SCALING FEATURE GENERATION FROM PROTOTYPING TO PRODUCTION AT REWE
AGENDA

1 / Introduction
2 / Example Project: Predicting Brand Market Fit
3 / Feature Generation in Prototyping – The Problems
4 / Moving From Prototyping to Production – Lessons Learned
INTRODUCTION / ME

Benjamin Greve
Data Scientist at REWE Systems

- Mathematician
- Working in data science projects for 5+ years (2 years at REWE)
- Likes hiking, climbing, dancing, cooking (especially the eating part)
INTRODUCTION / REWE GROUP

REWE Group facts (2017)

TOTAL REVENUE 2017

345,000

345,000

EMPLOYEES 2017

15,300

MARKETS 2017

57.8

57.8

Billion Euro

Billion Euro

2017

2017

2017

2017
INTRODUCTION / REWE SYSTEMS

- >1 Bil. data sets every day
- 1,200 IT specialists
- 30,000 cash registers
- 30 locations
- 7,500 markets
- 200,000 users

March 2019
EXAMPLE PROJECT: PREDICTING BRAND MARKET FIT

We want to predict how well a brand will be received in a particular market to assist category managers in selecting the right brands for each market.

BRAND / CATEGORY:
A set of products that form a logical group. Can contain between one and a few thousand products.

TARGET VARIABLE y:
Brand popularity in current market compared to average across all REWE markets

FEATURES x₁, x₂, ...
- Popularity of wider category in market
- Number of competing articles
- Location of market, incl. demographical information
- …

Technical definition

Not an actual project – just for illustration purposes
FEATURE GENERATION IS CENTRAL PART IN THE DATA SCIENCE WORKFLOW

CRISP-DM: Cross-industry standard process for data mining

A typical result of prototyping is a very long SQL script.

OUTPUT #1: Very long SQL script
- 2000+ lines of SQL
- 40+ intermediate tables
- interdependencies
- inconsistent naming, chaotic code style
- not optimized for performance
- not robust
→ not production-ready or scalable

OUTPUT #2: Very long R or Python script
- Different topic for another talk

Prototyping performed by Data Scientists
A TYPICAL RESULT OF PROTOTYPING IS A VERY LONG SQL SCRIPT

OUTPUT #1: Very long SQL script

- 2000+ lines of SQL
- 40+ intermediate tables
- interdependencies
- inconsistent naming, chaotic code style
- not optimized for performance
- not robust

→ not production-ready or scalable
AVOID LONG, MONOLITHIC SQL SCRIPTS

Weaknesses:

- Duplicate/redundant SQL for every feature
- Parameters hidden within the scripts
- Adding new features is hard due to interdependencies
USE A MODULAR FEATURE GENERATION INSTEAD

Strengths:
- Highly modular and scalable
- Parameters are centrally defined
- Feature scripts are mostly independent, making it easy to add new features

Input script
- Creates input table
  - i.e. table of all market-brand-combinations for which to calculate features

Derived helper tables
- Distinct list of brands
- Mapping brands to articles
- ...

Feature scripts A, B, C

merge features into feature store

Feature Store

| Market  | Brand | Feature A1 | Feature A2 | Feature B1 |...
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>10000001</td>
<td>20001</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10000002</td>
<td>20001</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10000003</td>
<td>20001</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
DECOMPOSING COMPLEX SQL WITHIN KNIME LEADS TO COMPLEX WORKFLOWS

✓ Code
✓ Powerful version control (releases)
✓ Analysts fluent in SQL
  • Harder to explain/show

✓ Visual, KNIME native
  • Gets too complex as SQL logic grows
  • Hard to translate into standard data warehouse ETLs

Example workflow from KNIME Examples server
A DEPLOYMENT WORKFLOW PULLS SQL SCRIPTS FROM VERSION CONTROL TO KNIME SERVER

A fully automated deployment workflow copies snapshot or release versions from the version control system to the KNIME Server.
To perform Feature Generation, execute a set of SQL scripts in a given order.

Just write down the name of the files to be executed in the correct order and run the workflow.
Useful metanodes help to implement a staging concept, e.g.:

Select staging environment \(\rightarrow\) obtain connection to the corresponding database
Currently:

- specify scripts manually
- order manually to take dependencies into account

Next level:

- specify features in arbitrary order

Feature dependency graph
- Automatic dependency resolution
- Already built into KNIME!
Thanks for your attention