Benchmark of LFS and other Item-based Recommendation Algorithms on Public Dataset

Michal Laclavik[1] and Marek Ciglan[1]

Magnetic Media Online, www.magnetic.com, laclavik@magnetic.com

Abstract. In this paper, we propose a simple non-personalized product recommendation algorithm for e-commerce websites. Algorithm uses purchase history as a training data and outputs product-to-product similarities, which are used for product recommendations on product detail pages. Product similarities are computed based on latent product features computed by Matrix Factorization on user-product matrix or session-product matrix. The recommendations include replacement for viewed products as well as complementary products to this product. Item-based algorithms fall into category of non-personalized product recommendation algorithm, but still are widely used for recommendations on any e-commerce websites. We compare the algorithm with item-KNN, Apriori and POP (top seller) algorithms on publicly available dataset and benchmark them on several important metrics, which need to be taken into account on any e-commerce site to provide useful recommendations.

1 Introduction

Product recommendation is now days essential part of any e-commerce site. Since the Netflix prize 2009[1], personalized recommendation algorithms became popular and delivered value in many areas, where personalized context is a key driver, for example in movie recommendations. The reality of many e-commerce sites is that personalized algorithms like matrix factorization [6] does not work that well for 2 reasons, which we observed also at Magnetic:

- The first is that current session context like product(s) viewed is much more stronger than user history, although both can be combined.
- The second problem is that many e-commerce sites have high number of traffic for unknown users (cold start), and you still want to serve relevant recommendations, so it is more important to focus on current product page context or context of the session.

That is why, there is still need for non-personalized item-based recommendation algorithms, which can deliver relevant recommendations just based on product context. Such algorithms can be used on checkout or add-to-cart page, but mainly on product-detail page, which serves more than 90% of recommendations on typical e-commerce site in user session. At the end, these algorithms are usually combined by machine learning algorithms like [8], in order to derive

[1] http://netflixprize.com/index.html

appropriate context and appropriate recommendation for current user session including also personalized user context, but good quality non-personalized recommendations are important input into ML models.

Typical item-based algorithms are: shopping basket analysis algorithms like apriori [1] [3], mahout item similarity[2], KNN [9] or content based algorithms. Also simple algorithms like new arrivals, top seller in category or random in the category can serve well in a product context, where not enough of purchase or click data is available yet.

The main contributions of the paper are the following:

- We propose a new item-based algorithm, which can be trained on purchase data, and it is able to deliver recommendations in much wider context than typical basket-analysis algorithms or item-KNN.
- We implemented three basic existing algorithms: item-KNN, Apriori and POP in the form of IPyton notebook, which is available for further experiments and extension
- We evaluate the algorithms on publicly available data set[3], which makes experiments reproducible
- implementation of algorithms and evaluation is available on-line[4]

2 LFS Algorithm

Latent Features Similarity Algorithm (LFS) is based on Matrix Factorization [6], concretely its MLlib implementation[5], based on ALS for implicit feedback [5]. Similar algorithm to LFS was proposed in [7], however it combine both user and product content, while here we focus on non-personalized algorithms and their benchmark.

In LFS, we compute product latent features by factorizing user-product or session-product matrix. Then we ignore the user (or session) latent vectors and use only product vectors to compute product-to-product similarity by simple dot product on latent vectors. These results into product to product similarity, where we take top-N similar products for each product and serve them as recommendations.

For product-to-product similarity we have used dot product, but we also tried products vectors cosine similarity in the latent space, however results with simple dot product were much better in offline evaluation.

In the paper we trained algorithm only on session-product matrix since user information is not available in the dataset. Anyhow, we tested both user-product and session-product training data on 4 e-commerce websites and in all cases models trained on session-product matrix delivered same or better results.

We are using 128 latent features. In most of the experiments, more features contributed to better quality performance, but going for more than 100 features

[2] https://mahout.apache.org/users/algorithms/intro-cooccurrence-spark.html
[3] http://2015.recsyschallenge.com/challenge.html
[4] https://github.com/misos/item-based-recommendation-algorithms-eval
[5] http://spark.apache.org/docs/latest/mllib-collaborative-filtering.html

gain us just small quality improvement, when compared with the size of the model and time to train the model. So we just did arbitrary decision to use 128 features based on the application needs.

It was interesting to see, how well latent features represented the reality. For example on e-commerce site selling pets related products, it nicely recommended products of different kind grouped together for kitten, puppy or reptile although training data had no information about category. On the home accessories e-commerce site it recommended similar products, where LFS algorithm detected features like product shape, pattern, product material (metal, glass) or also other features like seasoning, where it recommended other fall related products having artifacts like leaves or pumpkin to product with falling leaves artifacts.

3 Evaluation

We evaluate following algorithms:

- proposed LFS algoritm,
- a simple popularity algorithm based on recommending top seller products,
- classical Apriori [1] algorithm and
- item-KNN [9] algorithm.

LFS is implemented as defined in section 2. For the item-KNN algorithm, we are using cosine similarity, since we have just implicit user feedback, browsed products, no ratings. For Apriori, we tuned the algorithm to return best results on the given dataset, where we need to have at least 5 relations between products and the score is higher than 10%. Same score threshold was applied for recommendations from item-KNN.

Data set: we picked publicly available dataset of RecSys Challenge 2015 [2] for evaluation in order to make experiments reproducable. We are also providing ipython notebooks with all the code of creating models and evaluation.

Evaluation protocol: In the RecSys Challenge 2015 [2], algorithms were evaluated on two tasks: predicting of purchase sessions; and predicting purchased items. In our evaluation, we focus on predicting purchased items, but in a bit slightly different way, matching the business needs and metrics of e-commerce clients. In real e-commerce website, you always want to fill recommendations zones with useful recommendations. So if algorithm is not able to recommend anything or not confident enough, you still want to recommend at least top sellers in category, new arrivals or promoted products. This is why, we are also evaluating percentage of served recommendations and server sessions by each algorithm. We evaluated all algorithms with 1, 3, 5 and 10 recommendations per page. In the paper, we provide just results for 5 recommendations per page, because other numbers were directionally very similar.

We have used 90 days of data to train the models and next 14 days to evaluate the model. So we have generated 5 recommendations per viewed product page for browsing history of next 14 days after training period, and then compared recommended (session, product) pairs with purchased (session, product) pairs in

same period. This is obviously a bit simplified evaluation protocol, but evaluating and training data are clearly independent also in the terms of out of time sample.

In Table 1, we can see the results of evaluation. As already mentioned we have compared LFS with other 3 standard algorithms: TS (Top Sellers), Apriori and Item-KNN. TS was able to serve always, but with poor results. It could be a bit better if serving Top Seller in category instead of overall top seller products. Apriori and KNN had lower session conversion rate than LFS, but on the other hand higher recommendation conversion rate (recommended purchases/all recommendations). Still Apriori was able to serve only 9% of all requested recommendations and KNN 38% while LFS could serve 96% of all requested recommendations (5 per visited product detail page). In addition with 5 recommendation on page, LFS was able to recommend 49% of all purchases.

Table 1. Evaluation results.

Algo	CR (sessions)	CR (rec)	served rec	served sessions	rec purchases
LFS	6.70%	0.38%	96%	98%	49%
KNN	4.76%	0.54%	38%	77%	27%
Apriori	2.01%	0.61%	9%	50%	8%
TS	0.18%	0.01%	100%	100%	1%

As we can see, each of evaluated algorithm has advantage at least in one of the used metrics. TS can always fill in requested recommendation slots. KNN can identify decent portion of future purchases as well as it has higher conversion rate per recomendation than LFS. Classical Apriori algorithm on the other hand can serve best recommendation (highest purchase rate per provided recommendations) but only in limited context, but LFS combined most of good properties with winning in conversion rate per session and in recomending most of purchased products.

We have also used LFS similarity score as a feature for Machine Learning model similar to [8], which combines various algorithms and reorder final recommendations to be served. The similarity score between displayed product and to-be-served-recommended-product computed by LFS was one of top 3 features for machine learning model based on Gradient Boosting Decision Tree [4]

Scalability: To conclude the evaluation, we should also mention the scalability of the solution. LFS is trained on purchased history on ApacheSpark MLlib implementation of ALS [5] which is scalable. It scales well on Apache Spark. All the steps of algorithm can be scaled and computed in Apache Spark cluster.

4 Conclusion

Item-based recommendation algorithms provide basic non-personalized product recommendations on e-commerce sites. Nowadays more advanced machine learning algorithms exist, which can be applied and tuned well in many recommendation contexts, but item-based non-personalized algorithms are still valuable input to build better models for typical e-commerce site.

In this paper we implemented and evaluated simple LFS algorithm, which allows to recommend replacement or complementary products for product detail pages of e-commerce sites. Its advantage is that it can serve recommendations to almost all users with good quality of relevant recommendations. The model is trained on purchased data, so the only requirement is to have a purchased history of the product for which complementary or replacement recommendations are served. The LFS performs better than other state-of-the-art item-based algorithms like item-KNN or Apriori.

In this paper we also implemented and evaluated three basic item-based algorithms on publicly available dataset. We hope that provided implementation and code can serve as a benchmark for further evaluation of additional item-based recommendation algorithms

References

1. R. Agrawal, R. Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499, 1994.
2. D. Ben-Shimon, A. Tsikinovsky, M. Friedmann, B. Shapira, L. Rokach, and J. Hoerle. Recsys challenge 2015 and the yoochoose dataset. In *Proceedings of the 9th ACM Conference on Recommender Systems*, RecSys '15, pages 357–358, New York, NY, USA, 2015. ACM.
3. S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *ACM SIGMOD Record*, volume 26, pages 255–264. ACM, 1997.
4. J. H. Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378, 2002.
5. Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*, pages 263–272. IEEE, 2008.
6. Y. Koren, R. Bell, C. Volinsky, et al. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.
7. M. Kula. Metadata embeddings for user and item cold-start recommendations. In *2nd Workshop on New Trends on Content-Based Recommender Systems co-located with RecSys 2015, Vienna, Austria*, volume 1448 of *CEUR Workshop Proceedings*, pages 14–21. CEUR-WS.org, 2015.
8. P. Romov and E. Sokolov. Recsys challenge 2015: Ensemble learning with categorical features. In *Proceedings of the 2015 International ACM Recommender Systems Challenge*, RecSys '15 Challenge, pages 1:1–1:4, New York, NY, USA, 2015. ACM.
9. B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, pages 285–295, New York, NY, USA, 2001. ACM.