGQ still lose to "Pool"? A detailed study shows that, "kr-vs-kp" is the Chess end-game board-positions, thus the attributes are highly constrained. As there are a total of 36 attributes, the dataset (containing about 3,000 examples) is very sparse; that is, only a small fraction of the attribute value combinations is valid. Thus, the examples generated by AGQ from the generalized queries and included into training set (Section 3.4) are mostly invalid examples (i.e., meaningless board positions). These invalid examples may severely change the distribution of the original dataset thus degrading the performance of AGQ. We will study this issue further in our future work.

From Table 3 we can compare the number of iterations (queries) that AGQ and "Pool" have required to achieve 3/4 of the "maximum reduction" on the error rate. We notice that, on the four datasets where AGQ ties with "Pool", the two methods require almost the same number of iterations (queries). However, on the nine datasets where AGQ wins over "Pool", AGQ asks 61% fewer queries compared with "Pool". Over all 14 datasets, AGQ asks, on average, 36% fewer queries compared with "Pool". This clearly shows the advantage of AGQ: it requires much fewer queries than "Pool" on the tested UCI datasets.

To summarize, AGQ performs significantly better than "Pool" on most UCI datasets (9 out of 14). Moreover, on those datasets where AGQ wins, it requires 61% fewer queries needed for "Pool" to achieve the same error rate reduction. This clearly demonstrates the power of the generalized queries and the advantage of AGQ.

5.3 AGQ$^+$ on UCI Datasets

In this subsection, we conduct the experiments on the same 14 UCI datasets, to compare the performance of AGQ$^+$ and AGQ. All the experimental configurations are the same as in the previous subsection. We perform the same t-test on the average error rates between AGQ$^+$ and AGQ. The results show that for the 14 UCI datasets, AGQ$^+$ wins on 3, ties on the rest 11 datasets, compared with AGQ. This indicates that AGQ$^+$ can predict as well as AGQ in most datasets, and better than AGQ in some cases.

The advantage of AGQ$^+$ lies in not only its (slightly) better performance, but more importantly, its capability of producing natural and powerful generalized queries with meaningful new features during the active learning process. Taking the dataset "Diabetes" as an example. This dataset was originally used in [35] for predicting diabetes from eight numeric attributes of the patients. The meanings and valid range of these attributes can be found in Table 4. However, in [35], those numeric attributes were *manually* discretized into meaningful categories, in order to train a neural network model. With our AGQ$^+$ algorithm, in the learning process, the ranges of numeric attribute are automatically produced, and the generalized queries are accordingly constructed, all based directly on the raw numeric attributes.

To illustrate AGQ$^+$'s capability of producing such numeric ranges and generalized queries on "Diabetes", we list several typical attribute ranges and queries constructed in the active learning process. Table 5 lists all the

12. The certainty of oracle, calculated from the oracle described in Section 3, is always about the majority class (which can be either 1 or 0). Thus, the certainty value is between 0.5 and 1.

Dataset	Don't-care Attributes (% of Total Attributes)	Number of Examples	Certainty of Oracle	Iteration of "Pool"	Iteration of AGQ	% of Iteration Reduction	AGQ (w/t/l)
breast-cancer	2.7 (30%)	14.54	95%	35	18	49%	W
breast-w	5.35 (59%)	32.31	87%	18	18	0%	T
colic	13.15 (60%)	35.68	91%	15	8	47%	W
credit-a	6.38 (43%)	16.43	88%	12	5	58%	W
credit-g	8.54 (43%)	4.97	87%	50	12	76%	W
diabetes	3.02 (38%)	27.31	89%	50	16	68%	W
heart-statlog	5.92 (46%)	12.52	89%	50	25	50%	W
hepatitis	13.47 (71%)	14.96	96%	24	5	79%	W
ionosphere	27.15 (82%)	8	86%	29	29	0%	T
kr-vs-kp	14.89 (39%)	14.48	94%	38	50	-32%	L
mushroom	17.81 (81%)	20	94%	10	6	40%	W
sonar	48.27 (80%)	20	73%	41	34	17%	T
tic-tac-toe	0.07 (1%)	1.28	100%	108	108	0%	T
vote	7.28 (46%)	8.31	94%	12	5	58%	W
Average	12.53 (51.36%)	16.49	90.21%	35.14	24.21	36%	9/4/1

TABLE 3
Important statistics of AGQ and comparison with "Pool" on the 14 UCI datasets.

	Att. Name	Att. Range	Att. Meaning
1	preg	$\{0-17\}$	Number of times pregnant
2	plas	$\{0-199\}$	Plasma glucose concentration a 2 hours in an oral glucose tolerance test
3	pres	$\{0-122\}$	Diastolic blood pressure (mm Hg)
4	skin	$\{0-99\}$	Triceps skin fold thickness (mm)
5	insu	$\{0-846\}$	2-Hour serum insulin (mu U/ml)
6	mass	$\{0-67.1\}$	Body mass index (weight in kg/(height in m)^2)
7	pedi	$\{0.078-2.42\}$	Diabetes pedigree function
8	age	$\{21-81\}$	Age (years)

TABLE 4
Attribute ranges and meanings for "Diabetes".

eight attributes (and class) of the "Diabetes". The upper part of Table 5 shows the manually discretized range for every attribute used in [35]; the middle part shows several typical attribute ranges formed by AGQ$^+$; and the lower part shows eight typical generalized queries and the corresponding probability answers from the oracle (each row represents one generalized query).

Form the middle part of Table 5, we can see that AGQ$^+$ automatically produces necessary ranges of the numeric attributes, some of which are roughly the same as the manual ones, while others are completely different. For example, attribute "skin" is generalized with range $\{0-29\}$, which is very close to the manually discretized category $\{0-25\}$; attribute "insu" is generalized with range $\{142-227\}$, also close to the manually discretized category $\{151-240\}$. In addition, some numeric ranges of the same attribute can clearly form hierarchical structures. For example, for attribute "insu", the second range $\{48-132\}$ is roughly a subset of the first range $\{0-133\}$; and the fourth range $\{143-227\}$ is also roughly a subset of the fifth range $\{145-399\}$. The similar phenomena can also be discovered from other attributes. Such (hierarchical) ranges can form new meaningful features without any human interference, and can be used in further learning.

From the lower part of Table 5, we can see that, AGQ$^+$ can generalize the attributes to numeric ranges in most queries, and also obtain relatively certain answers from the oracle. For example, in Query 1, attribute "preg" is generalized with range $\{0-2\}$, attribute "plas" is generalized with range $\{>171\}$, and this query obtains a 100% certain answer from the oracle (see Column "class"). This clearly illustrates the behaviour of AGQ$^+$: it produces meaningful generalized queries (with automatically discretized attribute categories), and obtains certain answers from the oracle. We can also notice from Table 5 that Query 8 obtains an uncertain answer from the oracle (with 60% probability estimation). However, this type of low certainty queries rarely occur in the whole learning process, thus would not significantly affect the performance of AGQ$^+$.

To summarize, AGQ$^+$ performs slightly better than AGQ. Most importantly, AGQ$^+$ is capable of producing meaningful intermediate features during the active learning process.

6 Discussion

In the previous section, we demonstrate the outstanding performance of AGQ and AGQ$^+$, compared with the traditional pool-based active learning that only asks specific queries. However, one may be still concerned about some other issues of the proposed methods, such as: What if the oracle cannot provide accurate probability estimation for the generalized queries? What is the difference between AGQ (or AGQ$^+$) and active learning with feature selection? How does AGQ (or AGQ$^+$) perform with very few initial labeled examples? We will

	preg	plas	pres	skin	insu	mass	pedi	age	class
	Attribute Ranges Manually Discretized by [35]								
	$\{0-2\}$	$\{0-89.1\}$	$\{1-76.1\}$	$\{0-25\}$	$\{0-110\}$	$\{1-22.814\}$	$\{0-.244\}$	$\{21-24\}$	$\{0\}$
	$\{3-6\}$	$\{89.2-107.1\}$	$\{76.2.-98.1\}$	$\{26-32\}$	$\{111-150\}$	$\{22.815-26.84\}$	$\{.245-.525\}$	$\{25-30\}$	$\{1\}$
	$\{>7\}$	$\{107.2-123.1\}$	$\{>98.2\}$	$\{>33\}$	$\{151-240\}$	$\{26.841-33.55\}$	$\{.526-.805\}$	$\{31-40\}$	
		$\{123.2-143.1\}$			$\{>241\}$	$\{33.551-36.563\}$	$\{.806-1.11\}$	$\{41-55\}$	
		$\{143.2-165.1\}$				$\{>36.564\}$	$\{>1.11\}$	$\{>55\}$	
		$\{>165.2\}$							
	Typical Attribute Ranges for Individual Attributes Formed by AGQ$^+$								
	$\{0-2\}$	$\{37-77\}$	$\{6-55\}$	$\{0-25\}$	$\{0-133\}$	$\{20-27\}$	$\{.078-.488\}$	$\{24-30\}$	
	$\{3-5\}$	$\{47-53\}$	$\{12-22\}$	$\{16-26\}$	$\{48-132\}$	$\{25-32\}$	$\{.190-2.064\}$	$\{27-33\}$	
	$\{4-9\}$	$\{82-122\}$	$\{34-58\}$	$\{28-38\}$	$\{78-162\}$	$\{35-56\}$	$\{.261-.495\}$	$\{39-51\}$	
	$\{7-11\}$	$\{144-184\}$	$\{48-60\}$	$\{34-44\}$	$\{143-227\}$	$\{>36\}$	$\{.579-.813\}$	$\{42-54\}$	
	$\{>7\}$	$\{151-190\}$	$\{74-87\}$	$\{40-80\}$	$\{145-399\}$	$\{>40\}$	$\{.736-1.906\}$	$\{48-54\}$	
	$\{9-13\}$	$\{160-199\}$	$\{87-100\}$	$\{44-63\}$	$\{190-360\}$	$\{45-65\}$	$\{>0.897\}$		
	$\{11-13\}$	$\{>171\}$	$\{>94\}$	$\{>44\}$	$\{257-427\}$		$\{>.997\}$		
			$\{104-116\}$						
	Eight Typical Queries Produced by AGQ$^+$ (each row represents a query)								
1	$\{0-2\}$	$\{>171\}$	64	30	180	34.1	0.33	38	1(100%)
2	0	147	85	$\{>44\}$	0	$\{>39\}$	0.38	24	0(100%)
3	7	133	84	0	0	$\{>37\}$	$\{.579-.813\}$	37	1(85%)
4	0	188	82	$\{0-29\}$	$\{143-227\}$	32	0.68	22	1(100%)
5	$\{>7\}$	122	56	0	0	33.3	$\{>.997\}$	33	1(100%)
6	$\{4-9\}$	81	78	40	$\{0-132\}$	$\{>40\}$	0.26	42	0(100%)
7	6	$\{>173\}$	$\{87-100\}$	0	0	40.8	1.46	$\{39-51\}$	0(100%)
8	$\{>7\}$	119	$\{74-87\}$	35	0	$\{25-32\}$	0.26	29	1(60%)

TABLE 5
Typical attribute ranges and queries produced by AGQ$^+$ on "Diabetes".

answer these questions in this section. Note that, as the behavior of AGQ and AGQ$^+$ is mostly similar, we only consider AGQ in this section. All the conclusions are also applicable to AGQ$^+$.

6.1 Probability Estimation of Oracle

In this paper, we assume that for generalized queries, the oracle (or human expert) is capable of providing an accurate probability distribution. However, in real-world applications, it is common that the oracle or human experts can only provide "approximate answers" (i.e., estimated probability distributions). We speculate that small perturbations in probability distribution will not dramatically affect the performance of AGQ. This is because small perturbations in label probabilities only represent light noise of examples added in the training set. These light noises could be cancelled out in the successive updates of the training set. With a robust base learning algorithm (such as the bagged decision trees), such small noises would be insensitive. In this subsection, we study this issue experimentally.

We conduct experiments to compare the original AGQ and AGQ with inaccurate probability answers on the 14 UCI datasets (used in Section 5.2). To simulate inaccurate probability answers, we first calculate the exact probability answer as described in Section 3.3, and then randomly alter it with up to 10%, 20% and 50% noise (increase or decrease by up to 10%, 20% and 50% uniformly distributed random noise). All the other experimental configurations are the same as in Section 5.2.

Figure 4 plots the average error rates of AGQ, AGQ with 10% noise, AGQ with 20% noise, and AGQ with 50% noise, on a typical UCI dataset ("Hepatitis"). We can see that the error rates of AGQ with a low level of noise (10% and 20%) are similar to AGQ without noise, but AGQ with a high level of noise (50%) is significantly worse.

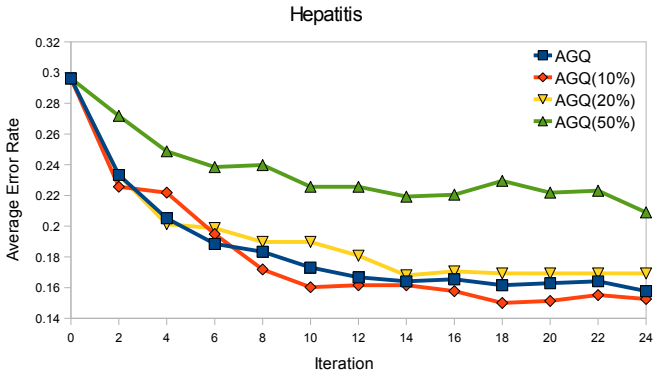

Fig. 4. Comparison of the average error rate between AGQ and AGQ with inaccurate probability answers (with 10%, 20%, and 50% noise respectively) on "Hepatitis".

Table 6 presents a summary of the t-test on the average error rates based on the 14 UCI datasets. Each entry in Table 6, $w/t/l$, means that the algorithm in the corresponding row wins on w, ties on t, and loses on l datasets, compared with the algorithm in the corresponding column. We can clearly see from Table 6 that AGQ with 10% noise is almost indistinguishable from the original AGQ (it ties with AGQ on 13 out of 14 datasets). AGQ with 20% noise is only slightly worse than AGQ without noise (it ties on 9 and loses on 5

datasets). However, AGQ with 50% noise is significantly worse (it loses on 12, and ties on 2 datasets). Clearly, high noise in the oracle answers will degrade the performance of AGQ, but low noise will not. Thus, AGQ is quite robust, and can tolerate a low level of noise in the probability distribution of oracle answers.

	AGQ (10%)	AGQ (20%)	AGQ (50%)
AGQ	1/13/0	5/9/0	12/2/0

TABLE 6
Summary of the t-test on the average error rates for comparing AGQ with AGQ (10% noise), AGQ (20% noise) and AGQ(50% noise).

6.2 AGQ vs. Feature Selection

One may notice that the essence of AGQ is to find irrelevant attributes, thus, it is closely related to feature selection. Features selection (e.g., [19], [25], [26]) attempts to discover and discard irrelevant attributes to improve the predictive accuracy. Thus, would it work if we simply apply the pool-based active learning (which only asks specific queries) after irrelevant attributes are discovered and discarded by feature selection? How does it compare with our AGQ? We study this issue in this subsection.

Indeed, a straightforward way to make the traditional pool-based active learning to ask generalized queries is to simply apply feature selection as a pre-processing step in active learning. That is, feature selection is conducted on the initial labeled training examples, to identify and eliminate all irrelevant features. Then traditional pool-based active learning is used to find the most uncertain specific query. Putting back the irrelevant features as the don't care, a generalized query is produced. However, as we are usually given only a small number of initial labeled examples in active learning, feature selection is most likely to be unreliable, thus eliminating too many relevant features in the pre-processing step.

A more sophisticated and improved method is to conduct feature selection in each active learning iteration. More specifically, in each iteration, active learning selects the most uncertain example based on the current learning model, and at the same time, feature selection identifies the irrelevant features based on the current labeled examples. Then, a generalized query can be constructed by substituting all irrelevant features as $*$ in the most uncertain example. Such generalized queries are asked to the oracle, and the labeled training set is updated, as in AGQ. In essence, irrelevant attributes are identified by feature selection, instead of using the method described in AGQ (Section 3.2).

We implement this feature selection active learning method, and compare its performance with the proposed AGQ on the same 14 UCI datasets. More specifically, we use the backward selection method to select irrelevant features, and use a bagging of 100 decision trees (the same base learning algorithm used in AGQ) as the classifier to evaluate them. All the other experimental configurations are the same as in Section 5.

Figure 5 plots the average error rates of AGQ and the pool-based active learning with feature selection (called "Feature Selection") on a typical UCI dataset ("Hepatitis"). We can see clearly that AGQ performs significantly better than "Feature Selection". We also perform the t-test on the average error rates on the 14 UCI datasets. AGQ wins on 12, and loses only on 2 datasets, compared with "Feature Selection". This clearly indicates that "Feature Selection" performs significantly worse than the proposed AGQ in most cases.

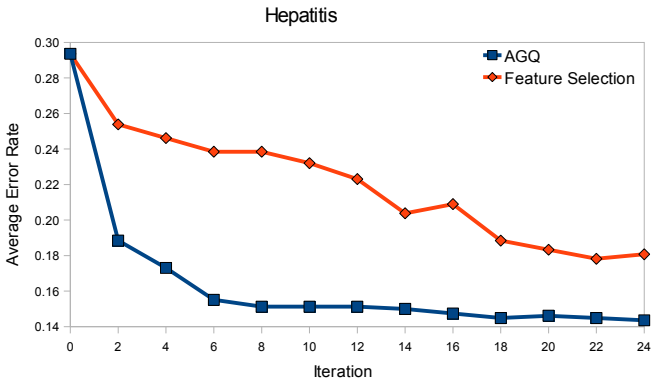

Fig. 5. Comparison of the average error rate between AGQ and Feature Selection on "Hepatitis".

After looking into the queries produced by "Feature Selection" and the corresponding oracle answers, we find the reason. The generalized queries constructed by "Feature Selection" are often overly general. More specifically, even though "Feature Selection" identifies the irrelevant attributes (and generalizes them as $*$) in each iteration, the available labeled training examples in each iteration are still limited (especially in the first few iterations). Thus "Feature Selection" tends to identify more attributes as irrelevant, and constructs overly generalized queries. Consequently, the oracle could only provide uncertain answers to these overly general queries, thus degrading the active learning performance.

On the other hand, AGQ has a more strict criterion to identify irrelevant attributes, compared with "Feature Selection". More specifically, only when all generated examples with different attribute values have very close class probability estimation (instead of the same class prediction in "Feature Selection"), the current attribute could be regarded as irrelevant. (See Section 3.2 for details.) Thus, the over general queries would not frequently occur in AGQ (see Table 1 in Section 5.1).

Of course, AGQ$^+$ can ask generalized queries with subsets of nominal attribute values, or ranges of numeric attributes. "Feature Selection" will not be able to produce this type of queries. Thus, AGQ$^+$ is inherently more powerful than "Feature Selection".

6.3 AGQ with Very Few Initial Labeled Examples

In the previous sections, we mentioned that when the initial training set is very small, the constructed learning model could be unreliable, thus the discovered don't attributes could be unreliable as well. Indeed, with the limited information from few labeled examples, it is difficult (or even impossible) to correctly identify the don't-care attributes. Thus, in this subsection, we study the performance of AGQ with very few initial labeled examples.

Given very few initial labeled examples, the original AGQ is more likely to consider many attributes as don't-care, and construct overly general queries. With the uncertain answers from the oracle, these overly general queries can severely degrade the performance of AGQ. Here, we design an additional heuristic to deal with this issue. Roughly speaking, for each query, we bond the number of don't-care attributes to the size of the current training set. When the training set is small, only a small number of attributes could be considered as don't-care, due to the limited information provided by the labeled examples. On the other hand, when the training set is relatively large, more don't-care attributes are allowed to be discovered, as more reliable information are provided.

Specifically, when constructing the generalized query (as in Section 3.2), we add the current attribute into the don't-care attribute list (when all the other conditions are satisfied), only if the number of all don't-care attribute value combinations is smaller than (or equal to) the current training set size. For example, given only two labeled training examples, at most one binary attribute could be considered as don't-care; given four labeled training examples, at most two binary attributes (or one four-value attribute) could be considered as don't-care; and so on.

We implement this heuristic on the base of the original AGQ algorithm, and compare its performance with "Pool" on the same 14 UCI datasets. All the experimental configurations are the same as in Section 5, except we only include two labeled examples (one positive and one negative) in the initial training sets.

Figure 6 plots the average error rates of AGQ and "Pool" on a typical UCI dataset ("Hepatitis"). We can see that AGQ still performs significantly better than "Pool" even with only two initial labeled examples.

In addition, the t-test of the average error rates on all the 14 UCI datasets shows that, AGQ wins on 8, ties on 6, and loses on 0 dataset, compared with "Pool". This also clearly indicates the similar conclusion as in the previous subsection: the error rate of AGQ still decreases much faster than "Pool", even with only few initial labeled examples.

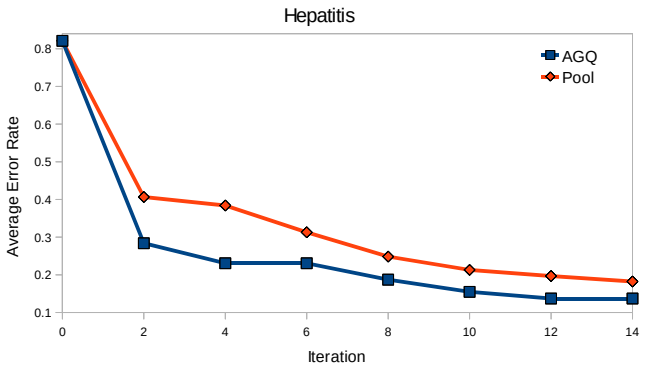

Fig. 6. Comparison of the average error rate between AGQ and "Pool" on "Hepatitis", with two initial labeled examples.

7 Conclusions and Future Works

Domain-driven data mining calls for domain experts involvement in the data mining process. Active learning involves domain experts in its need to obtain label information for the queries. However, previous active learning algorithms assume that the oracle can only answer specific queries that represent single examples. However, in real-world applications, the domain experts are often more readily to answer "generalized queries" with don't-care attributes and generalized attributes (such as subsets or ranges of values). Answers to such generalized queries can provide more information to improve learning. The difficulty of generalized queries is that the answers from the oracle can be uncertain, thus noisy labels might be introduced and performance might be degraded. This easily happens especially when the initial labeled training set is small. In this paper, we propose a novel active learning algorithm (AGQ) to ask as general queries as possible with still highly certain labels. AGQ is then extended to AGQ$^+$, which can produce subsets of nominal attribute values or ranges of numeric attributes. AGQ thus becomes a special case of AGQ$^+$. Our experiments show that, compared with the traditional pool-based active learning, AGQ can achieve the same error rates with significantly fewer queries (36% fewer on average). We also show that AGQ's performance is similar to the (unrealistic) optimal AGQ. AGQ works well even with only two labeled examples in the initial training set. In addition, our experiments verify the robustness of the proposed algorithm: AGQ with inaccurate answers from the oracle (up to 20% perturbation) still performs comparably to the original AGQ on most tested UCI datasets. AGQ can be readily deployed in real-world data mining tasks where obtaining labeled examples is costly.

In our future research, we will study the performance of AGQ with different base learning algorithms (we only use the bagging of decision trees in this paper). Strategies for dealing with highly uncertain answers from the oracle, and for preventing dramatic changes of data distribution when new examples are included in the training set are also interesting research issues to further improve the performance of AGQ.

ACKNOWLEDGMENT

The authors acknowledge the valuable assistance of other members of the Data Mining and E-Business Lab of The University of Western Ontario in this research.

REFERENCES

[1] N. Abe and H. Mamitsuka. Query learning strategies using boosting and bagging. In *ICML '98: Proceedings of the Fifteenth International Conference on Machine Learning*, pages 1–9, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[2] C. C. Aggarwal. Towards effective and interpretable data mining by visual interaction. In *In SIGKDD Explorations*, volume 3, pages 11–22, 2002.

[3] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In P. Buneman and S. Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, Washington, D.C., FebruaryJune–FebruaryAugust˜ 1993.

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

[5] M. Ankerst. Report on the sigkdd-2002 panel the perfect data mining tool: interactive or automated? *SIGKDD Explor. Newsl.*, 4(2):110–111, 2002.

[6] A. Asuncion and D. J. Newman. UCI machine learning repository [http://www.ics.uci.edu/˜mlearn/mlrepository.html], 2007.

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

[8] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[9] L. Cao. Introduction to domain driven data mining. In *Data Mining for Business Applications*, pages 3–10. 2009.

[10] L. Cao and C. Zhang. Domain-driven data mining: A practical methodology. *International Journal of Data Warehousing and Mining*, 2(4):49–65, 2006.

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

[12] D. A. Cohn, Z. Ghahramani, and M. I. Jordan. Active learning with statistical models. *Journal of Artificial Intelligence Research*, 4:129–145, 1996.

[13] W. Dai, Q. Yang, G. R. Xue, and Y. Yu. Boosting for transfer learning. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 193–200, New York, NY, USA, 2007. ACM.

[14] L. S. Dasgupta. Coarse sample complexity bounds for active learning. In *Neural Information Processing Systems*, 2005.

[15] G. Druck, G. S. Mann, and A. McCallum. Learning from labeled features using generalized expectation criteria. In S. H. Myaeng, D. W. Oard, F. Sebastiani, T. S. Chua, M. K. Leong, S. H. Myaeng, D. W. Oard, F. Sebastiani, T. S. Chua, and M. K. Leong, editors, *SIGIR*, pages 595–602. ACM, 2008.

[16] J. Du and C. X. Ling. Active learning with generalized queries. In *Proceedings of the 2009 IEEE International Conference on Data Mining (to appear)*, 2009.

[17] C. Elkan. The foundations of cost-sensitive learning. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 973–978, 2001.

[18] Y. Guo and D. Schuurmans. Discriminative batch mode active learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 593–600, Cambridge, MA, 2008. MIT Press.

[19] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, March 2003.

[20] S. C. H. Hoi, R. Jin, J. Zhu, and M. R. Lyu. Batch mode active learning and its application to medical image classification. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 417–424, New York, NY, USA, 2006. ACM.

[21] D. D. Lewis and J. Catlett. Heterogeneous uncertainty sampling for supervised learning. In W. W. Cohen and H. Hirsh, editors, *Proceedings of ICML-94, 11th International Conference on Machine Learning*, pages 148–156, New Brunswick, US, 1994. Morgan Kaufmann Publishers, San Francisco, US.

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

[23] C. X. Ling and J. Du. Active learning with direct query construction. In *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 480–487, New York, NY, USA, 2008. ACM.

[24] B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *Knowledge Discovery and Data Mining*, pages 80–86, 1998.

[25] H. Liu and H. Motoda. *Feature Selection for Knowledge Discovery and Data Mining*. Kluwer Academic Publishers, 1998.

[26] H. Liu and H. Motoda, editors. *Computational Methods of Feature Selection*. Data Mining and Knowledge Discovery. Chapman & Hall/CRC, 1 edition, October 2007.

[27] O. Longbing Ca and G. Chengqi Zhan. The evolution of kdd: Towards domain-driven data mining. *International Journal of Pattern Recognition and Artificial Intelligence (IJPRAI)*, 21(4):677–692, 2007.

[28] D. D. Margineantu. Active cost-sensitive learning. In *the Nineteenth International Joint Conference on Artificial Intelligence*, Edinburgh, Scotland, 2005.

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

[30] R. Raina, A. Battle, H. Lee, B. Packer, and A. Y. Ng. Self-taught learning: transfer learning from unlabeled data. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 759–766, New York, NY, USA, 2007. ACM.

[31] 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.

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

[33] 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.

[34] V. S. Sheng and C. X. Ling. Thresholding for making classifiers cost-sensitive. In *Proceedings of the Twenty-first National Conference on Artificial Intelligence (AAAI-06)*, 2006.

[35] J. W. Smith, J. E. Everhart, W. C. Dickson, W. C. Knowler, and R. S. Johannes. Using the adap learning algorithm to forecast the onset of diabetes mellitus. In *Proceedings of the Symposium on Computer Applications and Medical Care*, pages 261–265. IEEE Computer Society Press, 1988.

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

[37] 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.

[38] P. Yu, editor. *DDDM '07: Proceedings of the 2007 international workshop on Domain driven data mining*. ACM, New York, NY, USA, 2007.

[39] 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.