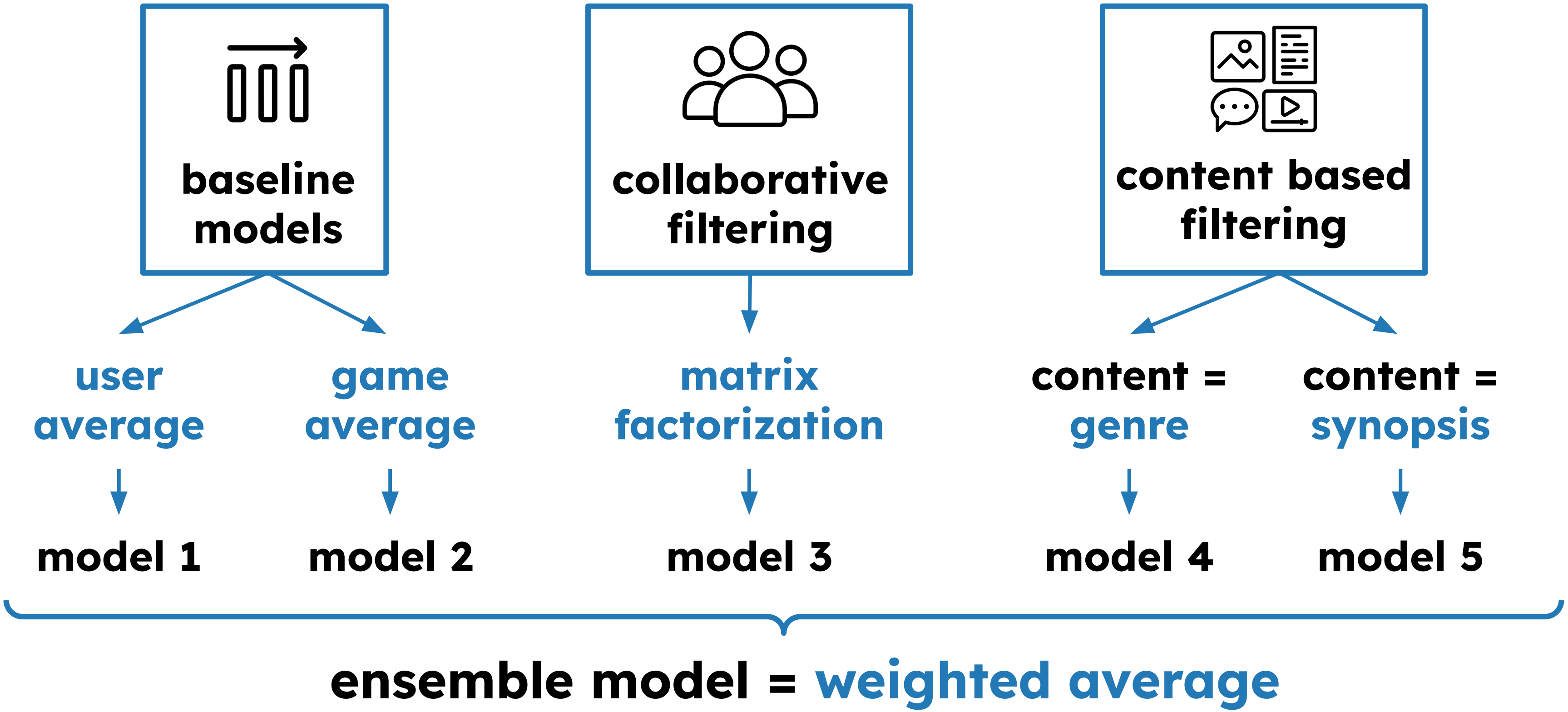
Recommender System for Games

Elisa Klunder

Aksel Joonas Reedi

Mihkel Mariusz Jezierski

/A general overview: the ensemble



/Insights about the data

1. Game ids and user ids were unnecessarily long and ended up in **"Out Of Memory" errors**
   ↪ Encoded the ids to be from 0 to n-1

2. Appids in game_metadata were **"objects"** while in train they were **ints**
   ↪ Cast the appids in game_metadata as ints

3. **Two rows were problematic** and prevented encoding/fitting models
   ↪ Dropped those two rows

/Model 3: Collaborative filtering

What we did: → Matrix Factorization
→ Alternating Least Squares

Insights: → The model performed better with a bigger **embedding dimension** (we started with 64 and ended up with 512)

→ The model overfitted if we let it run for too many **epochs** (we ended up setting the epochs to 7)

Open questions: → Was there a way to implement collaborative filtering considering **other user information** to compute similarity?

/Model 4: Content based on genres

What we did: → Used **one hot encoded** columns relating to **genre**, **category** and **other** information (e.g. is_free)

Insights: → The model performed better if we included **only the information of the “genre”** columns, excluding “category” and any other information

→ The model overshot the likelihood of a user to upvote a game, so putting the **threshold to 0.1** in the transformation from continuous to binary improved the model

Open questions: → The **user embedding** for every game was done by taking the **mean** of the “genre” columns, is there a better way? (we tried **TF-IDF encoding** but it didn’t seem to make it better)

/Model 5: Content based on game descriptions (BERT)

What we did:

→ Cleaned up text (e.g. HTML formatting)

→ Embedding of "short descriptions", "long descriptions" and "reviews" using BERT tokenizer

→ Fitted a content-based model with the embeddings

Insights:

→ Between reviews, short descriptions, long descriptions, and a combination of all three, **short descriptions worked the best**

→ **Stemming** was unnecessary because BERT has its own simplification method

Open questions:

→ Is there a smarter way of obtaining embeddings than just taking the **mean** of the elements of the vector?

/Ensamble

What we did: → **Weighted average** of the binary predictions

Insights: → Getting the most accurate weights is not trivial:

- using **logistic regression** with a val set didn't work well
- using **random search** on the weights worked better
- we obtained the best results by fine tuning the parameters **by hand**

Open questions: → We also tried to make predictions on an **ensemble of enable predictions** and it obtained the highest score (most probably overfitting the test data)

→ Maybe **keeping the gradients** and only making the predictions binary after the ensemble would have been better(?)