Deep Dive Into Production ML workflow for Recommender System on GCP

Adrian Hsu
UCLA CS Department
UID: 405331564
adrianhsu@g.ucla.edu

Abstract

Deep Learning requires a huge amount of computational power and available memory, and there are many cloud services that could boost training efficiency significantly. However, most of the cloud providers are aiming for a Business-to-Business model in their value proposition, resulting in that students and researchers at school have little exposure to a great number of billable products available on the cloud. In this paper, we will dive into Google Cloud Platform and explore several PaaS (Platform-as-a-Service) that could make our Machine Learning algorithms scalable and distributable. We will demonstrate an end-to-end Collaborative Filtering model targeting at Recommender System. The algorithm part is based on Matrix Factorization and optimized by Stochastic Gradient Descent (SGD) written in Tensorflow. Our model is deployed to Google AI Platform to leverage the computational power, and data pipelines are written in Apache Beam and hosted on Google Dataflow. The trained model and all preprocessed data will be stored on Google Cloud Storage in a distributed manner. This project is currently open-source on Github with an aim of helping ML researchers to get a glimpse of the benefits of Cloud Computing.

1 Introduction

Machine Learning is apparently dominating the technology industries sector, and we shall all agree that by leveraging cloud services like Amazon AWS, Microsoft Azure, or Google Cloud Platform, we could scale up our machine learning models with very little effort. However, for most computer science students at college, they mostly would prefer using self-host lab workstations or even buying their own GPU, unconsciously neglecting the huge benefits they could have owned if they chose to train their models on the cloud. Even if they have a basic understanding of the cloud, the most frequent services they will leverage are still Virtual Instances services like Amazon EC2 or Azure VM since that they are more comfortable with one single machine.

In fact, there are many great cloud services available, and lots of engineers from industries are already building models based on these billable cloud products in their day-to-day life. However, at schools and labs, students and young researchers are either not aware of them or overwhelmed by the complexity of product documents and their configurations.

In this paper, we divide our work into two sections: ML model creation and cloud service utilization. In the first section, we will go through a common user scenario in the large-scale Recommender Systems area: consider a recommendation system for movies in which the training data is composed of a significant amount of users and items ID, and the relationship between them is represented as a rating, which is a score ranging from 1 to 5. We want to find hidden features behind the given data through Collaborative Filtering [2] by training our ML model as a huge sparse matrix through matrix factorization, and then we will optimize the objective function of our model by Stochastic Gradient Descent (SGD) for fine-tuning [3]. As a result, we want our model to be capable of indicating the possible rating granted by a new user for a certain movie. We want to guess the users' taste based on AI.

In the second section, we will look into many aspects of cloud computing: Data Storaging for permanently keep our data available, Data Pipelining for data engineering tasks such as creating an ML workflow in a DAG (Directed-Acyclic-Graph) manner [5], and an AI platform that could serve and monitor our ML models with flexible computational power provided. Also, we will leverage an open-source unified programming model named Apache Beam for defining and executing data processing workflows in a MapReduce-like manner, and at the same time equipping with incomparable capabilities to run the workflow on any execution engine [6]. To be specific, we will have Google Cloud Storage (GCS) for data storaging [10], Apache Beam for defining portable and extensible ML workflows, Google Dataflow for auto-scaling resources and scheduling jobs with dynamic load balancing policies, and finally we have Google AI Platform to deploy our ML model and serve with elastic computational resources. The production ML workflow we built was evaluated and validated thoroughly, indicating that it could significantly boost the efficiency for deploying a model from ideation to deployment seamlessly [7].

Regarding the algorithm part, we will evaluate our Machine Learning models based on their Mean Squared Error, and then we plot the loss function to validate that our models are proved to be valid [8]. On the other hand, we will explore several combinations of resource management configurations such as using (1) only 1 master node versus (2) 1 master node and 2 worker nodes and more; furthermore, we will leverage the "Consumed ML units" provided by Google AI Platform to find the most cost-effective solution through trial and errors. Generally speaking, we believe that given a larger task, it will be beneficial to have multiple nodes to work together.

We tested our models with two datasets having different order of data size: MovieLens 100K and MovieLens 25M built by [12]. To be specific, MovieLens 100K denotes that there are 100,000 entries in our rating list, and 25M means 25 Million entries. By simple math calculation with respect to the approximate memory usage, we could see that to be able to accommodate 25 Million entries, our laptops should be not possible to run the tasks. We can only use `n1-highmem-16` that has 16 virtual CPUs and 104 GB memories to do so. Again, it infers that to scale up our ML models to a bigger scale, leveraging cloud services is definitely a good choice.

Finally, we will also discuss what is lacking in our end-to-end ML workflow, and provide some future directions. This project is open-source and available as a Github repository [13], enabling other ML researchers to leverage and explore more possibilities in Cloud Computing for Artificial Intelligence.

2 Methodology

Collaborative Filtering Collaborative filtering (CF) is a family of algorithms that could find the hidden features between users and items, and hence generate the recommendations [2]. It observes user behavior to make recommendations; moreover, there is no need to provide detailed information about each user – a great collaborative filtering model could learn these relationships by itself. To be specific, a collaborative filtering model is based on that user who interact with items in a similar manner should share some hidden preferences. Secondly, the model also follows the principle that users with shared preferences are likely to have similar tastes to the same items [8]. For instance, if user A is interested in Action Movies and so do user B, then it is possible that a random action movie such as Transformers could fit both of their tastes. Briefly speaking, we can leverage the model to find users' hidden interests. In this section, we will first go through how we implemented the algorithm, and then for the next section, we will dive into the billable cloud services.

2.1 Matrix Factorization

Matrix Factorization (MF) is a way to solve a collaborative filtering problem. There are many other algorithms available (e.g., Neural Network), but Matrix Factorization is generally recognized by the public that it is the most intuitive and relatively efficient one. Suppose that we have a matrix consisting of user IDs and their ratings for the products with item IDs. Each row corresponds to a unique user, and each column represents an item. Each entry in the matrix is a user's rating for the item. For our MovieLens scenario, the rating will range from 1 to 5. If a user has never rated an item, the matrix entry will be zero. Generally speaking, this rating matrix should be very sparse, meaning that each user has only rated a relatively small amount of items compared to the entire set. The majority of the matrix, often greater than 99 percent are zeros [8]. Let's say we have a rating matrix $\boldsymbol{R}$, and we want to factorize this matrix into a user matrix $\boldsymbol{P}$ and an item matrix $\boldsymbol{Q}$ as shown in Fig. 1. We can view this process as compressing sparse information in $\boldsymbol{R}$ into much lower- dimensional spaces $\mathbf{u}\times\mathbf{k}$ and $\mathbf{k}\times\mathbf{i}$. by multiplying the user matrix and the item matrix, we could theoretically find an $\boldsymbol{R'}$ that is an approximation of $\boldsymbol{R}$. [1] We take the dot product of the two vectors: **u** and **i** of the user u for item i, and then we can express this relationship as:

$$\mathbf{r}_{ij} = \sum_{k=1}^{k} \mathbf{p}_{ik}\mathbf{q}_{kj}$$

Now, we have to find a way to obtain P and Q, and that's when SGD comes into play.

2.2 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is an iterative method for optimizing an the objective function [3], and SGD can perfectly fit into such use cases as objective function for Matrix Factorization is differentiable [3].

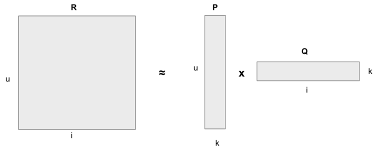


Figure 1: Matrix Factorization: Rating Relationship

$$minimize \;\; \mathbf{e}_{ij}^2 = (\mathbf{r}_{ij} - \sum_{k=1}^{k} \mathbf{p}_{ik}\mathbf{q}_{kj})^2$$

Here we consider the squared error since that the estimated rating could be either higher or lower than the rating score. To minimize the error, we will do partial differentiation to the above equation, with the two variables **p** and **q** respectively. As the formula is differentiable, we could finally obtain the gradient [1]. Thanks to Tensorflow, we don't need to compute the differential values ourselves; instead, we can do the following to get our expected result [4]. In our `trainer/task.py`, we have:

```
# trainer/task.py
with tf.GradientTape(persistent=True) as tape:
  predictions = tf.reduce_sum(
    tf.gather(model.U, A_train.indices[:, 0]) *
    tf.gather(model.V, A_train.indices[:, 1]),
    axis=1)
  loss_obj = tf.keras.losses.MSE(A_train.values,
       predictions)

gradients = tape.gradient(loss_obj,
     model.trainable_variables)
optimizer =
     tf.keras.optimizers.SGD(learning_rate=learning_rate)

optimizer.apply_gradients(zip(gradients,
     model.trainable_variables))
```

To be specific, `model.U` is our user matrix and `model.V` is our item matrix, and they are both sparse. We will compare the result with the `predictions` matrix. In Tensorflow, we can easily apply our gradients to the Mean Squared Error with very little effort. One thing to note is that we didn't add a regularization term in our settings. Theoretically, our model performance should improve if we have added it to our objective function [1].

3 Billable Google Cloud Services

Google Cloud Platform is a suite of cloud computing services that runs on the same infrastructure that Google uses internally. In this project, we leveraged four billable cloud services provided by Google: Google Cloud Storage (GCS), Apache Beam, Google Dataflow and Google AI Platform. As shown in Fig. 2, we could briefly decouple the process into three parts: load data, build data

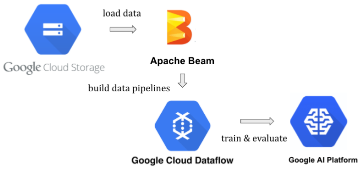
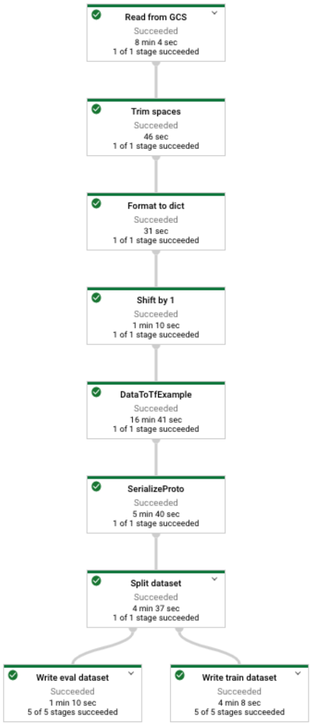


Figure 2: ML Workflow deployed on GCP



Read from GCS
Succeeded
8 min 4 sec
1 of 1 stage succeeded

Trim spaces
Succeeded
46 sec
1 of 1 stage succeeded

Format to dict
Succeeded
31 sec
1 of 1 stage succeeded

Shift by 1
Succeeded
1 min 10 sec
1 of 1 stage succeeded

DataToTfExample
Succeeded
16 min 41 sec
1 of 1 stage succeeded

SerializeProto
Succeeded
5 min 40 sec
1 of 1 stage succeeded

Split dataset
Succeeded
4 min 37 sec
1 of 1 stage succeeded

Write eval dataset
Succeeded
1 min 10 sec
5 of 5 stages succeeded

Write train dataset
Succeeded
4 min 8 sec
5 of 5 stages succeeded

Figure 3: Dataflow representation for Apache Beam executions

pipeline, and train the model. To enable all of these services, we have to first create a GCP project, and I named it as **reco-sys-capstone**; also, we have to set up our billing address and methods. Then, we could do the first step: load data and data extraction.
To enable GCS, we have to first create a GCS bucket. We named it as `gs://ahsu-movielens` (note that it has to be a *globally unique* identifier). We use a python script `data-extractor.py` to download MovieLens data directly from the official website, and then write the data to GCS. The data extractor is mostly based on [9] with slight modifications. We could utilize tf.gfile package methods implemented in Tensorflow to make directories and write data into the cloud bucket. GCS is relatively easy to comprehend as we could think of it like a normal file system, and each bucket is analogous to a root directory.
Secondly, we will do preprocessing on our data in `preprocess.py`. To address this, we have Apache Beam, being maintained mostly by Google, which is a unified programming model that enables users to implement batch and streaming data processing jobs that run on any execution engine [6]. It is unified, portable, and extensible. For instance, we could either work with Google Dataflow, Flink, Spark, and many big data processing cluster systems seamlessly. For our project, we will leverage Apache Beam to help us build Machine Learning pipelines to process data. We have the following steps to execute:

(1) Read data from GCS and do Map operations: Trim spaces
(2) Map operations: Format data into dictionaries, and shift each value by 1
(3) Transform data to TFExample and serialize
(4) Split dataset to Training and Testing by 80/20 percent
(5) Write data to TFRecord

With this in mind, we will first create a beam pipeline, and then we design our data pipeline based on the requirements. For instance, initially we will do:

```
with beam.Pipeline(options = beam_options) as p,
     beam_impl.Context(temp_dir = tft_temp_dir):

dataset = (
  p
  | 'Read from GCS' >>
       beam.io.ReadFromText(os.path.join(data_dir,
       FILENAME), skip_header_lines = SKIP_HEADER)
  | 'Trim spaces' >> beam.Map(lambda x: x.split(",") if
       SPLIT_CHAR else x.split())
  | 'Format to dict' >> beam.Map(lambda x: {"user": x[0],
       "item": x[1], "rating": x[2]})
  | 'Shift by 1' >> beam.Map(shift_by_one)
)
```

Similarly, we follow the programming models of Beam to transform data, split data, and write to TFRecord. Finally, we store our data back into GCS. Then, the next question we would ask should be: how should we monitor and schedule our jobs? To put things simply, we will need Google Dataflow to unblock this feature.
Google Dataflow is a service that could run jobs built by Apache Beam libraries serverlessly. [5] Generally speaking, when we run a job on Dataflow, it will distribute tasks into many VMs and dynamically managing resources. Moreover, what makes Google Dataflow stands out is its Directed acyclic graph (DAG) representation for each subtask that helps users monitor and debug [11]. In our context, we will have a dataflow that looks like Fig. 3.

Finally, we will want to deploy our Tensorflow training tasks to Google AI Platform. We have `trainer/task.py` to achieve this. Briefly speaking, the AI Platform makes it easy for data scientists and researchers to streamline their ML workflows easily [7]. We can build machine learning models using managed distributed training infrastructure. As it is integrated with Cloud Dataflow for preprocessing and thus allowing us to access data from GCS, we could correspondingly set our `-staging-bucket` and also our working directory, then the AI platform could access processed data either in batch or stream format. Actually, we don't even need to modify our model script, and everything could remain the same as a

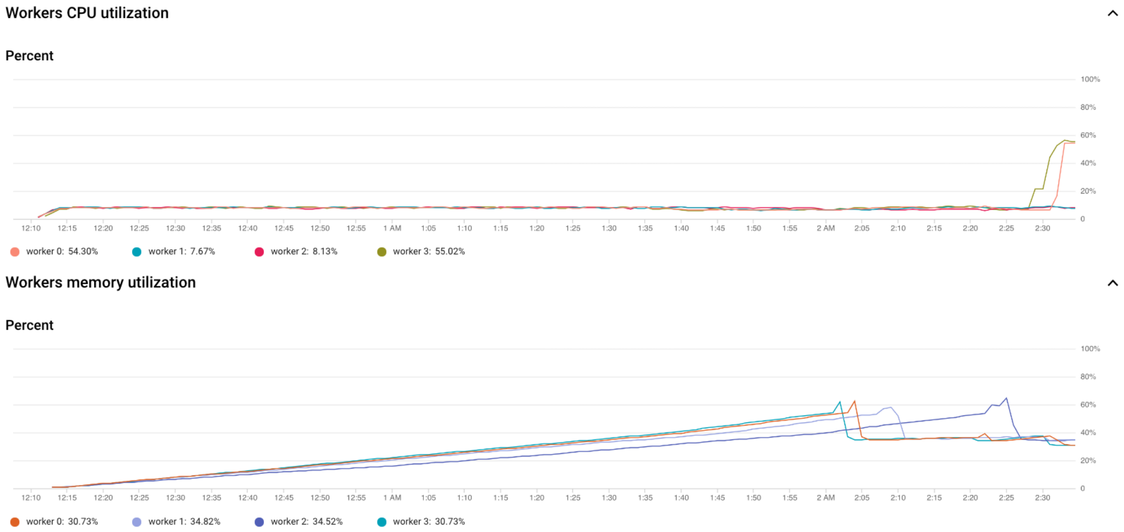


Figure 4: AI Platform real-time computational power visualization, performed on MovieLens 25M

locally running version. The only thing we have to configure is the command for submitting our job. For instance, we have:

```
run gcloud ai-platform jobs submit training $JOB \
  --module-name trainer.task \
  --package-path trainer \
  --staging-bucket $BUCKET \
  --runtime-version 1.15 \
  --region $REGION \
  --python-version 3.7 \
  --stream-logs \
  --config config.yaml \
  -- \
  --work-dir $WORK_DIR
```

Moreover, Google AI Platform provides simple yet powerful data visualization, helping customers know their CPU and memory utilization well, as shown on 4. To sum up, Google cloud services provide powerful tools for ML researchers, and obviously, these tools are not limited to enterprise only. It could also help students in a class project scale manner and proved to be easy-to-use.

4 Evaluation

For the evaluation section, we will completely test our model performance and model accuracy on the cloud. Thanks to Google AI Platform, we could see logs in real-time, and moreover, we fetch our up-to-date MSE loss value. By collecting the data after running our model for 1000 epochs, we could plot our MSE loss like Fig. 5. In epoch 1, the loss value is about 16 for both the training and testing set, and after 400 epochs the loss drops to around 5. In epoch 800 we can apparently see that the training and testing loss has dropped to around 1, denoting that our Collaborative Filtering model is successfully trained. We can see that the MSE loss for both training and testing set has decreased successfully. In this experiment,

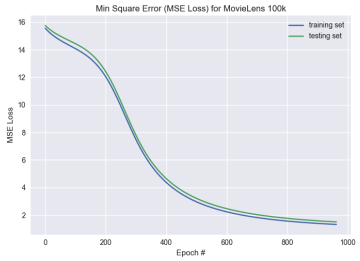


Figure 5: Training and Testing MSE Loss performed on MovieLens 100K trained on GCP, 1 master, 0 worker

we use the dataset MovieLens 100K. We have 943 users and 1682 items, and the training versus testing data split is 80 and 20. We Built a sparse matrix with size 943 x 1682, and there 100,000 entries scattered in the matrix. For the 25M dataset, our loss function for training and testing and drops to 14.5658 and 14.5705 in epoch 1000. This loss value should be able to further get optimized if we add a regularization term or fine-tune the other hyperparameters such as learning rate in SGD.

To fine-tune and maximize our efficiency and furthermore, find the most cost-effective solution for a Recommender System problem, we cannot only look into one dataset. Therefore, we introduced another dataset named MovieLens 25M. It has approximately 162541 users and 209171 items, and the number of existing entries is roughly 25 Million. This is the largest dataset that the MovieLens team offers currently, and we assumed that by leveraging this huge dataset, we can really quantitatively evaluate the scalability and model performance. We would argue that as the amount of data grows larger, the need for leveraging cloud services increases. One thing to note is that in our experiment settings, we don't do stream processing. We will only load data in a batch processing manner. We will distribute our tasks into different nodes in the cluster, and see how the

Task ID	Dataset	Machine Type	# of Master Nodes	# of Worker Nodes	Execution Time	Consumed ML Units
20201026_171017	ML100k	n1-standard-4	1	0	6 min 26 sec	0.06
20201026_215825	ML25m	n1-highmem-16	1	0	3 hr 4 min	5.76
20201029_235922	ML100k	n1-standard-4	1	2	5 min 31 sec	0.1
20201029_235935	ML100k	n1-standard-4	1	4	5 min 18 sec	0.32
20201030_000018	ML25m	n1-highmem-16	1	2	2 hr 57 min	15.53
20201030_000729	ML25m	n1-highmem-16	1	4	2 hr 26 min	22.68

Table 1: Quantitative analysis categorized into three variations: Dataset, Machine Type, # of Master and Worker Nodes

number of nodes effects our performance evaluated in execution time. In Fig. 4, we can see our computational resources that were used over time. For that specific example, we have 1 master with 4 workers doing the real computation part. It's apparent that all workers have similar patterns in CPU and memory utilization.

Table 1 demonstrates our experimental results. We have made 6 trials, and each with different variations. We divided them into 3 categories: Dataset, Machine Type, and the number of master and worker nodes. For the dataset part, we will try on two datasets in different orders: ML100k and ML25m. And for the machine type, we have several choices as described in Google's official guidance, but we will make them as control variables: we only use `n1-standard-4` for ML100k and `n1-highmem-16` for ML25m. According to the official document, n1-standard-4 has 4 virtual CPUs and 15 GB in memory; on the other hand, `n1-highmem-16` has 16 virtual CPUs and 104 GB in memory. Finally, we have the number of master nodes and the worker nodes for us to toggle. We tried three cases: only 1 master node, 1 master node plus 2 worker nodes, and 1 master node plus 4 worker nodes. One thing to note is that there is a column named *Consumed ML Units*. Google's Consumed ML Units are equivalent to training units with the duration of the job factored in. It helps users to know the cost of the training. We use the following formula: `Consumed ML units * $0.49` to get a general idea of how much each job costs.
By looking into Table 1, we can observe several interesting facts. Firstly, we noticed that the number of worker nodes could significantly boost our model training time, and even though it might cost more but seemingly upgrading from `1 master, 0 workers` to `1 master, 4 workers` is promising, especially in the context of ML25m or any larger dataset scenario. Secondly, we can also claim that as the model scale goes up, the necessity for leveraging cloud computing arises. If we were training on a lab machine or even our own laptop, or if we chose to create a virtual instance on Amazon EC2 or Azure VM, the memory and the cost is incomparable. Obviously, we could not find a machine easily to deal with a sparse matrix with size `162541 x 209171` or even larger, but Google AI Platform could handle these hardware capabilities seamlessly.

5 Conclusion

In this paper, we dived into the production Machine Learning workflow for recommender systems based on Google Cloud Platform. We created an end-to-end Collaborative Filtering model using Matrix Factorization and apply Stochastic Gradient Descent to optimize our objective function. The model part is relatively straightforward, and we put more emphasis on how we leveraged the tools. We leveraged Google Cloud Storage (GCS) for data storaging, Apache Beam for defining data pipelines, Google Dataflow for running jobs written in Beam matter, and for the last step, we looked into Google AI Platform to deploy and train our Tensorflow model. We validated our model by Mean Squared Error loss, and also we made several experiments for datasets of different sizes, and also a different number of master and worker nodes with an aim to searching for a cost-efficient plan under our use case. Finally, we released our code on Github and it is now open-source, with an aim of helping other ML researchers or students at school could get a glimpse of the benefits of Cloud Computing. To sum up, we *deeply* believe that the next step to Artificial Intelligence will be fully hosted on the cloud.

References

[1] L ´eon Bottou. 2010. Large-Scale Machine Learning with Stochastic Gradient Descent. *COMTSTAT* (2010). https://leon.bottou.org/publications/pdf/compstat-2010.pdf

[2] Badrul Sarwar et al. 2001. Item-Based Collaborative Filtering Recommendation Algorithms. *WWW* (2001). http://files.grouplens.org/papers/www10_sarwar.pdf

[3] H. Robbins et al. 1951. A Stochastic Approximation Method. (1951). https://projecteuclid.org/download/pdf_1/euclid.aoms/1177729586

[4] Martın Abadi et al. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. (2015). http://download.tensorflow.org/paper/whitepaper2015.pdf

[5] Tyler Akidau et al. 2015. The Dataflow Model: A Practical Approach to Balancing Correctness, Latency, and Cost in Massive-Scale, Unbounded, Out-of-Order Data Processing. *VLDB* (2015). https://www.vldb.org/pvldb/vol8/p1792-Akidau.pdf

[6] GCP. 2020. *Apache Beam: An advanced unified programming model.* https://beam.apache.org/

[7] GCP. 2020. *Google AI Platform: One platform to build, deploy, and manage machine learning models.* https://cloud.google.com/ai-platform

[8] GCP. 2020. *Google Cloud | Large-Scale Recommendation Systems - What and Why?* https://developers.google.com/machine-learning/recommendation/collaborative/basics

[9] GCP. 2020. *Google Cloud | Machine Learning with Apache Beam and TensorFlow - Molecules Walkthrough.* https://cloud.google.com/dataflow/docs/samples/molecules-walkthrough

[10] GCP. 2020. *Google Cloud Storage: Object storage for companies of all sizes. Store any amount of data. Retrieve it as often as you'd like.* https://cloud.google.com/storage

[11] GCP. 2020. *Google Dataflow: Unified stream and batch data processing that's serverless, fast, and cost-effective.* https://cloud.google.com/dataflow

[12] Konstan J. Borchers A. Riedl J.. Herlocker, J. 1999. An Algorithmic Framework for Performing Collaborative Filtering. *Proceedings of the Conference on Research and Development in Information Retrieval* (1999). http://files.grouplens.org/papers/algs.pdf

[13] Adrian Hsu. 2020. *git://AdrianHsu/reco-sys-on-gcp: Scalable Machine Learning (Recommender System) with Apache Beam, Google Dataflow and TensorFlow.* https://github.com/AdrianHsu/reco-sys-on-gcp