Movie Recommendation: Item-Item Collaborative Filtering

Mineração de Dados em Larga Escala
Universidade de Aveiro
2024/2025
Cristiano Nicolau, 108536

Introduction to the Problem and Item-Item Collaborative Filtering

Problem

Recommend new movies to users based on their past ratings and the ratings of others.

Item-Item CF

- Core Idea: "Users who liked movie X also tended to like movie Y."
- Computes similarity between pairs of items (movies). Predicts a user's rating for a movie based on their ratings of similar movies.

Challenge

Calculating similarity between all pairs of items is computationally expensive

- Complexity $O(N^2)$, where N is the number of items

General Approach and Data Preparation

Dataset: MovieLens (starting with 100k, scaling up to 1M, 10M, 20M).

Common Initial Steps:

- **Data Loading and Splitting:** 90% for training, 10% for testing.
- **Item Vectorization:** Each movie is represented as a sparse vector of user ratings.
- **Vector Dimensions:** Number of users.
- **Vector Values:** Ratings.
- **L2 Normalization:** Essential so that Euclidean distance approximates cosine similarity and for the proper functioning of both KMeans and LSH.

Approach 1 – LSH (Locality Sensitive Hashing)

Objective of LSH

Efficiently find approximately similar item pairs without computing all pairwise similarities.

How It Works

- Hash functions that tend to map similar items to the same "buckets".
- Uses Spark MLlib's BucketedRandomProjectionLSH.

Specific Process

- Train the LSH model on the L2-normalized item vectors.
- Use approxSimilarityJoin to find candidate pairs of similar items (based on a maximum Euclidean distance)
- Compute exact cosine similarity for these candidate pairs

Key Parameters: bucketLength, numHashTables

Approach 2 – Clustering (KMeans)

Objective of Clustering

Group similar items and restrict neighbor search to within the same cluster.

How It Works

- Uses the KMeans algorithm to partition items into *k* clusters.
- Items within the same cluster are assumed to be more similar to each other than to those in other clusters.

Specific Process

- Train the KMeans model on the L2-normalized item vectors (define *k*, e.g., based on the number of items).
- For each cluster, compute cosine similarity between all pairs of items within that cluster.

Key Parameters: K

Results and Comparison

Evaluation Metrics: RMSE (Root Mean Squared Error), MAE (Mean Absolute Error)

Dataset	Model	numHashTables	bucketLength	k	MAE	RMSE
MovieLens 1M	LSH	3	1		0.68	0.94
	Clustering			60	0.75	0.96
MovieLens 20M	LSH	3	1		0.61	0.84
	Clustering			100	0.69	0.90

Key Observations:

- Both approaches scaled well to larger datasets.
- **LSH** showed slightly better **RMSE/MAE**, especially on larger datasets.
- Execution times were high for large datasets but remained **feasible**.

Challenges and Conclusions

Main Challenges

- **Scalability and Resources:** Running the larger datasets (10M, 20M) required using more powerful platforms like Kaggle.
- **Memory Limitations:** The 25M dataset exceeded available memory and could not be fully processed.
- **Parameter Tuning:** Finding optimal values for LSH (bucketLength, numHashTables) and KMeans (k) was essential and often required several iterations.

Conclusions

- Both **LSH** and **KMeans clustering** are valid strategies to make **Item-Item Collaborative Filtering**
- They significantly reduce the search space for nearest neighbors.
- Choosing between LSH and Clustering often depends on the trade-off between **accuracy**, **implementation time**, and **parameter tuning complexity** for a specific dataset.

Thanks!

Do you have any questions?

CREDITS: This presentation template was created by **Slidesgo,** and includes icons, infographics & images by **Freepik**