Video Games Collaborative Filtering Recommendation System

Tim Laibacher and Craig Macartney

Problem Description and Motivation

- Design and evaluate a Recommendation System, at scale, by implementing and applying a Spark Machine Learning Pipeline
- Recommendation System evaluated by assessing its performance in the task of predicting user/item ratings by using matrix factorization via Alternating Least Squares (ALS)
- → users and products are described by a small set of latent factors that can be used to predict missing entries
- Pipeline evaluated with respect to preprocessing, parametrisation, and scaling

Initial Observations, Descriptive Statistics and Dataset Preparation

Analysis and Preparation of Dataset

Dataset: Amazon review dataset – Video Games

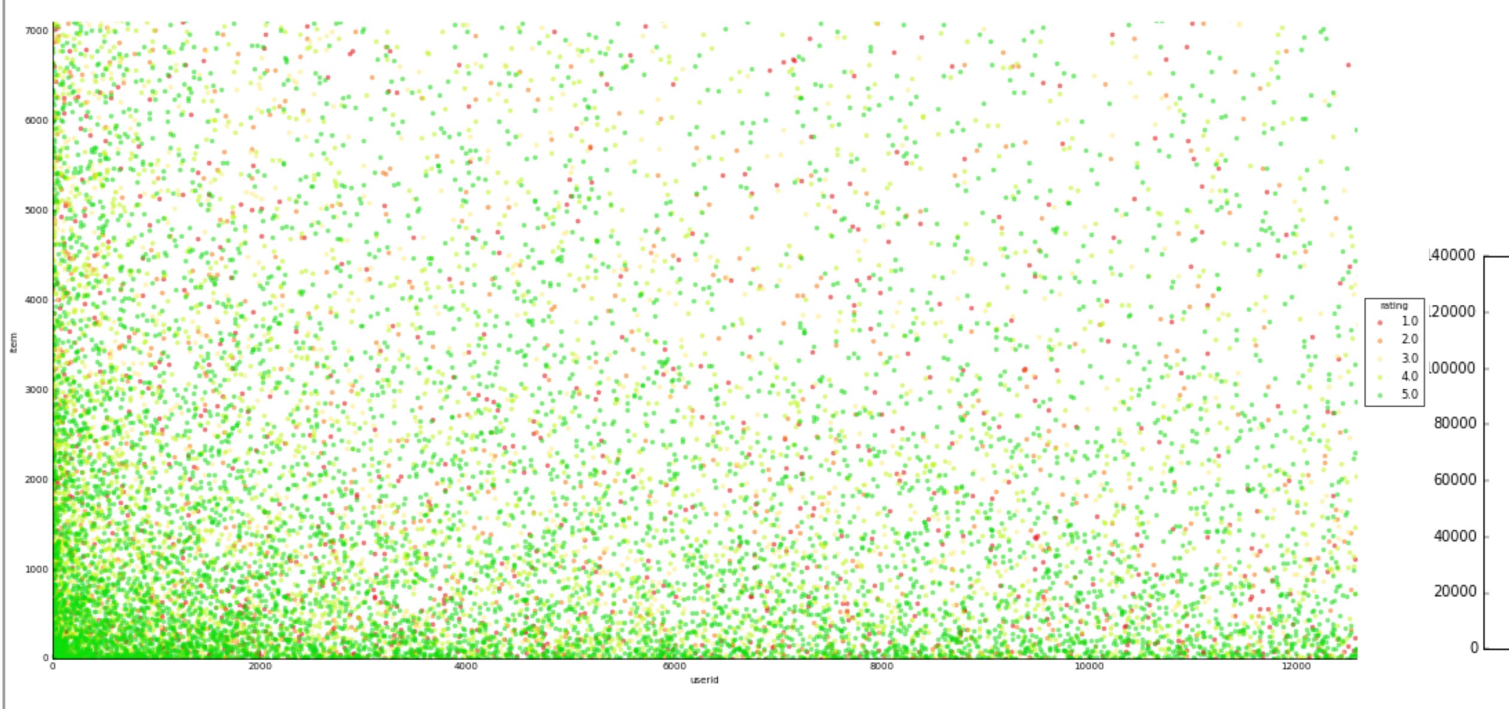
- Explicitly-gathered ratings and reviews described by 9 variables. Reduced to 3 discrete variables employed in task: asin (item ID), rating (explicit, out of 5), user (user ID). N.B. ratings considered as continuous scale from 1.0 to 5.0 (inclusive) for purposes of task.
- Task: recommending video games to individual users. Important to help users navigate "infinite shelf space" (Anderson, 2004) to receive benefit of online store's "Long Tail" advantage over brick-and-mortar stores (Leskovec et al., 2014). More formally, to fill-in missing entries of a user-item association matrix.
- Method: ALS algorithm for a Matrix-Factorization-based Latent Factor model for Collaborative Filtering (Apache Spark, 2018).

Descriptive Statistics

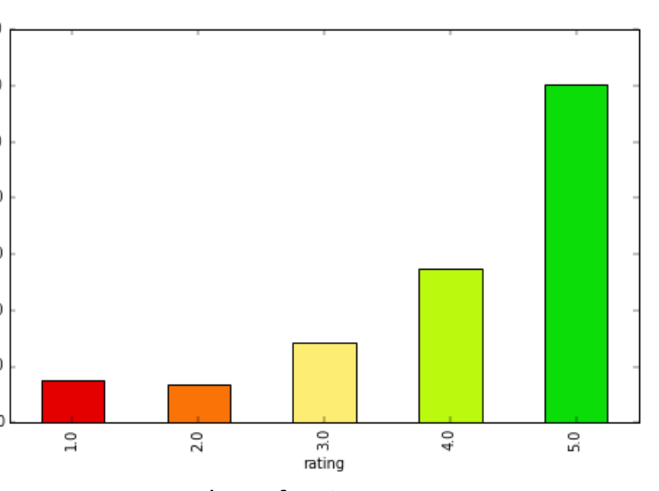
- 231,780 observations with the following ratings imbalance:

1) 14,853 *(6.4%)*
2) 13,663 *(5.9%)*
3) 28,275 *(12.2%)*
4) 54,804 *(23.6%)*
5) 120,185 *(51.9%)*

	rating
count	231,780
mean	4.09
std.dev.	1.20
min	1.0
max	5.0



Visualization of items, users and ratings on a 10% subset of the complete training data



Number of ratings per category

Processing Steps and Spark Machine Learning Pipeline

Processing Steps

- Transform alphanumeric ASIN and user IDs transformed into unique numerical IDs with SparkML's "StringIndexer", as required for Spark's ALS
- 70%:30% train:test split – i.e. 162,340 for training and 69,440 held-out
  => Matrix size: 259,080,096 I % of sparse matrix filled: 0.063% I # distinct users: 24,288 I # distinct items: 10,667
- Cold-Start Strategy – drop rows in the dataframe of predictions containing NaN values.

Parameter Settings

Grid-search through following parameter values (as per Apache Spark (2018)):

- numBlocks – default of 10 partition blocks of users and items for parallelization
- rank [1, 5, 10, 50, 70] – set of latent factors that can be used to predict missing entries – hidden features obtained via UV-decomposition using ALS algorithm (rank-based dimensionality reduction (number of linearly independent matrix columns yielding basis vectors))
- maxIter [15] – stopping criteria fixed to 15 iterations
- regParam [0.05, 0.1, 0.5, 5] – weighted-λ-regularization (ALS-WR) hyperparameter to avoid overfitting (Zhou et al, 2008).

Also:

- downsamples [1%, 10%, 5%, 80%] – the pipeline is run on this set of downsamples of the training set
- Postprocessing Bias – global bias correction technique to prediction solution (Zhou et al, 2008), inherent in Spark's ALS

Learning Algorithm

ALS algorithm for Matrix Factorization model

- Approximates user *u*'s rating of item *i.*
- Following Zhou et al, 2008, by alternating back and forth, updating users and items, we can robustly achieve rank-based dimension reduction, as suitable values for matrix decomposition are iteratively calculated – it becomes a convex optimisation problem (Koren et al, 2009).
- Objective function: $$f(U, M) = \sum_{(i,j)\in I} (r_{ij} - \mathbf{u}_i^T \mathbf{m}_j)^2 + \lambda \left(\sum_i n_{u_i} \|\mathbf{u}_i\|^2 + \sum_j n_{m_j} \|\mathbf{m}_j\|^2 \right)$$

"Step 1. Initialize matrix M by assigning the average rating for that movie as the first row, and small random numbers for the remaining entries.
Step 2. Fix M, Solve for U by minimizing the objective function;
Step 3. Fix U, solve for M by minimizing the objective function similarly;
Step 4. Repeat Steps 2 and 3 until a stopping criterion is satisfied." (Zhou et al, 2008)

- Compute distributed, updated matrices U and M (Ibid), in parallelized form via Spark's ALS
- Matrix Factorisation and reduced rank representation in ALS allows for efficiency and scalability
  - ALS is easily parallelized and preserves sparse structure of known matrix elements – storage-efficient (Zhou et al, 2008). Scalability of models is important for companies such as Amazon (Linden et al, 2003).
- Works well with explicit feedback (as well as with implicit)

Hypothesis Statements

1. Expected Percentage of Training Set to drive majority of material difference in performance, with large differences between Test and Validation RMSEs for small training set sizes.
2. ALS model to outperform baseline global average RMSE
3. Increase in wall-clock time with increasing number of training items
4. As per Zhou et al (2008), model shouldn't overfit training set with sufficiently large number of iterations or hidden features (latent factors).

Choice of Training, Parameter Choice and Model Selection

Training

- To minimize the sum of squared differences between the known elements and the corresponding elements of the factored low rank matrix, ALS has proven to be an effective approach (Zhou et al, 2009)
- The fast runtime achieved through parallelization is a competitive advantage for model building and parameter tuning in general

Model Selection

TrainValidationSplit (to reduce running times when compared with CV, as evaluates each combination of parameters once, rather than *k* times. but will not produce as reliable results when the training dataset is not sufficiently large)

Parameter Grid

- Varying at least one feature preprocessing step, one machine learning parameter, and the training set size (e.g. using the sample methods of RDD or DataFrame)
- Spark's 'rank' (# hidden variables) and 'regParam' (λ) are the only tunable parameters of ALS-WR (Zhou et al, 2009)
- Optimal λ value obtained through trial and error (Ibid)
- Fixed, arbitrary, stopping criteria used; in contrast to Zhou et al (2008), where authors base on observed RMSE on probe dataset, which we do not have available in the same way

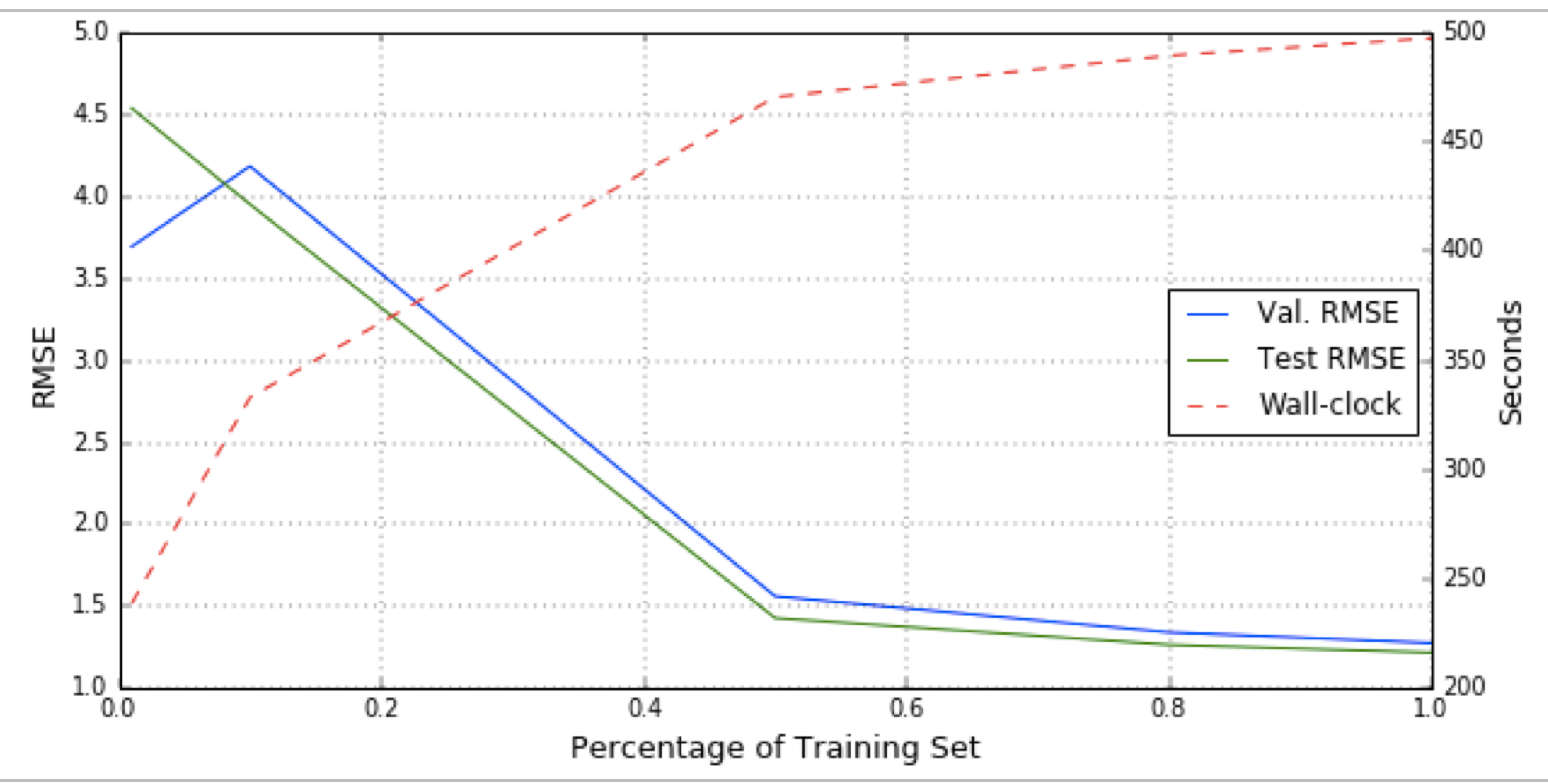
Evaluation Methodology and Experimental Results

Evaluation Methodology

- RMSE regression evaluator (as per Zhou et al (2008))
  => N.B evaluation metric computed over the non-NaN data, due to Cold-Start Strategy
- Wall-clock time, the time the CPU was used to execute the training algorithm

Experimental Results

- Document the training and test performance and the time taken for training and testing
- Compare test performance to global average baseline RMSE of 1.20



Percentage of Training Set	Maximum number of iterations	Number of latent factors	Regularization parameter	Test RMSE	Validation RMSE	Wall-clock time (s)
0.01	15	10	0.05	4.54	3.70	239.00
0.1	15	70	0.5	3.95	4.19	333.12
0.5	15	50	0.5	1.43	1.56	470.57
0.8	15	10	0.5	1.27	1.34	489.57
1	15	5	0.5	1.22	1.28	497.49

Analysis and Critical Evaluation of Results

- Wall-clock time increased with the number of training items, resembles O(log n) complexity
- Validation and test RMSE improve with the number of training items
- No indication of overfitting for downsampled training sets > 1 % , test RMSE is lower than validation set RMSE
- Global average baseline results (1.20) still better than the best ALS model fitted on the complete training set (1.22) → reject hypothesis 2.
  Potential reasons:
  - Insufficient number of parameters in parameter grid
  - Large imbalance of dataset

Lessons Learnt and Suggestions for Further Work

- Increase the size of the parameter grid
- Incorporate more niche items rated by users from more unique datasets. For example, video games rated on the Steam store – combine predictions
- Improve user ratings by performing sentiment analysis on respective review text (contained within original dataset) and attributing a score which would downgrade/upgrade/corroborate original rating (Ref?)
- Incorporating implicit feedback (Koren et al, 2009)
- Incorporating Temporal Dynamics, so as to deal with the real-world issue of constantly changing product perception (Ibid)
- Incoroporate varying confidence interval to mitigate against short-termism and adversarial users (Ibid)
- Compare with other models, such as a Neural Network implementation, e.g. Zhou et al's (2009) Restricted Botzmann Machine (RBM), or ensembled methods like that of the same authors' linear blending of ALS, with k-Nearest Neighbours and RBM

References

Anderson, C. (2004). *THE LONG TAIL.* <https://www.wired.com/2004/10/tail/> accessed 25/04/2018
Apache Spark – Documentation. (2018). *Collaborative Filtering* <https://spark.apache.org/docs/latest/ml-collaborative-filtering.html> accessed 25/04/2018
Koren, Y., Bell, R. & Volinsky, C. (2009). "Matrix Factorization Techniques for Recommender Systems". *Computer.* Volume 42 Issue 8, August 2009, pp.30-37
Leskovec, J., Rajaraman, A. & Ullman, J. D. (2014). *Mining of Massive Datasets.* Cambridge University Press
Linden, G., Smith, B. & York, J.. (2003). Amazon.com recommendations: Item-to-item collaborative filtering. IEEE Internet Computing 7, 76–80 (2003)
McCauley, J. *Amazon product data.* <http://snap.stanford.edu/data/amazon/productGraph/categoryFiles/reviews_Toys_and_Games_5.json.gz> accessed 25/04/2018
Zhou, Y., Wilkinson, D., Schreiber, R. & Pan, R. (2008). "Large-Scale Parallel Collaborative Filtering for the Netflix Prize," Proc. 4th Int'l Conf. *Algorithmic Aspects in Information and Management,* LNCS 5034, Springer, pp.337-348.