# Food $\chi$*:
# Building a Recommendation System for Chinese Dishes

CS229 Group Project Final Report (Category: Natural Language Processing )

Yu Zeng, Yiting Ji, Yogi Huang

December 13, 2018

## 1 Introduction

As applications and websites develop the trend to provide customized service for users, building recommendation systems has gain more popularity and attention from businesses to improve user experience. While a great number of recommendation systems for movies, audios, books, or restaurants exist, surprisingly there is hardly any for dishes. As food lovers, we intend to address this issue by exploring models for recommending Chinese dishes.

We scrape data from a popular Chinese recipe website/app called Xia Chu Fang Cooking Recipe[1], and implement word embedding algorithms and recommendation system algorithms to build our model. The original input to our algorithms are user IDs from Xia Chu Fang and dish names users have saved in their favourite list. We first preprocess our data and use word2vec on them. We then generate ratings and apply collaborative filtering to build our recommendation system. Specifically, we explore the skip-gram model in word2vec to calculate dish similarity, as well as apply the Non-Negative Matrix Factorization and Singular Value Decomposition methods in collaborative filtering to predict ratings on dishes. Limiting our dishes to Chinese cuisine allows us to focus on constructing a more tailored recommendation system, while still maintain a high practical value given the popularity and diversity of Chinese cuisines. We hope by doing this project, we can tackle a less touched topic of using Machine Learning for dish recommendation, and at the same time promote the greatness of Chinese food.

## 2 Related Work

Multiple methods for building recommendation systems are discussed in literature. In the natural language processing domain, word2vec is a set of methods in word embeddings that produces promising results in recommendation systems [1][2]. It takes a text corpus as input to a shallow neural network and learn vector representations of words [3][4] [5]. It could directly be used to extract similarities between text, but can be hard to interpret sometimes. Another commonly used method for product recommendations is collaborative filtering (CF) [6]. User-based CF measures similarity between users based upon their opinions on items, and makes recommendations using similar users' highly rated items [7]. Limitations to this method include its requirement of relatively large computational power and overlooking the fact that people's habits may change over time. Approaches that may improve user-based CF include Matrix-factorization (MF) methods, which prove to be highly accurate and scalable in addressing CF problems [8]. The Non-Negative Matrix Factorization (NMF) algorithm is very suitable for solving CF problems with non-negative constraint [8]. Singular Value Decomposition (SVD) is another MF algorithm applied to CF that reduces a matrix to two matrices with lower dimensions that represent generalized items and users. This generalization can capture vast amount of data more intuitively. And the lower ranked matrices often yield better estimates of the data than the original matrix [9].

## 3 Dataset and Features

### 3.1 Data

Xia Chu Fang is a platform where users can publicly post recipes of dishes. They can also search for recipes from other users and put them in their favourite, aka, "starred" list if they are interested in learning them. As one of the leading recipe websites in China, Xia Chu Fang has over 1.8 million published recipes and 100 million users, with at least 1.2 million registered users [10]. Scraping all data from the website is too computationally expensive and time-consuming, so we randomly scrape a portion of the user ids and their starred dishes. We utilize parallel crawling and proxies to

*Fun fact: the Greek letter $\chi$ is romanized as Chi, which by pronunciation means "to eat" in Chinese. It thus made into our title.

[1] https://www.xiachufang.com/.

fetch data more efficiently, and run a spider on Google Cloud which creates two virtual machines (each containing 8 vCPUs). This way, we manage to scrape roughly 230,000 valid users and more than 2.5 million dishes in their starred list (roughly 12,000 unique dishes).

## 3.2 Data Preprocessing

### 3.2.1 Dish Name Mapping

The number of total dishes is an overestimate of the true number of dishes, because a lot of them are duplicates with slight variations[2], resulting in sparseness of the data. Cleaning up dish names is quite challenging given Chinese dish names' different encoding and the complexity of the language itself. After trials and errors[3], we decide to utilize an online database of Chinese dish names with their English translations [11] as a dictionary, which has 1,871 unique keys representing recipe names. Using Jaro-Winkler distance[4], we map our dishes to the dictionary and reduce our data to $198,816$ unique users and $1,628$ unique dishes.

### 3.2.2 Ratings Calculation

Unlike most of the other datasets for building recommendation systems, our dataset does not have users' ratings on dishes. As a result, we need to generate ratings in order to implement NMF and SVD. We notice from the mapped data that many users put different versions of dishes in their starred list. Marking a dish multiple times implies that the user is really into this dish such that he or she is willing to save multiple recipes for it to try out different cooking methods. This indicates that the user is more interested in this dish than a dish that is only saved once. We therefore utilize this property and define a user's rating on a dish as:

$$R_k^{(i)} = \text{count of dish } i \text{ in user } k\text{'s starred list}$$

This way, we generate ratings ranging from 1 to 18. We then normalize them by casting them to a number between 5 and 10 to minimize the effect of outliers and seperate it from the traditional rating scheme[5]. Figure 1 below is a sample of 10 user ratings on dishes. Figure 2 shows a histogram of all calculated ratings[6]. We can see that a majority of ratings are 5, with a small portion between 6 and 10.

| | user_id | recipe_name | rating |
|---|---|---|---|
| **246195** | 19775 | 484 | 5 |
| **2127989** | 166145 | 1579 | 5 |
| **1786193** | 139774 | 211 | 5 |
| **1551874** | 119794 | 1534 | 5 |
| **1195275** | 92602 | 502 | 5 |
| **184341** | 14998 | 1358 | 5 |
| **2163147** | 168693 | 150 | 5 |
| **149293** | 12219 | 355 | 5 |
| **1116379** | 86287 | 1202 | 5 |
| **982108** | 76384 | 2003 | 5 |

Figure 1: A Sample of Users' Ratings

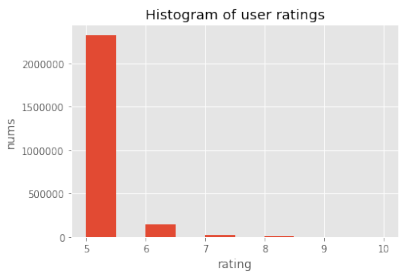


Figure 2: Histogram of User Ratings

# 4 Methods

## 4.1 Calculating Similarity Using Word2vec

We first try word embeddings on the original data without ratings. Among the various models in word2vec, we primarily focus on the skip-gram Neural Network Model [13]. This model takes one word as input, trains a neural network with a single hidden layer to perform, and outputs words most likely to appear in the "context" of the input word[7]. The default skip-gram model loss function to optimize each training iteration is defined as [13]:

[2]These are variations such as descriptive adjectives, unnecessary special characters, alternative name expressions, and typos.

[3]We have tried text segmentation techniques to remove wordiness in dish names, but did not get satisfying results.

[4]The Jaro-Winkler distance is a string metric measuring an edit distance between two sequences. We map our dishes to the dishes in the dictionary when their Jaro-Similarity is higher than a threshold of 0.5. See [12] at https://ai.tencent.com/ailab/nlp/embedding.html for more details on the topic.

[5]Note that unlike the traditional rating scheme where a low rating implies that the user dislikes the item, a low rating in our rating system means the user likes the dish but simply not as much as the more highly rated ones. In other words, our rating system is a metric that shows the degree of fondness of a user on dishes they *like*.

[6]recipe_name = dish name in Figure 1, encoded to numeric values for simplification.

[7]The skip-gram model typically represents the input word as a one-hot-vector with 300 features (represented as a $1 \times 300$ vector). It has no activation function, and uses softmax in the output layer. See [14] for more information.

$$
\begin{aligned}
E &= -\log p(w_{o,1}, w_{o,2}, ..., w_{o,c}|w) \\
&= -\log \prod_{c=1}^{C} \frac{\exp\left(u_{c,j_c^*}\right)}{\sum_{j'=1}^{V} exp(u_{j'})} \\
&= -\sum_{c=1}^{C} u_{j_c^*} + C \cdot \log \sum_{j'=1}^{V} \exp\left(u_{j'}\right),
\end{aligned}
$$

Heavy notations aside[8], this merely calculates the probability of the output words being in the context of the given input word. The model also returns a a similarity score for each output word. The format for our data under this model looks like [user $i$, sentence], where the sentence consists of strings of [dish 1, dish 2, ..., dish $N_i$], i.e., all $N$ dishes in user $i$'s starred list.

## 4.2 Collaborative Filtering (CF)

We then explore CF on our user-dish dataset with our calculated ratings and compare predicted ratings generated by the NMF and SVD approaches.

### 4.2.1 Non-negative Matrix Factorization (NMF)

To perform NMF, we implement the Python library "surprise"[9]. The predicted rating that user $k$ has on dish $i$ is

$$\hat{R}_k^{(i)} = q_i^T p_k,$$

where user factors and dish factors, $p_k$ and $q_i$, are kept positive. Regularized stochastic gradient descent is used with the following update rule for the factors $f$ of user $k$ and dish $i$ at each step:

$$p_{kf} \leftarrow p_{kf} \cdot \frac{\sum_{i \in I_k} q_{if} \cdot R_k^{(i)}}{\sum_{i \in I_k} q_{if} \cdot \hat{R}_k^{(i)} + \lambda_k |I_k| p_{kf}},$$

$$\mathrm{q}_{if} \leftarrow q_{if} \cdot \frac{\sum_{k \in K_i} p_{kf} \cdot R_k^{(i)}}{\sum_{k \in K_i} p_{kf} \cdot \hat{R}_k^{(i)} + \lambda_i |K_i| q_{if}},$$

where $\lambda_k$ and $\lambda_i$ are regularization parameters for user and dish (both set to a default of 0.06) [16].

### 4.2.2 Singular Value Decomposition (SVD)

Similarly, we exploit the surprise library [15] to implement SVD[10]. The predicted rating is defined as

$$\hat{R}_k^{(i)} = \mu + b_k + b_i + q_i^T p_k.$$

If user k is unknown, then the bias $b_k$ and the factors $p_k$ are assumed to be zero. The same applies for dish i with $b_i$ and $q_i$. To estimate all the unknowns, we minimize the following regularized squared error:

$$\sum_{R_k^{(i)} \in R_{train}} (R_k^{(i)} - \hat{R}_k^{(i)}) + \lambda(b_i^2 + b_k^2 + \parallel q_i \parallel^2 + \parallel p_k \parallel^2).$$

And we perform the minimization using stochastic gradient descent with the following update rules:

$$b_k \leftarrow b_k + \gamma((e_k^{(i)}) - \lambda b_k),\ b_i \leftarrow b_i + \gamma((e_k^{(i)}) - \lambda b_i),$$
$$\mathrm{p}_k \leftarrow p_k + \gamma((e_k^{(i)}) \cdot q_i - \lambda p_k),\ q_i \leftarrow q_i + \gamma((e_k^{(i)}) \cdot p_k - \lambda q_i),$$

where $e_k^{(i)}$ is the difference between the predicted rating and actual rating that user $k$ has on dish $i$. We use the default learning rate $\gamma = 0.005$ and a regularization factor $\lambda = 0.02$ (see [16] for more details).

[8] In this equation, $w_I$ is the input word, and $w_{o,c}$ is the $c^{th}$ word in $w_I$'s predicted context of length $C$, $u_{c,j}$ is the input to the $j^{th}$ node of the $c^{th}$ output word, and $j_c^*$ is the index of the $c^{th}$ output word.

[9] "Surprise" is a Python SciPy toolkit for building and analyzing recommender systems. It provides ready-to-use matrix factorization-based algorithms such as NMF and SVD. See [15] for more information and documentation on this library.

[10] Theoretically, let $A$ be a real $m \times n$ matrix with $m \geq n$. Then we have $A = U\Sigma V^T$, where $U^T U = V^T V = VV^T = I_n$ and $\Sigma = diag(\sigma_1, \sigma, ..., \sigma_n)$. The matrix $U$ consists of $n$ orthonormalized eigenvectors associated with the n largest eigenvalues of $AA^T$, and $V$ consists of the orthonormalized eigenvectors of $A^T A$. The diagonal elements of $\Sigma$ are the non-negative square roots of the eigenvalues of $A^T A$, and are called singular values [17].

# 5 Experiments and Results

Given our data size, we split our dataset into 80% training set, 10% development set, and 10% test set. This split results in 159,053 samples in the training set, 19,881 samples in the dev set and 19,882 samples in the test set. We train our models on the training set, and calculate errors on the dev set for model comparison, and obtain errors on the test set to examine model robustness.

## 5.1 Word2vec Results

The skip-gram model from word2vec calculates similarity scores of dishes, which allows us to directly output the context words with highest similarity scores as our recommendations of the input dish (see Figure 3 for two examples). Note that by looking at the recommendations, while we can tell some of the dishes with similar main ingredients are likely to show up, it is still not entirely straightforward to interpret why these dishes are similar, because the model learns the word context based on the data of thousands of users and calculates similarity in a hidden layer. It is therefore hard to quantify errors. Instead, we provide a visualization of the word2vec results by generating t-SNE graphs[11] to represent the similarity of a sample of dishes. See Figure 4. The distance between two dishes in the graph represent their dissimilarity[12].

**Input:** Sweet and sour fish
**Recommendations (with similarity scores):**
1. Fried Boiled Pork with Sea Cucumber: 0.948
2. Braised Fish Filets in Taiwan Black Bean Sauce: 0.946
3. Boiled Seafood with Bamboo Fungus: 0.944
4. Pan-Fried Flatfish: 0.944
5. Braised Intestines in Brown Sauce: 0.941
6. Perch with Mushrooms: 0.940
7. Fresh Squid in Hot Chili Oil: 0.936
8. Braised Seafood with Japanese Tofu: 0.936
9. Prawns Kebab with Curry: 0.935
10. Dry-Braised Prawn: 0.935

**Input:** Kung pao chicken
**Recommendations (with similarity scores):**
1. Sizzling Chicken in Black Bean Sauce: 0.737
2. Yu-Shiang Shredded Pork (Sautéed with Spicy Garlic Sauce): 0.728
3. Braised Shark's Fin in Yellow Wine Sauce: 0.703
4. Sautéed Sliced Pork with Pepper and Chili: 0.686
5. Dry-Fried French Beans with Minced Pork: 0.686
6. Fried Scallops with XO Sauce: 0.685
7. Sautéed Sliced Pork, Eggs and Black Fungus: 0.680
8. Sautéed Fresh Abalone with Spearmint: 0.679
9. Stewed Shark's Fin with Shredded Duck: 0.677
10. Griddle Cooked Chicken Gizzard: 0.675

Figure 3: Example of Word2vec Skip-Gram Model Recommendation Results (Similarity Scores in Parentheses)

Figure 4: t-SNE Visual Representation of a Sample of Dish Names' Similarity

## 5.2 Collaborative Filtering Results

Adopting the ratings we have calculated, we run collaborative filtering on our training set through NMF and SVD. See Figure 5 and Figure 6 for the predicted ratings from the two methods. We can see that they produce somewhat different predictions: the NMF results look more symmetric, while the SVD predictions seem to skew more towards the left.

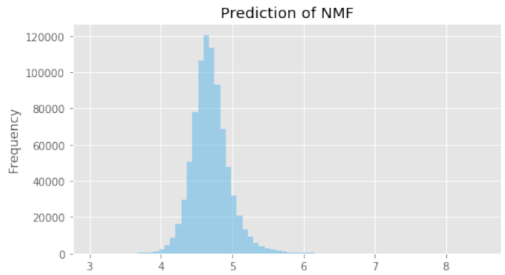


Figure 5: Predicted Ratings Using NMF

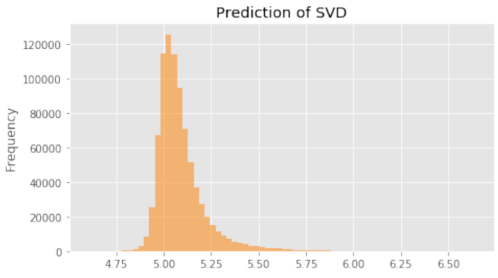


Figure 6: Predicted Ratings Using SVD

## 5.3 Error Metrics

### 5.3.1 RMSE of Prediction error

To evaluate prediction errors across models, we define the prediction error as the difference between the actual calculated rating and the predicted rating for the dish, i.e., $e_k^{(i)} = \hat{R}_k^{(i)} - R_k^{(i)}$, where $\hat{R}_k^{(i)}$ is the estimated rating that user $k$ has on dish $i$ predicted by our model. And we calculate the RMSE of prediction accuracy, defined as $RMSE = [\sum (e_k^{(i)})^2/N]^{1/2}$. Ideally a good model would have a small RMSE.

[11] T-Distributed Stochastic Neighbor Embedding (t-SNE) is a commonly used technique for dimensionality reduction and visualization of high-dimensional data. For more information, see [18].

[12] A complete and interactive version of the t-SNE graph is available in our github directory at https://github.com/zengyu714/food-chi/tree/master/res.

### 5.3.2 Miss and Recall

An alternative way to calculate error is using miss and recall. If the estimated prediction error of dish rating $e_k^{(i)} <= 0.05$, we define it as a prediction hit, and otherwise a miss[13]. Then we can calculate the recall, which measures the proportion of actual positives (hits) that are correctly identified, i.e., $\frac{TP}{TP+FN}$, where $TP$ = number of true positives and $FN$ = number of false negatives. So a higher recall indicates a higher model prediction accuracy.

### 5.3.3 Errors Results

We fit our models on both the dev set and the test set, and obtain the following values of RMSE and recall[14]:

| Model | Dev Set RMSE | Test Set RMSE | Dev Set Recall | Test Set Recall |
|---|---|---|---|---|
| NMF | 0.4574 | 0.5851 | 0.5081 | 0.5393 |
| SVD | 0.3317 | 0.3634 | 0.9173 | 0.9301 |

From the errors above, we can see that SVD has lower RMSEs in both the dev set and the test set compared to NMF. SVD also has much higher recall for both the dev set and the test set.

In addition, the test set RMSEs are very close to that of the dev set for SVD, whereas NMF has a larger gap between its test set RMSE and dev set RMSE (although still reasonably close). A similar observation can also be made for recall. This means that both the NMF and SVD algorithms are fairly robust and does not overfit, but SVD seems to outperform NMF in both accuracy and robustness.

As a result, SVD is selected as the winner and we provide Figure 7 as a sample of the SVD prediction errors. And Figure 8 is a histogram of all estimated prediction errors for SVD, with a vast majority of predictions having a prediction error less than 1.

| | user_id | item_id | rating | estimation | err |
|---|---|---|---|---|---|
| **251645** | 20794 | 1086 | 5 | 4.997827 | 0.002173 |
| **166778** | 56889 | 294 | 5 | 5.144476 | 0.144476 |
| **752427** | 158969 | 622 | 5 | 5.076316 | 0.076316 |
| **763904** | 105710 | 1226 | 5 | 4.9891 | 0.0109 |
| **162796** | 76755 | 621 | 5 | 5.010914 | 0.010914 |
| **752750** | 118278 | 313 | 5 | 5.085399 | 0.085399 |
| **292889** | 137962 | 1344 | 5 | 4.982749 | 0.017251 |
| **171279** | 122470 | 33 | 5 | 4.961425 | 0.038575 |
| **424391** | 137549 | 107 | 5 | 4.917739 | 0.082261 |
| **750003** | 20353 | 1888 | 5 | 4.953488 | 0.046512 |

Figure 7: Sample of Errors Using SVD

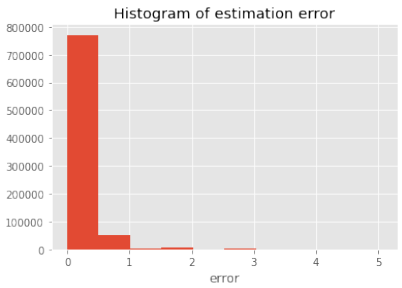


Figure 8: Prediction Errors of SVD

# 6 Conclusion and Future Work

Comparing the models, we think that the skip-gram model tackles the issue of a cold start, and directly gives us recommendations based on the input keywords and other users's dish preferences in our database. This allows us to bypass the issue that our dataset lacks ratings. However, it is not easy to make sense of these recommendations or find a good way to quantify its errors. Unlike the skip-gram model, we are able to conduct an error analysis on SVD and NMF. From the results, we conclude that the SVD model performs better at predicting ratings since it has lower RMSE on the dev set. Its test set error is also closer to the dev set error compared to NMF, which means it does not overfit and is fairly robust. Therefore it is the best algorithm for our recommendation system of Chinese dishes.

We recognize that different results may arise from different datasets, so there is still a lot of room for improvement. For future work, we can explore other recommendation algorithms such as doc2vec (an extension of word2vec), a hybrid system that combines both collaborative filtering and content-based filtering, a memory-based algorithm, and model-based algorithms. Implementing these models will allow us to conduct a more thorough error analysis and improve the prediction mechanism. We can also retrieve more data from Xia Chu Fang or even other recipe websites given the computational power to investigate model stability and robustness. Moreover, we can try to acquire more user and dish features and look for rating data on dishes so that we do not have to generate our own (which might introduce bias). Finally, we can also create a user interface where users can directly experience the dish recommendation system we build.

[13] As shown in Figure 8, errors are mainly densely distributed in range $[0, 1]$. Therefore, we have to choose a relatively small error threshold to calculate true positive rate (TPR), i.e., recall.

[14] Due to randomness, these values may vary slightly in each run, but are still very close to what is shown here.

# 7 Contributions

All students contributed fairly equally to this group project, each putting more focus on certain parts according to their specializations. Yu Zeng contributed more to data scraping, word2vec and matrix-factorization algorithm implementations. Yiting Ji contributed more to literature review, SVD and NMF algorithm research, error analysis, project management, and report write-ups. Yogi Huang contributed more to collaborative filtering research, data compiling, and poster write-up. Our code can be found here: https://github.com/zengyu714/food-chi.

# References


[1] O. Levy and Y. Goldberg, "Linguistic regularities in sparse and explicit word representations," *CoNLL. pp*, pp. 171 – 180, 2014. [Online]. Available: https://levyomer.files.wordpress.com/2014/04/linguistic-regularities-in-sparse-and-explicit-word-representations-conll-2014.pdf

[2] M. G. Ozsoy, "From word embeddings to item recommendation," *CoRR*, vol. abs/1601.01356, 2016. [Online]. Available: http://arxiv.org/abs/1601.01356

[3] [Online]. Available: https://code.google.com/archive/p/word2vec/

[4] TensorFlow, "Vector representations of words—tensorflow," 2018. [Online]. Available: www.tensorflow.org/tutorials/representation/word2vec

[5] X. Rong, "word2vec parameter learning explained," *CoRR*, vol. abs/1411.2738, 2014.

[6] X. Yan, "Manipulation robustness of collaborative filtering systems," 2009. [Online]. Available: https://search-proquest-com.stanford.idm.oclc.org/docview/305004354?accountid=14026

[7] H. Keshavan, Raghunandan, S. P. Boyd, A. Montanari, and B. V. Roy, "Efficient algorithms for collaborative filtering," Ph.D. dissertation, 2012. [Online]. Available: http://purl.stanford.edu/qz136dw4490

[8] X. Luo, M. Zhou, Y. Xia, and Q. Zhu, "An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems," *IEEE/IET Electronic Library (IEL) Journals*, vol. 1, 2014. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6748996

[9] Najaf, Safir, and Z. Salam, "Evaluating prediction accuracy for collaborative filtering algorithms in recommender systems." *KTH Royal Institute of Technology, School Of Computer Science And Communication*, 2016. [Online]. Available: kth.diva-portal.org/smash/get/diva2:927356/FULLTEXT01.pdf

[10] 2018. [Online]. Available: http://www.xtecher.com/Xfeature/view?aid=8321

[11] [Online]. Available: https://wenku.baidu.com/view/b825b8eb998fcc22bcd10d0d.html

[12] Y. Song, S. Shi, J. Li, and H. Zhang, "Directional skip-gram: Explicitly distinguishing left and right context for word embeddings," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. NAACL, 2018, short Paper.

[13] A. Minaar, "Word2vec tutorial part i: the skip-gram model," *Mccormickml.com*, April 2015. [Online]. Available: mccormickml.com/assets/word2vec/Alex_Minnaar_Word2Vec_Tutorial_Part_I_The_Skip-Gram_Model.pdf

[14] T. Mikolov *et al.*, "Efficient estimation of word representations in vector space." *Arxiv.org*, 2013. [Online]. Available: arxiv.org/pdf/1301.3781.pdf

[15] N. Hug, "SciPy: A Python scikit for recommender systems," 2015–. [Online]. Available: http://surpriselib.com/

[16] ——, "Matrix factorization-based algorithms," 2015–. [Online]. Available: https://surprise.readthedocs.io/en/stable/matrix_factorization.html?highlight=matrix%20factorization

[17] G. H. Golub and C. Reinsch, "Singular value decomposition and least squares solutions," *Handbook Series Linear Algebra*, 1970. [Online]. Available: https://link.springer.com/content/pdf/10.1007/BF02163027.pdf

[18] L. J. P. van der Maaten, "Accelerating t-sne using tree-based algorithms," *Journal of Machine Learning Research*, vol. 15, pp. 3221–3245, Oct 2014. [Online]. Available: https://lvdmaaten.github.io/publications/papers/JMLR_2014.pdf