



Personalized Recommender for Implicit Data

Walmart Internship Report

Zehao Guan, 2019/08/01

Agenda

- Introduction
- Project
- Future Work
- Conclusion

Agenda

- Introduction
- Project
- Future Work
- Conclusion

Abstract

- Target

Design a personalized recommendation system for implicit feedback data.

- Methods

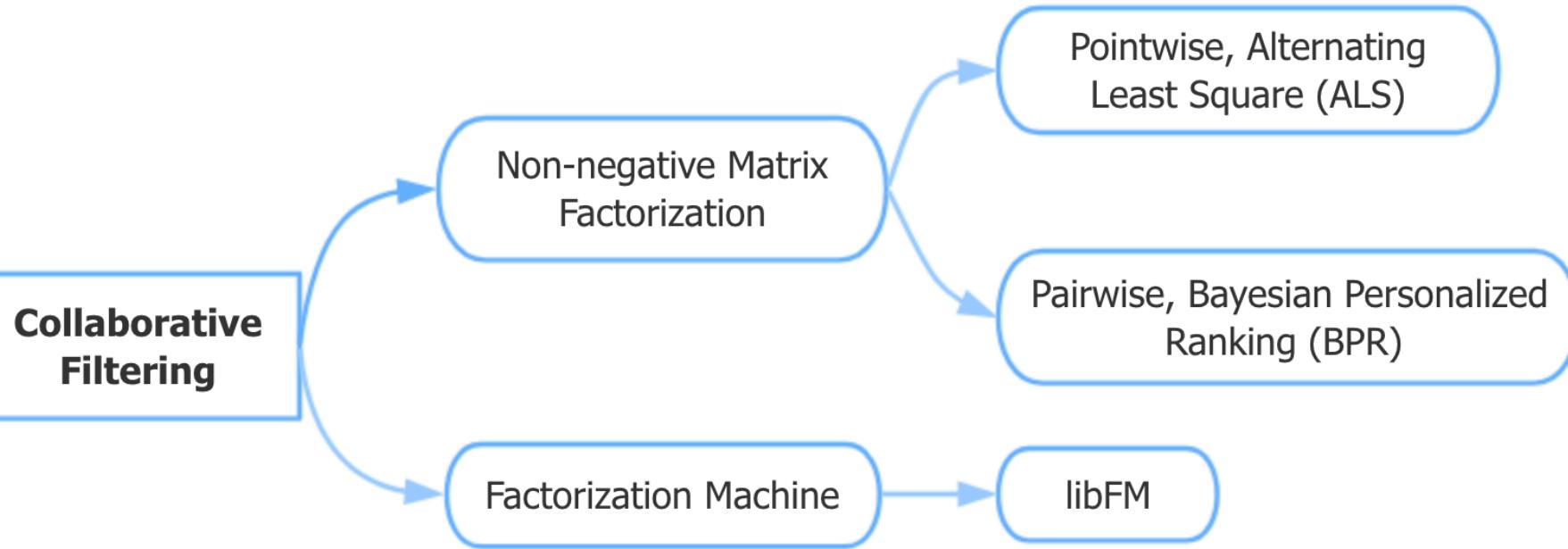
Latent factor model based **collaborative filtering**, like pointwise/pairwise matrix factorization and factorization machine with libFM.

- Experiment

Split into train/validation data in temporal manner;

Apply ranking based metrics for model evaluation.

Sketch



Agenda

- Introduction
- Project
- Future Work
- Conclusion

Background

- Unlike extensively researched explicit feedback, no direct input from the users regarding their preferences.
- Treat transaction data as indication of positive and negative preference associated with varying **confidence levels**.
- Should design a **scalable** optimization procedure and explainable factor model for recommendations.
- The target is to realize **top-N** personalized recommendation based on users' response.

Algorithm

- Pointwise NMF, Alternating Least Square (ALS)

$$\min_{x_\star, y_\star} \sum_{u,i} c_{ui} (p_{ui} - x_u^T y_i)^2 + \lambda \left(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right)$$

- Pairwise NMF, Bayesian Personalized Ranking (BPR)

$$\sum_{(u,i,j) \in D_S} \ln \sigma(\hat{x}_{uij}) - \lambda_\Theta \|\Theta\|^2$$

- Factorization Machines with libFM

$$\hat{y}(\mathbf{x}) := w_0 + \sum_{j=1}^p w_j x_j + \sum_{j=1}^p \sum_{j'=j+1}^p x_j x_{j'} \sum_{f=1}^k v_{j,f} v_{j',f}$$

Experiment Design

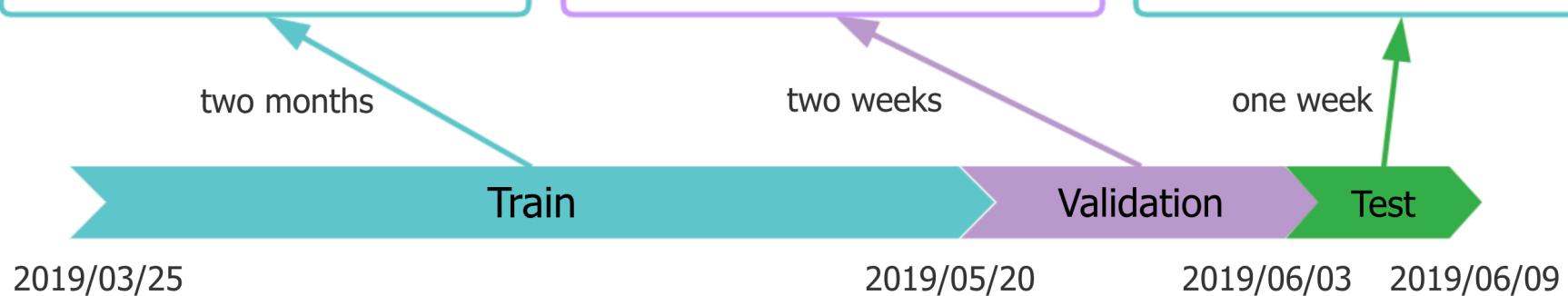
- Write models from scratch which is more flexible for metrics evaluation and prediction.
- To capture sequential users' purchase behavior, split into train/validation data in **temporal** manner.
- As items ranked higher should be more preferred by users, **ranking** based metrics are applied.
- Start with demos on small-scale data, and then improve the scalability.

Temporal Manner

Span: 2019/03/25 - 2019/05/19
Interactions: 47938318
Size: 5.1G

Span: 2019/05/20 - 2019/06/02
Interactions: 11546420
Size: 1.3G

Span: 2019/06/03 - 2019/06/09
Interactions: 6149672
Size: 667M



Ranking Metrics

- Hit ratio (HR)
- Normalized discounted cumulative gain (NDCG)
- Mean average precision (MAP)
- Area under curve (AUC)

Top-N recommendation !

Data

- Database: ucid
- Table: common_transaction_model
- Source: Walmart.com
- Columns: cid, catalog_item_id, purchase_date
- Features:
 - Implicit feedback
 - High **sparsity**
 - Imbalance

Preprocess

- Clean outliers like infrequent customers/items and resellers.
- Map user and item ids to categorical values.
- Store observed data in <userId, itemId, rating> triple format.
- Convert to **sparse matrix** for other operations.

Sample data in demo:

Ignore: 14847105 * 2958922

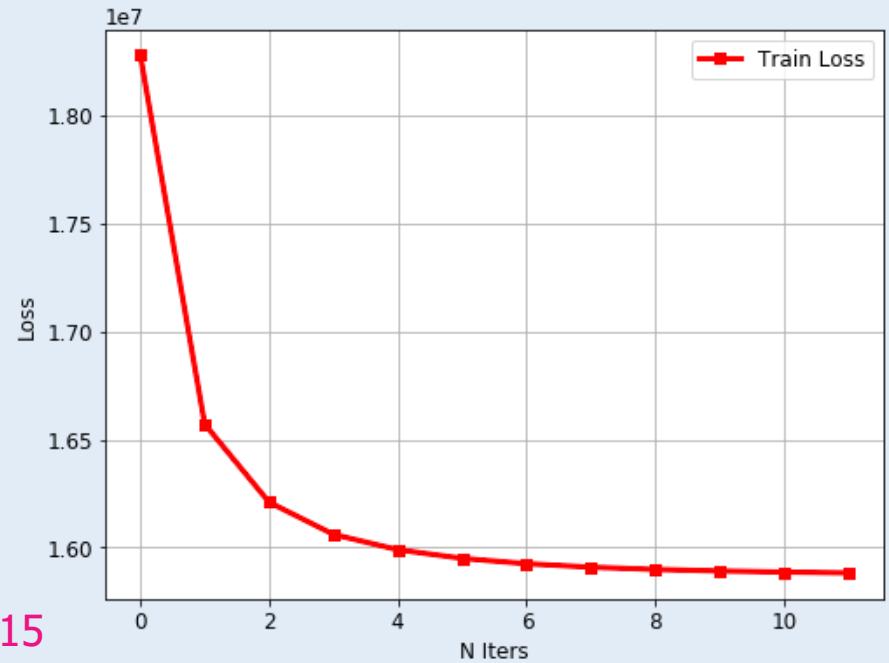
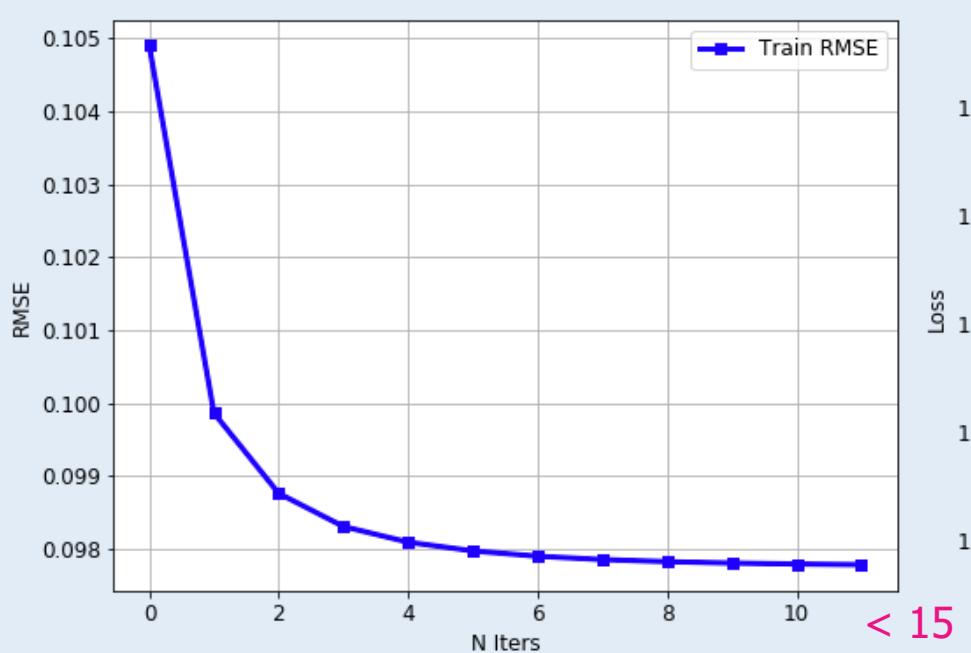
Raw: 14929995 * 3008950



Processed: 34209 * 47857

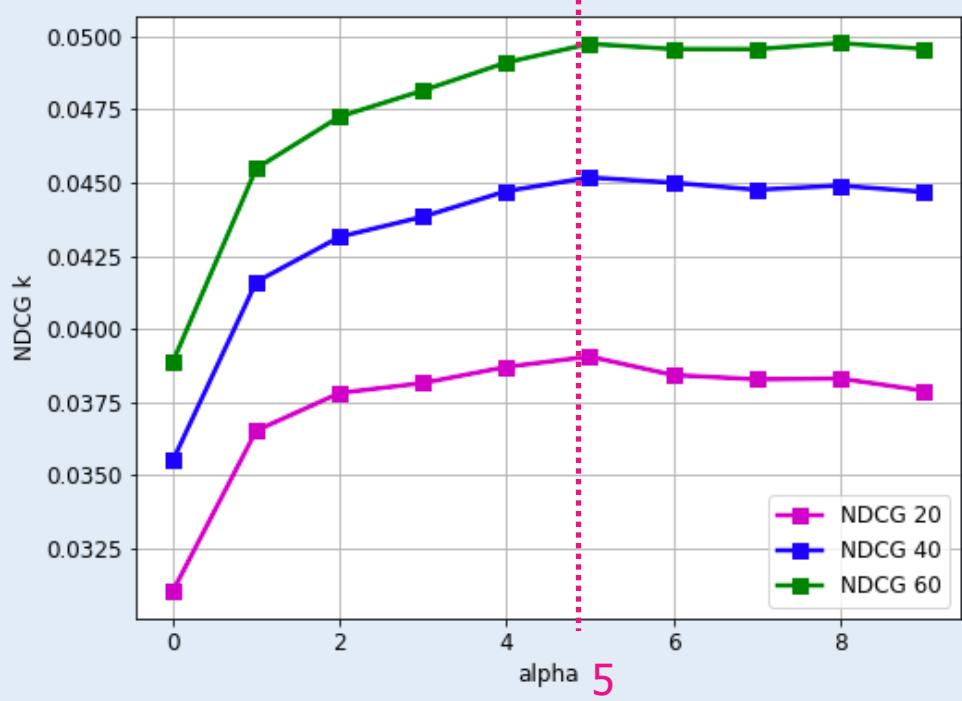
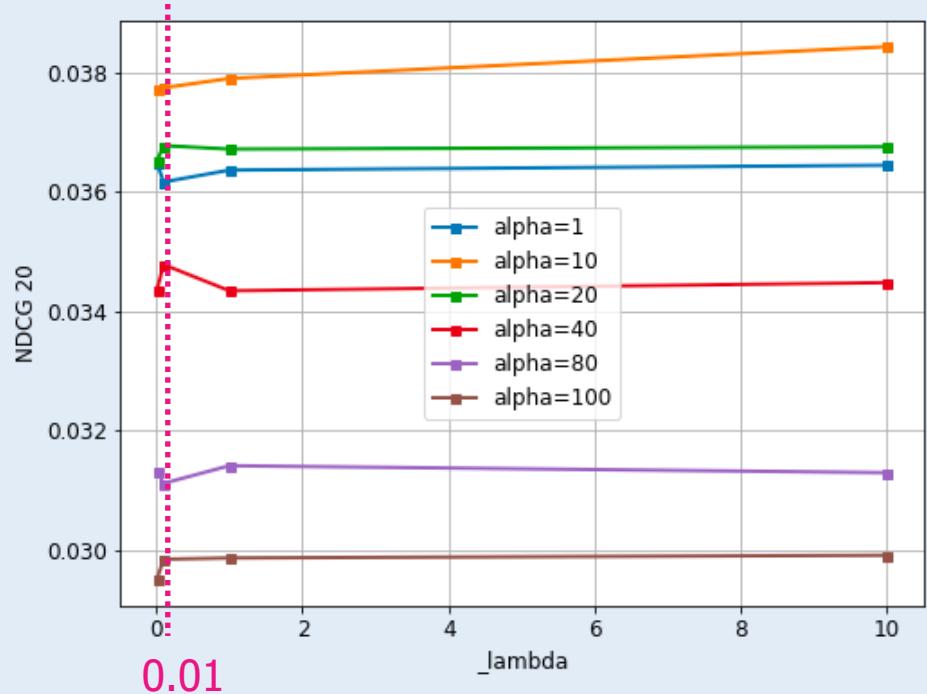
density: 0.0712%

Train



Assume n_iters that is enough for the loss to converge.

Evaluation

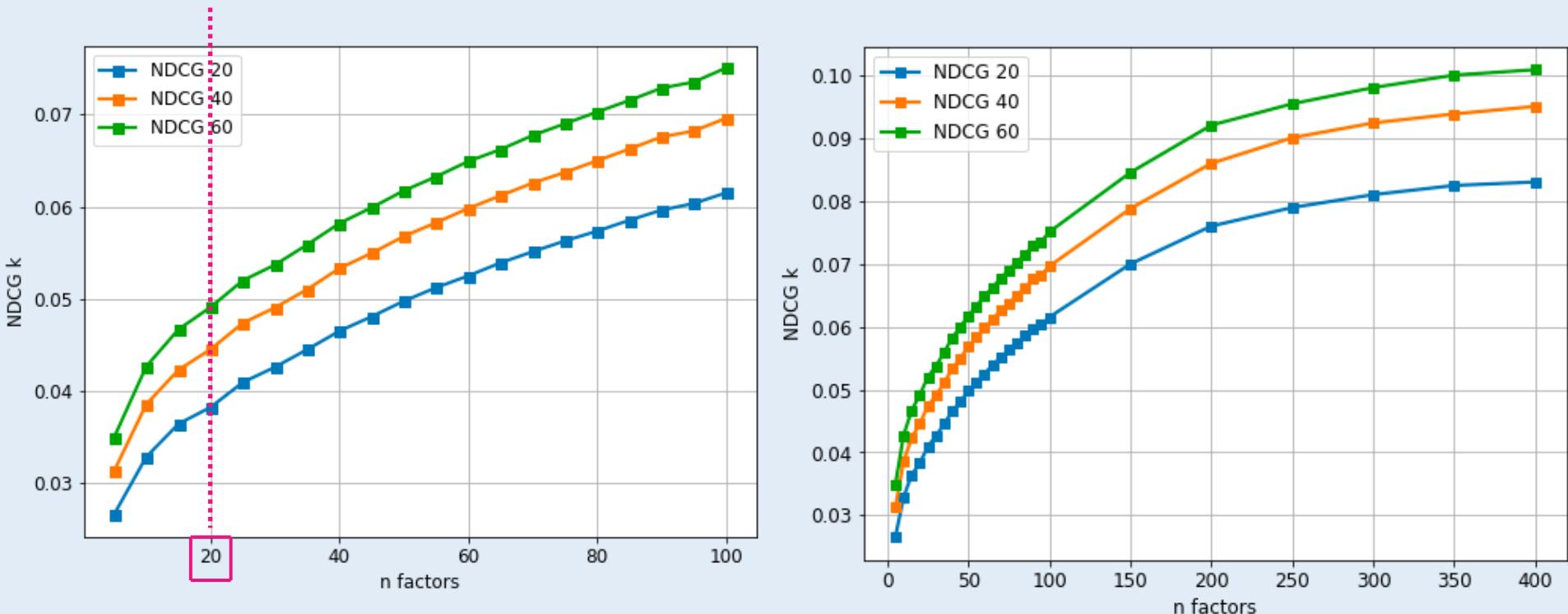


Tune on parameters like $_{\lambda}$ and α .

$$c_{ui} = 1 + \alpha r_{ui}$$



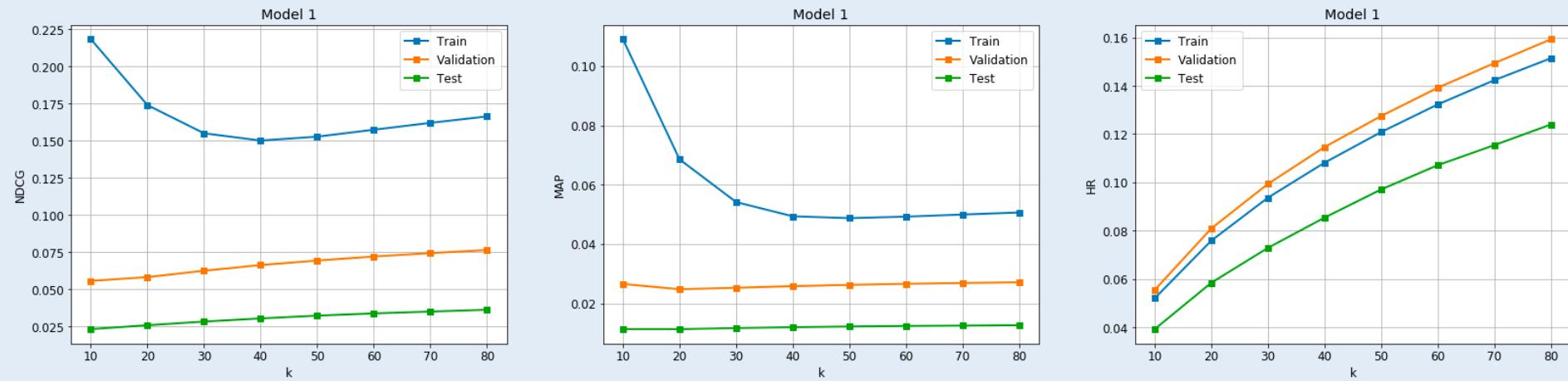
Evaluation



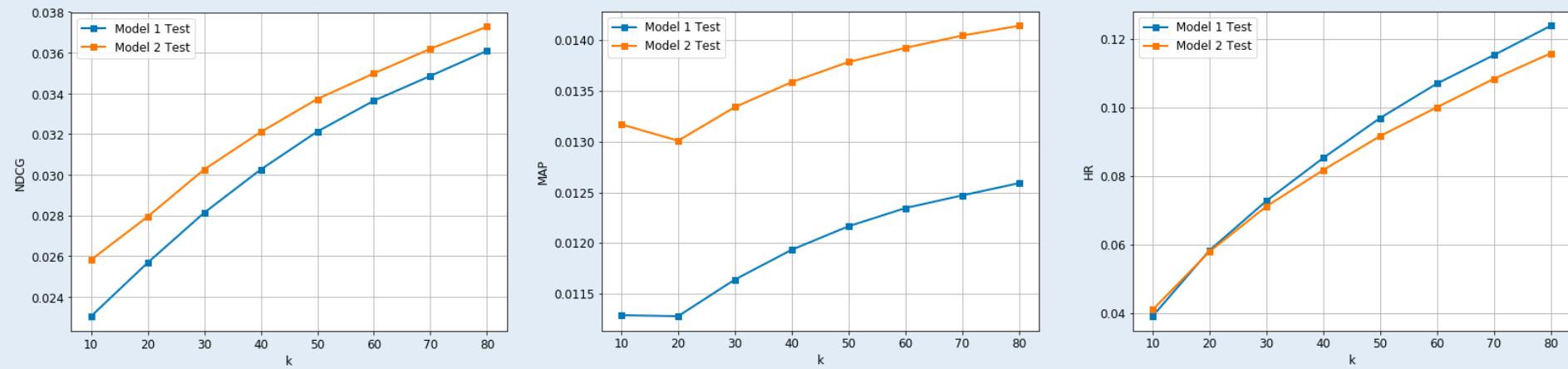
Extend scale to 0~400, and select best n_{factors} for test, also consider computation cost and avoid over-fitting.

Evaluation – Model selection

Top-k



0/1 v.s. normalized frequency



Test – Recommend for users

```
1 n_samples = 5
2 sampled_users = np.random.choice(list(train_user_map.keys()), n_samples)
3 sampled_users

array(['ad082399f3c546ccb21c22ed549ca908',
       'f05077c5608c0cf0ee044001517f43a86',
       '7b581a751de24fd48020fe151edd3f2b',
       'c17b06ffbbfd438da3ba75e618a74eff',
       '6c80ba0e97e44015abce52d19453b1f1'], dtype='<U32')
```

cid

```
1 sampled_user_ids = list(map(lambda x:train_user_map.get(x), sampled_users))
2 sampled_user_ids

[27025, 27943, 10784, 19260, 34192]
```

user_id

```
1 best_model._recommend_user(6753, N=6) # 'f0760cb6eeb034cce044001517f43a86
[305, 594, 386, 1062, 992, 436]
```

```
1 best_model.recommend(sampled_user_ids, N=6) # same as the previous test

array([[ 137,  1567,    714,  1467,    226,    693],
       [ 309,    992,   215,    305,  2026,    273],
       [ 535,  2051, 17703,  2415,    748,  1704],
       [ 2649,   215,    726,  2026,  4622,  3738],
       [ 4254,  1455,  9539,  6690,  1983,  1660]])
```

item_id

https://gecgithub01.walmart.com/z0g00mx/Intern_project/blob/master/als.ipynb

Test – Find similar items

```
1 sampled_items = np.random.choice(list(train_item_map.keys()), n_samples)
2 sampled_items
array(['16879847', '653330322', '10322221', '24797654', '53834652'],
      dtype='<U10') catalog_item_id
```



```
1 sampled_item_ids = list(map(lambda x:train_item_map.get(x), sampled_items))
2 sampled_item_ids
[11940, 17210, 22782, 21439, 46809]
```



```
1 best_model._similar_items(sampled_item_ids, N=5)
array([[14287, 21535, 5768, 1448, 9261],
       [20512, 12668, 249, 7587, 4939],
       [42478, 47871, 33527, 22274, 44058],
       [30214, 24232, 35489, 5927, 17440],
       [19244, 34813, 5583, 40392, 26451]]) item_id
```

https://gecgithub01.walmart.com/z0g00mx/Intern_project/blob/master/als.ipynb

Test – Case study



10307700.jpeg



14018038.jpeg



14089586.jpeg



14150016.jpeg



15783126.jpeg



19635718.jpeg



20606796.jpeg



23619931.jpeg



24360609.jpeg



25065575.jpeg



27231189.jpeg



33963183.jpeg



14504328.jpeg



23636896.jpeg



49509899.jpeg



35756520.jpeg



38453308.jpeg



45612361.jpeg



49333136.jpeg



49527355.jpeg



52459941.jpeg



52196241.jpeg



153202890.jpeg



401199665.jpeg



55630914.jpeg



127471323.jpeg



127722938.jpeg



151835792.jpeg



162433676.jpeg



165253444.jpeg



54048...44.jpeg



603122123.jpeg



327970267.jpeg



331157650.jpeg



610627633.jpeg



723575351.jpeg



762509212.jpeg



835076821.jpeg

Top-8 recommended items

User purchase history

Test – Case study



Similar
→



Sampled item



10824297.jpeg



46291110.jpeg



55139724.jpeg



278392073.jpeg

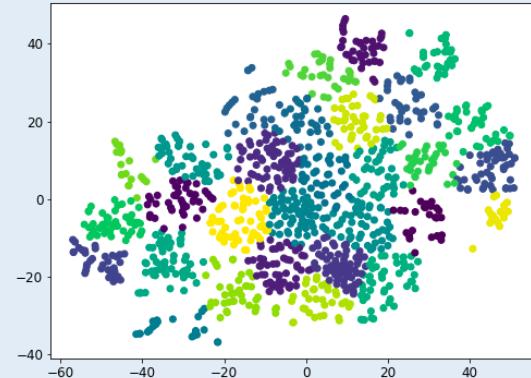
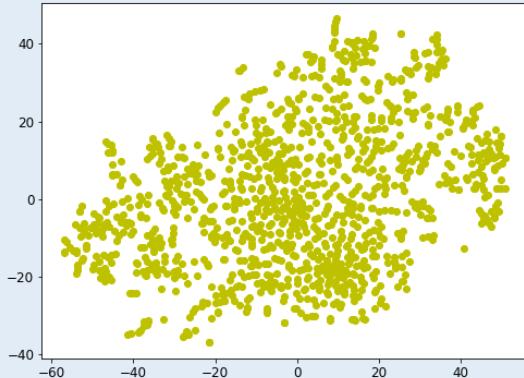


929828867.jpeg

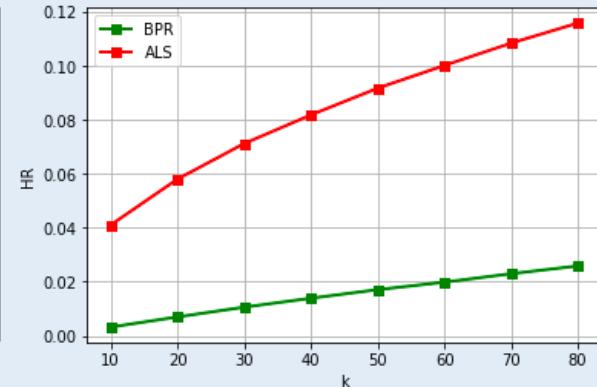
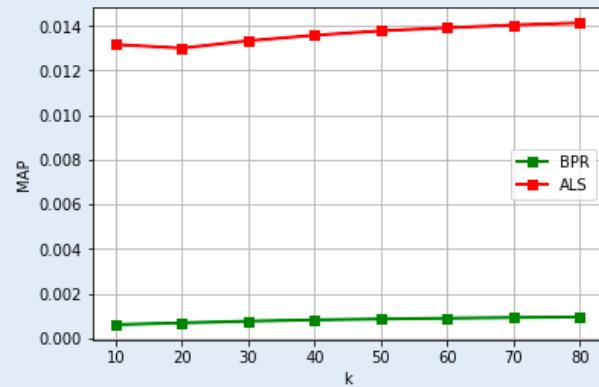
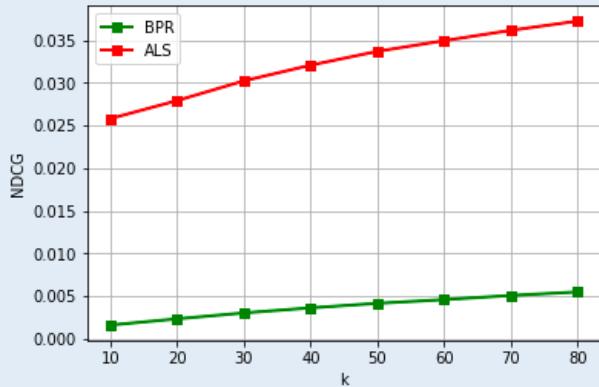
Top-5 similar items
(KNN, Euclidean metric)

Analysis

- Separability of latent factors with t-SNE, K-means



- Pointwise method performs **better** than BPR



Accomplishment

- Three recommender **demos** in same experimental procedures with Walmart.com transaction data.
- Code implementation for **large-scale** data. (1856520 * 831859)
 - Write bash scripts to use libFM by C++ for matrix factorization to get **latent factors**.
 - Python codes for preprocessing, ranking metrics evaluation and personalized recommendation.

https://gecgithub01.walmart.com/z0g00mx/Intern_project/tree/master/CF_libFM

Agenda

- Introduction
- Project
- Future Work
- Conclusion

Future Work

- Realize industrial level APIs of the recommender demos.
- Evaluate the model with different sources of data.
- Besides these **baseline models**, explore more advanced methods for matrix factorization and optimization.

Agenda

- Introduction
- Project
- Future Work
- Conclusion

Challenge

- Inexperience with recommendation system for implicit feedback dataset.
- Lots of efforts to process highly sparse and unbalanced transaction data.
- Difficulty to apply PySpark for evaluating ranking metrics.
- Problems in face of scalability.

What I've learnt

- Tools like Hive, Spark, Kubernetes, libFM...
- More familiar with libraries like pandas, scipy, sklearn...
- Knowledge about the field of **implicit recommendation**, like classic models and widely-used ranking methods.
- Experience with experiments design and conduct.
- Capability to **learn by doing**.

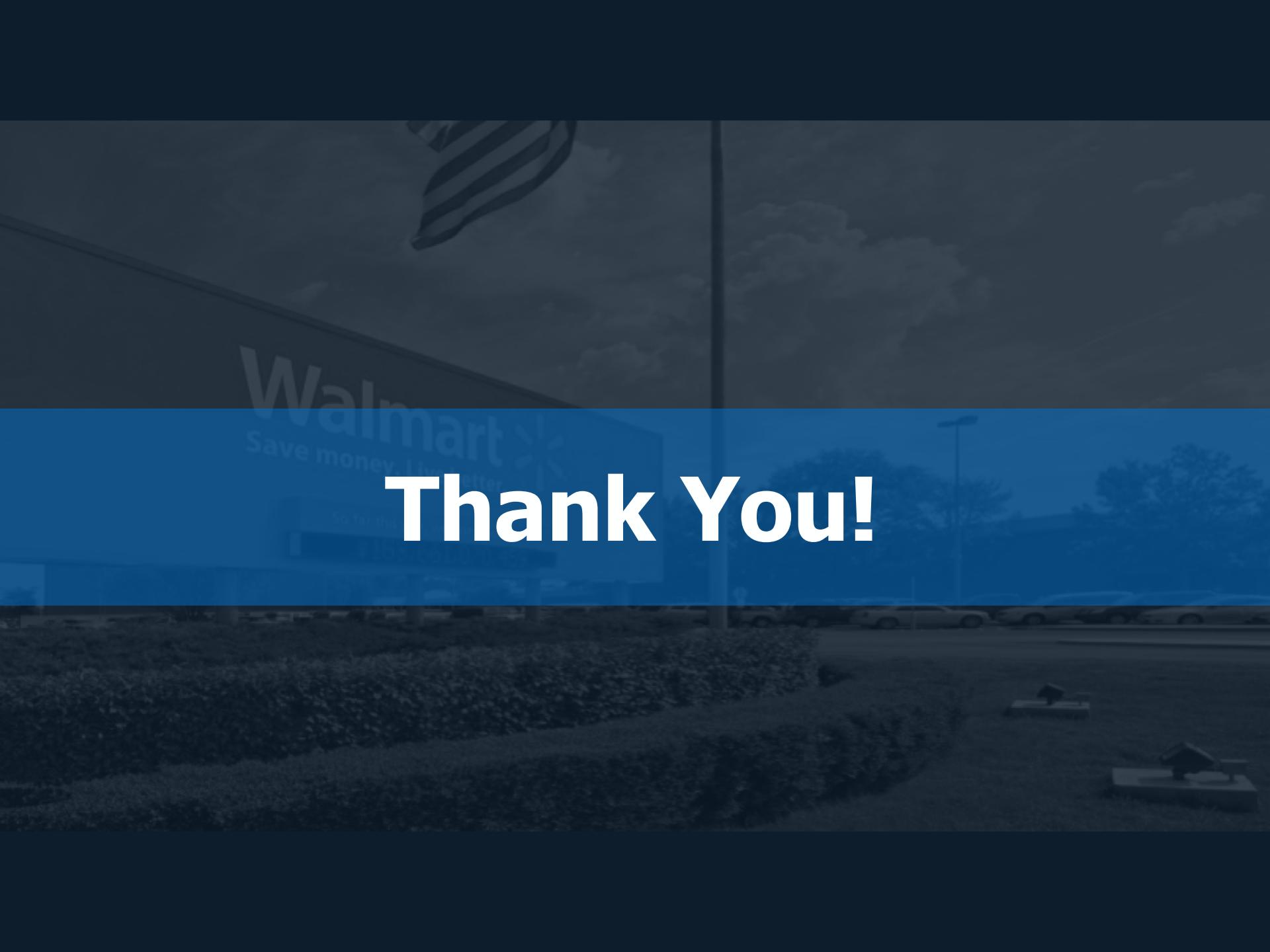
Reference

1. Hu, Yifan, Yehuda Koren, and Chris Volinsky. "Collaborative filtering for implicit feedback datasets." *2008 Eighth IEEE International Conference on Data Mining*. Ieee, 2008.
2. Koren, Yehuda, Robert Bell, and Chris Volinsky. "Matrix factorization techniques for recommender systems." *Computer* 8 (2009): 30-37.
3. Rendle, Steffen, et al. "BPR: Bayesian personalized ranking from implicit feedback." *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*. AUAI Press, 2009.
4. Rendle, Steffen. "Factorization machines with libfm." *ACM Transactions on Intelligent Systems and Technology* (TIST)3.3 (2012): 57.

Appendix

- GitHub link:

https://gecgithub01.walmart.com/z0g00mx/Intern_project



Walmart
Save money. Live better.
So far this year.

Thank You!