A Reciprocal Embedding Framework For Modelling Mutual Preferences

R. Ramanathan,*[1] Nicolas K. Shinada,[1] Michinobu Shimatani,[2] Yuhei Yamaguchi,[2] Junichi Tanaka,[2] Yuta Iizuka,[2] Sucheendra K. Palaniappan[1]

[1]SBX Technologies Corporation, Tokyo, Japan
[2]Tapple Inc., Tokyo, Japan
{ramanathan, shinada, suchee}@sbx-corp.com,
{shimatani_michinobu, yamaguchi_yuhei, junichi_tanaka, yuta_iizuka}@matchingagent.co.jp

Abstract

Understanding the mutual preferences between potential dating partners is core to the success of modern web-scale personalized recommendation systems that power online dating platforms. In contrast to classical user-item recommendation systems which model the unidirectional preferences of users to items, understanding the bidirectional preferences between people in a reciprocal recommendation system is more complex and challenging given the dynamic nature of interactions. In this paper, we describe a reciprocal recommendation system we built for one of the leading online dating applications in Japan. We also discuss the lessons learnt from designing, developing and deploying the reciprocal recommendation system in production.

Introduction

There has been a huge surge in people going online to seek potential partners in recent years with the rapidly evolving social media platforms on the internet. Owing to their ease-of-use, low costs and privacy proposition, online dating apps have become mainstream and are increasingly preferred over traditional dating channels. However, there are several factors that make recommending partners an interesting and challenging problem.

First, people have eclectic tastes and their preferences change over time. For instance, at the time of registering in an online dating app, people would hold a certain preconceived preference profile and start *liking* an initial set of people recommended to them according to this preference profile. However rejection may happen and in reacting to the feedback they receive, they might broaden their preferences quickly when they find their interests are not converting into *matches*. On the other hand, it could also turn out that their preference can become more concentrated towards a set of users who have certain traits if the feedback they receive reinforces their preconceived interests towards those traits. Hence people's preferences drift over time with every interaction they have with the recommendation system and as such it is hard to model user intent even when there is sufficient training data.

*R. Ramanathan is the corresponding author.

Second, unlike traditional recommendation systems where users interact with static items, human interactions are bidirectional. This means getting the right recommendations to the right user at the right time is just not enough. More important is to also get the right target users who would reciprocate the subject users' interests. This is called as the *reciprocal recommendation problem*. It pertains to encoding and learning bidirectional preferences of users for serving recommendations that maximize the potential for mutual *likes*.

Third, we observe that the interaction network is often extemely sparse (density of 0.00001). Network density here is defined as the fraction of actual interactions over all possible interactions across all possible pairs of users. This indicates that users are very selective about their preferences and have fewer interactions, despite having similar tastes with that of other users. Hence in addition to solving the information overload problem, reciprocal recommendation systems should be careful not to overload the users with poor recommendations as rejection directly impacts user retention for the application.

While a wide variety of methods have been developed for learning latent representations in classical recommender systems (Agarwal and Chen 2009), applying them to the reciprocal setting is challenging due to the bidirectional nature of preferences. Collaborative filtering (Sarwar et al. 2001) techniques in literature can be classified into neighborhood-based methods (Xia et al. 2015, 2016; Pizzato et al. 2010) and model-based methods (Kleinerman et al. 2018; Wang et al. 2017). Recently (Neve and Palomares 2019) proposed a latent factor model based approach to compute the unidirectional preferences between two potential partners and fuse them to derive a single mutual preference score. While the modelling strategy of fusing preferences in our framework is closely related to this line of work, which the authors developed for a matrix factorization based model, our framework is not limited by the choice of the preference modelling algorithm.

Overview

Our embedding framework encompasses three fundamentally different preference modelling algorithms from matrix factorization to learning-to-rank to neural network based algorithms. From a practical standpoint, this is critical for three reasons. First, in contrast to a single model, our frame-

work allows us to tailor the recommendations based on the data distribution in production. For instance, user demographics and other user metadata features are difficult to incorporate in matrix factorization for new users who dont have past interactions. A neural network based method which easily allows concatenating these features as categorical embeddings performs better in such cold start scenarios. Second, in our setting of a heterosexual online dating application, we observe men are more active in sending *likes* or *dislikes* vis-a-vis women who are often reactive. The choice of the preference modelling algorithm is sensitive to this asymmetric distribution of interactions. Finally, we have to tune the preference aggregation function based on the preference modelling algorithm adopted in the previous step.

In essence, we present a novel framework designed to handle the nuances of real-world data to provide reciprocal recommendations in production that goes above and beyond a single algorithm.

The interactions in our online dating application consists of user activity when they *like* or *dislike* another user recommended to them. We first separate this interactions dataset into two disjoint unidirectional interaction subsets: men-to-women and women-to-men. From these datasets, we then learn the user preference vectors that model the latent unidirectional preferences of men towards women and of women towards men respectively. Finally, these unidirectional preferences are then combined using an aggregation function to learn the bidirectional preferences and eventually to score the reciprocal recommendations.

This paper is organized as follows: Section presents the perspectives on the dataset. Section describes the proposed reciprocal recommender framework. In Section , we discuss the challenges in taking our models to production and our deployment strategy using a serverless framework. Finally Section closes with the conclusions and possible lines of future work.

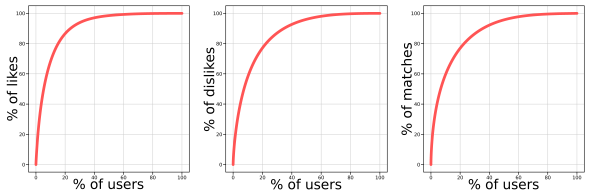
Data Analysis

Before we formalize our approach, we first conducted an empirical analysis of the data. Tapple[1] is a Japanese online dating application that connects young people based on their hobbies and interests. Once a user has been on-boarded to the application, they are presented profiles of the opposite gender which they can either *like* or *dislike*. A *match* happens when two users *like* each other and *matched* users can start a conversation by exchanging messages. The application, in service since 2014, has over 5 million registered users and has made over 200 million successful *matches*. The main focus of our approach is to optimize the potential for mutual *likes* between users.

Long-Tail Distribution of Interactions

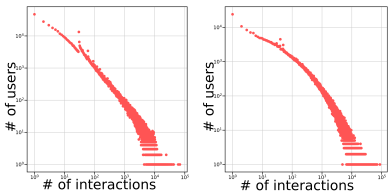
Figure 1 shows the cumulative distribution functions for the number of interactions that each user has been part of over 3 months. The activity quotient defined as the number of interactions per user for each of the interactions: *likes, dislikes* is shown in Figure 2. Similarly, the attractiveness quotient

[1]https://tapple.me/



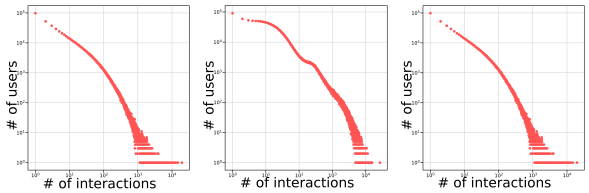
(a) Likes (b) Dislikes (c) Matches

Figure 1: Cumulative distribution of the interactions



(a) Likes (b) Dislikes

Figure 2: Distribution of the interactions given by a user



(a) Likes (b) Dislikes (c) Matches

Figure 3: Distribution of the interactions received by a user

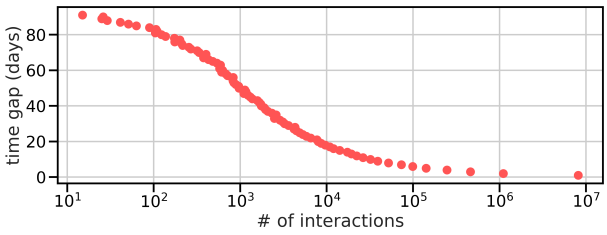
of each user both in terms of the received *likes*, *dislikes* and *matches* are depicted in Figure 3. It is easy to see that these quantities tend to approximately follow the power-law (X-axis and Y-axis are in log-scale), and suggest the presence of both 'power users' who are part of a large number of interactions, and a long tail of users with focussed interest in a small set of other users.

Temporal Evolution of User Activity

From Figure 4, we can understand the time-dependent behavior of a user's activity. Every point corresponds to counts of interaction pairs of a given user in log-scale. The Y-axis denotes the number of days passed since the particular user previously interacted with the reciprocal recommendation system. This indicates the existence of a large number of active users, as opposed to other reactive users with less frequent interaction patterns. If users' preferences change quickly with time which is prone to happen in a reciprocal recommendation setting, our models should also be updated frequently to account for this evolution.

Location Distribution of Users

Figure 5 shows a heatmap of location pairs extracted from all interactions between two users in a month across Japan's 47 prefectures. We see that most of the interactions happen



(a) Frequency distribution of the user interactions

Figure 4: Distribution of matches and user activity

within the same prefecture and rarely between neighbouring prefectures. Location specfic models developed to exploit this homogeneity will allow not only scalable solutions but also prevent online evaluation methods getting contaminated from network effects. More specifically, in A/B testing, if two users from different locations are assigned randomly to two different groups and interact with each other, higher order effects will result in one user affecting the behaviour of the other user as they interact. This can be alleviated by location-specific A/B testing where users within the same location can be assigned to the same group.

Evidently it is important to take into account the temporal and spatial dynamics of the dataset when designing a recommender system for such a large scale online dating platform. An interesting aspect of the data distribution here is the high match rate per user which could be directly modelled using traditional collaborative filtering algorithms devised for the user-item recommendation setting. However,

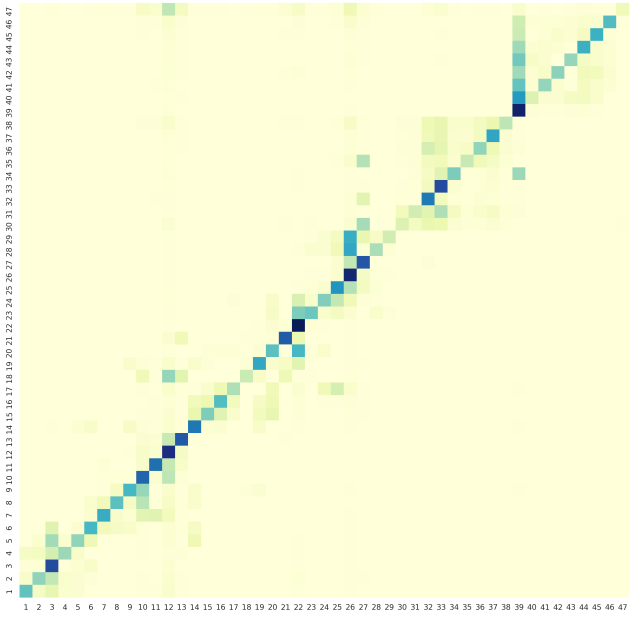


Figure 5: Interaction distribution across Japan's 47 prefectures

not all users would have past matches and one has to model user preferences using past behaviour i.e., from the profiles they have either *liked* or *disliked* in the past. As we will see from Section , recommendation quality can be improved for all users by first learning unidirectional preference embeddings of users separately and then fusing them to model their mutual preferences.

In the following section, we present our proposed recommender framework.

Reciprocal Recommendation System

We first formally define the reciprocal recommendation problem. In our setting focussed on heterosexual online dating, a reciprocal recommendation system can be viewed as a bipartite graph G defined by (X, Y, E) where X and Y denote the partitions of nodes that represent men and women in the system, E denotes the set of directed edges in G. Any directed edge from $x \in X$ to $y \in Y$ (or from $y' \in Y$ to $x' \in X$) is associated with a mapping ψ: $X \rightarrow Y$ (or $Y \rightarrow X$) that captures the preference of x for y (or the preference of y' for x'). Preference takes value 1 in case of a *like*, 0 in case of a *dislike* or unknown. More concretely, $\psi : (X \times Y) \cup (Y \times X) \rightarrow \{0, 1\}$.

Problem Formulation

Given a member $x_i \in X$ (or $y_j \in Y$), the objective of the reciprocal recommender algorithm –based on historical interactions between members– is to come up with a ranked list $[y_i^{(1)}, y_i^{(2)}, ..., y_i^{(n)}]$ (or $[x_j^{(1)}, x_j^{(2)}, ..., x_j^{(m)}]$) of potential preferences likely to result in mutual *likes*.

We first learn the unidirectional preferences from X to Y and Y to X using collaborative filtering. A number of techniques from latent factor models to neural networks have been widely adopted in the classical user-item recommendation context in literature. We will focus on three techniques which allow us to navigate the entire model spectrum from simple, linear models that are scalable, on one end to more flexible, non-linear models which enable feature-rich representations, on the other end.

Latent factor models based on matrix factorization have been popular in the industry for a long time. Training and serving matrix factorization models that are based on the simple inner-product operator, is easily scalable to large datasets. While matrix factorization optimizes for direct prediction for relevance by optimizing point-wise scores, learning-to-rank techniques offer a class of methods that optimize on partial ordering of scores to learn a ranked order of relevance. On the other hand, deep neural networks (Dziugaite and Roy 2015; Salakhutdinov, Mnih, and Hinton 2007; He et al. 2017) allow us to exploit the rich side information like demographic and temporal features of users which are hard to incorporate into matrix factorization based methods.

Once we learn the vector representations of X's preferences and the vector representations of Y's traits from their unidirectional preferences: X to Y, and that of Y's preferences and X's traits similarly from the unidirectional preferences: Y to X, they can be combined to produce the final prediction of the mutual preferences using an aggrega-

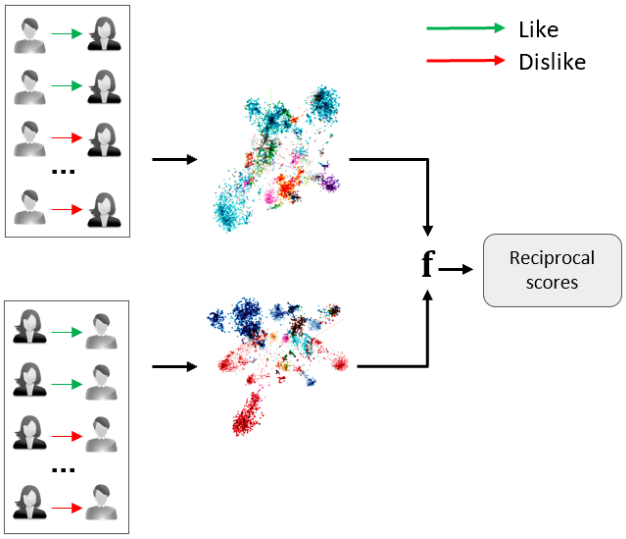


Figure 6: Modelling reciprocal user preferences

tion function. A schematic representation of the reciprocal embedding framework is shown in Figure 6. A range of aggregation strategies can be applied to fuse unidirectional preferences from arithmetic mean, harmonic mean, product, weighted mean, set union, intersection, inverse product between recommendation ranks, etc.

Matrix Factorization

We consider the weighted version of matrix factorization in our experiments. Given a binary (or real-valued) matrix $R_{|X|\times|Y|}$ (and $R'_{|Y|\times|X|}$) which captures the observed interactions, the matrix factorization model approximately decomposes $R_{|X|\times|Y|}$ (and $R'_{|Y|\times|X|}$) into a dot product of two low-rank matrices of dimension k:

$$\underbrace{R_{|X|\times|Y|}}_{\text{Interaction Matrix}} \approx \underbrace{P_{|X|\times k}}_{\text{X-User Matrix}} \cdot \underbrace{Q^T_{|Y|\times k}}_{\text{Y-User Matrix}} \quad (1)$$

These low-rank approximations constitute the embedding matrices that model the user preferences. Matrix factorization optimizes the regularized squared loss objective using stochastic gradient descent:

$$= \min_{P,Q} \sum_{x,y\in R} \underbrace{\left(r_{xy} - p_x^T q_y\right)^2}_{\text{Squared Loss}} + \underbrace{\lambda_p ||p_x||^2 + \lambda_q ||q_y||^2}_{\text{L2 Regularization}} \quad (2)$$

where the preference score between embeddings p_x and q_y is given by $p_x^T q_y = \sum_{j=1}^{k} p_{xj} \cdot q_{yj}$.

Learning to Rank

In contrast to the point-wise loss function of matrix factorization, Bayesian Personalized Ranking (Rendle et al. 2012) (BPR) formulates the unidirectional preference extraction of the reciprocal recommendation task as a pair-wise ranking optimization problem that directly optimizes the AUC (Area Under The Receiver Operating Characteristic Curve) metric. In BPR, training examples are pairs of positive and negative samples for a given user. (x, y_i, y_j) denotes that a user x prefers a potential partner y_i over y_j. The BPR-Loss function is given by the objective:

$$= \underbrace{-\sum_{x} \sum_{(y_i,y_j)\in R_x} \ln \sigma(s_{ij}(\Theta))}_{\text{Log-Loss}} + \underbrace{\lambda_\Theta ||\Theta||^2}_{\text{L2 Regularization}}, \quad (3)$$

where Θ is the posterior parameter by Bayes' rule that will be learnt, $s_{ij}(\Theta)$ is a difference function $s_{ij}(\Theta) = s_i(\Theta) - s_j(\Theta)$ where s_i is a preference scoring function for a potential partner y_i given a user and σ denotes the sigmoid function. The preference scoring function is usually given by the inner product $p_x^T q_{y_i}$ for any given user x as seen earlier in matrix factorization.

Neural Network

Recently, a number of neural network based recommender systems have shown that neural functions better model complex interactions (He et al. 2017) not modelled by matrix factorization based methods due to the limitations of the dot-product as it violates the triangle inequality. A feed forward neural network to model the interactions can be defined as:

$$= \Phi(p_x^T, q_y) \quad (4)$$

where Φ denotes the interaction function of the neural network which optimizes the squared loss objective from Equation 2. The users embeddings are also optimized jointly with the model parameters in Φ through stochastic gradient descent. Side features that embed user metadata can be concatenated to the embedding vectors to form a wide first layer. This is then followed by several layers of non-linear activations (fully connected ReLU layers). We use negative sampling to reduce the model complexity by modifying only a small set of weights (Goldberg and Levy 2014).

The Reciprocal Recommendation Pipeline

From a practical implementation perspective, we adopt the classical two stage information retrieval pipeline to generate top-K recommendations. The first stage is the candidate generation stage that generates a few hundred good candidates from hundreds of thousands of users, using the collaborative filtering algorithms presented above.

Recommendations from the candidate generation stage are then further filtered in the second stage where we perform online re-ranking to produce a few tens of high quality recommendations. Balancing for relevance and discovery is key to keep the recommendations fresh and diverse. Since users' active times may vary, it is important to recommend users who are more recently active in the application lest the possibility of inactive users not responding to *likes* and thus resulting in poor user experience. The core purpose of the online re-ranking stage is to balance multiple objectives to blend recommendations from relevance models, recently logged in users, newly registered users and debiasing popular recommendations.

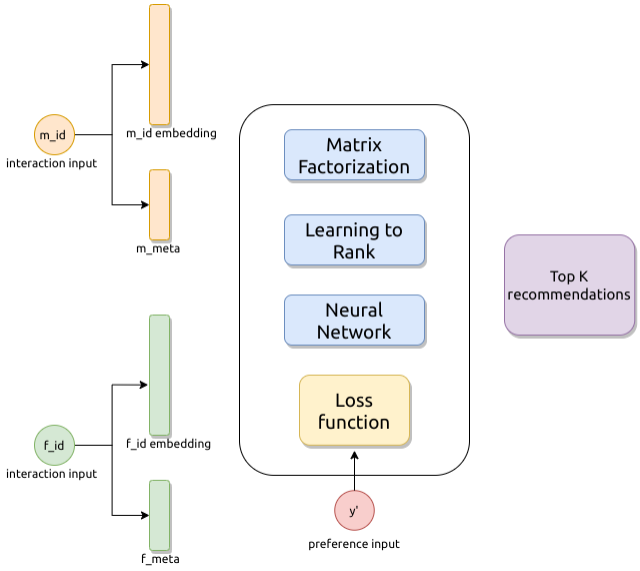


Figure 7: Candidate generation from reciprocal embeddings

We conduct our offline experiments in Python 3.6 on a Linux PC with Ubuntu 18.04 environment on an 8-core Intel Core-i9-9900K CPU with 64 GB of RAM. In all our experiments, we use one month's data as training data, and the subsequent week's data as validation data. We use average recall as an offline evaluation metric. Since further re-ranking of the candidate pool at serving time filters recently active users, online A/B testing can be performed for more accurate evaluation.

Evaluation

We present the initial results from our proposed candidate generation framework. To evaluate the quality of the recommendations, we compare against a baseline model that learns user preferences solely based on past matches information. We recall that the training data distribution has a significantly high number of matches per user as can be seen from Figure 3 from Section . It is reasonable to compare against a conventional recommender system (Match-RS) that learns mutual preferences directly from the match information (this does not need combining unidirectional preferences) as a baseline. In comparison to the best performing Match-RS model, our reciprocal recommender system (RRS) achieves significantly higher average recall on test users for both *likes* and *matches*.

In order to measure the live performance of our approach

Method	Average Recall	
	Matches	Likes
RRS vs Match-RS	+16.9%	+26.74%

Table 1: Performance comparison of our reciprocal recommender system against a conventional recommender system based on match data

in production, we devised *engagement* and *conversion* as online metrics. *Engagement* measures the fraction of recommendations which converted into *likes* while *conversion* quantifies the fraction of *likes* which eventually converted into mutual *likes* or *matches*.

Initially while the recommendations generated from the vanilla candidate generation stage resulted in significant performance gains in *engagement*, we observed hardly any change in *conversion*. After deploying the online re-ranking routine, *conversion* scores improved by upto +60%.

Parameter Configuration

For all our models, we used Adagrad (Duchi, Hazan, and Singer 2011) and Adam (Kingma and Ba 2014) which are popular extensions to the stochastic gradient descent based algorithm widely used in many applications. We set the learning rate parameter to 0.001 and embedding dimension hyper-parameter to 50. For the neural network model, we devised 3 hidden layers that handle the complexity, regularized the model using a dropout layer, with dropout factor set to 0.5 to avoid overfitting.

Model Serving

In this section, we present the deployment strategy for scaling our models in production. Our reciprocal embedding framework processes the input data (implicit feedback via clicks and explicit user-profile data) collected from the upstream mobile app and for each user returns a set of potential users who are likely to match. It consists of the following four core components:

1. Preprocessing
2. Training
3. Batch Inference
4. Live Inference End-point

Figure 8 shows the individual components of our architecture and the flow of control and data between them. Each of these components would have different operational environment requirements for different models and routines. To tackle this, we dockerize the routines to containerize each of the components into their respective microservices.

A trigger service listens at regular intervals for events that capture latest input dataset being dropped into a specific location in the cloud storage. This input dataset usually contains both the implicit and explicit user behaviour information. As soon as this dataset arrives, the trigger service spawns a cloud instance that runs the preprocessing service which subsequently triggers the training service that hosts our collaborative filtering framework. Upon successful retraining of user embeddings, another cloud instance is instantiated in which the inference algorithms generate the recommendations. Each cloud instance is dynamically configured to suit the compute and memory requirements pertaining to the nature of the data and compute load demanded by the particular service, and created only when required and turned off after their use.

Recommender API

Next we developed a REST-API service that processes requests from the mobile application environment via GET and POST methods for requested user IDs. Since our recommendations are precomputed and uploaded to a distributed key-value store, every request is forwarded to the distributed key-value store's data stream which in turn processes the requests. If the user ID in the request is that of an existing registered user in the system, their pre-comptued recommendations are returned from the distributed key-value store. On the other hand, if the user is a newly registered user, the API notifies the same to the application environment and that it expects a POST method containing the user's demographic features in its body. Based on the received user information, requests are forwarded to a live end-point service that hosts the models for online inference in memory. Consequently the recommendations are generated for the new user and before forwarding the recommendations to the application, a copy of the new recommendations generated are stored back in the distributed key-value store.

As new models are built in successive retrain cycles, recommendations generated by old models from previous runs become stale and have to be gracefully discarded. We devised a time-to-live mechanism that enables periodic disposal of old recommendations after a certain time period automatically.

Workflow Orchestration

Developing a solution architecture for production workflows from scratch is a complex process. When adding more features to our distributed recommender system (Ramanathan, Shinada, and Palaniappan 2020), the more fine-grained the microservices become, the more the chances of the connections between them getting messy. To avoid this overhead, its important we start with a simple architecture and gradually build granularity.

We organize the sequence of events in our recommendation workflow as a state machine. The notion of a state machine allows to abstract the individual microservices and the connections between them into a sequential structure that

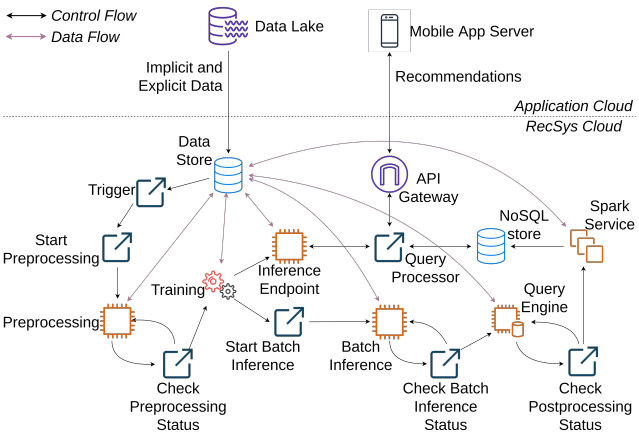


Figure 8: Solution architecture

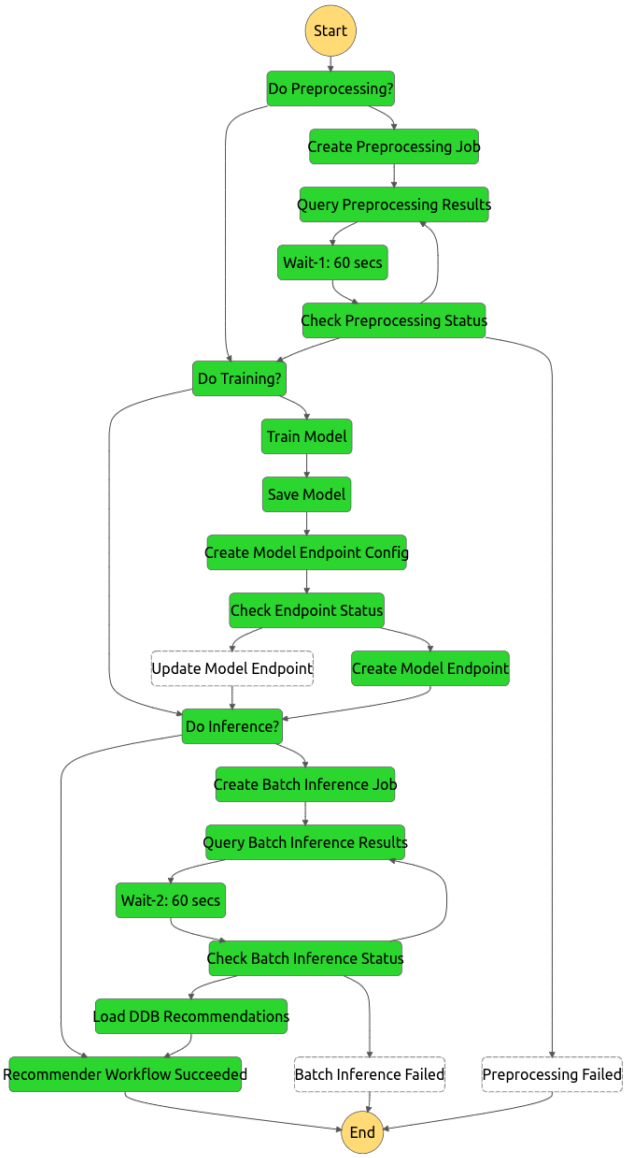


Figure 9: A successful execution of the workflow

streamlines data access and error handling. For each execution of the state machine, we parametrize the run by initializing a set of input parameters. In addition to forwarding the input parameters, every state also forwards a set of output parameters to its next state(s). Its through these parameters, the different microservices in our system communicate with each other dynamically. We used AWS managed infrastructure to deploy our workflows. Figure 9 shows a sample run of this end-to-end workflow.

Engineering for Production

Preprocessing the Dataset In live systems, filtering out noise from data is as important as the modelling algorithm. We remove users with invalid values for 'birth-date' and 'last login-date' fields. We ran multiple experiments on dif-

ferent sizes of training data with users who have logged in the last 7 days, 15 days and 1 month. We found 1 month to be a reasonable trade-off between user coverage and model accuracy.

Location-Aware Modelling Users' location plays a critical role in recommending relevant people in their neighbourhood. We build region-specific models and before every retrain cycle, user information is segregated based on their locations and sent to respective models for retraining. This reduces model complexity and at the same time allows to build a scalable system in production.

Precomputing Recommendations Once trained, keeping the models always online to serve recommendations would be expensive. We pre-compute the recommendations and store it in the distributed key-value store so recommendations can be retrieved in sub-100 ms query time. Furthermore, transferring the recommendations generated from a cloud instance to the distributed key-value store would take multiple hours. To this end, we first split the entire recommendations dataset into smaller chunks using an SQL query engine and store it in an intermediate distributed data store. One can then spawn multiple Spark workers which upload the data from the intermediate data store in parallel to the distributed key-value store.

Off-the-shelf distributed key-value stores like Amazon DynamoDB provide automatic bandwidth scaling for both burst loads used to transfer the recommendations dataset and for sporadic reads that serve recommendations. By leveraging this auto-scaling capability, our upload times reduced multi-fold from over 12 hours to under 30 minutes.

Live Inference End-point For cold start users who do not have past interactions, an always-on instance with the deployed model serves as a streaming end-point. For every new user, we create a pseudo embedding for them and generate recommendations using an approximate nearest neighbours algorithm[2] that retrieves recommendations in sub-linear query time. We note that the quality of our embeddings is robust to the choice of the approximate nearest neighbours algorithm.

Conclusion

We have presented a practical workflow for a reciprocal recommendation system from conception to serving the models in production. Our framework is generic enough that a wide variety of models can be built on it to learn user embeddings that model mutual preferences and subsequently generate candidate recommendations. From a modelling perspective, while we were able to address the compute-intensive fusion routines that aggregate unidirectional preferences using GPU instances on the cloud, it's interesting to explore more efficient methods to learn mutual preferences in a joint embedding space without having to fuse preferences from two different sub-spaces. Another line of approach would be to use neural attention methods to focus on relevant candidates that embed mutual interest from a large set of past unidirectional preferences.

From an evaluation perspective, while our method achieves significant gains in offline performance, they are not representative of online performance which places more importance on recently active users who would respond to other users' interests. Furthermore, online metrices for A/B testing to evaluate models in traditional recommender systems cannot be directly extended to reciprocal recommendation systems as they suffer from spill-over effects. Fine-grained clustering techniques and random assignment strategies for clusters could be devised to assign control and treatment groups. This will make evaluation more robust to network effects.

Finally, more efficient strategies using multi-objective models for balancing trade off between recommending users with high potential for mutual *likes* and new active users can boost recommendation quality.

[2]https://github.com/spotify/annoy

Acknowledgments

The authors would like to thank Prof. Hiroaki Kitano from SBX Corporation, Tokyo for guiding this effort.

References

Agarwal, D.; and Chen, B.-C. 2009. Regression-based latent factor models. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 19–28.

Duchi, J.; Hazan, E.; and Singer, Y. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research* 12(7).

Dziugaite, G. K.; and Roy, D. M. 2015. Neural network matrix factorization. *arXiv preprint arXiv:1511.06443* .

Goldberg, Y.; and Levy, O. 2014. word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722* .

He, X.; Liao, L.; Zhang, H.; Nie, L.; Hu, X.; and Chua, T.-S. 2017. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, 173–182.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* .

Kleinerman, A.; Rosenfeld, A.; Ricci, F.; and Kraus, S. 2018. Optimally balancing receiver and recommended users' importance in reciprocal recommender systems. In *Proceedings of the 12th ACM Conference on Recommender Systems*, 131–139.

Neve, J.; and Palomares, I. 2019. Latent factor models and aggregation operators for collaborative filtering in reciprocal recommender systems. In *Proceedings of the 13th ACM Conference on Recommender Systems*, 219–227.

Pizzato, L.; Rej, T.; Chung, T.; Koprinska, I.; and Kay, J. 2010. RECON: a reciprocal recommender for online dating. In *Proceedings of the fourth ACM conference on Recommender systems*, 207–214.

Ramanathan, R.; Shinada, N. K.; and Palaniappan, S. K. 2020. Building a reciprocal recommendation system at scale from scratch: Learnings from one of Japans prominent dating applications. In *Fourteenth ACM Conference on Recommender Systems*, 566–567.

Rendle, S.; Freudenthaler, C.; Gantner, Z.; and Schmidt-Thieme, L. 2012. BPR: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618* .

Salakhutdinov, R.; Mnih, A.; and Hinton, G. 2007. Restricted Boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning*, 791–798.

Sarwar, B.; Karypis, G.; Konstan, J.; and Riedl, J. 2001. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, 285–295.

Wang, S.; Tang, J.; Aggarwal, C.; Chang, Y.; and Liu, H. 2017. Signed network embedding in social media. In *Proceedings of the 2017 SIAM international conference on data mining*, 327–335. SIAM.

Xia, P.; Liu, B.; Sun, Y.; and Chen, C. 2015. Reciprocal recommendation system for online dating. In *2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, 234–241. IEEE.

Xia, P.; Zhai, S.; Liu, B.; Sun, Y.; and Chen, C. 2016. Design of reciprocal recommendation systems for online dating. *Social Network Analysis and Mining* 6(1): 32.