Interactive Predictive Analytics with Columnar Databases

Martin Oberhofer, Michael Wurst

{martino,mwurst}@ibm.de.com

Abstract: Predictive Analytics is usually seen as highly interactive task. Paradoxically, it is still performed mostly as a batch task. This does not only limit its applicability, it also sets it apart from a task that is conceptually very close to it, namely OLAP analysis. The main reason for considering mining a batch task is the usually very high execution time on large data warehouses. While novel hardware offers the ability of highly distributed execution of predictive analytics algorithms, this level of parallelism cannot be exploited within the traditional row-based database paradigm. Columnar databases offer a solution to this problem, as the underlying datastructures lend themselves very well to parallel execution. This reduces the repsonse time for mining queries several magnitudes for some algorithms.

While making mining faster and more responsive is already nice in itself, the real value of low response times is allowing completely new ways of interacting with huge data warehouses. In this arcticle we give a survey on the opportunities and challanges of interative, OLAP-like mining and on how columnar databases can support it. We exemplify these ideas on a task that is especially attractive for interactive mining, namely outlier detection in large data warehouses.

1 Introduction

On-line analytical processing (OLAP) enables simplified data analysis exploiting data summarization and aggregation techniques for drilling, pivoting and slicing of the data. OLAP is typically done in a Data Warehouse (DW) environment. Data mining complements the data analysis done through OLAP by automating the discovery of implicit patterns and interesting knowledge hidden in large amounts of data - which is not necessarily stored in a DW and has a broader set of functions such as association rule discovery, classification, prediction or clustering.

While the integration of mining and OLAP as well as interactive mining are highly relevant in many areas, they have received surprisingly little attention so far. Some research on the integration of OLAP and data mining has been done in the 1990's already, coining the term online analytical mining (OLAM). The value of an OLAM environment is typically justified by several reasons. First, a DW environment has information processing tools including tools to profile, cleanse and transform data from various operational sources into the DW. Second, data mining, data needs to have good quality to deliver good results. The effort for cleansing the data for data mining is not needed if data mining is performed in a DW. Third, before a user can effectivly mine data, the user might want to do explorative data analysis first. Within an OLAM environment, a user can leverage OLAP for explorative data analysis and then switch to data mining. In a tightly integrated OLAM environment, switching between the two is seamless for the end user. For the OLAM environment, various authors (e.g. [GRW08] and [HK06]) proposed architectures which work as follows:

Initially, through an Extract-Transform-Load (ETL) process data is extracted from various operational data sources which are typically relationalal databases. Then the data is profiled, cleansed, transformed to the common model of the DW and loaded. The data model in the DW is usually a snowflake or a star schema in a relational database. Based on the DW model aggregates such as cubes are computed or multi-dimensional databases are built - both to accelerate OLAP query execution but still in relational structures. For the user, there is one integrated UI tool for OLAP and data mining and the user can switch seamlessly between both, e.g. selecting a cube and then apply mining to only this subset. Through this UI the user submits OLAP queries or data mining requests which are received by the OLAM engine which then leverages the Data Mining or the OLAP engine for fulfilling the user request by processing it on data cubes or the multi-dimensional database.

Even though there has been some research and prototypes for a tighter coupling of OLAP and mining, so far the practical use is limited. A major reason for this is the large difference in performance and response times for OLAP queries and for mining procedures. While there are techniques to deliver very short response times for OLAP queries, mining queries often take hours to run. Data mining is still considered a batch job for the following reasons:

- The data volume is typically so large it can not be held in bufferpools which are the main memory area for relational database. Thus, the runtime of data mining processes is typically a batch-style procedure and not an interactive *online* process.
- The cost for large main memory areas for relational databases was too high.
- Existing relational database approaches do not exploit in an optimal manner the new hardware architectures with multi-core CPUs and multi-CPU servers.
- Data compression offered by columnar database techniques is not used in the relational database space yet.

In the following we will show how columnar databases used for accelerating OLAP queries can be used to accelerate data mining as well, enabling both to operate on the same level of interactivity.

2 Concepts of Columnar Databases

Columnar databases are a well-known concept - they date back to the 1970's. Figure 1 shows through an example the conceptual structure of a relational database and in contrast Figure 2 shows the same data in a columnar database. In essence, a columnar databases

Cust ID	First Name	Last Name	Age	City	State	Country
1	John	Smith	31	Santa Cruz	CA	US
2	Alex	Morgan	45	San Jose	CA	US
3	Sarah	Adams	23	Los Angeles	CA	US
4	John	Miller	21	San Diego	CA	US

Figure 1: Structure of a relational database

RID	CUST ID	RID	First Name	RID	Last Name	RID	Age	RID	City	RID	State	RID	Country
1	1	1, 4	John	1	Smith	1	31	1	Santa Cruz	1-4	CA	1-4	US
2	2	2	Alex	2	Morgan	2	45	2	San Jose				
3	3	3	Sarah	3	Adams	3	23	3, 4	San Diego				
4	4			4	Miller	4	21						

Figure 2: Structure of a columnar database

organizes the physical structure in columns rather then in rows. Values stored multiple times in the relational model are stored only once in the columnar model saving space. Using the columnar approach often shrinks the data volume by a factor of 10 to 100. That means a signifcantly larger amount of data can be loaded into bufferpools in main memory significantly improving query processing performance. Another advantage of the columnar approach is that it is much better suited for OLAP processing where typically only a subset of columns in a table is needed for processing. The columnar database allows access per column whereas the relational model only allows row access even if in each row read only a small number of columns is needed by the query. Thus, the columnar databases approach allows significant reductions on the amount of IO access needed to read the data from the hard disks improving performance. The downside of the columnar approach are insert or update operations which are significantly slower than in a relational model. Thus, columnar databases are very good for OLAP workloads whereas the relational database approach outperforms the columnar database in OLTP scenarios.

Columnar databases store data in a column-centric way. For each combination of column and value there is a list of data points that have this combination (e.g. GENDER=f $\rightarrow$ r1, r5, r7). These lists are represented in a way that makes intersecting them extremely efficient. This also makes selection operations over conjunctions very efficient, as this operation is reduced to intersecting several lists. Columnar databases are becoming very popular and there are many available products (e.g. Vertica, ParAccel, Infobright) that use different kinds of representation and indexing. In the following we simply assume that there is an efficient mechanism for selecting records based on attribute/value combinations and of intersecting two lists of records.

3 System Evolution Enabling Interactive Outlier Detection

3.1 Columnar Databases and new Hardware Architectures

Columnar databases for OLAP mostly benefit from lower I/O costs through in-memory processing, while they are often far less compute intensive. The opposite is true for data mining. Here I/O can be a minor factor compared to the computation time. Therefore, data mining would usually not directly profit from a columnar in-memory storage. The picture changes, however, if we look at the evolution of hardware:

- The CPU architecture changed from one core to multi-core CPUs. Per core, multiple threads are possible.
- Servers with 8, 16, 32 or 64 multi-core CPUs are normal today providing a previously unparalleled degree of parallelism.
- The prices for main memory significantly dropped and are very affordable now. Thus servers with 2 TB main memory are common computing infrastructure today. A 20 TB large DW in a relational model fits now easily into the 2 TB main memory with the columnar approach.

These capabilities can only be exploited by mining algorithms if the underlying data structures allow for massively parallel execution. This is where the real boost of mining through columnar storage comes from: the ability to solve problems with dozens of threads in parallel, with very fast communication among threads. While there is some existing research work, e.g. [AMS96] and [LOPZ97], research on mining data in columnar databses has again gained increasing attention with the capabilities offered by hardware enabling high degrees of parallel processing. We now show how columnar structures can be exploited for interactive outlier detection.

3.2 A Unified, Columnar Architecture for OLAP and Data Mining

As shown in Figure 3, the new solution architecture has as the two major changes:

- Replacement of the relational database serving as DW with a columnar database
- Replacement of the relational structures for the multi-dimensional database or cubes with a columnar appoach

With this proposed solution architecture, in essence we are making the following paradigm shift: The traditional approach in the DW environment was characterized by the fact that only a very limited fraction of the data was in main memory and only a few scans were done in the DB for data mining since they took very long to complete making data mining a batch process. With the new approach, a lot of data is in main memory due to two

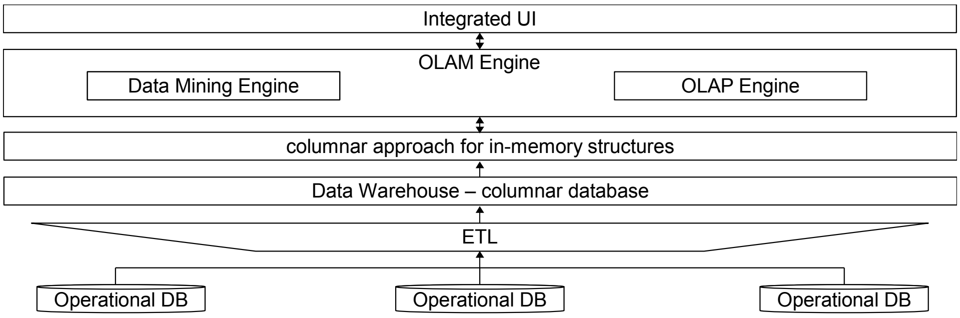

Figure 3: Conceptual overview of new OLAM architecture

reasons: cheap main memory and columnar database approach significantly reducing data volumne. With the underlying multi-core, multi-processer server architecture as well as by the fact that columns can be scanned in parallel independently from each other entire new levels of parallel processing for OLAP queries and data mining is possible. In some cases, the scans of data can be completely avoided due to the fact that the data is stored with the columnar approach - examples include min, max or count(distinct) operations.

4 Interactive Outlier Detection in Large Data Warehouses

4.1 The Challenge of Interactive Outlier Detection

Statistical data analysis is one of the key tasks to gain useful insight from large amounts of data. While there are many sophisticated methods for this task (ranging from simple cube analysis to sophisticated predictive analytics models), they have one challenge in common: outliers in the data. Outliers are data points that leaked into the dataset through errors in data acquisition, representation or processing. Typical examples are typos while manually acquiring the data, inconsistencies through false handling of special cases in pre-processing and many others. Unfortunately, such data quality issues appear very often.

Outliers are assumed to behave differently than the majority of data points. However, the opposite does not necessarily hold: just because something appears seldom does not necessarily make it an outlier. It could just be a case that does not occur very often. Depending on the application scenario, this could actually be an especially interesting and valuable case. Therefore, outlier detection is an inherently interactive task of finding unusual patterns in the data, validating and flagging them as outliers or as usual data.

Most algorithms to automatically identify unusual data rely on thresholds. A common approach in statistics, for instance, is to check how far a given value is from the mean value of corresponding data base column. If the age column in a record contains a values that is 100 times the average, then this could be seen as an outlier. However, the difficult problem now is to identify and set the right thresholds. As there is no a priori definition of where unusual but valid data points ends and where outliers starts, the analyst will subsequently try different thresholds, analyze the list of identified outliers and then re-adjust the threshold accordingly. Also, the analyst will already know that for some cases the predictions of the algorithm can be ignored altogether, as they are known to be usual data points or they are anyway filtered in a subsequent step.

To be able to do an analysis as just described, a system to interactively identify outliers must allow for the following:

1. Combine outlier analysis with slicing and dicing known from OLAP
2. Adjust outlier thresholds interactively
3. Update a list of outliers in near-real time
4. Offer fine grained functionality to filter/ignore some outliers

Existing solutions to outlier detection are mostly based on the idea of finding patterns of usual behavior in the data and then flagging everything that does not match these patterns as an outlier (see [BCV09] for a survey).

4.2 Interactive Outlier Detection on Column-store Warehouses

In the following we assume that our dataset consists of a set of attributes A in **A**, each of which can take a given set of values $\mathbf{V_i}$ (e.g. attribute gender with $\mathbf{V_{gender}} = \{m, f\}$). We also assume a set of records or data points **R**. Each record contains a value for all attributes in **A**.

The aim of outlier detection is to identify a subset of records $R'$of **R** that contain unusual combinations in at least one attribute A or in a combination of attributes.

We achieve this by first automatically analysing what the combinations between columns are that are very common. Everything that does not fit into this scheme is considered to be an outlier. This approach has been applied in the past successfully in many different domains (see [BCV09]). We extend this approach in a way that makes it possible to change the settings in a very fine grained way and get the corresponding results in real-time.

Two types of outliers are assumed:

1. Outliers are records that contain values that are statistically very uncommon
2. Outliers are records that contain values that are common - however they represent an statistically uncommon combination

For the first task we simply put a threshold on the frequency of an attribute value pair (or range for numerical fields). Pairs appearing at least in a given fraction of records are considered as normal. All other possible pairs are flagged as outliers. GENDER=f may appear in 40 % of the records, GENDER=fem only in 1%. If the threshold is 5% then the first record is flagged as normal data point, the second as outlier. Obviously, an optimal threshold could differ for each of the attributes and must usually be set interactively.

For the combination of attributes we refer to the concepts of rules (for an overview see [AS94]): We assume, without loss of generality, that a rule is always written as follows:

$$(\bigwedge A_i \;\; OP \;\; v_{ij}) \Rightarrow A_k \;\; OP \;\; v_{km} \tag{1}$$

where OP can be any of equality for categorical attributes or greater/lesser than for numerical ones. A simple example would be:

$$(AGE < 10) \;\; \wedge \;\; (BALANCE < 100) \Rightarrow (INCOME < 30) \tag{2}$$

A rule determined from data usually has a confidence score. The confidence score denotes the fraction of records for which the conclusion holds if the premises hold. It can be seen as the strength of a rule. We may assume that the strength of the above rule is 95%. If our threshold for usual behaviour is a rule strength of at least 90%, then we must assume the rule above as usual behaviour and a record with AGE=5, BALANCE=50 but INCOME=100000 would be flagged as outlier. If the threshold would 97%, this would not be the case.

The threshold once the strength of the rule is seen as a reliable describing normal behaviour strongly depends on the application. A suitable threshold must be determined in an interactive process of setting a threshold, determining the outlier this would yield, then resetting it and re-running the outlier detection until a suitable level is found.

It is also possible and quite likely that some columns or rules should be ignored. An example would be a rule that states if a flag is set, another check values is mostly true. We may however know that for a new source system this is no longer required and that violating this rule has no significance. We may also know that the column AGE will anyway not be used in the subsequent analysis, so any violations concerning AGE can be ignored.

It is essential that users can set the confidence threshold and ignore flags on columns and rules in a way that they get immediate feedback on which records would be considered at outliers given the settings, which is not supported by existing approaches. We therefore additionally need a procedure that allows determining in near real-time which records are not covered and must be flagged as outliers. Also, the user should be able to ignore individual outliers, so that the system must be able to keep an *ignore list* of records that should not appear in the result. The overall process is as follows:

1. Build initial rule set and attribute-value counts
2. User (de-)selects columns / rules / records and adjusts thresholds
3. Determine the outliers in real-time
4. Filter deselected outliers
5. Present the outliers to the user (which can then continue with step 2 to further refine results until satisfied)

4.2.1 Identifying Rules and common Attribute-value Pairs

Both common pairs of attribute/value and rules with high confidence can be identified using association rule mining algorithms. As this process only needs to be performed once, efficiency is not critical at this point. Association rule algorithms require a support threshold and a confidence threshold. For the initial run, both are set to a user-defined minimum. This minimum is the lowest possible value for interactive threshold adjustment. All rules and attribute/value combinations are attached with their actual support and confidence for later processing.

4.2.2 Setting Thresholds and Ignore Tags

The user can now adjust the support threshold for attribute/value pairs and the confidence threshold for rules, possibly differently for each column. The user can also tag one or more rules or columns with an *ignore* tag. A rule that is tagged as *ignore* is not evaluated in the subsequent process. A column that is tagged as *ignore* is removed from all rules, which may subsequently lead to the removal of rules with an empty premise (as there is only a single consequence, this case is already covered by the simple attribute/value pairs). Finally, the user can flag records as *ignore* and thus remove them from the results. These setting uniquely identify which records are flagged as outliers, namely exactly the ones, for which either a selected rule does not hold or for which at least one attribute/value pair occurs that is not frequent enough.

4.2.3 Real-time Evaluation of Outlier Records

Having defined the conceptual notion of an outlier, a core question is how to identify the set of outliers in near real-time for interactive work. After a user updated one or several of the settings, a new list of outliers needs to be identified. This is achieved efficiently by exploiting the fact that we are working on a columnar database. Columnar databases store all possible values for a given column and point them to a set of records ids with the corresponding value in that column. They also store all counts, which can be retrieved without accessing the actual records.

For attribute/value pairs, this is very simple, as the frequencies for each attribute/value combination can be read directly from the columnar database. All records that are considered outliers based on this criterion are added to a global outlier list. To capture outliers based on rules, we need a more complex procedure. In a first step, all attribute/value pairs are sorted using an arbitrary criterion. Now, they are stored in a prefix tree. Each node contains exactly one predicate $A_i \ OP \ v_{ij}$. A path from the root of the tree to a leaf is a conjunction of such predicates. Each node contains a list of all rules, with a premise that is identical to the path from the root to this node. These rules also contain their confidence and the *ignore* flag.

Upon a request, we need to traverse all paths from the root to a leaf node. Please note that this can happen very efficiently in parallel. During traversal, each node is attached with a list of records that fulfill all the predicates that lead to this node from the root. As we move on from a node to one of his child nodes, we simply intersect the currently attached list of records with the ones entailed by the predicate of the child node (intersection is very cheap). We continue this process until we reached all the leaf nodes. Depending on the implementation of underlying columnar database, the overall process is at most linear in the number of records.

Each of the nodes in the prefix tree is attached with a set of rules. As we traverse a node, we first check which of the rules are activated in the current configuration (based on threshold or ignore tag). All active rules are then evaluated by intersecting their consequence with the list of records at the node. All records not in the intersection are added to a global outlier list. Again, performing this intersection/difference can be achieved extremely efficiently on as the records that contain the consequence can be retrieved directly and intersecting two sets of records is directly supported by the columnar database.

After traversal has finished, the list of records flagged as outliers is further filtered by the list of outliers to ignore using a set difference operation. The result is presented to the user. This list can be sorted by the number of rules violated or any other criterion. Also, a partial list can be shown to the user while the tree is traversed, such that she can see first results while the outlier detection process is still running.

4.3 Evaluation

To evaluate the ability of the above procedure, we used a row-based and a columnar in-memory database, such that I/O would not affect the performance in either case. We used a set of rules, or more importantly literals that would be evaluated. The machine was a UNIX Server running AIX 6 with 8 CPU and 16 GB main memory. We varied the number of records and the number of rule literarals and measured the response time on both databases. Figure 4 shows the result.

In both approaches, the response time increases with the number of records and with the rule literals. However, the response time for the columnar storage is much lower than for the row based storage. While even for moderate database sizes, using row-oriented storage leads to response time that are inadequate for interactive work, the column store is still applicable. As all the steps in the rule evaluation can be executed in parallel, a scale-out is rather easy, in cases that the number of records (and thus the response time) exceeds a tolerable level.

5 Summary

OLAP and predictive analytics are two closely related task that can be nicely intergrated with each other. Such an integration is, however, not yet pratically achieved, as the response time for OLAP queries often allows for interactive queries, while the response time of data mining queries is much higher. We showed that highly parallel hardware, in conjuction with the columnar storage paradigm, allows for a massive accerlation of mining

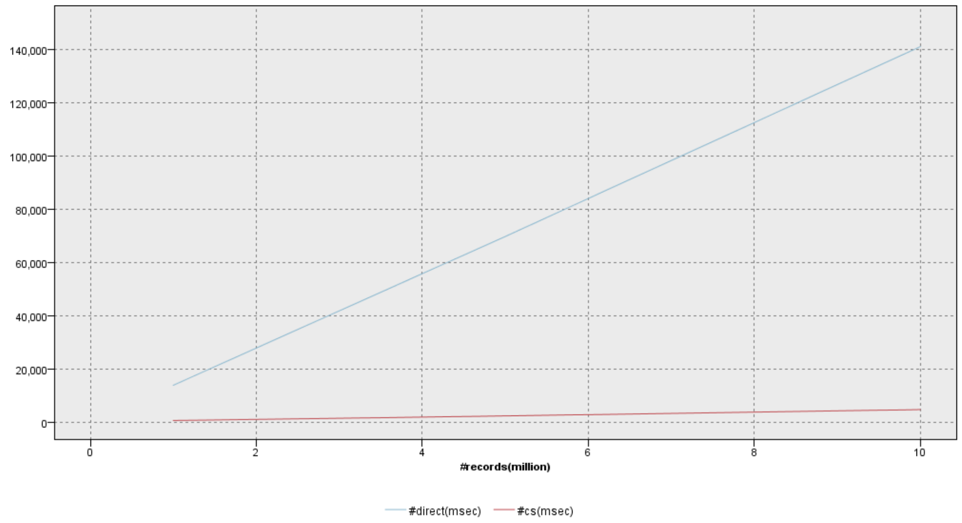

Figure 4: Performance comparison of a relational versus columnar database

queries. This allows for new kinds of applications. We demonstrated this on the task of outlier detection in large data warehouses. We expect to see many further applications of this kind in the near future.

References

[ABH09] Abadi, D. J. / Boncz, P. A. / Harizopoulos, S.: *Column-oriented database systems.* In: Proceedings of the 35th international Conference on Very Large Data, p. 1664-1665, 2009.

[AMS96] Agrawal, R. / Mehta, M. / Shafer, J. C.: *Sprint: A scalable parallel classifier for data mining.* In: Proceedings of the 22th international Conference on Very Large Data, p. 544-555, 1996.

[AS94] Agrawal, R. / Srikant, R.:*Fast Algorithms for Mining Association Rules in Large Databases.* In: Proceedings of the 20th international Conference on Very Large Data, p. 487-499, 1994.

[BCV09] Banerjee, A. / Chandola, V. / Kumar, V.: *Anomaly Detection: A Survey.* ACM Computing Surveys, Vol. 41(3), Article 15, July 2009.

[GRW08] Guo, Y. / Rao, W. / Wang, J.: *Research of OLAM Module based on SQL Server.* In: 2008 International Conference on Computer Science and Software Engineering, Volume 4, p. 419-422, 2008.

[HK06] Han, J. / Kamber, M.: *Data Mining. Concepts and Techniques.* Morgan Kaufmann, 2006.

[LOPZ97] Li, W. / Ogihara, M. / Parthasarathy, S. / Zaki, M. J.: *New Algorithms for Fast Discovery of Association Rules.* In: 3rd International Conference on Knowledge Discovery and Data Mining, p. 283-286, 1997.