Joke Recommender System

12/15/2022

Jiangyue Mao, Yeeun Jang

Introduction

The goal of our project is to construct a joke recommendation system that can recommend jokes to users based on the similarity among users' ratings (e.g. user-item interactions) using collaborative filtering and other advanced techniques. Although there are a few recommenders online that's using the same dataset, the conclusions that the authors drew are not very satisfying since most of them are using basic collaborative filtering techniques and/or even only picking the top 10 jokes with the overall highest ratings and recommending them to everyone, so the recommender is not customized on users' information. Therefore, we believe that using both collaborative filtering and advanced techniques involving autoencoders and even VAE will help the recommendation system become more developed and well-rounded. We've used collaborative filtering leveraging SVD, and we subtracted the mean ratings of all jokes and recommended the jokes with the highest ratings to users to be the naive system, and we implemented autoencoders to be our advanced recommender. We measured our success by evaluating the system on the test set using NDCG and hit rate, finding out that the baseline system has the highest NDCG @10 and hit rate @10 which is about 0.1 higher than that of the naive system and the hit rate is about 0.3 higher than that of the autoencoder. We realized that advanced systems may not always work better than simpler systems, and recommending items based on user-item interactions sometimes works better than simply recommending the most popular items or predicting items with high ratings using autoencoders.

We believe this recommendation system is very helpful for not only people who would love to enrich their lives by listening to more jokes but non-native English speakers and foreigners who are often asked to tell jokes in team-building events. Both authors of this project are non-native speakers and have encountered that situation during internships, and we believe such recommender systems can greatly relieve our pressure of not understanding native jokes and/or thinking of a new joke that everybody can easily understand.

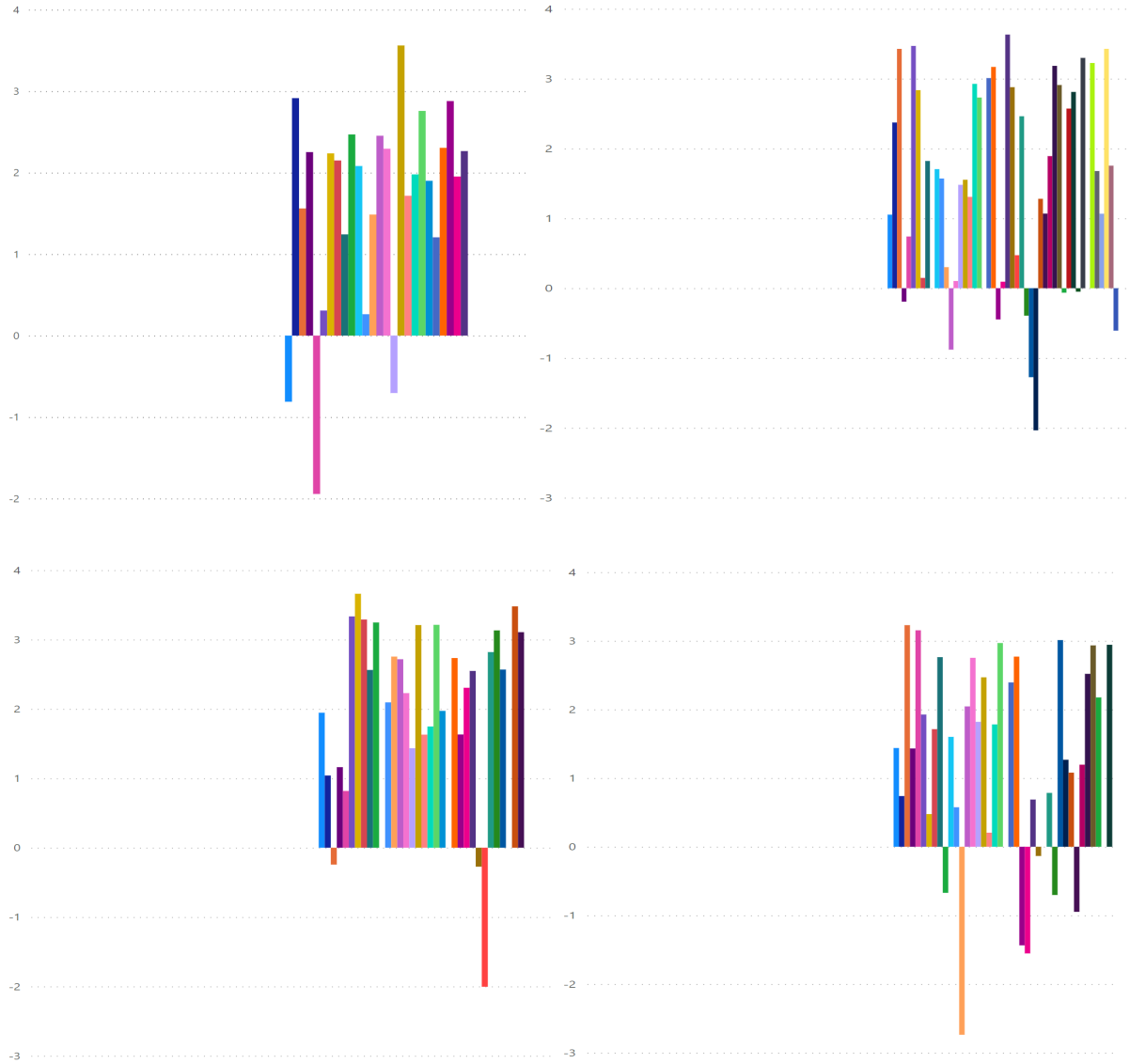
Data

The dataset we are going to use is from https://eigentaste.berkeley.edu/dataset/ dataset 3, which contains 150 jokes and around 2.3 million ratings. There are two separate CSV files, with the first one being the content (text) of the 150 jokes and the second one being the ratings. The rows of the second file represent users, and the columns represent jokes' IDs. 22 of these jokes have few ratings as they were removed as of May 2009 and deemed to be out of date. Their ids are: 1, 2, 3, 4, 5, 6, 9, 10, 11, 12, 14, 20, 27, 31, 43, 51, 52, 61, 73, 80, 100, 116. Each rating is from -10.00 to +10.00 and 99 corresponds to a null rating (the user did not rate that joke). Since there are some null values in the dataset that are shown as NA instead of 99, we translated those to 99 to maintain consistency. In order to preserve the accuracy of the analysis and minimize the

impact of numerical values of null values, we transformed all null values to 0 and ratings to 10-40 as a pre-processing process before model training. We did not give our own ratings, which corresponds to relevance to jokes as our dataset already contains ratings.

We also did a brief overview of the dataset:

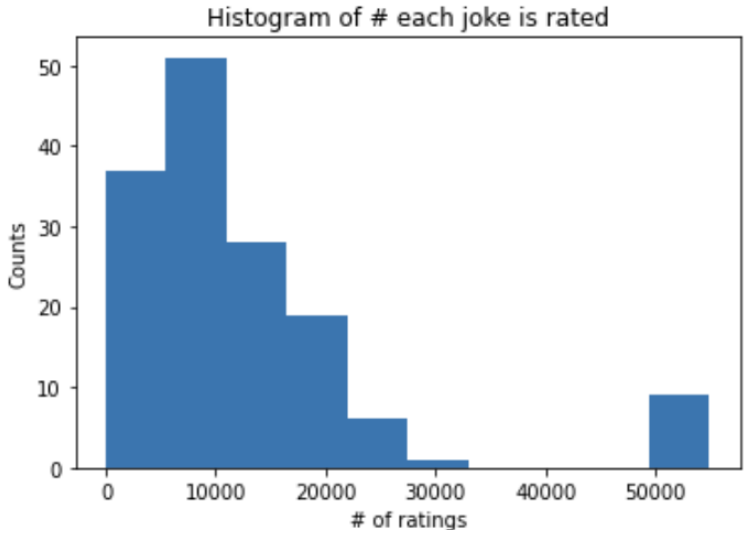
Y-axis is the average ratings of each joke, and different colors represent different jokes. Since there are more than 150 jokes, it's difficult to plot the ids all on the x-axis or make a legend for each plot.



Average ratings of each joke

From the charts above we can see that the ratings for all jokes are between 4 and -3, which means the jokes are not very hilarious to most users. The charts show that 99% of the jokes were rated by at least one user, and the bars that don't have values mean none of the users rated the joke so the joke only has null ratings. Since the authors of this project are all non-native

speakers and sometimes cannot tell if an English joke should be funny or not, we just expect our recommendation system to not recommend jokes with all null values.



A histogram of the number of each joke is rated

From the histogram above, we can see the distribution of the number of times each joke is rated by users. We expected this histogram to be right-skewed, which turned out to be true. One interesting fact is that there exists a few jokes that are rated by most of the users.

Related Work

Our dataset contains user ratings of the jokes, which are items. According to the class and a number of research studies in recommendation systems, user-item interaction data is widely analyzed, adding some auxiliary information. Adding auxiliary information in recommendation systems will be challenging to handle, as it depends on the dataset and the structure of the model we are going to use. Therefore, we believe that using advanced techniques involving neural networks will help the recommendation system become more developed and well-rounded. Here is our research on deep learning related recommendation systems and other related works that we refer to:

The structure of Neural Collaborative Filtering(NCF)[1] is a combination of two key features - Generalized Matrix Factorization(GMF) and Multi-layer perceptron (MLP). MLP can comparatively express more complex relationships than matrix factorization-based structures because it is non-linear and flexible. NCF is a remarkable approach that can learn both linear and non-linear representation. However, it also has some limitations that it is not using auxiliary information, and is not able to deal with the cold-start problem.

The second model we referenced, DeepFM[2], is a model that predicts Click-Through-Rate(CTR) by combining Factorization Machine(FM) structure and Multi-Layer-Perceptron(MLP) structure. FM component captures low-order feature interaction

and the MLP component captures high-order feature interaction. However, DeepFM has the same limitations as NCF.

In addition to the two advanced models above, we implemented AE according to AutoRec [3] with collaborative filtering by building encoders, hidden layers, and decoders. We think this is the most suitable method for our dataset so far among all of the papers that we referenced since it does not need timestamps and it takes auxiliary information into consideration. We also referenced VAE models for collaborative filtering[4] which combines item embeddings (learned from a sibling VAE network) with user ratings and applies it to the task of recommendation. It may potentially increase the performance of collaborative filtering but that comes at the cost of added model complexity, but some improvements are less than 0.01. We tried to implement this approach but the data we have does not really support this method.

The last approach that we considered is LightGCN[5], which includes only the most essential component in GCN — neighborhood aggregation — for collaborative filtering. Specifically, LightGCN learns user and item embeddings by linearly propagating them on the user-item interaction graph, and uses the weighted sum of the embeddings learned at all layers as the final embedding. Such a simple and linear model is much easier to implement and train. However, these models may also suffer from similar issues of NGCF since the user-item interaction graph is also modeled by the same neural operations that may be unnecessary.

Methods

We first preprocessed the data by dropping users only having negative ratings or few ratings(<5) and dropping the jokes that have never been rated. After filtering out, we had 44801 users and 140 jokes. In order to streamline our process with the existing research work, we transformed the ratings to 1 if a joke received a positive (high) rating and to 0 otherwise. We filled up null values(user-joke pair that has not been weighted) with 0. We then performed a train test split leveraging leave one out technique (we will describe in more detail in the next section). We also did some sanity checks to make sure the ratings of the test and training sets are not drastically skewed.

We used collaborative filtering as a baseline approach. We performed Matrix Factorization(MF), which is the most popular implementation. It captures the pattern between users and items by mapping both users and items to the integrated latent space, where user-item interaction is internally modeled. We first implemented the Latent Factor model (SVD) on the data and extracted user-factor and rating-factor matrices, and by multiplying them back together we got the predicted ratings for the missing data (i.e. jokes with null ratings). For the Naive model, we used the mean rating value. We calculated the mean rating for each joke and used that as a prediction. This can be also explained as recommending jokes with the highest rating(most favorite) to all users.

Furthermore, we implemented autoencoders leveraging Keras library referencing the documentation[6]. We built a single fully-connected neural layer as encoder and decoder, setting

the activation function to Relu and sigmoid for the encoder and decoder respectively. The architecture of our AE model is as below:

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 140)]             0

 dense (Dense)               (None, 32)                4512

 dense_1 (Dense)             (None, 140)               4620

=================================================================
Total params: 9,132
Trainable params: 9,132
Non-trainable params: 0
_________________________________________________________________
```

We fit the training data into the autoencoder model and we used the test set as the validation data for the model. We trained for 50 epochs with a batch size of 256, and we evaluated the results based on the predicted results from the test set.

Evaluation and Results

Splitting train and test data was a big challenge for our project. This is because our dataset is in a matrix format(user-item matrix) and both of the approaches we used take input data in a matrix format, but the training approach of the two models are different from each other. To be more specific, SVD does not have an actual training process. It decomposes the matrix into three matrices(U, D, and V) and then restores the user-item matrix by multiplication. On the other hand, Autoencoder uses separate users for training and then requires new users that are not included in the training set for prediction.

Therefore, we have two steps to split the test set: 1) Leave-one-out test data for each user among the top 10 ratings, 2) Split users for training and testing. We used the first approach only for the SVD model and used both for Autoencoder.

For the first approach, we used the whole dataset without splitting and predicting the ratings for every joke and user using SVD, in a user-item matrix format. The ground-truth matrix is the original matrix that contains both train and test data.

In the second approach, which is for Autoencoder, we split the result of the first approach(masked matrix) to train and test sets by users. We used around 10% of users as a test set.

To evaluate SVD, we calculated NDCG considering the ground truth binary matrix that includes test data, and the predicted (restored) result as a prediction user by user. For hit rate, we define it as a ‘hit’ if we have the test data joke for a certain user in the top k recommended jokes(with the highest prediction). Both NDCG and Hit rate are average values for all users.

To evaluate Autoencoder, we put the test users into the model trained and predicted the preference of test users. Then we calculated NDCG and Hit rate using the split test users of the ground truth matrix.

Evaluation of Naive predictor, SVD, and AE is as follows:

	Naive predictor(mean rating)	SVD(baseline)	AE(advanced)
NDCG@5	0.614	0.605	0.529
NDCG@10	0.563	0.608	0.529
Hit Rate@5	0.260	0.306	0.029
Hit Rate@10	0.440	0.514	0.274

Discussion

The performance of our autoencoder model is not very satisfactory as we only have a NDCG of around 0.5 and a low hit rate of around 0.2. This may not be enough to do useful science, but the baseline approach might be enough since it has a higher NDCG and hit rate. The results are not exactly what we expected since AE should be better at understanding user demands and item features. The difference might be due to the consistency of the dataset, i.e. the jokes of the same type (similar terms, similar context, etc.) are highly consistent that the users who like a certain type of joke will also like jokes of the same type since our SVD is based on similarities of user-item interaction. Also, since our dataset has some null values and we removed the jokes with few ratings and users with few ratings, the dataset may not be big enough to train the AE and let it learn weights successfully. Due to time constraints, we haven't tried parameter tuning or modifying the neural net architecture too much, but we expect the performance to be improved if properly implemented.

Conclusion

In order to build a recommendation system for jokes to benefit non-native speakers in a work setting and to entertain joke lovers, we implemented 3 different recommenders including collaborative filtering (baseline), mean rating recommendation (naive approach), and AE (advanced system), and compared their results. We found that the baseline model works the best with the highest NDCG and hit rate, whereas the AE works the worst. The baseline model may be used to preliminarily recommend some jokes. Possible future work includes switching to a bigger dataset with fewer null values and parameter tuning such as changing the batch size or number of epochs or changing layer architectures to improve the performance of AE. And if we can get datasets with timestamps, we could try VAE and other advanced deep learning methods as well.

GitHub repo: https://github.com/Annie-Yeeun-Jang/Info_retrieval_project.git

Other Things We Tried

In addition to AE, we also tried Variational AE referencing [7]. We tried to build encoders, decoders, and underlying architectures similar to AE, but it didn't work since our dataset does not contain timestamps (ie. the time that the users rated the jokes). We considered randomly assigning timestamps to each rating, but we thought it did not make sense since the dataset will lose its authenticity and the recommender may recommend entirely wrong jokes due to the wrong timestamps, which is not the result that we want.

What You Would Have Done Differently or Next

We may implement our methods on a different dataset given enough time, specifically on a larger dataset with timestamps and fewer null values. Given a better and larger dataset, we may implement other advanced machine learning and deep learning techniques such as NCF or light GCN. We hope our recommender system can eventually be served as a successful recommender for scientific use.

References

[1] He, Xiangnan, et al. "Neural collaborative filtering." Proceedings of the 26th international conference on world wide web. 2017.
[2] Guo, Huifeng, et al. "DeepFM: a factorization-machine based neural network for CTR prediction." arXiv preprint arXiv:1703.04247 (2017).
[3] Sedhain, Suvash, et al. "AutoRec: Autoencoders Meet Collaborative Filtering." WWW 2015 Companion, May 18–22, 2015, Florence, Italy. ACM 978-1-4503-3473-0/15/05. http://dx.doi.org/10.1145/2740908.2742726 .
[4] Kilol Gupta, Mukund Y. Raghuprasad, and Pankhuri Kumar. 2018. A Hybrid Variational Autoencoder for Collaborative Filtering. In Proceedings of Workshop on Intelligent Recommender Systems by Knowledge Transfer and Learning, Vancouver, Canada, October 2018 (RecSysKTL'18), 5 pages. arXiv:1808.01006v2 (2018).
[5] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '20), July 25–30, 2020, Virtual Event, China. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3397271.3401063.
[6] Francois Chollet. Sat 14 May 2016. https://blog.keras.io/building-autoencoders-in-keras.html.
[7] Sachdeva, Noveen, et al. "Sequential Variational Autoencoders for Collaborative Filtering." arXiv: 1911.09975.