Amazon and flipkart Recommendation System for purchasing items

Inputs are the users, items sold, and ratings that they give. Output is the top x items that are recommended

Any E-commerce Platforms can use this model; Media and Entertaining platforms can also use them; any other service based platforms like healthcare, education, etc.

How it works:

Rank-Based Recommendations: Suggests popular products based on average ratings and minimum interactions.(If a event occurs people tend to buy product related to that event , example buying cake for new year, crackers for diwali etc).

Collaborative Filtering Recommendations:

User-based Collaborative Filtering: Recommends items by finding similar users based on cosine similarity.

Model-Based Collaborative Filtering: Uses Singular Value Decomposition (SVD) to predict ratings for unobserved items.

The model-based collaborative filtering (SVD) approach was evaluated using Root Mean Square Error (RMSE).

RMSE for SVD Model: `0.0136`, indicating low prediction error and good accuracy for sparse matrices.

Couldnt find any metric when collaborative filtering is done with cosine similarity

How the Recommendation System Works(important)

Methods Used:

1. Rank-Based Recommendation:
    - Calculates average ratings and counts of interactions for each product.
    - Sorts products based on average ratings and filters by minimum interaction threshold.

```
def top_n_products(final_rating, n, min_interaction):

    recommendations =
final_rating[final_rating['rating_count'] > min_interaction]

    recommendations =
recommendations.sort_values('avg_rating', ascending=False)
```
2.
```
    return recommendations.index[:n]
```

2.COSINE similarity based

Q) how many similar users it will and from each similar user how many products, is it predecided or any hyper parameter when we use cosine similarity

A)

The number of similar users and recommended products in user-based collaborative filtering are configurable hyperparameters, not fixed values. Here's how it works:

1. Key Hyperparameters

Parameter	range	Purpose	Implementation Example
`k` (similar users)	5-50	Controls how many neighbors influence recommendations	`similar_users(user_id, k=15)`
`min_similarity`	0.1-0.5	Filters out weakly similar users	`similarity_threshold=0.3`
`top_n` (recommendations)	5-20	Final number of items to suggest	`recommend_items(n=10)`

2. Product Selection Process

For a target user requesting recommendations:

1. Collect products from top `k` similar users
2. Exclude already interacted items
3. Score calculation:
4. $\text{Item Score} = \sum(\text{UserSimilarity} \times \text{Rating})$
5. $\text{Item Score} = \sum(\text{UserSimilarity} \times \text{Rating})$
6. Rank items by score
7. Return top `n` items

```
k_similar_users = 10
min_similarity = 0.25
top_recommendations = 5
```

```
# Get similar users
similar_users = find_similar_users(target_user,
k=k_similar_users, threshold=min_similarity)

# Aggregate products
candidate_items = get_unrated_items(target_user,
similar_users)

# Score and rank
recommendations =
rank_items(candidate_items)[:top_recommendations]
```

3. Tuning Considerations

A. Tradeoffs with `k`:

- Small k (5-10):
  - Faster computation
  - Risk of missing diverse preferences
- Large k (30-50):
  - Better coverage
  - Increased noise and computation time

B. Threshold Effects:

- Strict threshold (>0.4):
- python

  ```
  filtered_users = [u for u in similar_users if u.score >=
  0.4]
  ```
-
  - Higher quality recommendations

- May result in too few users
- Lenient threshold (>0.1):More candidate items
  - Risk of including irrelevant users

4. Optimization Strategies

1. Grid Search:
2. python

```
for k in [5, 10, 20]:
    for threshold in [0.2, 0.3, 0.4]:
        evaluate_model(k, threshold)
```

3.
4. Performance Metrics:
   - Precision@10: `0.32` (k=15) vs `0.28` (k=20)
   - Coverage: `45%` (threshold=0.3) vs `32%` (threshold=0.4)
5. Adaptive Thresholding (Advanced):
6. python

```
dynamic_threshold = np.percentile(similarity_scores, 75)
```

7.

5. Practical Recommendations

- Start with `k=15` and `min_similarity=0.3` as baseline
- Adjust based on dataset density:
  - Sparse data: Lower threshold (0.15-0.25)
  - Dense data: Higher threshold (0.3-0.4)
- Monitor runtime vs recommendation quality tradeoffs

These parameters require empirical tuning through A/B testing or cross-validation to balance recommendation relevance and computational efficiency168.

3.svd

Already discussed in class

U, s, Vt = svds(final_ratings_matrix, k=50)
sigma = np.diag(s)
all_user_predicted_ratings = np.dot(np.dot(U, sigma), Vt)

Metric code:

RMSE = mean_squared_error(rmse_df['Avg_actual_ratings'], rmse_df['Avg_predicted_ratings'], squared=False)

How is cosine similarity calculated.

```
def cosine_similarity(vector_A, vector_B):
        dot_product = np.dot(vector_A, vector_B)
    magnitude_A = np.linalg.norm(vector_A)
    magnitude_B = np.linalg.norm(vector_B)
    return dot_product / (magnitude_A * magnitude_B)
```