
Machine Learning on Large Databases: Transforming Hidden Markov Models to SQL Statements

Dennis Marten, Andreas Heuer

Institute of Computer Science, Rostock University, Albert-Einstein-Strae 22, 18059 Rostock, Germany,
{dm, heuer}@informatik.uni-rostock.de

ABSTRACT

Machine Learning is a research field with substantial relevance for many applications in different areas. Because of technical improvements in sensor technology, its value for real life applications has even increased within the last years. Nowadays, it is possible to gather massive amounts of data at any time with comparatively little costs. While this availability of data could be used to develop complex models, its implementation is often narrowed because of limitations in computing power. In order to overcome performance problems, developers have several options, such as improving their hardware, optimizing their code, or use parallelization techniques like the MapReduce framework. Anyhow, these options might be too cost intensive, not suitable, or even too time expensive to learn and realize. Following the premise that developers usually are not SQL experts we would like to discuss another approach in this paper: using transparent database support for Big Data Analytics. Our aim is to automatically transform Machine Learning algorithms to parallel SQL database systems. In this paper, we especially show how a Hidden Markov Model, given in the analytics language R, can be transformed to a sequence of SQL statements. These SQL statements will be the basis for a (inter-operator and intra-operator) parallel execution on parallel DBMS as a second step of our research, not being part of this paper.

TYPE OF PAPER AND KEYWORDS

Regular research paper: *Machine Learning, Hidden Markov Model, Big Data Analytics, Database Support, SQL, R language*

1 INTRODUCTION

Assistive or smart systems are and will be a great part of our everyday lives. It is not surprising that in recent years researchers, especially in the areas of Big Data Analytics and Artificial Intelligence, have been putting a lot of effort into the development of assistive systems [35]. The basis for such systems is a large set of sensors providing data of the context, the situation, and the activities of those persons that want to use assistance functions. These sensors can be part

of the Internet of Things (IoT) and can be available in apartments, cars, mobile phones, coffee machines, home media equipment, as well as glasses and watches [39]. By analyzing these sensor data with Machine Learning (ML) algorithms [3], the developers of assistive systems can on the one hand derive information about the current situation which the user is in, and the current activity which the user is doing, and on the other hand also predict the user's activities in the near future.

Deriving and predicting situations and activities with ML techniques can be done with model-based or data-

driven techniques. At the University of Rostock, one research group in the Department of Computer Science concentrates on the data-driven aspect when researching and developing methods to detect and predict user activities [25, 45]. In this case, the researchers and developers need a small amount of probands or test persons (from 10 to 100), collecting sensor data for a short time period (maybe some weeks), annotate these sensor data with activity information by experts knowing what was going on during these weeks, and then trying to learn the activity models by means of ML algorithms. In this learning phase, one has a small amount of probands, but a large amount of sensors and a high frequency in the sensor data (e.g., measuring data every milisecond).

To support this data-driven development of assistive systems, one of the most challenging problems is usually the handling of enormous amounts of data, normally some Terabytes for the ML learning phase. However, due to the massive amounts of data, conventional statistical and ML tools suffer from performance issues. Therefore, supporting ML tools by database technology is one of the most challenging research directions in computer science, especially in information systems [1].

In [32], we already introduced and discussed a framework that combines the popular statistical development tool R, database technology and the widely known MapReduce framework. R was used in this approach since the ML algorithms of assistive systems have been implemented in R. In this paper, we want to show why and how one can transform the ML algorithms into SQL, also called *intergalactic data speak*. Though SQL seems to be old-fashioned for modern Big Data Analytics problems, it is shown that transforming ML algorithms to SQL is not only possible, but has a lot of advantages in

- using parallel database techniques to improve the performance of the ML algorithms, without having the need to design and implement the distribution by hand,
- using privacy-by-design construction techniques to automatically push (parts of the) analytics functions closer to the sensors preserving the privacy of the user of the assistive systems, and
- using data provenance (or data lineage) techniques to tell the developer of the assistive systems which of the thousands of sensors in the IoT environment and which frequency of measuring, aggregating, and transmitting these sensor data are really necessary to detect and predict situations and activities.

To support automatic parallelization, privacy-by-design construction, and data provenance, we can use

fundamental database research results after having transformed the ML algorithms to SQL statements.

The aim of this paper is to show how a specific ML algorithm (the Hidden Markov Model (HMM) [34], introduced in Section 6) can be transformed to SQL statements (Section 7), and how efficient this solution is for a centralized (non-parallel) database system on a computer with one processor and one disk (Section 8). Before analyzing the HMM, we will discuss what kind of subroutines can be supported on a database in a natural way (Section 5). The paper starts with the State of the Art (Section 2), where we focus on approaches to implementing ML algorithms with R and Big Data Analytics environments. The overall structure and aim of the PARADISE project is presented in Section 3, and the system architecture of the parallel ML engine based on a SQL DBMS in Section 4.

2 STATE OF THE ART

To understand the importance of processing Machine Learning algorithms via SQL it is useful to give a quick overview of several projects that are designated to process ML algorithms on Big Data.

Machine Learning meets Databases

In [19] one can find a concise summary on Machine Learning, its recent rise and its link to the research area of databases. Furthermore, existing projects and criteria for successful future projects are discussed. They present several extensions of relational database systems that allow to process Machine Learning algorithms via external user defined (UDFs) functions or extensions of their SQL dialect. Additional information regarding Machine Learning and databases can be found in [28] where technical aspects and open problems like compression, scan sharing, query generation, and others are presented.

Machine Learning and Big-Data-Analytics Environments

When computing Machine Learning algorithms on Big Data, one has to think about massively parallel computation techniques as introduced in the area of Big Data Analytics. This field is dominated by projects that are built on the MapReduce framework. Perhaps the most popular one is Apache Hadoop [44] introducing the well-known Hadoop File Systems (HDFS) and offering transparent compression and support for unstructured and structured file formats like CSV or JSON. Additionally, several other Apache projects are based on Hadoop. Regarding Machine

Learning and database technology the most interesting ones are Apache Hive [22], Apache Flink [6] and Apache Spark [46].

Apache Hive introduces an SQL-like language called HiveSQL and a procedural language called HPL-SQL to provide a pleasant way to access, process, and manipulate data. It supports larger parts of the SQL standard functionality and also introduces database techniques like indexes [22].

Apache Flink extends Hadoop-like processing of data with different stages of optimization, which are well-known in the database community. The optimization and the comparatively pleasant API makes Flink a good and promising choice for Big Data Analytics [6]. Furthermore, several interesting research projects can be found using Apache Flink. For example, the project Gilbert [37] combines a Matlab-like language with Apache Flink and an optimizer for sparse linear algebra operations.

Apache Spark provides four different main libraries supporting SQL, streaming, Machine Learning, and graph functionalities. It is also possible to write code in different languages like Java, Scala, Python, or R. Spark uses an advanced directed acyclic graph engine that supports in-memory processing, and can therefore outperform Hadoop in several Machine Learning applications [46].

Apart from Hadoop there exist other promising projects that are built on the MapReduce framework. One of the most noteworthy is IBM's **System ML** [4, 12] that can also run on Apache Spark. It provides an R-like language to express Machine Learning algorithms improving the usability for data scientists. Furthermore, SystemML provides multiple possible processing plans for single base operations like matrix multiplication, runtime optimization, data compression, and other techniques [11].

Coupling of R and SQL

Since this paper considers possible translations from R to SQL, the following discussion presents projects directly addressing the connection between database technology and R.

One of the most obvious possibilities to connect R and a database system is to use an R package that establishes a connection via a **JDBC driver**. Many of the popular systems, like PostgreSQL [33], MySQL [38], MonetDB [23], and others even have their own R-JDBC-Package offering some minor additional features. However, their main purpose is to establish an API for queries in R. Since we assume that the R developer is not an expert in using SQL, this approach does not fit our requirements.

MonetDB's R-Integration [24] is making R a first

class citizen of the database system. This offers the user the possibility to use R code on the database within SQL. The communication between both instances is working without actually sending the data through the network. This is crucial since network costs are one of the main bottlenecks in many Big Data problems. However, this approach still lacks transparency and is more suited for database experts using R.

SciDB [43] is a parallel database system with an array data model, which the authors claim to work faster for ML problems than classical relational database systems, due to its better compatibility to multi-dimensional problems. SciDB offers an R Integration version, in which it is possible to map database variables from R to the database. This R-API offers a somewhat transparent R-SQL connection for operations like matrix multiplication using few extra annotations in the R script. Anyhow, there are currently no papers available that elaborate on the way this connection works. Also, since this database works with arrays instead of tuples, it is not suitable for SQL analysis.

Other research projects like **RIOT-DB** of Duke University [47] have investigated whether relational database systems (in this case MySQL) are suitable for scientific calculations. This project extends R with new datatypes that map to data on the database. The mapping allows RIOT-DB to work transparently on the database rather than R. However, the researchers mention performance problems while working with matrices. This may have several reasons, starting by using the row-store MySQL as a backend system.

Other approaches

There are several other projects that are worth to study, like the highly developed LINQ [31], Database Supported Haskell [14] and many other.

While many projects share some of the criteria that we also want to address in our framework, all of these projects do not combine R as a source language, (parallel) DBMS technology as a computation platform, general parallel processing techniques, and SQL as the target language as we intend to do.

But why stick to DBMS technology and SQL as the target language for Machine Learning problems? As one will see in the next section, translating ML to SQL will give us the opportunity to use fundamental results of database research to design a support system for the privacy-aware and development of efficient and highly parallel assistive systems.

3 THE PARADISE PROJECT

As one of our current research projects, we are developing the PArADISE¹ framework. This framework aims at supporting developers of assistive systems in three development phases. In Figure 1, these phases are shown as Development (left-hand side), Data and Dimension Reduction (depicted by the arrow in the middle), and Usage (right-hand side):

- **Development:** ML Developers and Data Scientist are trying to detect and predict user activities, using data from a small amount of test persons, collecting sensor data for a short time period (maybe some weeks), annotating these sensor data with activity information, and then trying to learn the activity models by means of ML algorithms.
- **Data and Dimension Reduction:** In the development phase, there is a small amount of probands, but a large amount of sensors and a high frequency in the sensor data. After having derived the activity and intention models, one has to reduce the dimensions of the data (e.g., the number of sensors being evaluated) and the data itself (e.g., measuring and transmitting sensor data every minute instead of every milisecond). To derive the most important dimensions and data, we adapt well-known techniques of data provenance [21] and data reduction [20].
- **Usage:** When using the assistive system afterwards for a huge number of clients (millions of clients having billions of sensors) with the reduced set of sensor data, one has to decompose the SQL queries detecting the activities and intentions of the users. This query decomposition aims at better performance because the query will be vertically pushed down to the sources of the data (the sensors) as close as possible. Even more importantly, the decomposition of the query results in better privacy for the user of the assistive systems, since most of the original sensor data has not to leave his personal equipment, his apartment, or his car. Only a *remainder query*, the part of the query that cannot be pushed down to the clients and sensors, has to be evaluated on the large cluster computers of the provider of the assistive system. For the query decomposition, we adapt query containment techniques [7] and the principle of *Answering Queries using Views* (AQuV) [9, 30] to our case of *Answering Queries Using Restricted Capabilities of Query Processors* [17, 18].

In this context, it is assumed that the provider of the globally distributed system is called *Poodle* [16]. Poodle uses ML development tools such as R or higher-level languages to derive the activities and intentions of the user. This ML code will then be transformed to a sequence of SQL statements. These SQL statements will then be evaluated in parallel on a large computer cluster, and the parallelization will be introduced by the PArADISE system. This phase is called ML2PSQL in Figure 1. The parallel database environment will later also be used in the usage phase, to implement the evaluation of the remainder queries that cannot be precalculated by the lower levels of the vertically distributed system. In the area of cloud computing, this computing on lower levels, being the sensors and the ensemble of the local equipment of the user, is called *fog, edge, or dew computing* [18, 40, 41]). While fog computing supports IoT applications with real-time requirements, dew computing aims at pushing applications, data, and low level services away from centralized virtual nodes to the end users.

Before going into details of the ML2PSQL transformation process (being the the main focus of this paper), we conclude the PArADISE overview with some aspects of the query decomposition, aiming at the users' privacy. In [15], we present a simple XML schema in which the user can formulate his privacy claims towards the assistive system. The user can specify for each function, which information, in what level of detail, is forwarded to Poodle, the provider of the system. For this purpose, individual attributes can be summarized to attribute combinations which are allowed to be queried by the system.

Even for an experienced user it is difficult to set up his privacy preferences, e.g. which data are worth to be protected. For the identification of sensitive data so-called quasi-identifiers [8] are used and computed by the PArADISE system.

To be able to automatically decide about the privacy-oriented decomposition of the queries, we have to use SQL queries as a basis for query containment and Answering-Queries-using-Views techniques. Hence, it is crucial for this approach to be able to express ML code by a sequence of SQL queries in the development phase of the system. Only then, one can use the privacy-by-design principle when constructing the evaluation algorithms in the usage phase.

In the next section, the components of the ML2PSQL transformer will be discussed.

¹ Privacy-aware assistive distributed information system environment

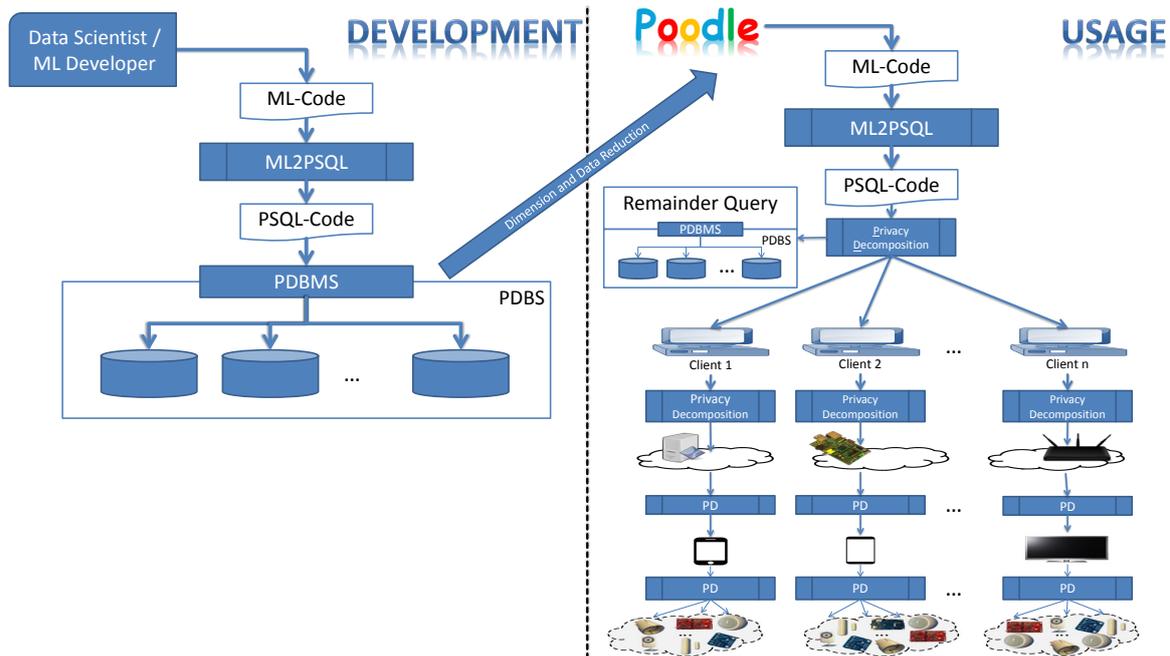


Figure 1: The PARADISE framework: privacy-aware support for assistive systems

4 SYSTEM ENVIRONMENT

In this section, more details of our development phase (left part of Figure 1) will be presented. In [32], we already introduced a general framework of processing ML algorithms implemented in R by means of SQL DBMS. As a first DBMS, the main-memory system MonetDB was used which turned out to be one of the best suited systems for analytics purposes. The ML algorithms have been splitted into SQL parts and additional computations in R.

In the revised architecture of the ML2PSQL transformer presented in Figure 2, we will stick to SQL as the target language, but decided to be independent of any particular DBMS such as MonetDB. More specifically, besides using main-memory DBMS for small data sets, we also want to use parallel DBMS for Big Data Analytics tasks, consisting of Terabytes of sensor data to be analyzed.

The new ML to parallel SQL transformation environment shown in Figure 2 translates the ML Code to SQL statements first. Parts of the ML Code, which cannot be transformed to database queries will be handled separately in a post-processor step. Afterwards, the SQL statements are parallelized onto n nodes of the computer cluster serving as a hardware basis for computation. For parallelization, techniques provided by parallel DBMSs such as Postgres-XL are used, or our own fragmentation and parallelization algorithms are developed, specifically for cases of intra-

operator parallelism not provided by most of the parallel relational DBMSs.

In the remainder of this paper, the transformation step from ML algorithms implemented in R to a sequence of SQL statements will be presented in more detail. These transformations are explained in detail using the Hidden Markov Model as a special example of an ML algorithm. The techniques are applied to a special assistive systems that was developed at the University of Rostock. This assistive systems aimed at assistive support in a meeting scenario. The Hidden Markov Model and the meeting scenario is introduced in Section 6. Before that, relational schemes and SQL statements are used for basic operations of Scientific Computing, such as matrices and matrix multiplication in Section 5.

5 ML-INTO-SQL TRANSLATION PROCESS

In this section, possible SQL solutions for basic operations from the area of scientific computations will be discussed. Since we introduce an experiment written in the well-known statistical language R, R-code will be used as a basis for the analysis. Furthermore, we would like to state that the long time aim is to develop a purely mathematical formula language (matrix equations) to minimize language restrictions and make the programming process for the researcher as easy as possible.

Before starting to analyze methods it is worth noting

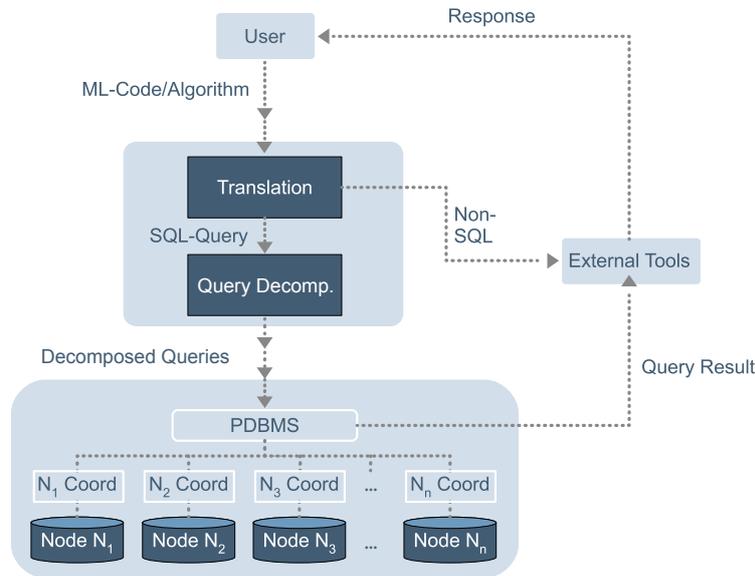


Figure 2: The ML to parallel SQL transformation environment

on how R objects can be translated into SQL types and schemes. Firstly, all R data types following the IEC 60559 (IEEE 754) standard are also supported in most of the common database systems. Therefore, one can assure that partial processing on the database system can be done without loss of information. With respect to R objects, we will concentrate on vectors and matrices since they are the most important when regarding statistical analysis. Here it is important to state that it is possible in R to name columns and rows and even select entries by addressing them:

```
A <- matrix(1:4, nrow = 2)
colnames(A) <- c('Col1', 'Col2')
rownames(A) <- c('Row1', 'Row2')
A['Row2', 'Col1']
[1] 2.
```

This will ultimately lead to a matrix scheme with data type T in the form of

```
A( i int,
    j int,
    rownames varchar,
    colnames varchar,
    v T)
```

which basically follows the well known scheme for storing sparse matrices in non-database systems. In an analogous manner a vector scheme can be defined as

```
V( i int,
    names varchar,
    v T)
```

If possible it would surely be beneficial to map the names to the indices and therefore make a storing of names for calculations obsolete. However, this might not work in every case, especially not in interactive programming. Anyhow, names will not be used in the upcoming analysis, which will shorten our scheme to

```
A( i int,
    j int,
    v double)
```

and

```
V( i int,
    v double).
```

Finally, it has to be mentioned that this approach may suffer performance disadvantages regarding calculations on small datasets when facing optimized matrix operations (for instance the LAPACK library [2]) that can be done entirely in main memory. But this methodology is supposed to work on big data sets, where database systems can surpass common in-memory environments, due to their optimized data management with respect to secondary storage besides their logical and physical optimization capabilities.

5.1 Basic Operations

Now, basic operations and translation possibilities are presented. After this, a Hidden Markov Model will be introduced and an analysis of its subroutines will be provided.

While simple discrete algebraic operations like $+$, $-$, \cdot , $/$ can be naturally implemented via SQL we would like to distinguish two cases for the addition and the subtraction. Here, the decisive factor is whether zero-values should be explicitly stored or be neglected like it is common in sparse-matrix-algorithms. If zero-values are stored one can simply use an inner join to calculate a discrete operator (here $+$) as

```
select A.i, A.j, A.v+B.v as v
from A join B on A.i = B.i and
      A.j = B.j
```

Anyhow, if zero-values are missing, the inner join will neglect value pairs where at least one of the values is zero and therefore returns an incorrect result. There are multiple ways to work around this problem, depending on the database system one uses. A simple solution is to use an outer join, which would return the **NULL**-value on the aforementioned pairs. If possible, one should set default values for the **NULL** case. Unfortunately, this is not possible in many systems but overwriting the method could be an option. Another possibility, with the down side of extra operations, is to replace **NULL** values with the value 0. As an example we use the **isNULL(v) then else end** syntax for the example above:

```
select A.i, A.j,
       isNULL(A.v)
       then 0.0
       else A.v end
+ isNULL(B.v)
  then 0.0
  else B.v end
from A outer join B on A.i = B.i
and A.j = B.j
```

It is also possible to calculate the inner join query and afterwards insert the tuple values without corresponding match into the result table. But, due to additional scans of both matrix tables, it is advisable to use the outer-join-method. On a side note: handling “divided-by 0” tuples can be dealt with analogously by catching **NULL** values in the second matrix.

Overall, these approaches will certainly need more processing time than the inner join method using dense matrices. On the other hand, if one tries to process big and sparse matrices, which are used in many applications (for instance PageRank [5], Finite Element Method

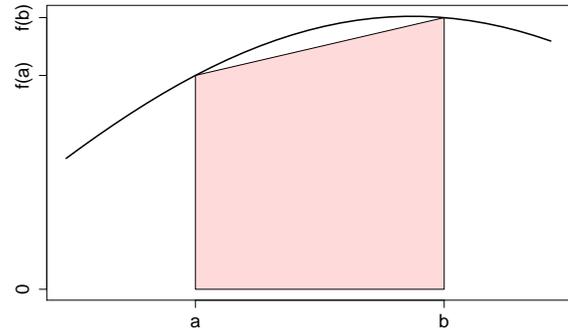


Figure 3: Graphical illustration of the trapezoidal rule for approximative integral calculation

[36], etc.), this method will heavily outperform the first one and additionally save storage. It is also worth noting that other fundamental basic operations, like matrix multiplication (see Section 5.2), do not need extra workarounds for their sparse approach, so that the user can fully benefit from the performance speed-up without any extra effort.

For the sake of simplicity we will avoid using the outer-join scenario from this point on and will continue now by giving a small example of discrete algebraic operations. As a relevant real-life example for discrete operations we would like to present the numerical calculation of integral values with respect to time (or basically any variable) following the trapezoidal rule (see Figure 3)

$$\int_a^b f(t) dt \approx (b-a) \frac{f(a) + f(b)}{2}$$

with individual time step size $b - a$. The corresponding SQL query could be written as

```
select Vf.i, sum((Vf.v + V1.v) / 2)
from V Vf join V V1
on Vf.i-1 = V1.i
```

where the time step size is assumed to be 1 in any step for convenience. Note that the sum function is used to accumulate the partial integrals to form the overall integral. With very little adjustment it is also possible to calculate derivations or even the jacobi matrix from a series of observations.

5.2 Aggregation and Matrix Multiplication

Similarly to the basic algebraic operations, aggregations can naturally be found in a very high amount of statistical methods. Since especially the sum function is used frequently in scientific analysis, for example in scalar products or vector and matrix norms, we will now

translate the *frobenius norm* of a matrix $A = (a_{ij})_{ij} \in \mathbb{R}^{n \times m}$ which is defined as

$$\|A\|_F = \sqrt{\sum_{i=1}^n \sum_{j=1}^m a_{ij}^2}.$$

In this case the matrix scheme

```
A(i int, j int, v double)
```

is very useful especially in column stores because every element has to be used with itself and can therefore be accessed sequentially. In SQL the norm can be calculated as

```
select sqrt(sum(v * v)) as fnorm
from A
```

Other very important operations are the matrix-matrix and matrix-vector multiplications which can be declared as basic methods since they appear in nearly every advanced algorithm in statistics or numerical analysis. Consider two matrices $A \in \mathbb{R}^{k \times m}$ and $B \in \mathbb{R}^{m \times n}$ with $k, m, n \in \mathbb{N}^{\geq 1}$ with schemes as given above. A matrix-matrix multiplication $C = A \cdot B$ can then be calculated as

```
select A.i, B.j,
       sum(A.v * B.v)
from A join B on A.j = B.i
group by A.i, B.j
```

A matrix-vector multiplication can be done analogously. As mentioned before, this approach of calculating matrix-matrix products especially excels if one is using sparse matrices. Furthermore, one can see parallels to the MapReduce framework (see for instance [29]) in the way of how elements are combined and processed. Aside from the matrix multiplication one frequently used method is the transposition of a matrix A which is written as A^T and its elements α_{ij} can be calculated by swapping the column and row indices of A

$$\alpha_{ij} = a_{ji}. \quad (1)$$

This operation can be done as

```
select A.j as i,
       A.i as j, A.v
from A
```

Since transposing is basically swapping two attributes, it is obvious that this operation should not be materialized if it is possible to combine it with other blocks of operations.

6 HIDDEN MARKOV MODEL

As one of the most popular techniques in the Machine Learning area, the Hidden Markov Model (HMM) has certainly become a very important part of many real life applications. Therefore, we will briefly describe the basic theory of the Hidden Markov Model in this section. We introduce a major example of an HMM-experiment done at the University of Rostock. This experiment will be used to not only clarify theoretical aspects, but also show the translation process into SQL presenting practical algorithms. The knowledge gathered in this section will be the basis of experimental evaluations we have done, comparing R as a representative for statistical software and MonetDB as a representative for (main memory, column-store, SQL-based) database systems. In order to give a coherent definition of the Hidden Markov Model it is reasonable to have some insights into the concept of Markov processes.

Since this paper is not meant to stress theoretical aspects, this section will be rather short. We would like to refer the interested readers to Rabiner [34] for a more elaborated discussion on theory and application. Furthermore we would like to introduce a meeting scenario, which has been realized by the chair of *mobile multimedia information systems* [13,26,27] at the University of Rostock.

6.1 Experiment

We now introduce an experiment of a meeting scenario as an example for the development process of the Hidden Markov Model. The original procedure used sensor data which has been stored in CSV files and was implemented entirely in R. Therefore, this experiment has turned out to be an ideal entry point for an analysis of possibly SQL affine algorithms in the area of Machine Learning.

In this meeting scenario (see Figure 4) the horizontal positions of three probands are tracked by personal sensor devices. These data are stored at irregular timestamps with an annotation of the current state for later evaluation. The possible states of the system derive from linking the individual states of each proband with possible values

$$S_{\text{Proband}} = \{ \begin{array}{l} \text{enter, moveDoorStage, moveDoorSeat,} \\ \text{moveSeatDoor, moveStageSeat,} \\ \text{moveSeatStage, present, sit, discuss,} \\ \text{exit} \end{array} \}. \quad (2)$$

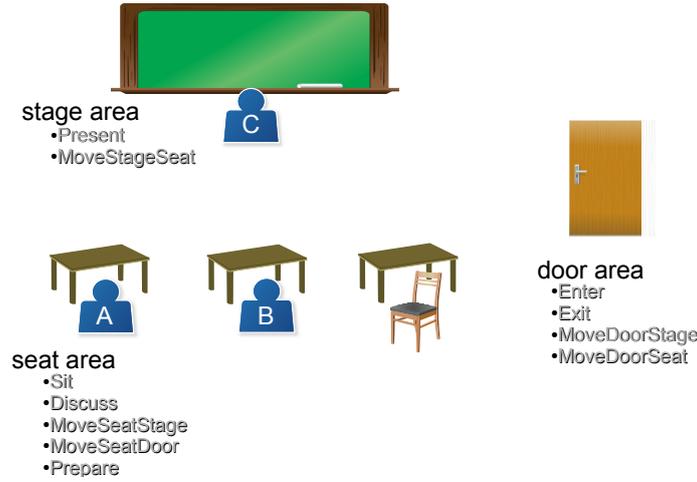


Figure 4: A possible system state: Proband C is presenting, while A and B are sitting/listening.

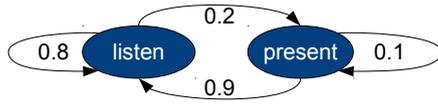


Figure 5: Graph representation of the Markov Chain example

At this point we would like to give some theoretical insights into Markov chains, which built the basis of the Hidden Markov Model.

6.2 Discrete finite Markov Processes

A discrete Markov process (or Markov chain) is a stochastic process for which the probability distribution of the next state depends only on a limited amount of preceding states. In its most common form, the probabilities even depend only on the **current** state. This characteristic is usually called *memorylessness* or Markov property.

For a formal view let N be the number of possible states in the state space $S = \{S_1, S_2, \dots, S_N\}$. Furthermore let $Q = \{q_1, q_2, \dots, q_T\}$ (with $T \in \mathbb{N}^{\geq 1}$) be a family of random variables over S , where T denotes the highest time value in the corresponding discrete time space $\{0, 1, \dots, T\}$. The family Q is called a discrete finite Markov chain of order 1 if

$$P(q_{t+1} = S_{j_{t+1}} | q_t = S_{j_t}, q_{t-1} = S_{j_{t-1}}, \dots, q_0 = S_{j_0}) = P(q_{t+1} = S_{j_{t+1}} | q_t = S_{j_t})$$

for all $t = 1, 2, \dots, T - 1$ and $j = \{j_{t+1}, j_t, \dots, j_0\} \in \{1, 2, \dots, N\}^{t+1}$.

Let us come back to the meeting scenario: consider a simple model that determines whether a proband is listening or presenting $S = \{S_1 = \text{listen}, S_2 = \text{present}\}$ in the next time step. Therefore, a couple of meetings have been recorded in 10 minute time steps and it has been found that a proband has a 80% chance to remain in his seat, and a 20% chance to present in the next time step. Furthermore there is a 10% chance to continue presenting and a 90% chance to stop presenting and to return to the listener role (see Figure 5). This can formally be written as

$$\begin{aligned} P(q_{t+1} = S_1 | q_t = S_1) &= 0.8 \\ P(q_{t+1} = S_2 | q_t = S_1) &= 0.2 \\ P(q_{t+1} = S_1 | q_t = S_2) &= 0.9 \\ P(q_{t+1} = S_2 | q_t = S_2) &= 0.1 \end{aligned}$$

with the matrix notation

$$\mathbf{A} = \begin{pmatrix} 0.8 & 0.2 \\ 0.9 & 0.1 \end{pmatrix}.$$

It is now easily possible to make statements on the likelihood of the possible behaviour of a proband. For example, we would like to know whether a proband is presenting in two time steps, i.e. $q_2 = S_1$, if he is also presenting at the current time. The result can be obtained by simply adding the two possible scenarios

$$\begin{aligned} P(q_2 = S_1 | q_0 = S_1) &= \\ &P(q_2 = S_1 | q_1 = S_1) \cdot P(q_1 = S_1 | q_0 = S_1) \\ &+ P(q_2 = S_1 | q_1 = S_2) \cdot P(q_1 = S_2 | q_0 = S_1) \\ &= 0.8 \cdot 0.8 + 0.9 \cdot 0.2 = 0.82, \end{aligned}$$

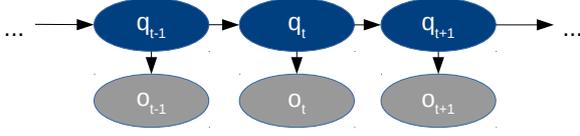


Figure 6: Dependencies of observations and system states in a Hidden Markov Model (of order 1)

stating that the probability for presenting is 82 %. For later calculations it is important to see that this calculation could also be done as

$$A^2 \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0.82 \\ 0.81 \end{pmatrix} = \begin{pmatrix} P(q_2 = S_1 | q_0 = S_1) \\ P(q_2 = S_2 | q_0 = S_1) \end{pmatrix}. \quad (3)$$

6.3 Definition of the Hidden Markov Model

For many situations a Markov model is not applicable because its states are not *observable*, i.e. they are hidden. Often there are different observable states that imply whether or not these *hidden* states are probable. These circumstances can be modeled via a Hidden Markov Model which introduces a second stochastic process for the observable states to the system.

For a formal definition let N be the number of different states in the hidden state space $S = \{S_1, S_2, \dots, S_N\}$, M the number of different states in the observation state space $V = \{V_1, V_2, \dots, V_M\}$ and $\{q_0, q_1, \dots, q_T\}$ and $\{o_0, o_1, \dots, o_T\}$ families of random variables with values in S and V respectively, where $T \in \mathbb{N}^{\geq 1}$ refers to the corresponding discrete time space $\{0, 1, \dots, T\}$.

A *Hidden Markov Model* (HMM) is a tuple $\lambda = (S, V, A, B, \pi)$, where $A = (a_{ij})_{i,j=1,\dots,N} \in [0, 1]^{N \times N}$ denotes the transition matrix with $a_{ij} = P(q_{t+1} = S_i | q_t = S_j)$, $B = (b_{ij})_{i=1..N, j=1..M}$ is the observation matrix, which holds the probabilities to observe V_i while in state S_j , i.e. $b_{ij} = P(o_t = V_i | q_t = S_j) \in [0, 1]^{M \times N}$ and $\pi = \{\pi_i\}_{i=1..N}$ is the initial state distribution. Note that the observations only depend on the current system states (see Figure 6). Furthermore, the model is called homogeneous if the probability matrices A and B are independent of time.

6.4 Experiment HMM-Parameters

After defining the Hidden Markov Model, we can add some properties to our meeting scenario. It is obvious that the main task is to generate the matrices A, B from the gathered sensor data in context of a set of possible

states (as introduced in Equation 2 in Subsection 6.1) and possible observations. In the given experiment, the sensor data has been used to check whether the probands are staying in key areas (observable symbols), namely

$$V_{\text{Proband}} = \{\text{in seat, at stage, at door, else where}\}.$$

With the information about the current position it is possible to make assumptions about the actions of the probands. Using the example described above, a proband is most likely listening in his seat when he is in the area around his seat. On the other hand, a proband might be presenting when he recently arrived at the presentation stage.

Closing the theory part, one can summarize the overall experimental process in the following steps:

1. Input
2. Train HMM
3. Run model
4. Evaluate model
5. Output

The corresponding equations for training, using the model, and evaluating the model will be presented in the next section together with their translation into SQL.

7 HMM INTO SQL

We will now give some insights into possible translations of the Hidden Markov Model into SQL statements. It will be shown that this can be done quite naturally in most cases.

Input

The whole R-Code starts with loading the observations and annotations (hidden states) with respect to time. The observation consists of (horizontal) positional data (x, y coordinates, double precision) for each proband at every timestamp. All the experimental data has been stored in CSV files. Therefore, each rerun of the code will also lead to reread these files into main memory. This might not be a problem for small files, but does heavily hurt the performance when file sizes start to get bigger. While using database systems, data has to be written once into respective tables. Usually, this loading process does take longer than simply loading a flat file into main memory. Anyhow, on experimental setups where input data has to be used (and reread) in multiple runs the initial overhead does pay for itself rather fast. For this model we used the relation schemes

```
Observations_j (i int, time double,
               double xA, double yA,
               double xB, double yB,
               double xC, double yC)
```

```
Annotations_j (i int, time double,
              AnnoA varchar,
              AnnoB varchar,
              AnnoC varchar)
```

where $j \in \{1, \dots, k\}$ denotes the number of experiments and i is used as an automatic increment for join operations, since the time parameter t is not necessarily unique. We will neglect j in later analysis.

7.1 Calculate Model

The first big part of the program is the training of the model (see Subsection 6.3), i.e. computing the transition matrix A , the observation matrix B and the initial state distribution π .

Transition matrix A

Calculating the transition matrix can be done by simply counting the transitions that have been annotated. If we denote (for simplicity) states with a number $i, j \in \{1, \dots, |S|\}$, we can compute each matrix entry $a_{ij} = P(q_{t+1} = j | q_t = i)$ using

$$\frac{\text{\#Transitions from } i \text{ to } j}{\text{\#Transitions starting from } i}$$

In R this can be done in the following way:

```
trainModel <- function(anno){
  tab <- table(anno[-1],
              anno[-length(anno)]);
  result <- tab%%diag(1/colSums(tab));
  dimnames(result) <- dimnames(tab);
  result
}.
```

For a better understanding, the whole progress is depicted in Figure 7. The input `anno` is a vector, whose entries hold the three individual hidden states of the probands in one string at each timestamp. The indices `-1` and `-length(anno)` in the first statement delete the first and the last entry of the respective vector, creating the predecessor/successor relationship.

The `table` function results in a table representing the count of the state transitions. For example, the entry (1,1) represents, how often the state (enter,enter,enter) changes to (enter,enter,enter) after one timestep - in this case no change after the state transition. As a second

example, the entry (1,3) represents, how often the state (enter,enter,enter) changes to (enter,move door stage,enter) after one timestep.

The following statement divides every column of the table by their respective sum. Therefore, each number of state transitions is divided by the sum of state transitions starting with the same predecessor, which is exactly what the transition matrix A is. The final statement keeps the names of the dimension for later calculations.

For simplicity, we consider a scheme

```
anno(i int, v varchar)
```

holding the states for all probands in only one entry. The training can then be done in SQL via

```
create view tmp as
  select al.v as pre, a2.v as suc,
         count(al.v) as v
  from anno a1 join anno a2
         on a1.i = a2.i-1
  group by al.v, a2.v
```

```
create view denom as
  select pre, sum(v) as v
  from tmp
  group by pre
```

```
select t1.pre, t1.suc,
       cast( t1.v as double ) / t2.v as v
  from tmp t1 join denom t2
         on t1.pre = t2.pre
```

Adding the row and column indices can be achieved by joining the result table with a relation containing the hidden states with an auto incremented index, ultimately leading to the desired representation of the transition matrix A . Note that one needs to cast the numerator into double precision in the last query, since integer division results in an integer as well. Furthermore, the `group by` statement in the first query only returns pairs (pre, suc) for which the corresponding count is greater than 0. The whole process can benefit from its sparse-matrix-like scheme by neglecting divisions with a 0 numerator. While this is not really pushing the performance in this model, it shows that the matrix representation can be really advantageous in certain situations and especially for big data.

Initial probability distribution π

While not necessary, it is usually meaningful to introduce extra states for ending and beginning a run. In this model, the initial state distribution is given using an extra "START" state and the corresponding probability

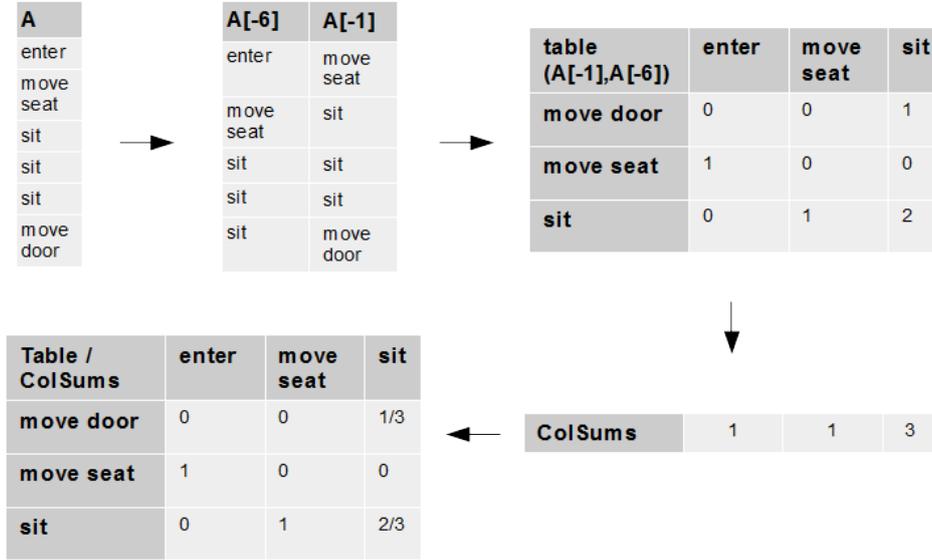


Figure 7: Example for a progress of the calculation of the transition matrix A

$P(q_0 = \text{"START"}) = 1$, leaving the probability for any other state at 0.

Observation matrix B

The final parameter to be calculated is the observation matrix B . Therefore, at each time stamp the measured positions for each proband are evaluated separately by using the euclidean distance to predefined areas, namely door, stage, seat and else where. Now that the observation symbols have been obtained, it is possible to give an estimated probability (using a-priori knowledge) for this observation for any hidden state. This can be done analogously to the method for transition matrices by counting the pairs of states and symbols in the following manner

$$b_{ij} = \frac{\#\text{Pairs of } S_i \text{ and } V_j}{\#S_i}.$$

In SQL this can be done using a nested query and groupings

```

select tt.state, tt.obs,
       cast(nom as double) / denom
from
  (select v, count(v) as denom
   from anno
   group by v) t
join
  (select anno.v as state,
         obs.v as obs,

```

```

count(anno.v) as nom
from anno join obs
on anno.i = obs.i
group by anno.v, obs.v) tt
on t.v = tt.state

```

assuming that

```
anno(i int, v varchar)
```

and

```
obs(i int, v varchar)
```

are relations holding the system states and observed symbols for any time stamp index i .

However, a different approach has been taken in this experiment. Here a parameter $\theta = 0.08$ is used to describe a relation between observation and hidden state and their corresponding probability in the following way: If the hidden state is in the same area as the proband is, due to the observation, the probability is $P = 1 - \theta$, otherwise the routine returns $P = \theta$. For example, the sensor data implies that proband A is at the door area at timestamp t . Being at the door area implies that the proband might be exiting or entering the room and therefore the probabilities can be given as

$$P^A(o_t = \text{is at door} | q_t = \text{exit}) = 1 - \theta$$

$$P^A(o_t = \text{is at door} | q_t = \text{enter}) = 1 - \theta.$$

On the other hand it is unlikely that the proband is

currently sitting in his seat which leads to

$$P^A(o_t = \text{is at door} | q_t = \text{sit}) = \theta.$$

If these results are multiplied by the corresponding results from the other probands $P = P^A P^B P^C$ one obtains the overall observation matrix B . Since the positions of each proband are variable, the distances have to be evaluated for every proband at any step, while the probability $P(o = \text{is in area} | q = \text{state})$ is independent with respect to time. This approach offers the benefit of a degree of freedom (θ), which can be used to adjust the relation between symbols and states.

For the SQL computation we introduce variables X_d , Y_d and R_d representing the x,y coordinates and the radius of the door area. It might be beneficial to compute the logical terms for every proband for every location *door, seat, stage and else where* in one table, rather than in one table for each location like we will do here, due to convenience. Using the **cast** function to convert logical values into integers, one can compute the observation matrix in SQL as

```
create table B as
select i, THETA+(1-2*THETA)*doorA
from (select i,
cast((o.AX - Xd)2+
(o.AY - Yd)2 < Rd) as int)
as doorA
from observation o ) tmp
```

Here the relation

```
observation (i int, t int,
AX double, AY double,
BX double, BY double,
CX double, CY double)
```

holds the positional data from every proband at every timestamp t .

7.2 Run Model

We will now use the model to calculate different properties, as for example explained in [10], for instance the probability of given state sequences as introduced in Equation 3 in Subsection 6.2 or the most likely state sequence given a observation sequence and an initial distribution. We would like to focus on the *forward algorithm* as an example for commonly used methods on the Hidden Markov Model. This algorithm calculates the probability of witnessing an observation sequence in a given model. Furthermore, this algorithm is a good representative for a Machine Learning algorithm, since it is composed of basic operations, i.e. vector-matrix- and discrete multiplications and a simple aggregation.

The algorithm consists of three main steps and can be described as follows:

Algorithm 1 Forward Algorithm

```
Input:  $\mathbf{o} = \{o_0, \dots, o_\theta\}$   $\triangleright$  Observation Sequence
 $\mathbf{p} := \pi \cdot B_{:,o_0}$   $\triangleright$  Initialization
for  $i = 1, \dots, \theta$  do  $\triangleright$  Recursion
     $\mathbf{p} := \mathbf{p}^T A \cdot B_{:,o_i}$ 
 $P(\mathbf{o}|A, B, \pi) = \text{sum}(\mathbf{p})$ ,  $\triangleright$  Termination
```

where \cdot describes the discrete vector multiplication $v \cdot w = (v_i w_i)_{i=1, \dots, n}$ and $B_{:,o_i}$ describes the column of B that holds the probabilities for the observation state o_i . A possible solution for implementing the forward algorithm in R can be found in the appendix. This code has been used as a reference for the experimental evaluation in Section 8.

There are multiple ways to approach this algorithm in SQL, depending on the SQL-dialect one is using. We tested two different approaches on MonetDB (see Section 8), where one is relying on updates and the other one has been nested into one big query. Using updates is straight forward and easy to implement:

```
insert into alpha
select pi.i, pi.v * B.v
from pi join B
on pi.i = B.i
where B.j = o_0;

for k=1, ..., theta
update alpha set v =
(select alpT.A.v * B.v
from
(select A.j as i, sum(aa.v * A.v)
as v from alpha aa join A
on aa.i = A.i
group by A.j) alpT.A
join B
on alpT.A.i = B.i
where B.j = o_k and
B.i = alpha.i)
where exists
(select * from
(select distinct j as i
from A) ttt
where ttt.i = alpha.i);

end for;

select sum(v) from alpha;
```

As we will show in the upcoming section, this

approach is inferior to the following nested query method, due to the extensive use of update operations, which will not only force the DBMS to write on secondary storage, but also write log files to guarantee transaction safety. A better approach is to use a big nested query. Since this query is big and hard to fit in a readable form, we would like to split the query into the three main parts. The inner query Q_0 calculates the initialization as follows

```
select pi.i as i, pi.v * B.v as v
from pi join B on pi.i = B.i
where B.j = o_0
```

With Q_0 we can define a recursive sequence of queries $Q_1, \dots, Q_{\text{theta}}$, where Q_k is defined as

```
select alptA.i as i, alptA.v * B.v as v
from
  (select A.j as i, sum(aa.v * A.v) as v
   from Q_{k-1} aa join A
    on aa.i = A.i
   group by A.j) alptA
join B on alptA.i = B.i
where B.j = o_k
```

Finally, the summation of the intermediate values can be done simply using the sum aggregation

```
select sum(v) from Q_theta.
```

It is noteworthy that various SQL-driven databases have the capacity to perform recursive queries, which will most likely result in a better speedup in comparison to the update version. Anyhow, we decided to use the sequence above since we evaluated our queries against the main memory DBMS MonetDB, which does not support recursive queries.

In the next section, we will present and discuss the performance results we obtain from calculating the transition matrix and the forward algorithm.

8 EVALUATION

We have implemented two experiments which calculate the transition matrix of an HMM and the forward algorithm as described in Subsection 7.1. In both experiments all approaches have lead to the same results, and therefore we will not discuss the quality of these any further. Moreover, we processed every experimental run five times and used the average time needed as the overall respective result.

Table 1: Time taken (in ms) for calculating transition matrices in R and MonetDB as described in Subsection 8.1 and depicted in Figure 8

Dimension	R	MonetDB	R / MonetDB
1e5	36	32	1.125
1e6	210	184	1.141
5e6	907	776	1.169
1e7	1118	1437	0.778
3e7	16817	4398	3.824
5e7	28657	9587	2.99
7e7	40830	16008	2.551

8.1 Transition Matrix

For this first test we created randomized test data, using the normal distribution and the state space $S = \{ 'a', 'b', \dots, 'z' \}$, meaning that the probability of having a certain system state is $1/26$ for any time step. We have chosen the dimensions (number of timesteps) $10^5 = 1e5, 10^6, 5 \cdot 10^6 = 5e6, 10^7, 3 \cdot 10^7, 5 \cdot 10^7$ and $7 \cdot 10^7$, while working on a server with 2×2.1 GHz Processors, 12 GB DDR-3 RAM, 60 GB secondary memory and Fedora 21 (64 bit) as the operating system.

In the experiment we compared the performance of the SQL queries on MonetDB 11.19.9 to the corresponding R-algorithm with R 3.3.1. As one can see in Table 1 and Figure 8, the database system outperforms R with increasing problem size, although using datasets that fit into main memory, demonstrating the power of grouping algorithms on databases. Note that the y-axis of the plot is logarithmic which might visually soften the impression of the performance difference. It is also noteworthy that the computation time has been taken after the data has already been read by R. Reading the data from csv-files took from $2 \times$ up to $16 \times$ of the calculation time in our experiment.

8.2 Forward Algorithm

The second experiment we conducted on the same setup as above measured the time needed to calculate the forward algorithm using a random observation sequence of length 10 on varying model sizes. For simplicity we used a materialized copy of the (dense) transition matrix as the observation matrix, since the results would not have an impact on the process speed, unless we would use sparse matrices.

The results can be seen in Figures 9 and 10. Various conclusions can be drawn from these measurements. First, it is clear that R outperforms the database solutions by quite a margin until R runs out of memory. This is what we expected, since our solutions rely on numerous joins, our matrices are dense, and R is perfectly suited

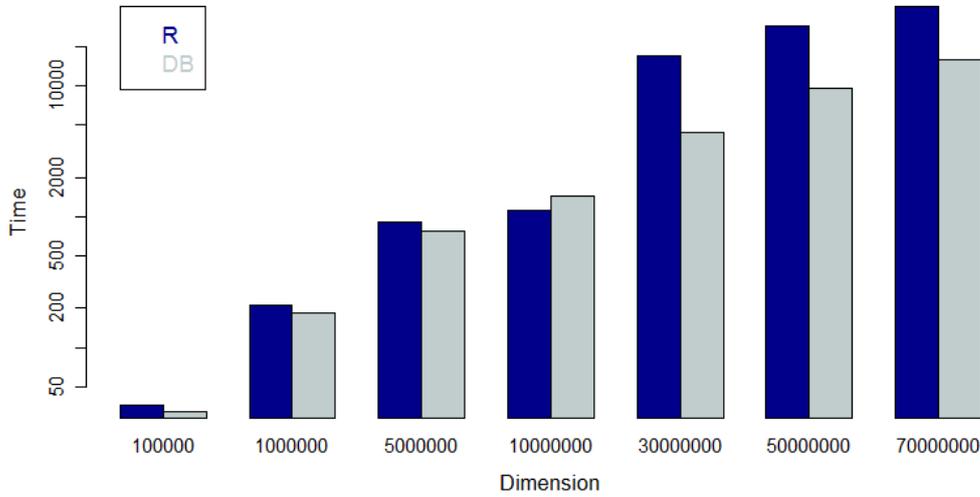


Figure 8: Time (in ms) used for pure calculation (log y-axis) with a database system (grey) and R (blue) on calculating a transition matrix

for such algorithms using the LAPACK/BLAS libraries. The most interesting aspect for our research is depicted in Table 2 and Figure 10. Here it is easy to observe that the factor between R and the database solutions remain stable (≈ 7.7) and that the database solution would not change its trend while moving out of the area of main memory calculations. The latter property is unlikely to be observed in R, when cutting the algorithm down into chunks, which fit into memory.

The result of these experiments confirms our hypothesis that using database systems on large Machine Learning problems can be very beneficial. This motivates further investigation on this topic, especially using intra-operator parallelization for the derived SQL-queries.

8.3 Limitations

Since relational database systems have not been developed for pure scientific calculations, SQL inevitably does have limitations regarding its expressiveness on certain operations.

As seen in the previous section, database solutions can surpass in-memory software in certain methods on small data sets, like the presented grouping scenario. However, environments like R or MatLab are built on the well-known LAPACK and BLAS libraries, which are highly tuned for linear algebra operations on SIMD computers. Therefore, it is not likely that database systems can outperform these systems while data fits into main memory. It is important to understand that

database solutions thrive on their ability of optimized access strategies, logical optimization, and transaction safety.

Regarding scientific calculations, the main limitations we found have been so-called *fine grained algorithms* that mostly consist of iterations and repeated manipulation of single elements. Frequently used examples are certain matrix decompositions, for instance the *Cholesky* decomposition. Consider a symmetric positive-definite matrix $A \in \mathbb{R}^{n \times n} = (a_{ij})_{ij}$. A decomposition

$$A = LL^T \quad (4)$$

with a lower triangular matrix $L = (l_{ij})_{ij} \in \mathbb{R}^{n \times n}$ is called Cholesky decomposition. As described in [42], it is possible to directly compute the latter with the following equations

$$l_{ij} = \sqrt{a_{ij} - \sum_{k=1}^{j-1} L_{jk}^2}$$

$$l_{ij} = \frac{1}{l_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik}l_{jk} \right) \quad \text{for } i > j.$$

While the appearance of this algorithm is somewhat similar to usual matrix operations at first glance, the main difference is that the calculation of an element l_{ij} with $j > 1$ uses updated elements l_{ik} with $k > 1$, which would lead to many selective low cost queries on

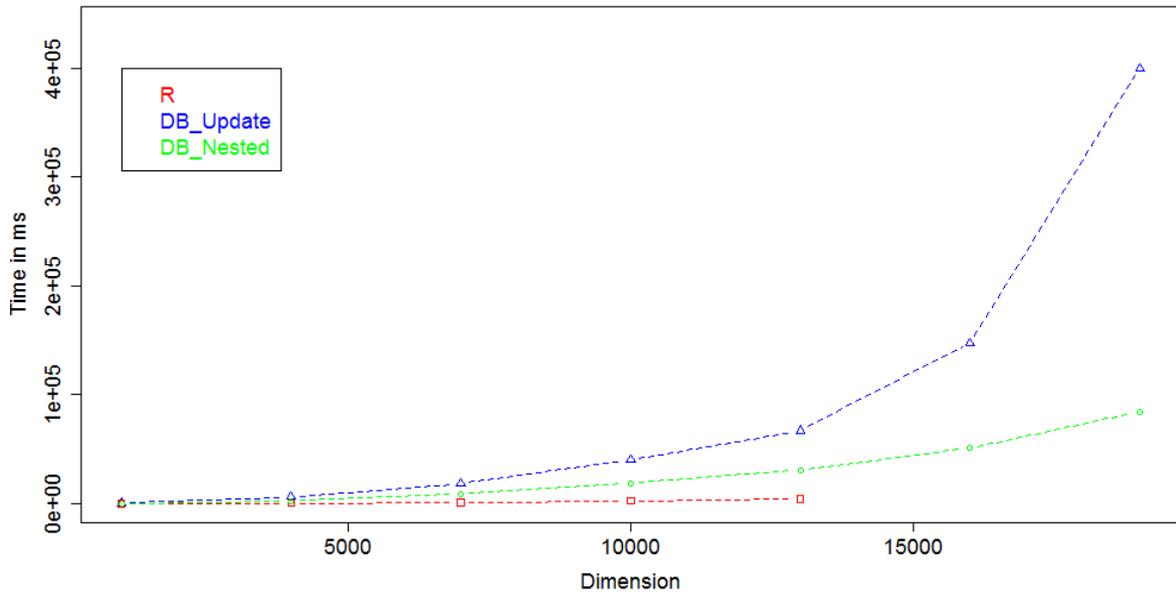


Figure 9: Time (in ms) measured for pure calculation of the forward algorithm

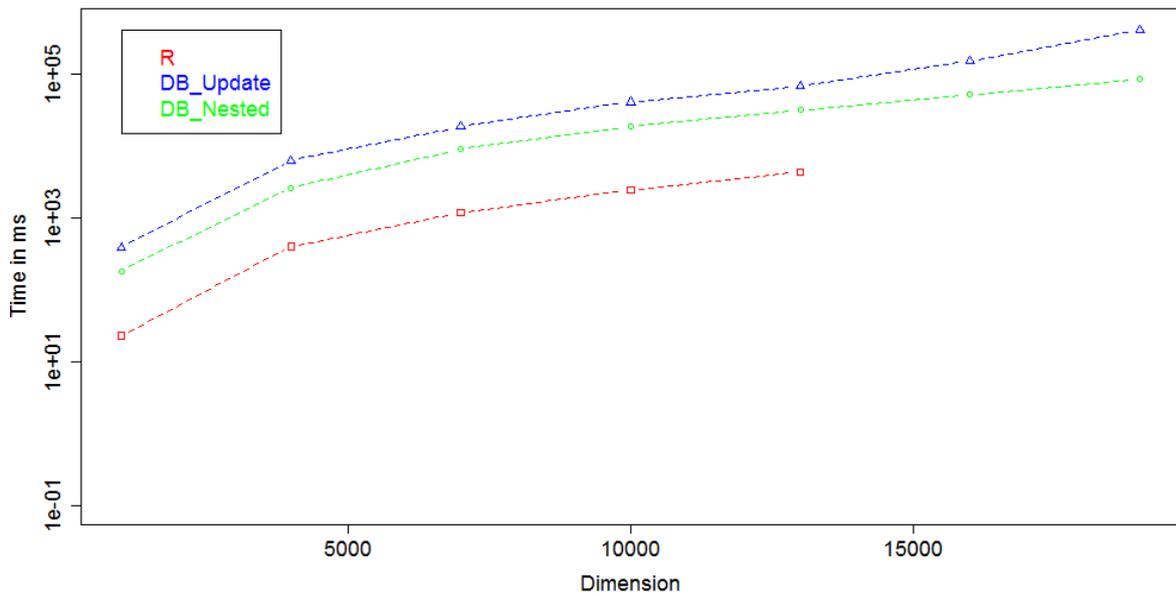


Figure 10: Time (in ms) measured for pure calculation of the forward algorithm using logarithmic y-axis

Table 2: Time taken (in ms) for calculating the forward algorithm in R and MonetDB as described in Subsection 8.2 and depicted in Figures 9 and 10

Dimension	R	DB Update	DB nested	DB Update / DB nested	DB nested / R
1e3	22	386	182	2.121	8.273
4e3	394	6163	2598	2.372	6.594
7e3	1159	18334	8922	2.055	7.698
1e4	2407	40352	18539	2.176	7.702
1.3e4	4217	66966	31026	2.158	7.357
1.6e4	NA	147295	50863	2.896	NA
1.9e4	NA	399473	84267	4.741	NA

a database. This is on the one hand not very elegant to do in SQL and on the other limits the possibilities of query optimization.

9 CONCLUSION

This paper has discussed how feasible Machine Learning algorithms can be processed on database systems. Preferable relational schemas for matrices and vectors have been discussed and possible strategies for calculating basic methods, like matrix multiplications and discrete algebraic operations, have been presented. Furthermore, a brief introduction to the theory of the Hidden Markov Model has been given, concluding in an analysis of a real Hidden Markov experiment which has been coded in the language R. In two experiments, we have shown that database solutions on the one hand show good scaling properties when data size surpasses the main memory wall, and on the other hand can outperform statistical in-memory-software on certain methods in the area of Machine Learning.

10 FUTURE WORK

So far we have analyzed and are analyzing several other algorithms in the area of Machine Learning and Scientific Computation. Among them, there are for instance Eigenvalue solver, QR-matrix-decomposition, QR-based linear regression, outlier detection, and principal component decompositions. The results mainly agree with our assumption that database solutions scale well with growing data sizes and are especially suited for implementing sparse matrix algorithms.

Furthermore, we have settled on a small group of basic linear algebra operators that we are analyzing for intra operator parallelism. When finished, we would like to test our strategies on our analyzed Machine Learning algorithms, like the Hidden Markov Model, and compare the results to commonly used frameworks for Big Data computations like Hadoop, Apache Flink, and Apache Spark. We already did some experiments in comparing

our ML2PSQL approach on PostgreSQL and its parallel version Postgres-XL with solutions on Flink and Spark. These results will be published in the near future.

After experimental evaluation, efforts will be put to enhance this framework with a special kind of provenance management. Hereby we will address projection problems: Consider the calculation of any statistical model. This model might be created using all or at least a really high number n of sensors. Since the model should be used for real time applications, the amount of data needs to be drastically reduced. Therefore we would like to choose those $k \ll n$ sensors with the most impact on the model. For this, several mathematical approaches like singular value decomposition, principal component analysis, and also methods of data provenance (why- and how-provenance) will be investigated.

As a last step, we have to improve the translation of ML code to SQL statements. When using R as an implementation language for the ML algorithms, we struggle with the power and complexity of R (and the undecidability of the equivalence of Turing machines, for the Turing-complete language R). In fact, our first step in this project was an attempt to build an automatic R-to-SQL compiler. This task turned out to be unsolvable due to the totally different approaches and styles data scientists use for, e.g., formulating and implementing a Hidden Markov Model in R. It would be more appropriate for data scientists, ML developers and our ML2SQL transformation process to use a matrix-equation formalism with recursion or iterations to describe an ML problem instead of a complete programming language like R.

ACKNOWLEDGEMENTS

We would like to thank Prof. Dr.-Ing. Thomas Kirste, Dr.-Ing. Frank Krüger and Martin Nyolt for their support on the basic theory of Machine Learning techniques and the provision of a lot of experiments and experimental data for our use case of data-driven development of

assistive systems. We would also like to thank the anonymous reviewers of the first version of this paper for their many detailed and constructive comments that helped to improve this paper.

REFERENCES

- [1] S. Abiteboul, M. Arenas, P. Barceló, M. Bienvenu, D. Calvanese, C. David, R. Hull, E. Hüllermeier, B. Kimelfeld, L. Libkin, W. Martens, T. Milo, F. Murlak, F. Neven, M. Ortiz, T. Schwentick, J. Stoyanovich, J. Su, D. Suciu, V. Vianu, and K. Yi, “Research directions for principles of data management (abridged),” *SIGMOD Record*, vol. 45, no. 4, pp. 5–17, 2016.
- [2] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users’ Guide*, 3rd ed. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1999.
- [3] C. M. Bishop, *Pattern recognition and machine learning, 5th Edition*, ser. Information science and statistics. Springer, 2007.
- [4] M. Boehm, D. Burdick, A. V. Evfimievski, B. Reinwald, P. Sen, S. Tatikonda, and Y. Tian, “Compiling machine learning algorithms with SystemML,” in *ACM Symposium on Cloud Computing, SOCC ’13, Santa Clara, CA, USA, October 1-3, 2013*, G. M. Lohman, Ed. ACM, 2013, p. 57:1.
- [5] S. Brin and L. Page, “Reprint of: The anatomy of a large-scale hypertextual web search engine,” *Computer Networks*, vol. 56, no. 18, pp. 3825–3833, 2012.
- [6] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, “Apache Flink™: Stream and Batch Processing in a Single Engine,” *IEEE Data Eng. Bull.*, vol. 38, no. 4, pp. 28–38, 2015.
- [7] R. Chirkova, “Query containment,” in *Encyclopedia of Database Systems*. Springer US, 2009, pp. 2249–2253.
- [8] T. Dalenius, “Finding a Needle In a Haystack or Identifying Anonymous Census Records,” *Journal of Official Statistics*, vol. 2, no. 3, pp. 329–336, 1986.
- [9] A. Deutsch and R. Hull, “Provenance-Directed Chase&Backchase,” in *In Search of Elegance in the Theory and Practice of Computation - Essays Dedicated to Peter Buneman*, ser. Lecture Notes in Computer Science, V. Tannen, L. Wong, L. Libkin, W. Fan, W. Tan, and M. P. Fourman, Eds., vol. 8000. Springer, 2013, pp. 227–236.
- [10] R. Durbin, *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*, ser. Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids. Cambridge University Press, 1998.
- [11] A. Elgohary, M. Boehm, P. J. Haas, F. R. Reiss, and B. Reinwald, “Scaling machine learning via compressed linear algebra,” *SIGMOD Record*, vol. 46, no. 1, pp. 42–49, 2017.
- [12] A. Ghoting, R. Krishnamurthy, E. P. D. Pednault, B. Reinwald, V. Sindhvani, S. Tatikonda, Y. Tian, and S. Vaithyanathan, “SystemML: Declarative machine learning on MapReduce,” in *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*, S. Abiteboul, K. Böhm, C. Koch, and K. Tan, Eds. IEEE Computer Society, 2011, pp. 231–242.
- [13] M. Giersich, T. Heider, and T. Kirste, “AI Methods for Smart Environments - A Case Study on Team Assistance in Smart Meeting Rooms,” in *Constructing Ambient Intelligence - Aml 2007 Workshops Darmstadt, Germany, November 7-10, 2007 Revised Papers*, ser. Communications in Computer and Information Science, A. Ferscha and E. Aitenbichler, Eds., vol. 11. Springer, 2007, pp. 4–13.
- [14] G. Giorgidze, T. Grust, T. Schreiber, and J. Weijers, “Haskell Boards the Ferry - Database-Supported Program Execution for Haskell,” in *Implementation and Application of Functional Languages - 22nd International Symposium, IFL 2010, Alphen aan den Rijn, The Netherlands, September 1-3, 2010, Revised Selected Papers*, ser. Lecture Notes in Computer Science, J. Hage and M. T. Morazán, Eds., vol. 6647. Springer, 2010, pp. 1–18.
- [15] H. Grunert and A. Heuer, “Generating privacy constraints for assistive environments,” in *Proceedings of the 8th ACM International Conference on Pervasive Technologies Related to Assistive Environments, PETRA 2015, Corfu, Greece, July 1-3, 2015*, F. Makedon, Ed. ACM, 2015, pp. 27:1–27:4.
- [16] H. Grunert and A. Heuer, “Datenschutz im PArADISE,” *Datenbank-Spektrum*, vol. 16, no. 2, pp. 107–117, 2016.
- [17] H. Grunert and A. Heuer, “Privacy protection through query rewriting in smart environments,”

- in *Proceedings of the 19th International Conference on Extending Database Technology, EDBT 2016, Bordeaux, France, March 15-16, 2016, Bordeaux, France, March 15-16, 2016.*, E. Pitoura, S. Maabout, G. Koutrika, A. Marian, L. Tanca, I. Manolescu, and K. Stefanidis, Eds. OpenProceedings.org, 2016, pp. 708–709.
- [18] H. Grunert, M. Kasparick, B. Butzin, A. Heuer, and D. Timmermann, “From cloud to fog and sunny sensors,” in *Proceedings of the Conference "Lernen, Wissen, Daten, Analysen", Potsdam, Germany, September 12-14, 2016.*, ser. CEUR Workshop Proceedings, R. Krestel, D. Mottin, and E. Müller, Eds., vol. 1670. CEUR-WS.org, 2016, pp. 83–88.
- [19] S. Günemann, “Machine learning meets databases,” *Datenbank-Spektrum*, vol. 17, no. 1, pp. 77–83, 2017.
- [20] A. Heuer and H. Lubinski, “Data Reduction - an Adaptation Technique for Mobile Environments,” in *In Interactive Applications of Mobile Computing (IMC'98)*, 1998, pp. 1–2.
- [21] A. Heuer, “METIS in PArADISE: Provenance Management bei der Auswertung von Sensordatenmengen für die Entwicklung von Assistenzsystemen,” in *BTW 2015 - Workshopband*, ser. LNI, vol. 242. GI, 2015, pp. 131–136.
- [22] Y. Huai, A. Chauhan, A. Gates, G. Hagleitner, E. N. Hanson, O. O'Malley, J. Pandey, Y. Yuan, R. Lee, and X. Zhang, “Major technical advancements in Apache Hive,” in *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, C. E. Dyreson, F. Li, and M. T. Özsu, Eds. ACM, 2014, pp. 1235–1246.
- [23] S. Idreos, F. Groffen, N. Nes, S. Manegold, K. S. Mullender, and M. L. Kersten, “Monetdb: Two decades of research in column-oriented database architectures,” *IEEE Data Eng. Bull.*, vol. 35, no. 1, pp. 40–45, 2012.
- [24] J. Lajus, H. Mühleisen, “Efficient data management and statistics with zero-copy integration,” in *International Conference on Scientific and Statistical Database Management*, vol. 24, 2014.
- [25] F. Krüger, M. Nyolt, K. Yordanova, A. Hein, and T. Kirste, “Computational State Space Models for Activity and Intention Recognition. A Feasibility Study.” *PLOS ONE*, Nov. 2014, 9(11): e109381.
- [26] F. Krüger, K. Yordanova, C. Burghardt, and T. Kirste, “Towards creating assistive software by employing human behavior models,” *Journal of Ambient Intelligence and Smart Environments*, vol. 4, no. 3, pp. 209–226, 2012.
- [27] F. Krüger, K. Yordanova, A. Hein, and T. Kirste, “Plan synthesis for probabilistic activity recognition,” in *ICAART (2)*, 2013, pp. 283–288.
- [28] A. Kumar, M. Boehm, and J. Yang, “Data management in machine learning: Challenges, techniques, and systems,” in *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, S. Salihoglu, W. Zhou, R. Chirkova, J. Yang, and D. Suciuc, Eds. ACM, 2017, pp. 1717–1722.
- [29] J. Leskovec, A. Rajaraman, and J. D. Ullman, *Mining of Massive Datasets, 2nd Ed.* Cambridge University Press, 2014.
- [30] A. Y. Levy, A. Rajaraman, and J. D. Ullman, “Answering queries using limited external query processors,” *J. Comput. Syst. Sci.*, vol. 58, no. 1, pp. 69–82, 1999.
- [31] F. Marguerie, S. Eichert, and J. Wooley, *Linq in Action*. Greenwich, CT, USA: Manning Publications Co., 2008.
- [32] D. Marten and A. Heuer, “A framework for self-managing database support and parallel computing for assistive systems,” in *Proceedings of the 8th ACM International Conference on Pervasive Technologies Related to Assistive Environments, PETRA 2015, Corfu, Greece, July 1-3, 2015*, F. Makedon, Ed. ACM, 2015, pp. 25:1–25:4.
- [33] R. Obe and L. Hsu, *PostgreSQL - Up and Running: a Practical Guide to the Advanced Open Source Database*. O'Reilly, 2012.
- [34] L. R. Rabiner, “A tutorial on hidden markov models and selected applications in speech recognition,” in *Readings in Speech Recognition*, A. Waibel and K.-F. Lee, Eds. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1990.
- [35] J. Rafferty, C. D. Nugent, J. Liu, and L. Chen, “From activity recognition to intention recognition for assisted living within smart homes,” *IEEE Trans. Human-Machine Systems*, vol. 47, no. 3, pp. 368–379, 2017.
- [36] J. Reddy, *An Introduction to the Finite Element Method*. McGraw-Hill Education, 2005.
- [37] T. Rohrmann, S. Schelter, T. Rabl, and V. Markl, “Gilbert: Declarative Sparse Linear Algebra on Massively Parallel Dataflow Systems,” in *Datenbanksysteme für Business, Technologie*

- und Web (BTW 2017), 17. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme“ (DBIS), 6.-10. März 2017, Stuttgart, Germany, Proceedings, ser. LNI, B. Mitschang, D. Nicklas, F. Leymann, H. Schöning, M. Herschel, J. Teubner, T. Härder, O. Kopp, and M. Wieland, Eds., vol. P-265. GI, 2017, pp. 269–288.
- [38] B. Schwartz, P. Zaitsev, and V. Tkachenko, *High Performance MySQL - Optimization, Backups, and Replication: Covers Version 5.5, Third Edition*. O’Reilly, 2012.
- [39] S. A. Shahrestani, *Internet of Things and Smart Environments - Assistive Technologies for Disability, Dementia, and Aging*. Springer, 2017.
- [40] W. Shi and S. Dustdar, “The Promise of Edge Computing,” *Computer*, vol. 49, no. 5, pp. 78–81, May 2016.
- [41] K. Skala, D. Davidovic, E. Afgan, I. Sovic, and Z. Sojat, “Scalable distributed computing hierarchy: Cloud, fog and dew computing,” *Open Journal of Cloud Computing (OJCC)*, vol. 2, no. 1, pp. 16–24, 2015. [Online]. Available: <http://nbn-resolving.de/urn:nbn:de:101:1-201705194519>
- [42] J. Stoer, R. Bartels, W. Gautschi, R. Bulirsch, and C. Witzgall, *Introduction to Numerical Analysis*, ser. Texts in Applied Mathematics. Springer New York, 2002.
- [43] M. Stonebraker, P. Brown, A. Poliakov, and S. Raman, “The Architecture of SciDB,” in *Scientific and Statistical Database Management - 23rd International Conference, SSDBM 2011, Portland, OR, USA, July 20-22, 2011. Proceedings*, ser. Lecture Notes in Computer Science, J. B. Cushing, J. C. French, and S. Bowers, Eds., vol. 6809. Springer, 2011, pp. 1–16.
- [44] T. White, *Hadoop: The Definitive Guide*, 1st ed. O’Reilly Media, Inc., 2009.
- [45] K. Yordanova and T. Kirste, “A process for systematic development of symbolic models for activity recognition,” *TiiS*, vol. 5, no. 4, pp. 20:1–20:35, 2016.
- [46] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica, “Apache Spark: a unified engine for big data processing,” *Commun. ACM*, vol. 59, no. 11, pp. 56–65, 2016.
- [47] Y. Zhang, H. Herodotou, and J. Yang, “RIOT: i/o-efficient numerical computing without SQL,” *CoRR*, vol. abs/0909.1766, 2009.

APPENDICES

Forward Algorithm in R

This code has been used for the experimental evaluation in Section 8.

```
forward <- function(A,B, pi_vec, obs){
  #init
  alpha <- piv * om[,obs[1]];

  #recursion
  for (i in 2:length(obs)){
    alpha <- (alpha %*% tm) *
              om[,obs[i]];
  }

  #termination
  sum(alpha)
}

#dimensions "n" passed via terminal
args = commandArgs(
  trailingOnly=TRUE);
if (length(args)==0) {
  stop("", call.=FALSE)
}
n <- as.numeric(args[1]);

#tm = transition matrix
tm <- matrix(as.matrix(
  fread("tm.csv",header=FALSE,
  sep=',')[,3],nrow=n);
om <- tm;

# random observation sequence o has
# been computed before
o <- c(124,84,195,40,122,142,
  13,61,7,125);
piv <- as.matrix(fread("pi.csv",
  sep=',')[,2];

ptm <- proc.time();
erg <- forward(tm,om,piv,o);
time_taken <- proc.time()-ptm;
```

AUTHOR BIOGRAPHIES



Dennis Marten received his M.Sc. in mathematics with emphasis in numerical analysis in 2012 at Rostock University. After two years of work in the wind energy industry he started as a doctoral student in 2014 at Rostock University. Dennis Marten is funded by the German

Research Foundation (DFG), Graduate School 1424 (Multimodal Smart Appliance Ensembles for Mobile Applications - MuSAMA).



Andreas Heuer, born 1958 in Uelzen (Germany), studied Mathematics and Computer Science at the Technical University of Clausthal from 1978 to 1984. He got his PhD and Habilitation at the TU Clausthal in 1988 and 1993, respectively. Since 1994, he is full professor for Database and Information Systems at the University of Rostock. Andreas

Heuer is interested in fundamentals of database models and languages, object-oriented databases and digital libraries, database support for assistive systems, and big data analytics, especially in the four P: performance, privacy, preservation and provenance.