Dynamic Active Probing of Helpdesk Databases

Shenghuo Zhu
NEC Labs America
zsh@sv.nec-labs.com

Tao Li*
Florida International University
taoli@cs.fiu.edu

Zhiyuan Chen
University of Maryland
Baltimore County
zhchen@umbc.edu

Dingding Wang
Florida International University
dwang003@cs.fiu.edu

Yihong Gong
NEC Labs America
ygong@sv.nec-labs.com

ABSTRACT

Helpdesk databases are used to store past interactions between customers and companies to improve customer service quality. One common scenario of using helpdesk database is to find whether recommendations exist given a new problem from a customer. However, customers often provide incomplete or even inaccurate information. Manually preparing a list of clarification questions does not work for large databases. This paper investigates the problem of automatic generation of a minimal number of questions to reach an appropriate recommendation. This paper proposes a novel dynamic active probing method. Compared to other alternatives such as decision tree and case-based reasoning, this method has two distinctive features. First, it *actively* probe the customer to get useful information to reach the recommendation, and the information provided by customer will be immediately used by the method to *dynamically* generate the next questions to probe. This feature ensures that all available information from the customer is used. Second, this method is based on a probabilistic model, and uses a data augmentation method which avoids overfitting when estimating the probabilities in the model. This feature ensures that the method is robust to databases that are incomplete or contain errors. Experimental results verify the effectiveness of our approach.

1. INTRODUCTION

High quality customer service is extremely important. According to a survey conducted by National Retail Federation, 99 percent of shoppers said that customer service was at least somewhat important when deciding to make a purchase [35]. Databases have been widely used to store past interactions between customers and companies to improve the quality of customer service. Such databases are called *helpdesk* databases.

In this paper, we assume the helpdesk databases are organized into a number of cases, where each case contains the interactions between a customer and the service team about a particular problem, the features extracted from these interactions (e.g., initial information provided by the customer, clarification questions asked by the service team and answers from the customer, etc.), and the final recommendations by the service team.

Table 1 shows an example database for a computer support service (not all the cases are shown). Features with question marks are questions being asked by the service team and users' answers are in the parenthesis. For instance, in the first case, user reports a login problem for an on-campus computer. The service team then asks whether the user has enrolled in CS courses because the account is only available for enrolled students. The user answers "yes" and the team confirms enrollment and enables the account.

A question can be modeled as a feature and answers to the question can be modeled as values of the feature. This paper assumes features are already extracted either manually or using natural language processing techniques [5].

One common scenario of using helpdesk database is to find whether recommendations exist given a new problem from a customer. However, directly matching the request to existing cases often does not work because customers often provide incomplete information. For example, if the user just says he/she has a login problem, all of the first four cases in Table 1 will match. Thus some clarification questions (e.g., whether the user is on-campus) need to be asked. These questions can be prepared manually. However, if the helpdesk database is very large or gets updated frequently, it will be difficult to manually generate and maintain the questions. This paper investigates the problem of automatic generation of a minimal number of questions to reach an appropriate recommendation.

There has been a rich body of work on searching databases, such as ranking data query results [3, 12, 2], similarity search [1, 27], evaluating top k queries [13, 22], and computing skyline queries [7, 31]. However, the focus of this paper is not to find the answers of a user query, but to find the appropriate questions to ask the user.

This paper models the problem as a classification problem (i.e., classifying a user's problem into an existing recommendation). The problem of question generation now becomes the problem of selecting a minimal number of features (questions) that can accurately predict the recommendation, based on known features of the new problem and existing cases. We also have two requirements for the solution:

1. It needs to utilize all the information provided by users.
2. It needs to be robust to databases that are incomplete (i.e., some important cases may not be in the database) or contain errors.

The first requirement ensures that the prediction is accurate. The

*Partially supported by NSF grant IIS-0546280.

VLDB '08, August 24-30, 2008, Auckland, New Zealand

Table 1: Example helpdesk database

Case ID	Features	Recommendation
1	login problem, on campus? (yes), enrolled in CS courses? (yes)	Confirm enrollment and enable account
2	login problem, on campus? (yes), enrolled in CS courses? (no)	User has no access to the account
3	login problem, on campus? (no), network connection ok? (yes), VPN running? (no)	Install and run VPN
4	login problem, on campus? (no), network connection ok? (yes), VPN running? (yes), enrolled in CS courses (yes)	Confirm enrollment and enable account
...	...	...

Table 2: Example new cases

Case ID	Known Features	Unknown Features	Correct Recommendation
5	login problem, on campus (yes), enrolled in CS courses (no)		User has no access to the account
6	login problem, on campus (no)	network connection ok (yes), VPN running (yes), enrolled in CS courses (no)	User has no access to the account

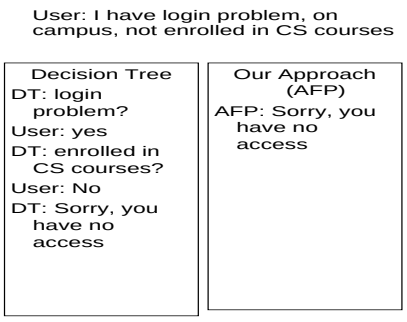

Figure 1: Decision Tree and Our Approach on Case 5

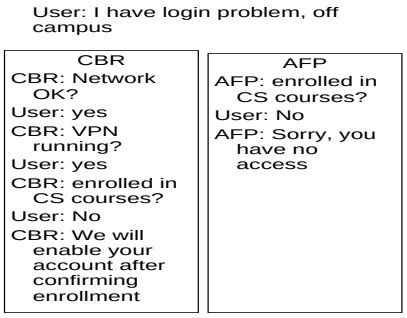

Figure 2: Case-based Reasoning and Our Approach on case 6

second requirement is needed because it is often very expensive to clean up a helpdesk database due to its size and frequent updates.

Problem of existing approaches: There are two existing approaches: decision tree [8, 37] or case-based reasoning [36]. However neither approach satisfies both requirements. Next we use two examples to illustrate the limitations of the two existing approaches.

Table 2 shows two new cases (case 5 and 6) from a user. For each case, we assume that the user has provided some known features and there are still some unknown features. Suppose we build a decision tree over existing cases in helpdesk database shown in Table 1. Each node in the tree will become a question to ask the user (starting from the root). Figure 1 shows the interactions between the decision tree approach and the user. Since the decision tree is built statically, it can not use the information provided by user at run time. Thus although the user has already told the system that he/she has a login problem, is on campus, and has not enrolled, the decision tree approach still asks two unnecessary questions.

One could certainly build a decision tree dynamically, e.g., to build the tree only over cases that match the information provided by a user. However, the following example will show that the decision tree approach does not satisfy the second requirement.

Consider case 6 in Table 2, where a user tells us he/she has a login problem and is off campus. Case 3 and 4 in the database match the known features. However, the correct recommendation is not in either of these cases (it is in case 2). Thus building a decision tree over the exact matching cases will never give the correct solution. One could try to alleviate this problem by including partially matched cases such as case 2. However it is unclear how to take into account the different degree of similarity between the matching cases and the new problem when building the decision tree.

Another possible solution is to use case-based reasoning (CBR) systems [16, 4, 9, 34, 29, 20]. A well known example is the QuickSource system that has been used by Compaq to support its helpdesk [36]. Such a system retrieves cases most similar to the new problem and returns the recommendation in the most similar cases[1]. The user may provide some initial information. The systems will use it to retrieve similar cases and then ask the user additional questions (typically questions in the matching cases) to reduce the number of matching cases. This process typically stops when the user finds the case with the appropriate solution or only one case remains. Thus case-based reasoning approach satisfies the first requirement on utilizing user provided information.

However, CBR does not satisfy the second requirement because it often overfits its solution to the few most similar cases in the database. For instance, consider case 6 in Table 2. Figure 2 shows the interactions between CBR and the user for this case. The user tells us he/she has a login problem and is off campus. Hence CBR finds case 3 and 4 most similar based on known features. It then asks some questions. Eventually it learns all the features of case 6. However, there is no exact match in the database for all features. Suppose the most similar case is case 4 (it matches 4 out of 5 features of case 6), CBR will return the solution to case 4, which is incorrect because the user is not enrolled.

One could certainly try a better similarity function. However, CBR is still vulnerable if the most similar case itself contains errors (e.g., if case 2 is found to be the most similar case, but its recommendation is wrongly put as the solution to case 1).

Our approach: This paper proposes a novel approach called *active feature probing* (AFP). Figure 3 shows the architecture of our

[1]In applications such as legal systems and product design, case-based systems also adapt or combine the solutions in the retrieved cases (e.g., adjusting length of sentences depending on severity of a crime or combine two different product designs). However, for helpdesk applications, it is unclear how to adapt or combine recommendations.

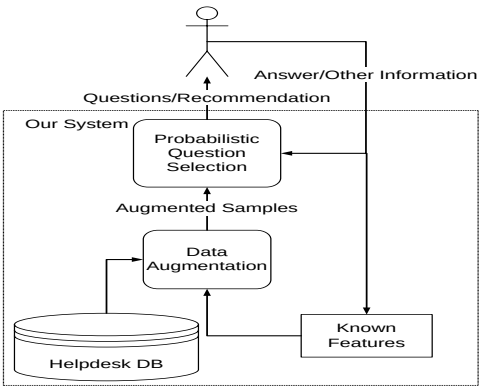

Figure 3: System architecture.

proposed approach. Our approach keeps a set of known features of the new case. Whenever a user provides the system any new information, it will be immediately added to the known features. This will trigger a probabilistic question selection module to generate a ranked list of questions and recommendations. Questions are ranked based on how informative they are given the known features, and recommendations are ranked based on their probabilities given the known features. The system will then actively probe the user on those questions. The user can answer any of those questions, or even provide information that is not being asked. The system then adds these information into known features and generates the next set of questions and recommendations. This process continues until the user finds the appropriate recommendation.

It is easy to estimate how informative a question is if all features of the new case is known. However usually many features are unknown. One solution is to draw samples directly from existing cases that match the known features, and use these samples for estimation (similar to building a decision tree over matching cases). However, as mentioned earlier, there are often too few existing cases that match the known features. A small sample size often leads to poor estimation.

To overcome the small sample problem, our approach uses a data augmentation method [18, 42] to generate a set of *augmented* samples based on the known features and the helpdesk database. These samples are called augmented because they have the same values as the new case on the known features, and have synthetically generated values on the unknown features of the new case. The generation of unknown feature values considers both the distribution of unknown features of existing cases and the chances of those cases will match the known features. A simple way of understanding this approach is that it associates some weights to cases based on their chances of matching the case. The benefit of the sampling approach is that we can generate as many sample cases as we want, and the estimation based on these samples is unbiased.

Compared to existing solutions, our AFP approach has two distinctive features. First, it *actively* probe the customer to get useful information to reach the recommendation, and the information provided by customer will be immediately used by the method to *dynamically* generate the next questions to probe. This feature ensures that all available information from the customer is used. Second, this method is based on a probabilistic model, and uses a data augmentation method which avoids overfitting when estimating the probabilities in the model. This feature ensures that the method is robust to databases that are incomplete or contain errors. Thus our approach satisfies the two requirements mentioned earlier.

Figure 1 shows the interactions between our approach (AFP) and the user for case 5. AFP first adds user provided information into known features. This will trigger the probabilistic module. The module will compute the probabilities of different recommendations given the known features. It finds that the recommendation in case 2 (the correct one) has very high probability and will presents it to the user. Thus no question needs to be asked by our approach (Decision Tree approach asks 2 questions). This example shows the value of the dynamic and active feature of our approach.

Figure 2 shows the interactions between our approach (AFP) and the user for case 6. Again, the known features will be added, which will trigger the probabilistic module. This module will generate sufficient number (we use 300 in this paper) of augmented samples. Note that case 2 also partially matches case 6, thus it will be also considered when generating the samples. This avoids overfitting the samples to the most similar cases (case 3 and 4). The probabilistic module then selects the most informative question (the enrollment status) to probe the user. Based on the user's answer (not enrolled), the module finds that the recommendation in case 2 (the correct one) is the most probable and returns it to the user. Thus our approach not only asks fewer questions than CBR, but also returns the correct solution while CBR does not. This is due to the probabilistic feature of our approach.

Even if there are errors in individual cases, our approach may still return the correct recommendation as long as these errors only occur in small number of cases. This is different from CBR where the error in a single case may lead to a wrong solution if that case happens to be the most similar one to the new case.

This paper has made the following contributions:

- We propose the problem of active probing for helpdesk databases and model it as a classification problem.
- We propose a solution to this problem. Unlike existing approaches, this solution satisfies both of the two requirements.
- We conduct experiments to verify the effectiveness of our approach.

The remainder of the paper is organized as follows. Section 2 reviews the related work. Section 3 gives an overview of our approach. Section 4 describes the probabilistic question selection method. Section 5 describes the data augmentation method. Section 6 reports experimental results. Section 7 concludes the paper and discusses directions for future work.

2. RELATED WORK

In this section, we review the following related topics.

Decision Tree: As mentioned in Section 1, decision trees [8, 37] could be used to select the most informative feature to probe. The most informative feature is the one with the largest information gain in C4.5[37], the largest Gini index in CART[8], or other similar measures. However, as mentioned in Section 1, decision trees must follow a strict decision path, and are unable to handle detour in the decision path, in other words, they cannot utilize the "volunteered" information. Further, as mentioned in Section 1, building a decision tree over matching cases may not work well when there are very few cases matching the new problem.

Query Form Generation: Jayapandian et al. [28] proposed a method to automatically generate query forms based on query workloads. The idea is to create forms that can answer many similar queries. However, this paper focuses on generating the minimal number of questions to reach a recommendation.

Query Expansion: The active feature probing is also related to query expansion [41, 44]. Query expansion is a solution to the problem of word mismatch in information retrieval. When a query

returns very few results, query expansion reformulates the query by adding additional related keywords in order to obtain more results. Some work uses mutual information between the input keyword and the expanding keywords [23]. For some difficult queries, Kumaran and Allan [32] ask simple questions to gather more information about the query from the user. The questions are pre-defined topics, bi-grams, or phrases. Though, both query expansion and active feature probing ask questions to gather information, their goals are different. The goal of the query expansion is to retrieve relevant documents when no query keywords exactly match the content of documents. However, the goal of the active feature probing is to reduce the uncertainty of prediction.

Case-based Systems: There has been a rich body of work on incremental/conversational case-based reasoning [16, 4], case-based recommender systems [9, 34, 29], and decision guides [20] where the user can only provide a brief initial description of problems or items. As mentioned in Section 1, CBR has an overfitting problem and is not robust to databases that are incomplete or contain errors. Further, when the description of cases or items becomes complicated, CBR suffers from *curse of dimensionality* and the similarity/distance between cases or items becomes difficult to measure [6]. Our work is more robust because it is probability based. We have conducted an empirical study to show the advantages of our solution in Section 6.

Active Learning: The problem of active learning is to gather the labels of data points in order to improve the classification accuracy. Recently, researchers have also applied active learning to problem diagnosis in distributed systems using test transactions [10]. Though both goals are to improve the classification accuracy [40, 15], the dynamic active probing is to probe unknown feature to gather information, while the active learning is to probe unlabeled data to gather information.

Database search and ranking: Many methods have been proposed to rank results of a SQL query [3, 12, 2]. Similarity search has also been studied in [1, 27]. There has also been work on how to efficiently return the top k results rather than all results [13, 22], and efficiently return results of skyline queries [7, 31]. However, the focus of this paper is not to find the answers of a user query, but to find the appropriate questions to ask the user.

Data Augmentation: Data augmentation is a technique for solving missing value problems (see [18, 42]). The basic idea is to fill the unknown features with certain values, then we can perform further analysis as if the data were complete. In this paper, we augment an instance to multiple factitious instances, which is also known as multiple imputation [38] in statistics literature. Instead of setting the features to the most likely values as the EM algorithm does [18], multiple imputation sets the features to values drawing from the distribution of the data. In our work, we assume the classifier and the data model can be learned from the existing training data.

Attribute/Feature Selection: A lot of work has been done on attribute/feature selection in machine learning community [33, 30] and many selection techniques have also been applied to dynamic dialogs [39] for question selection in recommender or diagnosis systems. The goal of attribute/feature selection is to reduce the number of attributes/features used to describe the input in order to improve the prediction or classification over all input instances. However, our active feature probing is to probe the most informative features of a given instance in order to gather information.

Natural Language Processing: There has been a rich body of work on natural language processing [5]. The most related is the information extraction techniques. Two excellent surveys can be found at [25] and [11]. Our approach can use such techniques to help extract the features automatically from existing cases. However, our focus is on generating minimal number of questions to find a recommendation.

Table 3: Symbols

$\mathcal{D}$	Helpdesk database
$\mathcal{Y}$	Set of recommendations (class labels)
y	A recommendation (class label)
$\mathcal{X}$	Feature space
d	Number of features
$\mathcal{X}_i$	Domain of i-th feature
$\mathbf{x}$	a case
x_i	value of the i-th feature of case x
$\mathbf{obs}$	Set of indexes of features whose values are known
$\mathbf{un}$	Set of indexes of features whose values are unknown
$\mathbf{x_{obs}}$	The known features of $\mathbf{x}$
$\mathbf{x_{un}}$	The unknown part of $\mathbf{x}$
H	Entropy
I	Mutual information
$\mathcal{S}$	Set of augmented instances
z	an existing case in $\mathcal{D}$
z_i	the i-th feature of existing case z

Faceted Search: Faceted search has been used to allow user to browse the results of a query [21, 45, 43]. It creates a set of category hierarchies, each of which corresponds to a different facet (or dimension). A user can choose a facet to browse and navigate the results of a query. Our approach bears some similarity to the faceted search. Each question being asked can be seen as a category in the hierarchy and all questions can be seen as forming a hierarchy of categories. However, the goal of the faceted search is to address the problem of having too many results for a query. The goal of this paper is on helping users to find the most appropriate recommendation. As a result, the criteria of selecting categories in faceted search is often based on relationship between terms [21, 45] or interestingness of results under a category [43] (measured as deviation from parent category). This paper selects the clarification question to minimize the uncertainty in possible recommendations.

3. OVERVIEW OF OUR APPROACH

We first introduce some notations. Table 3 lists the symbols we use in this paper. Let $\mathcal{Y}$ be the set of classes, and $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_d$ be the d-dimensional feature space. Let $\mathbf{x}$ be a case where $\mathbf{x} = (x_1, \ldots, x_d)$ where x_i is the value of the i-th feature of $\mathbf{x}$.

DEFINITION 1. *A Helpdesk database $D = \{(\mathbf{x}, y)\}$ where $\mathbf{x}$ is a case and $y \in \mathcal{Y}$ is the class label of $\mathbf{x}$.*

In the feature space, we use *NA* to represent an unknown feature value. For simplicity, in this study we assume all feature variables are binary. It is easy to generalize our methods to categorical data because a categorical attribute can be represented as a number of binary attributes, one for each possible value. For features with continuous values, we can discretize them first [26].

For a new case $\mathbf{x} = (x_1, \cdots, x_d) \in \mathcal{X}$ from user, let $\mathbf{obs}$ be the set of indexes whose corresponding feature values are currently known, and $\mathbf{un}$ be the set of the remainder indexes. $\mathbf{x_{obs}}$ represents the known part of $\mathbf{x}$, and $\mathbf{x_{un}}$ represents the unknown part of $\mathbf{x}$.

Algorithm 1 shows the sketch of our approach. Given the known features of the new case and the helpdesk database, a set of aug-

mented samples is generated. Using these samples, the probabilistic question selection module shown in Figure 3 ranks unknown features based on how informative they are to help classify the new case. It also ranks the recommendations based on their probabilities given the known features. The module then presents to the user the top k ranked questions and recommendations. The user can answer any of those questions (not necessary all of them), or provide additional information. The new information will be then added to known features, which will trigger the next round of ranking. The process stops when the probability of one of the recommendations exceeds threshold t, or the user has selected a recommendation. Next Section 4 gives details of the probabilistic question selection step and Section 5 describes the data augmentation step.

Algorithm 1 Active Feature Probing (AFP) Algorithm

Input: $\mathbf{x_{obs}}$, $\mathcal{D}$, a confidence threshold t, a value k.

1: **repeat**
2: Given $\mathbf{x_{obs}}$, draw a set of augmented samples, denoted by $\mathcal{S}$;
3: Use $\mathcal{S}$ to rank unknown features based on how informative they are;
4: Rank recommendations based on their probabilities given the known features (i.e. $Pr(y|\mathbf{x_{obs}})$);
5: Present the k top ranked features (questions) and recommendations to user;
6: User answers some of the questions or provides new information;
7: Update $\mathbf{obs}$, $\mathbf{un}$ and $\mathbf{x_{obs}}$ with information from user;
8: **until** $\max_y Pr(y|\mathbf{x_{obs}}) \geq t$ or the user selects some recommendation.

4. PROBABILISTIC QUESTION SELECTION

This section describes the probabilistic question selection method. Section 4.1 presents the probabilistic model and Section 4.2 describes how to estimate this model.

4.1 Probabilistic Model

To estimate how informative an unknown feature is to help classify a new case $\mathbf{x}$, we try to measure the uncertainty (entropy) of the class label y given known features of $\mathbf{x}$.

$$H(y|\mathbf{x_{obs}}) \stackrel{\text{def}}{=} -\sum_{y\in\mathcal{Y}} \Pr(y|\mathbf{x_{obs}}) \log \Pr(y|\mathbf{x_{obs}})^2. \quad (1)$$

We want to find an optimal set *opt* of unknown features that minimizes uncertainty of the class label, i.e., to find $opt \subseteq \mathbf{un}$ such that it minimizes $H(y|\mathbf{x_{obs}}, \mathbf{x}_{opt})$. However, this problem can be proved to be NP hard in a way similar to proving finding an optimal decision tree with a certain average length is NP hard.

Further, this set is only optimal given the known features and may become suboptimal given new user feedback. For example, for case 6 in Table 2, initially, the optimal set of features may consist of questions both on network and his enrollment status. Thus two questions need to be asked. However, if the system asks the question on enrollment status and the user says that he is not enrolled, then the system can immediately find the solution (the account is not accessible). Thus only one question is actually needed. Thus in this paper we use a Greedy heuristic to select the set of questions to probe and will re-select the questions whenever user provides some new information.

For each unknown feature $i \in \mathbf{un}$, we first compute the uncertainty of y after probing x_i, i.e., $H(y|\mathbf{x_{obs}}, x_i)$. We then rank all unknown features in ascending order of $H(y|\mathbf{x_{obs}}, x_i)$, and returns the top k ranked features.

Now the problem becomes to estimate $H(y|\mathbf{x_{obs}}, x_i)$. The main challenge is that the value of x_i is unknown before the probing of feature i. Thus we can not calculate $H(y|\mathbf{x_{obs}}, x_i)$. To overcome the problem, we use the estimated average entropy instead.

THEOREM 1. *The expectation of $H(y|\mathbf{x_{obs}}, x_i)$ over the distribution of x_i is $\mathbb{E}_{x_i|\mathbf{x_{obs}}}\{H(y|\mathbf{x_{obs}}, x_i)\} = H(y|\mathbf{x_{obs}}) - I(x_i; y|\mathbf{x_{obs}})$, where $I(x_i; y|\mathbf{x_{obs}})$ is the mutual information between x_i and y given the known values of $\mathbf{x_{obs}}$, which is defined as*

$$I(x_i; y|\mathbf{x_{obs}}) \stackrel{def}{=} H(y|\mathbf{x_{obs}}) + \sum_{\substack{x_i\in\mathcal{X}_i \\ y\in\mathcal{Y}}} \Pr(x_i, y|\mathbf{x_{obs}}) \log \Pr(y|\mathbf{x_{obs}}, x_i).$$

The proof can be found in the appendix. $H(y|\mathbf{x_{obs}})$ is fixed as we choose x_i to probe. Therefore, to minimize the expected entropy, we need to find the x_i with the highest $I(x_i; y|\mathbf{x_{obs}})$, i.e., with the highest mutual information with class label y given observed features of $\mathbf{x}$. This is similar to maximizing information gain in decision tree building algorithms [37]. The main difference though, is that we compute this mutual information given a set of known feature values ($\mathbf{x_{obs}}$).

4.2 Estimating the Model

The main challenge to estimate $I(x_i; y|\mathbf{x_{obs}})$ is that feature x_i is unknown for the new case $\mathbf{x}$. One possible solution is to generate a sample $\mathcal{S}$ directly using the existing cases that are matched with observed features of $\mathbf{x}$, and then use this sample to estimate the mutual information. However, when there are many observed features, the number of matched cases can be very limited. For example, consider case 6 in Table 2 and the database shown in Table 1. Only case 3 and 4 directly match the known features of case 6, and none of these cases have the same values as case 6 on enrollment status feature. Thus the sample generated will not be very useful to estimate how informative enrollment status is. In general, when there are very few matching cases, the prediction will have large variance. In addition, when there are errors in the observed feature values, the right cases may not be found. Therefore, this method could lead to data sparsity issue and low noise tolerance.

We take a data augmentation approach [18, 42, 24]. This approach generates a sufficient number of augmented instances. The generated sample has the property that it will have the same mutual information as the mutual information we want to estimate. Section 5 will discuss the details of the data augmentation step.

Given the augmented sample $\mathcal{S}$, next we show how to estimate $I(x_i; y|\mathbf{x_{obs}})$ using $\mathcal{S}$. Let $\tilde{I}(x_i; y|\mathcal{S})$ be the estimation of mutual information between x_i and y using sample $\mathcal{S}$. Let $\tilde{p}(E; \mathcal{S})$ be the estimation of probability of event E using sample $\mathcal{S}$. The conditional mutual information, $I(x_i; y|\mathbf{x_{obs}})$, can be approximated by the mutual information over sample, $\tilde{I}(x_i; y|\mathcal{S})$.

$$\tilde{I}(x_i; y|\mathcal{S}) \stackrel{\text{def}}{=} \sum_{x_i, y} \tilde{p}(x_i, y; \mathcal{S}) \log \tilde{p}(x_i, y; \mathcal{S}) - \sum_{x_i} \tilde{p}(x_i; \mathcal{S}) \log \tilde{p}(x_i; \mathcal{S}) - \sum_{y} \tilde{p}(y; \mathcal{S}) \log \tilde{p}(y; \mathcal{S}) \quad (2)$$

[2]Following the literature, if a probability is zero, we define $0 \log 0 = 0$, i.e., that probability has no impact on entropy.

The probabilities $\tilde{p}(x_i, y; \mathcal{S})$, $\tilde{p}(x_i; \mathcal{S})$, and $\tilde{p}(y; \mathcal{S})$ can be calculated from the contingency table of x_i and y over set $\mathcal{S}$. For example, when x_i and y are binary, the contingency table is

x_i	y = 0	y = 1
0	c_{00}	c_{01}
1	c_{10}	c_{11}

In the table, c_{kl} represents the number of concurrence of $x_i = k$ and $y = l$ (where $k = 0, 1$ and $l = 0, 1$) in sample $\mathcal{S}$. For example, the number of the concurrence of $x_i = 0$ and $y = 0$ in the set $\mathcal{S}$ is c_{00}. Let N be the total number of samples in $\mathcal{S}$. For $x_i = k$ and $y = l$ (where $k = 0, 1$, $l = 0, 1$), we have

$$\tilde{p}(x_i = k, y = l; \mathcal{S}) = \frac{c_{kl}}{N} \quad (3)$$

$$\tilde{p}(x_i = k; \mathcal{S}) = \sum_l \frac{c_{kl}}{N} \quad (4)$$

$$\tilde{p}(y = l; \mathcal{S}) = \sum_k \frac{c_{kl}}{N} \quad (5)$$

Plugging these probabilities in Equation (2) will give the estimate for $I(x_i; y|\mathbf{x_{obs}})$. Since all instances in the sample have the same value as $\mathbf{x}$ on known features, the probability of class y given known features (i.e., $Pr(y|\mathbf{x_{obs}})$) can be approximated by $\tilde{p}(y = l; \mathcal{S})$, which is computed in Equation (5).

Complexity Analysis: We now analyze the complexity of Algorithm 1. Suppose there are d features, we need to keep d contingency tables between feature x_i and y. Each contingency table contains $|\mathcal{X}_i||\mathcal{Y}|$ entries. Thus the space overhead is $O(d|\mathcal{X}_i||\mathcal{Y}|)$. The computation of the contingency table can be done in one pass over the sample data by checking the values of each feature x_i and y (i.e., $d + 1$ checks). Thus the cost of estimating $\tilde{I}(x_i, y; \mathcal{S})$ is $O(|\mathcal{S}|(d+1))$. Let n be the number of cases in $\mathcal{D}$. The cost of generating samples is $O(nd + |\mathcal{S}| \log n + |\mathcal{S}|d)$ (see Section 5.2). The computation of $Pr(y|\mathbf{x_{obs}})$ costs $O(|\mathcal{Y}|)$. Suppose the algorithm will run for m iterations (i.e., asking m questions). Thus the total cost of the algorithm is $O(mdn + m|\mathcal{S}| \log n + md|\mathcal{S}| + m|\mathcal{Y}|)$. Since typically the sample size and number of classes are smaller than database size, the complexity is $O(mdn + m|\mathcal{S}| \log n)$. In our experiments, typically $Pr(y|\mathbf{x_{obs}})$ will increase quickly as more questions are asked. Thus the value of m is quite small.

5. DATA AUGMENTATION

This section describes the data augmentation method. This method is based on the following theorem.

THEOREM 2. *$I(x_i; y|\mathbf{x_{obs}})$ only depends on the distribution of class label given all features (i.e., $\Pr(y|\mathbf{x})$) and the distribution of unknown features given known features (i.e., $\Pr(\mathbf{x_{un}}|\mathbf{x_{obs}})$)*

The proof is shown in the Appendix. Thus if we can generate sample data according to $\Pr(y|\mathbf{x})$ and $\Pr(\mathbf{x_{un}}|\mathbf{x_{obs}})$, we can estimate the mutual information $I(x_i; y|\mathbf{x_{obs}})$ correctly.

To generate samples according to $\Pr(y|\mathbf{x})$, we can use a classification method to assign for each sample $\mathbf{x}$ the label y. To generate a set of samples according to $\Pr(\mathbf{x_{un}}|\mathbf{x_{obs}})$, we go backwards. That is, we first assume that the set of samples have been generated, and then infer a relationship between these samples and $\Pr(\mathbf{x_{un}}|\mathbf{x_{obs}})$. Finally we design a sampling method that will generate samples according to the relationship (i.e., the generated samples will satisfy $\Pr(\mathbf{x_{un}}|\mathbf{x_{obs}})$). To infer $\Pr(\mathbf{x_{un}}|\mathbf{x_{obs}})$ from a set of samples, a natural way is to use a non parametric estimation method. We use kernel density method in this paper. Section 5.1 describes how we infer $\Pr(\mathbf{x_{un}}|\mathbf{x_{obs}})$ using the kernel density estimation method and Section 5.2 presents our sampling method.

5.1 Inferring $\Pr(\mathbf{x_{un}}|\mathbf{x_{obs}})$ Using Kernel Density Method

This section shows how to use kernel density method to infer $\Pr(\mathbf{x_{un}}|\mathbf{x_{obs}})$. We first give a brief introduction to kernel density method, then present how it can be used to infer $\Pr(\mathbf{x_{un}}|\mathbf{x_{obs}})$, and finally show how to learn the parameters in kernel functions.

The kernel density estimation [19] is a nonparametric method to estimate the distribution of $\Pr(\mathbf{x}|\mathcal{D})$. The key idea is to view the data distribution as a mixture of kernel functions (e.g., Gaussian distribution), each centering around a data point in the database.

Let z be an existing case in helpdesk database $\mathcal{D}$ and z_i be the i-th feature of z.

$$\Pr(\mathbf{x}|\mathcal{D}) \stackrel{\text{def}}{=} \frac{1}{|\mathcal{D}|} \sum_{\mathbf{z} \in \mathcal{D}} K(\mathbf{x}, \mathbf{z}), \quad (6)$$

where $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a kernel with $\int_{\mathcal{X}} K(x, y)dx = 1$, and $\mathcal{D}$ is the database of existing cases.

We use a type of kernel called product kernel, where

$$K(\mathbf{x}, \mathbf{z}) \stackrel{\text{def}}{=} \prod_i K_i(x_i, z_i),$$

With a product kernel, the feature variables around each training data point can be thought as "local independence".

For continuous features, we use a widely used product kernel, normalized radial basis function (RBF) kernel. The kernel function on the ith feature equals

$$K_i(x_i, z_i) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left(-\frac{(x_i - z_i)^2}{2\sigma_i^2}\right),$$

where σ_i^2 is known as the bandwidth of the kernel.

For binary features, we use a kernel in the form of

$$K_i(x_i, z_i) = \beta_i + (1 - 2\beta_i)[\mathbf{x}_i = z_i].$$

Here $[x_i = z_i] = 1$ if x_i equals to z_i, 0 otherwise. β_i mimics the bandwidth of the kernel.

The following theorem shows how we can use product kernel to infer $\Pr(\mathbf{x_{un}}|\mathbf{x_{obs}})$. The proof is in Appendix.

THEOREM 3.

$$\Pr(\mathbf{x_{un}}|\mathbf{x_{obs}}) \propto \sum_{\mathbf{z} \in \mathcal{D}} K_{\mathbf{obs}}(\mathbf{x_{obs}}, \mathbf{z_{obs}}) \prod_{i \in \mathbf{un}} K_i(x_i, z_i) \quad (7)$$

where $K_{\mathbf{obs}}(\mathbf{x_{obs}}, \mathbf{z_{obs}}) = \prod_{i \in \mathbf{obs}} K_i(x_i, z_i)$.

$K_{\mathbf{obs}}(\mathbf{x_{obs}}, \mathbf{z_{obs}})$ is the probability of an existing case $\mathbf{z}$ matching the known features $\mathbf{x_{obs}}$ of the new case $\mathbf{x}$. $K_i(x_i, z_i), i \in \mathbf{un}$ is the probability of the value of an unknown feature x_i of the new case matching the same feature from the existing case $\mathbf{z}$.

This theorem shows that an unknown feature x_i of an augment sample can be generated using feature z_i of existing case $\mathbf{z}$ in database. At the same time, the existing case $\mathbf{z}$ should be selected according to its probability of matching the known features of the new case (i.e., $K_{\mathbf{obs}}(\mathbf{x_{obs}}, \mathbf{z_{obs}})$).

Now we show how to learn the parameters in the kernel functions. We assume all σ_i^2's (or β_i's) are the same. The parameter σ^2 or β is estimated from the given training data using cross validation [26] as follows. For data point $\mathbf{x}$, we compute the probability $\Pr(\mathbf{x}|\mathcal{D})$ as Eq. (6), where $\mathcal{D}$ consists all data points other than $\mathbf{x}$. We select the parameter σ^2 or β which results the maximum product of probability[3]. This learning step happens offline, and its complexity is $O(kdn)$ where k is the number of iterations and n is the number of cases in $\mathcal{D}$. For most datasets the learning process

[3]Practically, we use the maximum sum of log probability.

converges quickly, thus k is quite small. For very large databases, we can also choose a sample of $\mathcal{D}$ (say 10,000 records) rather than using all records in $\mathcal{D}$ to further reduce the cost.

5.2 Algorithm for Data Augmentation

Algorithm 2 describes how to generate an augmented sample $\mathbf{x}$. The values of known features of $\mathbf{x}$ equals the feature values provided by user. The values of the unknown features of $\mathbf{x}$ is generated based on Equation (7). The equation requires probabilities $K_i(x_i, z_i)$, which can be estimated given a sample $\mathbf{z}$ from the database and the Kernel function K_i. K_i can be learned in a training step described in Section 5.1. We call the sample $\mathbf{z}$ a *factitious sample* because it is drawn directly from the database (not generated through augmentation). Step 1 of the algorithm draws $\mathbf{z}$ from $\mathcal{D}$ with probability $K_{\mathbf{obs}}(\mathbf{x_{obs}}, \mathbf{z_{obs}})$. This $\mathbf{z}$ is used to estimate $K_i(x_i, z_i)$ and to generate the values of unknown features (i.e., $x_i, i \in \mathbf{un}$) of the augment sample in step 2 to 4. Step 5 finally concatenates the known features of $\mathbf{x}$ with generated unknown features and adds the result to $\mathcal{S}$. Step 6 assigns class label for $\mathbf{x}$.

Algorithm 2 Sampling from a kernel density model

1: draw a $\mathbf{z} \in \mathcal{D}$ with a probability proportional to $K_{\mathbf{obs}}(\mathbf{x_{obs}}, \mathbf{z_{obs}})$
2: **for all** $i \in \mathbf{un}$ **do**
3: draw $x_i \in \mathcal{X}_i$ with a probability of $K_i(x_i, z_i)$
4: **end for**
5: add $\mathbf{x} = (\mathbf{x_{obs}}, \mathbf{x_{un}})$ to sample $\mathcal{S}$.
6: assign label to $\mathbf{x}$ based on logistic regression model.

To generate labels for the samples, we use logistic regression model [26] in this paper because many data mining literatures, such as [26], report that logistic regression gives results with higher precision than decision trees on most data sets. We train the logistic regression model based on existing cases in $\mathcal{D}$. The training step happens offline and its complexity is $O(kdn)$ where k is the number of iterations. We apply the model on the sampled data points to predict their labels.

Algorithm 2 assumes that unknown features $x_i, i \in \mathbf{un}$ of an augmented sample $\mathbf{x}$ are conditionally independent given the factitious samples $\mathbf{z}$ drawn from the database. However, the factitious samples $\mathbf{z}$ are drawn from the database, thus they keep the correlations between unknown features. Hence the unknown features of the augmented sample also keep such correlations.

Let n be the number of cases in $\mathcal{D}$. Step 1 scans the database once and takes $O(n|\mathbf{obs}|)$ time to compute the probabilities. Since these probabilities only depend on the observed part of $\mathbf{x}$ and all instances in $\mathcal{D}$, they can be computed once for the whole sampling set. Once we compute the probabilities, Step 1 takes additional $O(\log n)$ time to select an existing case $\mathbf{z}$. Step 3 takes $O(|\mathcal{X}_i|)$ time for unknown feature x_i. For binary features, the cost is $O(1)$. Thus step 2 to 4 take $O(|\mathbf{un}|)$ time because there are $O(|\mathbf{un}|)$ unknown features. Step 6 takes $O(d)$ time. The cost for generating samples is $O(n|\mathbf{obs}| + |\mathcal{S}|(\log n + d))$. There can be at most d features, thus the complexity of Algorithm 2 is $O(nd + |\mathcal{S}| \log n + |\mathcal{S}|d)$. The algorithm scans the database only once (at step 1) for the whole sample set.

For large databases, instead of using all records in the database in the kernel density method, we can use a small sample (say 10,000 records). One possible way to select this sample is to use locality sensitive hashing [17] to quickly identify a small set of of $\mathbf{z} \in \mathcal{D}$'s with high $K_{\mathbf{obs}}(\mathbf{x_{obs}}, \mathbf{z_{obs}})$ (those with low probabilities are unlikely to be selected in step 1 and thus can be ignored).

The choice of the size of sample set $\mathcal{S}$ is the trade-off between accuracy and the computational time. In our experiments, we choose the sample size to be 300.

6. EXPERIMENTS

This section compares the proposed method with existing methods. Section 6.1 describes the setup of the experiments. Section 6.2 presents the results of a user study. Since the scale of the user study is limited, we also conducted larger scale simulation in Section 6.3 to study the impact of various parameters such as the number of recommendations, the number of features, and the number of probes. Section 6.4 further examines the impact of using user provided initial information, correction information, and additional information. Section 6.5 reports the execution time.

6.1 Experimental Setup

Algorithms: We compare the following algorithms:

(1) Active Feature Probing (AFP). This is the proposed method (Algorithm 1). We use all the data records in the database during the data augmentation step. The sample size is 300. The top 5 ranked questions and recommendations are presented to user. The probabilities of the recommendations are also presented. Instead of selecting a stopping threshold t, we run the algorithm for a various number of iterations and report the results.

(2) Logistic Regression (LR): This method assumes all features of a new problem is known, and runs logistic regression [26] over training cases to predict the recommendation on a test case. Since the information is complete, the accuracy should be the upper bound of the active feature probing.

(3) Traditional decision tree (DT): This method uses a decision tree over all existing cases. Each node in the tree will be asked as a question, and each leaf will be a recommendation. This method ignores user provided information. We used C4.5 program[37] Release 8. (`http://www.rulequest.com/Personal/`).

(4) Empirical dynamic decision tree (EMP): This method improves upon the DT approach by building the decision tree over all cases that match the initial information provided by user. This method uses initial information provided by the customer. However, it does not use the other information (e.g., correction of a previous answer or some information not being asked) provided by the customer at run time.

(5) Case-based reasoning (CBR): This method is similar to the case-based reasoning method used in Compaq QuickSource [36]. The method works as follows. When a user provides some initial features, all cases matching at least one initial feature are retrieved. Questions in these cases are ranked by the product of the gain ratio [37] over these cases and the similarity scores. The similarity function assigns equal weights to all features. Thus CBR ranks questions in a similar way to how the decision tree method ranks the questions. The top 5 ranked questions are presented to the user. Recommendations in these cases are also ranked based on their similarity to the new case. The user can answer some of these questions, provide additional information, or correct answers to some previously answered questions, and then click a "submit" button. The ranking of questions and recommendations get updated based on the new set of known features. The process stops when the user selects a recommendation.

All the experiments are conducted on a machine with Intel PIII 3.2 GHZ CPU, 2 GB RAM, and running Windows XP. All algorithms are implemented in C++.

Datasets: We use a synthetic dataset and a real helpdesk database. Table 4 summarizes the properties of these datasets. The real helpdesk database is obtained from the technical support group at

Table 4: Properties of Datasets

Data	Number of cases	Number of features	Number of recommendations
Synthetic	1,000,000	500	35
Real	340,865	1000	25

a research university. Each case in the database contains a series of communications (reports/emails) between the users (students and faculty members) and the support staff. Right now, there are about 340,865 cases divided into 25 recommendations. 1000 features (questions) have been extracted manually.

We also use a synthetic dataset to examine the impact of various parameters (e.g., number of features) because it is quite difficult to vary these parameters on the real data. We use a data set that contains 1,000,000 documents drawn from the Topic Detection and Tracking (TDT2) Corpus [14]. The dataset is generated from news reports drawn on a daily basis from six English news sources from January 4 to June 30, 1998. The dataset contains 35 different categories. We treat each category as a class label. We also select 500 most informative (based on mutual information between class labels and words) words as features. This data set is similar to helpdesk data on three aspects: 1) the helpdesk database also stores text (email or reports) data; 2) each case in the helpdesk database is also associated with a recommendation as in this dataset; 3) the class label of a document can be predicted by its features (words), the same way as the recommendation of a case in the helpdesk database can be predicted by its features.

Performance measure: We use the accuracy of recommendation as the performance measure. It is computed as the fraction of testing cases where the predicted recommendation (when user finishes the question answering process) is the same as the real recommendation in the test case. We also count the number of questions answered by users before they find the recommendation.

Setup of user study: We performed a user study to evaluate various methods. The system was installed on the service homepage of the support group at a research university. The real dataset was used in the study. 19 users (as randomly selected students) used the system for 50 testing cases randomly selected from the real dataset. These cases include problems from 25 different recommendations such as account creation, password reset, hardware/software installation, patch download and file/directory deletion etc. There were 2 test cases for each recommendation. The average number of questions in the original test cases is 4. Each user was randomly assigned 6 test cases. Users read these cases such that they knew the description and answers to relevant questions about the cases, as well as the correct recommendation. The remaining cases were treated as existing cases (i.e., training data).

For each case, users used the tool that implemented AFP, DT, EMP, and CBR (LR is not tested because it assumes all features are known). The order of methods was randomly permuted for each user to reduce the possible bias introduced by the order. For each case and each method, a user inputs some initial description of the problem. The system repeatedly presented questions. The user answered such questions. The user can also provide additional answers, or correct previous answers. A list of recommendations was also presented to the user and the user could select a recommendation at any time during the process. The user could also exit the system at any time. At the end of experiment, we also asked the user to rank the effectiveness of four methods.

6.2 User Study Results

This section presents the results of user study. Table 5 summarizes the average number of questions and the accuracy for each method.

Table 5: Overall results of user study

Methods	Average # of questions	Accuracy
AFP	6	82.5%
DT	11.1	57.0%
EMP	9	63.2%
CBR	10.2	61.4%

The results show that the accuracy of our approach (AFP) is significantly higher than that of DT. Users also answer fewer questions using AFP than using DT. The main reason is that DT creates the decision tree statically, thus it does not use the information provided by the user.

The performance of EMP is better than that of DT, because EMP builds the decision tree on documents that match the user's request. Thus the initial feature value is used by EMP. However, AFP still outperforms EMP both in terms of accuracy and the number of questions. The reason is that EMP does not take into account the additional information and corrections provided by users. As described in Section 1, it also suffers the small sample problem when there are too few matching cases in the database.

The performance of AFP is also better than CBR. The reason is that AFP uses a probabilistic method such that it avoids the overfitting problems of CBR when the database is incomplete and contains errors. We found that about 62% (31 out of 50) of the testing cases do not have exact matches in the database. The performance of CBR is actually quite similar to that of EMP, because both methods rely on matching cases, and they rank the questions similarly.

We also divide the test cases into 3 categories based on the number of questions in the original cases. The three categories are those with 1-3 questions (low complexity), with 4-5 questions (medium complexity), and those with 6 or more questions (high complexity). There are 17 low complexity cases, 21 medium complexity cases, and 12 high complexity cases. Figure 4 reports the accuracy of the various algorithms for each category of cases. Figure 5 reports the average number of questions for each category of cases. The average number of questions in the original cases is also shown as a baseline (labeled as Best-Possible).

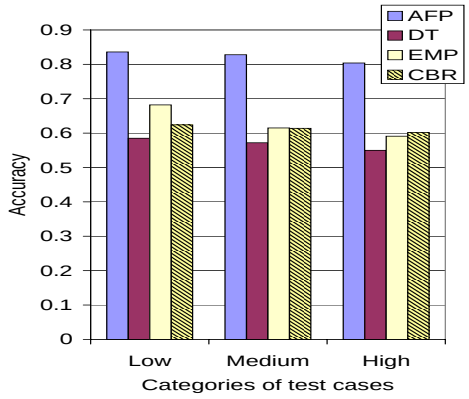

Figure 4: The accuracy over test cases with various complexity.

The results show that AFP has the highest accuracy and generates the smallest number of questions among four algorithms across all categories of cases. The performance of all algorithms also degrades a little as cases become more complex. However, the accuracy of AFP is still around 80% for the most complex cases. The average number of questions for AFP is also about a factor of 1.4 to 2 of the best possible case (i.e., using questions in the original test cases) for test cases with various complexities.

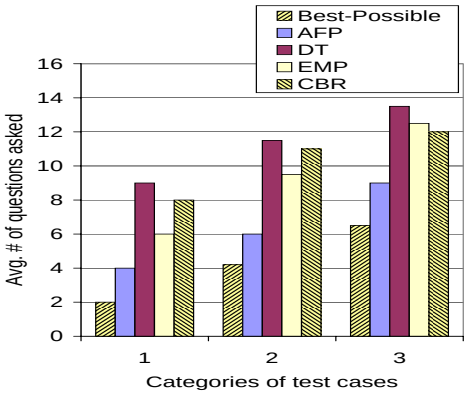

Figure 5: The average # of questions over test cases with various complexity.

We also ran a paired t-test over the number of questions for each algorithm over all cases and all users to verify whether the difference is significant. The probability of the null hypothesis that AFP and EMP have the same average number of questions is 0.0001, the probability of that AFP and DT have the same average number of questions is 0.0002, and the probability of that AFP and CBR have the same average number of questions is 0.0001. Thus the improvement of AFP over the other three methods is statistically significant.

Table 6 presents the number of users who rank each approach as the best. Majority of users rank AFP approach as the best.

Table 6: Results of survey

Approach	# users that rank it the best
AFP	14
DT	1
EMP	2
CBR	2

6.3 Simulation Results

There are three important parameters that may affect the performance of a helpdesk system: the number of recommendations, the number of features, and the number of questions (probes). This section presents the results of a larger scale simulation experiment to study the impact of these parameters.

Both the synthetic and real dataset are split into 70% cases for training and 30% cases for testing. For the synthetic data, we treat each document as a case. The experiment is conducted as follows. The algorithm being studied selects the top ranked question on the presence of a feature. A program that simulates a user will send an answer to the question to the algorithm. Based on the provided answer, the algorithm will select the next question to ask. This Q&A process repeats for a certain number of iterations.

For real data, we directly use the initial, correction (correction of a previous answer), and additional information in test cases. The initial information is provided in the beginning of the simulation. The correction information is provided at a random step after the wrong answer is given. The additional information is provided at the same step as in the test case (e.g., if the user provides additional information at the second question in the test case, it will be provided at the second question in the simulation). Since the synthetic data does not contain such information, we inject such information in the simulation process. We randomly select a feature (word) in each document as the initial information. The simulation program will give a wrong answer to a question with a small probability

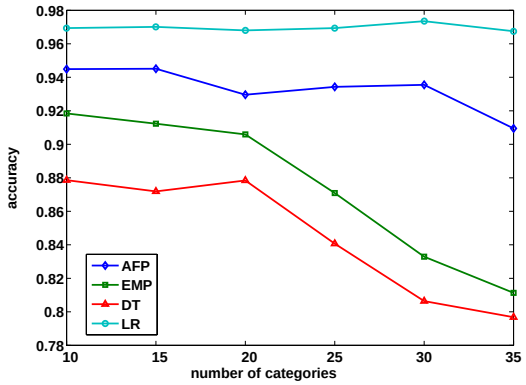

Figure 6: The accuracy results on synthetic dataset with different number of categories (recommendations)

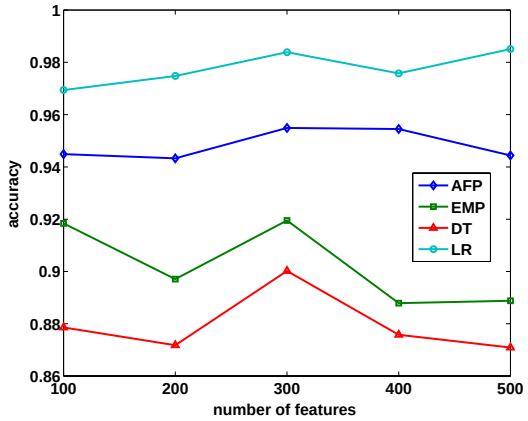

Figure 7: The accuracy results on synthetic dataset with different number of features.

(5%). The program will provide an additional answer (the presence of a feature in the case) with a small probability (5%) or correct a wrong answer previously given.

We only simulate AFP, DT, EMP, and LR. For AFP, we assume the user always answer the top ranked question. It is difficult to simulate CBR because it asks several questions at a time and we find in our user study that user often does not answer the first question. The results of the user study already show that AFP has better performance than CBR, and CBR has similar performance as EMP.

For synthetic dataset, we vary all three parameters. We vary the number of recommendations by first selecting a random subset of categories (recommendations) and then including documents with these categories. We vary the number of features by asking questions only on the selected features. For the real data, we use all available features and recommendations, and vary the number of probes. It is difficult to vary the number of features and recommendations for the real data set. For example, all features in a case in the real data set need be used to find the correct recommendation.

Figure 6 shows the results over synthetic data when we vary the number of categories (recommendations) and fix the number of features at 500 and the number of probes at 10. Figure 7 shows the results over synthetic data when we vary the number of features, but fix the number of categories at 25 and the number of probes at 10. Figure 8 reports the results over synthetic data when we vary the number of probes, and fix the number of categories at 25 and the number of features at 500. Figure 9 shows the results over real data when we vary the number of probes.

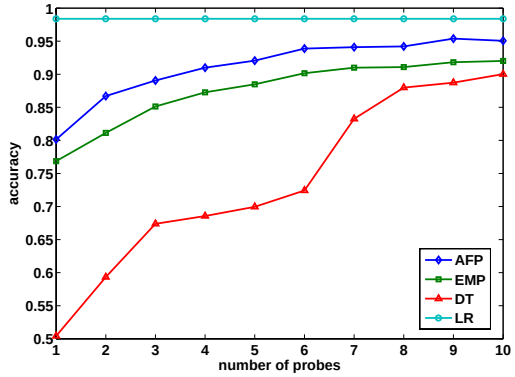

Figure 8: The accuracy results on synthetic dataset with different number of probes.

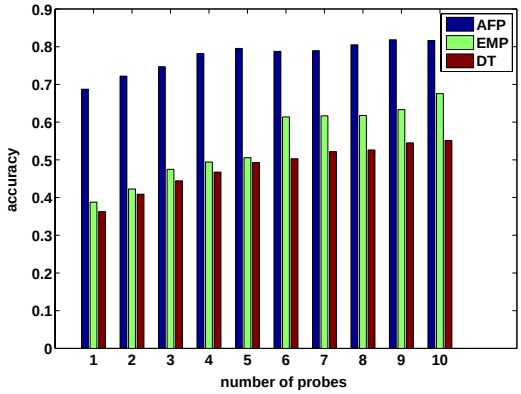

Figure 9: The accuracy results on real data with different number of probes. The accuracy of LR is 0.8263.

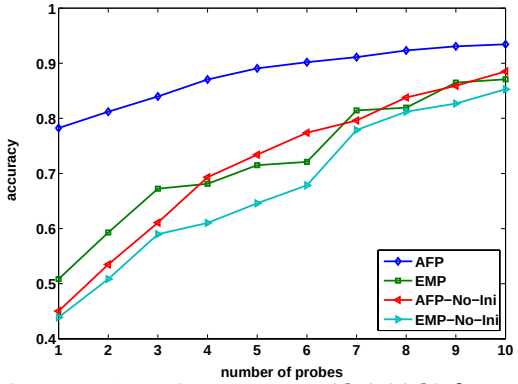

Figure 10: Scenario I: users provide initial information

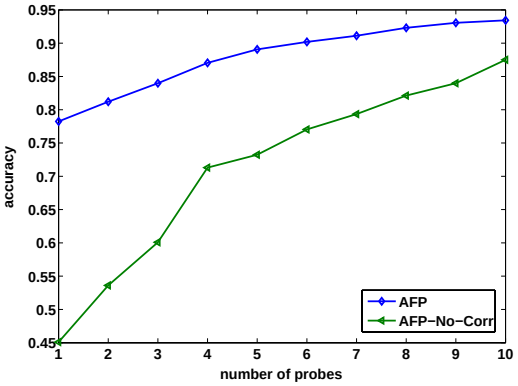

Figure 11: Scenario II: users correct previous answers

The results show that AFP has better accuracy than both DT and EMP under all settings because it uses all information provided by user. The accuracy of AFP is lower than LR. This is expected because LR knows all feature values, while AFP gradually learns feature values from the user. In practice, a user usually does not give all information in the beginning, thus LR only indicates a theoretical upper bound of the accuracy. Further, the accuracy of AFP is only about 3%-5% lower than that of LR when 10 probes are used.

Figure 6 shows that as the number of categories (recommendations) increases, the performance of all methods except LR degrades because it is more difficult to classify more categories. DT method degrades the most because it does not use initial information provided by users. The performance of LR does not change much because it assumes all features are known.

Figure 7 shows that as the number of features increases, the performance of all methods fluctuates, but there is no clear trend except that the performance of LR gets improved in most cases. LR uses all features, thus as the number of features increases, its performance gets improved. However, the other 3 methods only use 10 features (10 probes), thus the increase of the total number of features may or may not improve the performance of these methods.

Figure 8 and Figure 9 show that as the number of probes increases, the performance of all methods except LR gets improved. The performance of LR is flat because it uses all feature values, thus does not depends on the number of probes. The performance of the other 3 methods gets improved because they use more features when more probes are used. The results for real data also show that after 5 probes the accuracy of AFT is about 80%, very close to the optimal case of using LR. This indicates that a recommendation can be made with high accuracy after just a few probes. In general, our approach works well when user has provided enough information to classify the recommendation. Our approach does not work well if very limited information is provided.

6.4 Impact of Initial, Correction, and Additional Information

This section investigates the impact of initial, additional, and correction information provided by user. Since previous experiments already show that AFP has better performance than DT, EMP, and CBR, we focus on AFP in this section. We also use the real data set and run simulation on it.

We consider three different scenarios for the real data set. (1) Scenario I: users provide initial information. (2) Scenario II: users correct information provided previously. (3) Scenario III: users provide additional information. Again we use 70% of cases as training and 30% of cases as testing. All test cases belong to the first scenario. About 1000 test cases fall into the second scenario and about 4000 test cases fall into the third scenario. Some examples of these three scenario are shown in the Appendix (Section 8.4).

Figure 10 shows the results of AFP and a variant of AFP (AFP-No-Ini) which does not use initial information for all test cases under Scenario I. Since EMP also uses initial information, we also include the results of EMP and a variant of EMP (EMP-No-Ini) in Figure 10 for comparison.

Figure 11 shows the results of AFP and a variant of AFP (AFP-No-Corr) which does not use correction information for all test cases under Scenario II (i.e., those 1000 test cases). Figure 12 shows the results of AFP and a variant of AFP (AFP-No-Add)

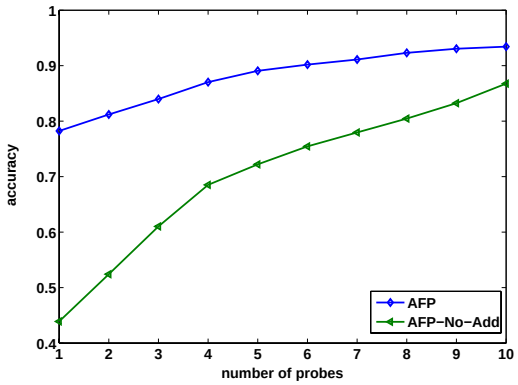

Figure 12: Scenario III: users provide additional information

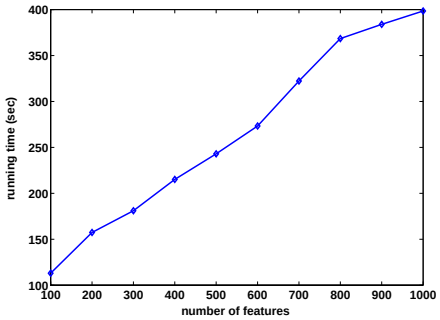

Figure 13: Execution time of AFP for all test cases over real data with different number of features

which does not use additional information for all test cases under Scenario III (i.e., those 4000 test cases).

The results show that using initial information, correction information, and additional information leads to much higher accuracy. The improvement however decreases as the number of probes increases. This is expected because more information is available as more questions are asked, thus it becomes easier to predict the recommendation.

6.5 Execution Time

This section reports the execution time of the proposed method. Again we use 70% of cases as training and 30% of cases as testing. Figure 13 reports the total execution time of AFP over all test cases in the real data as we vary the number of features. We fixed the number of probes at 10 and used all cases. The execution time is the time to generate questions for all testing cases. The time of training the kernel density model and building logistic regression model is also included. Figure 14 reports the results when we vary the number of recommendations by first randomly selecting a number of recommendations, and then including cases with these recommendations. We fix the number of probes at 10 and the number of features at 1000.

The results show that the execution time increases near linearly with the number of features and the number of recommendations. Note that the time reported here is the total execution time for all test cases. The time takes for each case is less than 0.1 second, thus it is more than enough for an interactive interface.

Figure 15 reports the execution time of AFP over real data as we vary the number of cases. We randomly selected 10 data sets, from 10% to 100% of cases in the original data set. We then use 70% of cases in each data set as training and 30% as testing. The results show that the execution time increases about linearly with the number of cases. The results on the synthetic data set are similar

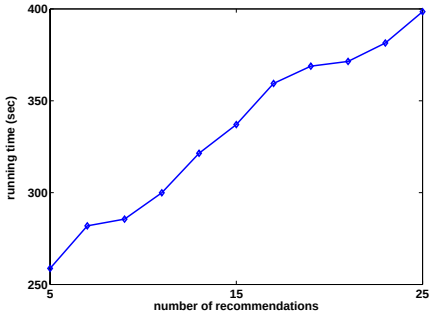

Figure 14: Execution time of AFP for all test cases over real data with different number of recommendations

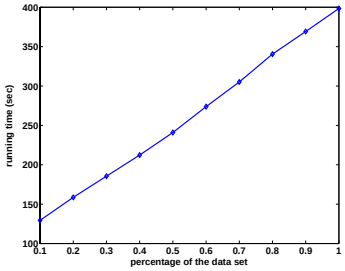

Figure 15: Execution time of AFP for all test cases over the real data with different number of cases

and thus are not included.

7. CONCLUSIONS

This paper investigates the problem of automatically generating clarification questions to provide appropriate recommendations to customers. The proposed approach models the problem as a classification problem, and proposes a solution that dynamically selects the most informative questions. Unlike existing decision tree and CBR approaches, the proposed approach can use all information provided by customers, and is robust to databases that are incomplete or contain errors. The experimental results verify the effectiveness of the proposed solution against the decision tree approach.

There are several directions for future research: First, we will investigate how to use NLP techniques to automatic extract features. Second, we will investigate more efficient algorithms. Algorithm 2 is required to go through the entire dataset to draw a sample $\mathbf{z}$ (or multiple samples in one pass). Although the computational complexity is linear, the algorithm is not very scalable for a dataset with millions of instances. To overcome the scalability issue, we may approximate the sampling results by filtering the instances with relatively low probabilities, then conducting the sampling on the remaining data. Finally, we will investigate the issue of possible sequential dependencies between features because the questions being asked are often related. A possible extension of our approach to solve the sequential problem is presented in Appendix.

8. REFERENCES

[1] C. C. Aggarwal and P. S. Yu. The igrid index: reversing the dimensionality curse for similarity indexing in high dimensional space. In *SIGKDD*, pages 119–129, 2000.

[2] R. Agrawal, R. Rantzau, and E. Terzi. Context-sensitive ranking. In *SIGMOD*, pages 383–394, 2006.

[3] S. Agrawal, S. Chaudhuri, G. Das, and A. Gionis. Automated ranking of database query results. In *CIDR*, 2003.

[4] D. W. Aha, D. Mcsherry, and Q. Yang. Advances in conversational case-based reasoning. *The Knowledge Engineering Review*, 20(3):247–254, 2006.

[5] J. Allen. *Natural Language Understanding*. Addison Wesley, 1994.

[6] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is nearest neighbor meaningful? In *ICDT*, 1999.

[7] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.

[8] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth and Brooks/Cole, Monterey, CA, 1984.

[9] D. Bridge, M. H. Goker, L. Mcginty, and B. Smyth. Case-based recommender systems. *The Knowledge Engineering Review*, 20(3):315–320, 2006.

[10] M. Brodie, I. Rish, and S. Ma. Intelligent probing: a cost-effective approach to fault diagnosis in computer networks. *IBM System Journal*, 41(3), 2002.

[11] C.-H. Chang, M. Kayed, M. R. Girgis, and K. F. Shaalan. A survey of web information extraction systems. *IEEE Transactions on Knowledge and Data Engineering*, 18(10):1411–1428, 2006.

[12] S. Chaudhuri, G. Das, V. Hristidis, and G. Weikum. Probabilistic ranking of database query results. In *VLDB*, pages 888–899, 2004.

[13] S. Chaudhuri and L. Gravano. Evaluating top-*k* selection queries. In M. P. Atkinson, M. E. Orlowska, P. Valduriez, S. B. Zdonik, and M. L. Brodie, editors, *VLDB*, pages 397–410. Morgan Kaufmann, 1999.

[14] C. Cieri, D. Graff, and M. Liberman. The TDT-2 text and speech corpus, 1999.

[15] D. A. Cohn, Z. Ghahramani, and M. I. Jordan. Active learning with statistical models. In *Advances in Neural Information Processing Systems*, pages 705–712, 1995.

[16] P. Cunningham and S. B. A comparison of Model-based and incremental case-based approaches to electronic fault diagnosis. Technical Report TCD-CS-94-21, 1994.

[17] T. Darrell, P. Indyk, and G. Shakhnarovich. *Nearest Neighbor Methods in Learning and Vision: Theory and Practice*. MIT Press, 2006.

[18] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Proceedings of the Royal Statistical Society*, pages 1–38, 1976.

[19] L. Devroye and G. Lugosi. *Combinatorial Methods in Density Estimation*. Springer, 2001.

[20] M. Doyle and P. Cunningham. A dynamic approach to reducing dialog in on-line decision guides. In *EWCBR*, pages 49–60, 2000.

[21] J. English, M. Hearst, R. Sinha, K. Swearingen, and K.-P. Yee. Hierarchical faceted metadata in site search interfaces. In *CHI'02*, pages 628–639, 2002.

[22] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, 2001.

[23] M. Franz, T. Ward, J. S. McCarley, and W.-J. Zhu. Unsupervised and supervised clustering for topic tracking. In *SIGIR*, pages 310–317, 2001.

[24] Z. Ghahramani and M. I. Jordan. Supervised learning from incomplete data via an EM approach. In *Advances in Neural Information Processing Systems*, pages 120–127, 1994.

[25] R. Grishman. Information extraction: Techniques and challenges. In *SCIE '97: International Summer School on Information Extraction*, pages 10–27, 1997. Springer-Verlag.

[26] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, Prediction*. Springer, 2001.

[27] H. V. Jagadish, B. C. Ooi, K.-L. Tan, C. Yu, and R. Z. 0003. idistance: An adaptive b^+-tree based indexing method for nearest neighbor search. *ACM Trans. Database Syst.*, 30(2):364–397, 2005.

[28] M. Jayapandian and H. V. Jagadish. Automating the design and construction of query forms. In *ICDE*, page 125, 2006.

[29] A. A. Kamis and E. A. Stohr. Parametric search engines: what makes them effective when shopping online for differentiated products? *Inf. Manage.*, 43(7):904–918, 2006.

[30] R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

[31] D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: An online algorithm for skyline queries. In *VLDB*, pages 275–286, 2002.

[32] G. Kumaran and J. Allan. Simple questions to improve pseudo-relevance feedback results. In *SIGIR*, pages 661–662, 2006.

[33] P. Langley. Selection of relevant features in machine learning. In *AAAI Fall Symposium on Relevance*, pages 140–144, 1994.

[34] N. Mirzadeh, F. Ricci, and M. Bansal. Feature selection methods for conversational recommender systems. In *Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service*, pages 772–777, 2005.

[35] National Retail Federation. Importance of customer service reinforced in nrf foundation/american express study. http://www.nrf.com/content/press/release2004/custserv1104.htm.

[36] T. Nguyen, M. Czerwinski, and D. Lee. COMPAQ QuickSource: Providing the consumer with the power of artificial intelligence. In *IAAI '93*, pages 142–151, 1993.

[37] J. R. Quinlan. Introduction of decision trees. *Machine Learning*, (1):81–106, 1986.

[38] D. B. Rubin. *Multiple Imputation for Nonresponse in Surveys*. Wiley, New York, 1987.

[39] S. Schmitt. simvar: A similarity-influenced question selection criterion for e-sales dialogs. *Artificial Intelligence Review*, 18:195–221, 2002.

[40] H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *Computational Learning Theory*, pages 287–294, 1992.

[41] K. Sparck Jones. *Automatic Keyword Classification for Information Retrieval*. Butterworth, London, 1971.

[42] M. A. Tanner and W. H. Wong. The calculation of posterior distributions by data augmentation (with discussion). *Journal of the American Statistical Association*, 82:528–550, 1987.

[43] P. Wu, Y. Sismanis, and B. Reinwald. Towards keyword-driven analytical processing. In *SIGMOD*, pages 617–628, 2007.

[44] J. Xu and W. B. Croft. Query expansion using local and global document analysis. In *SIGIR*, pages 4–11, 1996.

[45] K.-P. Yee, K. Swearingen, K. Li, and M. Hearst. Faceted metadata for image search and browsing. In *CHI'03*, pages 401–408, 2003.

Appendix

8.1 Proof of Theorem 1

$$
\begin{aligned}
&\mathbb{E}_{x_i|\mathbf{x_{obs}}}\{H(y|\mathbf{x_{obs}}, x_i)\} \\
&= -\sum_{x_i \in \mathcal{X}_i} \Pr(x_i|\mathbf{x_{obs}}) \sum_{y \in \mathcal{Y}} Pr(y|\mathbf{x_{obs}}, x_i) \log \Pr(y|\mathbf{x_{obs}}, x_i) \\
&= -\sum_{\substack{x_i \in \mathcal{X}_i \\ y \in \mathcal{Y}}} \Pr(x_i, y|\mathbf{x_{obs}}) \log \Pr(y|\mathbf{x_{obs}}, x_i) \\
&= H(y|\mathbf{x_{obs}}) - I(x_i; y|\mathbf{x_{obs}})
\end{aligned}
$$

8.2 Proof of Theorem 2

Using Bayes rule, we have

$$\Pr(y|\mathbf{x_{obs}}, x_i) = \frac{\Pr(x_i, y|\mathbf{x_{obs}})}{\sum_{y \in \mathcal{Y}} \Pr(x_i, y|\mathbf{x_{obs}})} \tag{8}$$

We also have

$$\Pr(y|\mathbf{x_{obs}}) = \sum_{x_i \in \mathcal{X}_i} \Pr(x_i, y|\mathbf{x_{obs}}) \tag{9}$$

Thus by Theorem 1, $I(x_i; y|\mathbf{x_{obs}})$ only depends on $\Pr(x_i, y|\mathbf{x_{obs}})$. In general it is difficult to directly model and learn all $\Pr(x_i, y|\mathbf{x_{obs}})$, because the number of possible combinations of features (all possible $\mathbf{obs}$) is exponentially large (in the order of 2^d).

Instead, our sampling method is based on the following equation:

$$\Pr(x_i, y|\mathbf{x_{obs}}) = \sum_{\mathbf{x_{un \backslash} i}} \Pr(y|\mathbf{x}) \Pr(\mathbf{x_{un}}|\mathbf{x_{obs}}), \tag{10}$$

Here $\mathbf{un} \backslash i$ represents indexes of unknown features excluding i-th feature. $\mathbf{x_{un \backslash} i}$ represents possible values of unknown features of $\mathbf{x}$ except the i-th feature. The proof is as follows.

$$
\begin{aligned}
&\Pr(x_i, y|\mathbf{x_{obs}}) \\
&= \sum_{\mathbf{x_{un \backslash} i}} \Pr(x_i, y, \mathbf{x_{un \backslash} i}|\mathbf{x_{obs}}) \\
&= \sum_{\mathbf{x_{un \backslash} i}} \Pr(y|\mathbf{x_{obs}}, \mathbf{x_{un \backslash} i}, \mathbf{x}_i) \Pr(\mathbf{x_{un \backslash} i}, \mathbf{x}_i|\mathbf{x_{obs}}) \\
&= \sum_{\mathbf{x_{un \backslash} i}} \Pr(y|\mathbf{x}) \Pr(\mathbf{x_{un}}|\mathbf{x_{obs}}).
\end{aligned}
$$

We already show that the mutual information $I(x_i; y|\mathbf{x_{obs}})$ depends on $\Pr(x_i, y|\mathbf{x_{obs}})$. This equation shows that $\Pr(x_i, y|\mathbf{x_{obs}})$ depends on $\Pr(\mathbf{x_{un}}|\mathbf{x_{obs}})$ and $\Pr(y|\mathbf{x})$. Thus Theorem 2 is proved.

8.3 Proof of Theorem 3

$$
\begin{aligned}
&\Pr(\mathbf{x_{un}}|\mathbf{x_{obs}}) = \Pr(\mathbf{x_{un}}, \mathbf{x_{obs}}) / \Pr(\mathbf{x_{obs}}) \\
&\propto \frac{1}{\Pr(\mathbf{x_{obs}})} \sum_{z \in \mathcal{D}} K(\mathbf{x}, \mathbf{z}) \quad \text{by Equation (6)} \\
&\propto \sum_{z \in \mathcal{D}} K(\mathbf{x}, \mathbf{z}) \quad \Pr(\mathbf{x_{obs}}) \text{ is constant for all } \mathbf{x} \\
&\propto \sum_{\mathbf{z} \in \mathcal{D}} \prod_{i \in \mathbf{obs}} K_i(x_i, z_i) \prod_{i \in \mathbf{un}} K_i(x_i, z_i) \\
&\propto \sum_{\mathbf{z} \in \mathcal{D}} K_{\mathbf{obs}}(\mathbf{x_{obs}}, \mathbf{z_{obs}}) \prod_{i \in \mathbf{un}} K_i(x_i, z_i)
\end{aligned}
$$

8.4 Examples in Experiments

Below are some examples in the experiments where our method AFP works well for the three types of user provided information.

- Sample I: initial information is provided

  User: I cannot get access to my mbox. I cannot read it and the directory is root not my home directory. I have had some other problems in the past with the mbox, and I suspect that somebody may be toying around with it.

  AFP: Correct the ownership of your mbox file.

- Sample II: the problem changes from "create account" to "install software".

  User: One of my graduate student needs to access the AUL server. Please create an AUL account for him.

  AFP: Current policy prohibits graduate students from using the AUL.

  User: Actually, he needs to use Matlab in the AUL for his thesis work.

  AFP: We have a license for Matlab which is good through June which can be installed in his machine in your lab.

- Sample III: additional information is provided

  User: I need to create a web page for our group. It is a work related to the server. Would you like to lend me a hand?

  AFP: What kind of web page tools you need?

  User: I need something like FrontPage, which can help to create HTML files, and also I need to set up the web page.

  AFP: We have installed Frontpage. Here's how you set it up on the server. . ..

8.5 Extension to Sequential Dependency Problem

There are often sequential dependencies between different questions. For example, to ask a question whether the user has any problem with the external hard disk, it is better to first ask whether the user is using an external hard disk. We plan to extend our approach to solve this problem as follows.

First, we assume that partial orders between questions have been provided by some domain experts. Thus the probabilistic question selection algorithm will first sort all questions according to the partial order using breadth-first-search. Each question is also associated with the level in the BFS search (we start with level 1, then level 2, etc.). The algorithm will select the currently unanswered question that has the highest score (based on how informative it is) and at the lowest level.

For example, consider 4 questions: 1, 2, 3, and 4. The partial order is $1 \rightarrow 2$ and $3 \rightarrow 4$. Thus after topological sort, the questions are in order 1, 3, 2, and 4. Question 1 and 3 are at level 1, question 2 and 4 are at level 2. Suppose the score for question 1 is 0.8 and the score for question 3 is 0.5. Question 1 will be selected first.

Since BFS search and topological sort can be done in $O(d)$ time where d is the number of features, the extended algorithm has the same complexity as the original version of the algorithm (Algorithm 1).