Adam Mickiewicz University
Poznań

Part 7: LightFM

Robert Kwieciński

Adam Mickiewicz University

May 29, 2021

Motivation for using hybrid recommender systems

Problems with collaborative models (e.g. classical Matrix Factorization):

- cold-start problem for users and items,
- ignores user and item features.

Motivation for using hybrid recommender systems

Problems with collaborative models (e.g. classical Matrix Factorization):

- cold-start problem for users and items,
- ignores user and item features.

Problems with content based approach:

- hard to catch dependencies which cannot be deduced from the content features (maybe the same people buy bikes and chocolate cakes),
- usually worse recommendations quality.

Motivation for using hybrid recommender systems

Problems with collaborative models (e.g. classical Matrix Factorization):

- cold-start problem for users and items,
- ignores user and item features.

Problems with content based approach:

- hard to catch dependencies which cannot be deduced from the content features (maybe the same people buy bikes and chocolate cakes),
- usually worse recommendations quality.

Solution

Hybrid recommender system like LightFM, which utilizes both collaborative and content data.

LightFM model

The full description you can read in the original paper [1].
In short words prediction is given by:

$$\hat{r}_{ui} = \mathbf{q}_u \cdot \mathbf{p}_i + b_u + b_i,$$

where $\mathbf{q}_u$ is the user latent vector, $\mathbf{p}_i$ is the item latent vector b_u is the user bias, b_i is the item bias.

Each user latent vector is a (weighted) sum of feature embeddings of each of his features.
The same rule applies to items and biases.

For example embedding of the item "white rabbit" might be a sum of the embedding of white and the embedding of rabbit.

Example-number of parameters

For our Movielens dataset we will use the following item features:

- genre (19 possibilities),
- data of release (240 possibilities).

As user features we'll take:

- age (61 possibilities),
- sex (2 possibilities),
- profession (21 possibilities),
- zip-code (795 possibilities).

Example-number of parameters

For our Movielens dataset we will use the following item features:

- genre (19 possibilities),
- data of release (240 possibilities).

As user features we'll take:

- age (61 possibilities),
- sex (2 possibilities),
- profession (21 possibilities),
- zip-code (795 possibilities).

If we take 10 latent factors for each user/item feature our model will have:

- $10 \cdot (19 + 240) = 2590$ parameters needed for item latent vectors,
- number of item features (259) parameters for item biases,
- $10 \cdot (61 + 2 + 21 + 795) = 8790$ parameters needed for user latent vectors,
- number of user features (879) parameters for user biases.

LightFM as a generalization of MF

We can treat as a feature being a given user or an item. In this case we will have:

- number of user features equals number of users,
- number of item features equals number of items.

Then LightFM model (possibly assuming biases equals 0) reduces to MF model.

Note that we can combine both types of discussed features.

BPR-OPT definition

In the paper BPR: Bayesian Personalized Ranking from Implicit Feedback [2] novel optimization criterion was introduced. It could be applied for various existing models, including Matrix Factorization.

Denote by D the set of all triples (u, i, j), where u interacted with i, but not interacted with j.

$$\text{BPR-OPT} = \sum_{(u,i,j) \in D} \ln \sigma(\hat{r}_{ui} - \hat{r}_{uj}) - \lambda ||\Theta||^2,$$

where $\hat{r}_{ui}$ is estimated by a given model and Θ represents the parameter vector.

BPR-OPT learning

For optimizing BPR-OPT we use the stochastic gradient ascent algorithm based on bootstrap sampling of training triples.

In simple words:

- we choose triple $(u, i, j) \in D$ randomly with equal probability for each triple,
- we compute a gradient of BPR-OPT with respect to our model parameters and update the parameters (gradient ascent),
- we repeat these steps until convergence.

BPR-OPT learning

For optimizing BPR-OPT we use the stochastic gradient ascent algorithm based on bootstrap sampling of training triples.

In simple words:

- we choose triple $(u, i, j) \in D$ randomly with equal probability for each triple,
- we compute a gradient of BPR-OPT with respect to our model parameters and update the parameters (gradient ascent),
- we repeat these steps until convergence.

In the paper it was shown that optimizing BPR-OPT is similar to optimizing AUC.

WARP loss

WARP loss (Weighted Approximately Ranked Pairwise Ranking Loss) is also designed for ranking optimization by focusing on triples (u, i, j), where the item i is a positive example for the user u and the item j is a negative example - for example user u interacted with i, but has not interacted with j.

The general idea is as follows:

- pick a random pair (u, i) where i is a positive example for u,
- sample negative items as long as score of sampled negative item j exceeds $\hat{r}_{ui}$,
- update model parameters based on a loss function

$$(\hat{r}_{uj} - \hat{r}_{ui}) \ln(\frac{|I| - 1}{N}),$$

where I is a set of all items and N is a number of samples needed to find j.

WARP loss

WARP loss (Weighted Approximately Ranked Pairwise Ranking Loss) is also designed for ranking optimization by focusing on triples (u, i, j), where the item i is a positive example for the user u and the item j is a negative example - for example user u interacted with i, but has not interacted with j.

The general idea is as follows:

- pick a random pair (u, i) where i is a positive example for u,
- sample negative items as long as score of sampled negative item j exceeds $\hat{r}_{ui}$,
- update model parameters based on a loss function

$$(\hat{r}_{uj} - \hat{r}_{ui}) \ln(\frac{|I| - 1}{N}),$$

where I is a set of all items and N is a number of samples needed to find j.

Read some details and properties of WARP implementation in LightFM [3].

Python implementation

To do (especially for absent students):

- Go through - *P7. LightFM* notebook to:
  - check implementation of LightFM model using LightFM library - especially user and item feature preparations
  - compare the results using different loss functions
  - observe evaluation measures for LightFM model with different features

Passing the course

- Preferable form is the copy of the course repository with solutions added in the Jupyter Notebooks in the place where the given task is presented.
- Each correctly solved task is worth 1 point. In case of mistakes 0, 0.5 or 1 point will be given.
- The final grade is:
  - 3 - for 3 points
  - 3.5 - for 3.5 points
  - 4 - for 4 points
  - 4.5 - for 4.5 points
  - 5 - for 5 and more points
- Solutions should be sent to robkwi@st.amu.edu.pl with [WSR] in the title. Please sent just a link to the repository with solutions on git.wmi.amu.edu.pl. Please remember to give me the access (robkw).
- The deadline is **12.06.2021** (2 weeks after the last meeting).

List of the tasks

List of the tasks:

- project task 1: implement self-made BaselineIU
- project task 2: implement some other evaluation measure
- project task 3: use a version of your choice of Surprise KNN algorithm
- project task 4: implement SVD on top baseline
- project task 5: generate recommendations of RP3Beta for hyperparameters found to optimize recall
- project task 6 (optional): implement graph-based model of your choice
- project task 7 (optional): check how the number of iterations of WRMF model influence the evaluation metrics

The full description of the tasks can be found in the corresponding Jupyter Notebooks.

References I

[1] M. Kula, "Metadata embeddings for user and item cold-start recommendations", vol. 1448, Jul. 2015.

[2] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "Bpr: Bayesian personalized ranking from implicit feedback," *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence, UAI 2009*, May 2012.

[3] L. Maciej Kula, "Learning-to-rank using the warp loss", https://github.com/lyst/lightfm/blob/master/examples/movielens/warp_loss.ipynb.