DATA MINING IN TEMPORAL DATABASES

George Koundourakis, Mohammad Saraee and Babis Theodoulidis
Information Management Group
Department of Computation, UMIST,
Manchester, United Kingdom
Email: (koundour,saraee,babis)@co.umist.ac.uk HTTP://timelab.co.umist.ac.uk

ABSTRACT

In this paper we describe our approach to data mining in temporal databases by introducing Easy Miner, a data mining system developed at UMIST. This system implements a wide spectrum of data mining functions, including generalisation, relevance analysis, classification and discovery of association rules. By integrating these interesting data mining techniques, the system provides a user friendly and interactive environment with good performance and of course, wide choice of functionalities. These algorithms have been tested on time-oriented medical data and experimental results show that the algorithms are efficient and effective for discovery of previously unknown knowledge in databases.

1. Introduction

Data mining is the nontrivial extraction of implicit, previously unknown, and potentially useful information from data [1]. In other words, it is the search for relationships and global patterns that exist in large databases, but are "hidden" among the vast amount of data, such as a relationship between patient data and medical diagnosis. These relationships represent valuable knowledge about the database and objects in it and (if the database is a faithful mirror) of the real world registered by the database.

The growth in the size and number of existing databases far exceeds human abilities to analyse such data with simple query and analysis approaches. Thus it created a need and an opportunity for extracting knowledge from databases. Recently, data mining has been ranked as one of the most promising research topics for the 1990s by both database and machine learning researchers.

There are several kinds of knowledge that data mining aims to extract from a database. We have been focused on the areas of relevance analysis, generalisation, classification, and association. We tried to integrate these areas and use the advantages of one in order to benefit and the others. In the following sections we describe our approaches to these areas, which also have been implemented and tested in a data mining tool called Easy Miner.

2. Temporal Constraints for Intervals.

Our first task to developing the time-oriented pattern discovery process is to move the data from the temporal representation to an equivalent static one that can be managed by the existing algorithms. Valid time intervals are associated with each tuple in time-oriented databases to record the time for which these tuples are valid within the modelled domain. Because these intervals represent important semantics about the application, it may be desirable to include them within the data used to produce generalisation, classification or associations rules. We introduce two concepts for integrating temporal semantic into the process: namely the Moving Window clause and the Time Slice clause.

The TIME SLICE Clause: The User specifies a time interval and only those objects in the database whose valid time is contained in this interval get selected for discovery process. Example: Include all patients who have had stroke between 1/10/1995 to 1/10/1997.

The MOVING WINDOW Clause: The user specifies an interval width and the objects in the databases are examined through a temporal window of this width moving along the temporal axis. Example: Include all patients whose Blood Pressure has raised more than 2 times within certain time (1 month, 2 months, 3 months, 6 months, 2 years).

3. Generalisation

Easy Miner uses generalisation rules as they are defined in [9]. By using these rules, values of the attributes are generalised at multiple concept levels. As a result of this process, a set of attributes is generated for every generalised attribute of the original data set. The generated attributes are instances of the same attribute but in different concept levels. In order to determine which of these attributes (concept levels) is most appropriate to the required data-mining task, relevance analysis is applied on them and only the one that is most relevant to the specific data-mining task is kept. In Easy Miner there are several procedures for the automatic generation of generalisation rules for numeric attributes. Date-time attributes can be easily generalised into years, seasons, seasons of a year, months, months of a year and days. Finally, considering relevance with another attribute can lead to automatic generation of generalisation rules for any attribute.

4. Relevance Analysis

We perform relevance analysis as defined by Kemper et al.[8]. Examples of obvious relevancies amongst database attributes is ‘Age’ to ‘Date of birth’ and ‘Title’ to ‘Sex’ and ‘Marital Status’. This kind of knowledge is qualitative and it is quite useful to mine from large databases that hold information about many objects (fields). For example a bank could look in its data and identify the factors that it should consider in order to give a credit limit to a customer. Applying Easy Miner to our credit history database revealed that Credit Limit is relevant to *Account Status*, *Monthly Expenses*, *Marital Status*, *Monthly Income* and *Gender*.

A number of statistical and machine learning techniques for relevance analysis have been proposed until now. We use an information-theoretic asymmetric measure of relevance introduced by Loether and McTavish [10] known as the *uncertainty coefficient*. Assume having a data set P of p data samples. Select an attribute of it as the classifying attribute and assume that it has m distinct values (classes). Let us suppose also that P contains p_i records for each class P_I (i=1...m). Then a random selected record belongs to class P_i with probability p_i/p. The expected information needed to classify a given sample is given by: $I(p_1, p_2,.., p_m) = -\sum_{i=1}^{m} \frac{p_i}{p} \log_2 \frac{p_i}{p}$

An attribute A with values $\{a_1, a_2, \ldots, a_k\}$ can be used to partition P into $\{C_1, C_2, \ldots, C_k\}$, where C_j contains those records that have value a_j of A. Let C_j contain p_{ij} records of class P_i. The expected information based on the partitioning using as split attribute the A, is: $E(A) = \sum_{j=1}^{k} \frac{p_{ij} + \ldots + p_{mj}}{p} I(p_{ij} + \ldots + p_{mj})$

So, the information gained by partitioning on attribute A is: $gain(A) = I(p_1, p_2, \ldots, p_m) - E(A)$

The uncertainty coefficient U(A) for attribute A is obtained by normalising the information gain of A so that U(A) ranges from 0 to 1. The value 0 means that there is significant independence between the A and the target attribute, and 1 means that there is strong relevance between the two attributes. The normalisation of U(A) is achieved by the following equation:

$$U(A) = \frac{I(p_1, p_2, \ldots, p_m) - E(A)}{I(p_1, p_2, \ldots, p_m)}$$

It is upon the user to keep the *n* most relevant attributes, or all the attributes that have value of uncertainty coefficient greater than a pre-specified minimum threshold.

5. Classification

Classification is an important area of research in data mining. Classification partitions massive quantities of data into sets of common characteristics and properties [5], [6]. In classification a set of records, acting as *training set*, is analysed in order to produce a model of the given data. Each record is assumed to belong to a predefined class, as determined by one of the attributes, called *classifying attribute*. Table 1 shows a part from a sample training set of the medical database, where each record represents a patient and *Lived* is the *classifying attribute* of the training set.

Once derived, the classification model can be used to categorise future data samples, as well as providing a better understanding of the database contents. Classification is particularly useful when a database contains examples that can be used as the basis for future decision making, e.g. for assessing credit risks, for medical diagnosis, or for scientific data analysis.

…..	Sex	Date of Birth	Date of Stroke	Lived	…..
...	Male	9/21/1924	9/23/94	Died	...
...	Female	5/8/1921	12/10/94	Died	...
...	Male	1/18/34	2/7/94	Survived	...
...	Male	9/26/1925	11/13/94	Survived	...

...	Female	3/28/51	12/9/94	Died	...
...	Male	1/19/34	2/7/94	Survived	...

Table 1: Sample Training Set

The classification technique that we have developed in Easy Miner is based on the decision tree structure. By using a decision tree, untagged data sample can be classified by testing the attribute values of the sample data against the decision tree. A path is produced from the root to a leaf node, which has the class identification of the sample.

5.1 Decision Tree Classifier

A decision tree is a class discriminator that recursively partitions the training set until each partition consists entirely or dominantly of examples from one class. Each internal node of the tree contains a *split point*, which is a test on one attribute and determines how the data is partitioned. Figure 1 shows a sample decision-tree classifier based on the training set shown in Table 1. This decision tree can be used in order to discriminate future patients that had stroke into *Survived* or *Died* categories.

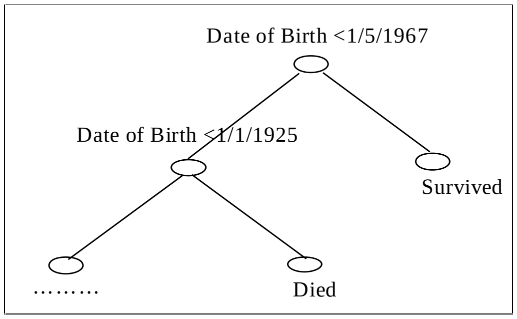

Figure 1: Decision Tree

5.2 Collection of Training Set and Construction of Decision Tree

In order to construct a decision tree classifier, the first step is to retrieve the classification task relevant data and store them in a relation (table or query). This is an ease task that executing a usual query can perform it. The second step is to perform generalisation, as it has been described previously. The third step is to examine the generated genaralised training set with the purpose of "purifying" it and making it suitable for classification process. Including irrelevant attributes in the training set would slow down and possibly confuse the classification process. Therefore, *relevance analysis* is performed to the data set and then only the most relevant attributes are kept. The algorithm that we use for tree building is, in general, the following:

```
MakeTree(Training Data T)
if (all points in S are in the same class) then
    return
EvaluateSplits( )
if (T can not be further partition) then
    Evaluate Each Class Probability(S)
    return
```

```
        Use best split found to partition S into S1 and S2
        Delete(S)
        MakeTree(S1)
        MakeTree(S2)
```

5.3 Data Structures

For each attribute of the training data a separate list is created. An entry of the attribute list has the following fields: (i) attribute value, (ii) class label, (iii) index of the record (id) from which these value has been taken. Initial lists for numeric attributes are sorted when first created. In the beginning, all the entries of the attribute lists are associated with the root of the tree. As the tree grows and nodes are split to create new ones, the attribute lists of each node are partitioned and associated with the children. When this happens, the order of the records in the list is preserved, so resorting is unnecessary and not required.

Histograms are other data structures that are being used. These histograms are capturing the class distribution of the attribute records at a given node. For numeric attributes, two histograms are associated with each decision-tree node that is under consideration for splitting C_{above} and C_{below}. C_{below} contains the distribution of attributes that have been already processed, and C_{above} contains it for those that are not. For categorical attributes a histogram (*count matrix*) is associated with every node. So, for a numeric attribute the histogram is a list of pairs of the form <class, frequency> and for a categorical attribute, the histogram is a list of triples of the form <attribute value, class, frequency>.

5.4 Finding Split Points

While growing the tree, the goal at each node is to determine the split condition that best divides the training records belonging to the node under examination. Several splitting indexes have been proposed in [3], [5] and [6] to evaluate the goodness of the split. Two of the most popular and acceptable indexes are the *gini index* and the *entropy index*. Having a data set T which contains examples from n classes and p_j is the frequency of class j in data set T then:

$$gini(T) = 1 - \sum_{j=1}^{n} p_j^2$$ and the entropy index *Ent(T)* is: $$Ent(T) = -\sum_{j=1}^{n} p_j \log_2(p_j)$$

If a split divides S into subsets S_1 and S_2, then the values of the indexes of the divided data are given by:

$$gini_{split}(S) = \frac{n_1}{n} gini(S_1) + \frac{n_2}{n} gini(S_2)$$ and $$Ent_{split}(S) = \frac{n_1}{n} Ent(S_1) + \frac{n_2}{n} Ent(S_2)$$

To find the best split point for a node, we scan every attribute's lists and evaluate splits based on that attribute. The attribute containing the split point with the lowest value for the split index is then used to split the node. There are two kinds of attributes that we can perform a split:

- *Numeric attributes*. For a numeric attribute A, a binary split A<u is performed to the set of data that it belongs. The candidate-split points are midpoints between every two successive attribute values in the training data. In order to find the split for an attribute on a node, the histogram C_{below} is initialised to zeros whereas C_{above} is initialised with the class distribution, for all the records for the node. This distribution is obtained when the node is created. Attribute records are read one at a time and C_{below} and C_{above} are updated for each record read. After

each record is read, a split between values we have and we have not yet seen is evaluated. C_{below} and C_{above} have all the information to compute the *gini* or *Entropy* index. Since the lists for numeric attributes are kept in sorted order, each of the candidate split points for an attribute is evaluated in a simple scan of the associative attribute list. If a successful point is found, we save it and we de-allocate the C_{below} and C_{above} histograms before we continue on the next attribute

- *Categorical Attributes.* If S(A) is the set of possible values of a categorical attribute A, then the split test is of the form $A \in S'$, where $S' \subset S$ and $[S'] \leq [S]/2$. A single scan is made through the attribute list collecting counts in the count matrix for each combination of class label that has size and attribute value found in the data. Once we have finished, we consider all subsets of the attribute values as possible split points that that have size less or equal to [S]/2 and compute the corresponding index. The information that is needed for computing the index is available in the count matrix.

5.5 Implementation of the split

When the best split point for a node is found, split is performed by creating two children nodes and dividing the attribute records between them. The partition of the attribute list of the splitting attribute is straightforward. We simply scan the list, apply the split test and move the records to the two attribute lists of the children nodes. As we partition the list of the splitting attribute, the "id" of each record is inserted into a hash table. Once all the "id" are collected, we scan the lists of the remaining attributes and examine the hash table with the "id" of each record. The result of the comparison specifies in which child to place the record. During the splitting operation, class histograms are built for each new leaf. There are some cases that a data set can not be further partitioned. For example in the training set of Table 2, it is impossible for a classifier to split the first two records into two separate data sets since there all have identical attributes except the classifying attribute *Lived*. There is no classification rule that can explain why the two records have different class labels. This is very usual case in large databases and especially in temporal databases where the stable data during the time appears to have the same values in different time-points. So, classification of this temporal data considering as classifying attribute the time, will lead to this situation.

…..	Gender	Date of Birth	Date of Stroke	Lived	…..
...	Female	5/8/1921	12/10/94	Survived	...
...	Female	5/8/1921	12/10/94	Died	...
...	Male	1/18/34	2/7/94	Survived	...
...	Female	3/28/51	12/9/94	Died	...

Table 2: Case of Impossible Partition

In this case, we consider the node with that data set as a leaf with more than one class label. A record of the data set of such a leaf belongs to a class j with probability r_j /r, where r_j is the number of tuples of each class *j* in that data set and *r* is the total number of the records of this data set. When an unclassified data set is given as input to the classifier it is possible that some of its records satisfy the classification rule that corresponds to the path from the root to such a leaf. Then these records belong to a class j of that leaf with probability: r_j /r.

5.6 Example of using Classification

We used Easy Miner to classify records of patients that had had heart attack based on the values of attribute *Lived*. By using a relatively small training set, we built a classifier for attribute *Lived* and after that we used that classifier to classify all the records in the medical database. A part of the results of the classification process is shown in Figure 1. The attribute *Predicted Lived* contains the prediction/classification for attribute *Lived*.

…	Sex	Date of Birth	Date of Stroke	Lived	Predicted Lived	…..
…	Male	9/26/1925	11/13/94	Survived	Survived (Probability = 100%)	…..
...	Male	9/21/1924	9/23/94	Died	Died (Probability = 100%)	....
...	Male	11/1/1921	9/23/94	-	Survived (Probability = 100%)	....
...	Male	10/7/1921	9/23/94	-	Died (Probability = 100%)	....
...	Female	3/28/51	12/9/94	Died	Died (Probability = 100%)	....
...	Female	4/24/51	12/9/94	-	Survived (Probability = 100%)	....
...	Female	5/8/1921	12/10/94	Died	Died (Probability = 100%)	....
...	Female	12/8/1921	12/10/94	-	Survived (Probability = 100%)	....
...	Male	1/18/34	2/7/94	Survived	Survived (Probability = 100%)	....

Figure 6: Classified Medical Data

6. Association Rules

Agrawal describes discovery of association rules in large databases in [8]. The initial motivation for association rules was to aid in the analysis of large transaction databases, such as those collected by supermarkets. The discovery of associations between various line items can potentially aid decision making within organisations. Using the formalism provided by Agrawal, association rules can be defined as follows.

Let I={i_1, i_2,..,i_m} be a set of *items*. Let DB be a database of transactions, where each transaction T consists of a set of items such that $T \subseteq I$. Given an *itemset* $X \subseteq I$, a transaction T *contains* X only and only if $X \subseteq T$. An association rule is an implication of the form $X \Rightarrow Y$, where $X \subseteq I$, $Y \subseteq I$ and $X \cap Y = \emptyset$. The association rule $X \Rightarrow Y$ holds in DB with *confidence c* if the probability of a transaction in DB which contains X also contains Y is *c*. The association rule $X \Rightarrow Y$ has *support s* in DB if the probability of a transaction in DB contains both X and Y is *s*. The task of mining association rules is to find all the association rules whose support is larger than a *minimum support threshold* σ_l' and whose confidence is larger than a *minimum confidence threshold* φ_l'.

6.2 A method for mining association rules

We use the semantic characteristic of decision trees to develop a method for finding association rules from a database. The rules that our method discovers have the form: $\mathrm{X} \Rightarrow \mathrm{Y}$, where X is a set of conditions upon the values of several attributes and Y a specific value of one attribute or a combination of several attributes' values. This attribute (or combination of attributes) Y is called *target attribute*. The method for mining association rules consist of

3 simple steps including Data Preparation, Generation of rules and Selection of strong rules which will be discussed in the following sections.

6.3 Data Preparation

Data preparation involves the following phases:

1. The *target attribute*, in which there is interest on finding association rules, must be selected and discriminated from the others. In case that there is interest on more than one attribute, then the *target attribute* can be constructed from the join of these attributes.

2. Generalisation may be performed on the *target attribute(s)* by using relative generalisation rules. If among the target attributes there are numeric ones, then their values have to be generalised by using generalisation rules or by grouping them in to ranges. By this method the target attribute is being converted into a categorical one with only a few values of higher level. As a result of this process, the discovered rules $X \Rightarrow Y$ will have higher confidence since Y will have also higher support.

3. Generalisation also could be applied and to the other attributes of the data set. Moreover, it is recommended for attributes that have a large number of distinct values. This results in viewing the data in abstractions that are more useful and in generating candidate data sets for finding association rules, that have quite significant support.

6.4 Generation of Rules

After the stage of preparation of data, the *target attribute* can be considered as the class label of the whole set of data. Hence, a decision tree can be constructed, based on that classifying attribute. Each path from the root to node of this decision tree represents one or more rules of the form:

IF *(sequence of intermediate conditions)* **THEN** *(classifying attribute value)*

The confidence of each generated rule from such a path is r_j / r, where r_j is the number of tuples of each class j that has records in the data set of that node, and r is the total number of the records of that data set.

6.5 Selection of Strong Rules

After extracting all the rules from the built decision tree, we select only the strong ones. As strong rules have been defined the rules $A \Rightarrow B$ that the support of A and B are above the minimum support threshold, and their confidence is greater than the minimum confidence threshold. In the following table, there are rules that were discovered by using this technique of Easy Miner. These rules concern the credit history database of bank customers.

Rule's Body	Support	Confidence
IF ((Marital_Status IN {Single})) THEN (Home = Rent)	16.22%	65.15%
IF ((Marital_Status NOT_IN {Single}) (Account_Status IN {60 days late}) (1 < Nbr_Children) (Savings_Account IN {Yes}) (1187< Mo_Income)) THEN (Home = Own)	12.29%	66.00%
IF ((Marital_Status NOT_IN {Single}) (Account_Status IN {Balanced}) (1611.50 < Mo_Income <= 2918.50) (Checking_Account IN {No})) THEN (Home = Own)	14.99%	100.00%

IF ((Marital_Status NOT_IN {Single}) (Account_Status IN {Balanced}) (3047.50 < Mo_Income <= 3633) (Nbr_Children < 3)) THEN (Home = Own)	10.57%	100.00%
IF ((Marital_Status NOT_IN {Single}) (Account_Status IN {Balanced}) (3633.50 < Mo_Income) (Nbr_Children < 3)) THEN (Home = Own)	10.57%	93.02%

7. Use of Sampling

Finally, in Easy Miner we have the option of using sampling method. For some data mining tasks in very large databases it is time consuming to perform data mining in the whole database. For this reason we have adopted the sampling method of the simple validation. In this method, a percentage of the data mining task relevant data is randomly selected from the whole data set. After that, this sample data set is treated as the training set for building a decision tree. When the decision tree is built, we validate it and update its statistics by testing the whole data set on it and watching how the records are distributed to the nodes of the tree. The advantage of this method is that there is need for sorting the numeric attributes and finding split points only in a relatively small sample data set and not in the whole data set.

8. Comparison with other methods

By performing generalisation on the examining data set, Easy Miner allows us to handle row data at higher conceptual levels and more meaningful abstractions. Moreover, generalisation addresses the scalability issue by compressing the examining data set. The generalised data is much more compact than the original one and so fewer operations are required in the process of building a decision tree.

By performing relevance analysis on the generalised data set, we reduce further the amount of the training data since we remove the less relevant attributes from it. This reduction on the size of the training set, results in faster and more accurate building of a smaller decision tree. As Quinlan mentions in [7], having irrelevant attributes in the data set that is examined increases greatly the error rate and size of the resulting trees. Because of the integration of generalisation and relevance analysis into decision tree building, Easy Miner construction of decision trees is much more efficient than SLIQ [5] and the serial version of SPRINT [6] and leads into better results.

The most established algorithms for discovery of association rules have problems when the examining data set contains numeric attributes. The methods that are based on the attribute oriented induction [8] have to generalise the numeric attributes and convert them into categorical ones. During this human-driven generalisation there is a lost of information because the user-defined generalisation is not always the most adequate and optimum to the request data mining task. In Apriori algorithm [8] the values of the numeric attributes are replaced by intervals. If the number of intervals is large, the support for any single interval is low. On the other hand, if the number of intervals is low, it can lead into rules with low confidence. To determine the number of intervals the algorithm uses K-partial completeness, which is based only on the ordinal properties of the data. Easy Miner does not have those problems since numeric attributes values are splitted into groups according to how well those groups are related to the values of the target attribute. Therefore it handles them according the needs of every user-requested data mining task.

9. Summary and Future Work

In this paper, we introduced our approach for data mining in time-oriented data. We presented Easy Miner, a data mining tool designed and developed at UMIST. All components of Easy Miner including relevance analysis, classification and association rules have been described in detail with examples. We are currently in the stage of optimising the performance of the tool and also carrying out extensive testing. At the same time we carry out tests with other domains, namely shipping data and stock data.

In the future, we will apply our methods to distributed databases using many workstations to hold the data. This has two justifications, firstly, to represent a distributed environment and secondly, to improve performance and reduce the costs. As far the temporal aspects are concerned, we would like to enhance the algorithm for automatic-generation of generalisation rules by employing temporal constraints as part of the problem definition. We believe that data mining over temporal data can lead to the discovery of high quality knowledge that can be used for predictive decision making.

REFERENCES

[1] Fayyad U, Piatetsky-Shapiro, Smyth Uthurusamy R. Advances in Knowledge Discovery and Data Mining. AAAI/MIT Press, 1995.

[2] Chen X, Petrounias I, Heathfield H. Discovering Temporal Association Rules in Temporal Databases. In the Proceedings of the International Workshop on Issues and Applications of Database Technology (IADT98), Berlin, Germany, July 1998.

[3] Kamber M, Winstone L, Gong W, Cheng S, Han J. Generalisation and Decision Tree Induction: Efficient Classification in Data Mining. In Proceedings of 1997 International Workshop on Research Issues on Data Engineering (RIDE'97), Birmingham, England, April 1997, pp 111-120.

[4] Loether H.J, McTavish D. G. Descriptive and Inferential Statistics: An Introduction. Allyn and Bacon 1993.

[5] Mehta M., Agrawal R., and Rissanen J. SLIQ: A Fast Scalable Classifier for Data Mining. In Proc. of the Fifth Int'l Conference on Extending Database Technology (EDBT), Avignon, France, March 1996.

[6] Shafer J, Mehta M., Agrawal R. SPRINT: A Scalable Classifier for Data Mining. In Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB), Mumbai (Bombay), India, 1996, pp 544-555.

[7] J. R. Quinlan. Improved Use of Continuous Attributes in C4.5. Journal of Artificial Intelligence Research 4, 1996, pp 77-90.

[8] Rakesh Agrawal and R. Srikant. Mining Quantitative Association Rules in Large Relational Tables. In Proc. of the ACM SIGMOD Int'l Conf. on Very Large Data Bases (VLDB), Zurich, Switzerland, September 1995.

[9] D. W. Cheung, Ada W. Fu, J. Han. Knowledge Discovery in Databases: A Rule-Based Attribute-Oriented Approach. In Proc. of 1994 Int. Symp. On methodologies for intelligent systems, Charlotte, N.C. Oct 1994.