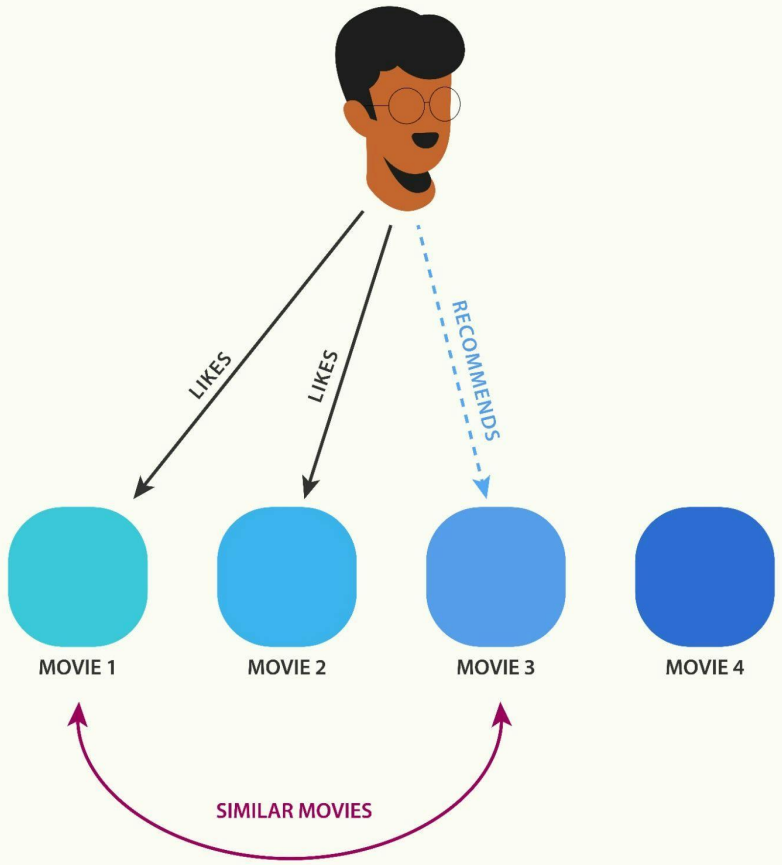

SISTEM PREPORUKA
Content based filtering
(Na osnovu sadržaja)
&
Collaborative filtering
(Na osnovu sličnih korisnika)

Type	Definition	Example
content-based filtering	Uses *similarity between items* to recommend items similar to what the user likes.	If user A watches two cute cat videos, then the system can recommend cute animal videos to that user.
collaborative filtering	Uses *similarities between queries and items simultaneously* to provide recommendations.	If user A is similar to user B, and user B likes video 1, then the system can recommend video 1 to user A (even if user A hasn't seen any videos similar to video 1).

Izvor: Google Developers – vodič za sisteme preporuke
https://developers.google.com/machine-learning/recommendation/overview/candidate-generation

CONTENT-BASED FILTERING RECOMMENDER SYSTEM



Turing

Content-based filtering advantages & disadvantages

Send feedback

Save page

Advantages

- The model doesn't need any data about other users, since the recommendations are specific to this user. This makes it easier to scale to a large number of users.
- The model can capture the specific interests of a user, and can recommend niche items that very few other users are interested in.

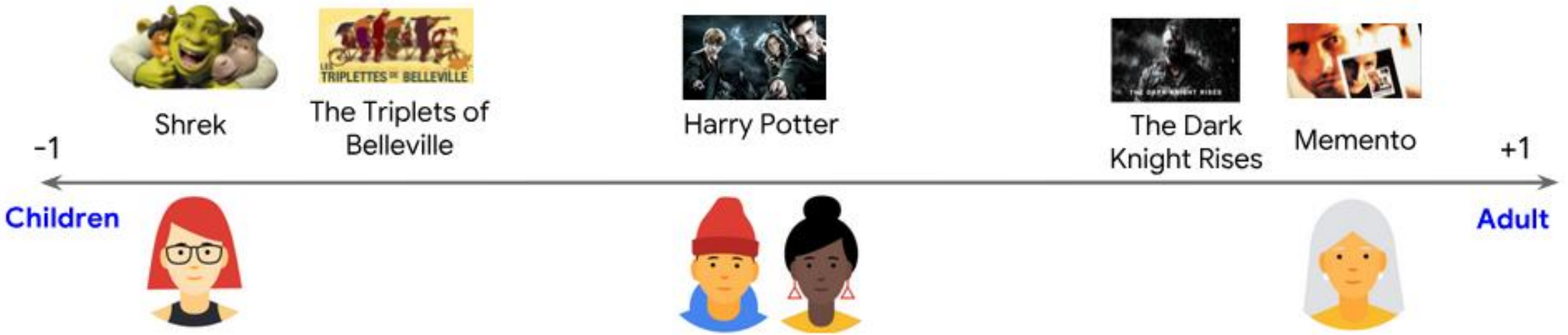
Disadvantages

- Since the feature representation of the items are hand-engineered to some extent, this technique requires a lot of domain knowledge. Therefore, the model can only be as good as the hand-engineered features.
- The model can only make recommendations based on existing interests of the user. In other words, the model has limited ability to expand on the users' existing interests.

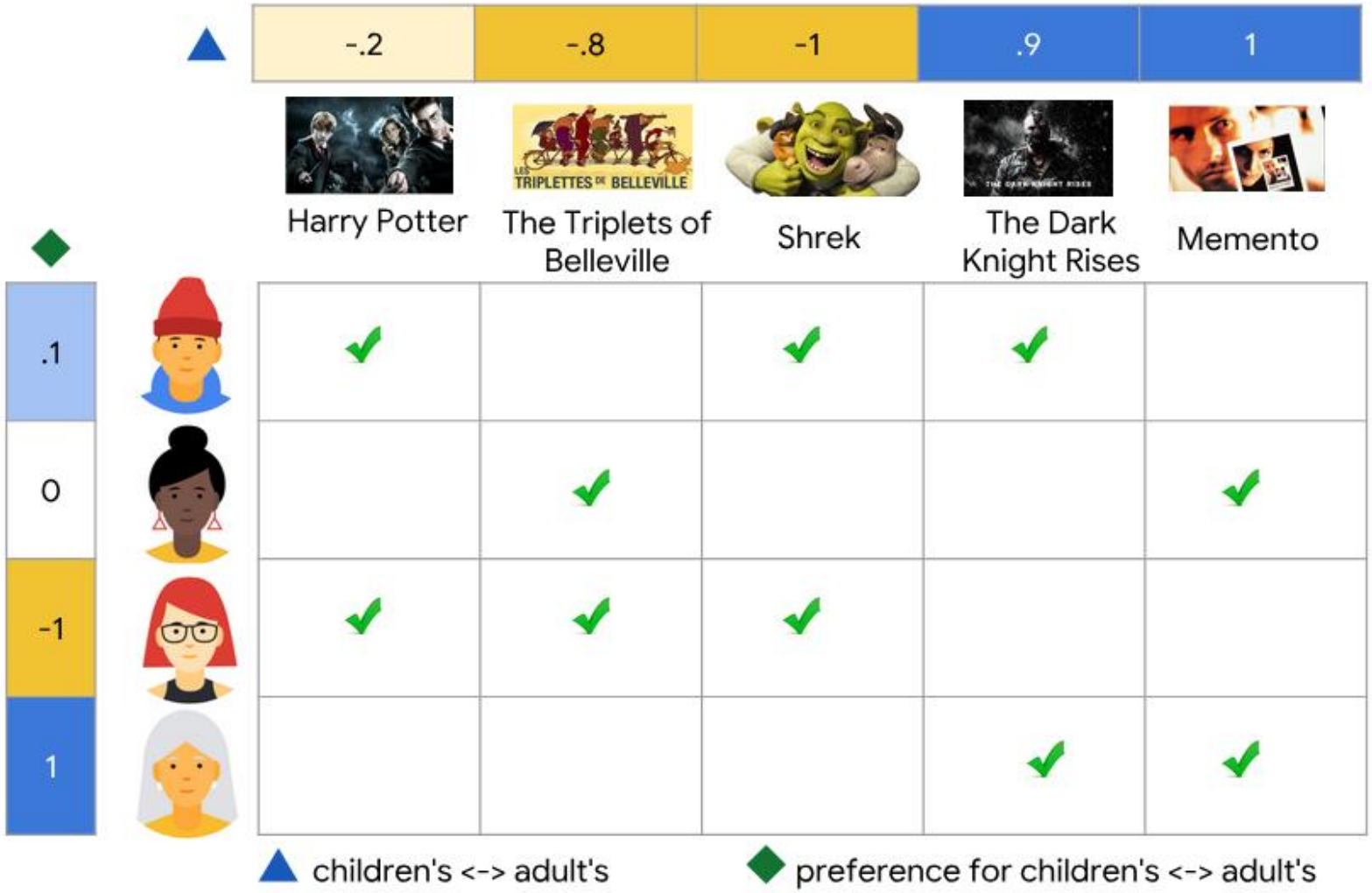
Collaborative filtering

1D embedding

Suppose we assign to each movie a scalar in $[-1, 1]$ that describes whether the movie is for children (negative values) or adults (positive values). Suppose we also assign a scalar to each user in $[-1, 1]$ that describes the user's interest in children's movies (closer to -1) or adult movies (closer to +1). The product of the movie embedding and the user embedding should be higher (closer to 1) for movies that we expect the user to like.

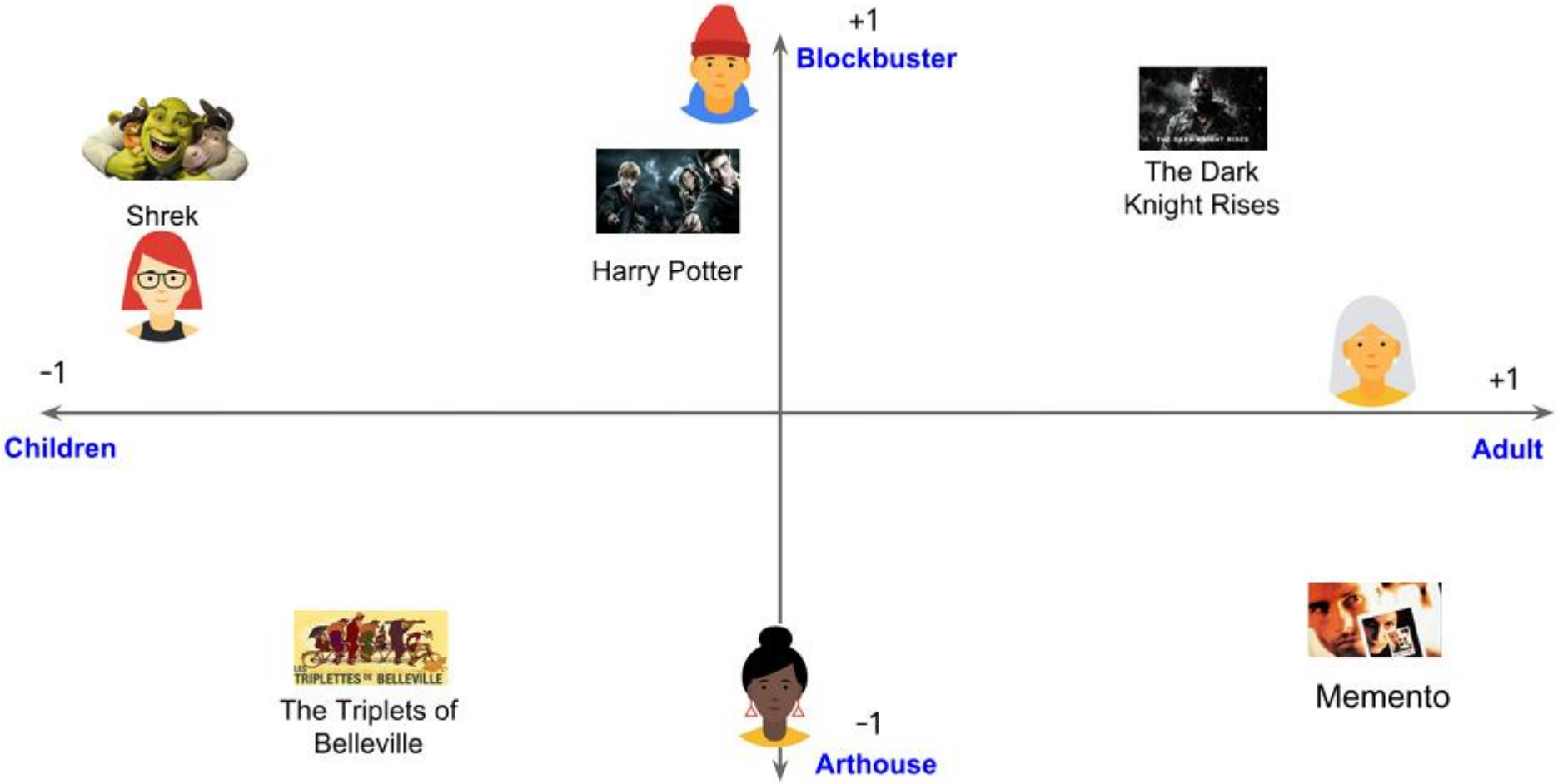


In the diagram below, each checkmark identifies a movie that a particular user watched. The third and fourth users have preferences that are well explained by this feature—the third user prefers movies for children and the fourth user prefers movies for adults. However, the first and second users' preferences are not well explained by this single feature.

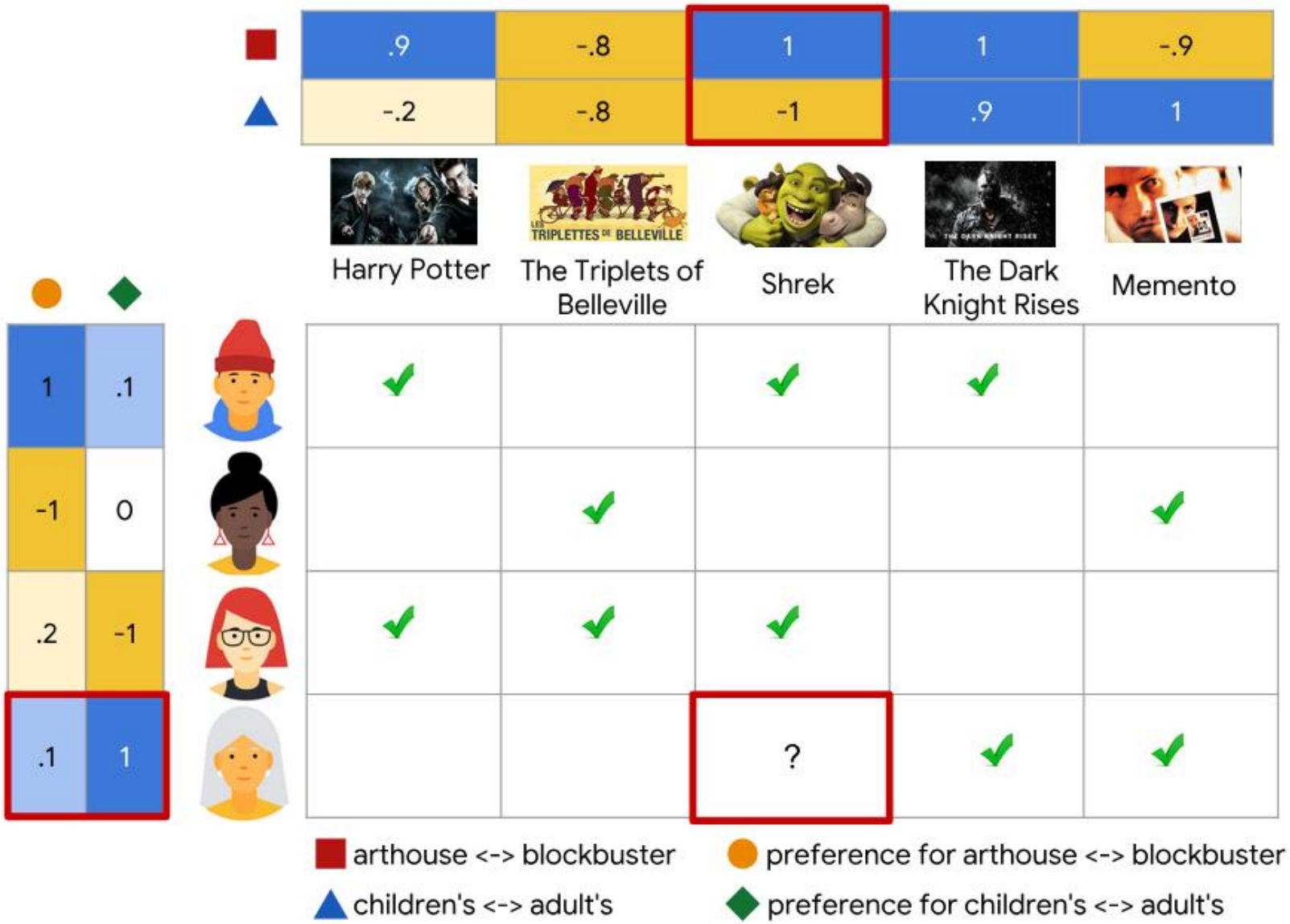


2D embedding

One feature was not enough to explain the preferences of all users. To overcome this problem, let's add a second feature: the degree to which each movie is a blockbuster or an arthouse movie. With a second feature, we can now represent each movie with the following two-dimensional embedding:



We again place our users in the same embedding space to best explain the feedback matrix: for each (user, item) pair, we would like the dot product of the user embedding and the item embedding to be close to 1 when the user watched the movie, and to 0 otherwise.



Matrix factorization

Send feedback

Matrix factorization is a simple embedding model. Given the feedback matrix A $\in R^{m \times n}$, where m is the number of users (or queries) and n is the number of items, the model learns:

- A user embedding matrix $U \in \mathbb{R}^{m \times d}$, where row i is the embedding for user i.
- An item embedding matrix $V \in \mathbb{R}^{n \times d}$, where row j is the embedding for item j.

The embeddings are learned such that the product UV^T is a good approximation of the feedback matrix A. Observe that the (i, j) entry of $U.V^T$ is simply the dot product $\langle U_i, V_j \rangle$ of the embeddings of user i and item j, which you want to be close to $A_{i,j}$.

Collaborative filtering advantages & disadvantages

Advantages

No domain knowledge necessary

We don't need domain knowledge because the embeddings are automatically learned.

Serendipity

The model can help users discover new interests. In isolation, the ML system may not know the user is interested in a given item, but the model might still recommend it because similar users are interested in that item.

Great starting point

To some extent, the system needs only the feedback matrix to train a matrix factorization model. In particular, the system doesn't need contextual features. In practice, this can be used as one of multiple candidate generators.

Disadvantages

Cannot handle fresh items

The prediction of the model for a given (user, item) pair is the dot product of the corresponding embeddings. So, if an item is not seen during training, the system can't create an embedding for it and can't query the model with this item. This issue is often called the **cold-start problem**. However, the following techniques can address the cold-start problem to some extent:

Praktična primena

Podaci: MovieLens 32M

ratings.csv:

- 32 miliona ocena user -> movie
- 10 mogućih ocena (od 0.5 do 5)

Movies.csv:

- 87,5 hiljada filmova, njihovi naslovi i žanrovi (20 različitih žanrova, moguće je da film ima više od jednog žanra)

✓ ratings ···

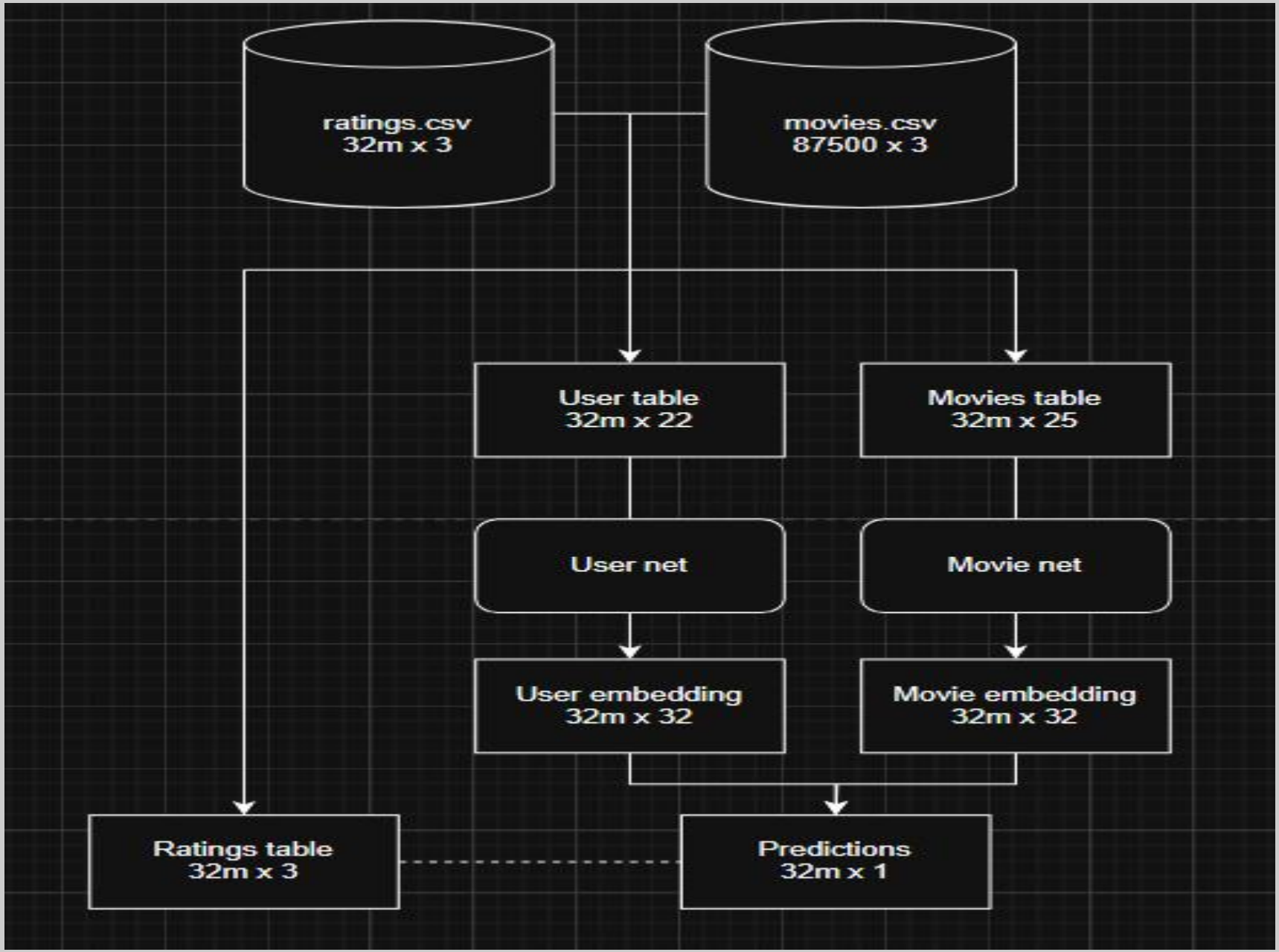
shape: (32_000_204, 3)

userId	movieId	rating
i64	i64	f64
1	17	4.0
1	25	1.0
1	29	2.0
1	30	5.0
1	32	5.0
…	…	…
200948	79702	4.5
200948	79796	1.0
200948	80350	0.5
200948	80463	3.5
200948	87304	4.5

✓ movies ···

shape: (87_585, 3)

movieId	title	genres
i64	str	str
1	"Toy Story (1995)"	"Adventure|Animation|Children|C…
2	"Jumanji (1995)"	"Adventure|Children|Fantasy"
3	"Grumpier Old Men (1995)"	"Comedy|Romance"
4	"Waiting to Exhale (1995)"	"Comedy|Drama|Romance"
5	"Father of the Bride Part II (1…	"Comedy"
…	…	…
292731	"The Monroy Affaire (2022)"	"Drama"
292737	"Shelter in Solitude (2023)"	"Comedy|Drama"
292753	"Orca (2023)"	"Drama"
292755	"The Angry Breed (1968)"	"Drama"
292757	"Race to the Summit (2023)"	"Action|Adventure|Documentary"


ratings.csv
32m x 3
movies.csv
87500 x 3
User table
32m x 22
Movies table
32m x 25
User net
Movie net
User embedding
32m x 32
Movie embedding
32m x 32
Ratings table
32m x 3
Predictions
32m x 1

```
✓ user, movies, y = prep_pipeline(ratings.sample(1_000_000), movies) ···
```

```
✓ user ···
```

shape: (1_000_000, 22)

userId	movieId	no genres listed	Action	Adventure	Animation	Children	Comedy	Crime	Documentary	Drama	Fantasy	Film-Noir	Horror	IMAX	Musical	Mystery	Romance	Sci-Fi	Thriller	War
i64	i64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64
10554	586	0.0	3.5	2.75	0.0	5.0	4.25	2.5	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	4.75	3.5	1.5	5.0
30603	1307	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	3.5	0.0	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0
146262	50	0.0	5.0	4.5	4.5	4.5	5.0	5.0	0.0	4.25	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	5.0	0.0
95481	87298	0.0	3.442308	3.4	2.681818	2.653846	2.758929	3.5	0.0	2.984127	2.71875	0.0	2.722222	3.857143	4.5	2.0	3.092593	3.545455	3.044118	3.375
27863	184721	0.0	3.25	3.5	3.75	4.0	3.555556	3.5	3.75	3.416667	3.375	3.5	2.75	3.0	3.5	3.4	3.357143	0.0	3.25	3.25
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
137072	19	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
45213	1722	0.0	5.0	5.0	0.0	0.0	5.0	5.0	0.0	5.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	5.0	0.0
126941	2502	0.0	0.0	0.0	0.0	0.0	5.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
52329	85881	0.0	3.5	3.666667	3.5	3.5	2.85	3.5	0.0	3.269231	0.0	0.0	3.0	3.5	3.5	3.75	2.75	3.5	3.833333	3.5
189966	1219	0.0	0.0	0.0	0.0	0.0	0.0	4.0	0.0	3.0	0.0	0.0	4.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0

✓ movies ···

shape: (1_000_000, 25)

userId	movieId	#ratings_film	year	avg_rating	no genres listed	Action	Adventure	Animation	Children	Comedy	Crime	Documentary	Drama	Fantasy	Film-Noir	Horror	IMAX	Musical	Mystery	Romance
i64	i64	u32	i16	f64	i8	i8	i8	i8	i8	i8	i8	i8	i8	i8	i8	i8	i8	i8	i8	i8
10554	586	1157	1990	3.183665	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
30603	1307	836	1989	3.81878	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
146262	50	2074	1995	4.281823	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0
95481	87298	19	2010	3.157895	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0
27863	184721	33	2017	3.848485	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
137072	19	755	1995	2.634437	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
45213	1722	441	1997	3.22449	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
126941	2502	867	1999	3.983276	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
52329	85881	20	2011	3.625	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0
189966	1219	873	1960	4.049828	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0

y ···

shape: (1_000_000, 3)

userId	movieId	rating
i64	i64	f64
10554	586	5.0
30603	1307	5.0
146262	50	5.0
95481	87298	0.5
27863	184721	3.0
…	…	…
137072	19	2.0
45213	1722	5.0
126941	2502	5.0
52329	85881	3.0
189966	1219	4.0

Obrada podataka

- Kako je reč o ogromnom skupu podataka (user tabela je 32m x 23, movies tabela je 32m x 25), podaci ne mogu da stanu u memoriju od jednom

- Zato ih učitavam u serijama (batch processing) pomoću TensorFlow-a, dok za svaku seriju koristim Polars za brzu obradu i pripremu podataka, koje zatim vraćam u TensorFlow u obliku tenzora.

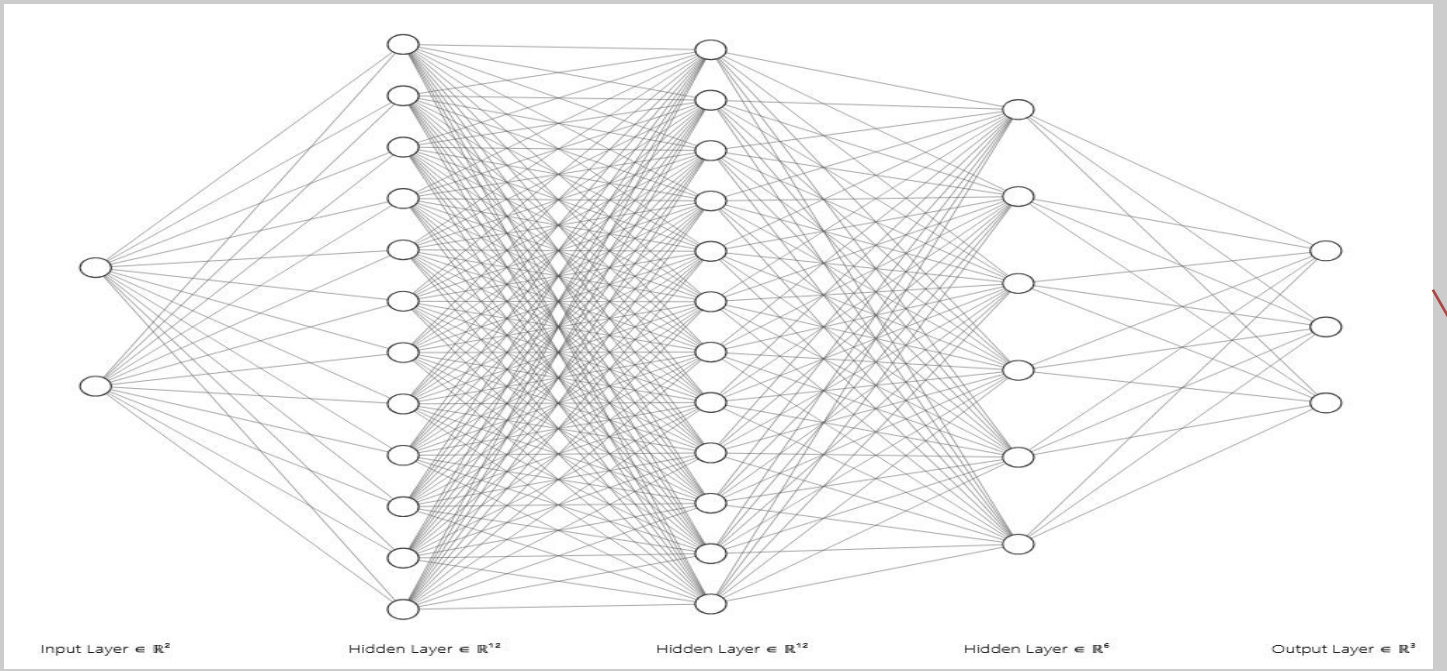
Arhitektura neuralne mreže

- User mreža:
- ulazne dimenzije:20,
- dimenzije skrivenih slojeva: [128, 128, 64]
- Izlazne dimenzije (embedding vektor): 32

- Movies mreža:
- ulazne dimenzije:23,
- dimenzije skrivenih slojeva: [128, 128, 64]
- Izlazne dimenzije (embedding vektor): 32

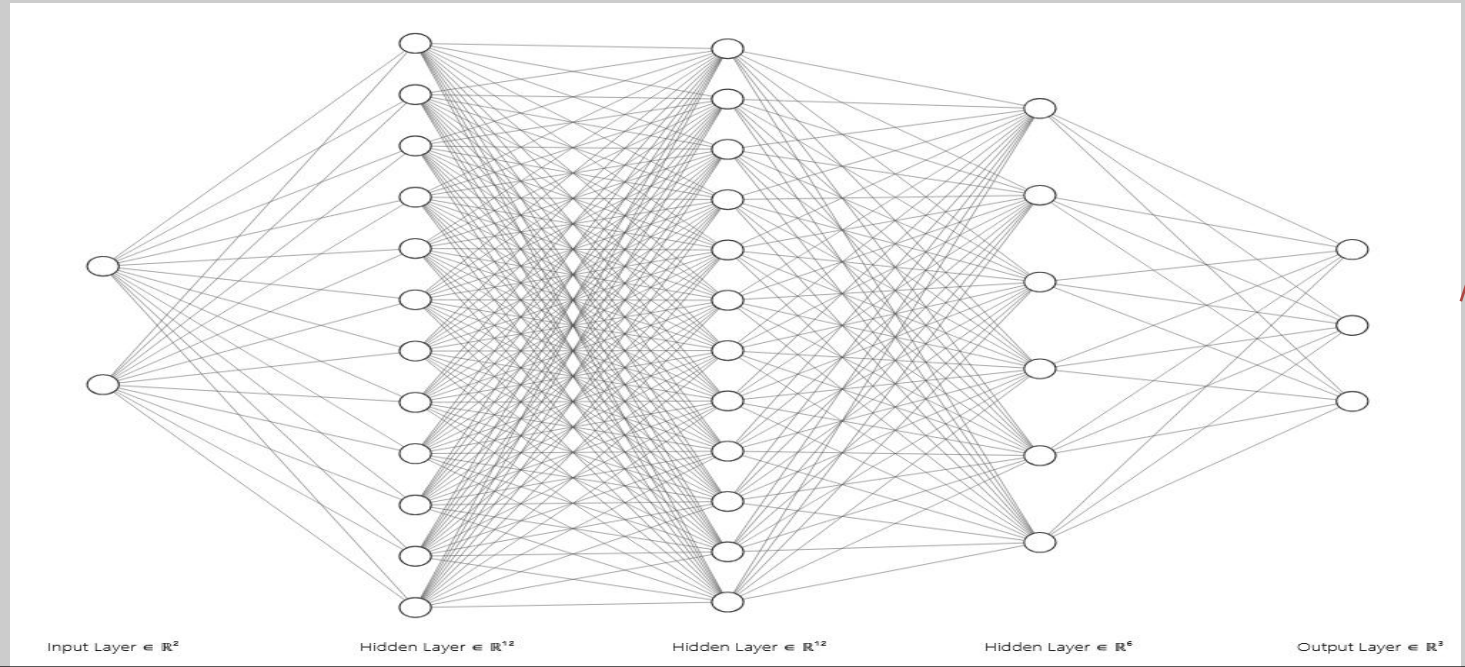
- Embedding vektori se L2-normalizuju (skaliraju tako da imaju dužinu 1)
- Tačkasti proizvod ta dva normalizovana vektora predstavlja kosinus ugla izmedju njih, što je mera njihove sličnosti. (na skali -1 do 1)

User mreža



Tačkasti proizvod = $\widehat{Y}$

Movies mreža



U svakom sloju, broj neurona je smanjen za oko 10 puta radi preglednosti
U svakom sloju se koristi ReLu aktivaciona funkcija, i batch normalization, za trening se koristi Adam optimizator

✓ model.user_net.summary() ···

Model: "user_mreza"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 128)	2,688
batch_normalization (BatchNormalization)	(None, 128)	512
activation (Activation)	(None, 128)	0
dense_1 (Dense)	(None, 128)	16,512
batch_normalization_1 (BatchNormalization)	(None, 128)	512
activation_1 (Activation)	(None, 128)	0
dense_2 (Dense)	(None, 64)	8,256
batch_normalization_2 (BatchNormalization)	(None, 64)	256
activation_2 (Activation)	(None, 64)	0
dense_3 (Dense)	(None, 32)	2,080
l2_normalize_layer (L2NormalizeLayer)	(None, 32)	0

Total params: 30,816 (120.38 KB)

Trainable params: 30,176 (117.88 KB)

Non-trainable params: 640 (2.50 KB)

✓ model.movie_net.summary() ···

Model: "movie_mreza"

Layer (type)	Output Shape	Param #
dense_4 (Dense)	(None, 128)	3,072
batch_normalization_3 (BatchNormalization)	(None, 128)	512
activation_3 (Activation)	(None, 128)	0
dense_5 (Dense)	(None, 128)	16,512
batch_normalization_4 (BatchNormalization)	(None, 128)	512
activation_4 (Activation)	(None, 128)	0
dense_6 (Dense)	(None, 64)	8,256
batch_normalization_5 (BatchNormalization)	(None, 64)	256
activation_5 (Activation)	(None, 64)	0
dense_7 (Dense)	(None, 32)	2,080
l2_normalize_layer_1 (L2NormalizeLayer)	(None, 32)	0

Total params: 31,200 (121.88 KB)

Trainable params: 30,560 (119.38 KB)

Non-trainable params: 640 (2.50 KB)

✓ model.summary() ···

Model: "movie_recommender"

Layer (type)	Output Shape	Param #
user_mreza (Sequential)	(None, 32)	30,816
movie_mreza (Sequential)	(None, 32)	31,200
cosine_similarity (Dot)	(None, 1)	0
squeeze_layer (SqueezeLayer)	?	0

Total params: 183,490 (716.76 KB)

Trainable params: 60,736 (237.25 KB)

Non-trainable params: 1,280 (5.00 KB)

Optimizer params: 121,474 (474.51 KB)

Podela podataka

Test skup ~ milion podataka
Dev skup ~ milion podataka
Train skup ~30 miliona podataka
Svaki odvajam u posebni .csv fajl

Na velikim podacima nema potrebe za klasičnom podelom 60:20:20, po milion podataka je sasvim dovoljno da se provere preformanse na dev i na test skupovima, dok za trening dubokih neuralnih mreža uvek znači više podataka

Svaki skup podataka prosleđujem u poseban TensorFlow.Dataset objekat

```python
def batch_generator(ratings_path, movies_path, batch_size, train = False):
    """
    Pravi batch-eve iz CSV fajla sa ocenjivanjem filmova (ratings).
    """
    movies = pl.read_csv(movies_path)
    offset = 0
    #Lazy prebroji redove
    total_rows = pl.scan_csv(ratings_path).select(pl.len()).collect()[0, 0]
    while offset < total_rows:
        batch = pl.read_csv(ratings_path).slice(offset, batch_size)
        if batch.height < batch_size:
            break
        user_df, movies_df, y_df= prep_pipeline(batch, movies)

        user_tensor = tf.convert_to_tensor(user_df.to_numpy(), dtype=tf.float32)
        movies_tensor = tf.convert_to_tensor(movies_df.to_numpy(), dtype=tf.float32)
        y = tf.convert_to_tensor(y_df.to_numpy(), dtype=tf.float32)
        if train: #ako je za trening vrati bez ID
            yield (user_tensor[:,2:], movies_tensor[:,2:]),  y[:, 2:]
        else: #ako je za inference prve dve kolone su ID u svakom tenzoru
            yield (user_tensor, movies_tensor), y

        offset += batch_size

data_train = tf.data.Dataset.from_generator(lambda: batch_generator(ratings_path= train_path, movies_path= csv_movies,batch_size= batch, train = True),
    output_signature= ((tf.TensorSpec(shape=(None, 20), dtype=tf.float64, name = 'user'), tf.TensorSpec(shape=(None, 23),
    dtype=tf.float64, name = 'movie')), tf.TensorSpec(shape=(None,1), dtype=tf.float32))).prefetch(tf.data.AUTOTUNE)

data_dev = tf.data.Dataset.from_generator(lambda: batch_generator(ratings_path= dev_path, movies_path= csv_movies,batch_size= batch, train = True),
    output_signature= ((tf.TensorSpec(shape=(None, 20), dtype=tf.float64, name = 'user'), tf.TensorSpec(shape=(None, 23),
    dtype=tf.float64, name = 'movie')), tf.TensorSpec(shape=(None,1), dtype=tf.float32))).prefetch(tf.data.AUTOTUNE)
```

Priprema podataka za trening

Skaliram podatke o ocenama na interval -1 do 1

```
def scale_y(y):
    '''od -1 do 1 '''
    return 2 * (y - 0.5) / 4.5 - 1
```

Da bih standardizovao user i movie matrice moram da prodjem ceo trening set i da izracunam sredinu i varijansu na njemu. Bitno je da se sa istim tim parametrima skalira i dev i test skup (da se ne racunaju opet)

```
def standardizacija(var):
    '''var -> dataset.map()
    vraca adaptiran normalization sloj
    '''
    x = layers.Normalization()
    x.adapt(var)
    return x
```

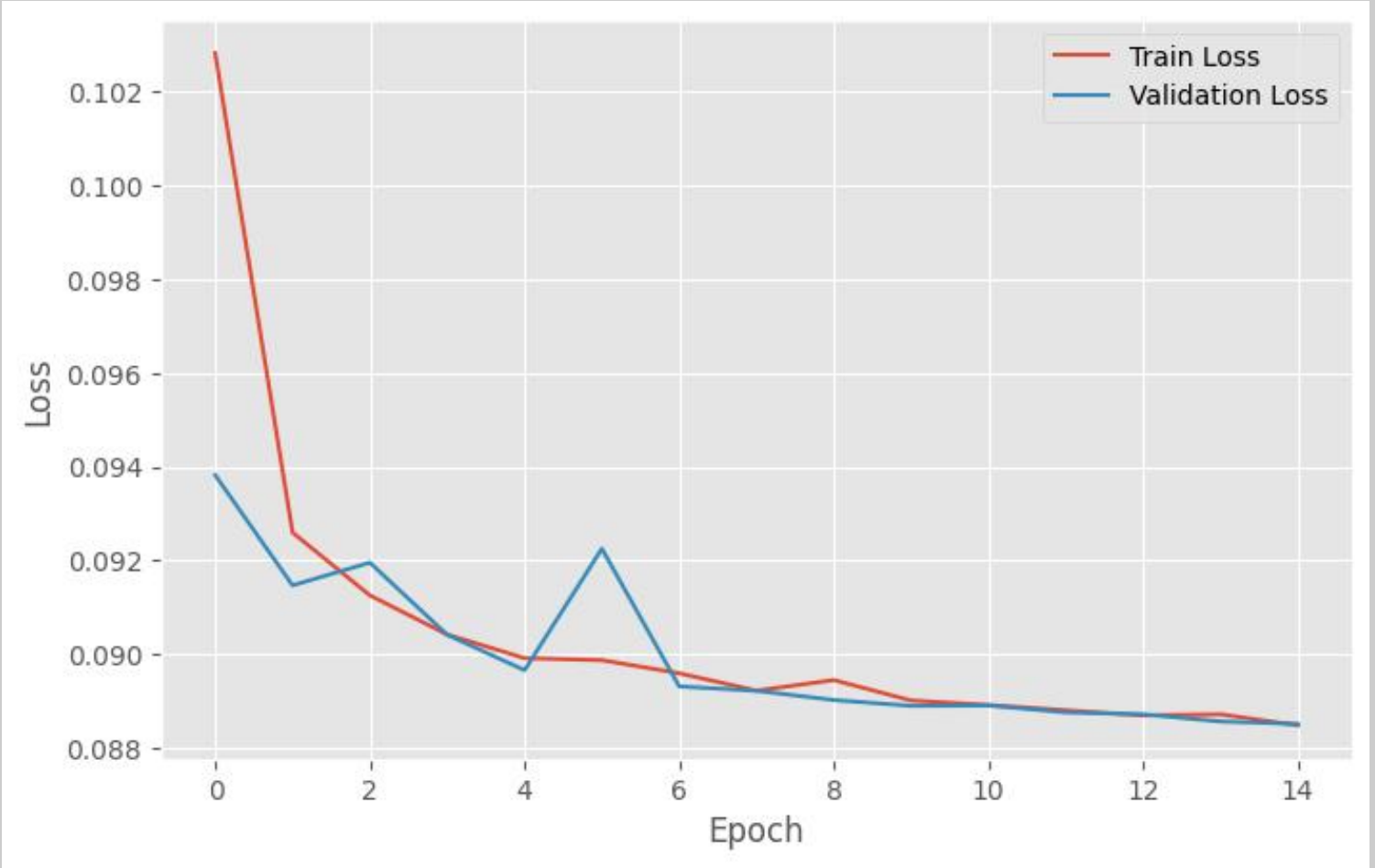
```
#skaliranje
data_train = data_train.map(lambda x, y: ((norm_user(x[0]),tf.concat([tf.cast(norm_movies(x[1][:, :3]), tf.float32),tf.cast(x[1][:, 3:], tf.float32)], axis=1)),scale_y(y))).repeat().prefetch(tf.data.AUTOTUNE)
next(iter(data_train))
data_dev = data_dev.map(lambda x, y: ((norm_user(x[0]),tf.concat([tf.cast(norm_movies(x[1][:, :3]), tf.float32),tf.cast(x[1][:, 3:], tf.float32)], axis=1)),scale_y(y))).prefetch(tf.data.AUTOTUNE)

model = MovieRecommender(user_input_dim=20,movie_input_dim=23,hidden_layers=[128, 128, 64], embedding_dim=32)

model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.001), loss='mse', metrics=['mae', 'mse'])

train_size = train.height // batch

history = model.fit(data_train, validation_data=data_dev, epochs=15, steps_per_epoch=train_size)
```




```
✓ mean_absolute_error(y_test, y_test_pred) ···
```

```
0.17496332053721503
```

```
✓ mean_squared_error(y_test, y_test_pred) ···
```

```
0.08847107060218981
```