Jump-starting the deep learning revolution on relational business data

AI Monday @ ExB Leipzig

Leipzig, 25.11.2019

alex@get.ml

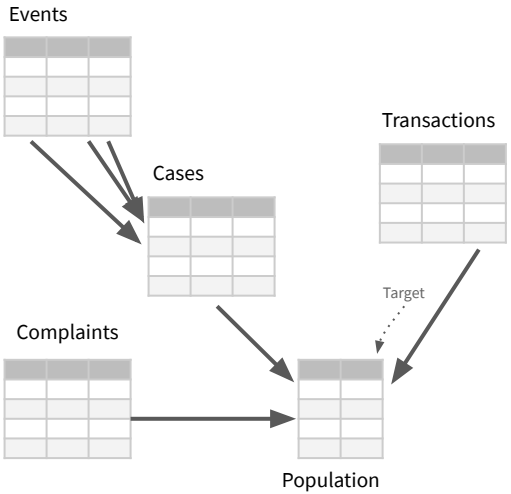
getML

Imaging you are a Data Scientist working in a bank: Predicting customer churn

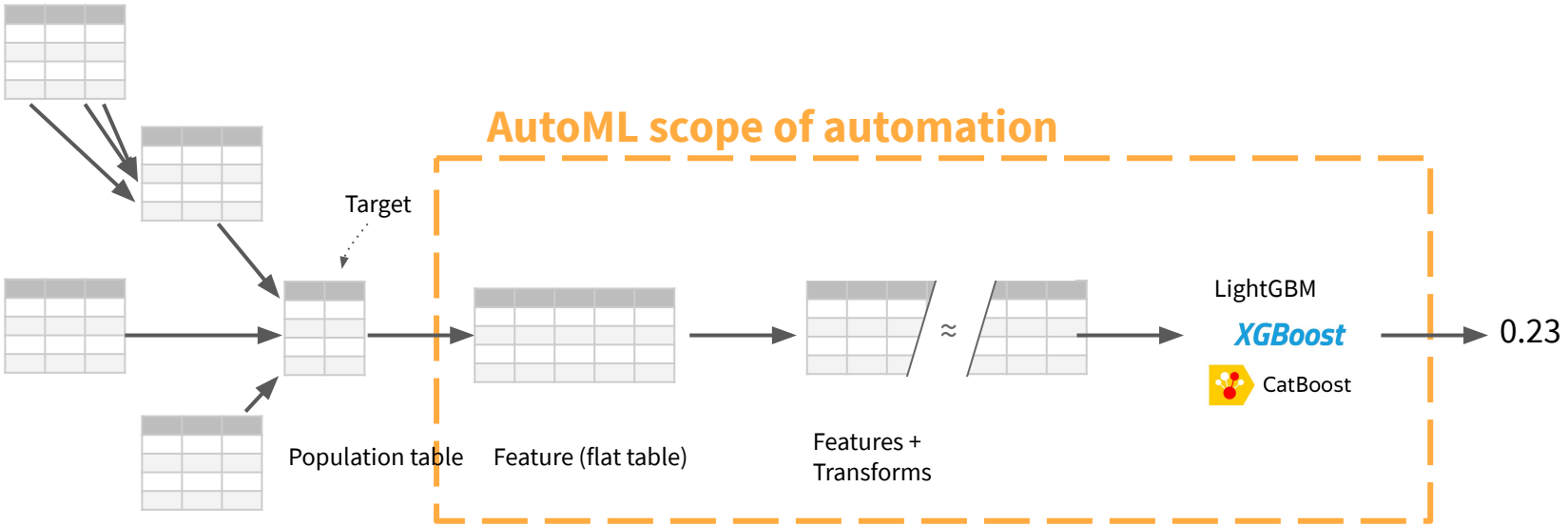
Classification on relational Data

Business Case

1. Predict which customer will likely close his account
2. Target the customer with favorable offers



Data Science Workflow with AutoML



RELATIONAL DATA MODEL & TARGET

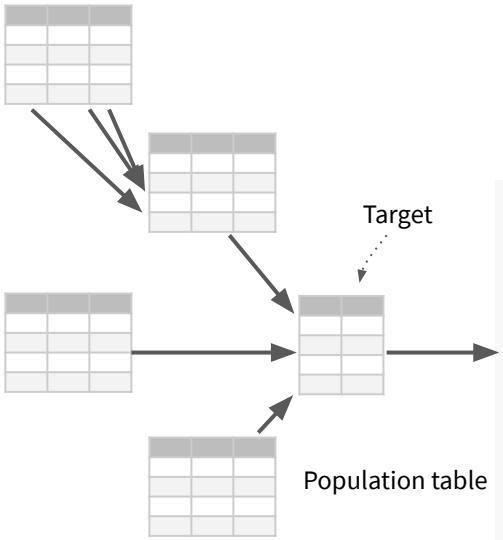
MERGED & AGGREGATED FLAT TABLE

NUMERICAL TRANSFORMS OR ENCODINGS

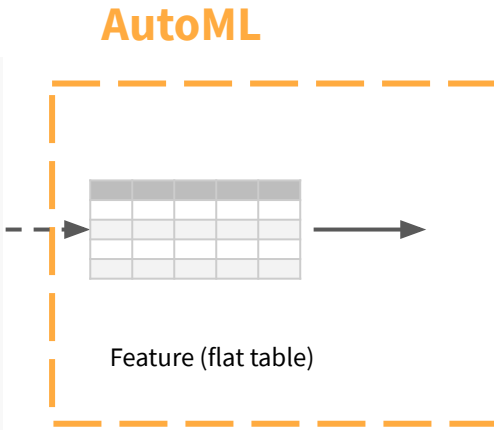
ML MODEL TRAINING

CHURN SCORE

AutoML is not enough




```sql
--Feature 235: Sum of customer transactions in last 90 days
SUM( t2.value )  AS sum_trans_90
FROM POPULATION_TABLE t1
LEFT JOIN TRANSACTIONS t2
ON t1.customer_id = t2.customer_id
WHERE
       t1.reference_date - t2.transaction_date <= 90.0
       AND t2.transaction_type IN ('sala', 'rent', 'cc')
GROUP BY t2.customer_id;
```

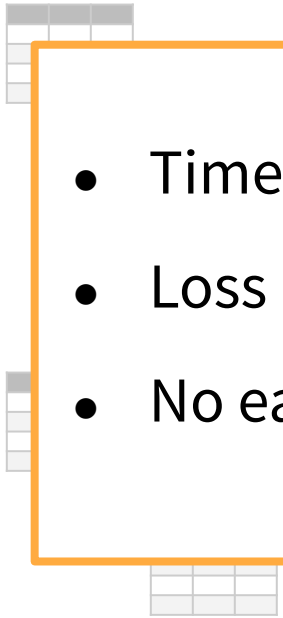


RELATIONAL DATA MODEL & TARGET

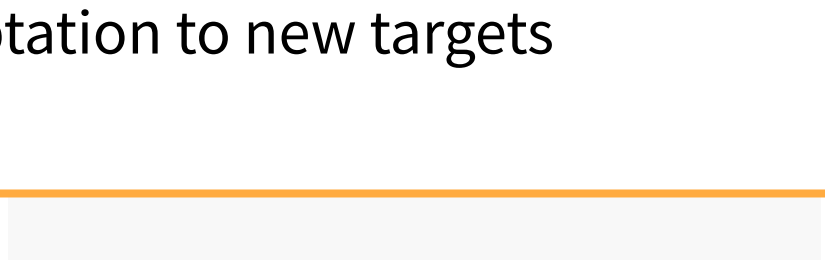
Manual Feature Engineering

MERGED & AGGREGATED FLAT TABLE

Problem with Manual Feature Engineering



- Time consuming to code & maintain
- Loss in accuracy due to missing information in flat table
- No easy adaptation to new targets



RAW DATA & DEFINITION OF TARGET

Feature Engineering in SQL

MERGED & AGGREGATED RAW DATA

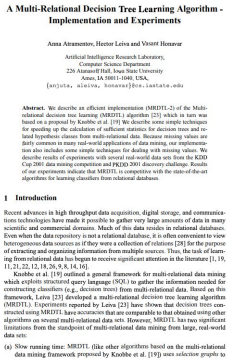
Researchers have forgotten about relational data

kaggle

Challenges usually start with **flat tables**

python™ Package Index

Brute force based implementation in python



A Multi-Relational Decision Tree Learning Algorithm - Implementation and Experiments

Anna Atramentov, Hector Leiva and Vasant Honavar

Artificial Intelligence Research Laboratory,
Computer Science Department
226 Atanasoff Hall, Iowa State University
Ames, IA 50011-1040, USA,
{anjuta, aleiva, honavar}@cs.iastate.edu

Abstract. We describe an efficient implementation (MRDTL-2) of the Multi-relational decision tree learning (MRDTL) algorithm [23] which in turn was based on a proposal by Knobbe et al. [19] We describe some simple techniques for speeding up the calculation of sufficient statistics for decision trees and related hypothesis classes from multi-relational data. Because missing values are fairly common in many real-world applications of data mining, our implementation also includes some simple techniques for dealing with missing values. We describe results of experiments with several real-world data sets from the KDD Cup 2001 data mining competition and PKDD 2001 discovery challenge. Results of our experiments indicate that MRDTL is competitive with the state-of-the-art algorithms for learning classifiers from relational databases.

1 Introduction

Recent advances in high throughput data acquisition, digital storage, and communications technologies have made it possible to gather very large amounts of data in many scientific and commercial domains. Much of this data resides in relational databases. Even when the data repository is not a relational database, it is often convenient to view heterogeneous data sources as if they were a collection of relations [28] for the purpose of extracting and organizing information from multiple sources. Thus, the task of learning from relational data has begun to receive significant attention in the literature [1, 19, 11, 21, 22, 12, 18, 26, 9, 8, 14, 16].

Knobbe et al. [19] outlined a general framework for multi-relational data mining which exploits structured query language (SQL) to gather the information needed for constructing classifiers (e.g., decision trees) from multi-relational data. Based on this framework, Leiva [23] developed a multi-relational decision tree learning algorithm (MRDTL). Experiments reported by Leiva [23] have shown that decision trees constructed using MRDTL have accuracies that are comparable to that obtained using other algorithms on several multi-relational data sets. However, MRDTL has two significant limitations from the standpoint of multi-relational data mining from large, real-world data sets:

(a) Slow running time: MRDTL (like other algorithms based on the multi-relational data mining framework proposed by Knobbe et al. [19]) uses selection graphs to

MRDTL (Atramentov **2003**) most recent research

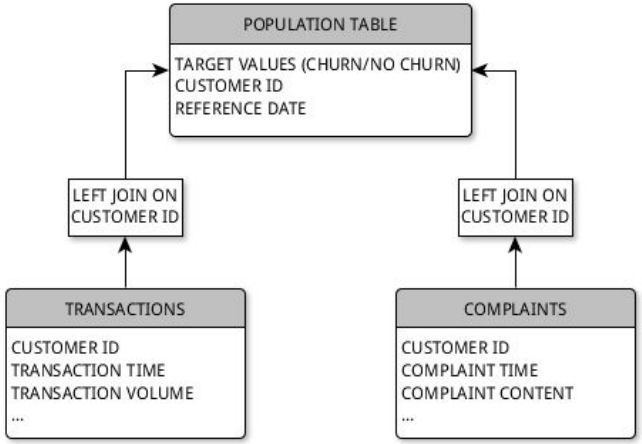
Wish list: Automated Feature Engineering

Objective

Automatically generate flat table from relational data model

Prerequisite

- Data model
- Target variable
- No expert knowledge



Gradient Boosting For Feature Engineering

Core idea of our **MultiRel** algorithm

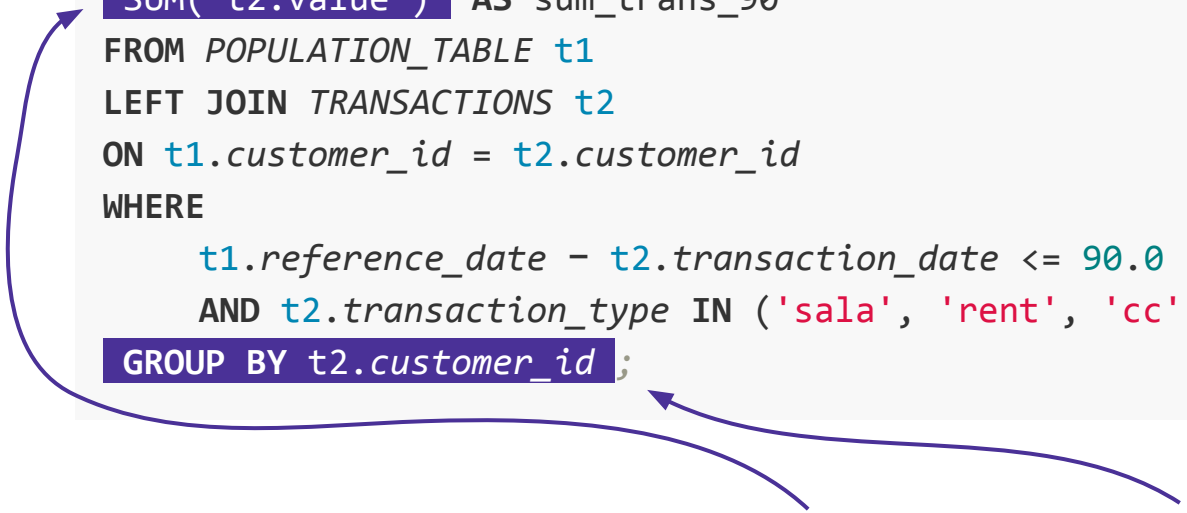
Incremental Updates of Aggregations

+

Ensemble learning

MultiRel tests aggregations over different join keys

```
--Feature 235: Sum of customer transactions in last 90 days
 SUM( t2.value )  AS sum_trans_90
FROM POPULATION_TABLE t1
LEFT JOIN TRANSACTIONS t2
ON t1.customer_id = t2.customer_id
WHERE
     t1.reference_date - t2.transaction_date <= 90.0
     AND t2.transaction_type IN ('sala', 'rent', 'cc')
 GROUP BY t2.customer_id ;
```



Finds optimal (column) **aggregation** function on **join key** level.

MultiRel learns and combines conditions

```
--Feature 235: Sum of customer transactions in last 90 days
SUM( t2.value )  AS sum_trans_90
FROM POPULATION_TABLE t1
LEFT JOIN TRANSACTIONS t2
ON t1.customer_id = t2.customer_id
WHERE
     t1.reference_date - t2.transaction_date <= 90.00
     AND t2.transaction_type IN ('sala', 'rent', 'cc')
GROUP BY t2.customer_id;
```

Combination of multiple **numerical** and **categorical** conditions

getML python API workflow

```
import getml

population_placeholder = getml.models.Placeholder(
 name="POPULATION_TABLE",
 targets=TARGET,
 discrete=DISCRETE)

transactions_placeholder = getml.models.Placeholder(
 name="TRANSACTIONS",
 numerical=NUMERICAL)

population_placeholder.join(transactions_placeholder,
 "join_key", "time_stamp")

model = getml.models.MultirelModel(
 aggregation=[getml.aggregations.Count, getml.aggregations.Sum],
 population=population_placeholder,
 peripheral=[transactions_placeholder],
 predictor=getml.predictors.XGBoostRegressor(),
 num_features=100).send()

model = model.fit(  ...train_data... )
scores = model.score(  ...validation_data... )
```

Define the **data model**

Set **target variable**

Automated feature extraction with **no expert knowledge**

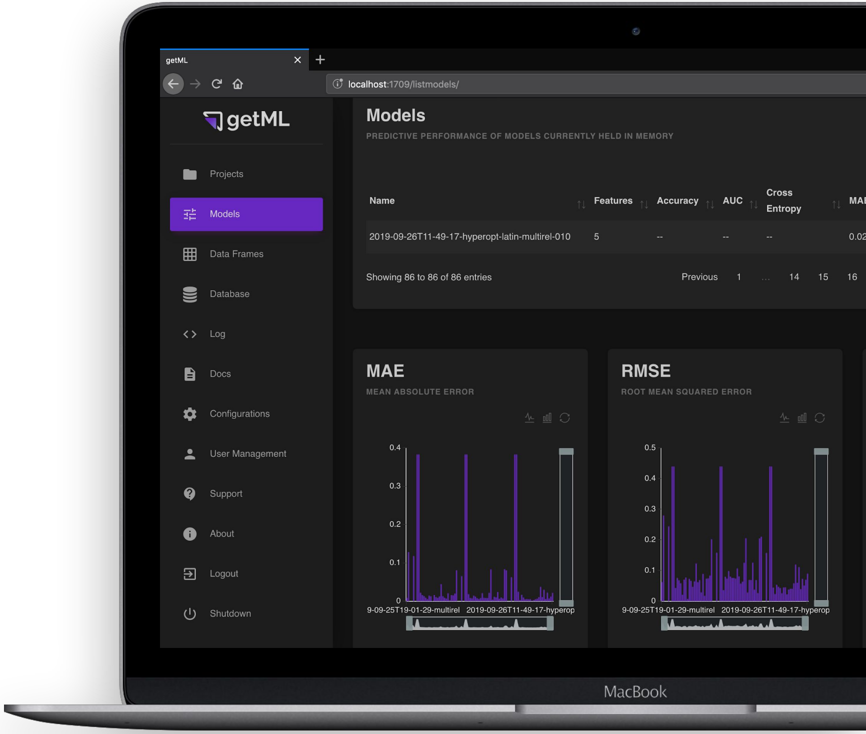
Example feature generated with MultiRel

```
SELECT COUNT( * ) - COUNT( DISTINCT t1.TIME_STAMP - t2.TIME_STAMP ) AS feature_286,
       t1.BASKETID,
       t1.TIME_STAMP
FROM (
     SELECT *,
            ROW_NUMBER() OVER ( ORDER BY BASKETID, TIME_STAMP ASC ) AS rownum
     FROM EXPD
) t1
LEFT JOIN EXPD t2
ON t1.BASKETID = t2.BASKETID
WHERE (( t1.UCC NOT IN ( '180320', '330410', '270000', '390120', '600210', '170533', '430120', '340410', '130121', '370314', '390210', '410120', '410901', '330610', '610110', '690119', '600420', '620112', '030610', '004100', '004190' ) AND
t2.UCC NOT IN ( '190324', '270210', '210110', '380333', '360312', '360420', '010120', '020410', '002100' ) AND t2.UCC IN ( '360350', '350110', '130121', '320430', '520110', '620420', '380311', '560400', '620510', '600900', '020620', '001000' )
AND t1.UCC NOT IN ( '110410', '110510', '120310', '120410', '130212', '140110', '170210', '190111', '190211', '320140', '330510', '100210', '100510', '110210', '150110', '180220', '180710', '330210', '190112', '190113', '470111', '630110',
'190114', '190212', '190321', '560110', '190311', '100110', '330310', '170310', '640310', '380315', '520531', '180611', '540000', '200532', '640410', '320232', '520110', '560400', '320521', '010210', '090110', '090210', '020110', '020210',
'020310', '010320', '020510', '040410', '070110', '004000' ) )
OR ( t1.UCC NOT IN ( '180320', '330410', '270000', '390120', '600210', '170533', '430120', '340410', '130121', '370314', '390210', '410120', '410901', '330610', '610110', '690119', '600420', '620112', '030610', '004100', '004190' ) AND t2.UCC IN
( '190324', '270210', '210110', '380333', '360312', '360420', '010120', '020410', '002100' ) AND t1.UCC IN ( '180310', '320410', '330510', '620912', '650210', '170520', '180510', '280120', '320233', '190313', '270310', '190322', '240320',
'550410', '320904', '340110', '280140', '320345', '340901', '640410', '590230', '380340', '610320', '130211', '140340', '320130', '470220', '360311', '620330', '530311', '590110', '660110', '380901', '340530', '520110', '180620', '670210',
'310232', '620710', '630220', '620930', '090210', '020710', '070230', '050310', '030410', '030510' ) )
OR ( t1.UCC IN ( '180320', '330410', '270000', '390120', '600210', '170533', '430120', '340410', '130121', '370314', '390210', '410120', '410901', '330610', '610110', '690119', '600420', '620112', '030610', '004100', '004190' ) AND t2.UCC NOT IN
( '110410', '120410', '180320', '180410', '190211', '320140', '320410', '330410', '330510', '100210', '100510', '110210', '180420', '320233', '320370', '190112', '330110', '470111', '190313', '190114', '560110', '190311', '180520', '320905',
'640310', '660000', '370125', '390120', '340110', '190312', '200532', '390310', '640410', '200210', '380340', '200310', '200410', '360311', '370220', '370314', '340520', '440210', '410120', '410901', '380901', '620926', '300218', '340210',
'320232', '610110', '380410', '180620', '240120', '320380', '640120', '620420', '010120', '040110', '010320', '040510', '050310', '004000', '004190' ) AND t1.UCC NOT IN ( '600210', '340410', '410120', '690119', '600420', '620112', '004190' ) AND
t2.UCC IN ( '280210', '370213', '620214', '160110', '360350', '470211', '160320', '520531', '430120', '690120', '130110', '590230', '620310', '140340', '380333', '360312', '620121', '620111', '230110', '660110', '660210', '660900', '380210',
'550320', '320430', '680903', '380320', '370110', '620213', '320521', '290440', '620710', '380510', '690230', '020610', '020620', '020510', '030810', '040410', '070110', '030210' ) )
OR ( t1.UCC IN ( '180320', '330410', '270000', '390120', '600210', '170533', '430120', '340410', '130121', '370314', '390210', '410120', '410901', '330610', '610110', '690119', '600420', '620112', '030610', '004100', '004190' ) AND t2.UCC NOT IN
( '110410', '120410', '180320', '180410', '190211', '320140', '320410', '330410', '330510', '100210', '100510', '110210', '180420', '320233', '320370', '190112', '330110', '470111', '190313', '190114', '560110', '190311', '180520', '320905',
'640310', '660000', '370125', '390120', '340110', '190312', '200532', '390310', '640410', '200210', '380340', '200310', '200410', '360311', '370220', '370314', '340520', '440210', '410120', '410901', '380901', '620926', '300218', '340210',
'320232', '610110', '380410', '180620', '240120', '320380', '640120', '620420', '010120', '040110', '010320', '040510', '050310', '004000', '004190' ) AND t1.UCC IN ( '600210', '340410', '410120', '690119', '600420', '620112', '004190' ) )
OR ( t1.UCC IN ( '180320', '330410', '270000', '390120', '600210', '170533', '430120', '340410', '130121', '370314', '390210', '410120', '410901', '330610', '610110', '690119', '600420', '620112', '030610', '004100', '004190' ) AND t2.UCC IN (
'110410', '120410', '180320', '180410', '190211', '320140', '320410', '330410', '330510', '100210', '100510', '110210', '180420', '320233', '320370', '190112', '330110', '470111', '190313', '190114', '560110', '190311', '180520', '320905',
'640310', '660000', '370125', '390120', '340110', '190312', '200532', '390310', '640410', '200210', '380340', '200310', '200410', '360311', '370220', '370314', '340520', '440210', '410120', '410901', '380901', '620926', '300218', '340210',
'320232', '610110', '380410', '180620', '240120', '320380', '640120', '620420', '010120', '040110', '010320', '040510', '050310', '004000', '004190' ) AND t2.UCC4 NOT IN ( '1104', '1204', '1803', '1901', '1902', '3305', '1002', '1005', '1102',
'3301', '4701', '6403', '1806', '2005', '3903', '6404', '4101', '3804', '3809', '3402', '6101', '0101', '0103', '0405', '0503', '0040' ) )
OR ( t1.UCC IN ( '180320', '330410', '270000', '390120', '600210', '170533', '430120', '340410', '130121', '370314', '390210', '410120', '410901', '330610', '610110', '690119', '600420', '620112', '030610', '004100', '004190' ) AND t2.UCC IN (
'110410', '120410', '180320', '180410', '190211', '320140', '320410', '330410', '330510', '100210', '100510', '110210', '180420', '320233', '320370', '190112', '330110', '470111', '190313', '190114', '560110', '190311', '180520', '320905',
'640310', '660000', '370125', '390120', '340110', '190312', '200532', '390310', '640410', '200210', '380340', '200310', '200410', '360311', '370220', '370314', '340520', '440210', '410120', '410901', '380901', '620926', '300218', '340210',
'320232', '610110', '380410', '180620', '240120', '320380', '640120', '620420', '010120', '040110', '010320', '040510', '050310', '004000', '004190' ) AND t2.UCC4 IN ( '1104', '1204', '1803', '1901', '1902', '3305', '1002', '1005', '1102',
'3301', '4701', '6403', '1806', '2005', '3903', '6404', '4101', '3804', '3809', '3402', '6101', '0101', '0103', '0405', '0503', '0040' ) AND t2.UCC IN ( '180320', '100210', '640310', '200532', '390310', '010120', '004000' ) )
)
t2.TIME_STAMP <= t1.TIME_STAMP
GROUP BY t1.rownum,
         t1.BASKETID,
         t1.TIME_STAMP;
```

Using getML in your own projects

Download getML at

https://get.ml



Thank you for your attention. Questions?

Alexander Uhlig
Co-founder & Data Scientist

alex@get.ml

getML is a brand of **The SQLNet Company GmbH**

Offices: Spinnereistraße 7, c/o SpinLab Halle 14, 04179 Leipzig, Germany I
Registration Court: Amtsgericht Leipzig I Commercial Register No.: HRB 34030 I
VAT No.: DE313712523 | Managing Directors: Alexander Uhlig, Dr. Patrick Urbanke