Bookommender

Tabea Schaeffer, Stefan Maier, Ismail Halili, Lena Ebner

Bookommender ist eine Buchempfehlungsplattform, die auf Basis von Ratings und Metadaten dem Benutzer Bücher empfiehlt.

Verfahren/Theoretischer Ansatz

Recommendations: UBCF + IBCF mit FM

Cold-Start: Most Popular + Least Popular, r.Cold_Start _Recommend()

Datensplit: k-Fold Cross Validation -> Generalisierung

Evaluierung: Truncated Precision, Recall, nDCG

Datenaufbereitung: next slide

Datenaufbereitung

From 278.858 to 167.723 users.

- Remove all user where 5 > age < 100 || age == null

From 271.379 to 228.144 books.

- Merge books with same title, author and year
- Remove bad characters from all strings
- Set Publication Year between 1800-2025 otherwise average
- Extend with Excerpt, Tags, NumberOfPages, PublishedPlaces via OpenLibrary

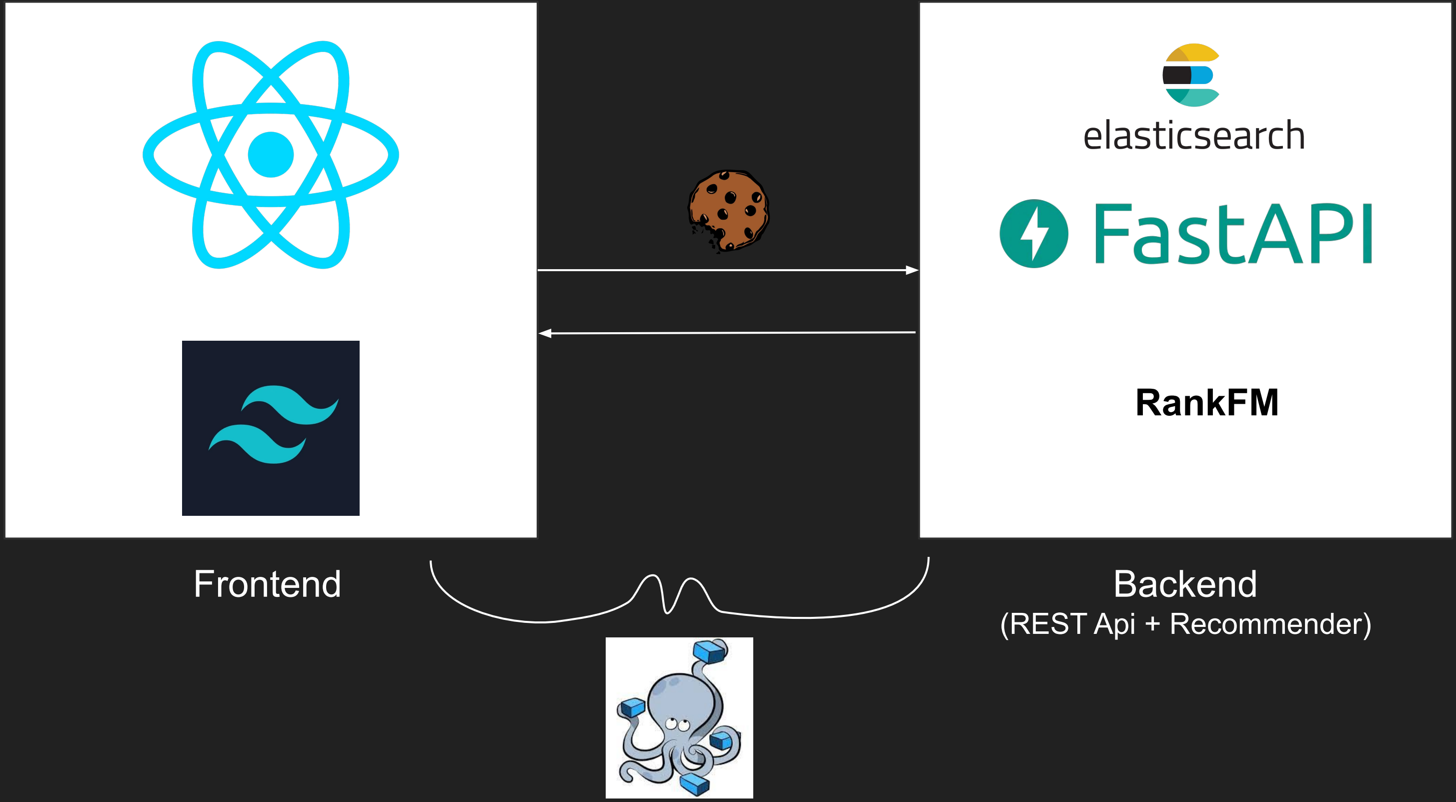
From 1.048.574 to 656.293 ratings.

- Remove all ratings where user is not found
- Change isbn to isbn from book with same title, author, year
- We used 0 ratings = implicit feedback

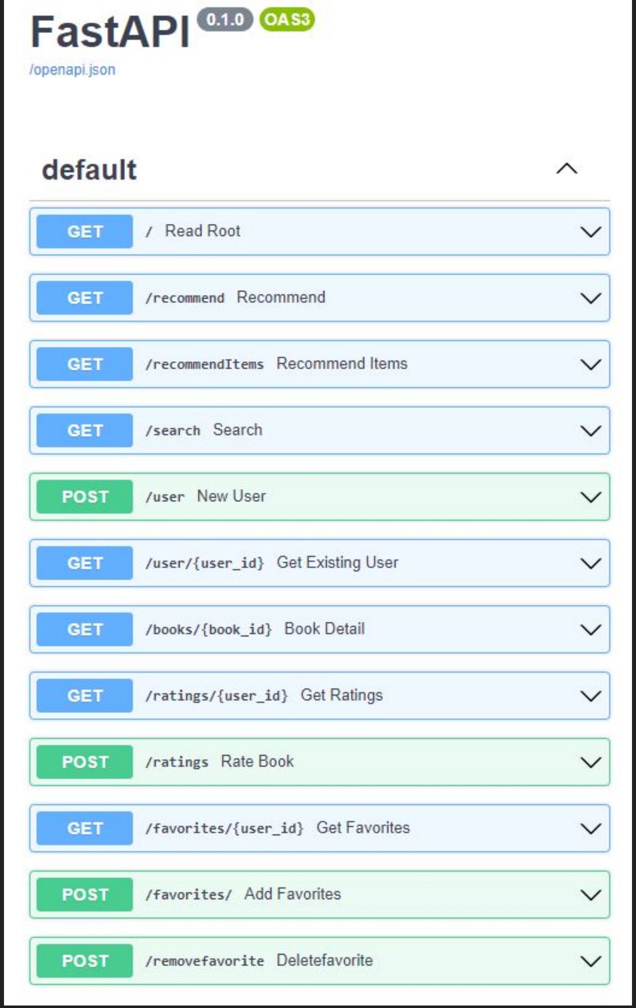
Feature Engineering

- Buch-**Bewertung** mit Sternen 1-10
- **IBCF**: Jahr, Tags in Vorschläge einbeziehen
- **UBCF**: Alter und Ort in Vorschläge einbeziehen
- **Suche**: Suchergebnisse + Boosting
  (title, excerpt, author)
- **Favoriten**: Rating wird stärker gewichtet
- **Boosting**
- **Login** as existing User

Techstack & Architektur



Schnittstellen



Implementierung

- Most+Least Popular
- Boosting
- RankFM

Implementierung

- **Most+Least Popular**
- Boosting
- RankFM

Most Popular

1. Daten laden
2. Sortierung der Daten nach der Häufigkeit von Ratings pro Buch
3. den Durchschnitt der Bewertungen pro Buch ermitteln
4. Sortieren der durchschnittlichen Bewertungen (beste bis schlechteste)
5. Cut auf Top 50 Bücher

Least Popular

1. Filtern nach alle Büchern ohne Rating
2. random 15 Bücher daraus auswählen

Most-least Popular Books

1. Zusammenfügen von Most- und Least Büchern
2. Shufflen der Bücher
3. Bücher in der Datei MostLeastPopular.csv speichern

Implementierung

- Most+Least Popular
- **Boosting**
- RankFM

Boosting

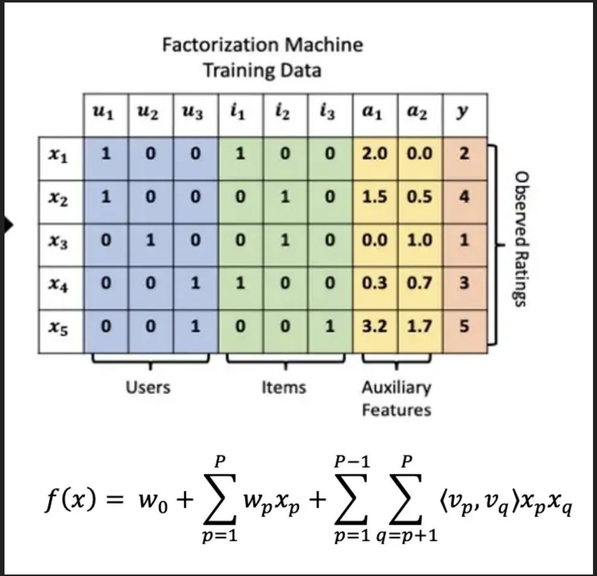
```
def boost(recommendations, favs, searches, user):
    withind = list()
    for i, book in enumerate(recommendations):
        boostscore = 0
        for fav in favs:
            fb = books.bookshash[fav]
            if(fb.author == book.author or fb.publisher == book.publisher):
                boostscore += 200
        for search in searches:
            if(search in book.author or search in book.title or search in book.excerpt):
                boostscore += 200

        if book.author == user.favoriteAuthor or book.publisher == user.favoritePublisher:
            boostscore += 200

        withind.append([book, i+boostscore])
    withind.sort(key=takescore, reverse=True)
    return list(map(lambda x: x[0], withind))
```

Implementierung

- Most+Least Popular
- Boosting
- **RankFM**

RankFM

- FM
- Implizite Daten mit Gewichtung
- Learning-to-Rank (LTR)
  - Bayesian Personalized Ranking (BPR)
- Rangordnung optimieren



```python
class Recommender:
    _model_file = open("trained_models/final", "rb")
    _users_file = open("known_users", "rb")
    _user_converter = {
        'User-ID': int,
        'Age': int,
        'City': str,
        'State': str,
        'Country': str
    }


    def __init__(self):
        self.model: RankFM = pickle.load(self._model_file)
        self._users: DataFrame = pickle.load(self._users_file)

    def get_for_user(self, user_id: int, n_items: int = 10, filter_previous=False):
        recs = self.model.recommend([user_id], n_items, filter_previous).loc[user_id]
        rec_array = []

            rec_array.append(rec)



    def get_similar_items(self, item_id: str, n_items: int = 10):
        return self.model.similar_items(item_id, n_items)


    def get_similar_users(self, user_id: int, n_items: int = 10):
        return self.model.similar_users(user_id, n_items)

    def cold_start_similar_users(self, age: int, country: str):
        age_tolerance = 5

        same_country_users = self._users.loc[self._users["Country"] == country]

        if len(same_country_users) == 0:
            return self._similar_age(self._users, age, age_tolerance)
        else:
            return self._similar_age(same_country_users, age, age_tolerance)


    def _similar_age(self, filtered_users: DataFrame, age: int,  age_tolerance: int):
        while len(filtered_users.loc[abs((self._users["Age"] - age)) <= age_tolerance]) == 0:
            age_tolerance = age_tolerance * 1.5

        return filtered_users.loc[abs((self._users["Age"] - age)) <= age_tolerance]
```

```python
def cold_start_recommend(self, age: int, country: str, has_rated_items: List[str] = [], n_items: int = 10):
    similar_users = self.cold_start_similar_users(age, country)["User-ID"].values[:400]
    rec_candidates_by_users = []

    if(len(similar_users) < 10):
        rec_candidates_by_users.append(self.model.recommend(similar_users, n_items=n_items).values.flatten())
    else:
        n_processes = 4
        user_recs_queue = Queue()
        chunk_size = len(similar_users) // n_processes
        users_chunks = [similar_users[i:i + chunk_size] for i in range(0, len(similar_users), chunk_size)]
        tasks = []

        for chunk in users_chunks:
            tasks.append({"target": rec_in_process, "kwargs": {"chunk": chunk, "model": self.model, "queue": user_recs_queue}})

        run_in_parallel(tasks)
        rec_candidates_by_users = user_recs_queue.get()

    rec_candidates_by_items = []

    for item in has_rated_items:
        try:
            rec_candidates_by_items = np.append(rec_candidates_by_items, self.model.similar_items(item))
        except:
            continue

    all_recs = np.append(rec_candidates_by_users, rec_candidates_by_items)
    np.random.shuffle(all_recs)
    return list(set(all_recs))[:n_items]
```

Evaluierung

Baseline: (65 Empfehlungen)

- Precision: 0.15 %
- Recall: 1.56 %
- nDCG: 0.261

Faktorisierungsmaschine: (10 Empfehlungen)

- Precision: 2.62%
- Recall: 1.98%
- nDCG: 0.411
- Hit Rate: 16.5% (Für wie viele User kann etwas sinnvolles empfohlen werden)

```python
for i, (train_index, test_index) in enumerate(stratified_kfold.split(X=min_10_r_per_u["user"], y=min_10_r_per_u["isbn"])):
    print(f"Fold {i}:")
    train_set = min_10_r_per_u.iloc[train_index]
    test_set = min_10_r_per_u.iloc[test_index]

    train_user_features = user_features[user_features["User-ID"].isin(train_set['user'])]
    train_user_features.loc[:,"Age"] /= train_user_features["Age"].max()
    train_book_features = book_features[book_features["ISBN"].isin(train_set['isbn'])]
    train_book_features.loc[:,"bf_1"] /= train_book_features['bf_1'].max()

    weights =[]

    for i, row in train_set.iterrows():
        if row.rating == 0:
            weights.append(1)
        elif row.rating >= mean_rating_per_user.loc[row.user].item():
            weights.append(2)
        else:
            weights.append(0)

    print("fitting...")
    model.fit(
        interactions=train_set[["user", "isbn"]],
        user_features=train_user_features,
        item_features=train_book_features,
        sample_weight=np.array(weights),
        epochs = 15
    )

    print("validating...")
    all_precisions.append(precision_at_k(model=model, test_set=test_set, k=10))
    all_recalls.append(recall_at_k(model=model, test_set=test_set, k=10))
    all_ndcgs.append(rank_at_k(model=model, test_set=test_set, k=10))
    all_hit_rates.append(hit_rate(model=model, test_interactions=test_set[["user", "isbn"]], k=10))
```

```
user f age, country
item f age
rating / 10 als weight
0er und avg+ ratings

precision:  0.013470196290293534
recall:     0.011826408493971203
```

```
user f age, country
item f age
no weight
0er und avg+ ratings

precision:  0.015685215198991537
recall:     0.02015197914663626
```

```
user f age, country
no item f
no weight
0er und avg+ ratings

precision: 0.01559517377993877
recall: 0.020342086518964825
```

```
user f age, country
with item f
no weight
avg+ ratings

precision: 0.008052276559865094
recall: 0.017104661404281886
```

```
no user f age, country
no item f
no weight
avg+ ratings

precision: 0.00851602023608769
recall: 0.01837799664369502
```

```
no user f age, country
no item f
with binary weight
all cleaned ratings

precision: 0.017743919251920894
recall: 0.02208053985730808
```

```
user f: age, country
item f: age in months, tags
with binary weight
all cleaned ratings

precision: 0.02490240841697551
recall: 0.025399523273584326
```

https://tenor.com/view/demoday-itsdemoday-fixerupper-chipgaines-gif-13280826

Lessons Learned

- **Daten Cleanen** viel mehr Aufwand als gedacht
- **Evaluierung**: RSME nicht sinnvoll gewesen
- **FM**:
  - Library die incremental fitten kann schwierig zu finden
  - Performance der Library (Laufzeit + Speicherprobleme)
  - RankFM kann nur für User recommenden, die im Trainingsset sind
  - mehr Features adden bringt einiges
- Initiale Zeitschätzungen waren ziemlich falsch
- Team-Zusammenarbeit war super ❤️