Part 1. Classical Approaches To Recommendation

Olivier Koch

Criteo Research

June 28, 2018

Overview

1. Neighborhood methods
2. Matrix Factorization
3. Latent factor models
4. Hybrid methods

Problem statement

- Item recommendation with implicit or explicit feedback data
- Given interactions between users and items, find the k most relevant items for a given user.

More formally...

Given:

- $U = (u_1, ..., u_n)$, a group of n users
- $I = (i_1, ..., i_m)$, a group of m items
- $R_{u,i}$, a set of interactions between users and items

Find the k most relevant items for each user

Overview of methods

- Neighborhood-based methods
- Matrix Factorization
- Other latent factor models
- Hybrid methods

Lessons from the Nextflix Prize Challenge

- Open competition to beat the Netflix baseline (CineMatch)
- Training set: 100,480,507 ratings (480,189 users, 17,770 movies)
- Qualifying set: 2,817,131 ratings (50% test set, 50% quiz set)
- Improve by 10% to win $1,000,000

Lessons from the Netflix Prize Challenge, R. Bell and Y. Koren, SigKDD 2007

Lessons from the Nextflix Prize Challenge

- A trivial solution is almost as good as the baseline (average all ratings for a movie)
- Use ensemble methods
- In particular, combine neighborhood-based methods and MF
- Incorporate side information (e.g. time of rating)

Lessons from the Netflix Prize Challenge, R. Bell and Y. Koren, SigKDD 2007

Neighborhood methods

- User-oriented: find similar users
- Item-oriented: find similar items

Item-based recommendation

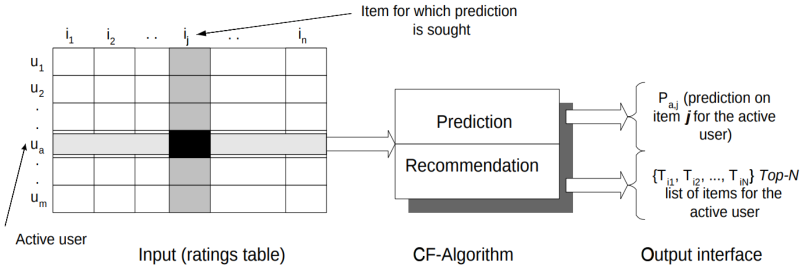
To predict the rating of u_i and i_j:

- Find the k most similar items to u_i
- Keep items that i_j has rated
- Weight each rating by the similarity between these items and i_j

$$P_{u,i} = \frac{\sum_{allsimilaritems} s_{i,j} \times R_{u,j}}{\sum_{allsimilaritems} s_{i,j}}$$

Item-based top-n recommendation algorithms, M.Deshpande and G. Karypis. Trans. Inf. Syst. 2004.

Item-based recommendation



Item for which prediction is sought
i_1 i_2 . . i_j . . i_n
u_1 u_2 . . u_a . . u_m
Active user
Input (ratings table)
Prediction
Recommendation
CF-Algorithm
$P_{a,j}$ (prediction on item ***j*** for the active user)
$\{T_{i1}, T_{i2}, \ldots, T_{iN}\}$ *Top-N* list of items for the active user
Output interface

Item-based collaborative filtering recommendation algorithms, B.M. Sarwar, G. Karypis, J.A. Konstan, and J. Reidl. WWW'2001.

Similarity definition

Cosine-based similarity

$$s_{i,j} = \cos(\vec{i}, \vec{j}) = \frac{\vec{i} \cdot \vec{j}}{\|\vec{i}\| \cdot \|\vec{j}\|}$$

Pearson similarity

$$s_{i,j} = \frac{\sum_{u \in U}(R_{u,i} - \overline{R_i})(R_{u,j} - \overline{R_j})}{\sqrt{\sum_{u \in U}(R_{u,i} - \overline{R_i})^2}\sqrt{\sum_{u \in U}(R_{u,j} - \overline{R_j})^2}}$$

Adjusted cosine similarity

$$s_{i,j} = \frac{\sum_{u \in U}(R_{u,i} - \overline{R_u})(R_{u,j} - \overline{R_u})}{\sqrt{\sum_{u \in U}(R_{u,i} - \overline{R_u})^2}\sqrt{\sum_{u \in U}(R_{u,j} - \overline{R_u})^2}}$$

Let's get hands-on...

Exercise 1: item-based collaborative filtering in Python

Neighborhood vs Matrix Factorization

- Neighborhood models detect localized relationships
- MF captures the totality of weak signals contained in the matrix
- Both methods can be fused (e.g. SVD++)
- Both methods have trouble making predictions for users with few ratings

Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model, Y. Koren, KDD'08

SVD vs Matrix Factorization

- SVD (Singular Value Decomposition) is closed form, requires a full-filled matrix (biased towards unseen observations)
- Matrix Factorization uses optimization and has been shown to work better (e.g. on known entries only)

Singular Value Decomposition

$$\mathbf{M} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^*$$

where **U** and **V** contain the left- and right-singular vectors of M.

Matrix factorization

- Find a low rank approximation to a sparse ratings matrix by minimizing a loss function on the known ratings
- Popular choice for applications in industry
- Simple
- Highly scalable to large datasets
- Captures global signals (but fails to capture strong local signals)

More formally...

$$R_{u,i} = U^T_{n \times k} * V_{k \times m}$$

where $U_{n \times k}$: user matrix and $V_{k \times m}$ item matrix

Steps for matrix factorization

- Build user-item matrix
- Define factorization model (cost function)
- Remove global mean
- Remove item bias and user bias
- Add regularization to prevent overfitting
- Minimize cost function (e.g. stochastic gradient descent, alternating least squares)

More formally...

Find U and V such that:

$$\min_{U,V} \sum_{(u,i)\in k} (R_{u,i} - \mu - b_i - b_u - U_i^T V_u)^2 + \lambda(\|U_i\|^2 + \|V_u\|^2 + b_i^2 + b_u^2)$$

Matrix Factorizations parameters

- cost function
- vector size
- learning rate
- regularization
- optimization method

Let's get hands-on...

Exercise 2: MF with tensorflow

Matrix Factorization for Implicit Feedback

- Model the probability that a user will interact with a given item with a logistic function
- Weigh zero-entries
- Optimize log likelihood with Alternating gradient descent
- Outperforms regular MF on small vectors (Spotify dataset)

Probabilistic Matrix Factorization, R. Salakhutdinov and A. Mnih, NIPS 2007

Probabilistic Matrix Factorization

- Scales linearly with number of observations
- Can incorporates the fact that users who rate similar sets of movies have similar preferences
- Evaluation on the Netflix dataset

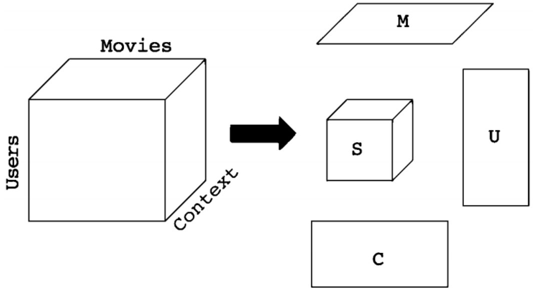
$$p(R \mid q, p, \sigma^2) = \prod_{i=1}^{N} \prod_{j=1}^{M} \left[\mathcal{N}(R_{ij} \mid U_i^T V_j, \sigma^2) \right]^{I_{ij}}$$

Latent factor models

- Probabilistic Latent Semantic Indexing (PLSA)
- Latent Dirichlet Allocation (LDA)
- Restricted Boltzmann Machines (RBM)

Contextual side information

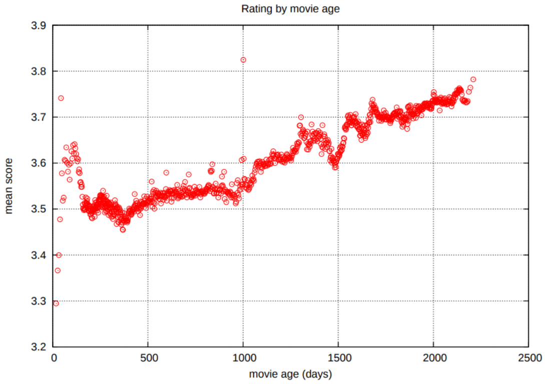
- High-order SVD optimized on observed entries only
- Batch sub-space descent $\rightarrow$ SGD for larger datasets



Multiverse Recommendation: N-dimensional Tensor Factorization for Context Aware Collaborative Filtering, Karatzoglou A. Amatriain X. Baltrunas L. Oliver N. RecSys 2010

Temporal dynamics

- Including temporal dynamics in user and item behaviors
- Applicable to neighborhood-based and factorization methods



Collaborative Filtering with Temporal Dynamics, Koren Y., SigKDD 2009

Unified Boltzmann machines

- Model the joint distribution of a set of binary variables through their pairwise interactions
- Encode collaborative and content information as features
- Learn weights that reflect how well each feature predicts user actions

A Unified Approach to Building Hybrid Recommender Systems, Gunawardana A. Meek C., RecSys 2009

Lessons from the Real World

- What problem are you trying to solve? Predicting DVD rentals is different from predicting instant streaming selection.
- Pick a single solution (one that fits your problem)
- There is often a deceptively simple baseline solution
- Improve on it incrementally
- Evaluate online early

Exercise 1: item-based recommendation with Movie Lens

Fetch the code and data:

https://github.com/oakfr/intro-to-reco

Exercise 1: item-based recommendation with Movie Lens

```
from sklearn.metrics import pairwise_distances
from scipy.spatial.distance import cosine, correlation

def compute_movies_similarities (method='cosine'):
    [...]
    movie_sim = # PUT YOUR CODE HERE
    return pd.DataFrame( movie_sim )
```

Exercise 1: item-based recommendation with Movie Lens

```
def get_similar_movies( sim_df, movieid, topN = 5 ):
    """ get top-N similar movies given an input movie ...
        sim_df is the output of compute_movies_similarities
    """
    movies_df = pd.read_csv( "data/ml-100k/u.item", ...)
    movies_df = movies_df.iloc[:,:2]
    movies_df.columns = ['movieid', 'title']
    movies_df['similarity'] = sim_df.iloc[movieid -1]
    movies_df.columns = ['movieid', 'title', 'similarity']
    top_n = # PUT YOUR CODE HERE
    return top_n
```

Exercise 1: item-based recommendation with Movie Lens

```
def predict_rating (rating_df_pivoted, movie_sim_df, user_ic
    """ predict rating for a user and a movie
    """
    similar_movies = get_similar_movies (...)
    sim_ratings = []
    sim_scores = []
    for row2 in similar_movies[1:].itertuples():
        _, item_id_2, _, similarity = row2
        sim_rating = get_rating (...)
        if not numpy.isnan (sim_rating):
            sim_ratings.append (...)
            sim_scores.append (...)
    if len(sim_ratings) > 0:
            return numpy.dot(...) / numpy.sum (...)
    return numpy.nan
```

Exercise 1: item-based recommendation with Movie Lens

```
def evaluate (rating_df_pivoted, movie_sim_df, num_ratings)
    """ predict ratings for the testing set and compute RMS
    """
    predicted_ratings = []
    true_ratings = []
    for row in ...[:num_ratings].itertuples():
        _, user_id, item_id, rating, _ = row
        predicted_rating = predict_rating (...)
        if not numpy.isnan (predicted_rating):
            predicted_ratings.append (predicted_rating)
            true_ratings.append (rating)

    rmse_val = rmse (...)
    return (rmse_val, len(predicted_ratings))
```

Exercise 1: item-based recommendation with Movie Lens

Bonus points:

- play with the parameters to improve the RMSE
- compare to random
- ignore movies with less than k scores

Exercise 2: MF with tensorflow

Given:

- data loader and matrix initialization (steps 1 and 2)
- default params settings, training and evaluation (steps 5 to 8)

Answer the following questions:

- Implement L1 and L2 losses (step 3)
- Implement L1 and L2 regularizations and evaluate (step 4)
- Bench the parameters around the default values

Exercise 2: MF with tensorflow

Hints for step 3 (with L2 norm)

```
diff_op = # use tf.subtract and tf.add with
          # result_values, mean_rating and
          # rates as variables
diff_op_squared = # use tf.square
base_cost = # use tf.reduce_sum
```

Exercise 2: MF with tensorflow

Hints for step 4 (with L2 norm)

```
l2_norm_sums = # use tf.add and tf.reduce_sum
               # with U and P
regularizer = # multiply l2_norm_sums and lamda
# regularizer_cost = # add base_cost and regularizer
```

Exercise 2: MF with tensorflow

Bonus points:

- How about another optimizer (e.g. AdamOptimizer)?
- Tune hyper parameters
- How should you set the number of epochs?