ABSTRACT

The complexity of large-scale data-warehouses makes it difficult for users to create reports. We consider the problem of presenting data warehouse metadata to report creators and propose personalized recommendations and a search mechanism based on collaborative filtering. Recommendations guide report creators to help them to pick relevant data and to avoid duplicating already existing reports. The technology is deployed within HP for reporting against HP’s global data warehouse.

1. INTRODUCTION

Enterprise-scale data warehouses are often huge and complex. Any Business Intelligence (BI) tools operating against them face a big problem in making it simple for users to extract information. There is just too much in there for novice users to understand and to be confident that they are asking for exactly what they really want. For example, Yotta, the standard reporting and data extraction tool for HP’s own Enterprise Data Warehouse (EDW), has exactly this issue. There are about 56,000 registered users worldwide. The vast majority of them are business users and not information management specialists. Our goal is to make it easy for them to create reports. The challenge is that it is impossible for a user to make intelligent choices among thousands of tables and dimension hierarchies, tens of thousands of columns (Table 1), and their explicit and implicit relations in the data warehouse metadata.

Such large data warehouse use-case has some distinguishing features that are overlooked in the previous art. First of all, users cannot simply explore the system by running multiple reports because the overwhelming majority of meaningful reports require considerable time to execute. Moreover, the data warehouse has execution report queues, which extends the time between submitting a report for execution and getting the results. This aspect makes tuple-based approaches almost impossible to apply. Second, the logical structure that is available to users is very different from the physical schema of the data warehouse because it hides a non-trivial integration level. As a result, it is difficult to address the problem with SQL-level models like those described in [3].

We address the problem by utilizing user-centric data, namely, reports execution logs. We use collaborative filtering to recommend the most relevant columns to be included in a report and to provide contextual search over relational data. Data column recommendations are based on the content of existing successfully executed reports created by a particular user and other users and on the popularity of these reports. The way the data is used in a report (displayed or used as a filter) is taken into account. We also provide recommendations on existing reports that are similar to the one being created and which might provide a ready-made alternative. Recommendations are personalized and are computed instantly when a user adds or removes a column from a report. The search mechanism for both reports and columns is based on the full-text index of a materialized data graph. However, the textual data is very sparse and we apply custom relevance ranking based on the same item popularity mechanism that we use for recommendations. Our main contribution is in application of collaborative filtering to metadata exploration in real-world data warehouse.

Table 1: The scale of the EDW content

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg daily data load</td>
<td>5Tb</td>
</tr>
<tr>
<td>Physical tables</td>
<td>2260</td>
</tr>
<tr>
<td>Columns</td>
<td>82202</td>
</tr>
<tr>
<td>Dimension hierarchies</td>
<td>1122</td>
</tr>
<tr>
<td>Reports</td>
<td>191000</td>
</tr>
</tbody>
</table>

2. RELATED WORK

The focus of our paper is in application of a recommender system in the enterprise data warehouse. There are several works that use context for recommending items from a database. A traditional approach in such case considers data tuples, for example, contextual collaborative preferences are used in [2] to rank tuples from query results.

QueRIE project [5, 3] discusses a use-case which is the closest to ours. QueRIE recommends select-project-join SQL queries based on the queries executed during the current user session. They consider query similarity that is defined based on explored tuples [5] and on fragmenting queries [3].
CareDB [11] is an example of a personalized database that introduces context aware query operators. From the execution point of view, the operators are more expensive than their SQL counterparts due to necessity of out of database calls. The authors employ user context and profile to recommend SQL queries. They apply the proposed technique to restaurant recommendations.

Recommender systems have been applied in various domains. The most well-known are Web recommendations that include product recommendations “users who bought this also bought” (Amazon1) and music and movie recommendations (MovieLens2). There are recommendations of news (Google News3), places to visit like restaurants, clubs, and so on (Foursquare4), travel (GoGoBot5), software (Apple App Store6), etc. A big list of Web applications is given in [13]. The applications can be classified by the type of recommended item: entertainment, content, e-commerce, services; or by environment: mobile, private, enterprise. The notions of context and personalization are tightly bound up with recommender systems. Recommendations that make use of user context tend to be more helpful than static ones [1]. Recommender systems can be content-based [15], collaborative filtering [14, 7], demographic, knowledge-based, community-based [6], hybrid [4]. In our paper we follow the collaborative filtering approach since we have enough statistics for users (however, combination with knowledge-based approach is also feasible). The approach examines user-item interaction data and builds a recommender model based on similarities between items or users [14, 8]. We adapt this approach to the requirements of our system, which are scalability and robustness.

3. OUR SOLUTION

3.1 Approach

Most interactions between a user and EDW (mediated by Yotta) require the user to make selections: of standard reports, of filters to be applied, etc. Yotta originally presented these choices as hierarchical lists of all the options available to the user. Although Yotta used knowledge-based ways of reducing the number of options, sometimes the user had to browse through hundreds of alternatives. Our strategy was, firstly, to supplement all the browse functionality in Yotta with lists of recommendations. Secondly, we provided a custom keyword search mechanism. Both recommendations and search results are ranked by their relevance to the user. Yotta uses a metadata repository to define all its operating parameters: data sources, content, users, access rights, reports and so on. It also maintains logs of all the jobs it runs against the EDW. We mine that data to determine what the user is likely to want to do.

3.2 Architecture

The response time for offering recommendations must be fast, so the (heavy) task of analyzing the Yotta metadata and report execution data, generating ranking data and building full-text indexes is done offline at regular intervals. The business logic of finding recommendations and doing searches is implemented in T-SQL. Yotta calls these components through lightweight web-services (Figure 1).

![Figure 1: Architecture of YottaRanking.](image)

3.3 Recommendations

We introduced recommendations in several key points of the user interface of the business intelligence tool. One of these points is the process of building reports. We provide recommendations on what columns to include in a report and on what existing reports might make a custom report unnecessary. For both, we adapted an item-based approach derived from the collaborative filtering domain [8]. The idea is to create an interaction matrix \( A^{m \times n} = \{a_{ij}\}_{m \times n} \), with \( m \) reports and \( n \) reporting options (that is, data warehouse columns with their type of usage in the report, which can be display or filter) as rows and columns. Then the item similarity matrix \( S^{n \times n} \) is computed. After this preprocessing stage, the scores of recommended reporting options are calculated on-line as the product of the current user’s choice vector \( u^{1 \times n} \) and the similarity matrix \( S \).

We created the interaction matrix \( A \) using the statistics of the reports and the columns. Usage types of columns were taken into consideration as well (this is a step beyond the conventional user-based approach which is ignorant of the recommended items usage context). We use the number of distinct users who successfully executed a particular report as the interaction number \( a_{ij} \). Similarity between reporting options is calculated as the total number of their co-occurrences (interactions) inside reports. Then the rows of the resulting symmetric matrix \( S \) are normalized by the row sums in order to treat the resulting numbers as the posterior probabilities \( P_j(c_i|c_j) \) of appearance of column \( c_j \) given column \( c_i \). This is also different from the conventional approach. Normalization smooths effect of very popular columns with high frequencies. The resulting matrix \( S' \) is used as a potential score matrix to calculate recommendations. Table 2 contains an example of recommendations calculation.

A similar approach was used to implement report recommendations. Matrix \( A' \) is obtained from matrix \( A \) via normalizing rows by row sums and is used as a potential score matrix. It is multiplied by the user’s choice vector \( u \) in order to obtain the rank of items.

We calculate instant recommendations based on the user’s current choice that includes the reporting options that the user has selected up to now. Each time the user adds or removes a column from her report, the user’s choice vector

---

1http://www.amazon.com
2http://www.movielens.org
3http://news.google.com
4http://www.foursquare.com
5http://www.gogobot.com
6http://www.apple.com/apps
Table 2: Recommendation calculation example

Sample reports (column usage type omitted):

| R1: col1 | col2 | col3 |
| R2: col2 | col3 | col4 | col5 |
| R3: col3 | col5 |

Assuming every report executed once:

\[ A = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 2 & 1 & 1 \end{pmatrix} \]

\[ S = \begin{pmatrix} 1 & 0 & 2 & 1 & 1 \\ 1 & 2 & 0 & 1 & 2 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 2 & 1 & 0 \end{pmatrix} \]

\[ S' = \begin{pmatrix} 0 & 1/2 & 1/2 & 0 & 0 \\ 1/5 & 0 & 2/5 & 1/5 & 1/5 \\ 1/6 & 1/3 & 0 & 1/6 & 1/3 \\ 0 & 1/3 & 1/3 & 0 & 1/3 \\ 0 & 1/4 & 1/2 & 1/4 & 0 \end{pmatrix} \]

Recommendation scores when columns col2 and col3 are selected:

\[ S' \times \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1/2 \\ 2/5 \\ 1/3 \\ 2/3 \\ 3/4 \end{pmatrix} \]

Recommendation rank (with selected items omitted): col1, col3, col4

\[ u^{1 \times n} \] changes and we calculate new recommendations for her. In the initial stage, before the user has chosen anything, we show the most popular items based on the potential score.

3.4 Contextual search and results ranking

We adapted the tuple graph-based approach taken from the domain of keyword search in relational databases [12] for our implementation of contextual search. Data is spread across tables and the idea is to represent the data graph as a tuples graph, materialize it and do search in this structure. Yotta metadata is stored in an object repository [9]. Such a structure has not been previously addressed in the keyword search literature. The repository is a mesh of inter-related objects of many types held as a triple-store in MSSQL. The relevance of an object in a search depends not just on the properties of the object itself but on the properties of the object related to it (and of objects related to them). For example, reports have names and descriptions. They may also contain charts, which in turn contain axis items with names and descriptions. We address this issue by representing data as a tuple graph and materializing it. Then we create a conventional full-text index on the graph table and add the item potential score which represents the popularity of an item. After this preprocessing stage, search can be performed on-line. MSSQL Server tokenizes the text and creates an index of word stems allowing search for different forms of any word in the query. It ranks the results according to the BM25 ranking [10]. In MSSQL implementation it ranks higher those tuples that have high occurrences of terms from the search query. Sometimes this results in giving a lower rank to the tuples that contain the exact query as compared to the tuples that contain many occurrences of one word from the query. In order to address the text sparsity issue we make up the search results of three parts. The first one is the exact match query results ordered by the item potential score. The second part is “all words from the query” results, ordered by the fulltext index score [10] and the potential score. The third part is the “any-word from the query” search results, ordered by the fulltext index score and the potential score. The potential score allows ranking the items with an equal text score according to their popularity. This is very important when a query contains common terms.

4. EXPERIMENTS

In this section we present the results obtained by the recommendation algorithm on existing recommendation benchmarks as well as the analysis of the deployed Yotta system.

4.1 Movielens

We used the movie-ratings dataset from the Movielens project (http://www.grouplens.org/node/73) to benchmark our recommendation approach. We followed the experimental settings described in [8]. We used the one-million ratings dataset. It contains ratings on approximately 6,000 movies given by 3,500 users over three years. We ignored the value of the actual rating and treated it as a transaction. We considered only such users that watched from 5 to 100 movies. This gives us 1000 users and around 10600 transactions. We used 80% of the data as a training set and the remaining 20% as a test set. The split was made randomly. Three algorithms were implemented: item-based, user-based, and the proposed one. The number of recommendations provided by the algorithms was set to 10. We measured precision and recall averaged per user (i.e. macro-averaged). Table 3 contains the results.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item-based</td>
<td>0.0766</td>
<td>0.0811</td>
</tr>
<tr>
<td>User-based</td>
<td>0.0618</td>
<td>0.0615</td>
</tr>
<tr>
<td>Our approach</td>
<td>0.0657</td>
<td>0.0671</td>
</tr>
</tbody>
</table>

We observe that the item-based algorithm shows the best performance, which corresponds to the results in [8]. Our algorithm models similarity via set intersection which is less precise that the cosine similarity of an item-based approach. However, our algorithm is better than the user-based approach. Our advantage is in the use of a simple similarity function with a complexity of \(O(n)\). It is easier to implement in a database as compared to cosine similarity which is used in item-based and user-based approaches and which has a complexity of \(O(n^2)\). We can conclude that there is no need in a more sophisticated algorithm which is hard to implement in a database and which requires supporting outside of a database with up-to-date data.

4.2 Yahoo KDD Cup

We also tested our algorithm on the second track of the Yahoo KDD Cup challenge. The challenge contains user ratings of musical tracks on the scale from 0 to 100. There are 249012 users, 296111 tracks, and 61944406 ratings in the challenge. Six ratings from each of the 101172 users that rated
enough tracks are withheld and it is required to assess if the
ratings are not less than 80. It is known that in each of sets
of six ratings exactly three are not less than 80. In addition
to the described above data, there is information on what
artists, albums, and genres each track belongs to. The data
is anonymized and in all cases including genres indexes are
given instead of actual names. The performance of the rec-
ommender algorithm is measured as accuracy. We ignored
the album, artist and genre information and achieved 80% ac-
curacy on the second track of the Yahoo KDD Cup. This is
a mediocre result (the top performers are not surprisingly
combinations of tens of independent recommendation meth-
ods that give up to 96%) but the advantage of our approach
is simplicity and speed of the algorithm rather than per-
formance.

4.3 User Validation

We released the ranking and recommendations system
in October 2011 and the recommendation-based search fol-
lowed in the next release.

We utilized both qualitative and quantitative user feed-
back to assess the quality of our ranking and recommenda-
tions system. We surveyed the top 50 Yotta users who most
heavily utilized the report building subsystem. The tone
of the feedback was very positive and demonstrated a high
level of new features adoption.

Due to limited resources, the split testing was not done.
Instead, we collected the quantitative feedback by analyz-
ing logs of user interaction with the report creation subsys-
tem. Yotta stores only aggregate statistics of recommenda-
tions usage and the current user interface discourages users
from switching back to the recommendations once they have
started browsing for a selection. Thus, it is difficult to get
an accurate measure of ranking utility, i.e. how often a user
utilizes the suggested columns rather than browsing through
the column hierarchies. Consequently, we measure a poten-
tial (rather than actual) ranking utility by analyzing how
often the ranking suggestions would help in creation of ex-
isting reports. We express it as a ratio of the number of
times the suggestions list contains an appropriate column,
i.e. a column that would eventually go into the report being
created, to the total number of times column suggestions are
requested by a report builder. We also measure the average
position of selected options in ranking lists. It shows how
many items, on average, a user has to scan in the recommend-
atations list prior to selecting something from it. In the liter-
ature, this metrics is also known as search length (or search
length 1 to be more precise). This last data can be extracted
from Yotta logs and we provide the actual statistics. We did
not consider other common information-retrieval measures
like those used in [8], e.g. first 20 precision or discounted
cumulative gain, because our user interface recalculates rec-
ommendation when a column is added to a report, which
makes user behavior models behind those measures irrele-
vant in our case.

To evaluate our ranking algorithm, we measure it against
an alternative baseline — the suggestion of top 20 most
widely adopted columns. The potential utility of the base-
line was calculated in the same way as it was done for the
proposed ranking. The actual average position data for the
baseline was not available and we calculated it assuming that
the user would always select the best possible appropriate
option available in the suggestions list. The comparison with
the baseline shows that we recommend the correct columns
significantly more often than the baseline and that most of
the time we put the correct columns not lower in the list
than the baseline does (Table 4).

<table>
<thead>
<tr>
<th>Date</th>
<th>Report Count</th>
<th>Ranking utility</th>
<th>Ranking position</th>
<th>Baseline utility</th>
<th>Baseline position</th>
</tr>
</thead>
<tbody>
<tr>
<td>10/11</td>
<td>412</td>
<td>0.919</td>
<td>6.19</td>
<td>0.346</td>
<td>7.41</td>
</tr>
<tr>
<td>10/12</td>
<td>281</td>
<td>0.917</td>
<td>7.19</td>
<td>0.373</td>
<td>6.75</td>
</tr>
<tr>
<td>10/13</td>
<td>273</td>
<td>0.923</td>
<td>6.83</td>
<td>0.361</td>
<td>6.95</td>
</tr>
<tr>
<td>10/14</td>
<td>164</td>
<td>0.903</td>
<td>6.65</td>
<td>0.357</td>
<td>7.18</td>
</tr>
<tr>
<td>10/15</td>
<td>8</td>
<td>0.614</td>
<td>4.80</td>
<td>0.137</td>
<td>9.00</td>
</tr>
<tr>
<td>10/16</td>
<td>26</td>
<td>0.929</td>
<td>N/A</td>
<td>0.232</td>
<td>10.33</td>
</tr>
</tbody>
</table>

The custom report search functionality was added recently
and so far we have assessed only the average position of the
test selected from the search results by analyzing the logs
of the system. Table 5 contains the corresponding statistics.

<table>
<thead>
<tr>
<th>Date</th>
<th>Searches Count</th>
<th>Clicked result position</th>
</tr>
</thead>
<tbody>
<tr>
<td>03/12</td>
<td>5</td>
<td>4.20</td>
</tr>
<tr>
<td>03/13</td>
<td>3</td>
<td>1.00</td>
</tr>
<tr>
<td>03/14</td>
<td>1</td>
<td>14.00</td>
</tr>
<tr>
<td>03/15</td>
<td>8</td>
<td>8.25</td>
</tr>
<tr>
<td>03/16</td>
<td>3</td>
<td>8.33</td>
</tr>
<tr>
<td>03/19</td>
<td>32</td>
<td>3.16</td>
</tr>
<tr>
<td>03/20</td>
<td>11</td>
<td>5.00</td>
</tr>
</tbody>
</table>

So far we can see that the system is only getting attention
of users and it is hard to describe its behavior qualitatively.

5. CONCLUSIONS

In this paper we considered the problem of presenting
large data warehouse metadata in business intelligence ap-
lications. We demonstrated how application of personal-
ized recommendations and search helps to alleviate the re-
port creation task for the users in a real large enterprise
environment.

Our next priority task is to get more feedback on usage,
especially on the search functionality; our current statistics
are incomplete. In terms of functionality, the next step is
to increase the level to which recommendations are person-
alyzed for a specific user. We are segmenting the user base
on several dimensions (job function, geography, data access
rights, organizational affiliation, etc). The long term goal
is radically to simplify the Yotta user interface; we want to
be able consistently to predict what the user wants to do at
least 50% of the time. We consider this to be necessary if
we are to use mobile devices in BI applications.

6. REFERENCES

Recommender Systems Recommender Systems
Handbook. In F. Ricci, L. Rokach, B. Shapira, and
P. B. Kantor, editors, Recommender Systems
Boston, MA, 2011.


