Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

Track 1 – Matrix Factorization

Implementation Details

Collaborative Filtering

Markus Freitag, Jan-Felix Schwarz

9 June 2011

Agenda

1. Recap
2. Roadmap & Implementation Status
3. SGD Implementation: Performance Details
4. Implementation of Biases
5. Algorithm Parameters
6. Submissions & RMSEs
7. Implementation Outlook
8. Discussion

Recap: Matrix Factorization

$$
\begin{array}{cc}
 & \textbf{items} \\
\textbf{users} & \begin{pmatrix}
10 & 50 & & & 0 & 0 & & 25 \\
 & & 0 & & 100 & & & 0 \\
100 & & 80 & & & & & \\
0 & & & 50 & 30 & & & \\
 & & & & & 20 & & 0 \\
20 & 100 & & 0 & & 50 & 60 & \\
 & & 0 & & 90 & & & \\
10 & 50 & & 20 & & & & \\
 & & & & 90 & 100 & & 20 \\
 & & & 20 & & & 100 & 0 \\
0 & & 0 & & 40 & & & \\
 & & 70 & 10 & & & 10 &
\end{pmatrix}
\end{array}
$$

Recap: Matrix Factorization

item features · **user features**

$$
\begin{pmatrix}
5.1 & 10.0 & 21.3 \\
4.7 & 9.2 & 1.9 \\
0.0 & 21.9 & 14.7 \\
7.9 & 8.5 & 40.2 \\
10.1 & 0.2 & 2.9 \\
9.1 & 8.1 & 8.7 \\
16.6 & 20.1 & 4.1 \\
7.8 & 1.0 & 0.1
\end{pmatrix}^{T}
\bullet
\begin{pmatrix}
1.9 & 20.1 & 9.4 \\
23.1 & 0.1 & 4.2 \\
10.2 & 4.0 & 1.9 \\
1.2 & 0.7 & 12.2 \\
7.3 & 9.3 & 13.7 \\
6.3 & 28.1 & 7.2 \\
9.0 & 5.3 & 3.2 \\
5.2 & 11.1 & 12.0 \\
5.7 & 3.9 & 2.7 \\
0.3 & 0.0 & 0.1 \\
6.7 & 21.2 & 0.0 \\
6.4 & 7.9 & 3.2
\end{pmatrix}
=
\begin{pmatrix}
10 & 50 & & & 0 & 0 & & 25 \\
 & & 0 & & 100 & & & 0 \\
100 & & 80 & & & & & \\
0 & & & 50 & 30 & & & \\
 & & & & & 20 & & 0 \\
20 & 100 & & 0 & & 50 & 60 & \\
 & & 0 & & 90 & & & \\
10 & 50 & & 20 & & & & \\
 & & & & 90 & 100 & & 20 \\
 & & & 20 & & & 100 & 0 \\
0 & & 0 & & 40 & & & \\
 & & 70 & 10 & & & 10 &
\end{pmatrix}
$$

Recap: Matrix Factorization

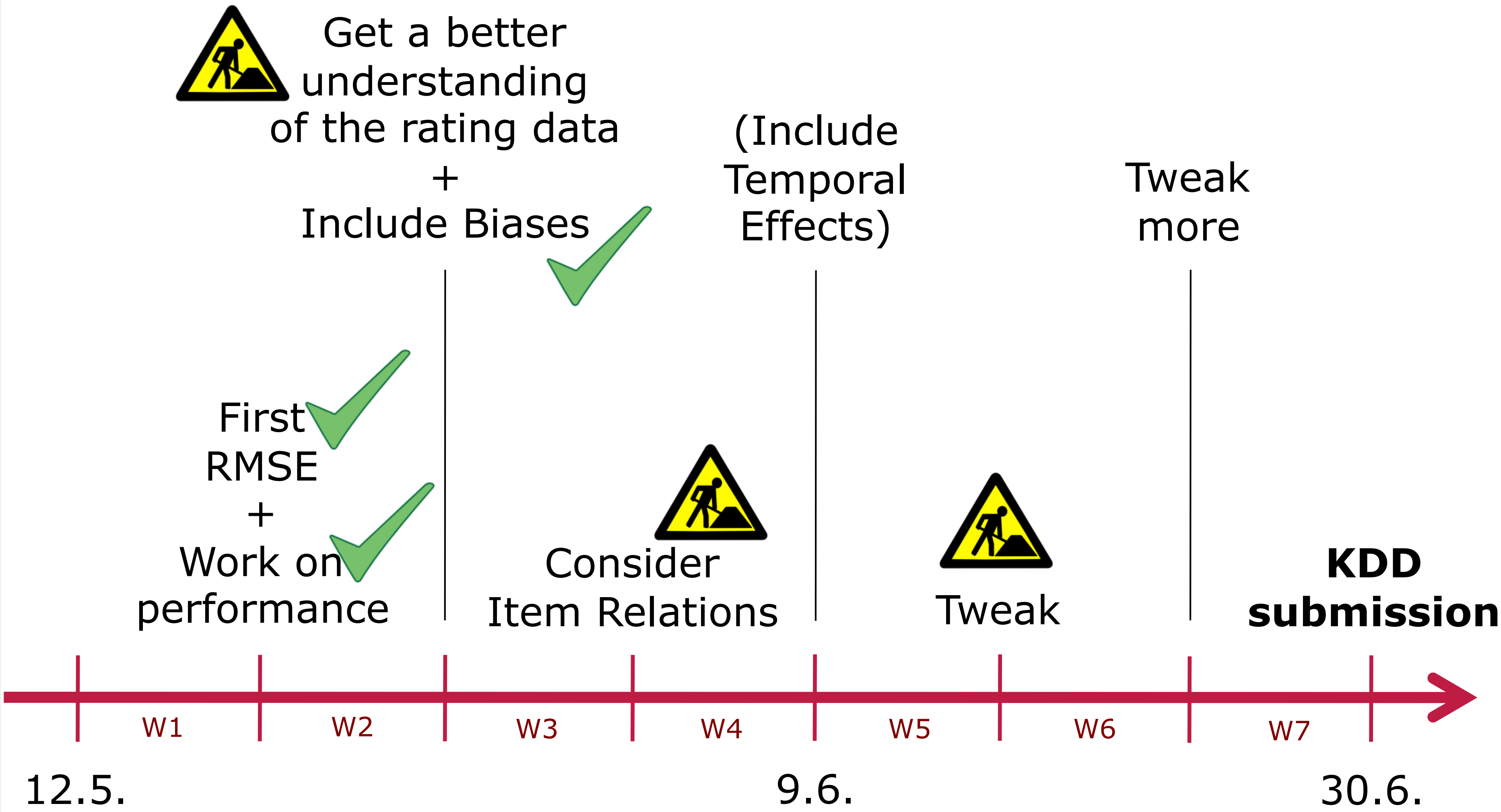
$$
\begin{array}{cccc}
\textbf{item features} & & \textbf{user features} & \\
\begin{pmatrix}
5.1 & 10.0 & 21.3 \\
4.7 & 9.2 & 1.9 \\
0.0 & 21.9 & 14.7 \\
7.9 & 8.5 & 40.2 \\
10.1 & 0.2 & 2.9 \\
9.1 & 8.1 & 8.7 \\
16.6 & 20.1 & 4.1 \\
7.8 & 1.0 & 0.1
\end{pmatrix}^{T}
& \bullet &
\begin{pmatrix}
1.9 & 20.1 & 9.4 \\
23.1 & 0.1 & 4.2 \\
10.2 & 4.0 & 1.9 \\
1.2 & 0.7 & 12.2 \\
7.3 & 9.3 & 13.7 \\
6.3 & 28.1 & 7.2 \\
9.0 & 5.3 & 3.2 \\
5.2 & 11.1 & 12.0 \\
5.7 & 3.9 & 2.7 \\
0.3 & 0.0 & 0.1 \\
6.7 & 21.2 & 0.0 \\
6.4 & 7.9 & 3.2
\end{pmatrix}
& =
\begin{pmatrix}
10 & 50 & 80 & 90 & 0 & 0 & 100 & 25 \\
70 & 55 & 0 & 90 & 100 & 15 & 10 & 0 \\
100 & 76 & 80 & 90 & 10 & 30 & 20 & 0 \\
0 & 90 & 10 & 50 & 30 & 90 & 100 & 10 \\
0 & 10 & 100 & 70 & 40 & 20 & 10 & 0 \\
20 & 100 & 100 & 0 & 10 & 50 & 60 & 90 \\
80 & 20 & 0 & 80 & 90 & 76 & 0 & 10 \\
10 & 50 & 90 & 20 & 10 & 90 & 100 & 10 \\
0 & 0 & 10 & 50 & 90 & 100 & 40 & 20 \\
60 & 70 & 50 & 20 & 90 & 10 & 100 & 0 \\
0 & 10 & 0 & 90 & 40 & 20 & 50 & 30 \\
40 & 80 & 70 & 10 & 100 & 0 & 10 & 10
\end{pmatrix}
\end{array}
$$

Recap: SGD Algorithm

Stochastic Gradient Descent (SGD)

- Approximation procedure for learning one feature
- For each rating in the training set the feature values are modified relative to the prediction error
  - *User value += Learning Rate * Error * Item value*
  - *Item value += Learning Rate * Error * User Value*
- Iterate over the training set until the sum of squared errors (SSE) converges
- Training set split into 4 subsets (track, album, artist, genre)
  - Don’t presume a common underlying model

Roadmap & Implementation Status



SGD Algorithm: Memory

- Ratings are read once, then cached in memory
  - For largest subset: ~120MIO * 3 * Integer = **1.44 GB**
- Learned vectors are persisted every 10 dimensions
  - Save intermediate results, limit number of UPDATE statements
  - User vectors: ~1MIO * 10 * Float = **40 MB**
  - Item vectors: ~625K * 10 * Float = **25 MB**
- Optimization: store interims of dot products
  - For largest subset: ~120MIO * Float = **480 MB**
- Cache biases: ~(1MIO + 625K) * Float = **6.5 MB**

- **Total** (for track set)**:** **2 GB**

SGD Algorithm: CPU

- Feature values are stored in (sparse) arrays
  - Constant access times (ID is index)
  - Only 6.5 MB needed per feature
- Optimizations of operations
  - E.g.: `error * error` instead of `Math.pow(error, 2)`
- Avoid object instantiation, use primitive data types
- Memory consumption leaves room for parallelization
  - One process per type → use up to 4 cores
  - Requires no implementation effort

Biases

- A Bias is a user- or item-specific offset
  - Items that are constantly rated higher than others
  - Users who rate mostly critical

- Estimation is based on training ratings
- Biases for each user and each item
- Biases are precalculated

Calculation of Biases

$$\boldsymbol{Baseline_{u,i} = TypeAvg + UserBias_u + ItemBias_i}$$

- $UserBias_u = UserAvg_u - GlobalAvg$
- $ItemBias_i = ItemAvg_i - TypeAvg$

- Calculated for each item type

Algorithm Parameters

- Learning rate
  - Almost never changed
- Limit for the number of iterations
  - Tried many between 1 and 2500
  - Not fix anymore
- Improvement threshold for sum of squared errors (SSE)
  - Very low in the beginning (0.1)
  - Changed to 10,000 (fast but too high)
  - Currently set to 0.001%
- Number of features
  - Started with only 1
  - 10/20/40 already tested (10 works best)
  - 100 is meant to be a good value

Submissions & RMSEs

Starting with one feature

#	Description	LR	Iterations	RMSE
1	Test with 50	-	-	37.8262
2	First complete run	0.01	100	28.3295
5	With biases	0.001	446	28.9182
6	Used item types	0.001	100	29.6343

Submissions & RMSEs (ctd.)

Using more features

#	Description	LR	Features	RMSE
7	Test with more f.	0.002	10	27.3462
9	20 features	0.002	20	27.5172
11	40 features	0.002	40	28.0550
12	With validation set	0.002	10	26.5217

Outlook: Improve Biases

- Idea: best guess is a linear blend between the user/item mean and the global mean
  - → Better baseline for items/users with few ratings in the training set

- *V_a: Variance of all the items' average ratings*
- *V_b: Variance of individual item ratings*
- *$K = V_a / V_b$*

$$BetterItemAvg = K * GlobalAvg + \frac{sum(ObservedRatings)}{K * count(ObservedRating)}$$

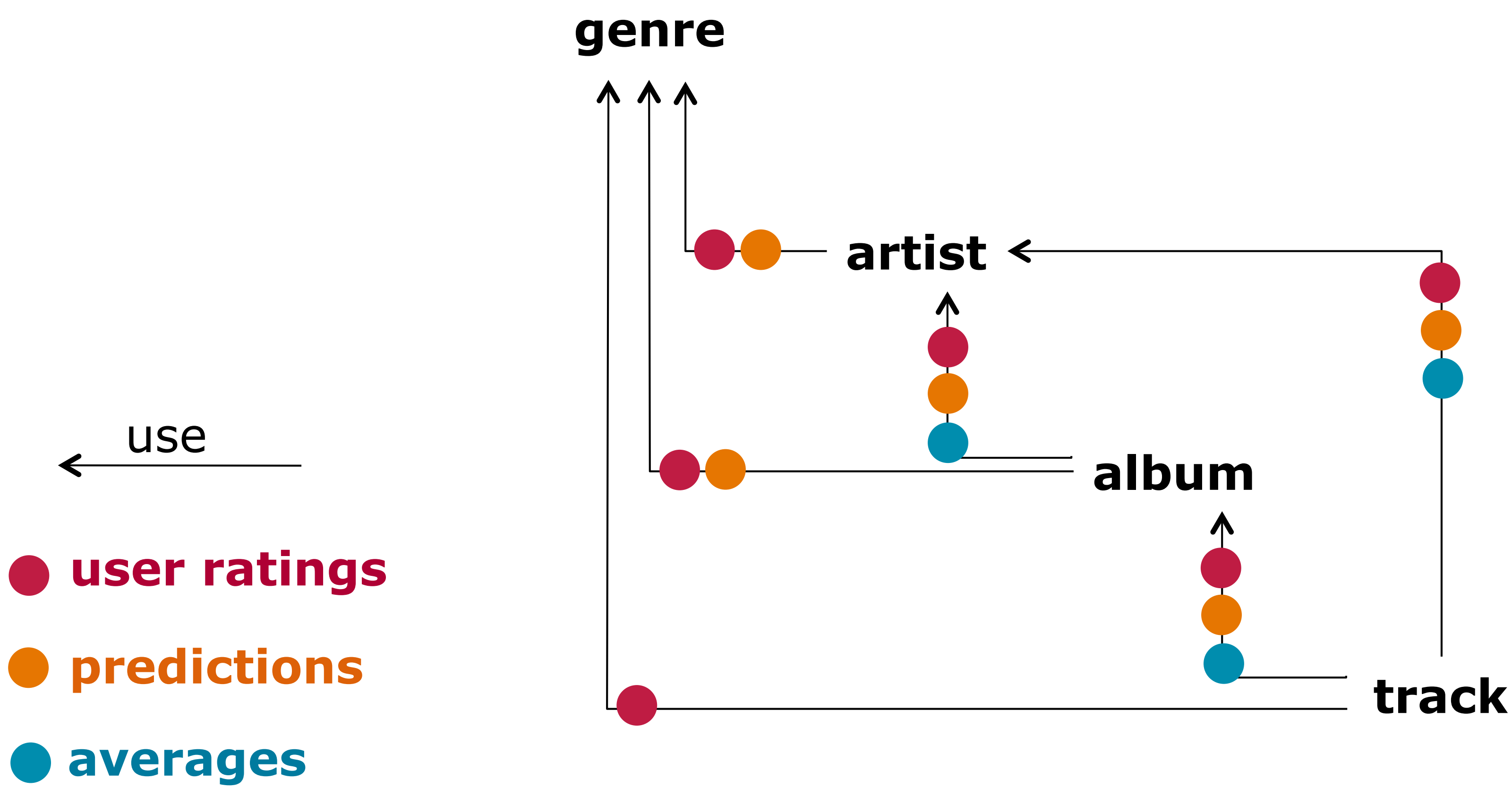
Outlook: Make Use of Hierarchy

- For now we have 4 different prediction models
- Combine these models to improve predictions

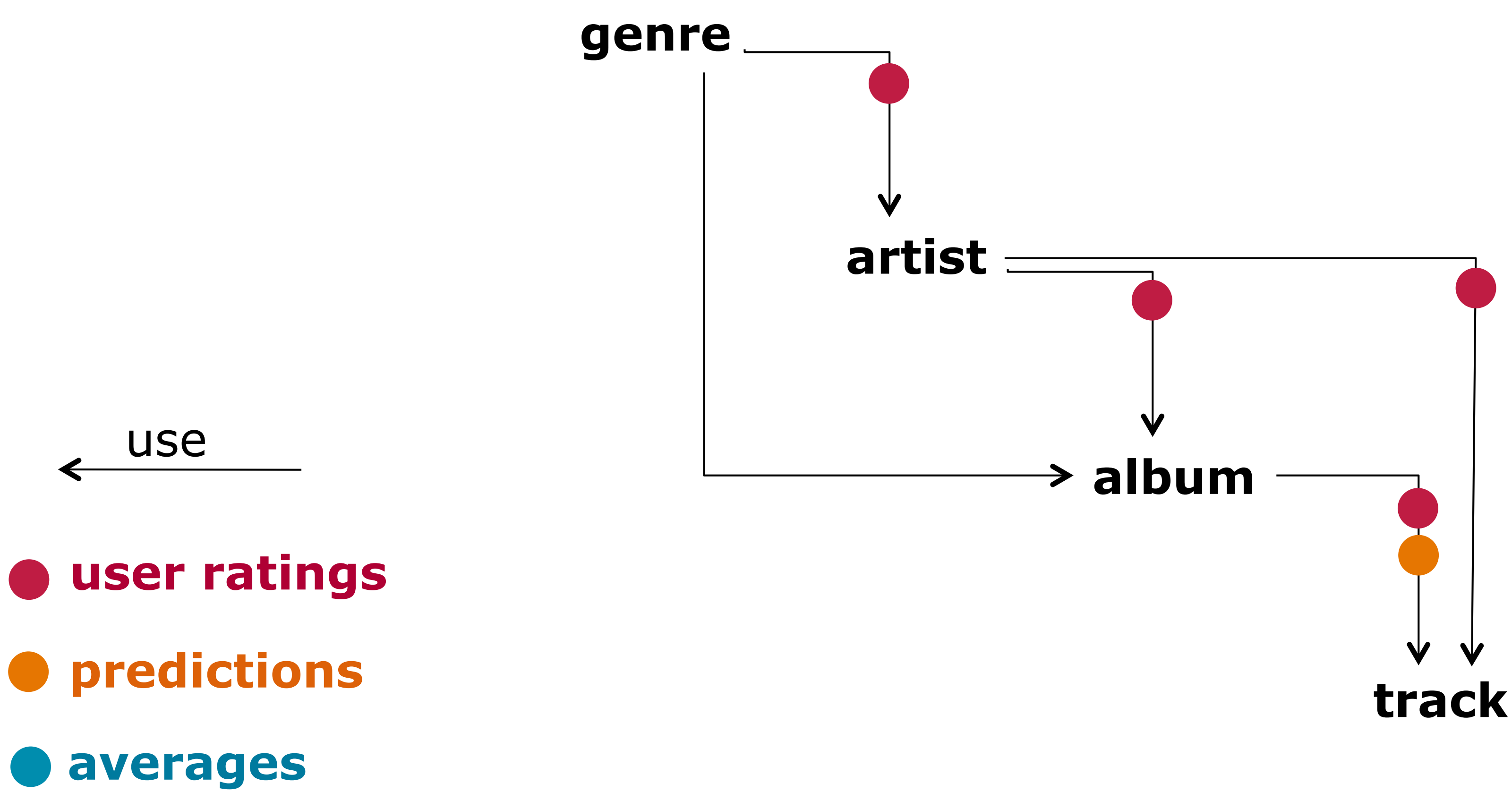
- Blend prediction with
  - **Ratings** by the user for associated items
  - **Predictions** for associated items and the user
  - **Averages** for associated items

- Calculate confidences based on number of observed ratings
- Machine learning for weighting factors

Outlook: Make Use of Hierarchy



Outlook: Make Use of Hierarchy



Summary

- Implemented SGD algorithm for matrix fatorization
  - Parallel processing of the four subsets using 6GB RAM
- Using simple biases to improve predictions
  - Planned: Improve bias calculation
- Experimented with different algorithm parameters
  - Submissions to validate on test set
  - Best RMSE so far: 26.5217
- Upcoming: Use hierachy to improve predictions
  - Linear (weighted) blend of associated:
    - ratings by the same user
    - predictions for the same user
    - item averages

Discussion: Combine Approaches

- Identify weaknesses and strengths of each approach

- Collaborate for
  - Identifying samples (good predictions/bad predictions)
  - Calculating item and user statistics (metrics)

- Find correlations between metrics and prediction errors
- Linearly blend predictions of both approaches weighted according to significant metrics

Discussion: Combine Approaches

User Metrics	Item Metrics
▪ number of distinct rating values (+ variance)	▪ number of distinct rating values (+ variance)
▪ average rating (+ variance)	▪ average rating (+ variance)
▪ difference to global average	▪ difference to global average
▪ absolute number of ratings	▪ absolute number of ratings