Designing the Data Warehouse for Effective Data Mining

Iain McLaren
i.mclaren@searchspace.com

Searchspace Limited
60 Charlotte Street
London, UK, W1P 2AX

1 Introduction

Data mining is slowly gaining acceptance as a method for obtaining business intelligence from corporate data banks. This market has largely been driven by the development of desktop tools, such as IBM's Intelligent Miner, SGI's MineSet and ISL's Clementine. These tools offer a variety of techniques for automatically extracting potentially useful patterns from large data sets (Frawley, Piatetsky-Shapiro and Matheus (1991)). Some examples include the use of supermarket loyalty data to drive future marketing strategies, or the analysis of prior insurance claims in the setting of future insurance premiums. As desktop tools, the current crop of data mining products have often been criticised for offering poor support for data mining at the enterprise level. Enterprise data mining is characterised by the large amounts of data involved, and the integration of data mining with existing database technology (Gupta, McLaren and Vella (1997)). This paper looks at an issue which arises from these problems, namely the effective design of a database to support data mining, and the integration of these designs with existing data warehouse designs.

The requirements of decision support applications, including data mining, are entirely different from traditional transaction processing systems and therefore require new database design methodologies. Decision support applications range from simple query tools, such as those available from database vendors or specialist query tool suppliers such as Crystal and Brio, to OLAP multidimensional analysis systems which enable different aggregated views of data, such as those produced by MicroStrategy and Arbor Software, to intelligent data mining solutions, such as those available from IBM and SGI. These types of tool are characterised by generating much more complex queries than transaction processing systems, which rather than returning small result sets have to examine large sets of data in order to return their results. Also, the queries which are generated by these tools are largely unpredictable. On the other hand, these applications have fewer users than transaction processing systems, and updates are done in overnight batch processes rather than during online processing. These new characteristics have led to the development of a wide range of database design techniques which aim to minimise the amount of data which must be transferred from disk to answer a query.

This paper is structured as follows. Section 2 presents a review of the database design techniques employed by established decision support applications such as query and OLAP tools, highlighting their appropriateness to data mining solutions. Section 3 then introduces a number of advanced design techniques which have particular significance to supporting enterprise data mining. Finally, section 4 presents some conclusions about how these designs fit within a traditional data warehouse environment.

2 Traditional decision support designs

A number of database designs have evolved which are particularly suited to supporting the established decision support applications represented by query and OLAP tools. Many of these techniques are also appropriate in data mining environments and hence a review of these approaches follows.

2.1 Database design for query tools

There are a wide range of different query tools, from off the shelf tools supplied by tool vendors, such as Business Objects or Brio, and database vendors, such as Oracle or Sybase, to bespoke systems which are coded to support particular applications. Bespoke systems may be developed using either a 3GL, or a 4GL query development tool such as PowerBuilder. The aim of the query tool is to allow the user to set some query constraints, often through a forms based interface, as shown in Figure 1, and to have these constraints applied to a query on the underlying database. Depending on the flexibility of the query tool, the range of possible queries can be large. This means that the underlying database design must be able to support ad-hoc query patterns. Unlike traditional transaction processing systems which return small result sets, query tools often pose queries which can return large result sets. Query tools will also often allow users to view summary information of the underlying database, resulting in small result sets which have been generated from scans of lots of data.

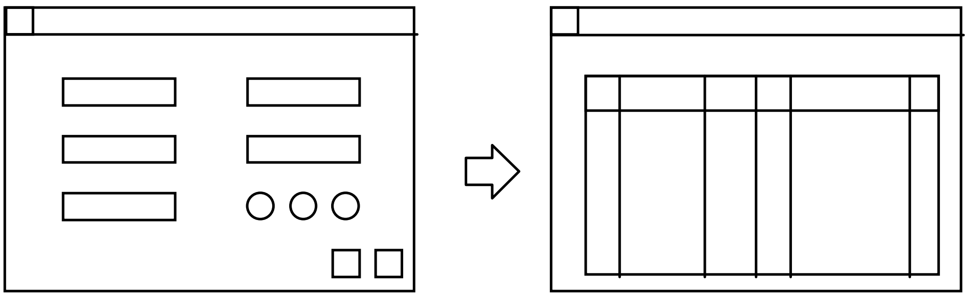

Figure 1. Query tool

There are two main ways in which to support ad-hoc queries. The first is to identify the most effective indexes on the data in order to support the majority of situations which the user is likely to query. Although the tool may allow a large variety of queries to be generated, there is normally a subset of attributes which can be indexed to support many of the cases. The other approach which can be used is to begin to denormalise the underlying database design. Although the queries are unpredictable, the dependencies between the underlying tables are fixed. That is, the joins which are to be applied to the tables from the query tools are known in advance. As the join is one of the most expensive operations to perform, avoiding performing joins is normally a good idea. This can be achieved by preparing the join in advance by holding it explicitly in another database table. Figure 2 shows *Buyer firm* and *Buyer client* tables joined explicitly in the *Buyer* table. This means that queries which previously ran against the two original tables, requiring a join, could now run as a simple query against the single *Buyer* table.

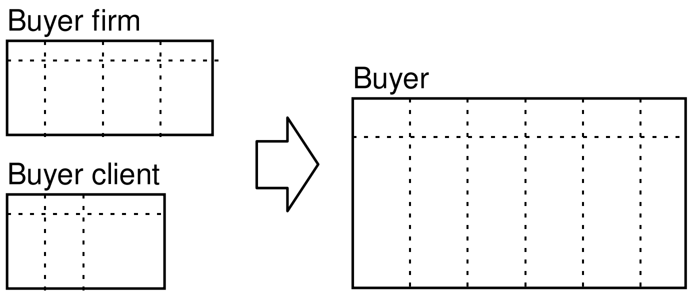

Figure 2. Explicit table joins

The cost of performing this denormalisation is that updates to the data will now take longer as there will be duplicated information held in the explicit join table. This can be minimised by only denormalising those columns which are actually used by the query tool rather than denormalising the whole table. For example, the query tool may query a single main table but perform joins with lookup tables to allow the user to query by name rather then the identifier which is held in the main table. To avoid this join the name column from the lookup table could be held explicitly in the main table.

The other characteristic of query tools, the retrieval of summary information, can also be supported via denormalisation. The generation of summary information is normally an expensive operation because large table scans are required to generate the required aggregations, such as summary of sales positions or revenue to profit ratios. To support these quickly at run time, summary tables can be produced, see Figure 3. These would be generated during the overnight batch processing window for the most appropriate summary positions. If there is a certain amount of flexibility in the summary positions which the user can request then some intermediate state could be generated overnight, leaving a small amount of additional work to be done at query time depending on the precise nature of the query.

Stock positions

Date	Firm	Stock	Position
10/1	BZW	BT	100000
10/1	BZW	BG	76500

Figure 3. Summary tables

The use of these techniques based on denormalisation should be made with care because of the necessity of maintaining the integrity of the data held. However a combination of these techniques can dramatically improve the performance of ad-hoc query tools, and are applicable in a range of data mining scenarios.

2.2 Database design for OLAP

OnLine Analytical Processing (OLAP) tools allow users to view summaries of data over many different dimensions at various levels of granularity. For example, viewing the total sales broken down by quarter, or viewing sales by region broken down by year. These views are normally displayed through graphs, such as those shown in Figure 4. The aggregated views of numeric columns can be presented over many different dimensions and OLAP tools allow users to drill down to a lower level of detail and roll up to a higher level of detail. There are two main approaches to OLAP. These are using OLAP specific multidimensional databases, and using a layer above a general relational database. As multidimensional databases, such as those produced by Arbor software, are optimised for OLAP they can provide good performance. However, they are not as flexible as relational based tools and often do not scale well to large data sets. The database design for relational based tools, such as those produced by MicroStrategy, are discussed in this section.

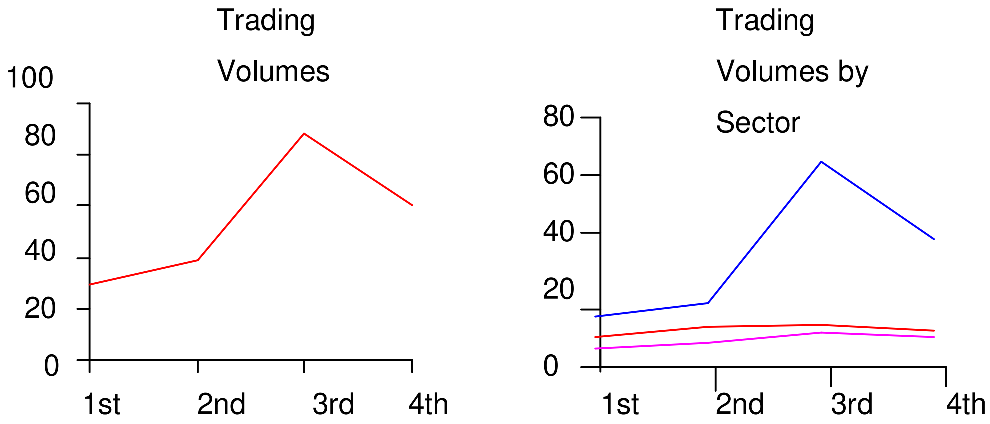

Figure 4. Multidimensional data views

To generate the multidimensional aggregates required for OLAP requires large scans of tables based on selections from the required dimensional attributes. If the underlying database is designed in the conventional manner then this can be very resource intensive. It will require retrieving complete records, containing many unwanted attributes, just to read the required numeric attributes being aggregated. The solution to this problem is to use a database design called the star schema (Inmon (1996)). This is illustrated in Figure 5. In the star schema there is a central fact table, off which dimension tables hang.

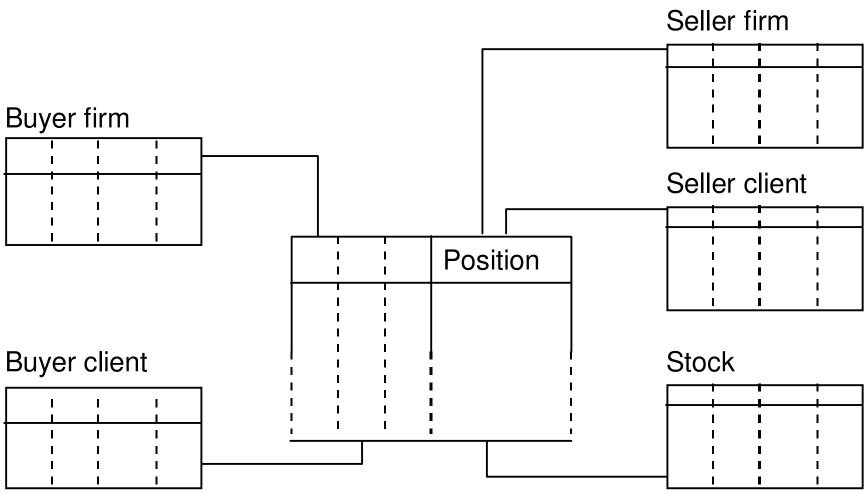

Figure 5. Star schema

The central fact table contains each of the numeric attributes which are to be aggregated and a link into each of the dimension tables. Each of the dimension tables contain the actual data about each of the dimensions over which aggregates can be produced. Using the star schema means that generating multidimensional aggregates requires a scan of only the columns which are actually required by the aggregate. This can speed up processing considerably. It is particularly appropriate to many data mining algorithms, as they are often based on generating statistical measures across the data.

To have a large impact on OLAP, the underlying database system's query processor must be aware of the star schema design. This is because database query processors have traditionally been developed to support conventional database designs, such as normalisation. Therefore, a traditional query processor will select a join order by first selecting one of the smaller dimension tables (e.g., the *buyer firm* table) and then look for a table which that table can be joined with. In the star schema, this means a join with the central fact table (e.g., the *position* table). As this is generally a very large table, a very large temporary relation will have been created. This temporary table will then be joined with each of the remaining dimension tables in turn. This approach is very inefficient. It was this weakness of relational database systems which lead to creation of many data warehouse specific database vendors, such as Red Brick.

The relational vendors are now catching up, making their products star schema aware. However, they have not yet reached the sophistication, in this respect, of the data warehouse specific systems. A star schema aware query processor will choose a different join order from the conventional approach. The processor will recognise that the database design follows the star schema, and first produce a cross product of all of the relevant rows in the dimension tables. This temporary result set will only then be joined with the larger fact table. This approach attempts to avoid generating the large temporary relation of the conventional approach. However, it is not ideal as it cannot support lots of large dimension tables. This would again lead to the generation of a large temporary result set. The real limitation of relational query processors in this respect is that they can only join on two tables at a time. To support the star schema well requires multitable joins. This is achieved by the data warehouse specific query processors. Relational vendors are beginning to release similar solutions.

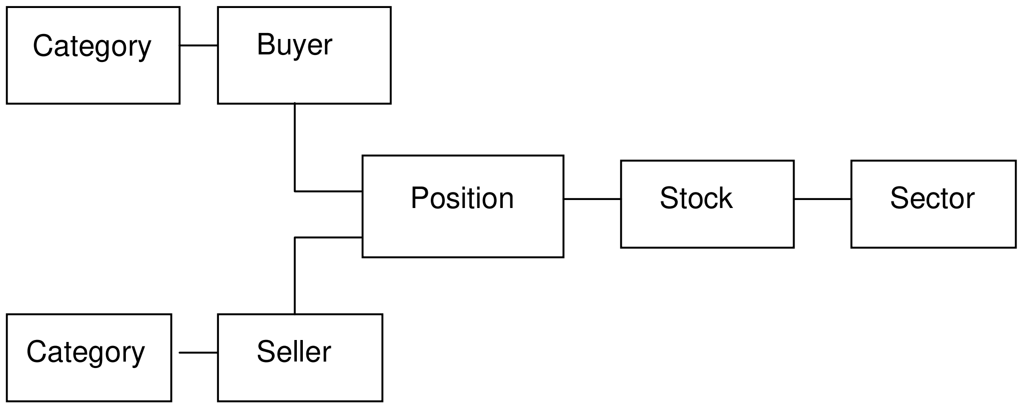

Figure 6. Snowflake schema

Despite the efforts of the relational database vendors, the relative immaturity of the OLAP market has meant that OLAP design has been continually evolving. One aspect of this has been the evolution of the star schema. An extension of this schema is often used where further hierarchical tables hang off the dimension tables connected to the central fact table. These hierarchical tables can define the various granularities to which a dimension can be investigated. This design is known as the snowflake schema, see Figure 6. This evolution means that for the time being many query processor developers are continually trying to catch up with ever more sophisticated database designs.

3 Database design for data mining

Data mining includes a wide range of techniques supporting the fundamental applications of classification, clustering and forecasting (Agrawal and Srikant (1994), Quinlan (1993)), as illustrated in Figure 7. The general aim of these techniques is the automatic identification of patterns in large amounts of data. Patterns which would be time consuming to find via human driven investigation.

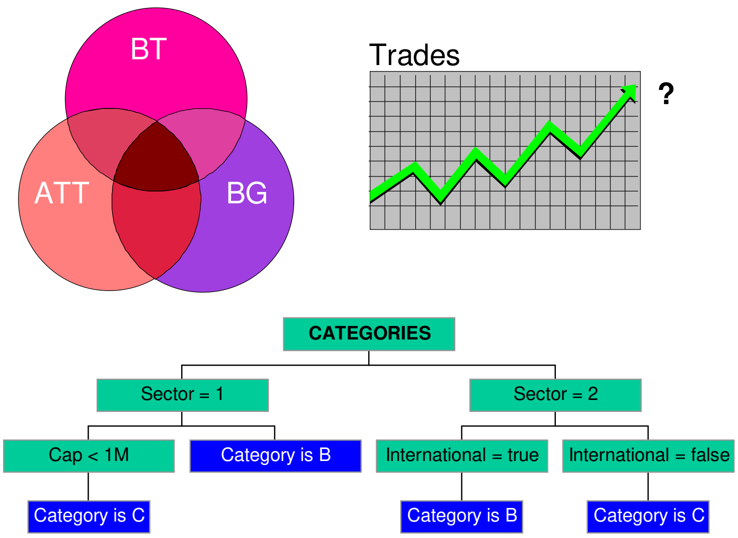

Figure 7. Data mining

To find these patterns requires scans of very large amounts of data to generate advanced statistical measures of the data. These scans are often applied on an attribute by attribute basis, to identify which attribute contains the most interesting information. Furthermore, because these attribute scans are computer driven, the access patterns are very unpredictable.

3.1 Attribute oriented storage

The large attribute scans used by data mining techniques benefit from an attribute oriented storage strategy for tables. Conventional relational databases store tables in a row by row fashion. This means that when a single attribute must be scanned, the complete row must be read from the database. This is very expensive when driven from an attribute oriented data mining technique. An approach which uses an inverted index, as illustrated in Figure 8, overcomes this problem by storing the table in an attribute oriented fashion. That is, all the values of each attribute are stored together. This means that when doing an attribute scan only the relevant information is read. It also means that the unpredictable access patterns of data mining techniques are supported, as tables are accessed on an attribute basis.

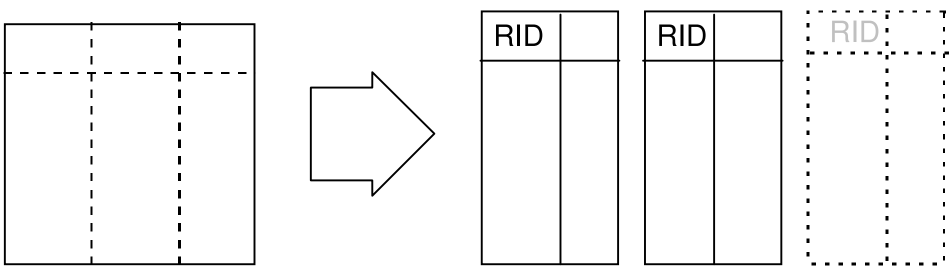

Figure 8. Inverted index

A further optimisation of the attribute oriented storage strategy is possible for symbolic values. This involves the use of bit mapped indexes. In a bit mapped index, the symbolic values are binarised to allow very fast look up for each value. This is illustrated in Figure 9. Each bit position in the bit mapped index represents a different symbolic value. If a bit position is *on* then the value represented by that bit position is the value of the attribute. This means that looking up particular values require a scan of a single bit column, using very efficient bitwise comparison operators. This approach works very well for low cardinality symbolic values, but can become inefficient when the cardinality becomes large. However, for many of the columns of interest to data mining algorithms, such as sex or status, it is a very efficient solution.

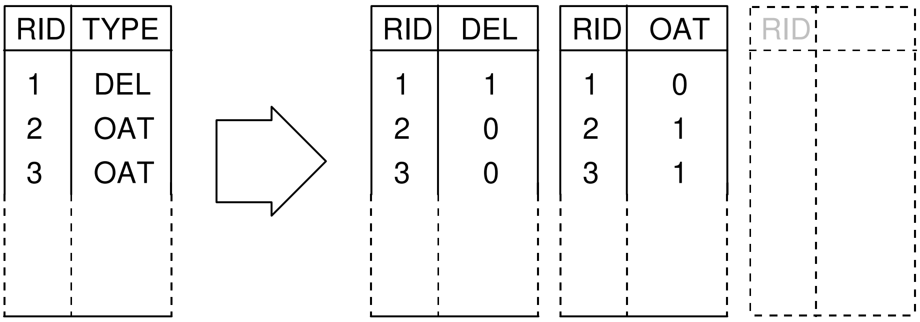

Figure 9. Bit mapped index

The performance improvements possible using these techniques can be dramatic. For example, to scan 10000 symbolic 10 character attributes from a 60 byte record requires reading 600,000 bytes using traditional methods. Using an inverted index requires reading 100,000 bytes. Using a bit mapped index, assuming there are 8 different symbolic values, requires reading just 10,000 bytes. When a data mining algorithm is repeatedly scanning different columns, the performance can improve by several orders of magnitude. These figures assume that the underlying database system is capable of efficiently storing and processing data stored in an attribute oriented fashion. This requires the underlying database to be able to efficiently find things by row identifiers, storing attribute values adjacently on pages on disk.

3.2 Sample oriented storage

Many data mining algorithms are compute intensive and require many passes over the data being examined in order to discover useful patterns. This has led to two potential solutions, either the introduction of parallel processing or the use of sampling (McLaren, Babb and Bocca (1997)). Most database systems now provide support for parallelism, however sampling is not something which databases naturally provide. This means that if sampling is of importance, because the data sets to be mined are very large, then an approach to sampling must be introduced by some other means.

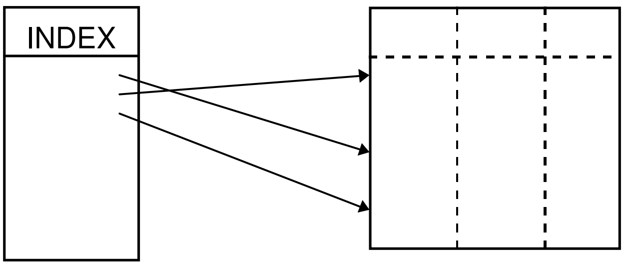

Figure 10. Hash index

One step to sampling is to create a pseudo random index. This is achieved by creating an index on a combination of attributes in the data which will lead to the records being extracted in a random order. The best way of achieving this is by deriving a new column which contains values generated by a random number generator as the records were loaded into the database, and then indexing this column. However, if this overhead is too great then an unusual combination of existing attributes can often lead to a pseudo random ordering of the records. To pick a random sample from the data simply requires the retrieval of a fixed number of records from the table with the hash index forced on the query processor, as illustrated in Figure 10.

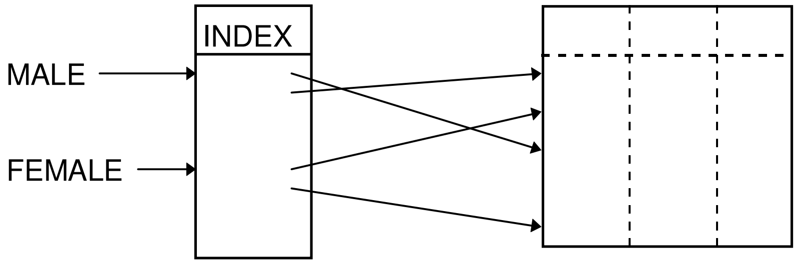

Figure 11. Index striding

A problem with generating samples from databases is that where you need the sample to include a representative sample from particular types of records within the table, where the different record types are defined by attribute values in the table, extracting a random sample will not guarantee that each type of record is equally represented. This may lead to the introduction of bias to the data mining algorithm. A possible solution to this is the use of index striding (Hellerstein, Haas and Wang (1997)). Index striding requires an index to be placed on the attributes which require a balanced sample to be taken. Unless this attribute somehow defines the order of data on disk, for example if it is a component of the clustered index on the table, then the order of records accessed by this index will largely be random and therefore can form a good basis for sampling. The distinct values within this attribute can then be selected via simultaneous queries to extract the required number of records for each value which is to compose the sample. This is shown in Figure 11. This approach also allows explicit bias to be introduced by configuring the sample components with different weightings. Obviously this type of approach is enhanced by the techniques for attribute oriented storage described earlier.

4 Summary

This paper has described a range of design techniques relevant to the development of data warehouses for data mining. These contrast with conventional techniques used for transaction processing systems due to the different requirements of these two types of system. These design methods have included the use of denormalisation, star and snowflake schemas, as used in traditional decision support, and the use of attribute and sample oriented storage, specifically aimed at enterprise data mining.

It is clear from these techniques that a modern decision support system which aims to meet a range of different requirements needs to obey a hybrid design strategy. This strategy should consider the requirements placed on the database, and use a mix and match approach to choosing combinations of the different strategies to obtain the optimum solution which is suitable to both data mining and the other decision support applications. When developing the design, careful consideration of the underlying database's capabilities is required. The designs outlined in this paper are not yet fully supported by all the database vendors. Also at present, existing data mining tools are unaware of how to exploit these designs effectively. This level of sophistication is required if true enterprise data mining is to be achieved.

References

Agrawal R and Srikant R (1994). *Fast algorithms for mining association rules*. In: Proceedings of the 20th International Conference on Very Large Data Bases.

Frawley W, Piatetsky-Shapiro G and Matheus C (1991). *Knowledge discovery in databases: an overview*. In: Knowledge Discovery in Databases, G. Piatetsky-Shapiro and W. Frawley (eds.), The AAAI Press.

Gupta H, McLaren I D and Vella A D (1997). *A framework for database mining*. In: Proceedings of British National Conference on Databases.

Hellerstein J M, Haas P J and Wang H J (1997). *Online aggregation*. In: Proceedings of ACM SIGMOD Conference on Management of Data..

Inmon W H (1996). *Building the Data Warehouse*. Wiley.

McLaren I D, Babb E and Bocca J B (1997). *DAFS: Supporting the knowledge discovery process*. In: Proceedings of the First International Conference on Practical Applications of Knowledge Discovery and Data Mining.

Quinlan J R (1993). *C4.5: Programs for machine learning*. Morgan-Kaufmann.