DMtools – Open Source Software for Database Mining

Peter Christen, Ole M. Nielsen and Markus Hegland

Peter.Christen@anu.edu.au

Data Mining Group

Department of Computer Science, FEIT

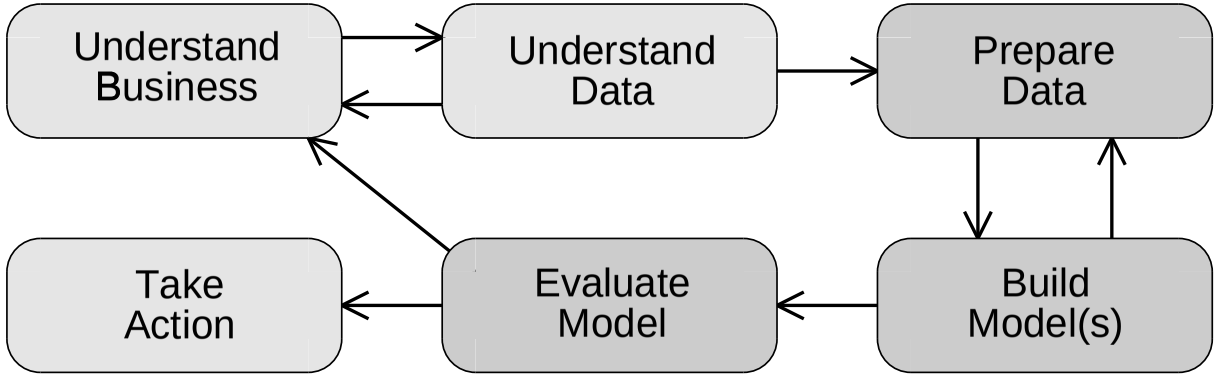
Australian National University, Canberra

http://csl.anu.edu.au/ml/dm/

Talk Outline

- The Data Mining Process
- Data Understanding and Exploration
- Requirements for Exploration Tools
- Related Approaches
- DMtools
  - The Choice of Software
  - Architecture
  - Data Manager and Caching
  - Some Examples
- Conclusions and Outlook

The Data Mining Process



- Much data mining research in algorithms (association rules, clustering, predictive modelling, etc.)
- Initial data exploration is also very important
- The first three phases can take up to 80% of the time and efforts spent in a data mining project
- Data mining is iterative and interactive

Data Understanding and Exploration

- Become familiar with the data and domain
  (ideally through interaction with client)
- Exploration through *ad-hoc* database querying
- Outcomes lead to new ideas and questions
- Interactive querying often prohibitively slow
- Caching of intermediate results needed

A flexible and easy to use toolbox can facilitate the data understanding and exploration phases

Requirements for Exploration Tools

- Facilitate interactive querying of the data
- Rapid code development
  (fast implementation of ideas as exploration evolves)
- Flexible data access and multiple, changing data formats (DBMS, text, binary, XML, Web, etc.)
- Scalable with data size and complexity
- Teamwork; sharing and reuse of data, results and code (Open Source)

Flexibility, caching of (intermediate) results and parallelism are needed

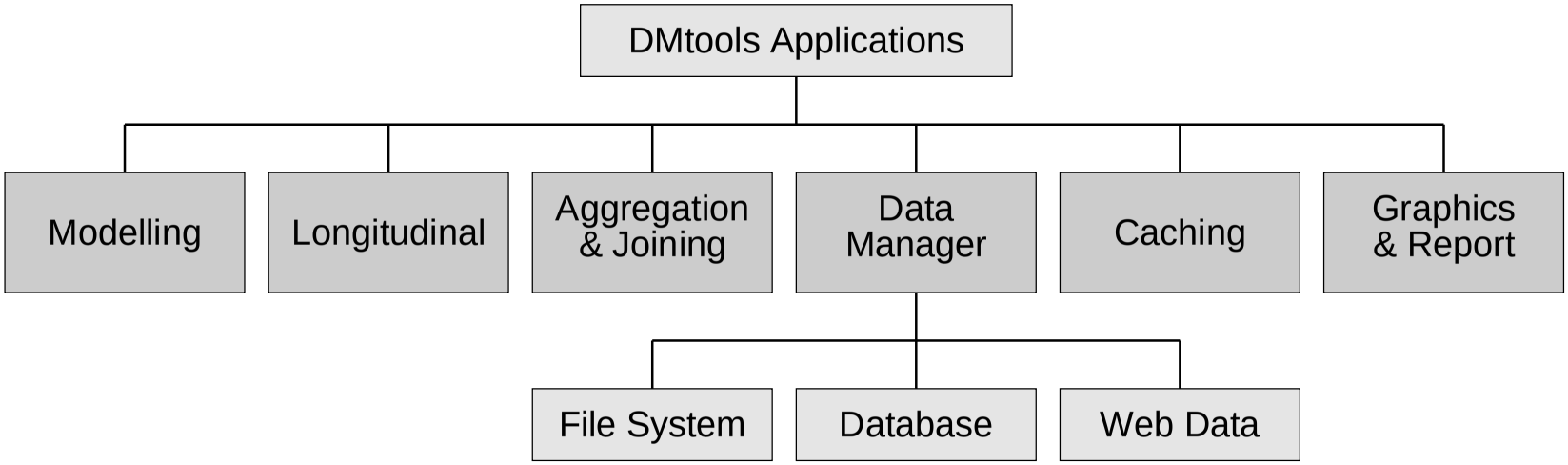
Related Approaches

- Data warehouses / OLAP
  (aggregation, summaries, drill-down, roll-up; precomputed)
- IDEA (Interactive Data Analysis and Exploration)
  (querying, segmentation, aggregation, external tools, history)
- Data Miner's Arcade
  (integration of data mining tools and formats, GUI)
- Database aware mining
  - Write data mining algorithms in SQL
  - Extend SQL with data mining constructs (DMQL)
  - Hide/encapsulate algorithms in database engines (black-box approach)

DMtools – Choice of Software

- Based entirely on Open Source Software
- Written in *Python*
  - Object-oriented scripting language
  - Can handle large data sets efficiently
  - Uses lists and dictionaries (hash-tables)
  - Large number of external modules available
  - Easy to extend with modules that call C
- Currently uses *MySQL* as database engine
- Graphics and reports with external modules (*Gnuplot*, *GDcharts*, *R*, *LaTeX*, *HTML* or *XML*)

Architecture of DMtools



Core modules Caching, Data Manager *and* Aggregation *are available as Open Source Software*

http://csl.anu.edu.au/ml/dm/dm_software.html

DMtools – Data Manager

- Deals with connection to SQL database engine and retrieving of data
- List of queries is parallelised *on the fly*
  (on a multiprocessor)
- Speedup depends on complexity of the queries
  (limited by slowest query in the list)
- Example:

```
q1 = 'select min(DATE_OF_SERVICE) from health1997'
q2 = 'select max(DATE_OF_SERVICE) from health1998'
q3 = 'select sum(BENEFITS) from health1999'
query_list = [q1, q2, q3]

res_list = exec_query(query_list, parallel=1, caching=1)
```

DMtools – Caching

- Supervised caching mechanism
- Trades space for speed
- Makes result of any function call persistent
- Assists in code development
- Shared use of cached results
- Compression of results

Caching is useful for computationally intensive functions with small results and few frequently used combinations of input arguments

DMtools – Caching (cont'd)

- Any given Python function can be replaced with:

```
T = func(arg_1, arg_2, ..., arg_n)
T = cache(func, (arg_1, arg_2, ..., arg_n))
```

- Caching statistics August – October 2000

Function Name	Hits	Time (sec)		Gain(%)
		Exec	Cache	
exec_query	8,185	138	7	91.68
get_cohort	532	338	< 1	97.30
get_selected_trans	815	1,560	4	99.55
get_drug_usage	167	1,389	< 1	99.99

Five users saved 918 hours of waiting

Health Mining Examples

- Get female patients for all doctors over three years

```
tables = ['health97', 'health98', 'health99']
selector = [('GENDER', 'female')]
result_dict = standard_breakdown('DOCTOR_ID', 'PATIENT_ID',
              selector, tables)
```

- Longitudinal counting of items of care for doctors

```
tables = ['health97', 'health98', 'health99']
result_dict = time_count('DOCTOR_ID', 'CARE_ITEM',
              'DATE_OF_SERVICE', 'MONTH', tables)
```

- Episode extraction for health care episodes

```
tables = ['health97', 'health98', 'health99']
result_dict = extract_episodes('DOCTOR_ID', 'PATIENT_ID',
              'DATE_OF_INIT', 'DATE_OF_SERVICE',
              'CARE_ITEM', tables)
```

Health Mining Examples (cont'd)

- Definition and usage of a *cohort*
  - Verbal definition:

    All individuals who have seen a psychiatrist in 1997 – 1999

  - Operational definition:

    Patient identifiers appearing in the database with items of care numbers 300 – 352 or 14224

  - *DMtools* definition:

```
tables = ['health97', 'health98', 'health99']
psych_patients = ('CARE_ITEM', [[300,352], 14224])
psych_cohort = get_cohort(psych_patients, tables)
```

DMtools hide complex SQL Queries

- Health mining example

```
tables = ['health97', 'health98', 'health99']
result_dict = time_count('DOCTOR_ID', 'CARE_ITEM',
                         'DOS', 'MONTH', tables)
```

  - Step 1: Get minimal and maximal values for time attribute (for all tables)

```
select min(DOS), max(DOS) from 'health97';
```

  - Step 2: Combine minimal and maximum values over tables and create empty vectors for time intervals
  - Step 3: Query database (for each table) to get counts:

```
select DOCTOR_ID, DOS, count(CARE_ITEM) from
       'health97' group by DOCTOR_ID, DOS;
```

  - Step 4: Insert results into interval vectors (aggregate)

Conclusions

- *DMtools* is a flexible and efficient toolbox for data exploration
- Computational time is saved through a supervised caching mechanism and parallel database querying
- Ease of use is achieved by hiding complex SQL queries within flexible and extendable Python functions and scripts
- Domain dependent data definitions through Python lists and dictionaries

Outlook: Ongoing Work

- Proper object-oriented model and framework
- Extension with more analysis functions
- Integration of a framework for predictive modelling algorithms
- Integration of (parallel) high-performance data mining algorithms

Visit our web site at:

http://csl.anu.edu.au/ml/dm/