Discovery of Actionable Patterns in Databases: The Action Hierarchy Approach

Gediminas Adomavicius
Computer Science Department
New York University
251 Mercer Street, New York, NY 10012
adomavic@cs.nyu.edu

Alexander Tuzhilin
Information Systems Department
Stern School of Business, NYU
44 West 4th Street, New York, NY 10012
atuzhili@stern.nyu.edu

Abstract

An approach to defining actionability as a measure of interestingness of patterns is proposed. This approach is based on the concept of an action hierarchy which is defined as a tree of actions with patterns and pattern templates (data mining queries) assigned to its nodes. A method for discovering actionable patterns is presented and various techniques for optimizing the discovery process are proposed.

Introduction

Discovering interesting patterns is one of the central problems in data mining. Several approaches have been proposed in the KDD literature to define the concept of interestingness. One of the main reasons why we want to discover patterns in business applications is that we may want to do something about them, such as acting on them to our advantage. Patterns that satisfy this criterion of interestingness are called *actionable* (Piatetsky-Shapiro & Matheus 1994; Silberschatz & Tuzhilin 1995; 1996).

As it was stated in (Silberschatz & Tuzhilin 1996), the difficulty of capturing actionability arises because defining any pattern to be actionable requires specification of actions for that pattern. This implies that we have to list all the possible actions for a given application and associate these actions with various patterns. This is problematic for the following reasons. First, there can be very many different actions for a given application, and it can be difficult to list all of them in advance. Second, even if we managed to list all the possible actions, we still have to assign these actions to various groups of patterns, and this can also be a very difficult task. Finally, actions and actionable patterns can change over time making it difficult to maintain actions and actionable patterns.

In this paper, we propose an approach to defining actionability as a measure of interestingness that is based on the concept of *action hierarchy.* Action hierarchy specifies the set of possible actions in an application in a hierarchical way through the action/subaction relationship. We maintain that this approach solves the difficulties explained above.

Action trees

Hierarchies of actions. To address the problems presented in introduction, we propose a hierarchical approach to action specification. In particular, we propose to maintain a hierarchy of actions from more general actions at the top of the hierarchy to more specific actions at the bottom. This is illustrated with the following example.

Example. Consider a customer purchase data for a supermarket application and the actions that the store manager can take based on this data. All the possible actions that a supermarket manager can take are grouped into the product stocking actions, promotion related actions, customer related actions, advertising actions, etc. These broad classes of actions can be further subdivided into more specific actions (*subactions*). For example, product stocking actions can be subdivided into determining what products to buy for the supermarket and how to arrange products in the store, and these subactions can be divided into even finer actions. This shows that the set of all the actions that a manager can take can be organized into a hierarchy.

An example of a fragment of such a hierarchy for the supermarket application is presented in Figure 1. To demonstrate the process of the top-down construction of an action hierarchy, consider the node *"Product stocking actions"* in Figure 1. As the figure shows, this action is divided into two subactions: *"Determining what and when to buy"* and *"Determining how to arrange products in the store"*. The former is subdivided further into subactions: *(Determining what and when to buy) "Based on selling statistics"*, *"Based on season"*, and *"Based on customer demographics"*. □

The user of a knowledge discovery system should be given an opportunity to describe actions in some *iterative* (incremental, step-by-step) fashion, because it is generally impossible to know all the possible actions in advance. From this point of view the hierarchi-

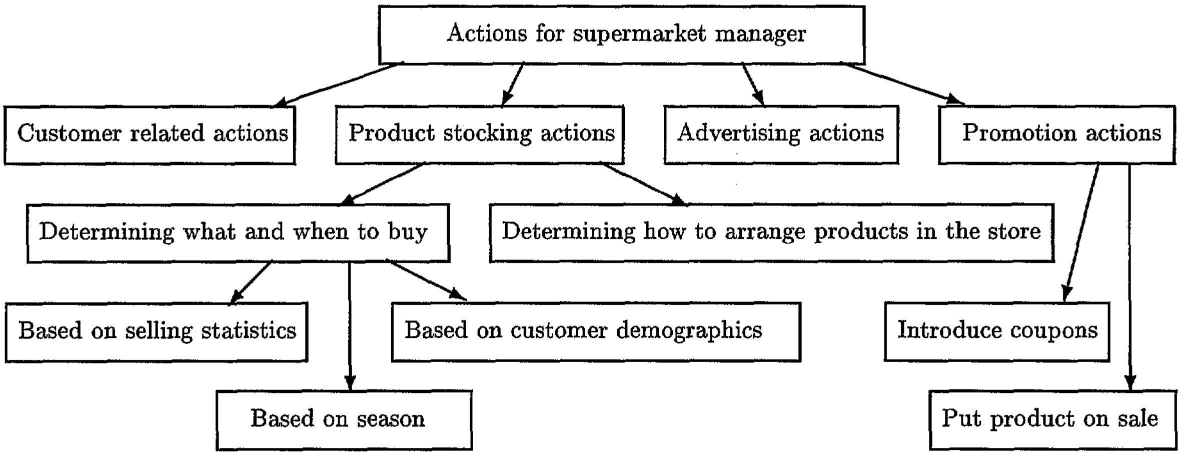
Actions for supermarket manager
Customer related actions
Product stocking actions
Advertising actions
Promotion actions
Determining what and when to buy
Determining how to arrange products in the store
Based on selling statistics
Based on customer demographics
Introduce coupons
Based on season
Put product on sale

Figure 1: Fragment of an action tree for the supermarket management.

cal approach is useful because it allows to build the set of actions gradually in *several stages*, and not at once. Moreover, such a top-down design makes actionable pattern discovery system modular. Therefore, it is easier to maintain a hierarchical set of actions when actions change over time. This hierarchical approach to building a set of actions has the same benefits as the hierarchical file organization in modern operating systems (as opposed to the flat structures of files of the legacy operating systems).

The action hierarchy presented in Figure 1 forms a tree. However, it can be a directed acyclic graph (DAG) in general because some of the actions may have common subactions. Although the DAG representation of action hierarchies is more general than the tree representation, nevertheless, we use the tree model because a tree is simpler than a DAG, easier to maintain, and has less complicated dependencies (that is, each node has at most one parent node in the hierarchy). Also, for a tree structure we can have a convenient implementation using the file system of any modern operating system (as described below). Also, trees are more visual than DAGs and provide a more intuitive description of the hierarchy. We also expect multiple-parent hierarchies, giving rise to a DAG structure, to happen infrequently in most of the applications[1]. Finally, if a node with the multiple parent nodes does occur, it can always be replaced by multiple nodes with one parent each. For all these reasons, we decided to use trees to represent action hierarchies.

Action attributes. We show now how to specify actionable patterns using action trees. This can be achieved in two ways. First, we may want to assign individual patterns to various nodes of the tree, thus declaring these patterns to be actionable (in terms of the action of the corresponding node). Patterns have to be specified in some pattern description language. For example, we can assign the following ***association rule*** (Agrawal, Imielinsky, & Swami 1993), specifying the extent to which families with small children buy sweets, to the action node in Fig. 1 ***"(Determining what and when to buy) Based on customer demographics"***:

$$\textit{ChildrenAgeLess6} \implies \textit{CategorySweets}\ (0.55, 0.01) \qquad (1)$$

This pattern is assigned to the aforementioned node in the action hierarchy of Figure 1, because the management can use it for the sweets buying decisions.

However, we may end up assigning very many patterns to specific nodes of the action tree in this way. Therefore, in the second approach, we may want to assign *classes* of patterns to the nodes of the action tree. In this paper, we specify these classes with *data mining queries* (Imielinski, Virmani, & Abdulghani 1996; Han *et al.* 1996; Shen *et al.* 1996) or *pattern templates* (Klemettinen *et al.* 1994). A data mining query (or pattern template) defines a set of patterns of a certain type. To "execute" a data mining query means to find all patterns that match this query. For example, consider the request: "Find all rules in customer purchase data specifying which product categories the customers with children of various ages are buying". This request can be expressed in the pattern description language proposed by (Klemettinen *et al.* 1994) as

$$\textit{ChildrenAge}\,* \implies \textit{Category}\ (0.5, 0.01) \qquad (2)$$

This request can also be expressed using other data mining query languages, such as M-SQL (Imielinski, Virmani, & Abdulghani 1996). More details can be found in (Adomavicius & Tuzhilin 1997).

Given an action tree, we assign one or several data mining queries to a node of the tree. To illustrate this approach, again consider the node *"Based on customer demographics"* from the action tree in Figure 1. We clearly could assign the pattern template (2) to this node because the patterns generated by this query would give insights to the store management about the decisions regarding product stocking.

[1] Here we can also draw an analogy with UNIX file systems: although UNIX files can belong to multiple directories, this happens quite infrequently in practice.

We can also combine these two approaches by assigning individual patterns and data mining queries to the same node of a tree. Note, that *any* pattern description language or *any* data mining query language can be used to specify patterns and data mining queries because the action tree concept does not depend on any specific language.

We call data mining queries and patterns assigned to a node of an action tree *attributes* of a node. We call action trees with attributes assigned to its nodes *attributed action trees.*

Note that, when trying to define the actionability measure of interestingness, one of the key decisions that we have made was not to specify the semantics of actions. We felt that the meaning of an action in the KDD context is very elusive, highly subjective, and varies significantly from one application to another. Therefore, we left the task of assignment of data mining queries to the nodes of an action tree to the user.

On implementation of action trees. One can draw a comparison between the organization of an attributed action hierarchy and an organization of a file system in a modern operating system. More specifically, actions can be compared to directories of the file system, and attributes can be compared to files containing attribute information (set of data mining queries and/or patterns). Thus, we implement an attributed action tree as a hierarchy of directories of the file system where each directory contains files in which data mining queries and patterns corresponding to the given action node are specified. Such implementation also allows to make changes to the action trees (add/remove actions, add/remove/modify attributes of actions) conveniently.

Alternatively, action trees can be implemented as class/subclass hierarchies using object oriented programming techniques.

Discovery of actionable patterns

We use action trees for the discovery of actionable patterns using the following steps.

Building an action tree. First, an action tree must be built (and maintained later on) for a given application. This can be done using techniques described in the previous section.

Assigning data mining queries. Second, data mining queries defining actionable patterns for the specific actions should be assigned to the corresponding nodes of the tree. For example, a possible data mining query assigned to the node *"Based on customer demographics"* of the tree in Figure 1 could be the query (2). Additional examples of data mining queries expressed in pattern template language similar to (Klemettinen *et al.* 1994) are:

- Query "Find what kinds of product categories sell well on different days of week" (assigned to the action *"Based on season"*):

$$\textit{DayOfWeek} \implies \textit{Category+}\ (0.4, 0.01) \qquad (3)$$

- Query "Find 'cross-selling' categories, that is, find categories of products that are selling together" (assigned to the action *"Determining how to arrange products in the store"*):

$$\textit{Category+} \implies \textit{Category+}\ (0.5, 0.01) \qquad (4)$$

Executing data mining queries. Given an attributed action tree, the pattern discovery process consists of the traversal of the whole action tree (say, using depth-first search) and execution of all the data mining queries. The discovered actionable patterns are written to the files associated with data mining queries.

Discovery optimization

The method for discovering actionable patterns described in the previous section does not give an answer to the question: *when* or *how often* to reexecute data mining queries that are assigned to the nodes of an action tree to obtain up-to-date patterns. The straightforward approach, which would be to reexecute all data mining queries whenever data changes in the database, is too computationally expensive in general. This is especially true for big applications with large action trees and many data mining queries. In the remainder of this section we present two optimization techniques and explain when they can be used in practice.

Partial tree traversal. The natural optimization of the action tree traversal technique is a *partial* traversal of an action tree. In this case, only the nodes of the tree selected by the user are traversed and only those data mining queries that are assigned to these nodes are executed. Nodes can be selected as individual nodes or as belonging to the user specified subtree.

The partial tree traversal approach can be used for applications in which there is no need to keep patterns up-to-date all the time. Therefore, data mining queries can be executed "on demand". That is, whenever there is a need to consider some specific action, only then data mining queries assigned to that action must be reexecuted to supply the user with the latest patterns to help make decisions.

Triggers. Consider an application where it is important to have up-to-date actionable patterns, such as stock market analysis applications. In such an application, we should rerun data mining queries only when "substantial" changes occur in the data that affect the patterns discovered by the queries. This would save computational resources by avoiding unnecessary executions of the queries not affected by data changes.

One way to detect such changes is to use the data monitoring method presented in (Tuzhilin & Silberschatz 1996), which uses *extended triggers* (sometimes also called *DMDT*[2] triggers). Such triggers are defined as follows. Let D be the data stored in the database and let ΔD be new data to be added to this database. In our supermarket application, D could be the supermarket customer purchase data for the last 6 months,

[2] Data-Monitoring and Discovery-Triggering

and ΔD could be the daily customer purchase data (scanner data) which is recorded daily in the central database. An extended trigger has the following form:

WHEN	new data ΔD becomes available
IF	"significant changes" in the data are found when ΔD is added to the old data D
THEN	execute the data mining query

We call these triggers "extended" because they are extensions of classical triggers used in active databases.

We will assign a trigger to each data mining query in the action tree. Specifications of the WHEN- and THEN- clauses are straightforward. The most interesting issue is the specification of the IF- clause (or the so called *monitor* (Tuzhilin & Silberschatz 1996)). A well-specified monitor could prevent data mining queries from being executed when newly added data does not alter existing patterns, thus saving computational resources. We consider two ways to specify monitors: manual and automatic.

We can let the user specify monitors manually, as it is done in the belief-driven discovery framework (Tuzhilin & Silberschatz 1996). Notice that in this case the important issues of correctness and computational efficiency of a monitor depend on the competency of the user. However, one of the main problems with manual specification of triggers is the scalability problem: there can be too many triggers (therefore, also monitors) in large applications. Therefore, another way to generate monitors is to do it automatically without any user involvement. We propose to implement the monitors in the following way, which allows them to be generated automatically.

Suppose, Q_i is a data mining query assigned to some node in the action tree. When new data becomes available, then the monitor selects a sample of some previously discovered actionable patterns by the query Q_i and measures[3] how much they have changed with the addition of new data. If changes are substantial, we have to rerun Q_i. The sample can be selected using any standard sampling techniques (Sudman 1976).

More details and examples of triggers for data mining queries can be found in (Adomavicius & Tuzhilin 1997).

Conclusions and future work

In this paper we proposed an approach to defining actionability as a measure of interestingness of patterns based on the concept of *action hierarchy.* Such hierarchies are defined as trees of actions with patterns and pattern templates assigned to the nodes of these trees and can be implemented in a simple and convenient way using file systems of modern operating systems. The hierarchical approach to defining actions and actionable patterns significantly simplifies the problem of actionable pattern discovery in databases for the reasons discussed in the paper. We also presented an approach to discovering actionable patterns using action trees and proposed two optimization techniques for actionable pattern discovery.

We are currently working on the following topics. First, we are developing methods and algorithms of maintaining attributed action trees, including determination of when and how to modify the structure of an action tree. Second, we are determining how our system can *simultaneously* support different forms of patterns and different data mining query languages. Finally, we are considering features that could be added to expand our action hierarchy model.

[3] Different measures are used for different types of patterns. For example, for association rules it may be confidence and support.

References

Adomavicius, G., and Tuzhilin, A. 1997. Discovery of actionable patterns in databases: The action hierarchy approach. Working Paper IS-97-8, Stern School of Business, New York University.

Agrawal, R.; Imielinsky, T.; and Swami, A. 1993. Mining association rules between sets of items in large databases. In *Proceedings of ACM SIGMOD Conference*, 207–216.

Han, J.; Fu, Y.; Wang, W.; Koperski, K.; and Zaiane, O. 1996. DMQL: A data mining query language for relational databases. In *Proceedings of the SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery.*

Imielinski, T.; Virmani, A.; and Abdulghani, A. 1996. DataMine: Application Programming Interface and Query Language for Database Mining. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining.*

Klemettinen, M.; Mannila, H.; Ronkainen, P.; Toivonen, H.; and Verkamo, A. I. 1994. Finding interesting rules from large sets of discovered association rules. In *Proceedings of the Third International Conference on Information and Knowledge Management.*

Piatetsky-Shapiro, G., and Matheus, C. J. 1994. The interestingness of deviations. In *Proceedings of the AAAI-94 Workshop on Knowledge Discovery in Databases.*

Shen, W.-M.; Ong, K.-L.; Mitbander, B.; and Zaniolo, C. 1996. Metaqueries for data mining. In *Advances in Knowledge Discovery and Data Mining.* AAAI Press. chapter 15.

Silberschatz, A., and Tuzhilin, A. 1995. On subjective measures of interestingness in knowledge discovery. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining.*

Silberschatz, A., and Tuzhilin, A. 1996. What makes patterns interesting in knowledge discovery systems. *IEEE Transactions on Knowledge and Data Engineering* 8(6).

Sudman, S. 1976. *Applied Sampling.* San Francisco: Academic Press.

Tuzhilin, A., and Silberschatz, A. 1996. A belief-driven discovery framework based on data monitoring and triggering. Working Paper IS-96-26, Stern School of Business, New York University.