Learning Data Trend Regularities From Databases in a Dynamic Environment *

Xiaohua Hu, Nick Cercone, Jinshi Xie
Department of Computer Science, University of Regina
Regina, SK, Canada, S4S 0A2
e-mail: {xiaohua, nick} @cs.uregina.ca

Abstract

Knowledge discovery in databases has attracted a lot of attention from the AI and databases community because of the huge information stored in the databases. There are a lot of algorithms developed to find rules from databases directly, but all these algorithms assume that the data and the data scheme are stable and most of the algorithm focus on discovering the regularities about the current data in the databases. In this paper we present a method which can learn rules from the current data in the database to predict the data trend in the future. Our method combines the techniques of attribute-oriented induction, object-oriented databases and transition network. In our model, both the database contents and the database structure may evolve over the lifetime of a database.

Keywords: Machine Learning, Discovery in Databases, Use of Domain Knowledge

*The authors are members of the Institute for Robotics and Intelligent Systems (IRIS) and wish to acknowledge the support of the Networks of Centres of Excellence Program of the Government of Canada, the Natural Sciences and Engineering Research Council, and the participation of PRECARN Associates Inc.

1 Introduction

Knowledge discovery in databases has attracted a lot of attention from the AI and databases community because of the huge information stored in the databases. There are a lot of algorithms developed to find rules from databases directly [3], but all these algorithms assume that the data and the data scheme are stable and most of the algorithms focus on discovering the regularities about the current data in the databases. The reality is that the contents of databases and database scheme may change over time and users are often interested in finding the general trends of data evolution in the future. So it is important to discover data evolution regularities in a dynamic evolving database. Because of the large volume of data, data evolution regularity can not be simply expressed by enumeration of the actual data. Machine learning technology should be adopted to extract such regularities in databases.

In this paper we propose a new method for discovering rule from the current data in the databases to predict the data trend in the future. Our method combines the techniques of attribute-oriented induction [2], object-oriented databases and transition network. In our model, both the database contents and database structure (schemes) may evolve over the lifetime of a database.

This paper is organized as follow: In section 2 we discuss the primitives for learning data trend regularities. The principle and algorithms for learning in dynamic environment is presented in Section 3. The study is summarized in Section 4.

2 Primitives for Learning Data Trend Regularities

The real world is a dynamic and evolving place, to represent and simulate it efficiently and effectively, we use the object-oriented data model because object oriented representations are very convenient to describe real world. Our method involves elements of the following areas: attribute-oriented induction, object-oriented database model and transition network. In this section, previous work from each of these areas is reviewed.

2.1 Attribute-Oriented Induction

The attribute-oriented induction was first proposed in [2]. The key to the approach is an attribute-oriented concept tree ascension technique for generalization which was implemented using well-developed set-oriented database operations, substantially reducing the computational complexity of the database learning task. An attribute is generalizable if there are a large number of distinct values in the relation but there exists a concept hierarchy for the attribute (i.e., there are higher level concepts which subsume these attribute values). Otherwise, it is nongeneralizable.

The general idea of basic attribute-oriented induction is one in which generalization is performed attribute by attribute using attribute removal and concept tree ascension. If an attribute is nongeneralizable, then it should be removed in the generalization. Attribute removal corresponds to the generalization rule, *dropping conditions* [5]. Consider a tuple as a set of conjuncts in the logical forms; an attribute value together with its attribute name form one of the conjuncts. By removing a conjunct, a constraint is eliminated and the concept is generalized. If there are a large set of distinct values for an attribute, the large set of values must be generalized. However, if there is no higher level concept provided for the attribute , it can not be further generalized by ascending the concept tree. Therefore, the attribute should be eliminated in generalization. Attribute removal can also be viewed as a generalization of the attribute to the most general concept ANY and then removed from the representation.

If an attribute is generalizable, then it should be generalized to a higher level concept value by concept tree ascension techniques. Concept tree ascension corresponds to the generalization rule, *climbing generalization trees* [5]. If there exists a higher level concept for the value in the concept tree, then the substitution of the value in each tuple in the relation by the corresponding higher level concept makes the tuple cover more cases than the original one, and thus it generalizes the tuple.

As a result, different tuples may be generalized to identical ones, and the number of distinct tuples in the generalized relation is reduced. In the procedure of generalization, the tuples in database are generalized to the desirable level, the table gained at this stage is called generalized relation. The generalized relation contains all the essential information of the original data in the database.

2.2 Object-Oriented Model

Object-oriented data models and systems embody rich data structures and semantics in the construction of complex databases, such as complex data objects, class/ subclass hierarchy, property inheritance, methods and active data etc.

An OODB organize a large set of complex data objects into classes which are in turn organized into class/subclass hierarchies with rich data semantics. Each object in a class is associated with (1) an object identifier, (2) a set of attributes which may contain sophisticated data structures, set or list-valued data, class composition hierarchies, multimedia data etc., and (3) a set of methods which specify the computational routines or rules associated with the object class.

Objects & Object identifies: Objects can serve to group data that pertain to one-world entity. For example, we can treat a document as an object which group chapters, indexes, etc. into one entity, namely, a document. Chapters serve as attributes of the document object. In like manner, a chapter can be defined as another type of object which groups sections into one entity. The uniform treatment of any real-world entity as an object simplifies the user's view of the real world. This implies that the state of an object consists of values for the attributes of the objects, possibly with their own.

Objects can have a unique identity independent of the values that they contain. A system that is identity-based allows an object to be referenced via a unique internally generated number, an object identifier, independent of the value of its primary key, if any. The adoption of object identifier facilitates the representation of the state of an object, namely, the state of an object is naturally represented as a set of identifiers of the objects which are the values of the attributes of the object

Complex Attributes & Methods: In an object-oriented database, object attributes may be complex attributes: *references, collection* and *procedures*. Reference attributes are analogous to pointers in programming language, or to foreign keys in a relation system. Collection is used for lists, or arrays of values. The collection may include simple attribute values and also references. Derived attributes are those whose values can be defined procedurally rather than stored explicitly, by specifying a procedure to be executed when the value is retrieved or assigned. For example, we may store such personal information as birth date and age in a personal databases. The birth date will not change but the age will. It would be desirable to define a procedure for the age attribute so that it always represents the difference between the current date and the birth date. Derived date correspond roughly to views in the relational database, but procedure languages may define more complex derivations than views, and are generally used to define individual attributes rather than relation. Since knowledge discovery in database is a read-only operation to the database and does not change the state of the database in any way, derived attribute values, once retrieved from the database can be treated just like regular attribute values. Derived attributes do not pose any special problem to the knowledge discovery process.

Method is another important component of OODBs. Many behavior data of objects can be derived by application of methods. Since a method is usually defined by a computational procedure/function or by a set of deduction rules, it is difficult to perform generalization on the method itself unless the generalization of the method is clearly understood by application programmers and is coded as a new method which directly performs the required generalization. In general, the generalization on the data derived by method application should be performed in two steps : (1) deriving the task-relevant set of data by application of the method and, possibly, also data retrieval; and (2) performing generalization by treating the derived data as the existing ones.

Class: Class is used to group together object that respond to the same message, use the same methods, and have variable of the same name and type. Each such object is called an instance of its class. A class defines an object type or intent-the structure of a particular type. The intent includes structure (that is, the attributes and relationships in which objects having this type can participate) and behavior (that is , the methods associated with the type).

2.3 Transition network

Object-oriented model is supposed to model the real world more closely than traditional relational data model because each real world entity can be modeled by an object in the computer. Unfortunately, while the real world is a dynamic and evolving place, most object-oriented data models are essential static: although objects can be dynamically created and destroyed, class can not. They either do not support, or do not conveniently support, structural and behavioral changes to instantiated objects. Databases, including object-oriented databases, often allow historical accesses, but only for objects whose structure does not change. None of these systems provides the mechanism necessary to describe data relation across transition in a dynamic system [1]. Further, classes can not be dynamically modified as further attributes and behaviors of the objects are discovered or needed. One of the key issues when combining a dynamic object-oriented data model with time and histories to produce a dynamic system is how the relationships between the instance variables in different state are defined. In our method, we use transition constraints to model the relationship between the instances variables in different state.

We say that an object which is an instance of one class (called the source class) undergoes a transition when it becomes an instance of another class (called target class). We distinguish two types of transition **evolution** and **extension** [4], based on whether or not the object undergoing the transition is preserved as an instance of the source class or not. In other words, an evolution occurs when the transition object ceases to be an instance of the source class. For example, when an object representing an applicant changes to reflect the acceptance of the applicant, it undergoes an evolution; that is, it ceases to be an instance of the applicant subclass and becomes an instance of the student subclass. An extension is a transition with the negative of the additional condition associated with evolution. In other words, an extension occurs when the object remains an instance of the source class with the negation of the additional condition associated with evolution. For example, when an alumnus with a Master's degree applies to the Ph.D program, the transition of the object representing the alumnus into an instance subclass is an extension.

Note that some of the transition events are triggered solely by time whereas others are triggered by other events in the dynamic system. In this paper we assume only evolution occurs in our dynamic environment model.

2.4 An Example Database

Consider a simple version of the social security database in some social benefit office in Canada as shown in Table 1,2,3. Figure 1 is the concept hierarchies for attributes **age, salary** and **pension.** Figure 2 is the corresponding class hierarchy and transition network. Citizen may start as a child. When children reach the age of 18, they become the instance of Adult class. Later, at age 65, they retire (senior citizen) and eventually die. The transition from senior citizen to death is weak

oid	name	sex	birthday	age	employer	salary	dependents
a1	Sam	M	Dec. 5, 1954	method(birthday, today)	NCR	70k	{c1,c2,c3}
a2	Janet	F	Aug. 4, 1965	method(birthday, today)	BNR	53k	{c4, c5}
a3	Mary	F	June 23, 1945	method(birthday, today)	NT	60k	{c6}
a4	Tom	M	July 17, 1963	method(birthday, today)	Gov.	36k	0
..	...	..	....	....	..	..	..
al	Jay	M	Oct. 24, 1970	method(birthday, today)	MPE	40k	{ci}
am	Mark	M	Jan. 29, 1940	method(birthday, today)	NGE	100k	{cj,ck}

Table 1: Instance of Class **Adult**

oid	name	sex	birthday	age	parent/guardian	school
c1	Jane	F	Oct. 5, 1984	method(birthday, today)	a1	No. 1
c2	Janet	F	June. 4, 1986	method(birthday, today)	a1	No.1
c3	Mary	F	June 23, 1985	method(birthday, today)	a1	No. 2
c4	Peter	M	July 17, 1979	method(birthday, today)	a2	Bran
..	...	..	....	....	..	..
cx	John	M	Feb 24, 1980	method(birthday, today)	az	MMM
cz	Frank	M	Jan. 29, 1982	method(birthday, today)	az	PCC

Table 2: Instance of Class **Child**

because some people may live older than 85 while some other may not. We use $\not\longrightarrow$ to represent weak transition. For brevity, we still list database objects in tables similar to how we list tuples in a relational table; the differences are, in addition to showing all the attribute-value pairs, we also show the object identities (oids) and the values for reference attributes are oids. From the knowledge discovery point of view, it is unnecessary to distinguish which data are stored within the class and which are inherited from its superclass. As long as the set of relevant data are collected by query processing , the KDD will treat the inherited data in the same way as the data stored in the object class and perform generalization accordingly.

{0–4} : kids; {4–14}: children; {14–20} : young
{20-29}: twenties; {30–39}: thirties; {40–49}: forties
{50–64}: late_mid; {65-}: old
{kids, children}: child_age; {young, twenties}: young_age
{thirties, forties, late_mid}: mid_age; { old}: old_age
{child_age, youth_age, mid_age, old_age}: Any(age)
{0–20k}: low_income; {20K–34k}: low_middle_income; {35k-45k}: mid_income
{46–65k}: high_income; {66k–}:very_high_income;
{low_income, low_mid_income, mid_income, high_income, very_high_income}: Any(income)

Figure 1: The concept hierarchy for age, salary, pension

oid	name	sex	birthday	age	pension
s1	Woope	F	Oct. 5, 1925	method(birthday, today)	17k
s2	Jason	M	July 14, 1929	method(birthday, today)	23k
s3	Rose	F	Jan. 28, 1913	method(birthday, today)	60k
..	...	..	....	...	..
sl	Codoba	M	Aug.,24, 1910	method(birthday, today)	40k
sk	Clark	M	Feb. 23, 1914	method(birthday, today)	10k

Table 3: Instance of Class **Senior citizen**

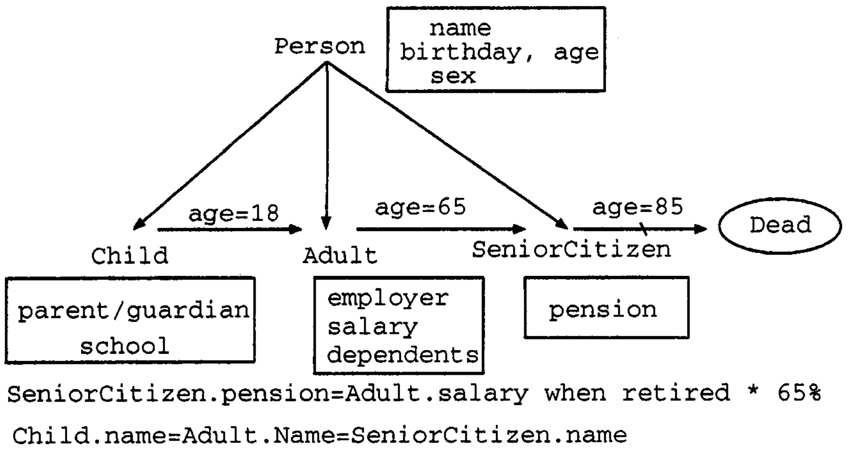

Figure 2: The class hierarchy and transition network for people

3 Principle and Algorithms for Learning in Dynamic Environment

3.1 General Discussion

The general trend of evolution describes how a particular set of data evolves over a period of time, for example, how the population changed over the past 10 years, or what is the trend 5 years later. To discover data evolution regularities in the future, the evolving data should be identified first and be extracted from the database.

In a relational database, we start learning by collecting relevant data into a relational table using selection, projection, or join operations provided by the query language; namely, we make a copy of the data portion in the database that are relevant to our learning task and the database remains intact. We should abide by the same principle in an OODB. However, in an OODB, instances are grouped into classes and related objects of different classes are connected through references, i.e., oids, which acts as a counterpart of join in the relational sense. Attribute projection may not be supported in an OODB in a dynamic environment. Moreover since in a dynamic

oid	name	sex	pension
s1	Woope	F	17k
s2	Jason	M	23k
s3	Rose	F	60k
..	...	..	...
sl	Codoba	M	40k
sk	Clark	M	10k

Table 4: Instance of Class **Senior citizen**

environment the data contents and scheme may change over the time, the data extraction procedure is more complicated. For example, if the city administrator wants to know the general situation about the senior citizen 5 years later, the query may be submitted as below:

learn data evolution trend for *seniorcitizen S*
5 years **later**
in relevant to *S.name, S.sex, S.pension*

The data extraction procedure is performed in two steps (1) based on the query, extract the target class objects; (2) examine the class hierarchy and transition network to check whether there are any source class objects which can transform to the current learning class as time goes by. For the above query, the first step is to extract all the citizens from the current senior citizen class except those who are 80 years old (because we assume that senior citizen die at 85). Then we examine the class hierarchy and transition network and find an Adult becomes a senior citizen when he reaches 65. Hence we have to look through the Adult object class and extract those adults who are older than 60 and derive the corresponding attributes values, e.g. replace salary by pension. (we can assume that adult salary increases 4% each year, first compute the adult salary when he retires , and then apply the method: seniorcitizen.pension=adult salary when retired * 65 %). As a result, we get a set of task-relevant instances objects as shown in Table 4.

The "relevant data " returned from a query into an OODB is generally a set of oids pointing to instances of a specific class-the queried class. After obtaining object instance relevant to our learning task, we may start the induction process. The basic strategies of attribute-oriented inductions [2] are still applicable when the induction is conducted on an attribute whose values are of a primitive class. For example, in the generalization process, the attribute "S.name" is removed (even if it were in relevance to the learning query) since it has no superordinate concept. Since attribute projection may not be supported in an OODB, we encounter some technical differences here from in a relational databases. Relevant as well as irrelevant attributes of objects of this class can be accessed by dereferencing the oids in this set; attributes of related objects of other classes are accessed through reference attributes.

One of the essential components of an OODB is object identifier (oid) which uniquely identifies objects. It remains unchanged over structural reorganization of data. Since objects in an

OODB are organized into classes which in turn belong to certain class and subclass hierarchy, the generalization of objects may refer to their corresponding class and subclass hierarchy. In the object-oriented database model, no two object instances are equal even if they have the identical values for each of their attributes, because their oids will never be the same. The relational attribute-oriented induction method is value-based. In an object-oriented database, when we judge whether two object instance are "mergeable" in the relation sense, we only compare values in the relevant attributes, ignoring the oids. When checking whether two reference attributes have equal values, we see if the composing attributes of the pointed object instances are equal instead of comparing the oid values of the two reference attributes. Irrelevant attribute values are not compared. If two instances have identical values for every task-relevant attributes, one of them is considered redundant even though their oids are different. To "merge" two instances, we simply delete the oid of one of them from the class extent and record the number of redundant object in vote.

The evolving data may have two kinds of attributes: stable attributes and evolving attributes. The stable attributes, in which the data values do not change over time, can be generalized by attribute-oriented induction in the same way as those discussed in [2]. The evolving attributes, in which the data values change over time, can be generalized according to a generalized time slots when appropriate. For example, adult's salary keeps changing yearly and so we need to update the salary based on the time value. Once we get the value for the salary, then we can still apply attribute-oriented induction.

As to generalization of collection attributes and reference attributes, these two categories of complex attributes maybe mixed. A collection attribute can have elements of pointers to another class; a reference attribute can point to class consisting of collection attributes. Generalization of a collection attribute should be performed on its composing elements and the generalized concept for the collection is expressed in terms of its generalized elements. Generalization on a reference attribute is conducted on each composing attribute of the object class being pointed to. Since the objects being pointed to may be of a class that contains reference attribute as well, we may have to apply the strategy of collection attribute several times. By the least commitment principle [2], generalization should start on the finest concepts in order to guarantee correctness. This means we may have to keep on dereferencing pointers until we reach a non-reference attribute. The class composition hierarchy contains the information that will guide this pointer dereference. When there exists a link from an attribute in one class to another class, and when this attribute is relevant to the learning task, this link (pointer) should be dereferenced, that is, generalization moves on to the referenced class. No more dereferencing is needed when we have come down to a leaf class in a class composition hierarchy.

As a result of generalization, different objects may be generalized to equivalent ones where two (generalized) objects are equivalent if they have the same corresponding attribute values without considering their object identifiers and a special internal attribute count, which register

oid	sex	pension	count
g1	F	low_income	30000
g2	M	low_income	25000
g3	F	low_mid_income	10000
g4	M	low_mid_income	15000
g5	F	mid_income	5000
g6	M	mid_income	10000
g7	M	high_income	3000
g8	M	very_high_income	2000
total			100000

Table 5: Instance of Class **Senior citizen**

the number of objects in the initial working class. Notice that a generalized object, though has its own new oid (object identifier) in an OODB, is a carrier of the general properties of a set of initial objects because the original identifier has been generalized into a class or superclass name. The count accumulated in the generalized class incorporates quantitative information in the learning process. After the generalization process, we get a generalized table as shown in Table 5.

Based on this generalized table, we can derive some useful data trend information: for example, *5 years later, there are about 100,000 senior citizens, 55% are male, and 45 % are female. Among the female senor citizens, 66.7 % have lower income and so on*

3.2 Algorithm

In summary, we present the algorithm below. After task-relevant instances have been selected, the procedure **OOinduction** is called. This procedure processes each attribute such that the attribute threshold is satisfied. It then goes on to ensure that the final learned result contains no more instances than the threshold. For each relevant attribute can be simple as of a primitive class, or complex as of a set, a list, an array, or a reference, **OOinduction** has to treat each attribute according to its type; this is done by the subprocedure **Resolve(A)**. The element in a collection attributes can again be a complex class, and the attributes of the class pointed to by a reference attribute may also be of a complex class. This gives **Resolve(A)** a recursive nature. Since complex attributes are defined using primitive classes and can always break down to primitive classes, **Resolve(A)** is guaranteed to terminate. Moreover, since all the extension we made strictly follow the *least commitment principle* [2], the correctness of our algorithm is also guaranteed. (In the algorithm "*A" refers to the object class pointed to by 'A'.)

Algorithm 1 An Data Trend Discovery Algorithm

Input: (1) an object-oriented databases (2) a set of class hierarchies and transition network, (3) a set of concept hierarchies H_i for each attributes, where H_i is a hierarchy on the attribute A_i, if available; (4) the threshold value T and
Output. A set of data trend regularities

Method

Step 1. *extract relevant data from object oriented databases into data set S*

(1) extract data according to the query

(2) extract data according the class hierarchy and transition network

Step 2. *Call* **Procedure OOinduction(S,T)**

Step 3. *Transform the final relation into a feature table, and extract the relevant rules*

```
Procedure OOinduction(S,T);
   /* S is the set of instances of some classes relevant to the learning task*/
Begin
   For (each task-relevant attribute A of the class) DO
         WHILE (the number of distinct values of A > T) DO
            IF there does not exist a concept tree for A
            THEN Mark A as "irrelevant" and exit
            ELSE call Resolve (A)
   /* now the threshold constraint is satisfied by each attribute */
WHILE | S | > T DO
   BEGIN
         select the attribute containing substantially more distinct values or with
         a better reduction ratio, and replace each value of them by its corresponding
         parent in the concept tree, and record the number of redundant instances in vote;
   END
   /* now the threshold constraint is satisfied by the instance set */
END

Resolve(A);
   /* A is an attribute name on which generalization is done in this procedure */
   BEGIN
         CASE  A of
         Reference: If (*A has not been dereferenced)
            THEN FOR (each attribute of the class) DO Resolve(attribute)
            ELSE FOR (each non-reference attribute) DO Resolve(attribute);
         Set: For (each element of A) DO Resolve(element);
            Delete repetitive elements from A;
         List: FOR (each element of A) DO Resolve (element)
         Primitive: For (each instance in set S) DO
            Replace the value of attribute A by its parent concept in the concept tree;
            Delete repetitive instance in set S;
```

END

4 Conclusion

In this paper we propose a new method to predict the data trend regularities in the future. Our method combines the techniques of attribute-oriented induction, object-oriented model and transition network. Our method has some practical value because in real application because it is very necessary and important to provide the data evolution regularities to people for for making decisions.

References

[1] Michel Augeraud, Freeman-Benson, (1991). Dynamic Objects, *COCS'91 Conference on Organizational Computing System*, Atlanta, Georgia

[2] Y. Cai, N. Cercone and J. Han, (1991). Attribute_Oriented Induction in Relational databases, *Knowledge Discovery in Database, AAAI/MIT Press*, G.Piatetsky-Shapiro and W.J. Frawley (eds) pp. 213-228

[3] W. J. Frawley, G. Piatetsky (1991). *Knowledge Discovery in Database, AAAI/MIT Press,*

[4] G. Hall and R. Gupta, (1990). Modeling Transition, *7th international Data Engineering Conference*

[5] R.S. Mickalski, (1983). A Theory and Methodology of Inductive Learning, *in Machine Learning: An Artificial Intelligence Approach, Vol. 1.* Michalski et. al. (eds), Morgan Kaufmann, pp 41-82.

[6] S. Nishio, H. Kawano and J. Han, (1993). Knowledge Discovery in Object-oriented Databases: The First Step, *Knowledge Discovery in Databases Workshop* 299-313

[7] J.S. Xie, (1993). Attribute-Oriented Induction in Object-Oriented Databases, MS thesis, School of Computer Science, Simon Fraser University, Canada