University of Zurich^UZH

Department of Informatics

Robert Jan Stucki

SQL implementation of Singular Value Decomposition by Housholder transformation and QR formalization

December 2012

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

Contents

1 Introduction **3**

2 Implementation **5**

2.1 Correlation Computation 6

2.2 Householder Transformation 7

2.3 QR Algorithm 8

3 Experimental Evaluation **10**

3.1 Setting 10

3.2 Results 10

3.3 Interpretations 11

4 Summary & Conclusion **13**

Chapter 1

Introduction

This report is part of a Bachelor deepening task (Vertiefungsarbeit) done at the Database Technology Group of the University of Zurich under the direction of Prof. Dr. Michael Böhlen. Advisor and supervisor was Mourad Khayati who gave really helpful advice and also set the performing student thinking about the topic in-depth.

The aim of the task was to investigate and implement the Singular Value Decomposition (SVD) method using SQL queries. SVD is a matrix decomposition method that decomposes a matrix V into three matrices L, Σ and R^T. The product of the three matrices is equal to V. The SVD method is performed by using the Householder transformation and the QR factorization. This thesis shows the computation of Σ and R^T but does not include L.

Formally, a matrix $V = [V_0|V_1|...|V_{n-1}] \in R^{m \times n}$ can be decomposed into a product of three matrices as follows:

$$\mathbf{SVD(V) = L \times \Sigma \times R^T}$$

Additionally, it was also required to first compute the correlation of the underlying data. The correlation matrix was used then as input for V.

The assignment was divided into the following subtasks:

- Understand and implement the QR formalization and the householder transformation that perform SVD using PL/SQL
- Evaluate the scalability of the implemented decomposition on an Oracle server: horatio.ifi.uzh.ch (test datasets are available on the server)
- Empirical comparison of the running time between the SQL implementation and a main memory implementation provided by the supervisor
- Report of 5-10 pages
- Oral exam (approx. 25 min)

Furthermore, an important requirement of the task was to use as much standard SQL code as possible instead of procedural language code, i.e. PL/SQL in Oracle, which is dependent on the underlying Database Management System.

As a base for the implementation of the SVD computation, the paper "Efficient computation of PCA with SVD in SQL"[2] was used.

The time horizon of this project was two months from the 25th of September until the 1st of December 2012.

Chapter 2

Implementation

As already mentioned in the introduction, the implementation includes the three substeps Correlation Computation, Householder Transformation and the QR Algorithm. For clarity and independency use, the three steps were implemented in three separate PL/SQL procedures. If needed, they could easily be assembled to one continuous procedure.

Furthermore it was decided at the beginning of the project that the following data structure would be used for the tables in Oracle:

row_id	col_id	value
0	0	0.4845848845
0	1	0.4721235568
0	2	0.4612587456
0	3	0.3712455255
...	...	...

Table 2.1: Table structure used in Oracle Database

There is one column used for the row index and another one for the column index. Both together build the primary key of the table. In the third column is the basic value stored. One advantage of this structure is that it is very flexible as it can store any 2D matrix. This is especially an asset for the given test datasets where the number of given observations and time series can vary depending on what data the user wants to examine. Furthermore, there is no limit in number of columns or rows for the input matrix. All DBMS have a limit in number of columns. So, the used data structure cannot run out of available columns. In the paper[2], they did use SQL script generators because some of the tried implementations were optimized on a fix number of columns. This step is not needed for the chosen data structure as the number of columns in the tables is always 3.

Table 2.2 shows the main symbols used in this report and their definitions.

Position	Symbol	Definition
General	L	Left eigen vectors matrix (table)
	Σ	Eigen values matrix (table)
	R	Right eigen vectors matrix
Correlation	X	Input/start table
	Q	Helper table with summarizations for correlation computation
	L	Helper table with summarizations for correlation computation
Householder	I	Identity matrix
	P	Rotation matrix (table) needed for Householder Transformation
	V	Table of correlation matrix
QR algorithm	Q	Helper table for QR algorithm
	R	Helper table for QR algorithm
	U	Helper table for computing L

Table 2.2: Table of notations

2.1 Correlation Computation

The main focus of ths assigment thesis is not set on the computation of the Pearson correlation. But it is required to get from the provided datasets to the starting position for the actual SVD calculation. Therefore it is considered as a relevant part of the task.

Before the correlation coefficients can be computed, the summarization vector L and the summarization matrix Q have to be worked out from the input matrix X. Once they are prepared, the correlation matrix V can be computed. The formulas therefore are visible in Figure 2.1.

$$\boxed{\begin{gathered} L_i = \sum x_i \\[1em] Q_{ij} = \sum x_i x_j^T \\[1em] V_{ij} = \frac{nQ_{ij} - L_i L_j}{\sqrt{nQ_{ii} - L_i^2}\sqrt{nQ_{jj} - L_j^2}} \end{gathered}}$$

Figure 2.1: Formulas for correlation computation[2]

In the PL/SQL procedure, the computation is implemented with normal insert/delete statements applying the abovementioned formulas. Extracts of the source code with explanations are left out in this report in respect of the minor importance towards the computation of the Singular Value Decomposition.

2.2 Householder Transformation

The goal of Householder's algorithm (c.f. Algorithm 1) is to transform the symmetric matrix V_0 into a similar symmetric tridiagonal matrix V_{n-2}. A symmetric tridiagonal matrix is a matrix that its nonzero elements are found only on the diagonal, subdiagonal and superdiagonal of the matrix and its subdiagonal elements and superdiagonal elements are equal.

The Householder Transformation [4] is the first step to compute the SVD, starting with the quadratic correlation matrix V_0 as input. The Householder Transformation is done in n - 2 steps where V is a n x n matrix. A rotation P_k is generated to obtain $V_k = P_k A_{k-1} P_k$, with $k = 1, 2, .., n-2$. At each step there is also computed $U_k = U_{k-1} P_k$ with $U_0 = I$, where I is the identity matrix.

```
   input  : Square correlation matrix $V_0$
   output: $V_k$ and $U_k$ as input for the QR algorithm
 1 for $k$ = $0$ to $n$-$3$ do
 2     $\alpha$ = -sign$(v_{k+1,k})\sqrt{\sum_{j=k+1}^{n-1}(v_{jk})^2}$
 3     $r = \sqrt{\frac{1}{2}\alpha^2 - \frac{1}{2}\alpha v_{k+1,k}}$
 4     $w_0 = w_1 = w_2 = ... = w_{n-1} = 0$
 5     $w_{k+1} = \frac{v_{k+1,k}-\alpha}{2r}$
 6     for $j$ = $k$+$2$ to $n$-$1$ do
 7         $w_j = \frac{v_{jk}}{2r}$
 8     end
 9     $P_k = I - 2ww^T$
10     $V_{k+1} = P_k V_k P_k$
11     $U_k = U_{k-1} P_k$
12 end
```

Algorithm 1: Householder transformation[4]

The computation of α, w and r in SQL is rather straight forward and does not need any extra explanation. Whereas the computation of V_{k+1} could be implemented in different ways. It was decided to multiply first $V_{k-1} * P_k$ and after that in a second step $P_k * M_k$ with $M_k = V_{k-1} * P_k$. The implementation in SQL was done as follows:

```
--compute A(k)= P * A * P

  --compute A * P = a2

delete t_a2;

insert into t_a2(
select tableH.row_id, tableH.col_id, sum(tableH2.val * tableP2.val)
from t_h_full tableH2, t_p_full tableP2, t_h_full tableH
where tableH2.row_id = tableH.row_id
and tableH2.col_id = tableP2.row_id
and tableP2.col_id = tableH.col_id
group by tableH.row_id, tableH.col_id
);

commit;

  --compute P * (a2) = A(k)

  ...
```

2.3 QR Algorithm

During each iteration of the QR algorithm, there is a QR factorization, two matrix multiplications and an error computation[2]. The QR factorization is a substep of the complete QR algorithm. It is explained in Algorithm 3. The iterations are executed until a given number of iterations is reached or better the computed error converges towards a given error threshold. In the implementation both break conditions were realized. For comparable tests it makes more sense to use the same number of iterations whereas it is more reasonable in real applications to use an error criteria.

```
input : Symmetric tridiagonal matrix $V$, matrix $U$, error tolerance $\epsilon$
output: Eigenvectors matrix $L$ and Eigenvalues matrix $\Sigma^2$
1  do
2      compute $Q_k$ and $R_k$ which fulfill $V_{k-1} = Q_k R_k$ (QR factorization)
3      $L_k = L_{k-1} Q_k$ with $L_0 = U$
4      $V_k = R_k Q_k$
5  while $(error > \epsilon)$;
6  forall the elements $e$ of $L$ do
7      $e = abs(e)$
8  end
9  forall the elements $e$ of $V$ do
10     $e = abs(e)$
11 end
12 $\Sigma^2 = diag(V)$
```

Algorithm 2: Pseudo code of QR algorithm[2]

The pseudo code in Algorithm 2 shows the basic idea how Σ^2 and L are computed through the QR algorithm.

```
input  : Symmetric tridiagonal matrix $V = [v_0\ v_1\ ...\ v_n]$
output: Matrix $Q$ and matrix $R$
1 $Q = V$, $Q = [q_0\ q_1\ ...\ q_n]$
2 for $i$ = 0 to n-1 do
3     $r_{ii} = \|q_i\|$
4     $q_i = q_i / r_{ii}$
5     for $j$ = i+1 to n-1 do
6         $r_{ij} = q_i * q_j$
7         $q_j = qj - r_{ij} * q_i$
8     end
9 end
```

Algorithm 3: Computation of Q and R by the QR factorization[4]

The following SQL source code snippet shows the implementation of the QR factorization step of the QR algorithm. Two for loops are needed for the computation. The first loop is iterating over all rows and the second one over the rows beginning at i + 1.

```
--compute Q and R
for j in 0..(n-1)
loop

insert into t_r(select j, j, sqrt(sum(Power(val, 2)))
from t_q_qr where col_id = j);
commit;

update t_q_qr set val = val /
(select val from t_r where col_id = j and row_id = j)
where col_id = j ;
commit;

for i in (j+1)..(n-1)
loop
insert into t_r(select j, i, sum(tableQ1.val * tableQ2.val)
from t_q_qr tableQ1, t_q_qr tableQ2
where tableQ1.row_id = tableQ2.row_id
and tableQ1.col_id = j and tableQ2.col_id = i);
commit;

update t_q_qr tableQ1 set val = val -
(select val from t_r where col_id = i and row_id = j) *
(select val from t_q_qr tableQ where tableQ.col_id = j
and tableQ.row_id = tableQ1.row_id)
where tableQ1.col_id = i;
commit;

end loop; end loop;
```

Chapter 3

Experimental Evaluation

An important part of this work was to evaluate the scalability of the implemented decomposition on an Oracle server. Secondly, the SQL implementation should be compared by running time with a main memory implementation, which was provided by the supervisor. Additionaly to these tasks, the results of the SQL implementation were compared with a similar SQL implementation in the underlying paper[2].

3.1 Setting

The source code of the main memory implementation[3] was provided in C++ but was converted into Java to be able to use JDBC. The computation of the correlation matrix was implemented with a helper class which can compute the correlation of two vectors. The source code of this helper class in Java was also provided by the supervisor and was slightly adapted to the requirements of the task.

It was used a HP EliteBook 8560p, Intel i7 2.70 GHz, 8GB RAM, 500GB disk, for running the test samples of the main memory implementation. Whereas the Oracle server for running the SQL implementation did have the following specifications: QEMU Virtual CPU version 0.12.5, 2.30 GHz, 4GB RAM, 1Gbit disk. And in the mentionned paper[2], they did use a Intel Duo Core CPU, 2.6 GHz, 4GB RAM, 1TB disk for the SQL experiments and a workstation with a 1.6 GHz CPU, 256MB RAM, 40GB disk for the Java tests.

3.2 Results

First of all, it was checked for reference reasons how long insert/delete transactions take time on the provided server which the SQL implementation is tested on. The results of this test are visible in Table 3.1. It shows that an insert transaction takes almost double of the time a delete transaction takes.

Quantity	Insert [sec]	Delete [sec]	Inserts/sec	Deletes/sec
1’000’000	72	39	0.000072	0.000039
2’000’000	163	76	0.000082	0.000038

Table 3.1: Duration of insert/delete transaction on Oracle server

Table 3.2 shows the performed experiments comparing the own SQL implementation against the Java main memory implementation. The results show that the SQL implementation is massively slower than the Java implementation. For the given sizes of the input matrix, the Java implementation did not throw any <out of memory> exception.

Size		Correlation [sec]		SVD [sec]			
n	d	SQL	Java	SQL HH	SQL QR	SQL Total	Java Total
10000	30	5	0.3	6	102	108	0.014
10000	50	14	0.6	31	438	469	0.016
10000	100	52	2.2	437	3066	3503	0.031
15000	30	7	0.3	7	104	111	0.015
15000	50	19	0.9	30	435	465	0.016
30000	30	14	0.6	6	103	109	0.015
30000	50	42	1.6	29	418	447	0.016
50000	50	75	2.6	31	420	451	0.016

Table 3.2: Comparison of SQL implementation with main memory implementation

A second comparison was made between the SQL implementation and the SQL implementation of the underlying paper[2]. The results are visible in Table 3.3. It seems like the own implementation of the correlation computation is faster whereas the SVD computation is slower, but this would have to be statistically proven. The performed tests can only give indications.

Size		Correlation		SVD					
n	d	own[s]	paper[s]	own[s]	error	iter.	paper[s]	error	iter.
10000	30	5	7	108	0.00007	100	92	0.00376	78
10000	50	14	20	469	0.0006	100	481	0.00365	146

Table 3.3: Comparison of SQL implementation with SQL implementation of the paper[2]

3.3 Interpretations

The results in Section 3.2 look extraordinarily at a first glance. But after a little bit of reflection they are reasonable.

That the Java implementation is that much faster than the SQL implementation was not expected. There might be several reasons which can explain this result. First of all, the tests of the SQL code and the Java code were not run on the same computer. The used computer for the Java code did have a better processor, more RAM and also more disk space. Especially the additional RAM and the better processor may have led to a better performance of the tests.

While the SQL code for the correlation computation and the Java Code use more or less the same algorithm, the used algorithms for the Singular Value Decomposition differ strongly. Especially in the number of iterations does the Java code strongly decrease the computing time.

Another point which might explain a large slice of the time differences are the insert, update and delete transactions which are used for the computations in SQL. Even read transactions might take longer on the SQL server because in most cases the data has first to be loaded from disk into main memory. Table 3.1 shows how long insert and delete transactions without any computations or queries can take on the provided Oracle server.

It was also tried to improve the performance of the SQL implementation by using truncate-commands instead of deletes, but this measure did not bring any time saving, rather to the contrary. Also the use of views instead of inserting the tuples into a table did not provide any economy of time. It seems that the Oracle DBMS already optimizes the SQL code by itself.

In the experiments of the paper[2], the tests with the Java library JAMA[1] did lead to <out of memory> exceptions. The biggest possible combination for the input matrix was 100'000 x 30 and the tests were run on a workstation with 256MB RAM only. Therefore it does not surprise that the new tests with the provided C++ code[3] on a computer with 8GB RAM do not throw an exception. Considering only the space needed for storing the input matrix in main memory it would have need of 1'073'741'824 double values to fill the 8GB of RAM and create an exception. This quantity will never be reached by the provided data set with a maximum size of 11'353'335 values. Also paying attention to other RAM consumers of the program would not change this fact.

The second comparison between the own SQL implementation and the one of the paper[2] does not show a significant discrepancy. Also both used computers for the tests have approachingly identical specifications. To make a more meaningful statement about the comparison, more tests would have to be performed and also the source codes would have to be compared. Unfortunately, the paper[2] does not provide any source code.

Chapter 4

Summary & Conclusion

It turned out that this project was a really challenging and ambitious task for a short time period of two months. After reading literature and trying to understand the underlying mathematical algorithms, the implementation of these algorithms in SQL took quite a while. The limitation of using as much SQL as possible did not make the realization easier.

The next step was to analyze the written SQL code and try to improve it concerning time performance. Towards the end of the project, the implemented SQL code was compared with a Java implementation and the results of the underlying paper[2].

It was not expected that the SQL code would take that much longer than the main memory implementation, but after some hours of thinking it became comprehensible. It could also be extracted from the results that the main advantage of the SQL implementation, which is no memory limitation, never gets a chance for the given datasets and computer hardware. It is also uncertain if the SQL implementation would scale for such a large dataset as the QR algorithm has a complexity of O(sn^3) whereas s is the number of iterations for convergence and n the number of rows/columns of the correlation matrix.

The short time period assigned to the task does leave some extensions open. For example, it would be interesting to compare the implemented SQL code with the SQL code of the paper[2]. On both sides there might be some optimizations possible as the paper does not provide to much information about how they did implement it in SQL. Another uncleared point is how user defined function in PL/SQL would perfom against the standard SQL code. This was not part of the task, but it might still be interesting to examine it further.

Befor ending the assignment with an oral exam, this report had to be written. Overall, it was a successful and educationally valuable project, but on the other hand also very time consuming and demanding.

References

[1] JAMA Java Matrix Package, `http://math.nist.gov/javanumerics/jama/`, 2012

[2] Navas M., Ordonez C., Efficient computation of PCA with SVD in SQL, in DMMT, 2009

[3] Numerical Recipes Software, Webnote No. 2, Rev.1, `http://www.nr.com/webnotes/nr3web2.pdf`, 2012

[4] Salleh S., Zomaya A., Bakar S., Computing for numerical methods using Visual C++, 2008