Part 1: Introduction and baseline

Robert Kwieciński

OLX Group and Adam Mickiewicz University

May 22, 2020

Passing the workshops

- Students will be asked to do 7 tasks which they can complete during the workshops or finish later.
- All solutions should be sent to r.kwiecinskipl@gmail.com with [MLRS] in the title.
  Preferable form is a link to the repository on git.wmi.amu.edu.pl.
  Please remember to give me the access (robkw).
- Each correctly solved task is worth 1 point. In case of mistakes 0, 0.5 or 1 point will be given.
- Grade from this part of workshops 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
- The deadline is **14.06**.

Examples of recommender systems

Some examples:

- movie recommendations (Netflix),
- friends suggestions (Facebook, LinkedIn),
- products recommendations (Amazon),
- job recommendations (OLX),
- playlists building (Spotify),
- recipe recommendations.

Business examples

Netflix

75% of movies watched on Netflix came from recommendations. Netflix said that they save yearly about 1 billion dollars thanks to recommendations.

Amazon

Changes in recommender system in 2016 were crucial for 29% increase in sale.
Amazon credits recommender systems with 35% of their total revenue.

Source: `https://sigmoidal.io/recommender-systems-recommendation-engine/`

Properties of a good recommender system

It strongly depends on the use case. Some examples are:

- scalability,
- fast updating (after each rating, user/item profile change),
- recommending new users/items,
- good predictions.

Typical problems in recommender systems

- sparsity (sometimes more than 99,99% unknown (item, user) pairs),
- long-tail (most of the items have a small number of ratings),
- cold-start (new items/users),
- changing preferences (over day/week, after user already bought something),
- attack resistance.

Recommender systems - what is it?

Recommender system should suggest users' items which they will like.
Suppose we have M users, N items and matrix $R \in \mathbb{R}^{M \times N}$:

$$\mathrm{R} = \begin{bmatrix} r_{1,1} & ? & r_{1,3} & \dots & r_{1,M} \\ r_{2,1} & r_{2,2} & ? & \dots & ? \\ ? & r_{3,2} & ? & \dots & r_{2,M} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ r_{M,1} & ? & r_{M,2} & \ddots & r_{M,N} \end{bmatrix},$$

where $r_{u,i}$ denotes the preference of user u to item i. The most of entries $r_{u,i}$ are unknown, because most of the users have not rated most of the items.
Crucial part of the recommendation problem is to estimate missing entries of the matrix R. Namely, predicts rating $\hat{r}_{u,i}$ of $r_{u,i}$ using information about previous ratings, items' and users' properties.

Approaches

Most commonly we distinguish:

- **Content-based filtering** - we use some similarity measures of the items based on their attributes. Usually we use given user's ratings. In case when we do not use any ratings we call the system **knowledge-based**.
- **Collaborative filtering** - we use some measure of similarity based on the ratings (for example we recommend items which similar users already liked).
- **Hybrid** - when we mix both of these approaches.

Difference between collaborative and content-based filtering

Sample content-based approach

Let $R(u,i)$ be the set of k movies which are already rated by user u and have the **greatest number of common actors with movie** i.
Predict a rating $\hat{r}_{u,i}$ as

$$\hat{r}_{u,i} = \frac{1}{k} \sum_{t \in R(u,i)} r_{u,t}.$$

Sample collaborative approach

Let $R(u,i)$ be the set of k movies which are already rated by user u and have the **greatest number of users who rated movie** i **and given movie**.

Content-based filtering

Content-based filtering

Show me more items similar to what I have liked.

Pros:

- no cold-start problem for items,
- no risk of recommending totally irrelevant items.

Cons:

- knowledge about items needed,
- cold-start problem for users (if the system is not knowledge-based),
- no surprises.

Collaborative filtering

Collaborative filtering

Show me items based on the other users' ratings.

- Memory based - we need to go through all ratings in memory to make predictions. It is sometimes called neighborhood based approach. There are two main conceptions:
  - user-based - looking for similar users ("show me items liked by users similar to me"),
  - item-based - looking for similar items ("show me items which are rated similar to items I have rated high").
- Model based - we do not store everything in memory, but we have some model (Matrix Factorization, Restricted Boltzmann Machine).
- Hybrid.

Model-based models usually have many parameters, whereas memory based models might have only hyperparameters.

Implicit and explicit datasets

Usually different models are used depending on type of feedback from users.

Explicit feedback

Explicit feedback - user directly specify his preference towards the product by liking/disliking or giving 1-5 stars.

Implicit feedback

Implicit feedback - users preference are not directly specified, but shown by purchase/visiting the site/reading longer than x minutes etc.

Implicit and explicit datasets

Usually different models are used depending on type of feedback from users.

Explicit feedback

Explicit feedback - user directly specify his preference towards the product by liking/disliking or giving 1-5 stars.

Implicit feedback

Implicit feedback - users preference are not directly specified, but shown by purchase/visiting the site/reading longer than x minutes etc.

In implicit case:

- usually lack of negative feedback,
- much more data,
- noise (clicking accidently),
- assumption that user probably does not like unseen items,
- different evaluation measures.

Netflix Prize Competition

Problem

Improve RMSE of prediction of users' ratings on movies.

Netflix Prize Competition

Problem

Improve RMSE of prediction of users' ratings on movies.

Prize

1 million dollars for improvement by 10%.

Netflix Prize Competition

Problem

Improve RMSE of prediction of users' ratings on movies.

Prize

1 million dollars for improvement by 10%.

Training set

- 100 480 507 ratings (from 1 to 5) of 17 770 movies by 480 189 users (user id, movie id, rating, date)
- Information about movies (movie id, date of release, title)

Winner algorithm

Winners

On July 26, 2009 BellKor's Pragmatic Chaos won with RMSE=10.06%. Full description of algorithm can be found in [1] (other results in [2]).

Some conclusions

- Best results after blending many models (101 models blended).
- Good model - if decreases RMSE after blending with previous models. Not necessarily with low RMSE itself.
- Train model on residuals of other models.
- Many models learned without gradient descent (APT).

Winner algorithm

Winners

On July 26, 2009 BellKor's Pragmatic Chaos won with RMSE=10.06%. Full description of algorithm can be found in [1] (other results in [2]).

Some conclusions

- Best results after blending many models (101 models blended).
- Good model - if decreases RMSE after blending with previous models. Not necessarily with low RMSE itself.
- Train model on residuals of other models.
- Many models learned without gradient descent (APT).

Despite the best RMSE Netflix has never implemented model in production due to its complexity (and also new sources of valuable clickstream data, which have not been used in the model).

Global Effects

Before training model it might be good to exclude some simple dependencies like:

- user's/movie's average rating,
- user's/movie's change in average rating over time,
- impact of movie average/ratings on user's rating.

For user specific parameters we set:

$$r_{u,i} = \theta_u x_{u,i} + error.$$

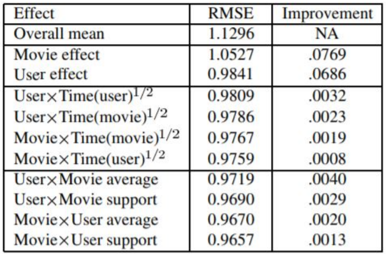
Then in the next model we use *error* instead of $r_{u,i}$.

Example - User $\times$ Time(user)$^{\frac{1}{2}}$

$$x_{u,i} = \sqrt{t_{u,i}} - \frac{1}{|N(u)|} \sum_{j \in N(u)} \sqrt{t_{u,j}},$$

$t_{u,j}$ is a number of days between first user's u rating and rating item j.

Global Effects - Netflix



Effect	RMSE	Improvement
Overall mean	1.1296	NA
Movie effect	1.0527	.0769
User effect	0.9841	.0686
User$\times$Time(user)$^{1/2}$	0.9809	.0032
User$\times$Time(movie)$^{1/2}$	0.9786	.0023
Movie$\times$Time(movie)$^{1/2}$	0.9767	.0019
Movie$\times$Time(user)$^{1/2}$	0.9759	.0008
User$\times$Movie average	0.9719	.0040
User$\times$Movie support	0.9690	.0029
Movie$\times$User average	0.9670	.0020
Movie$\times$User support	0.9657	.0013

Figure: RMSE for Netflix probe data after adding a series of global effects to the model introduced in [3].

Note that the RMSE Cinematch achieved on the probe dataset is 0.9474.

Python implementation

To do (especially for absent students):

- Go through - *P0. Data preparation* notebook to:
  - split data for train and test
  - understand the data structure and properties
- Go through - *P1. Baseline* notebook to:
  - preprocess data
  - learn about Scipy sparse matrices
  - look at implementation of simple recommender systems: TopPopular, GlobalAverage
  - **project task 1: implement TopRated**
  - look at implementation of self-made BaselineUI
  - **project task 2: implement self-made BaselineIU**
  - read next 2 slides to understand Surprise implementation of Baseline and Random
  - look at Surprise ready-made implementations in the notebook

Surprise-Baseline

The prediction is:

$$\hat{r}_{ui} = \mu + b_u + b_i,$$

where μ, b_u and b_i are parameters minimizing:

$$\sum_{(u,i)\in R} (r_{ui} - \mu - b_u - b_i)^2 + \lambda(b_i^2 + b_u^2),$$

where λ is used for regularization and R is a set of training pairs (u, i)
Usually we find them by stochastic gradient descent (sgd).

Surprise-Baseline

The prediction is:

$$\hat{r}_{ui} = \mu + b_u + b_i,$$

where μ, b_u and b_i are parameters minimizing:

$$\sum_{(u,i)\in R} (r_{ui} - \mu - b_u - b_i)^2 + \lambda(b_i^2 + b_u^2),$$

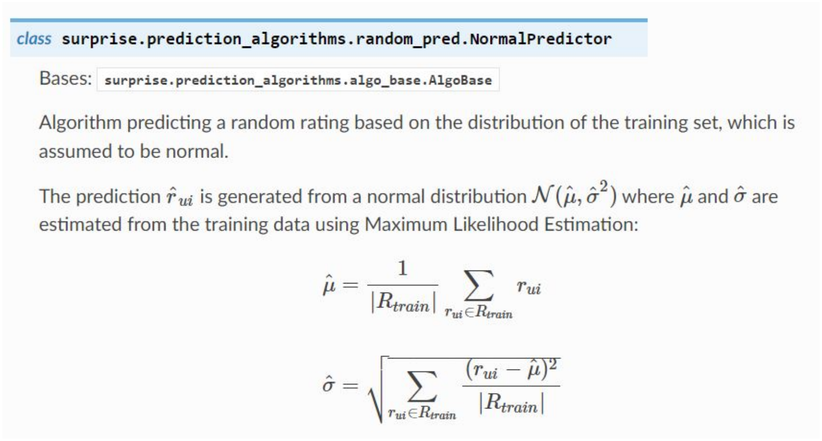
where λ is used for regularization and R is a set of training pairs (u, i)
Usually we find them by stochastic gradient descent (sgd).

Instead of sgd we can minimize the function in different procedure (als).
We can initialize weights (in Surprise: to zeros) and consecutively set:

$$b_i = \frac{\sum_{(u,i)\in R}(r_{ui} - \mu - b_u)}{\lambda_2 + |\{u|(u,i) \in R\}|},$$

$$b_u = \frac{\sum_{(u,i)\in R}(r_{ui} - \mu - b_i)}{\lambda_3 + |\{i|(u,i) \in R\}|}.$$

Surprise-Random



class `surprise.prediction_algorithms.random_pred.NormalPredictor`

Bases: `surprise.prediction_algorithms.algo_base.AlgoBase`

Algorithm predicting a random rating based on the distribution of the training set, which is assumed to be normal.

The prediction $\hat{r}_{ui}$ is generated from a normal distribution $\mathcal{N}(\hat{\mu}, \hat{\sigma}^2)$ where $\hat{\mu}$ and $\hat{\sigma}$ are estimated from the training data using Maximum Likelihood Estimation:

$$\hat{\mu} = \frac{1}{|R_{train}|} \sum_{r_{ui} \in R_{train}} r_{ui}$$

$$\hat{\sigma} = \sqrt{\sum_{r_{ui} \in R_{train}} \frac{(r_{ui} - \hat{\mu})^2}{|R_{train}|}}$$

Figure: Source: surprise package documentation [4]

ADAM MICKIEWICZ UNIVERSITY POZNAŃ

References I

[1] A. Töscher and M. Jahrer, "The bigchaos solution to the netflix grand prize,", Sep. 2009, http://https://www.netflixprize.com/assets/GrandPrize2009_BPC_BigChaos.pdf/.

[2] Netflix, "Netflix prize competition,", https://netflixprize.com/.

[3] R. M. Bell and Y. Koren, "Scalable collaborative filtering with jointly derived neighborhood interpolation weights," in *Proceedings of the 2007 Seventh IEEE International Conference on Data Mining*, ser. ICDM '07, Washington, DC, USA: IEEE Computer Society, 2007. [Online]. Available: https://doi.org/10.1109/ICDM.2007.90.

[4] Surprise, "Surprise package documentation,", https://surprise.readthedocs.io/.