1 Introduction

In this project we want to take a shot on writting a system that recommends items to users. This kind of system is generally called recommendation system. In the vast domain of recommendation systems we specialize in the subdomain of collborative filtering. The is characterized by the makings of the input data. Where recommendation systems try to utilize all kinds of input data, in collborative filtering (CF) the input data only includes the user-interaction with the items in the form of ratings, as opposed to metadata, e.g. gender, country or state of residence or age.

In this, the input data for our system takes the form of a Matrix $I \in \mathbb{K}^{(u,i)}$ where $\mathbb{K}$ is the space of all forms a rating can take, e.g. $\mathbb{K} = \{0, \dots, 5\}$. This is typically an interval scale or a binary as like and dislike. Ratings The dimension of I are determined by the number of users u and the count of items i a user can rate. Typically this matrix is sparse as users only rate some of the items.

2 Method

The range of types of models for CF is vast. The currently best performing models are mosty ensemble methods that combine the strengths of multiple other models. Two of the good performing simple models that are part of those ensembles are models based on matrix factorization or approaches utilizing a distance measure between items and/or users to recommend based on a neighborhood. A good overview can be found in Töscher and Jahrer, 2008, that presentend their solution to the Netflix price challange as a large and well explained ensemble.

For this article we will implement a basic model that combines the strengths of neighborhood-models and matrix factorization by extending the widely known SVD++-Model. It has been proposed in Koren, 2008, a researcher part of the team BelKor that won the Netflix prize.

2.1 Datastructures

The following strcutures are used to model the CF problem, where k is the number neighbors taken into account and f is the count of latent factors estimated in the training process.

$b_{ui} = \mu + b_u + b_i$

$R(u)$ Function that delivers all items that user u rated.

$N(u)$ Function that gives all items that a user provided implicit feedback for.

$S^k(i)$ Function that returns the set of k item most similiar to item i in terms of the slimiarity-measure s_{ij}.

$R^k(u)$ Function that computes $R(u) \cap S^k(i)$.

$N^k(u)$ Function that computes $N(u) \cap S^k(i)$

μ The global average of the training ratings.

b_u The observed deviations of user u from the mean.

b_i The observed deviations of item i from the mean.

r_{ui} The rating of user u for item i.

$\hat{r}_{ui}$ The estimated rating user u for item i.

e_{ui} The error of the estimation to the real rating $r_{ui} - \hat{r}_{ui}$.

b_{ui} The baseline estimate for a rating r_{ui} is $\mu + b_u + b_i$

w_{ij} Global user-indipendent weights for the interaction between items.

c_{ij} Global user-indipendent weights for implicit preference.

q_i A factor vector for each item.

p_u A factor Vector for each user.

s_{ij} The matrix holding the similiarity between items i, j.

n_{ij} The matrix counting how many times user have rated both items i, j.

p_{ij} The matrix holding the pearson correlation between items i, j.

The neighborhood for the function $S^k(i)$ is defined by the similiarity measure $s_{ij} = \frac{n_{ij}}{n_{ij}+\lambda_2} \cdot p_{ij}$. The pearson-correlation is weighted by the count of users that rated both items i, j.

Generally other measurements for the similiarity can be used, e.g. cosine or manhattan distance.

2.2 Prediction

The prediction for an unknown rating $\hat{r}_{ui}$ is modelled as follows;

$$
\begin{aligned}
\hat{r} &= \mu + b_u + b_i && (1) \\
&+ q_i^T \left(p_u + |N(u)|^{-1/2} \sum_{j \in N(u)} y_j \right) && (2) \\
&+ |R^k(i;u)|^{-1/2} \sum_{j \in R^k(i;u)} (r_{uj} - b_{uj}) \, w_{ij} && (3) \\
&+ |N^k(i;u)|^{-1/2} \sum_{j \in N^k(i;u)} c_{ij} && (4)
\end{aligned}
$$

User and Item properties Term (1) models the basic properties of users and items without interactions.

User and Item Interactions Term (2) describes the interaction between user and item factors. It represents the factorizaion part of the model.

Weighted nearest items Term (3) takes the nearest items into account weighted by the trained values in w_{ij}.

Implicit preference Term (4) introduces the implicit preferences.

2.3 Training

For the training process Koren suggests the following variation of gradient descent. Unfortunately he objective function is non-convex, this leaves us in uncertainty about the count of (local) extrema.

$$
\begin{aligned}
b_u &\leftarrow b_u + \gamma_1 \cdot (e_{ui} - \lambda_6 \cdot b_u) \\
b_i &\leftarrow b_i + \gamma_1 \cdot (e_{ui} - \lambda_6 \cdot b_i) \\
q_i &\leftarrow q_i + \gamma_2 \cdot (e_{ui} \cdot (p_u + |N(u)|^{-1/2} \sum_{j \in N(u)} y_j) - \lambda_7 \cdot q_i) \\
p_u &\leftarrow p_u + \gamma_2 \cdot (q_{ui} \cdot q_i - \lambda_7 \cdot y_j) \\
&\forall j \in N(u): \\
y_j &\leftarrow y_j + \gamma_2 \cdot (e_{ui} \cdot |N(u)|^{-1/2} \cdot q_i - \lambda_7 \cdot y_j) \\
&\forall j \in R^k(u): \\
w_{ij} &\leftarrow w_{ij} + \gamma_3 \cdot (|R^k(i;u)|^{-1/2} \cdot e_{ui} \cdot (r_{uj} - b_{uj}) - \lambda_8 \cdot w_{ij}) \\
&\forall j \in N^k(u): \\
c_{ij} &\leftarrow cij + \gamma_3 \cdot (|N^k(i;u)|^{-1/2} \cdot e_{ui} - \lambda_8 \cdot c_{ij})
\end{aligned}
\tag{5}
$$

The time for one iteration dependes on the count of factors to be learned. On an Intel Xeon CPU E5-2630 v2 we get roughly about processed ratings per second. A whole iteration over the dataset with 434641 ratings takes 2.414 hours. For the netflix dataset Koren suggests about 30 iterations. This leads to a training time of about three days.

Potentially the training time could be reduced massively by parallelization. As updates of the datastructures are applied for every training point one could partition the training set in batches and apply the updates in parallel while excluding the case of multiple threads working on the same user/item.

2.4 Parameter Estimation

One of the hardest problems with the implementation of the Hybrid model is the large number of parameters. We have to choose values for the count of iterations and factors, as well as the eleven lambdas and gammas steering the training of our model. Hyperparameter tuning is gernerally a hard problem as it is still subject to active research.

We will try a rather simple ad-hoc approach and use a derivative-less optimization method on the cross-validation RMSE-error calculated on a small subset of the dataset. Using only a subset of the training data does not guarantee good generalization on the whole dataset, but a full training run takes days we have too use less samples to make cross-validation feasible.

3 Implementation

To ease the training and evaluation of recommender models we implement an abstract base class `Recommender` that provides the abstract function `fit` and `predict`. Furthmore we add a function `save` to serialize our model and save it into files. Given `fit` and `predict` it should be straigt forward to implement a cross validation. Nonetheless, in the context of recommendation systems we have to be aware of the concept of multiple ratings by the same user. Generally, in Machine Learning we try to predict the performance of algorithms by a strict divide between test and trainig-data. If we have interactions of the same user in the test and in the training set, this concept is kind of broken. On the other hand, in this case, our algorithm performs good because of the knowledge about the user. If nothing about the previous interactions of the user known, we can not take the neighborhood into account. In this case won't partition the set by user but in other settings it could be necessary. This concept is referred to as *strong generalization* by **Liang2018** We use `numpy.seterr(all='raise')` to avoid proceeding after overflows or occurences of np.nan, as choosing the wrong parameters can lead to numerical errors that we want to be aware of.

References

Koren, Yehuda (2008). "Factorization meets the neighborhood: a multifaceted collaborative filtering model". In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pp. 426–434.

Liang, Dawen et al. (Feb. 16, 2018). "Variational Autoencoders for Collaborative Filtering". In:

Töscher, Andreas and Michael Jahrer (2008). "The bigchaos solution to the netflix prize 2008". In: *Netflix Prize, Report*.