When Is the Right Time to Refresh Knowledge Discovered from Data?

Xiao Fang
Department of Information Operations and Technology Management
University of Toledo
Toledo, Ohio 43606
xiao.fang@utoledo.edu

Olivia R. Liu Sheng
School of Accounting and Information Systems
University of Utah
Salt Lake City, Utah 84112
actos@business.utah.edu

December 2006
Abstract

Knowledge Discovery in Databases (KDD) provides organizations necessary tools to sift through vast data stores to extract knowledge. This process supports and improves decision making in organizations. In this paper, we focus on knowledge refreshing, a critical process to ensure the quality and timeliness of knowledge discovered in a KDD process. This has been unfortunately overlooked by prior researchers. Specifically, we study knowledge refreshing from the perspective of when to refresh knowledge. We introduce knowledge loss, a key concept in knowledge refreshing, and propose how to measure and estimate knowledge loss. We propose a Markov decision process model for knowledge refreshing and derive from the model an optimal policy and a heuristic method for knowledge refreshing. We demonstrate the robustness and the effectiveness of the proposed policy and method through simulation. The proposed policy and method provide decision makers guidance in running KDD effectively and efficiently.

Keywords: Knowledge Refreshing; Knowledge Loss; Knowledge Discovery in Databases; Markov Decision Process
1. Introduction

The need for extracting knowledge from massive data sets has been growing rapidly, owing to the vast amount of data accumulated in organizations and the critical role of knowledge in organizational decision making. Knowledge Discovery in Databases (KDD) provides organizations necessary tools to sift through vast data stores to extract knowledge. KDD, which is defined as a nontrivial process of discovering useful knowledge from data (Fayyad et al. 1996), consists of such steps as data pre-processing, data mining, and pattern post-processing (Fayyad et al. 1996). In the step of data pre-processing, relevant data are selected, cleaned and transformed into formats suitable for data mining. Next, patterns (e.g., decision trees) are extracted from preprocessed data by applying appropriate data mining algorithms. Finally, extracted patterns are transformed into knowledge through pattern evaluation (e.g., filtering out invalid or uninteresting patterns) and pattern interpretation (e.g., transforming patterns into terms understandable by users). Knowledge discovered by running KDD is defined as valid, novel, potentially useful, and ultimately understandable patterns (Fayyad et al. 1996).

Prior research reported KDD applications in functional areas such as marketing (Cooper and Giuffrida 2000), finance (Tam and Kiang 1992, Sarkar and Sriram 2001), and operations management (Chen and Wu 2005). For example, Cooper and Giuffrida (2000) employed classification techniques to enhance a traditional market-response model, whereas Chen and Wu (2005) developed an order batching approach based on the knowledge discovered through association rule mining. Most of the earlier KDD research assumed that data are static, and focused on either efficiency improvement of the KDD process (e.g., designing more computationally efficient data mining algorithms) or
innovative KDD applications. However, for real world applications, new data are continuously accumulated and data are dynamic in reality. As a result, knowledge discovered using KDD becomes obsolete over time as the discovered knowledge only relates the nature of data at the time the KDD process was run. Newly added data could bring in new knowledge and invalidate part or even all of earlier discovered knowledge.

To support effective and efficient decision making, knowledge discovered using KDD needs to be kept current with its dynamic data source (Cooper and Giuffrida 2000). Bourgeois and Eisenhardt (1988) and Eisenhardt (1989) pointed out that the use of obsolete information/knowledge in decision making is a major cause of poor and delayed strategic decisions. In a recent survey conducted by King et al. (2002), maintaining currency of knowledge was rated as the third most important knowledge management issue by 2,073 knowledge management practitioners and managers. In this research, we study knowledge refreshing, which we define as the process of keeping knowledge discovered using KDD up-to-date with its dynamic data source.

Rest of our paper is organized as follows. In Section 2 we review related work and discuss the research gaps tackled by this study. We define a key concept in knowledge refreshing – knowledge loss, and propose how to measure and estimate knowledge loss in Section 3. In Section 4, a Markov decision process model for knowledge refreshing is proposed. An optimal knowledge refreshing policy and an adaptive heuristic method are subsequently derived from the model. Section 5 contains numerical experiments on the robustness and the effectiveness of the proposed policy and method. We provide some concluding remarks and outline future research directions in the last section.
2. Related work

Prior research related to this work focused on how to maintain patterns learned from data mining over a dynamic data source. Specifically, this stream of research can be grouped into two categories: incremental data mining and data stream mining. Incremental data mining keeps patterns current with a dynamic data source by revising patterns learned from a previous run of data mining, instead of learning from scratch. A number of incremental data mining algorithms were proposed for major data mining models (Ganti et al. 1999a): classification, association rule mining and clustering. For classification, algorithms such as ID4 (Schlimmer and Fisher 1986) and ID5 (Utgoff 1988, 1989) were developed for revising a decision tree induced from old data as new data were added in. For association rule mining, the FUP algorithm (Cheung et al. 1996) was first proposed for maintaining large itemsets over a dynamic data source. A more efficient incremental algorithm for association rule mining was given in (Thomas et al. 1997). Can (1993) designed an incremental clustering algorithm for updating document clusters over an evolving document database.

Current data-intensive applications are based on huge-size data sources with rapid and continuous loading of large volumes of new data, termed as data streams. Examples of data streams include web logs, transaction databases in retail chains, network traffic data and financial tickers. Due to hardware constraints such as bounded memory space and limited processing speed, incremental data mining algorithms are not capable of maintaining patterns over data streams (Domingos and Hulten 2000). A number of data stream mining algorithms (Rastogi 2003) were proposed recently to solve this emerging problem. Data stream mining algorithms usually employed data reduction techniques
Barbara et al. 1997, such as sampling, to maintain approximate patterns over data streams. The Hoeffding tree algorithm, which applied the Hoeffding bound (Hoeffding 1963) to determine the number of samples required to learn a test attribute, was introduced in (Domingos and Hulten 2000, Hulten et al. 2001) for maintaining approximate decision trees over data streams. The lossy counting algorithm (Manku and Motwani 2002) was proposed for maintaining association rules over data streams. BIRTCH (Zhang et al. 1996) is one of the early algorithms suitable for clustering data streams. The algorithm kept a summary of a data source in memory, and then clustered the in-memory summary. Motivated by BIRTCH, a series of algorithms (Guha et al. 2000, 2003) were introduced for clustering data streams.

Both the incremental data mining research and the data stream mining research only addressed one step in KDD – the data mining step, and focused on updating patterns over a dynamic data source. However, it is knowledge, not patterns that can support organizational decision making effectively. For example, association rules (i.e., an example of patterns) learned using association rule mining are usually too many to be handled by decision makers (Brin et al., 1997). Moreover, a substantial number of association rules are just common sense, or already known to decision makers. To support effective decision making, the pattern post-processing step in KDD needs to be executed to extract knowledge (e.g., unexpected association rules) from all learned association rules, using various criteria and techniques (Siberschatz and Tuzhilin 1996, Kleinberg et al. 1998).

The KDD process cannot be fully automated, except for the data mining step. It requires people (e.g., domain experts) in cleaning data before mining them and extracting
knowledge from patterns discovered using data mining. Hence, KDD is a costly process (Manku and Motwani 2002), as personnel costs dominate equipment and computational costs (Gibson and Meter 2000). As a result, it may be impractical to run KDD whenever there is an update in a data source. Further, it may be unnecessary to run KDD whenever there is an update in a data source. Such a practice may often result in no new knowledge discovered because successive snapshots of real world data sources are very likely to overlap considerably (Ganti et al. 1999b). On the other hand, running KDD too seldom could result in losing critical knowledge. This will have adverse effect on decision making. Therefore, it is critical to determine when to run KDD so as to optimize the trade-off between the loss of knowledge and the cost incurred by running KDD. Our research studies the knowledge refreshing problem from this perspective. Our paper is different from previous research on incremental data mining and data stream mining in the following ways. First, this research studies when to refresh knowledge while past research investigated how to refresh patterns. Second, this research considers the complete KDD process while past research focused on one step in the KDD process, i.e., the data mining step. Additionally, earlier research investigated the knowledge refreshing problem from the standpoint of computer science, and focused on computational aspects of data mining algorithms. However, this research studies the knowledge refreshing problem from a managerial perspective – balancing the trade-off between the loss of knowledge and the cost incurred by running KDD.

This research is also related to previous papers that utilized analytical models to study important design and implementation issues for databases and data warehouses. For example, Chandy et al. (1975) proposed models and techniques for determining optimal
time intervals between checkpoints in a database. Park et al. (1990) introduced models and solution procedures for determining optimal intervals of reorganizing a database system. Segev and Fang (1991) derived and analyzed optimal polices for updating distributed materialized views. Specifically, they proposed policies for determining optimal time intervals between view updates and optimal sources for view updates which incur the minimum updating cost. A recent paper by Dey et al. (2006) studied frequencies for synchronizing a data warehouse with its source databases. The paper proposed polices for determining optimal intervals, measured in number of time units, number of queries to a data warehouse, or number of updates to source databases, to synchronize a data warehouse with its source databases. Compared with previous analytical research on databases and data warehouses, this paper complements the literature in two respects. First, this paper is a first study on knowledge refreshing policies for the KDD process. The new problem context (of KDD) addressed in this research requires introduction of new concepts and a consequently different model structure from models reviewed above. For example, the concept of knowledge loss is introduced in this paper. We also propose how to quantify and estimate knowledge loss. In addition, we introduce a constraint on knowledge loss in the proposed model. The constraint is used to emulate the situation that the percentage of valid knowledge in a knowledge base should be always kept above certain level. For example, it may be practically unacceptable to make decisions based on a knowledge base with 50% of its knowledge invalid/obsolete.
3. Knowledge loss

3.1 Definition of knowledge loss

Before formulating our model for knowledge refreshing, a key concept in knowledge refreshing -- knowledge loss, needs to be defined. Knowledge loss refers to the phenomenon that knowledge discovered by a previous run of KDD becomes obsolete gradually as new data are continuously added in. Specifically, knowledge loss is resulted because 1) part or even all of the earlier discovered knowledge becomes invalid due to incoming new data; and 2) new knowledge brought in by incoming new data is not captured in a previous run of KDD. Given that the latest KDD run before time $t$ occurs at time $t - s$, where $t \geq 0$ and $0 \leq s \leq t$, knowledge loss $l_t$ at time $t$ can be quantified as

$$l_t = 1 - \frac{|K_{t-s} \cap K_t|}{|K_{t-s} \cup K_t|},$$

where $K_{t-s}$ denotes knowledge discovered by running KDD at time $t - s$, $K_t$ denotes knowledge discovered if KDD were run at time $t$, and $|\bullet|$ denotes the cardinality of a set. $|K_{t-s} \cap K_t|$ represents the amount of knowledge discovered at time $t - s$ and still valid at time $t$. $|K_{t-s} \cup K_t|$ can be rewritten as $|K_{t-s} - K_t| + |K_{t-s} \cap K_t| + |K_t - K_{t-s}|$, where $|K_{t-s} - K_t|$ represents the amount of knowledge discovered at time $t - s$ but invalid at time $t$ and $|K_t - K_{t-s}|$ represents the amount of new knowledge brought in by incoming new data.

Note that we apply set operations (i.e., intersection, union and difference) on $K_{t-s}$ and $K_t$ in (1). Knowledge discovered by KDD using any of the three major data mining models can actually be represented as a set of rules or be converted into a set of rules. For
instance, knowledge discovered through association rule mining is either a set of large itemsets or a set of association rules (Agrawal and Srikant 1994); knowledge discovered using classification consists of a set of classification rules (Quinlan 1993) or can be converted into a set of classification rules (Baesens et al. 2003); while knowledge discovered using clustering can be represented as a set of cluster centers, each of which characterizes a cluster (Jain et al. 1999). Clearly, knowledge loss defined in (1) is within the range of [0,1]. \( l_t = 0 \) if and only if \( K_{t-s} = K_t \), which indicates that no previously discovered knowledge are invalidated because of incoming new data and no new knowledge are brought in by incoming new data. On the other hand, \( l_t = 1 \) if and only if \( K_{t-s} \cap K_t = \emptyset \), which indicates that previously discovered knowledge has been fully out of date.

Following example illustrates how to apply (1) for real world applications. Consider recommender systems used in many online retailers. According to (Linden et al. 2003), recommender systems in Amazon.com are powered by co-purchasing knowledge discovered from purchasing transactions at the company. Let \( K_{t-s} = \{\{a,b\},\{c,d\}\} \), which reveals that items \( a \) and \( b \), and items \( c \) and \( d \) are frequently purchased together. Let \( K_t = \{\{c,d\},\{e,f\}\} \). Applying (1), \( l_t = 2/3 \). In this example, due to incoming new purchasing transactions, one co-purchasing knowledge (i.e., \( \{a,b\} \)) disappears at time \( t \) while another co-purchasing knowledge (i.e., \( \{e,f\} \)) emerges at time \( t \).

### 3.2 Estimation of knowledge loss

It is practically difficult to keep track of exact knowledge loss in real time since it is computational too costly to derive up-to-date knowledge instantaneously and
continuously. However, it is possible to keep track of the amount of incoming new data in real time. Knowledge loss may be estimated using the amount of incoming new data since knowledge loss is directly caused by incoming new data. Let \( d_t \) denotes the amount of incoming new data accumulated by time \( t \), where \( d_t \) are nonnegative integers and measured at the lowest granularity of interest (e.g., number of records, or number of blocks). The question is whether we can estimate knowledge loss \( l_t \) at time \( t \) using some function of \( d_t \), i.e., \( l_t = f(d_t) \).

A two-parameter Weibull function shown below is chosen to estimate \( l_t \) from \( d_t \).

\[
l_t = 1 - e^{-\left(\frac{d_t}{\alpha}\right)^\beta} + \varepsilon_i.
\] (2)

In (2) \( \alpha \) and \( \beta \) are parameters. Error terms \( \varepsilon_i \) are independent and identically distributed random variables with mean \( E[\varepsilon_i] = 0 \) and variance \( Var(\varepsilon_i) = \sigma^2 \). Weibull models have been widely applied for modeling product degradation in reliability engineering (Murthy et al., 2003), whereas knowledge loss could be considered as knowledge degradation in the filed of KDD. More importantly, through extensive experiments, we show that the function in (2) fits some real world data sets very well.

Our first set of experiments employs panel data collected by ComScore Media Matrix (www.Comscore.com), an Internet marketing firm that records web navigating behaviors and purchasing transactions of recruited families using client-side programs installed on the families’ home computers. The data used in this research capture 63,999 purchasing transactions conducted by the recruited families during a 6-month time period from July 2002 through December 2002. Products in 29 different product categories, from books and travel to pet supplies, were purchased in these transactions. We applied KDD to the
data to discover co-purchasing knowledge over product categories (e.g., computer electronics and toys are frequently shopped together), which are key knowledge in supporting decisions such as cross-selling and products recommendation. The suitable data mining model used in the KDD process is association rule mining (Agrawal and Srikant, 1994). To observe knowledge loss over time, we divided the data into two parts: the base data, which consisted of the (chronically) first $n_b$ purchasing transactions, and the incremental data, which consisted of the rest of the transactions. By running KDD, we first extracted co-purchasing knowledge from the base data, namely base knowledge. Next, to simulate incoming new data, $n_i$ purchasing transactions from the incremental data were added to the base data, knowledge (namely new knowledge) were extracted from the updated base data, and knowledge loss between base knowledge and new knowledge was calculated using (1). The above-mentioned step continued until all purchasing transactions in the incremental data had been added to the base data. Finally, we obtained a series of knowledge loss and its corresponding amount of incoming new data. For all experiments reported in this paper, the amount of incoming new data was measured in number of transactions. A total of seven experiments were conducted on the panel data. In experiment 1, $n_b$, $n_i$ and support$^1$ were set at 10000, 100, and 0.1% respectively. We varied $n_b$ in experiments 2 and 3, $n_i$ in experiments 4 and 5, and support in experiments 6 and 7. Table 1a lists descriptive statistics of each experiment.

Applying the SAS nonlinear regression procedure (i.e., the SAS NLIN procedure) to the knowledge loss data collected in each experiment, we obtained least squares estimates of the parameters in (2), which were given in Table 1b. For each of the seven experiments,

---

$^1$ Support is a user-specified threshold required in association rule mining. For the details of association rule mining and support, please refer to (Han and Kamber, 2006, pp. 230).
the fit between the function and the observed knowledge loss data is statistically
significant \( p < 0.0001 \) and the function accounts for 99\% of the variance \( R^2 = 0.99 \).

Table 1a Experiments using panel data: descriptive statistics

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Num. of transactions in the base data ( n_b )</th>
<th>Num. of added transactions ( n_i )</th>
<th>Support</th>
<th>Num. of observed knowledge loss</th>
<th>Range of Knowledge loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10,000</td>
<td>100</td>
<td>0.1%</td>
<td>540</td>
<td>0-0.77</td>
</tr>
<tr>
<td>2</td>
<td>7,500</td>
<td>100</td>
<td>0.1%</td>
<td>565</td>
<td>0.08-0.77</td>
</tr>
<tr>
<td>3</td>
<td>12,500</td>
<td>100</td>
<td>0.1%</td>
<td>515</td>
<td>0-0.70</td>
</tr>
<tr>
<td>4</td>
<td>10,000</td>
<td>50</td>
<td>0.1%</td>
<td>1080</td>
<td>0-0.77</td>
</tr>
<tr>
<td>5</td>
<td>10,000</td>
<td>200</td>
<td>0.1%</td>
<td>270</td>
<td>0-0.77</td>
</tr>
<tr>
<td>6</td>
<td>10,000</td>
<td>100</td>
<td>0.075%</td>
<td>540</td>
<td>0-0.69</td>
</tr>
<tr>
<td>7</td>
<td>10,000</td>
<td>100</td>
<td>0.125%</td>
<td>540</td>
<td>0-0.73</td>
</tr>
</tbody>
</table>

Table 1b Experiments using panel data: parameter estimates

<table>
<thead>
<tr>
<th>Experiment</th>
<th>( \alpha )</th>
<th>( \beta )</th>
<th>Variance of errors ( \sigma^2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>27102.9</td>
<td>0.56</td>
<td>0.001</td>
</tr>
<tr>
<td>2</td>
<td>30710.9</td>
<td>0.51</td>
<td>0.001</td>
</tr>
<tr>
<td>3</td>
<td>37911.0</td>
<td>0.60</td>
<td>0.0009</td>
</tr>
<tr>
<td>4</td>
<td>27123.2</td>
<td>0.56</td>
<td>0.001</td>
</tr>
<tr>
<td>5</td>
<td>27109.3</td>
<td>0.55</td>
<td>0.001</td>
</tr>
<tr>
<td>6</td>
<td>36676.3</td>
<td>0.49</td>
<td>0.0007</td>
</tr>
<tr>
<td>7</td>
<td>37272.0</td>
<td>0.67</td>
<td>0.002</td>
</tr>
</tbody>
</table>

The second set of experiments used a different data set and tested the Weibull function on a different data mining model. The data set used in the experiments is a census data set with 45,222 records, which is publicly available at the University of California, Irvine repository for machine learning research (Newman et al., 1998). Each record consists of 14 description attributes (e.g., age, education, and gender) and 1 class label attribute that represents whether or not a person’s income is above $50,000. The objective is to learn from the data set a set of rules (i.e., knowledge) that can be used to classify a person’s
income as either above $50,000 or below $50,000 based on values of the description attributes of the person. An example rule could be “If a person is male and has a professional degree, then his income is above $50,000”. Obviously, the suitable data mining model for this task is classification (Quinlan, 1986). Following the same experimental procedure as that used in the first set of experiments, we obtained a series of knowledge losses and their corresponding amount of incoming new data. Table 2a lists descriptive statistics of each experiment. Applying the SAS NLIN procedure to the knowledge loss data collected in each experiment, least squares estimates of the parameters in (2) were obtained and given in Table 2b. For each of the five experiments, the fit between the function and the observed knowledge loss data is statistically significant ($p < 0.0001$) and the function accounts for 99% of the variance ($R^2 = 0.99$).

**Table 2a Experiments using census data: descriptive statistics**

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Num. of records in the base data ($n_b$)</th>
<th>Num. of added records ($n_a$)</th>
<th>Num. of observed knowledge loss</th>
<th>Range of Knowledge loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>35,000</td>
<td>50</td>
<td>204</td>
<td>0.04-0.88</td>
</tr>
<tr>
<td>2</td>
<td>30,000</td>
<td>50</td>
<td>305</td>
<td>0.02-0.93</td>
</tr>
<tr>
<td>3</td>
<td>40,000</td>
<td>50</td>
<td>105</td>
<td>0.03-0.82</td>
</tr>
<tr>
<td>4</td>
<td>35,000</td>
<td>25</td>
<td>409</td>
<td>0.004-0.88</td>
</tr>
<tr>
<td>5</td>
<td>35,000</td>
<td>100</td>
<td>102</td>
<td>0.06-0.88</td>
</tr>
</tbody>
</table>

**Table 2b Experiments using census data: parameter estimates**

<table>
<thead>
<tr>
<th>Experiment</th>
<th>$\alpha$</th>
<th>$\beta$</th>
<th>Variance of errors ($\sigma^2$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1730.9</td>
<td>0.49</td>
<td>0.001</td>
</tr>
<tr>
<td>2</td>
<td>1961.5</td>
<td>0.58</td>
<td>0.001</td>
</tr>
<tr>
<td>3</td>
<td>2408.6</td>
<td>0.64</td>
<td>0.0006</td>
</tr>
<tr>
<td>4</td>
<td>1726.1</td>
<td>0.49</td>
<td>0.001</td>
</tr>
<tr>
<td>5</td>
<td>1748.1</td>
<td>0.50</td>
<td>0.001</td>
</tr>
</tbody>
</table>
4. Knowledge refreshing: a model and solution methods

4.1 A model for knowledge refreshing

The knowledge refreshing problem studied in this paper is illustrated in Figure 1. At the start of a time horizon, knowledge is discovered from a database by running KDD and stored in a knowledge base. The database evolves over time as new data are continuously populated into it. During the time horizon, running KDD keeps the knowledge base synchronized with the evolving database but incurs the cost of running KDD. Without running KDD, knowledge in the knowledge base becomes obsolete gradually, i.e., knowledge loss. There are requests submitted to the knowledge base to retrieve knowledge for various decision support purposes. For example, knowledge could be employed to support designing marketing strategies or be used by recommender systems for product recommendations. Obviously, obsolete knowledge also incurs cost, namely the cost of knowledge loss. For example, marketing strategies developed based on obsolete knowledge are ineffective or even wrong (Bourgeois and Eisenhardt 1988); products recommended using obsolete knowledge are not likely to be purchased since they are not attractive to consumers. For real-world applications, the following system constraint must be enforced. Knowledge loss should be below \( l_c \) when a request is answered from the knowledge base, where \( 0 < l_c \leq 1 \). An obvious way to satisfy the system constraint is to run KDD whenever knowledge loss of a knowledge base exceeds \( l_c \) and a request is submitted to the knowledge base. The knowledge refreshing problem addressed in this research determines when to execute KDD so that the total system cost, including costs of running KDD and costs of knowledge loss, over a time horizon is
minimized and the system constraint is always satisfied. For the convenience of readers, notation used in this paper is summarized in Table 3.

**Figure 1: The knowledge refreshing problem**

<table>
<thead>
<tr>
<th>Clarity</th>
<th>Clarity</th>
<th>Clarity</th>
<th>Clarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l_c$</td>
<td>$d_m$</td>
<td>$m$</td>
<td>$M$</td>
</tr>
<tr>
<td>System constraint on knowledge loss</td>
<td>Amount of new data accumulated by decision point</td>
<td>Decision points</td>
<td>Number of decision points in a time horizon</td>
</tr>
<tr>
<td>$\mu$</td>
<td>$y_{m-1,m}$</td>
<td>$l_m$</td>
<td>$A$</td>
</tr>
<tr>
<td>Data arrival rate</td>
<td>Amount of new data accumulated between decision points</td>
<td>Knowledge loss at decision point</td>
<td>Action space</td>
</tr>
<tr>
<td>$r_i$</td>
<td>$P_{s_m^m}$</td>
<td>$c(s_m, a_m)$</td>
<td>$A_m$</td>
</tr>
<tr>
<td>Type of request to a knowledge base, $i = 1, 2, \ldots, n$</td>
<td>Transition probability</td>
<td>System cost at decision point</td>
<td>Action at decision point</td>
</tr>
<tr>
<td>$\lambda_i$</td>
<td>$c_{r_i}$</td>
<td>$c_k$</td>
<td>$q_m$</td>
</tr>
<tr>
<td>Arrival rate of request $r_i$</td>
<td>Cost incurred by a type $r_i$ request</td>
<td>Cost of running KDD</td>
<td>Type of request at decision point</td>
</tr>
<tr>
<td>$M$</td>
<td>$\pi$</td>
<td>$\pi$</td>
<td>$q_m$</td>
</tr>
<tr>
<td>Number of decision points in a time horizon</td>
<td>Knowledge refreshing policy</td>
<td>Knowledge refreshing policy</td>
<td>Type of request at decision point</td>
</tr>
<tr>
<td>$A$</td>
<td>$\pi$</td>
<td>$\pi$</td>
<td>$q_m$</td>
</tr>
<tr>
<td>Action space</td>
<td>Knowledge refreshing policy</td>
<td>Knowledge refreshing policy</td>
<td>Type of request at decision point</td>
</tr>
<tr>
<td>$a_m$</td>
<td>$\pi$</td>
<td>$\pi$</td>
<td>$q_m$</td>
</tr>
<tr>
<td>Action chosen at decision point $m$</td>
<td>Knowledge refreshing policy</td>
<td>Knowledge refreshing policy</td>
<td>Type of request at decision point</td>
</tr>
<tr>
<td>$s_m$</td>
<td>$\pi$</td>
<td>$\pi$</td>
<td>$q_m$</td>
</tr>
<tr>
<td>System state at decision point $m$</td>
<td>Knowledge refreshing policy</td>
<td>Knowledge refreshing policy</td>
<td>Type of request at decision point</td>
</tr>
<tr>
<td>$q_m$</td>
<td>$\pi$</td>
<td>$\pi$</td>
<td>$q_m$</td>
</tr>
<tr>
<td>Type of request at decision point $m$</td>
<td>Knowledge refreshing policy</td>
<td>Knowledge refreshing policy</td>
<td>Type of request at decision point</td>
</tr>
</tbody>
</table>

**Table 3 Notation**
The knowledge refreshing problem is modeled as a finite horizon Markov decision process. We assume that the arrival of new data follows a Poisson process with intensity $\mu$. Requests to the knowledge base are classified into different types according to purposes of the requests. Different types of requests have different arrival rates and the same amount of knowledge loss has different impact on different types of requests. For example, requests for knowledge supporting marketing strategy development could have lower arrival rate than requests for knowledge supporting product recommendations; while the same amount of knowledge loss could have higher impact on the former than the latter. We assume that there exists $n$ different types of requests, $\{r_i\}, i = 1, 2, \cdots, n$, $n \geq 1$. Each type of requests is assumed to follow a Poisson process with intensity $\lambda_i$ independently. The robustness of the model with respect to the Poisson arrival assumption is studied numerically in Section 5.

The decision points in the model are the moments when a request to the knowledge base arrives. Let $M$ denote the number of decision points in a time horizon. The system time of the model is labeled as $0, 1, 2, \cdots, M$, with 0 being the starting time of the time horizon, and time $m$, where $1 \leq m \leq M$, denoting the system time of decision point $m$ or the system time when the $m$th request arrives. The action space of the model is defined to be a binary set $A = \{0, 1\}$, where 0 denotes not running KDD and 1 denotes running KDD. Let $a_m \in A$ be the action chosen at time$^2 m$. Since KDD is run at the start of a time horizon, we set $a_0$ to be 1. The system state at time $m$ is represented by a vector $s_m = (q_m, d_m)$, where $q_m$ denotes the type of request arrived at time $m$ and $d_m$ denotes

---

$^2$ In the rest of the paper, if time is used alone, it refers to system time.
the amount of new data accumulated by time \( m \), \( d_m \in \mathbb{Z}^+ \), where \( \mathbb{Z}^+ \) denotes nonnegative integers. Since there is no new data at the start of a time horizon, we set \( d_0 \) to be 0. The set of all feasible request types at time \( m \) is \( \{r_i\}, i = 1, 2, \cdots, n \). 

Let \( y_{m-1,m} \) denote the amount of new data accumulated between time \( m-1 \) and time \( m \), \( y_{m-1,m} \in \mathbb{Z}^+ \). We have,

\[
d_m = \begin{cases} 
    d_{m-1} + y_{m-1,m} & \text{if } a_{m-1} = 0 \\
    y_{m-1,m} & \text{if } a_{m-1} = 1
\end{cases}
\tag{3}
\]

where \( 1 \leq m \leq M \), \( d_0 = 0 \), and \( a_0 = 1 \). If KDD is not run at time \( m-1 \), new data accumulated by time \( m-1 \) is carried forward to the next time point. On the other hand, if KDD is executed at time \( m-1 \), new data accumulated by time \( m-1 \) is absorbed into the updated knowledge base and hence they are not treated as new data at time \( m \).

Let \( l_m \) denote knowledge loss at time \( m \). The system constraint can be written as,

\[
\text{if } l_m \geq l_c \quad \text{then } a_m = 1. \tag{4}
\]

Estimating \( l_m \) using (2), (4) can be rewritten as³,

\[
\text{if } 1 - e^{-\frac{d_m}{\alpha}} \geq l_c \quad \text{then } a_m = 1. \tag{5}
\]

Solving the inequality in (5) for \( d_m \), we have,

\[
\text{if } d_m \geq \alpha \left( \ln \left( \frac{1}{1 - l_c} \right) \right)^{1/\beta} \quad \text{then } a_m = 1. \tag{6}
\]

Considering that \( d_m \) takes only integral values, we have,

³ In (5), we omit the error term for the sake of model tractability, since it is a small random error with mean equal to 0.
\[
\text{if } d_m \geq d_c \quad \text{then } a_m = 1,
\]
where \( d_c = \left\lfloor \alpha(\ln\left(\frac{1}{1-l_c}\right))^\frac{1}{\beta} \right\rfloor. \)

Taking the system constraint into consideration, the state space of \( d_m \) can be written as \( \{0,1,\cdots,d_c - 1, d_c, d_c + 1, \cdots\} \). If \( d_m \) equals to any state in \( \{d_c, d_c + 1, \cdots\} \), only one action can be taken (according to (7)) and the amount of new data accumulated by time \( m+1 \), \( d_{m+1} \), is always \( y_{m,m+1} \) (according to (3)). Hence, we can create an “integrated” state \( D_c = \{d_c, d_c + 1, \cdots\} \). The state space of \( d_m \) can be rewritten as \( \{0,1,\cdots,d_c - 1, D_c\} \).

Having discussed the state space of the model, we next derive transition probability \( P_{s_m,s_{m+1}}^{a_m} \) from system state \( s_m = (q_m, d_m) \) at time \( m \) to system state \( s_{m+1} = (q_{m+1}, d_{m+1}) \) at time \( m+1 \) under action \( a_m \) using the following lemmas.

**Lemma 1.** The joint probability mass function of \( q_m \) and \( y_{m-1,m} \) is
\[
P\{q_m = r, y_{m-1,m} = k\} = \frac{\mu^k \lambda^i}{(\mu + \sum_{j=1}^{n} \lambda_j)^{k+1}} \quad i = 1,2,\cdots,n \quad \text{and } k \in Z^+
\]  

**Proof.** Let \( T_{m-1,m} \) denote the elapsed chronicle time between decision points \( m-1 \) and \( m \).

According to the Poisson arrival assumption for queries,
\[
P\{T_{m-1,m} \leq \tau, q_m = r_i\} = \int_0^\tau \lambda_i e^{-\lambda_i x} \prod_{j=1, j \neq i} \frac{n}{\lambda_j} e^{-\lambda_j x} dx
\]
\[
= \frac{\lambda_i}{\sum_{j=1}^{n} \lambda_j} \left(1 - e^{-\sum_{j=1}^{n} \lambda_j \tau}\right).
\]
Differentiating (9) on $\tau$, the probability density function of $T_{m-1,m}$, given that type $r_i$ request arrives at time $m$, is

$$f_{T_{m-1,m}}(\tau) = \lambda_i e^{-\sum_{j=1}^{n} \lambda_j \tau}.$$ \hspace{1cm} (10)

According to the Poisson arrival assumption for new data,

$$P\{q_m = r_i, y_{m-1,m} = k\} = \int_{0}^{\infty} e^{-\mu \tau} \frac{(\mu \tau)^k}{k!} f_{T_{m-1,m}}(\tau) d\tau$$

$$= \int_{0}^{\infty} e^{-\mu \tau} \frac{(\mu \tau)^k}{k!} \lambda_i e^{-\sum_{j=1}^{n} \lambda_j \tau} d\tau \quad \text{by applying (10)}$$

$$= \frac{(\mu)^k \lambda_i}{(\mu + \sum_{j=1}^{n} \lambda_j)^{k+1}}. \quad \text{Q.E.D.}$$

The following two lemmas, which are derived based on Lemma 1, specify system transition probabilities from $s_m = (q_m, d_m)$ to $s_{m+1} = (q_{m+1}, d_{m+1})$ under action $a_m = 0$ and $a_m = 1$ respectively. Note that, in Lemma 2, $d_m$ can only take values from \{0,1,\cdots,d_c-1\} because action $a_m = 1$ must be enforced if $d_m \geq d_c$.

**Lemma 2.** System transition probabilities under action $a_m = 0$ are given by,

$$P\{q_{m+1} = r_i, d_{m+1} = k_2 \mid q_m = r_j, d_m = k_1, a_m = 0\} = \begin{cases} \frac{\mu^{k_2-k_1} \lambda_i}{(\mu + \sum_{j=1}^{n} \lambda_j)^{k_2-k_1+1}} & \text{if } k_2 \geq k_1 \\ 0 & \text{if } k_2 < k_1 \end{cases}$$ \hspace{1cm} (11)

where $i, j = 1, 2, \cdots, n$ and $k_1, k_2 \in \{0,1,\cdots,d_c-1\}$;

$$P\{q_{m+1} = r_i, d_{m+1} \in D_c \mid q_m = r_j, d_m = k_1, a_m = 0\} = \frac{\lambda_i}{\sum_{j=1}^{n} \lambda_j} \left( \frac{\mu}{\mu + \sum_{j=1}^{n} \lambda_j} \right)^{d_c-k_1}$$ \hspace{1cm} (12)

where $i, j = 1, 2, \cdots, n$ and $k_1 \in \{0,1,\cdots,d_c-1\}$.
Proof. Let us first prove (11). If $k_2 \geq k_1$, according to (3) and the independent arrival assumption for requests,

$$
P\{q_{m+1} = r_i, d_{m+1} = k_2 \mid q_m = r_j, d_m = k_1, a_m = 0\} = P\{q_{m+1} = r_i, y_{m,m+1} = k_2 - k_1\} = \frac{\mu^{k_2-k_1} \lambda_j}{(\mu + \sum_{j=1}^n \lambda_j)^{k_2-k_1}} \quad \text{by applying Lemma 1}
$$

If $k_2 < k_1$, the transition probability given in (11) is 0, since, under action $a_m = 0$, the accumulated amount of new data always increase over time. We then prove (12).

$$
P\{q_{m+1} = r_i, d_{m+1} \in D_c \mid q_m = r_j, d_m = k_1, a_m = 0\} = \sum_{k_2 \in D_c} P\{q_{m+1} = r_i, d_{m+1} = k_2 \mid q_m = r_j, d_m = k_1, a_m = 0\} = \sum_{k_2 \in D_c} \frac{\mu^{k_2-k_1} \lambda_j}{(\mu + \sum_{j=1}^n \lambda_j)^{k_2-k_1}} \quad \text{by applying Lemma 1}
$$

$$
= \frac{\lambda_i}{\sum_{j=1}^n \lambda_j} \left( \frac{\mu}{\mu + \sum_{j=1}^n \lambda_j} \right)^{d_c-k_1} \cdot \text{Q.E.D.}
$$

Lemma 3. System transition probabilities under action $a_m = 1$ are given by,

$$
P\{q_{m+1} = r_i, d_{m+1} = k_2 \mid q_m = r_j, d_m = k_1, a_m = 1\} = \frac{\mu^{k_2} \lambda_i}{(\mu + \sum_{j=1}^n \lambda_j)^{k_2+1}} \quad (13)
$$

where $i, j = 1, 2, \cdots, n$ and $k_1 \in Z^+, k_2 \in \{0, 1, 2, \cdots, d_c - 1\}$;

$$
P\{q_{m+1} = r_i, d_{m+1} \in D_c \mid q_m = r_j, d_m = k_1, a_m = 1\} = \frac{\lambda_i}{\sum_{j=1}^n \lambda_j} \left( \frac{\mu}{\mu + \sum_{j=1}^n \lambda_j} \right)^{d_c} \quad (14)
$$

where $i, j = 1, 2, \cdots, n$ and $k_1 \in Z^+$. 
**Proof.** Let us first prove (13). According to (3) and the independent arrival assumption for requests,

\[
P\{q_{m+1} = r_i, d_{m+1} = k_2 \mid q_m = r_j, d_m = k_1, a_m = 1\} = P\{q_{m+1} = r_i, y_{m,m+1} = k_2\}
\]

\[
= \frac{\mu^{k_2} \lambda_i}{(\mu + \sum_{j=1}^{n} \lambda_j)^{k_2+1}}.
\]

by applying Lemma 1

We then prove (14).

\[
P\{q_{m+1} = r_i, d_{m+1} \in D_c \mid q_m = r_j, d_m = k_1, a_m = 1\} = \sum_{k_2=1}^{m} P\{q_{m+1} = r_i, d_{m+1} = k_2 \mid q_m = r_j, d_m = k_1, a_m = 1\}
\]

\[
= \sum_{k_2=1}^{m} \frac{\mu^{k_2} \lambda_i}{(\mu + \sum_{j=1}^{n} \lambda_j)^{k_2+1}}
\]

by applying Lemma 1

\[
= \frac{\lambda_i}{\sum_{j=1}^{n} \lambda_j} \left( \frac{\mu}{\mu + \sum_{j=1}^{n} \lambda_j} \right)^{d_c}.
\]

Q.E.D.

The system cost \(c(s_m, a_m)\) incurred at time \(m\) depends on the system state and the action chosen at time \(m\). If not running KDD at time \(m\), request at time \(m\) suffers from the cost of knowledge loss, denoted as \(c^l_m\). On the other hand, running KDD at time \(m\) saves the cost of knowledge loss but incurs the cost of running KDD, denoted as \(c^k_m\). In general, \(c(s_m, a_m)\) can be expressed as,

\[
c(s_m, a_m) = \begin{cases} 
  c^l_m & \text{if } a_m = 0 \\
  c^k_m & \text{if } a_m = 1
\end{cases}.
\]

(15)
Let \( c_{r_i} \) denote the cost incurred if a type \( r_i \) request is answered from a fully out-of-dated knowledge base (i.e., knowledge loss is 1). Estimation of \( c_{r_i} \) is application dependent. Considering a type \( r_i \) request that retrieves co-purchasing knowledge for product recommendations, \( c_{r_i} \) may be estimated as potential revenue loss resulted from using fully out-of-dated knowledge. \( c_{m}^{i} \) is modeled as \( c_{q_m}(l_m)^{\phi} \), where \( q_m \in \{r_i\}, i = 1, 2, \ldots, n \), and \( \phi \) is a shape parameter, \( \phi > 0 \). Shape parameter \( \phi \) determines how \( c_{m}^{i} \) increases as \( l_m \) increases. If \( \phi < 1 \), the increase rate of \( c_{m}^{i} \) decreases as \( l_m \) increases, while the increase rate of \( c_{m}^{i} \) increases as \( l_m \) increases if \( \phi > 1 \). The increase rate of \( c_{m}^{i} \) is constant as \( l_m \) increases if \( \phi = 1 \). The value of \( \phi \) is also application dependent. According to (2),

\[
l_m \text{ can be estimated as } 1 - e^{-\frac{d_m}{\alpha}}. \text{ Therefore,}
\]

\[
c_{m}^{i} = c_{q_m}(1 - e^{-\frac{d_m}{\alpha}})^{\phi}.
\]  

We model the cost \( c_{m}^{k} \) of running KDD at decision point \( m \) as a fixed cost \( c^{k} \). Although the computational cost of running KDD is proportional to the size of the mined data, the personnel cost of running KDD typically does not vary much with data volume. The latter dominates the cost of running KDD. However, note that the model can easily be generalized to vary \( c_{m}^{k} \) at different time. For real world applications, \( c^{k} \) can be estimated using the number of man-hours involved in a KDD process times pay rates.
A knowledge refreshing policy $\pi$ is defined as $\pi = (\delta_1, \delta_2, \cdots, \delta_m, \cdots, \delta_M)$, where $\delta_m$ denotes the decision rule at time $m$ such that $a_m = \delta_m(s_m)$. The expected total system cost over a time horizon under a refreshing policy $\pi$, $EC_\pi$, can be expressed as,

$$EC_\pi = E \left\{ \sum_{i=1}^{M} [\delta_i(s_i)c_i^k + (1 - \delta_i(s_i))c_i^l] \right\}$$

The optimal knowledge refreshing policy, $\pi^*$, minimizes the expected total system cost over the time horizon such that $EC_{\pi^*} = \min_{\pi \in \Pi} EC_\pi$, where policy space $\Pi$ is the set of all feasible knowledge refreshing policies.

4.2 Solution methods

An exhaustive enumeration of all knowledge refreshing policies in $\Pi$ to search for the optimal one requires a time complexity $O(2^M)$. The exponential nature of the method discourages its application when $M$ is large. Dynamic programming methods normally solve Markov decision process problems in polynomial time (Sutton and Barton, 1998). Therefore, we employ value iteration, a dynamic programming method, for solving the Markov decision process model developed in subsection 4.1.

Let $v_{s_m}(m)$ be the optimal expected total system cost from time $m$, with state $s_m$, to the end of a time horizon, where $1 \leq m \leq M$. By the principle of dynamic optimality (Bellman and Dreyfus 1962),

$$v_{s_m}(m) = \min_{a_m \in A} \{c(s_m, a_m) + \sum_{s_{m+1} \in S} P_{s_m a_m s_{m+1}} v_{s_{m+1}}(m+1)\},$$

(18)
where $S$ is the state space, and $v_{s_{m+1}}(m+1)$ be the optimal expected total cost from time $m+1$, with state $s_{m+1}$, to the end of the time horizon. The value iteration method solves the recursive equation (18) from $m = M$ to $m = 1$, given $v_{s_{M+1}}(M+1) = 0$ for all $s_{M+1} \in S$. The method finally obtains $v_{s_1}(1)$ for all $s_1 \in S$. At the same time, the optimal knowledge refreshing policy $\pi^*$ is also derived. The decision rules in $\pi^*$ have a general format of “IF the system state at the current decision point is ..., THEN the optimal action is ...”.

The optimal expected total system cost over the time horizon $EC_{\pi^*}$ is given by,

$$EC_{\pi^*} = \sum_{s_1 \in S} P_{s_1} v_{s_1}(1),$$  \hspace{1cm} (19)

where $P_{s_1}$ is the probability of system state being $s_1$ at time 1 and $s_1 = (q_1, d_1)$. $P_{s_1}$ can be calculated using Lemma 4.

**Lemma 4.** The probability $P_{s_1}$ of system state being $s_1$ at time 1 is given by,

$$P_{s_1} = P\{q_1 = r_i, d_1 = k\} = \frac{\mu^k \lambda_i}{(\mu + \sum_{j=1}^n \lambda_j)^{k+1}} \quad \text{where } i = 1, 2, \cdots, n \text{ and } k \in \{0,1,2,\cdots,d_c - 1\} \hspace{1cm} \text{(20)}$$

$$P_{s_1} = P\{q_1 = r_i, d_1 \in D_c\} = \frac{\lambda_i}{\sum_{j=1}^n \lambda_j} \left( \frac{\mu}{\mu + \sum_{j=1}^n \lambda_j} \right)^{d_c} \quad \text{where } i = 1, 2, \cdots, n. \hspace{1cm} \text{(21)}$$

**Proof.** We first prove (20).

$$P_{s_1} = P\{q_1 = r_i, d_1 = k\}$$

$$= P\{q_1 = r_i, y_{0,1} = k\} \quad \text{by applying (3)}$$

$$= \frac{\mu^k \lambda_i}{(\mu + \sum_{j=1}^n \lambda_j)^{k+1}}. \quad \text{by applying Lemma 1}$$

24
We next prove (21).

\[
P_{s_i} = P\{q_1 = r_i, d_1 \in D_c \}
\]

\[
= \sum_{k \in D_c} P\{q_1 = r_i, d_1 = k \}
\]

\[
= \sum_{k \in D_c} P\{q_1 = r_i, y_{0,1} = k \} \quad \text{by applying (3)}
\]

\[
= \sum_{k \in D_c} \frac{\mu^k \lambda_i}{(\mu + \sum_{j=1}^{n} \lambda_j)^{k+1}} \quad \text{by applying Lemma 1}
\]

\[
= \frac{\lambda_i}{\sum_{j=1}^{n} \lambda_j} \left( \frac{\mu}{\mu + \sum_{j=1}^{n} \lambda_j} \right)^{d_c}. \quad \text{Q.E.D.}
\]

The optimal knowledge refreshing policy is derived under perfect estimations of system parameters. Hence, it is of limited utility when addressing real world problems. For example, data arrival rate may change over time. As a result, the optimal policy derived based on old data arrival rate may not be optimal anymore after the change. In light of this consideration, we propose an adaptive heuristic for knowledge refreshing based on the model developed in subsection 4.1. In contrast to the value iteration method which generates optimal actions for the entire time horizon, the adaptive heuristic recommends actions only for the current decision point. The adaptive heuristic decides on action using the most recent estimates of future system parameters so as to act on changes in system parameters. Consequently, the adaptive heuristic is adaptive in nature, and the performance of the heuristic is dependent on the accuracy of the estimates. The proposed adaptive heuristic considers estimating future data arrival rate and future request arrival rates, system parameters which are more often and more likely to change over time than other system parameters. However, the heuristic can be easily extended to address changes of other system parameters.
We employ a single exponential smoothing method to derive an estimate of future data arrival rate at each decision point. According to the forecasting method in (22), the new estimate $\mu^{new}$ for data arrival rate is derived as a weighted sum of the old estimate $\mu^{old}$ for data arrival rate and the actual data arrival rate $\mu^a$ observed.

\[ \mu^{new} = \omega \mu^a + (1 - \omega) \mu^{old}, \]  

(22)

where $\omega$ is the smoothing constant and $0 < \omega \leq 1$. Similarly, the new estimate $\lambda_i^{new}$ for arrival rate of request $r_i$ can be derived as,

\[ \lambda_i^{new} = \omega \lambda_i^a + (1 - \omega) \lambda_i^{old}, \]  

(23)

where $\lambda_i^{old}$ is old estimate for arrival rate of request $r_i$ and $\lambda_i^a$ is the actual arrival rate of request $r_i$. The duration $T$ of a planning time horizon can be estimated as the duration of a time period, in which newly estimated data arrival rate and request arrival rates are likely to be stable. The number of decision points $M$ over the planning time horizon can be approximated as $M = T \sum_{i=1}^{n} \lambda_i^{new}$. The adaptive heuristic is outlined in Figure 2.
Input: Duration of a planning time horizon, $T$
Current system state, $s \in S$
Smoothing constant, $\omega$
Other system parameters

Output: Action $a \in A$

Estimate current knowledge loss $l$ using (2)

If $l \geq l_c$

\[ a = 1 \]

Return $a$

Else

Estimate $\mu^{\text{new}}$ using (22)

Estimate $\lambda_i^{\text{new}}$ using (23) for $i = 1, 2, \ldots, n$

Approximate $M$ as $M = T \sum_{i=1}^{n} \lambda_i^{\text{new}}$

Derive $a_1$ by solving (18) from $m = M$ to $m = 1$, given $s_1 = s$

\[ a = a_1 \]

Return $a$

End if

Figure 2: Sketch of the adaptive heuristic method

5. Numerical analysis

The objective of this section is to examine the robustness and the effectiveness of the optimal knowledge refreshing policy and the adaptive heuristic method through simulation. We first describe the design of the simulation, and then show the simulation results, discussing the robustness of the optimal knowledge refreshing policy with respect to the Poisson arrival assumption. We also show the effectiveness of both the optimal knowledge refreshing policy and the adaptive heuristic method. Each simulation result presented in this section represents an average of results obtained from 500 simulation runs.
5.1 Simulation design

In this subsection we assign values to major system parameters based on the panel data used in subsection 3.2. \( \alpha \) and \( \beta \) in the Weibull function are set to be 27102.9 and 0.56 respectively, which are the estimates of \( \alpha \) and \( \beta \) in our first experiment with panel data.

In the simulation, a chronicle time unit is used to simulate a day and the amount of new data is measured in number of transactions. The data arrival rate \( \mu \) is set to be 348 (i.e., 63999/184), since the panel data consist of 63999 transactions arrived within a 6-month period (i.e., 184 days). We simulate three different types of requests \( r_1 \), \( r_2 \) and \( r_3 \), which generally arrive daily, weekly, and bi-weekly respectively. Hence, we set arrival rates of these request \( \lambda_1 \), \( \lambda_2 \) and \( \lambda_3 \) as 1, 1/7, and 1/14 respectively. The costs of these requests \( c_{r_1} \), \( c_{r_2} \) and \( c_{r_3} \) are assumed to be 1500, 4500, and 18000 respectively. The costs are set to emulate the scenario that requests with higher cost arrive less frequently. We assume that one run of KDD requires 20 man-hours with an average pay rate 35. Hence, the cost of running KDD \( c^k \) is set to be 700 (i.e., 35×20). Values of major system parameters are summarized in Table 4.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha )</td>
<td>27102.9</td>
</tr>
<tr>
<td>( \beta )</td>
<td>0.56</td>
</tr>
<tr>
<td>( \mu )</td>
<td>348</td>
</tr>
<tr>
<td>( \lambda_1 )</td>
<td>1</td>
</tr>
<tr>
<td>( \lambda_2 )</td>
<td>1/7</td>
</tr>
<tr>
<td>( \lambda_3 )</td>
<td>1/14</td>
</tr>
<tr>
<td>( c_{r_1} )</td>
<td>1500</td>
</tr>
<tr>
<td>( c_{r_2} )</td>
<td>4500</td>
</tr>
<tr>
<td>( c_{r_3} )</td>
<td>18000</td>
</tr>
<tr>
<td>( c^k )</td>
<td>700</td>
</tr>
</tbody>
</table>
5.2 Robustness analysis

To examine the robustness of the optimal knowledge refreshing policy with respect to the Poisson arrival assumption, several sets of simulation experiments were performed in which different distributions of interarrival times were used to determine arrivals of data and requests. For each set of experiment, the optimal knowledge refreshing policy was generated based on the Poisson arrival assumption. The generated policy was then implemented in an experiment with exponentially distributed interarrival times of data and requests and in several experiments with non-exponentially distributed interarrival times of data and requests. We carefully selected non-exponential distributions with varied coefficients of variation (CV). The selected non-exponential distributions, listed in the order of increasing CV, include Erlang-4 (CV=0.5), uniform (CV=0.58), Erlang-2 (CV=0.71), and hyperexponential (CV=1.41) distributions. Both the exponential distribution and the non-exponential distributions are ensured to have the same mean value.

For each set of simulation experiments, major system parameters are set as those listed in Table 4. In experiment set 1, the duration of a time horizon $T$, knowledge loss constraint $l_c$, and shape parameter $\phi$ were set to be 180 time units, 0.2, and 1 respectively. The difference between the simulated performance of an optimal knowledge refreshing policy implemented under non-exponential interarrival times of data and requests and the simulated performance of the policy implemented under exponential interarrival times of data and requests, represents the effect of the Poisson arrival assumption on the optimal knowledge refreshing policy. Specifically, the difference is
measured as \( \text{diff} = \frac{EC^P_\pi - EC^{NP}_\pi}{EC^P_\pi} \), where \( EC^P_\pi \) is the total system cost resulted from implementing an optimal knowledge refreshing policy under exponential interarrival times of data and requests, and \( EC^{NP}_\pi \) is the total system cost resulted from implementing the policy under non-exponential interarrival times of data and requests.

We varied \( T \) in experiment sets 2 and 3, \( l_c \) in experiment sets 4 and 5, and \( \phi \) in experiment sets 6 and 7. The simulation results in Table 5 showed that the differences between the total system cost under each non-exponential distribution and that under the exponential distribution are very small. The results indicated that the Poisson arrival assumption has little impact on the robustness of the optimal knowledge refreshing policy.

| \( T \) | \( l_c \) | \( \phi \) | Exponential (cost) | Exponential (diff. (%)) | Erlang-2 | Erlang-4 | Hyperexponential | Uniform | diff. (%)
--|--|--|--|--|--|--|--|--|--|--|
180 | 0.2 | 1 | 62652.88 | 63144.24 | 3.2 | 63528.29 | 0.8 | 63247.58 | 1.0 |
90 | 0.2 | 1 | 31407.26 | 31266.52 | 0.5 | 31460.10 | 0.2 | 31270.62 | 0.4 |
360 | 0.2 | 1 | 125718.91 | 126372.30 | 0.5 | 127141.31 | 1.1 | 126154.57 | 0.3 |
180 | 0.15 | 1 | 63962.37 | 65741.76 | 2.8 | 66519.31 | 4.0 | 65226.10 | 2.0 |
180 | 0.25 | 1 | 62640.62 | 63754.31 | 1.8 | 63121.81 | 0.8 | 63046.35 | 0.7 |
180 | 0.2 | 0.5 | 112753.27 | 114079.20 | 1.2 | 115943.97 | 2.8 | 114227.13 | 1.3 |
180 | 0.2 | 2 | 28830.88 | 29549.86 | 2.5 | 29776.15 | 3.3 | 29521.82 | 2.4 |

5.3 Effectiveness analysis for the optimal knowledge refreshing policy

In this subsection, the optimal knowledge refreshing policy (hereafter called the optimal policy) is compared with a fixed-interval knowledge refreshing policy (hereafter called a fixed-interval policy), a knowledge refreshing policy popularly applied in practice. Under
a fixed-interval policy, KDD is executed after the passage of a fixed length of interval, e.g., running KDD after every three requests to a knowledge base. Specifically, the cost saving generated by the optimal policy over the best-performed fixed-interval policy, which incurs the least cost among all fixed-interval policies, is examined through simulation experiments. Experimental setups are same as those described in subsection 5.2. The cost saving generated by the optimal policy is measured as

$$\frac{|EC_{\pi^*} - EC_{\pi_f}|}{EC_{\pi_f}},$$

where $EC_{\pi^*}$ is the total system cost resulted from implementing an optimal policy, and $EC_{\pi_f}$ is the total system cost resulted from implementing a best-performed fixed-interval policy. $EC_{\pi_f}$ is obtained through exhaustive search in costs incurred by all possible fixed-interval policies.

As shown in Table 6, the optimal policy outperforms the best-performed fixed-interval policy in all the experiments. The superiority of the optimal policy is attributed to the fact that the optimal policy is globally optimal among all possible knowledge refreshing policies while the best-performed fixed-interval policy is the best policy within the reduced policy space containing only fixed-interval knowledge refreshing policies. The optimal policy normally saves cost by around 20% with two exceptions, where $l_c$ equals to 0.15 or $\phi$ equals to 2. It is, therefore, interesting to examine the impact of $l_c$ and $\phi$ on the cost saving generated by the optimal policy.
Table 6 Comparisons between the optimal knowledge refreshing policy and the best performed fixed-interval knowledge refreshing policy

<table>
<thead>
<tr>
<th>$T$</th>
<th>$l_c$</th>
<th>$\phi$</th>
<th>$EC_{\pi^*}$</th>
<th>$EC_{\pi_f}$</th>
<th>Cost saving (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>180</td>
<td>0.2</td>
<td>1</td>
<td>62652.88</td>
<td>80840.20</td>
<td>22.5</td>
</tr>
<tr>
<td>90</td>
<td>0.2</td>
<td>1</td>
<td>31407.26</td>
<td>39190.17</td>
<td>19.9</td>
</tr>
<tr>
<td>360</td>
<td>0.2</td>
<td>1</td>
<td>125718.91</td>
<td>165495.20</td>
<td>24.0</td>
</tr>
<tr>
<td>180</td>
<td>0.15</td>
<td>1</td>
<td>63962.37</td>
<td>75242.63</td>
<td>15.0</td>
</tr>
<tr>
<td>180</td>
<td>0.25</td>
<td>1</td>
<td>62640.62</td>
<td>81574.13</td>
<td>23.2</td>
</tr>
<tr>
<td>180</td>
<td>0.2</td>
<td>0.5</td>
<td>112753.27</td>
<td>149016.16</td>
<td>24.3</td>
</tr>
<tr>
<td>180</td>
<td>0.2</td>
<td>2</td>
<td>28830.88</td>
<td>29446.18</td>
<td>2.1</td>
</tr>
</tbody>
</table>

Keeping $T$ at 180 and $\phi$ at 1, we conducted additional simulation experiments with $l_c$ increased from 0.05 to 0.35 by a step of 0.05. As shown in Table 7, the cost saving improves as $l_c$ increases from 0.05; however it becomes stable after $l_c$ reaches 0.3. To explain the phenomenon, we further obtained the percentage of KDD runs enforced by the system constraint (hereafter called the percentage) for each value of $l_c$. The percentages are shown in the last column of Table 7. At a decision point in a time horizon, a KDD run is either enforced by the system constraint if knowledge loss at the decision point exceeds $l_c$ or triggered by an optimal decision that running KDD optimizes system cost from the decision point to the end of the time horizon. The percentage is the ratio of the number of KDD runs enforced by the system constraint out of the total number of KDD runs in a time horizon. It is clear that the percentage decreases as $l_c$ increases. It finally stays at zero after some value of $l_c$ (in this case 0.3). As the percentage decreases, more KDD runs are triggered by optimal decisions. Therefore, the advantage of the optimal policy over the best-performed fixed-interval policy increases, which is reflected in the increase of the cost saving. The increase of the cost saving stops after the percentage decreases to its limit (i.e., 0). The finding on the
impact of $l_c$ suggests that setting the value of $l_c$ for a real world application needs to balance the performance of the optimal policy and the requirements of the application. Setting $l_c$ too tight may damage the performance of the optimal policy while setting $l_c$ too loose may not be accepted by the application.

Table 7 The impact of $l_c$ on the cost saving

<table>
<thead>
<tr>
<th>$l_c$</th>
<th>$EC_{\pi^*}$</th>
<th>$EC_{\pi_f}$</th>
<th>Cost saving (%)</th>
<th>Percentage of KDD runs enforced by the system constraint (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.05</td>
<td>109314.74</td>
<td>109451.94</td>
<td>0.1</td>
<td>98.5</td>
</tr>
<tr>
<td>0.1</td>
<td>75277.48</td>
<td>79531.88</td>
<td>5.3</td>
<td>88.1</td>
</tr>
<tr>
<td>0.15</td>
<td>63962.37</td>
<td>75242.63</td>
<td>15.0</td>
<td>64.4</td>
</tr>
<tr>
<td>0.2</td>
<td>62652.88</td>
<td>80840.20</td>
<td>22.5</td>
<td>25.6</td>
</tr>
<tr>
<td>0.25</td>
<td>62640.62</td>
<td>81574.13</td>
<td>23.2</td>
<td>0.1</td>
</tr>
<tr>
<td>0.3</td>
<td>62128.55</td>
<td>81411.06</td>
<td>24.0</td>
<td>0</td>
</tr>
<tr>
<td>0.35</td>
<td>62075.05</td>
<td>81184.62</td>
<td>24.0</td>
<td>0</td>
</tr>
</tbody>
</table>

We then studied the impact of $\phi$ on the cost saving. Keeping $T$ at 180 and $l_c$ at 0.2, we conducted simulation experiments with $\phi$ varied from 0.1 to 2.25. Simulation results summarized in Table 8 show that the variation of the cost saving with increasing $\phi$ follows a bell-shaped curve. The cost saving starts increasing from $\phi=0.25$, it reaches its peak when $\phi=0.75$ and begins decreasing. The bell-shaped curve can be explained by the following two boundary conditions, under which the cost saving generated by the optimal policy equals to 0 (i.e., $EC_{\pi_f} = EC_{\pi^*}$). If, at any decision point over a time horizon, the cost of knowledge loss is not less than the cost of running KDD (boundary condition 1), KDD is required to be executed at every decision point by both the optimal policy and the best-performed fixed-interval policy. Therefore, $EC_{\pi_f}$ equals to $EC_{\pi^*}$ under boundary
condition 1. If the total cost of knowledge loss throughout a time horizon is less than the cost of running KDD (boundary condition 2), neither the optimal policy nor the best-performed fixed-interval policy suggests running KDD throughout the time horizon. Hence, $EC_{\pi_j}$ also equals to $EC_{\pi}$ under boundary condition 2. If $\phi$ is small enough, boundary condition 2 tends to be satisfied. On the other hand, if $\phi$ is large enough, boundary condition 1 tends to be satisfied. As shown in Table 8, $EC_{\pi}$ is close to $EC_{\pi_j}$ when $\phi=0.1$ or $\phi=2.25$.

<table>
<thead>
<tr>
<th>$\phi$</th>
<th>$EC_{\pi}$</th>
<th>$EC_{\pi_j}$</th>
<th>Cost saving (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>150993.50</td>
<td>153929.50</td>
<td>1.9</td>
</tr>
<tr>
<td>0.25</td>
<td>146637.45</td>
<td>153699.00</td>
<td>4.6</td>
</tr>
<tr>
<td>0.5</td>
<td>112753.27</td>
<td>149016.16</td>
<td>24.3</td>
</tr>
<tr>
<td>0.75</td>
<td>83023.17</td>
<td>110937.89</td>
<td>25.2</td>
</tr>
<tr>
<td>1</td>
<td>62652.88</td>
<td>80840.20</td>
<td>22.5</td>
</tr>
<tr>
<td>1.25</td>
<td>48513.22</td>
<td>58281.33</td>
<td>16.8</td>
</tr>
<tr>
<td>1.5</td>
<td>39666.55</td>
<td>44584.82</td>
<td>11.0</td>
</tr>
<tr>
<td>1.75</td>
<td>33029.09</td>
<td>35118.98</td>
<td>6.0</td>
</tr>
<tr>
<td>2</td>
<td>28830.88</td>
<td>29446.18</td>
<td>2.1</td>
</tr>
<tr>
<td>2.25</td>
<td>25926.75</td>
<td>26015.81</td>
<td>0.3</td>
</tr>
</tbody>
</table>

**Table 8 The impact of $\phi$ on the cost saving**

5.4 Effectiveness analysis for the adaptive heuristic method

The effectiveness of the adaptive heuristic method is studied under a system environment, where the data arrival rate $\mu$ changes over time. The duration of a time horizon $T$ is set to be 180 time units. Major system parameters are set as those listed in Table 4, except for $\mu$. The data arrival rate $\mu$ is set to be 240, 360, and 240 during the first, the second, and the last 60 time units respectively. The heuristic method is
implemented under the assumption that only the average data arrival rate for the entire horizon is known in advance. In this case, the data arrival rate known at the start of the time horizon is 280, and the smoothing constant \( \omega \) is set to be 0.5. Table 9 summarizes the simulation results under various values of \( \phi \) and \( l_c \). The cost saving generated by the heuristic method is measured as \( \frac{E C_{h} - E C_{\pi_f}}{E C_{\pi_f}} \), where \( E C_{h} \) is the total system cost resulted from implementing a heuristic method, and \( E C_{\pi_f} \) is the total system cost resulted from implementing a best-performed fixed-interval policy. The heuristic method outperforms the best-performed fixed-interval policy in all the simulation experiments. Since both the heuristic method and the optimal policy are built on the same model, the variation of the cost saving generated by the heuristic method is similar to the variation of the cost saving generated by the optimal policy, which is explained in subsection 5.3.

<table>
<thead>
<tr>
<th>( \phi )</th>
<th>( l_c )</th>
<th>( E C_{h} )</th>
<th>( E C_{\pi_f} )</th>
<th>Cost saving (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.2</td>
<td>59773.85</td>
<td>70578.11</td>
<td>15.3</td>
</tr>
<tr>
<td>1</td>
<td>0.15</td>
<td>60623.07</td>
<td>67012.67</td>
<td>9.5</td>
</tr>
<tr>
<td>1</td>
<td>0.25</td>
<td>59451.29</td>
<td>71671.23</td>
<td>17.1</td>
</tr>
<tr>
<td>0.5</td>
<td>0.2</td>
<td>114563.70</td>
<td>140216.50</td>
<td>18.3</td>
</tr>
<tr>
<td>2</td>
<td>0.2</td>
<td>24166.00</td>
<td>24485.87</td>
<td>1.3</td>
</tr>
</tbody>
</table>

**Table 9 Comparisons between the adaptive heuristic method and the best performed fixed-interval knowledge refreshing policy**

6. Conclusions

This paper introduced and studied the important problem of knowledge refreshing, a critical process to ensure the quality and timeliness of knowledge discovered in a KDD process. Specifically, we studied knowledge refreshing from the perspective of when to
refresh knowledge so that the total system cost over a time horizon was minimized. We introduced knowledge loss, a key concept in knowledge refreshing, and demonstrated how to measure and estimate knowledge loss through experiments with real world data sets. We proposed a Markov decision process model for knowledge refreshing and developed the optimal knowledge refreshing policy and the adaptive heuristic method from the model. Through extensive simulation studies, we showed the robustness and the effectiveness of the proposed policy and method. The adaptive heuristic method can be easily implemented to address real world knowledge refreshing problems.

In this paper, knowledge loss was proposed to be estimated using a two-parameter Weibull function. We discussed the proper motivation of employing the Weibull function and demonstrated that the function fitted two real world data sets very well. It would be interesting to explore the estimating performance of the Weibull function using more real world data sets. It would be particularly interesting to search for theoretical explanations on why knowledge loss can be estimated using the Weibull function.

We demonstrated, through extensive simulation experiments, the robustness and the effectiveness of the optimal policy and the heuristic method. Continued study is required to evaluate the performance of the heuristic method in a real world setting. We mentioned in the paper that the performance of the heuristic is dependent on the accuracy of estimating future values of system parameters. It would be interesting to study how to design effective estimation methods for the heuristic method.

Although fixed-interval policies incur higher cost than the optimal policy and the heuristic method, they are easier to be implemented. We showed, through numerical studies, that the performance of the best-performed fixed-interval policy is close to that of
the optimal policy under some specific conditions. Therefore, another area merit worthy of exploration is to study analytically the boundary conditions under which the best-performed fixed-interval policy performs as good as the optimal policy.

References


Han, J. and M. Kamber. 2006. Data Mining: Concepts and Techniques (2nd ed.). Morgan Kaufmann.


Rastogi, R. 2003. Guest editor introduction: special section on online analysis and querying of continuous data streams. *IEEE Transactions on Knowledge and Data Engineering* 15(3) 513-514.


